



---

# Innovus User Guide

**Product Version 15.1**

**May 2015**

© 2014-2015 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

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

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

All other trademarks are the property of their respective holders.

**Patents:** Licensed under U.S. Patent Nos. 7,526,739; 8,032,857; 8,209,649; 8,266,560; 8,650,516

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

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

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

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

---

# Contents

---

About This Manual	32
Audience	32
How This Manual Is Organized	32
Conventions Used in This Manual	33
Related Documents	34
Additional Learning Resources	35
<b>1</b>	<b>37</b>
Product and Licensing Information	37
Overview	37
Innovus System Products and Product Options	38
Innovus Implementation System	38
Virtuoso Digital Implementation and First Encounter Product Packaging	40
Product Options	42
Licensing Terminology	45
Optional license requirement for 10/20/32nm Nodes	49
<b>2</b>	<b>50</b>
Flows	50
Design Implementation Flow	51
Introduction	52
Recommended Timing Closure Flow	53
Software	53
Foundation Flow	54
Data Preparation and Validation	54
Flow Preparation	61
Pre-Placement Optimization	64
Floorplanning and Initial Placement	64

PreCTS Optimization	69
Clock Tree Synthesis	75
PostCTS Optimization	77
Detailed Routing	81
PostRoute Optimization	84
Chip Finishing	88
Timing Sign Off	90
Final Timing Analysis and Optimization using Tempus/Quantus	91
Additional Resources	91
Hierarchical and Prototyping Flow	92
Introduction	92
Top-down and Bottom-up Hierarchical Methodologies	94
Hierarchical Floorplan Considerations	96
Hierarchical Partitioning Flow and Capabilities	98
Chip Planning	101
Supporting Giga-Scale Designs in Planning stage	114
Top-level Timing Closure	114
Chip Assembly	116
3	120
Infrastructure Related Capabilities	120
Getting Started	121
Product and Installation Information	121
Setting the Run-Time Environment	121
Temporary File Locations	122
OpenAccess	123
Launching the Console	123
Tab Completing Command Names, Parameter Names, Global Variable Names and Enum Values	123
Command-Line Editing	125
Setting Preferences	129

Starting the Software	131
Interrupting the Software	131
Using the Log File Viewer	134
Accessing Documentation and Help	135
Customizing the User Interface	142
Overview	142
Creating a New Menu	143
Modifying an Existing Menu	144
Adding a New Toolbar and Toolbutton	147
Querying and Configuring Interface Elements	149
Accelerating the Design Process By Using Multiple-CPU Processing	151
Overview	151
Running Distributed Processing	154
Running Multi-Threading	154
Running Superthreading	155
Memory and Run Time Control	155
Checking the Distributed Computing Environment	157
Setting and Changing the License Check-Out Order	157
Limiting the Multi-CPU License Search to Specific Products	158
Releasing Licenses Before the Session Ends	158
Controlling the Level of Usage Information in the Log File	158
Where to Find More Information on Multi-CPU Licensing	159
Data Preparation	159
Generating a Technology File	160
Preparing Physical Libraries	160
Unsupported LEF and DEF Syntax	161
Generating the I/O Assignment File	165
Preparing Timing Libraries	189
Encrypting Libraries	189
Preparing Timing Constraints	189
Preparing Capacitance Tables	190

Preparing Data for Delay Calculation	190
Preparing Data for Crosstalk Analysis	190
Checking Designs	190
Preparing Data in the Timing Closure Design Flow	191
Converting iPRT Format to LEF	191
Importing and Exporting Designs	191
Overview	193
Verifying Data before Importing a Design	194
Preparing the Design Netlist	194
The init_design Import Flow	194
Importing Designs using the GUI	198
Loading a Previously Saved Global Variables File	200
Handling Verilog Assigns	201
Configuring the Setup for Multi-Mode Multi-Corner Analysis	201
Saving Designs	214
Loading and Saving Design Data	215
Converting an Innovus Database to GDSII Stream or OASIS Format	219
About the GDSII Stream or OASIS Map File	225
Updating Files During an Innovus Session	235
SKILL to TCL Mapping	236
4	239
Design Planning Capabilities	239
Floorplanning the Design	240
Overview	241
Common Floorplanning Sequence	242
Viewing the Floorplan	243
Module Constraint Types	245
Grouping Instances	250
Creating and Editing Rows	256
Using Vertical Rows	257

Using Multiple-height Rows	260
Performing I/O Row Based Pad Placement	271
Editing Pins	278
Running Relative Floorplanning	287
Saving and Loading Floorplan Data	290
Snapping the Floorplan	291
Resizing the Floorplan	293
Checking the Floorplan	304
FinFET Technology	306
Related Topics	310
<b>Using Structured Data Paths</b>	<b>311</b>
Overview	311
Benefits of Using SDP	312
General SDP Flow	314
Support for High-Speed Flip Flop Columns	315
SDP Placement Flow	317
Implementing SDP Capability	323
SDP Relative Placement File	324
Aligning SDPs by Pins	336
Setting SDP Options	338
Optimizing a Design with SDPs	340
Checking SDP Placement	342
<b>Bus Planning</b>	<b>343</b>
Overview	343
Bus Planning Flow in Innovus	344
Creating a Bus Guide	345
Moving and Stretching a Bus Guide	354
Cutting, Splitting, and Merging Bus Guides	354
Customizing the Bus Guide Display	356
Saving and Restoring Bus Guide Information	358
Verifying Bus Guide	358

Limitations of Bus Planning	358
Power Planning and Routing	360
Overview	360
Before You Begin	361
Results	361
Loading, Saving, and Updating Special Route	362
Global Net Connections	362
Creating a Ring with User Defined Coordinates	364
Fixing LEF Minimum Spacing Violations	365
Adding Stripes to Power Domains	365
Adding Stripe in Multi-CPU mode	367
5	368
Design Implementation Capabilities	368
Low Power Design	369
Overview	371
Power Domain Shutdown and Scaling	371
Support for the Common Power Format (CPF)	373
Support for IEEE1801	376
Flow Special Handling for Low Power	382
Multiple Supply Voltage Top-Down Hierarchical Flow	400
Example of Block-Level CPF Generated by Innovus	406
Example of Top-Level CPF Generated by Innovus	410
Multiple Supply Voltage Bottom-Up Hierarchical Flow	414
Leakage Power Optimization Techniques	417
Power Shutdown Techniques	421
Power Switch Optimization	447
Power Switch Prototyping	449
Placing the Design	457
Overview	458
Loading a Design	458

Preparing for Placement	458
Guiding Placement With Blockages	459
Adding Well-Tap Cells	461
Adding End-Cap Cells	462
Placing Spare Cells and Spare Modules	465
Adding Padding	471
Placing Standard Cells	474
Running Placement in Multi-CPU Mode	475
Checking Placement	478
Adding Filler Cells	480
Placing Gate Array Style Filler Cells for Post-Mask ECO	481
Adding Decoupling Capacitance	482
Adding Logical Tie-Off Cells	483
Saving Placement Data	483
Specifying and Placing JTAG and Other Cells Close to the I/Os	483
Optimizing and Reordering Scan Chains	484
Clock Tree Synthesis	497
The Clock Tree Synthesis Engines	498
Overview	500
Flow and Quick Start	502
Configuration and Method	506
Concepts and Clock Tree Specification	517
Reporting	540
CCOpt Clock Tree Debugger	547
Additional Topics	557
CCOpt Property System	563
Migrating from FE-CTS	567
Legacy FE-CTS Flow	569
Legacy FE-CTS Capabilities	574
Before You Begin	577
Results	577

Understanding the CTS Operation Modes	577
How CTS Calculates Skew Values	580
Improving PostRoute Correlation	581
Specifying Macro Model Delays	582
Grouping Clocks	586
Analyzing Hierarchical Clock Trees	586
Module Placement Utilization	588
Clock Designs with Tight Area	588
Balancing Pins for Macro Models	588
Timing Model Requirement for Cells	588
Delay Variation and OCV	588
Understanding PostCTS Clock Tree Optimization	590
Creating a Clock Tree Specification File	593
CTS Report Descriptions	644
Supported SDC Constraints	649
Working with Clock Mesh Structures	651
Overview	651
Clock Meshes Versus Clock Trees	651
Creating Clock Meshes	654
MultiSpine Clock Mesh	664
Optimizing Timing	665
Overview	667
Before You Begin	668
Results	668
Interrupting Timing Optimization	670
Performing Optimization Before Clock Tree Synthesis	671
Performing PostCTS Optimization	674
Performing PostRoute Optimization	678
Optimizing Power During optDesign	683
Using Useful Skew	687
Distributed Timing Analysis for Hold Fixing	689

Using Active Logic View for Chip-Level Interface Circuit Timing Closure	690
Optimizing Timing in On-Chip Variation Analysis Mode	690
Using Conformal Constraint Designer During Timing Optimization	694
Optimizing Timing Using a Rule File	698
Optimizing Timing When the Constraint File Includes the set_case_analysis Constraint	698
Using the Footprintless Flow	698
Using Cell Footprints	700
Viewing Added Buffers, Instances, and Nets	701
Using Signoff Timing Analysis to Optimize Timing	702
Running MMMC SignOff ECO within Innovus	703
<b>Using the NanoRoute Router</b>	<b>710</b>
About NanoRoute Routing Technology	712
Routing Phases	712
NanoRoute Router in the Innovus Flow	714
Before You Begin	714
Interrupting Routing	717
Using the routeDesign Supercommand	717
Results	719
Use Models	720
Using NanoRoute Parameters	721
Accelerating Routing with Multi-Threading and Superthreading	723
Following a Basic Routing Strategy	727
Checking Congestion	731
Resolving Open Nets	735
Running Timing-Driven Routing	738
Routing Clocks	741
Preventing and Repairing Crosstalk Problems	743
Running ECO Routing	746
Evaluating Violations	747
Concurrent Routing and Multi-Cut Via Insertion	755
Postroute Via Optimization	756

Optimizing Vias in Selected Nets	757
Via Optimization Options	757
Performing Shielded Routing	758
Routing Wide Wires	762
Repairing Process Antenna Violations	764
Creating RC Model Data in tQuantus Model File	767
Using a Design Flow that Includes Astro or Apollo	770
Troubleshooting	771
Optimizing Metal Density	772
Overview	772
Before You Begin	773
After You Complete Adding Via and Metal Fills	774
Metal Fill Features	774
Specifying Metal Fill Parameters	781
Recommendations for Adding Timing-Aware Metal Fill	782
Adding Metal Fill Over Macros	785
Recommendations for Power Strapping Mode	787
Adding Via Fill	788
Recommendations for Metal/Via Fill Flow	788
Recommendations for In-design Sign-off Metal Fill Flow	792
Achieving Gradient Density with Preferred Density Setting	794
Specifying Metal Fill Spacing Table	796
Trimming Metal Fill	799
Trimming Metal Fill for Timing Closure	801
Verifying Metal Density	803
Adding Metal Fill Using the GUI	804
Adding Metal Fill with Iteration	804
Flip Chip Methodologies	806
Overview	808
Flip Chip Flow in Innovus	810
Introduction to Flip Chip Methodology	811

Data Preparation	819
Flip Chip Floorplanning	834
Viewing Flip Chip Flightlines	857
Power Planning in Flip Chip Design	867
RDL Routing	868
Advanced Flip Chip Features	912
RDL Extraction	920
SI and Timing Analysis	921
<b>6</b>	<b>924</b>
<b>Hierarchical Flow Capabilities</b>	<b>924</b>
Partitioning the Design	925
Overview	925
Flow Methodologies	925
Specifying Partitions and Blackboxes	932
Working with Nested Partitions	943
Assigning Pins	947
Inserting Feedthroughs	980
Generating the Wire Crossing Report	1013
Estimating the Routing Channel Width	1017
Running the Partition Program	1018
Restoring the Top-Level Floorplan with Partition Data	1033
Concatenating Netlist Files of a Partitioned Design	1033
Saving Partitions	1034
Loading Partitions	1034
Working with OpenAccess Database	1036
Parallel Job Processing	1037
Focused Methodologies	1037
Timing Budgeting	1054
Overview	1056
Is My Design Ready for Budgeting?	1057

Deriving Timing Budgets	1058
Calculating Timing Budgets	1061
Master Clone Budgeting	1063
Constraints Adjustment	1066
Analyzing Timing Budgets	1068
Budgeting Libraries	1072
Customizing Budget Generation	1076
Fixing Budget	1077
Modifying Budgets	1079
Reading the Justify Budget Report	1080
Reading the Justify Exception Report	1088
Support for Distributed Processing in Budgeting	1093
Constraints Support in Budgeting	1094
Warning Report	1097
Using ART in Hierarchical Designs	1101
Overview	1101
Types of Active Logic Views	1101
Creating an Active Logic View	1102
The flexILM PreCTS Closure Flow	1103
Top-level Timing Closure Methodologies	1105
Using Interface Logic Models (ILM)	1105
Extracting Timing Models	1130
Overview	1132
ETM Generation	1133
ETM Generation for MMMC Designs	1134
ETM Generation Flows	1134
Basic Elements of Timing Model Extraction	1135
Clocks with Sequential and Combination Arcs	1146
Secondary Load Dependent Networks	1146
Characterization Point Selection	1147
Constraint Generation during Model Extraction	1149

Slew Propagation Mode in Model Extraction	1152
Parallel Arcs in ETM	1152
Latency Arcs Modeling	1153
Latch-Based Model Extraction	1153
Stage Weight Extraction in ETM	1154
PG Pin Modeling During Extraction	1156
Extracting Timing Models with Noise (SI) Effect	1158
Merging Timing Models	1158
Limitations of ETM	1159
Validation of Generated ETM	1164
ETM Extremity Validation	1172
<b>7</b>	<b>1175</b>
<b>Prototyping Flow Capabilities</b>	<b>1175</b>
Creating An Initial Floorplan Using Automatic Floorplan Synthesis	1177
Overview	1177
Data Preparation	1179
Importing the Design	1187
Setting Automatic Floorplan Synthesis Global Parameters	1187
Creating an Initial Floorplan	1188
Creating Floorplan for Hierarchical Design	1189
Creating Multiple Alternative Floorplans	1193
Analyzing the Floorplan	1194
Adjusting Macro Placement	1195
Saving the Floorplan	1201
Using Trial Route for Congestion and Timing Analysis	1202
Overview	1202
Data Preparation	1203
Routing a Flat Design	1203
Routing a Partitioned Design	1204
Routing Two-Metal Layer Designs	1206

Routing Using the NanoRoute Global Router	1206
Loading and Saving Route Data	1206
Analyzing Route Data	1207
Improving Route Congestion	1216
Using Bus Guides	1217
Trial Route Behavior For Hierarchical Flow	1217
Additional Information	1225
What-If Timing Analysis	1227
Performing What-If Timing Analysis	1227
Fast Slack Timing Analysis	1234
Performing Fast Slack Timing Analysis	1234
Prototyping Methodologies	1238
Overview	1238
Prototyping Methodologies	1239
<b>8</b>	<b>1268</b>
<b>Analysis Capabilities</b>	<b>1268</b>
RC Extraction	1269
Overview	1270
Before You Begin	1271
Extraction Flow in Innovus	1272
PreRoute Extraction	1274
PostRoute Extraction	1274
Scale Factor Setting	1279
Generating a Capacitance Table	1279
Reading a Capacitance Table	1287
Reading a Quantus Techfile	1287
PreRoute Extraction Flow without Capacitance Table Data	1287
Correlating Native Extraction With Sign-Off Extraction	1289
Distributed Processing	1295
Using Advanced Virtual Metal Fill	1298

Calculating Delay	1300
Overview	1300
Data Preparation	1300
Delay Calculation Modes and Related Controls	1301
Running Delay Calculation	1303
Calculating Delay in Multi-Thread Mode	1303
Timing Analysis	1304
Overview	1304
Timing Analysis Features	1305
MMMC-On By Default Functionality	1306
Before You Begin	1306
Calculating Clock Latency	1307
Path Exceptions Priority	1308
Specifying Timing Analysis Modes	1309
Clock Path Pessimism Removal	1323
Analyzing Timing Problems	1330
Debugging Timing Results	1333
Overview	1333
Timing Debug Flow	1334
Generating Timing Debug Report	1334
Displaying Violation Report	1335
Analyzing Timing Results	1335
Creating Path Categories	1342
Using Categories to Analyze Timing Results	1349
Editing Table Columns	1352
Viewing Schematics	1355
Running Timing Debug with Interface Logic Models	1356
Power and Rail Analysis	1357
Overview	1357
Early Rail Analysis	1357
Signoff-Rail Analysis	1371

TCL Command	1371
Innovus and Voltus Menu Differences	1372
Power Analysis and Reports	1374
Static Power Analysis Overview	1374
Vector-based Average Power Calculation	1383
Propagation-based average power calculation	1383
Static Power Analysis Flow	1387
Static Power Reports	1394
Static Power Analysis Plots	1396
Viewing and Debugging Static Plots	1398
Interactive Queries of Power Data	1400
Static Power Histograms and Pie-charts	1400
Analyzing and Repairing Crosstalk	1401
Overview	1401
Inputs for SI Analysis	1401
Setting Up Innovus for SI Analysis	1402
Preventing Crosstalk Violations	1411
Fixing Crosstalk Violations	1411
Performing XILM-Based SI Analysis and Fixing	1415
9	1416
Verification Capabilities	1416
Identifying and Viewing Violations	1417
Overview	1417
Interrupting Verification	1420
Verifying Connectivity	1421
Verifying Metal Density	1425
Verifying Geometry	1427
Verifying Process Antennas	1433
Verifying Well-Process-Antenna Violations	1436
Verifying End Cap Violations	1438

Verifying Maximum Floating Area Violations	1442
Verifying AC Limit	1443
Verifying Isolated Cuts	1460
Verifying Tie Cells	1461
Viewing Violations With the Violation Browser	1463
Saving Violations	1465
Clearing Violations	1466
Verifying Well Pins and Bias Pins	1467
Overview	1467
Flow	1467
Important Considerations for Usage	1472
Integration with LPA and CCP	1474
Overview	1474
Results	1475
Before You Begin Running LPA	1475
Running LPA from Innovus	1475
Before You Begin Running CCP	1496
Running CCP from Innovus	1497
10	1506
ECOs and Interactive Design Editing	1506
ECO Flows	1507
Overview	1507
Pre-Mask ECO Changes from a New Verilog File	1508
Pre-Mask ECO Changes from a New DEF File	1512
Pre-Mask ECO Changes from an ECO File	1516
Post-Mask ECO Changes from a New Verilog Netlist (Using Spare Cells Flow)	1519
Post-Mask ECO Changes from a New Netlist (Using Gate Array Cells Flow)	1524
Post-Mask ECO Changes from a New Verilog Netlist (Using Gate Array Filler Cells Flow)	1527
ECO Directives	1531
ADDHIERINST	1532

ADDINST	1533
ADDMODULEPORT	1534
ADDNET	1535
ATTACHMODULEPORT	1536
ATTACHTERM	1537
CHANGECELL	1539
CHANGEINST	1540
CHANGEINSTNAME	1542
DELETEBUFFER	1542
DELETEINST	1543
DELETEMODULEPORT	1544
DELETENET	1545
DETACHMODULEPORT	1546
DETACHTERM	1546
INSERTBUFFER	1548
Example ECO File	1550
HECO Directives	1553
Interactive ECO	1558
Overview	1558
Before You Begin	1558
Results	1558
Adding Buffers	1559
Changing the Cell	1561
Deleting Buffers	1563
Displaying Buffer Trees	1564
Running ECO Placement	1566
Naming Conventions for Interactive ECO	1566
Comparing Physical Design Data	1567
Editing Wires	1572
Overview	1572
Before You Begin	1573

Using Keyboard Shortcuts	1574
Selecting Wires	1577
Deleting Wires	1577
Moving Wires	1578
Copying Wires	1579
Adding Wires	1580
Cutting Wires	1587
Trimming Antennas on Selected Stripes	1587
Changing Special Wire Width	1588
Repairing Maximum Wire Width Violations	1588
Duplicating Special Wires	1589
Stretching Wires	1589
Changing Wire Layers	1590
Splitting and Merging Special Wires	1591
Adding Vias	1592
Changing Vias	1592
Moving Vias	1593
Reshaping Routes	1593
Controlling Cell Blockage Visibility	1594
11	1596
Design Methodology for 3D IC with Through Silicon Via	1596
Overview	1596
TSV/Bump/Back Side Metal Modeling in Innovus	1597
Defining Keep Out Area in Hard Macros	1599
Check Bump Keep Out Area Violation	1601
3D IC Flow in Innovus	1602
Design Import	1603
Stacked IC Verilog Input	1603
Stack Configuration Input	1604
Power Connectivity Input	1604

Interface Synchronization and Information Exchange between Dies	1605
TSV and Bump Manipulation	1607
Feedthru Handling	1607
TSV and Bump Routing	1608
Cross Die Connectivity Verification	1609
12	1612
Syntax and Scripts	1612
CCOpt Properties	1613
add_driver_cell	1619
add_exclusion_drivers	1619
adjacent_rows_legal	1619
allow_non_standard_inputs_clock_gate	1619
allow_resize_of_dont_touch_cells	1620
associated_row	1620
auto_limit_insertion_delay_factor	1620
auto_limit_insertion_delay_factor_skew_group	1620
auto_sinks	1621
auto_sinks_filter	1621
balance_mode	1621
bounds	1621
buffer_cells	1622
call_cong_repair_during_final_implementation	1622
cannot_clone_reason	1622
capacitance_margin_absolute	1625
capacitance_margin_percentage	1625
capacitance_override	1626
case_analysis	1626
ccopt_auto_limit_insertion_delay	1626
ccopt_check_db_every_tcl_command	1626
ccopt_merge_clock_gates	1627

ccopt_merge_clock_logic	1627
ccopt_worst_chain_report_timing_too	1627
cell_density	1627
cell_halo_x	1628
cell_halo_y	1628
center_line	1628
check_route_follows_guide	1629
check_route_follows_guide_max_deviation	1629
check_route_follows_guide_min_length	1629
clock_gate_buffering_location	1630
clock_gating_cells	1630
clock_gating_depth	1630
clock_gating_depth_top_down	1631
clock_gating_only_optimize_above_flops	1631
clock_tree	1631
clock_tree_generator_sink_is_leaf	1631
clock_tree_source_group	1632
clock_trees	1632
clone_clock_gates	1632
clone_clock_logic	1633
compatibility_warning	1633
consider_em_constraints	1633
consider_power_management	1634
constraints	1634
cts_edge_sensitivity	1634
cts_isolated	1634
cts_merge_clock_gates	1635
cts_merge_clock_logic	1635
debug_stage_delays_for_nodes	1635
def_lock_clock_sinks_after_routing	1635
delay_cells	1636

detailed_cell_warnings	1636
effective_clock_period	1636
effective_clock_period_sources	1636
effective_sink_type	1637
effort	1637
enable_all_views_for_io_latency_update	1637
enable_datapoints	1637
enable_initial_clustering	1637
enable_locked_node_check_failure	1638
engine_implemented	1638
error_on_problematic_slew_violating_nets	1638
error_on_problematic_slew_violating_nets_max_printout	1638
exclusive_sinks_rank	1639
expand_multi_child_regions	1639
extract_balance_multi_source_clocks	1639
extract_check_timing_slacks	1639
extract_clock_generator_skew_group_name_prefix	1640
extract_clock_generator_skew_groups	1640
extract_clock_node_timing_endpoints	1641
extract_create_explicit_ignores_for_implicit_excludes_beyond_sta_clock_network	1641
extract_cts_case_analysis	1641
extract_faster_sdc_clocks_as_clock_trees	1641
extract_merge_identical_skew_groups	1642
extract_network_latency	1642
extract_no_exclude_pins	1642
extract_skew_group_sinks_at_clock_node_timing_endpoints	1643
extract_source_latency	1643
extract_through_multi_output_cells_with_single_clock_output	1643
extract_through_to	1644
extracted_from_clock_name	1644
extracted_from_constraint_mode_name	1644

extracted_from_delay_corners	1644
extract_pin_insertion_delays	1645
fastest_buffer	1645
fastest_buffer_max_trans	1645
fastest_inverter	1645
fastest_inverter_max_trans	1646
final_cell	1646
flexible_htree_max_synthesis_grid_points	1646
force_all_module_boundaries_dont_touch	1646
force_all_virtual_delay_updates	1646
force_clock_objects_to_propagated	1647
force_clock_topology_changed_after_adjustment	1647
force_clock_tree	1647
force_cluster_only	1647
force_update_io_latency	1647
generated_by_sinks	1648
grid_step	1648
hard	1648
htree_sinks	1648
hv_balance	1649
ignore_pins	1649
implicit_sink_type	1649
include_source_latency	1649
insertion_delay	1650
insertion_delay_wire	1650
inverter_cells	1650
is_user_skew_group	1650
is_vertical	1651
last_target_insertion_delay	1651
last_target_insertion_delay_wire	1651
last_target_max_trans	1651

last_target_skew	1651
last_target_skew_wire	1652
leaf_buffer_cells	1652
leaf_inverter_cells	1652
legalized_on_clock_spine	1652
lock_on_clock_spine	1653
locked_originally	1653
log_precision	1653
log_special_case_cell_selections	1653
logic_cells	1654
low_power_clustering	1654
manage_power_management_illegalities	1654
max_buffer_depth	1654
max_clock_cell_count	1655
max_delta_band_factor	1655
max_delta_slack_factor	1655
max_fanout	1655
maximum_insertion_delay	1655
max_skew_passes_with_insufficient_progress	1656
max_source_to_sink_net_length	1656
max_source_to_sink_net_resistance	1656
min_delta_band_factor	1656
move_clock_gates	1657
move_clock_nodes_for_slack	1657
move_logic	1657
move_middle_cell_first_when_adding_wire_delay	1657
name_prefix	1658
net_unbufferable	1658
no_symmetry_buffers	1658
opt_ignore	1658
optimize_ir_drop_skew_amount	1658

override_minimum_max_trans_target	1659
override_minimum_skew_target	1659
override_vias	1659
override_zero_placeable_area	1659
parents	1660
pin	1660
pin_target_max_trans	1660
post_conditioning	1660
post_conditioning_enable_drv_fixing	1660
post_conditioning_enable_drv_fixing_by_rebuffering	1661
post_conditioning_enable_drv_fixing_by_rebuffering_ccopt	1661
post_conditioning_enable_skew_fixing	1661
post_conditioning_enable_skew_fixing_by_rebuffering	1661
post_conditioning_enable_skew_fixing_by_rebuffering_ccopt	1661
primary_delay_corner	1662
print_clock_tree_modifications	1662
put_driver_in_centre_of_fanout_bounding_box	1662
recluster_to_reduce_power	1662
remove_bufferlike_clock_logic	1662
rename_clock_tree_nets	1663
report_only_skew_group_with_target	1663
report_only_timing_corners_associated_with_skew_groups	1663
resistance_margin_absolute	1663
resistance_margin_percentage	1664
restricted_cells_buffers	1664
restricted_cells_inverters	1664
route_balancing_buffers_with_default_rule	1664
route_type	1665
route_type_autotrim	1665
routing_override	1665
routing_top_fanout_count	1665

routing_top_min_fanout	1666
routing_top_transitive_fanout	1666
save_old_viz_on_cluster	1666
schedule	1666
sink_grid	1667
sink_grid_box	1667
sink_grid_exclusion_zones	1667
sink_grid_prefix	1667
sink_grid_sink_area	1668
sink_type	1668
sinks	1668
sinks_active	1668
size_clock_gates	1669
size_logic	1669
skew_band_size	1669
skew_group_insertion_delay	1670
skew_group_insertion_delay_wire	1670
skew_group_report_columns	1670
skew_group_report_histogram_bin_size	1671
skew_groups_active	1672
skew_groups_active_sink	1672
skew_groups_ignore	1672
skew_groups_sink	1672
skew_groups_source_pin	1673
skew_pass_sufficient_progress_band_size_factor	1673
skew_passes_per_cluster	1673
skip_move_gates_in_chains	1673
source_driver	1673
source_group_clock_trees	1674
source_input_max_trans	1674
source_latency	1674

source_max_capacitance	1674
source_output_max_trans	1675
source_pin	1675
sources	1675
stop_at_sdc_clock_roots	1675
stripes_bbox	1675
stripes_bottom_edge	1676
stripes_cells	1676
stripes_height	1676
stripes_left_edge	1676
stripes_pitch	1676
stripes_repeat	1676
stripes_width	1677
target_insertion_delay	1677
target_insertion_delay_wire	1677
target_max_capacitance	1677
target_max_trans	1678
target_max_trans_sdc	1678
target_multi_corner_allowed_insertion_delay_increase	1678
target_skew	1679
target_skew_wire	1679
timing_delta	1679
top_buffer_cells	1680
top_inverter_cells	1680
trace bidi_as_input	1680
transitive_fanout	1680
trunk_cell	1680
trunk_override	1681
update_io_latency	1681
update_slacks_before_skewing_adjustors	1681
use_estimated_routes_during_final_implementation	1681

use_inverters	1681
use_macro_model_pin_cap_only	1682
use_skew_implementation_cache_hold_slacks	1682
useful_skew_max_delta	1682
useful_skew_min_delta	1682
useful_skew_post_implement_db	1683
useful_skew_pre_implement_db	1683
user_skew_group	1683
virtual_delay	1683
visualization_clock_trees_initially_collapsed	1683
weak_driver_check_bounding_box_vs_drive_distance	1684
<b>Creating the ICT File</b>	<b>1684</b>
Overview	1684
Format	1685
Data	1685
Comments	1685
Case Sensitivity	1685
Warnings and Errors	1685
Invalid Layer Names	1686
Commands	1686
Sample ICT File	1696
<b>Clock Mesh Specification File</b>	<b>1710</b>
Overview	1710
Routing Type Definitions	1710
Cutout Definitions	1711
Clock Mesh Definitions	1712
Clock Mesh Specification File Example	1731
Supported CPF 1.0 Commands	1740
CPF 1.0 Script Example	1749
Supported CPF 1.0e Commands	1760
CPF 1.0e Script Example	1771

Supported CPF 1.1 Commands	1780
CPF 1.1 Script Example	1791
Supported SAI Commands	1799
add_clock	1800
add_macro	1801
add_module	1802
connect	1803
delete_macro	1804
delete_module	1805
insert_boundary_flops	1805
set_floorplan	1806
set_ref_flop	1807
set_ref_gate	1807
set_ref_memory	1808
set_sai_version	1809
Cadence-Specific Liberty Extensions	1809
Overview	1809
Guidelines For Adding ECSM Extensions	1810
Representing ECSM Information in a Library	1810
Defining ECSM Extensions in a Library	1811
Example	1818

# About This Manual

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

## Audience

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

## How This Manual Is Organized

The *Innovus User Guide* provides an extensive description of the major design flows, methodologies, and software capabilities.

The Flows section describes the key flows in the Innovus software and the recommended methodologies. It is organized into the following chapters:

- [Design Implementation Flow](#)
- [Hierarchical and Prototyping Flow](#)

The rest of the guide describes the capabilities supported by Innovus. Related capabilities are grouped together. Refer to the following Capabilities sections to see the detailed chapters under them:

- [Infrastructure Related Capabilities](#)
- [Design Planning Capabilities](#)
- [Design Implementation Capabilities](#)
- [Hierarchical Flow Capabilities](#)
- [Prototyping Flow Capabilities](#)
- [Analysis Capabilities](#)
- [Verification Capabilities](#)
- [ECOs and Interactive Design Editing](#)

- Design Methodology for 3D IC with Through Silicon Via

## Conventions Used in This Manual

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

<i>text</i>	Indicates text that you must type exactly as shown. For example:  <code>create_what_if_shape -method auto</code>
<i>text</i>	Indicates information for which you must substitute a name or value.  In the following example, you must substitute the name of a specific file for <i>configfile</i> :  <code>wroute -filename configfile</code>
<i>text</i>	Indicates the following: <ul style="list-style-type: none"> <li>● Text found in the graphical user interface (GUI), including form names, button labels, and field names</li> <li>● Terms that are new to the manual, are the subject of discussion, or need special emphasis</li> <li>● Titles of manuals</li> </ul>
[ ]	Indicates optional arguments.  In the following example, you can specify none, one, or both of the bracketed arguments:  <code>command [-arg1] [arg2 value]</code>
[   ]	Indicates an optional choice from a mutually exclusive list.  In the following example, you can specify any of the arguments or none of the arguments, but you cannot specify more than one:  <code>command [arg1   arg2   arg3   arg4]</code>

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

## Related Documents

- *What's New in Innovus*  
Provides information about new and changed features in this release of the Innovus family of products.
- *Innovus Known Problems and Solutions*

Describes important Cadence Change Requests (CCRs) for the Innovus family of products, including solutions for working around known problems.

- [\*Innovus Text Command Reference\*](#)

Describes the Innovus text commands, including syntax and examples.

- [\*Innovus Menu Reference\*](#)

Provides information specific to the forms and commands available from the Innovus graphical user interface.

- [\*Innovus Database Access Command Reference\*](#)

Lists all of the Innovus database access commands and provides a brief description of syntax and usage.

- [\*Innovus Foundation Flows User Guide\*](#)

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

- [\*Mixed Signal Interoperability Guide\*](#)

Describes the digital mixed-signal flow.

- [`README` file](#)

Contains installation, compatibility, and other prerequisite information, including a list of Cadence Change Requests (CCRs) that were resolved in this release. You can read this file online at [downloads.cadence.com](http://downloads.cadence.com).

## Additional Learning Resources

Cadence offers the following training courses on Innovus:

- [\*Innovus Implementation System \(Block\)\*](#)
- [\*Innovus Implementation System \(Hierarchical\)\*](#)
- [\*Low-Power Flow with Innovus Implementation System\*](#)

For further information on the training courses available in your region, visit the [Cadence Training](#) portal. You can also write to [training\\_enroll@cadence.com](mailto:training_enroll@cadence.com).

**Note :** The links in this section open in a new browser. They initially display the requested training information for North America, but if required, you can navigate to the courses available in other regions.



# Product and Licensing Information

---

- Overview
- Innovus System Products and Product Options
  - Innovus Implementation System
  - Virtuoso Digital Implementation and First Encounter Product Packaging
  - Product Options
  - Licensing Terminology
    - Dynamic Licensing
    - Multi-CPU Licensing
  - Optional license requirement for 10/20/32nm Nodes

## Overview

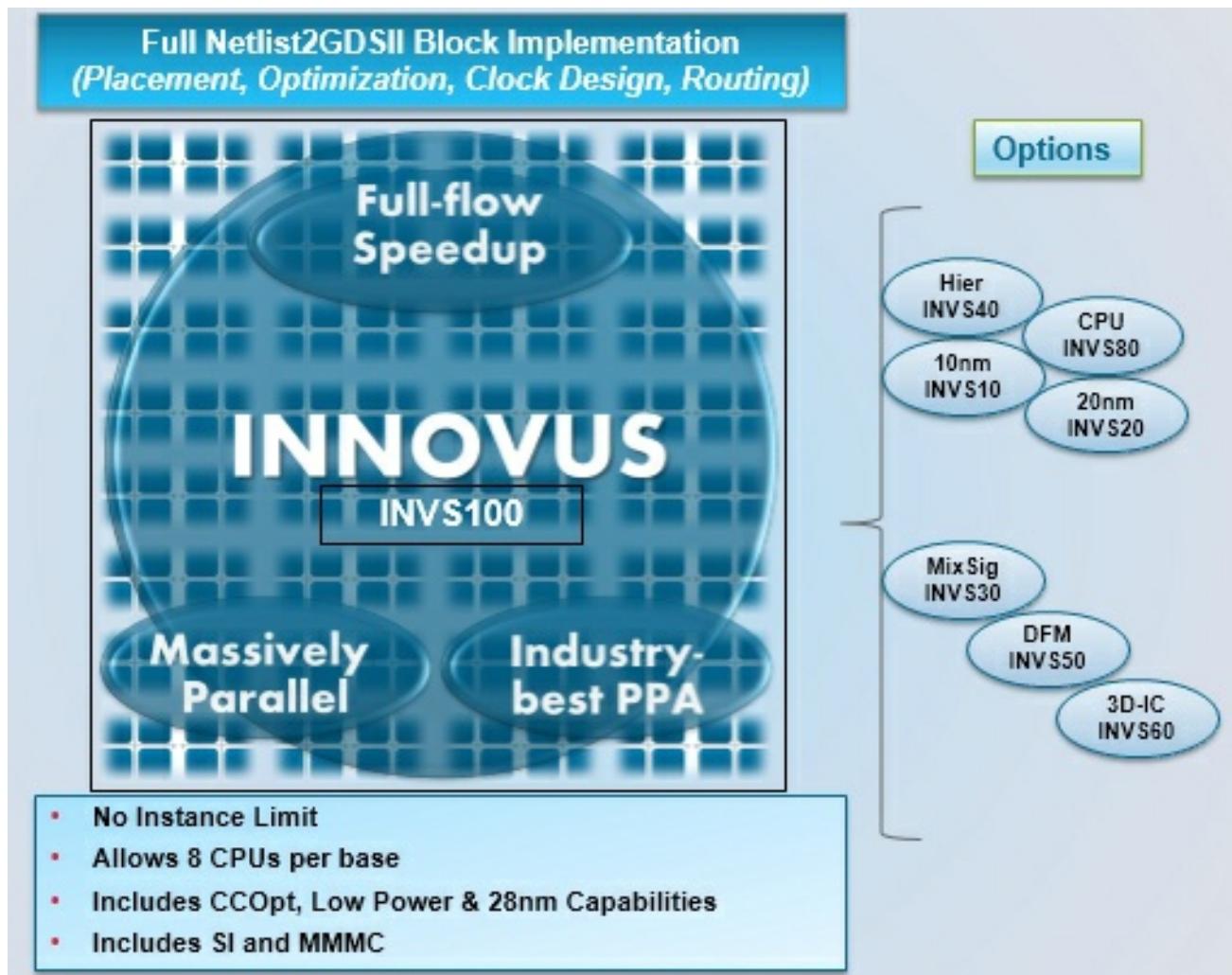
Cadence® Innovus™ Implementation System is sold as a package with a base product and additional option products. The options provide advanced features and capabilities, such as advanced node support, hierarchical design support, mixed signal design flow support etc. Each product and product option has a corresponding license. The software uses licenses to determine the features that are available as it runs.

# Innovus System Products and Product Options

This release of the Innovus System software includes the following product packages and options:

- [Innovus Implementation System](#)
- [First Encounter and Virtuoso Digital Implementation Product Packaging](#)
- [Product Options](#)

## Innovus Implementation System

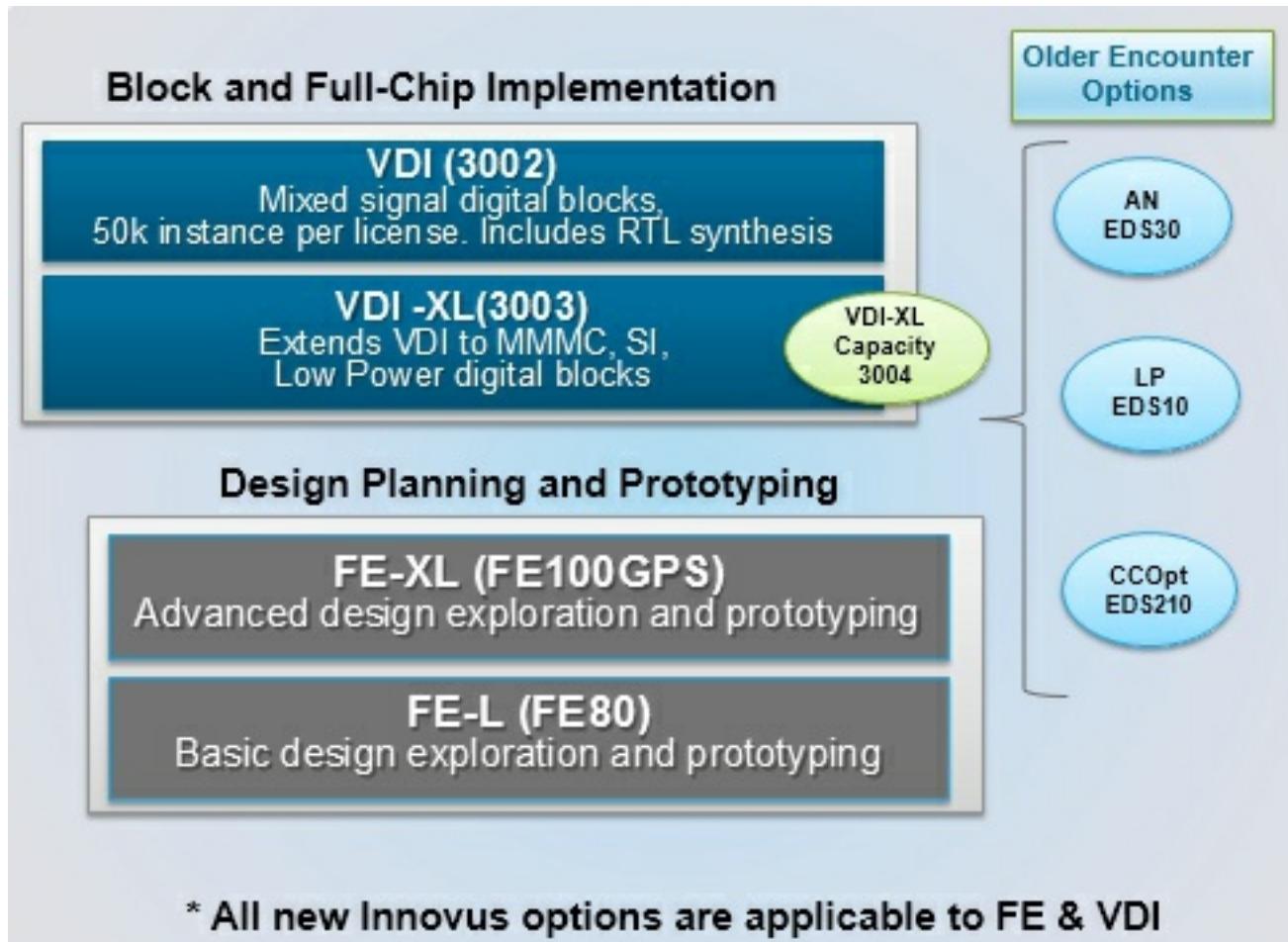


This package has only one main product, Innovus Implementation System. To start this product, type `innovus` on the UNIX command line. For more information on using this command, see "Starting the Software" section in the [Getting Started](#) chapter in the *Innovus User Guide* and the `innovus` command description in the *Innovus Text Command Reference*.

## Key Features in Innovus (INVS100)

- Full Netlist2GDSII Block Implementation (Placement, Optimization, Clock Design, Routing)
- Massively parallel multi-threaded and distributed computing architecture
- Supports all process nodes, including the latest 16nm, 14nm, and 10nm FinFET devices
- New GigaPlace solver-based placement technology that is slack-driven and topology-/pin access-/color-aware, enabling optimal pipeline placement, wirelength, utilization and PPA,
- Advanced timing and power-driven optimization that is multi-threaded and layer aware, reducing dynamic and leakage power with optimal performance
- Includes Low Power, Advanced Node and CCOpt capabilities
- Unique concurrent clock and datapath optimization that includes automated hybrid H-tree generation, enhancing cross-corner variability and driving maximum performance with reduced power
- Slack-driven routing with track-aware timing optimization that tackles signal integrity and improves post-route QOR
- Full-flow multi-objective technology enables concurrent electrical and physical optimization for best PPA
- Tight integration with signoff Tempus, Quantus and Voltus technologies to accurately model parasitics, timing, signal, and power integrity issues and converge smoothly to signoff.
- Mixed-signal design seamless Interoperability flow through integration to Virtuoso® and our custom/analog tools
- Hierarchical model creation (ILM, black box) and top-level assembly and signoff optimization
- Complete flip-chip support, 45 degree routing (Area/peripheral IO support)
- Global Timing Debug with links to Conformal Constraint Designer, Global Clock Debug, Global Power Debug

# Virtuoso Digital Implementation and First Encounter Product Packaging



## Key Features in VDI (3002)

- RTL Compiler Synthesis and N2N (Netlist to Netlist) optimization
  - 50K instances per license (Up to 2 VDI licenses can be stacked up to 100K instances)
- Block-level floorplanning, wire editing, clock tree synthesis, routing, optimization and design closure
  - 50K instances per license (multiple VDI licenses can be stacked up to 100K instances)
- Hierarchical model creation (ILM, black box)
- Automatic floorplan synthesis, automatic macro placement, wire editing
- SMART routing (Signal integrity, Manufacturing Aware, Routability, and Timing), metal fill, verify DRC/LVS, ECOs
- Multi-Vth optimization, clock gating for low power, early rail analysis using signoff power

analysis engine

- Signoff timing and delay calculation, Global Timing Debug with links to Conformal Constraint Designer (CCD)
- GDSII, Oasis, and OpenAccess (OA) support and interoperability

### **Key Features in VDI-XL (3003)**

- Capacity limited to 50K instances per license, stackable up to 100K instances with two licenses
- Capacity can be increased to 300K using the VDI-XL Capacity option
- All VDI features
- Multi-mode and multi-corner support
- SI analysis and fixing
- Low power synthesis capabilities from the RC Low Power option
- Low power implementation capability from the EDI Low Power option

### **Key Features in FE-L (FE80)**

- Silicon Virtual Prototyping (SVP), floorplanning
- Hierarchical planning (floorplanning, budgeting, partition pin assignment)
- Hierarchical model creation (ILM, black box)
- Wire editing
- Power grid planning and routing, flat and hierarchical support, basic flip-chip support
- Common Power Engine (power analysis) and Early Rail Analysis with port power views
- Fast Mode Placement and Optimization
- First Encounter extraction
- Signoff timing and delay calculation, Global Timing Debug, Global Clock Debug

### **Key Features in FE-XL (FE100GPS)**

- All FE-L features included
- Netlist-to-netlist optimization and advanced netlist restructuring
- Automatic floorplan synthesis, automatic macro placement
- Placement and optimization
- Clock Tree Synthesis
- Complete MMMC support including MMMC-ILM top-level assembly

- Power domain floorplanning, power driven place, Multi-Vth opt, DVFS, PSO, and Global Power Debug
- Tri-lib support for multi-voltage, multi-temperature delay calc
- Advanced flip-chip support with 45 degree RDL routing (Area/peripheral IO support)
- Timing Aware ECO/spare-cell remapping

## Product Options

The product options provide the extendability and cost-effective access to additional advanced technologies for specific design needs, such as low power design, mixed-signal design, design at advanced nodes and signoff analysis. These product options are available with Innovus Implementation System and First Encounter Hierarchical Prototyping Solution product packages. Innovus Implementation System includes the following product options:

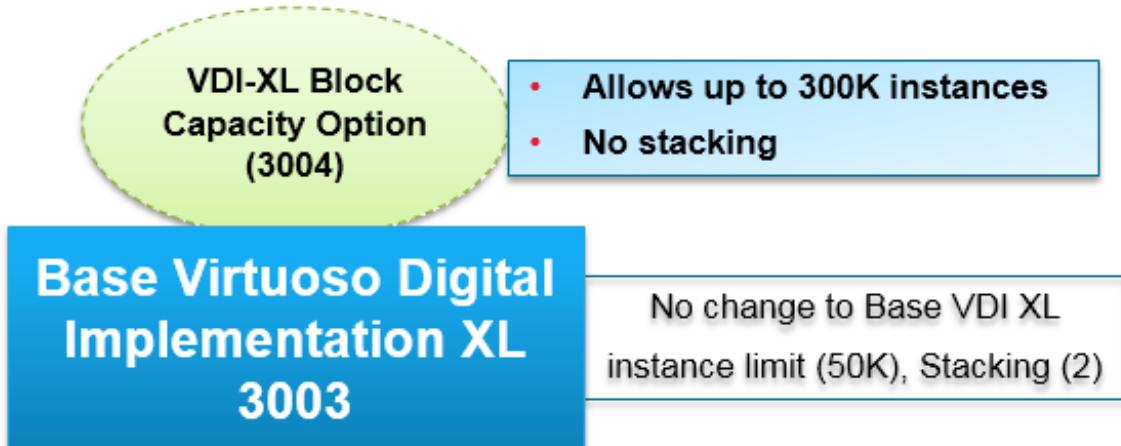
	<b>CPU Accelerator Option (INVS80)</b> <ul style="list-style-type: none"><li>● Multi-CPU acceleration with 8 additional CPUs throughout the flow</li></ul>
	<b>Hierarchical Option (INVS40)</b> <ul style="list-style-type: none"><li>● Enables all Hierarchical design capabilities</li><li>● Partitioning and Budgeting features supported</li><li>● Top level assembly and post-assembly closure</li><li>● FlexModel design exploration and prototyping</li><li>● FlexILM for hierarchical implementation</li><li>● Partition-in-partition capability</li><li>● Early floor-planning and exploration using SoC Architecture Information (SAI)</li><li>● psPM.model creation and usage</li><li>● No instance limit restrictions</li></ul>

	<b>20nm Option (INVS20)</b> <ul style="list-style-type: none"><li>• Enables 20/22/16/14nm node features across all foundries</li><li>• FinFET support</li><li>• Double patterning-correct placement and optimization</li><li>• NanoRoute double patterning-correct routing for all rules</li><li>• Rules, colorization for standard cells/hard macros</li><li>• Real-time colorization for uncolored metal shapes</li><li>• Implementation-stage DPT conflict/DRC checks</li><li>• Enables all higher nodes (28/32nm and above) features as well</li></ul>
	<b>10nm Option (INVS10)</b> <ul style="list-style-type: none"><li>• Enables 10nm node features across all foundries</li><li>• Supports self-aligned double patterning (SADP)</li><li>• Supports triple mask/color (TPT) on M1 and cut layers</li><li>• Enables all higher nodes (14/16/20nm and above) features as well</li></ul>
	<b>Mixed Signal Option (INVS30)</b> <ul style="list-style-type: none"><li>• Mixed Signal floor-planning and interoperability with Virtuoso</li><li>• Hierarchical propagation of integrated routing constraints through pull and push</li><li>• GUI to create and modify MS routing constraints for special nets</li><li>• GUI to launch Virtuoso space-based Router for routing special nets</li><li>• Enables accurate digital timing analysis using full timing model</li><li>• Integrated constraint verification using PVS</li></ul>
	<b>DFM Option (INVS50)</b> <ul style="list-style-type: none"><li>• Litho hotspot analysis and fixing with Foundry or user-defined tech file</li><li>• SAMSUNG PHR for 32nm and below</li><li>• GLOBALFOUNDRIES DRC+ for 28nm and below</li><li>• Limited model-based LPA litho checks for TSMC</li><li>• User-defined and/or foundry-certified pattern search and optimization</li><li>• Context-Aware LDE path analysis</li><li>• Limited model-based CMP analysis and hotspot detection</li></ul>

	<b>3D-IC Option (INVS60)</b> Stacked die design capabilities Applies to implementation and signoff
	<b>Advanced Node Option (EDS30)</b> <ul style="list-style-type: none"><li>• 32/28nm support in routing and verify</li><li>• Context-driven placement</li><li>• Structured datapath support</li><li>• DFM/DFY optimization for wires, cell, vias</li><li>• Litho-aware routing with prevention and fixing</li><li>• 1-D routing support</li><li>• Clock mesh &amp; hybrid implementation</li></ul>
	<b>Low Power Option (EDS10)</b> <ul style="list-style-type: none"><li>• Power-intent driven low power methodology with CPF/IEEE1801 specification</li><li>• End-to-end multi-supply voltage (MSV) support</li><li>• Power domain-aware automatic floorplan synthesis and routing</li><li>• Dynamic Voltage Frequency Scaling support</li><li>• Power shut-off and power switch prototyping</li><li>• State Retention Power Gating support</li><li>• Always-on buffer &amp; Dual-flop support</li><li>• Hierarchical Macro Model support</li></ul>
	<b>CCOpt Option (EDS210)</b> <ul style="list-style-type: none"><li>• Simultaneous clock tree synthesis and physical optimization</li><li>• True useful-skew and Multi-mode multi-corner timing including OCV derates</li><li>• Worst chain design closure with time borrowing across the delay chain</li><li>• FlexH driven Hybrid H-tree CTS</li></ul>

### VDI-XL Block Capacity Option (3004)

This release also introduces the VDI-XL Block Capacity Option (3004).



License check-out behavior is as follows:

- Default
  - Under 50k instances: vdixl checked out
  - Between 50 and 100k instances: Second vdixl checked out, as in previous releases. If not found, new vdixl\_capacity\_opt checked out
  - Between 100 and 300k instances: New capacity\_opt checked out
  - Greater than 300k instances: Errors out with appropriate message
- Explicit (at startup)
  - `-lic_startup vdixl -lic_startup_options vdixl_capacity`

## Licensing Terminology

The following terminology is useful in understanding licenses.

- **Base license** - The license that is checked out when the software starts. Only a full-fledged product license can be used as a base license. You cannot use a product option license as a base license to start the software.
- **Dynamic license** - A license for a product option that is not checked out until a feature provided by the product option is needed. You can check out more than one dynamic license per base license.
  - `innovus -lic_startup_options "Opt1 Opt2"` will check out licenses immediately at startup

- innovus -lic\_options "Opt1 Opt2" will check out licenses when needed dynamically
  - innovus -lic\_options "" will not allow the system to check out any optional licenses
  - Tcl command `setLicenseCheck -checkout "Opt"` will check out the Opt license immediately
  - Tcl command `setLicenseCheck -optionList "Opt1 Opt2"` specifies options that can be checked out when needed
- **Multi-CPU license** - A license that enables additional CPUs for multithreading, Superthreading, or distributed processing. Multi-CPU licenses must be product licenses, and can be checked out after the base license is checked out. You can check out more than one multi-CPU license per base license. For more information on these products and options, see [Innovus Packaging and Licensing](#).
  - **License Mgr & Daemon** - License Manager (lmgrd) and the Cadence license Daemon (cdslmd) should be at 11.11.1.1 or higher

## Dynamic Licensing

The Dynamic Licensing table shows the product options that each base product can check out:

- The first column lists the base product names in abbreviated format.
- The second column lists the base product number.
- The top row lists the product option names in abbreviated format.
- The second row lists the product option numbers.
- License check-out order is from left to right.
- A tick mark in a table cell means that the base product in that row can check out the product option in that column.
- A gray box means that the base product in that row is not allowed to check-out the product option in that column

### Dynamic Licensing Table

	Product Name	INVS HIER Opt	INVS 20mm Opt	INVS 10mm Opt	INVS MS Opt	INVS 3D-IC Opt	VDI-XL Capacity Opt	ENC-LP Opt	ENC-AN Opt	ENC-CCO Opt	YDS-T	TPS-L	TPS-XL	TPS-TSO	YDS-P	VTS-L	VTS-XL	VTS-AA
Base License	Product Number	INVS40	INVS20	INVS20	INVS30	INVS60	3004	EDS10	EDS30	EDS210	YDS100	TPS100	TPS200	TPS300	YDS200	VTS100	VTS200	VTS201
INVS	INVS100	✓	✓	✓	✓	✓					✓	✓				✓	✓	✓
VDI	3002		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
VDI-XL	3003		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓
FE-L	FE80	✓	✓	✓					✓			✓	✓			✓	✓	
FE-XL	FE100GPS	✓	✓	✓					✓			✓	✓			✓	✓	
YDS-T	YDS100														✓			
TPS-L	TPS100														✓	✓	✓	✓
TPS-XL	TPS200				✓									✓		✓	✓	✓
YDS-P	YDS200					✓				✓								✓
VTS-L	VTS100						✓				✓	✓						✓
VTS-XL	VTS200						✓	✓			✓	✓						✓

## Multi-CPU Licensing

The following table shows the number of CPUs that are enabled by each base license and the number of additional CPUs enabled by each multi-CPU license.

- The first column lists the base product names in abbreviated format.
- The second column lists the base product number.
- The third column shows the number of CPUs enabled by the base product in that row.
- The top row lists the names of products whose licenses can be used as multi-CPU licenses in abbreviated format.
- The second row lists the product numbers.
- License check-out order is from left to right.
- Column 4 and subsequent columns show the number of additional CPUs enabled by each multi-CPU license for that product.
- Product option licenses cannot be used as base licenses or multi-CPU licenses.
- If you request more CPUs than are available (based on the number of available licenses), the software issues a warning and runs with the number of CPUs that are available.

**Multi-CPU Licensing Table**

	Product Name	Base	INVS CPU Opt	INVS	VDI	VDI-XL	FE-L	FE-XL	VDS-T	TPS-L	TPS-XL	TPS-MP	VDS-P	VTS-L	VTS-XL	VTS-MP
Base License	Product Number	Base Count	INVS80	INVS100	3002	3003	FE80	FE100GPS	VDS100	TPS100	TPS200	TPS400	VDS200	VTS100	VTS200	VTS300
INVS	INVS100	8	8	8												
VDI	3002	1	8													
VDI-XL	3003	1	8													
FE-L	FE80	1	8				1	4								
FE-XL	FE100GPS	2	8				1	4								
VDS-T	VDS100	1														
TPS-L	TPS100	8							8	16	16					
TPS-XL	TPS200	16							8	16	16					
VDS-P	VDS200	1														
VTS-L	VTS100	1											0	0	0	
VTS-XL	VTS200	8										0	8	8		

## Optional license requirement for 10/20/32nm Nodes

The following Innovus product options provide licenses for lower technology nodes with these requirements:

- **INVS10nm**
  - Needed if the minimum width specified on any routing layer is <= 26nm
  - Is a superset of 20 & 32nm node features
- **INVS20nm**
  - Needed if the minimum width specified on any routing layer is <= 40nm
  - Is a superset of 32nm node features
- **EDS30**
  - Needed for the VDI products if the minimum width specified is <= 50nm
  - This is not needed for the Innovus product which has the capability in the base license

For more information on these products and options, see [Innovus Packaging and Licensing](#).

# Flows

---

- Design Implementation Flow
- Hierarchical and Prototyping Flow

# Design Implementation Flow

- Introduction
- Recommended Timing Closure Flow
- Software
- Foundation Flow
- Data Preparation and Validation
  - Data Preparation
  - Data Validation
    - Loading the Design
    - Checking Timing Constraint Syntax
    - Extraction File Checks
    - Validating Timing Constraints
    - Checking Logically Equivalent Cells Available for Optimization
    - checkDesign command
  - Data Preparation and Validation for Low Power Designs
- Flow Preparation
  - Setting the Design Mode
  - Extraction
  - Timing Analysis
- Pre-Placement Optimization
- Floorplanning and Initial Placement
  - Ensuring Routability
  - Validating the Floorplan
  - Timing-Driven Placement
  - Placement Analysis
- PreCTS Optimization
  - Guidelines for PreCTS Optimization
  - PreCTS optDesign Command Sequences
  - Checking and Debugging Timing Optimization Results
  - Path Group Optimization
- Clock Tree Synthesis
  - Configuring CCOpt-CTS or CCOpt

- Running CCOpt-CTS or CCOpt
- Reporting after CCOpt-CTS or CCOpt
- Visualization of Clock Trees after CCOpt-CTS or CCOpt
- PostCTS Optimization
  - PostCTS SDC Constraints
  - PostCTS Setup Optimization Command Sequences
  - Hold Optimization
  - Checking Timing
- Detailed Routing
  - Routing Command Sequence
  - Improving Timing during Routing
  - PostRoute Extraction
  - Checking Timing
- PostRoute Optimization
  - Data Preparation for SI analysis
  - PostRoute Optimization Command Sequences
  - Analysis and Debug of PostRoute Optimization Results
  - Optimizing With Third-Party SPEF or SDF
- Chip Finishing
- Timing Sign Off
- Final Timing Analysis and Optimization using Tempus/Quantus
- Additional Resources

## Introduction

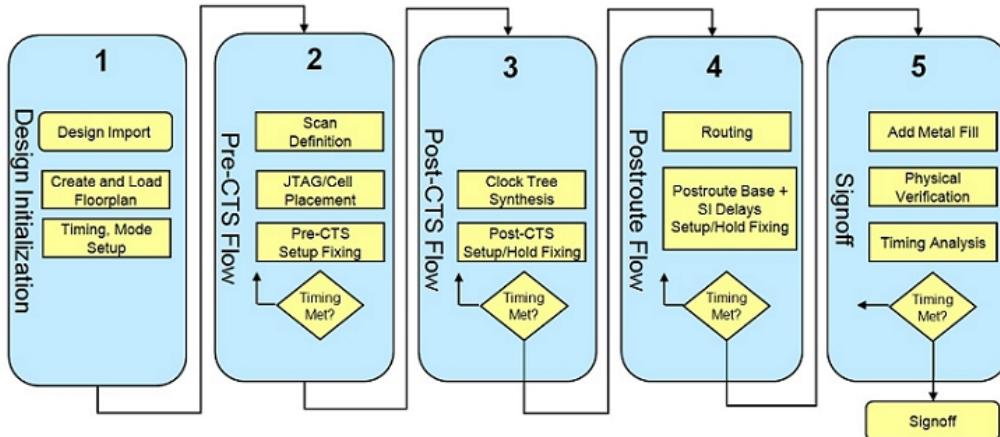
Achieving timing closure on a design is the process of creating a design implementation that is free from logical, physical, and design rule violations and meets or exceeds the timing specifications for the design. For a production chip, all physical effects, such as metal fill and coupling, must be taken into account before you can confirm that timing closure has been achieved.

Timing closure is not just about timing optimization. It is a complete flow that has to converge, including placement, timing optimization, clock tree synthesis (CTS), routing, and SI fixing. Each step has to reach the expected targets or else timing closure will likely not be achieved.

This chapter discusses each step in the implementation flow as it relates to timing closure in the Innovus™ Digital Implementation System (Innovus), and provides the recommended settings specific to high performance, congested, or high utilization designs.

## Recommended Timing Closure Flow

Below is a diagram showing the steps in the flat implementation flow:



As you proceed through the flow, it is important to investigate and validate each step before continuing. The goal is to have consistent and predictable timing results as you proceed through the flow, so it is best to debug and resolve issues early in the flow when changes have the least impact on the physical design. It is also a good idea to run the full flow in parallel to identify any roadblocks that may occur later on.

## Software

The Innovus software is constantly being improved to provide better quality of results, reliability, and ease of use. To ensure that you are running with the latest improvements, it is recommended that you run the latest software version available from <http://downloads.cadence.com>.

Specific features added in the recent versions to improve design closure include the following:

- **GigaOpt**: a multi-threaded optimization engine used for preCTS, postCTS, and postRoute optimization.
- **Clock Concurrent Optimization (CCOpt)**: combines CTS with datapath optimization to achieve better timing, power, and area results.
- **Adaptive Placement**: enables optimization-aware placement to improve the local density in optimization for better congestion and timing closure.
- **Layer-Aware Optimization**: controls both preRoute and postRoute layer-aware optimization, and is able to improve timing by assigning a minimum layer constraint on some timing-critical nets.

## Foundation Flow

If you have developed your own flow scripts, you know that maintaining and updating them can be time consuming and error prone. Also, ensuring that you are running with the latest recommended commands and options can be challenging. Therefore, we recommend using the Foundation Flow scripts. The Foundation Flow provides Cadence-recommended procedures for implementing flat, hierarchical, and low-power/CPF designs using the Innovus software. The Foundation Flow is a starting point for building an implementation environment, but you can augment them with design-specific content. Utilizing the Foundation Flow helps achieve timing closure because it provides the latest recommended flow which you can further customize based on your design requirements.

If you are new to the Foundation Flow, we recommend you start with the Foundation Flow video demonstrations. These provide examples of setting up and using the flows. They are accessible from within the Innovus GUI by selecting *Flows - Foundation Flow Demo*.

When using the Foundation Flow it is important to re-generate the scripts with each release so you run the latest flow qualified with the software. Also, review custom plug-ins on a regular basis to confirm they are still needed and do not adversely affect the flow.

## Data Preparation and Validation

This section outlines the data (libraries, constraints, netlist) required for implementing the design closure flow and how to validate that data.

The goals of data preparation and validation include:

- Confirming that Innovus has a complete and consistent set of design data (all library views and versions must be consistent).
- Ensuring that all tools in the flow interpret the timing constraints consistently.
- Making sure logically equivalent cells are defined properly.
- Correlating parasitics among the prototyping and sign-off extraction tools.

## Data Preparation

This section lists the data required and data setup recommended for the design closure flow.

### Timing Libraries

- Every cell used in the design should be defined in the timing library.
  - If multiple delay corners are being analyzed, then each cell needs to be characterized for each corner.

- Innovus supports Non-linear Delay Models (NLDM), ECSM, and CCS. ECSM libraries are recommended.
  - CCS/ECSM are less pessimistic than NLDM and, therefore, you can gain about 5% to 10% of the clock period on the slack by using these libraries.

## Physical Libraries

- You need to have an abstract defined for every cell in either a LEF file or in the OpenAccess database.
- Define Non-Default Rules (NDRs) for routing, as needed. These can be defined in the LEF file or added within Innovus using the `add_ndr` command.
- The technology should have an optimized set of vias to be used for routing. Confirm that you have the latest technology LEF file from your library vendor or foundry. Alternatively, you can generate it using the `generateVias` or the `setGenerateViaMode` commands.

## Verilog Netlist

- The netlist should be unique.
  - Set the `init_design_uniquify` global variable to 1.

## Timing Constraints

Timing constraints in the form of SDCs are required. You should have an SDC file for each operational mode required for analysis.

## Extraction

A Quantus technology file is used by Innovus to accurately extract parasitics and is required for each RC corner in order to run extraction. A Quantus tech file is recommended for postRoute extraction for 65nm and below and for both preRoute and postRoute extraction for 32nm and below. For older technologies, a capacitance table (captable) file can be used with the native extractor of Innovus, but a Quantus tech file is required for tQuantus, IQRC, and signoff Quantus QRC.

## Signal Integrity (SI) Libraries

Noise models are required for performing Signal Integrity (SI) analysis and optimization to fix delay and glitch violations due to crosstalk. The noise models can be defined in the ECSM or CCS libraries, or separately in the form of cdb libraries.

## Multi-Mode Multi-Corner (MMMC) Setup for Timing

Multi-Mode Multi-Corner (MMMC) setup is required for optimizing and analyzing designs over multiple operating conditions. It defines the view(s) to analyze for setup and hold. Each view is defined by an operating mode and a delay corner. The operating mode is a set of SDC constraints used for timing analysis in that mode. A delay corner is made up of a library set, operating

conditions, and RC corner. See the [Configuring the Setup for Multi-Mode Multi-Corner Analysis](#) in the "Importing and Exporting Designs" chapter of the *Innovus User Guide* for information on defining the MMMC environment.

## Data Validation

This section explains how to identify problems when importing the data and the checks you can run to catch data issues early in the flow.

## Loading the Design

Once you have prepared the necessary data, it is ready to be imported. Use the Design Import form or load a global variables file and run `init_design` to import the libraries, netlist, and timing environment.

```
source design.globals  
init_design
```

The `init_design` command executes a number of checks to validate the data and highlight problems. It is important that you review the log file to understand and resolve the warnings and error messages that it reports. The Log Viewer (*Tools - Log Viewer*) can make debugging the log file easier by highlighting error and warning messages.

Look for the following when reviewing the `init_design` output:

- `init_design` reports cells in the LEF file that are not defined in the timing libraries. Look for the following and confirm if these cells need to be analyzed for timing:

```
**WARN: (ENCSYC-2) : Timing is not defined for cell INVXL.
```

- A blackbox is an instance declaration in the netlist for which no module or macro definition is found. Unless your design is being done using a blackbox style of floorplanning, there should be no blackboxes in the design. If there are blackboxes to be reported, be sure to load the Verilog file that defines the logic module, and make sure you include the LEF file that defines the macro being referenced in the netlist. The following is reported for blackbox (empty) modules:

```
Found empty module (bbox) .
```

- Verify that the netlist is unique. The following is reported if it is not unique:

```
*** Netlist is NOT unique.
```

- By default, Innovus utilizes a "footprintless" flow. This means that instead of relying on the "footprint" definitions inside the timing libraries, it uses the "function" statement to determine cells that are functionally equivalent and can be swapped during optimization. Additionally, it identifies buffers, inverters, and delay cells. Inconsistencies in how the cell functions are defined can lead to sub-optimal or erroneous optimization results. Review the log file to confirm that the buffers, inverters, and delay cells are properly identified. Below is an example of what you will see:

```
List of usable buffers: BUFX2 BUFX1 BUFX12 BUFX16 BUFX20 BUFX3 BUFX4 BUFX8 BUFXL
CLKBUFX2 CLKBUFX1 CLKBUFX12 CLKBUFX16 CLKBUFX20 CLKBUFX3 CLKBUFX4 CLKBUFX8
CLKBUFXL

Total number of usable buffers: 18

List of unusable buffers:

Total number of unusable buffers: 0

List of usable inverters: CLKINVX2 CLKINVX1 CLKINVX12 CLKINVX16 CLKINVX20
CLKINVX3 CLKINVX4 CLKINVX8 CLKINVXL INVX1 INVX2 INVX12 INVX16 INVX20 INVX3 INVXL
INVX4 INVX8

Total number of usable inverters: 18

List of unusable inverters:

Total number of unusable inverters: 0

List of identified usable delay cells: DLY2X1 DLY1X1 DLY4X1 DLY3X1

Total number of identified usable delay cells: 4

List of identified unusable delay cells:

Total number of identified unusable delay cells: 0 Also, look for cells that do
not have a function defined for them: No function defined for cell 'HOLDX1'. The
cell will only be used for analysis.
```

## Checking Timing Constraint Syntax

In addition to checking the libraries, `init_design` also checks the syntax of timing constraints. After running `init_design`, check for the following problems:

- **Unsupported constraints**
  - The Innovus software may not support the SDC constraints being used with the design, or the constraints may not match the netlist. If the constraints are not supported, they may need to be re-expressed (if possible) in constraints that the Innovus software does support.
- **Ignored timing constraints**

- Syntax errors can cause the tools to ignore certain constraints resulting in the misinterpretation of important timing considerations. Check for warnings or errors about unaccepted SDC constraints. The following are possible causes for ignored constraints.
  - A design object is not found. If the constraints refer to pins, cells, or nets that are not found in the netlist, then consider the following possible causes:
    - There could be a naming convention problem in the constraint file.
    - The netlist and constraints are out of sync, and a new set of constraints and/or a new netlist needs to be obtained.
  - An incorrect type of object is being passed to a constraint.
  - An option is being used incorrectly or an unknown option is used.
  - Illegal endpoints are used in assertions. Use the primary IOs (top-level ports, CK or D register pin) to define the starting and endpoints of `set_false_path` and `set_multicycle_path`. A combinatorial pin or a Q register pin is not valid.

- **Other things to consider when defining constraints**

- `set_ideal_network` will prevent optimization on these nets
- `set_propagated_clock` will limit preCTS optimization by not allowing resize on sequential elements.
- `set_dont_use`, `set_dont_touch` confirm that the proper settings are used
- Have a constraint file for every mode required for signoff timing analysis
- Understand and adjust clock uncertainty depending on the stage of the design flow (preCTS / postCTS / postRoute / signoff).

## Extraction File Checks

- Make sure the Quantus techfile and LEF files match. The layer names, widths, spacings, and pitches should be consistent between the files.
- Use the correct temperature for resistance extraction.
- If using a cap table, ensure that it is current and generated with a recent version of the `generateCapTbl` command. This ensures that the capacitance table information is used most effectively by extraction.

## Validating Timing Constraints

As described in the previous section, the `init_design` command checks the syntax of specified timing constraints. However, it is also important to ensure that the timing constraints are valid for the design. A good first-pass method is to check the zero wire-load model timing.

To validate timing constraints, use the following command:

```
timeDesign -prePlace -outDir preplaceTimingReports
```

This command generates a quick timing report using zero wire load and provides a first indication, before placement and routing, of how much effort will be required to close timing and whether the timing constraints are valid for the design. During pre-placement timing analysis, high fanout nets are temporarily set as ideal so that more immediate timing issues can be addressed first.

Additionally, you can run the command, `check_timing -verbose`, to report timing problems that the Common Timing Engine (CTE) sees.

## Checking Logically Equivalent Cells Available for Optimization

Run the `checkFootPrint` command to report any problems with footprint functions.

- If there are problems reported, run the `reportFootPrint -outfile file_name` command to create a footprints file. You can review this file to see which cells are identified as logically equivalent.
- You can edit the footprints file if needed and run `loadFootPrint -infile file_name` command to load it.
- If you made updates, run the `checkFootPrint` command again to verify that the file you loaded does not have problems.

## checkDesign command

After importing the design you can run the `checkDesign -all` command to check for missing or inconsistent library and design data. This will run a number of checks and output the results to a text file. Review the file to understand any problems that are found.

## Data Preparation and Validation for Low Power Designs

If your design is utilizing a low power flow using a Common Power Format (CPF) file, then also

check the following:

- If you are defining the MMMC setup in the CPF, make sure the CPF points to the proper libraries and constraints.
  - Note delay corner names are 'CPF-generated' so keep that in mind when attaching RC corners and derating timing.
- For low power designs utilizing power shutdown, ensure that an always-on buffer is available and usable.
- Run the `runCLP` command to verify the CPF and make sure the results are well understood before proceeding.

Optimization of leakage and/or dynamic power is typically on top of the presented flows:

- For designs where leakage is a high priority, the recommended flow is to enable leakage optimization at the beginning of the flow and allow the tool to manage the optimization. This is done by setting the following:

```
setOptMode -leakagePowerEffort high
```

- High effort leakage comes with a runtime and area penalty and is particularly time consuming because the ratio of higher VT cells drops.
- Low leakage effort, enabled via the command, `setOptMode -leakagePower low`, only does postRoute reclaim.

**Note:** This flow has the least impact on the runtime but typically results in a 5-10% loss of potential leakage savings.

- For designs where dynamic power is a high priority, dynamic power optimization can be enabled at the beginning of the flow by using the following command:

```
setOptMode -dynamicPowerEffort [ low | high ]
```

- High effort does reclaim throughout the flow.
- Low effort only performs postRoute downsizing.

**Note:** This method typically works best with a VCD or TCF to apply the activity rates properly

## Flow Preparation

Setting the design mode and understanding how extraction and timing analysis are used during the flow are important for achieving timing closure.

## Setting the Design Mode

The `setDesignMode` command specifies the process technology value and the flow effort level.

- The `setDesignMode -process` command specifies the process technology you are designing. Use this command to change the process technology dependent default settings globally for each application instead of setting several mode options. When you specify a process technology value using the `setDesignMode` command, Innovus automatically assigns coupling capacitance threshold values to the RC extraction filters. These values determine whether the coupling capacitances of the nets in a design will be lumped to the ground or not.

**Note:** In postRoute extraction mode, the grounding of coupling capacitances also depends on the capacitance filtering mode set by the `-capFilterMode` parameter of the `setExtractRCMode` command.

- The `setDesignMode -flowEffort` command is used to force every super command, such as `placeDesign`, `optDesign`, `routeDesign` and so on to use their high-effort settings and the additional non-default options. In this mode, the target is to achieve the best possible timing/yield at the expense of some increase in the CPU runtime.
  - `none` : Leaves all super commands to use their default settings. In this mode, the target is to achieve good QoR with the minimum runtime.
  - `high` : Forces all super commands to use their high-effort settings and the additional non-default options. In this mode, the target is to achieve the best possible timing/yield at the expense of increased CPU runtime.
- The following example sets the process to 45nm and effort level to high:  
`setDesignMode -process 45 -effortLevel high`

## Extraction

Resistance and Capacitance (RC) extraction using the `extractRC` command is run frequently in the flow each time timing analysis is performed. The `setExtractRCMode` options, `-engine` and `-effortLevel`, control which extractor is used by `extractRC`. The lower the effort level, the faster the extraction runs at the expense of being less accurate. Fast extraction is used early in the flow to provide a quick turnaround time so you can experiment with different floorplans and solutions. As you progress through the flow, the effort level is increased, which improves the accuracy of the extraction at the expense of the runtime.

The `setExtractRCMode -engine` option indicates whether to use the `preRoute` or `postRoute` extraction engine.

- Use the `-engine preRoute` option when the design has not yet been detail routed by NanoRoute. When the `-engine preRoute` option is set, RC extraction is done by the fast density measurements of the surrounding wires; coupling is not reported.
- Use the `-engine postRoute` option after the design has been detail routed by NanoRoute. RC extraction is done by the detailed measurement of the distance to the surrounding wires; coupling is reported. The `-effortLevel` parameter further specifies which `postRoute` engine is used for balancing the performance versus accuracy needs.

The `setExtractRCMode -effortLevel` value controls which extractor is used when the `postRoute` engine is used.

- `low` - Invokes the native detailed extraction engine. This is the same as specifying the `-engine postRoute` setting.
- `medium` - Invokes the tQuantus extraction mode. tQuantus performance and accuracy falls between native detailed extraction and IQRC engine. This engine supports distributed processing. tQuantus is the default extraction mode for process nodes 65nm and below whenever Quantus techfiles are present. **Note:** This setting does not require a Quantus QRC license.
- `high` - Invokes the Integrated QRC (IQRC) extraction engine. IQRC provides superior accuracy compared to tQuantus. IQRC is recommended for extraction after ECO. In addition, IQRC supports distributed processing. **Note:** IQRC requires a Quantus QRC license.
- `signoff` - Invokes the Standalone Quantus QRC extraction engine. This engine choice provides the highest accuracy. The engine has several run modes, thereby, providing maximum flexibility. **Note:** Quantus QRC obviously requires a Quantus QRC license.

The default value for the `-effortLevel` parameter depends on the value of `setDesignMode`. The table below shows how extraction is run based on the process and whether a Quantus tech file and/or captable is provided.

	Design – 32nm and Below		Design – Above 32nm		
	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
PreRoute Extraction	Captable not needed	Captable Present	Captable Absent	Captable Present	Captable Present
	Quantus Techfile Present	Quantus Techfile Absent	Quantus Techfile Present	Quantus Techfile Absent	Quantus Techfile Present
PostRoute Extraction	Flow without Captable	Captable-based Flow	Flow without Captable	Captable-based Flow	Captable-based Flow
	Detailed extraction is not allowed.  tQuantus extraction engine is run.	Detailed extraction is run.	tQuantus engine is run.	Detailed extraction is run.	TQuantus extraction engine is run for 65nm and below – but detailed extraction is allowed by explicit setting.  For process nodes greater than 65nm, detailed extraction is run.

## Timing Analysis

Timing analysis is typically run after each step in the timing closure flow using the `timeDesign` command. If timing violations exist, we recommend that you use the Global Timing Debug (GTD) GUI to analyze and debug the results:

- GTD is an invaluable tool that provides forms and graphs to help you view timing problems.
  - To learn more about GTD, see the [Global Timing Debug Rapid Adoption Kit \(RAK\)](#). This RAK provides a lab and includes instructions to demonstrate the features of GTD.

Initial timing analysis should not be performed after placement. Instead, we recommend that you wait until after preCTS optimization to report your initial timing. This is because of the following reasons:

- After placement, there will be some basic buffering and resizing to get cells into their characterized region of timing as defined by their look-up tables. Until the cells are in that

region, any timing analysis results are suspect.

- A placement should always be followed by an optimization run to bring the cells into line before making a timing and routability judgment. It is not uncommon to have the original "worse" placement come out better post-optimization than the "better" placement. This seems non-intuitive, but it is often the case.

## Pre-Placement Optimization

Data preparation is complete and you are now ready to implement your design. It is important to note that using options from a previous design might not necessarily apply to your current design. Therefore, we recommend that you start with the default flow (or Foundation Flow), then apply additional options based on your design requirements. Additionally, as you proceed through the flow you should investigate each flow step and validate it before moving to the next one.

The goals of pre-placement optimization are to optimize the netlist to:

- Improve the logic structure
- Reduce congestion
- Reduce area
- Improve timing

In some situations, the input netlist (typically from a poor RTL synthesis) is not a good candidate for placement because it might contain buffer trees or logic that is poorly structured for timing closure. In most cases, high-fanout nets should be buffered after placement. It is more reliable to allow buffer insertion algorithms to build and place buffer trees rather than to rely on the placer to put previously inserted trees at optimal locations. Additionally, having buffer trees in the initial netlist can adversely affect the initial placement.

Because of these effects, it can be advantageous to run pre-placement optimization or simple buffer and double-inverter removal (area reclamation) prior to initial placement. This can be accomplished by using the `deleteBufferTree` and `deleteClockTree` commands.

**Note:** By default, the `deleteBufferTree` command is run by `placeDesign`.

## Floorplanning and Initial Placement

The goals of floorplanning and initial placement include the following:

- Creating prototypes using multiple iterations with a focus on routability
- Moving toward timing-driven placement as routability stabilizes

- Adding power routing once timing and congestion converge

The initial floorplan and placement have a primary impact on the performance of a design. Innovus allows you to use prototypes to analyze various placements and floorplans before you begin the optimization process. Prototyping allows you to create a floorplan that can be implemented with high confidence before you spend time and effort on optimization and routing.

Prototyping involves multiple placement iterations that converge on a solution that meets a design's requirements for routability, timing (including clocks), power, and signal integrity. The initial floorplan drives the constraints leveraged by placement and partitioning to meet these objectives. The following steps outline a basic procedure for obtaining an initial placement.

- Run the initial placement without any regions and guides. It is best to get a baseline placement without constraining the placer.
  - Early on, use the `setPlaceMode -fp true` command to run placement in prototyping mode for a faster turnaround.
    - Prototype placement does not produce legal placement so make sure you run placement with the `setPlaceMode -fp false` command as you converge on a floorplan.
  - Use Automatic Floorplan Synthesis or the Prototyping Foundation Flow if your design contains a large number of hard macros.
    - Automatic Floorplan Synthesis is a set of natively-integrated automatic floorplan capabilities that can create a quick, prototype floorplan. Given a gate-level netlist and design physical boundary, Automatic Floorplan Synthesis can analyze the signal flow and generate a floorplan that includes automatic module and macro placement for large chips. See the [Floorplanning the Design](#) chapter in the *Innovus User Guide* for more information.
    - The Prototyping Foundation Flow utilizes Flex Models that can improve the run time by 20X, while still providing accurate area, timing and congestion analysis. The [Prototyping Foundation Flow RAK](#) provides a lab, instructions, and other information to demonstrate its features.
- Analyze the placement for timing and routability issues, and make the necessary adjustments.
- Employ module guides, placement blockages, and other techniques to refine the floorplan. The placement engine automatically detects low-utilized designs and turns on the options required to achieve an optimal placement.

## Ensuring Routability

Initial prototype iterations should focus on routability as the key to achieving predictable timing closure. You should attempt to resolve congestion before attempting timing closure. Designs that are congested are more likely to have timing jumps during timing and Signal Integrity (SI) closure. Tools such as module guides, block placement, block halos, obstructions, and partial placement blockages (density screens) are used to control the efficient routing of the design.

Use the following guidelines during floorplanning and placement to avoid congestion.

- Choose an appropriate floorplan style.
  - If possible, review a data flow diagram or a high-level design description from the chip designer to determine an appropriate floorplan style.
  - Assess different floorplan styles such as hard macro placement in periphery, island, or doughnut (periphery and island). Keep the macro depth at 1 to 2 for best CTS, optimization, and Design-for-test (DFT) results. If possible, consider different aspect ratios to accommodate a shallower macro depth. Consider using Automatic Floorplan Synthesis, the Prototyping Foundation Flow, and relative floorplanning constraints to simplify floorplan iterations.
- Preplace I/Os and macros.
  - Review hard macro connectivity and placement based on the minimum distance from a hard macro to its target connectivity.
  - Preplace high-speed and analog cores based on their special requirements for noise isolation and power domains.
- Review I/O placement to identify I/O anchors and the associated logic.
  - Verify that logic blocks and hard macros that communicate with I/O buffers are properly placed and have optimal orientation for routability. Push down into module guides to further assess the quality of the floorplan and resulting placement.
- Allow enough space between preplaced blocks.
  - Allow space between I/Os and peripheral macros for critical logic such as JTAG, PCI, or Power management logic. Use the `specifyJtag` and `placeJtag` commands prior to placing blocks.
  - Use block halos, placement obstructions or fences around blocks prior to optimization or Clock Tree Synthesis (CTS). Placement generally does not do a good job of placing cells between macros. Reduce or remove halos and obstructions after placement to make sufficient space available around macros for optimization, CTS, DRV, or SI fixing to add buffers. Placement blockages of type Soft or Partial are useful tools to control cell placements between blocks.

- Place other cells such as endcaps, welltaps, and decaps prior to placement, as required.
- Use module guides carefully.
  - Place module guides, regions, or fences only when greater control is required. Be careful not to place too many module constraints early in the floorplanning process because it is time consuming and greatly constrains the placement. Module guides should be used for floorplan refinement or hierarchical partitioning.
  - Review the placement of module guides related to the datapath and control logic relative to the associated hard macros. Datapath logic can be a source of congestion problems due to poor aspect ratios, high fanout, and large amounts of shifting. Consider tuning the locations of these module guides and lowering the density to reduce congestion.
  - Review the placement of module guides related to memories. These modules can typically have higher densities due to the inclusion of the memories.
- Reorder scan chains
  - When evaluating congestion, make sure that scan chains are reordered to eliminate "false" hot spots in the design. Failing to reorder the scan chain can cause a routable floorplan to appear unroutable. By default, placeDesign performs scan tracing and scan reordering based on global or user-specified scan reorder settings and specified or imported scan chain information. If scan chain information is missing, no reordering is performed.

## Validating the Floorplan

A congested or unroutable design at this stage will not get better during optimization. With a good floorplan you should be able to:

- Place the design in the floorplan without issues
- Create a routable placement

Make sure to consider the following when finalizing the floorplan:

- The power grid should be defined:
  - Global net connections are properly defined using the `globalNetConnect` command.
  - Nets requiring optimization should be defined as signal (regular) nets. Optimization treats nets in the SPECIALNETS as `dont_touch`.
  - PINS marked with + SPECIAL cannot be optimized.

- Followpin routing should align to rows/cells with the correct orientation (VDD pin to VDD followpin).
- Gaps between standard cells and blocks should be covered with soft or hard blockages. Placement cannot place cells in the area of soft blockages but optimization and CTS can.
- All blocks should be marked fixed (`setBlockPlacementStatus -allHardMacros -status fixed`).
- Tracks should match IO pins and placement grid (rows). Use `generateTracks` to update the tracks.

## Timing-Driven Placement

As the routability of the floorplan stabilizes, you should shift your focus to timing-driven placement.

Timing-driven placement (`setPlaceMode -timingDriven true`) and pre-placement optimization (buffer tree deletion) are enabled by default:

`placeDesign`

Placement will remove buffer trees, followed by timing driven global placement based on virtual in-place-optimization and detail placement. By default, placement will identify clock gates and place them in a good position for the rest of the flow (`setPlaceMode -clkGateAware true`). The congestion repairing effort is auto-adaptive based on the routing congestion (`setPlaceMode -congEffort auto`). If the flow effort is set to high (`setDesignMode -flowEffort high`), then depending on the design, `placeDesign` may enable adaptive placement. Adaptive placement enables optimization-aware placement to improve the local density in optimization for better congestion and timing closure.

## Placement Analysis

Once placement is complete it is important to analyze the global and local congestion to identify any areas that will be difficult to route.

- Review the trial route overflow values in the log file. Typically they should be below 1% but will depend on the design and technology.
  - The following option increases the numerical iterations and makes the instance bloating more aggressive. It also automatically enables the `congRepair` command.
- ```
setPlaceMode -congEffort high
```
- Turn on the congestion map and analyze any hot spots. Although the global congestion may

be low, a hot spot can make the design impossible to route.

- Use partial placement blockages to reduce the density in specified areas.
- There is a standalone command called `congRepair` that can be called in any part of the `preRoute` flow to attempt to relieve congestion. The command uses `trialRoute + incremental placement`. This can have a significantly detrimental effect on timing and often requires additional `optDesign` calls (and may not converge). So, use the `congRepair` command with caution.
- The following specifies whether to pull the first-level flops closer to their macro to improve the timing between the macro and flops. Use `-groupFlopToMacroLevel` and `-groupFlopToMacroList` to control how many levels of logic it applies to and which macros:  
`setPlaceMode -groupFlopToMacro true`
- Clocks are consuming more routing resources because they are routed with wider rules, greater spacing and shielding. So it is important to model this early on by reading in the clock routing constraints prior to placement. The CTS specification file should contain the routing constraints including non-default rules (NDRs), spacing and shield:  
`specifyClockTree -file cts.spec`

## PreCTS Optimization

The goal of preCTS optimization is to fix timing based on ideal clocks including:

- Setup slack (WNS - Worst Negative Slack)
- Design rule violations (DRVs)
- Setup times (TNS - Total Negative Slack)

The use of the `optDesign` command for postCTS optimization and postRoute optimization is described in the corresponding sections of this document. For more information on optimizing the design, refer to the [Optimizing Timing](#) chapter in the *Innovus User Guide*.

In the default mode, which is the high effort `optdesign` used in flow implementation, timing optimization begins with the first phase in which following transforms are called; netlist simplification, high fanout net buffering (including high fanout net ), multidriver buffering, DRV fixing at high level, and global optimization. After this stage, the state is good enough to start core optimization. By default, optimization is performed on all the negative end points. While working on WNS and TNS, the software controls timing convergence by updating the design state, placement

and routing, incrementally. Adaptively calling area reclaim allows you to control utilization at the preCTS stage. GigaOpt takes advantage of the upper layer characteristics by using layer assignment transform on the optimization of critical nets.

In low-effort mode, which is the low effort `optdesign` used for feasibility on WNS estimation, a fast and simple first phase is launched to fix global timing and then a light WNS/TNS optimization is performed. While running low-effort optimization, you can expect preCTS optimization to be three to five times faster with a clock frequency estimation, approximately in common cases, of around 5% as compared to default optimization.

## Guidelines for PreCTS Optimization

Consider the following points during preCTS optimization.

Before starting optimization, perform the following sanity checks:

- Review `checkDesign-all` results
- Check that the SDC is clean
- Check that the timing is met in zero wireload using `timeDesign -prePlace`. A short path violated in this mode will not be fixed further in the flow implementation
- Check that the NDRs are good and well selected for NDR aware optimization – too many NDRs will slow down optimization
- Check the don't use report (`reportDontUseCells`)
- Activate all required views
- Adjust settings depending on specific scenarios: high performance (timing/power), high routing congestion, high utilization, mixture

While performing timing optimization, check the following:

- Follow WNS/TNS convergence for each active path group
- Check physical update – large max move of instances, large mean move of instances, routing congestion, and so on
- Check DRVs fixing convergence
- Monitor routing congestion at different stage

## PreCTS optDesign Command Sequences

You can use `optDesign` for preCTS optimization in the following ways:

**Note:** You can use any of these features separately or in combination. Use the [setOptMode](#) command to control optimization behavior.

- To optimize timing-placed designs for the first time (with ideal clocks), use the following command:  
`optDesign -preCTS`
- To further optimize a design after you have already run `optDesign -preCTS`, use the following command:  
`optDesign -preCTS -incr`
- To perform rapid timing optimization for design prototyping, use the following commands:  
`setOptMode -effort low`  
`optDesign -preCTS`
- To perform DRV fixing only  
`optDesign -preCTS -drv`

## Checking and Debugging Timing Optimization Results

When `optDesign` completes, you will see a summary of the timing results. Additionally, you can use the command `timeDesign -preCTS` to check the current timing.

```
timeDesign -preCTS -outDir preCTSOptTiming
```

If timing violations exist, use Global Timing Debug (GTD) to analyze the violations.

- Graphically check the critical paths, not just the first one
- Investigate cell/net delay to identify bad buffering, bad sizing, and weak cell usage
- Investigate routing :
  - Scenic routing on standard cells area – rework on placement
  - Check for bad routing over macros – add soft placement blockage or modify floorplan
  - Critical paths go through congested area. Try using cell padding (using the [specifyCellPad](#) or [specifyInstPad](#) commands) or partial placement blockages (also known as density screens) to reduce the congestion – check global density and hot spot.
  - Critical path(s) go through routing congested area – review floorplan / macro placement /

narrow channels.

- Check if the worst path is going through deep logic level – If yes, see if the initial netlist is to be further improved during synthesis

Following are some common types of timing and congestion problems seen during preCTS optimization and suggestions for resolving them:

- If some nets are not being optimized, run the following to output a report of the ignored nets during optimization. Use the abbreviation in the last column with the key at the bottom of the file to determine why certain nets are not being optimized.

```
reportIgnoredNets -outfile fileName
```

- If timing after optimization is poor or degraded, check the log file for slack jumps. This can be related to physical update placement legalization/trialrouting. In this case, check routing congestion, scaling factors, and initial placement
- Run the following command to identify cells with a `set_dont_touch` or `set_dont_use` attribute. The file will identify these cells in the far right column. Ensure that the cells intended for optimization are available for use.

```
reportFootPrint -dontTouchNUse -outfile fileName
```

- If similar paths are not meeting timing, create a custom path group for these paths and optimize them separately. See the "Path Group Optimization" section for details.
- If critical paths go through congested area, then try using cell padding (running the `specifyCellPad` or `specifyInstPad` commands) or partial placement blockages (also known as density screens) to reduce the congestion.
- Evaluate the timing constraints and identify false paths that may be affecting the critical path. The Conformal Constraint Designer offers a complete, functional constraint validation solution that enables rapid timing closure when working with implementation tools. Conformal Constraint Designer includes the following:
  - Formal validation of SDC exceptions, such as false paths (FPs)
  - SDC Quality checks
  - Extensive debug and analysis that leverages both RTL and gate-level information
  - Ability to identify false paths from critical paths
  - Ability to validate the consistency between top-level and lower-level constraints

- Useful skew is often required on hard to close timing designs. In preCTS, useful skew optimization advances and delays sequential elements such that preCTS optimization can continue to optimize other paths. In postCTS, useful skew optimization delays sequential elements and physically adds additional delay through the use of buffers. Use the `setUsefulSkewMode` command to control the buffer types and other options used by useful skew optimization. To enable useful skew optimization run the following command:  
`setOptMode -usefulSkew true`
- If clock gating checks are on the critical path, you can create separate path groups for the clock gating cells and over-constrain them. Following is a sample script to do this. Set the `$clkgate_target_slack` to your desired value. Use caution when over-constraining designs as it increases the runtime:

```
# Set target overshoot slack for clock gating elements preCTS (in nanoseconds):
set clkgate_target_slack 0.15

# Create separate reg2reg and clkgate groups
group_path -name reg2reg -from all_registers -to [filter_collection
[all_registers] "is_integrated_clock_gating_cell != true"]

group_path -name clkgate -from [all_registers] -to [filter_collection
[all_registers] "is_integrated_clock_gating_cell == true"]

setPathGroupOptions reg2reg -effortLevel high -criticalRange 1
setPathGroupOptions clkgate -effortLevel high -targetSlack $clkgate_target_slack
-criticalRange 1
```

- The risk for timing optimization on designs with high-routing congestion mainly comes from nets detouring that are unpredictable. In addition to floorplan and placement recommendations made previously, it is usually beneficial to identify the weak cells and to set them as `dont_use`.

```
setDontUse cellName(s)
```

**Note:** It is recommended to use `setDontUse` rather than the SDC `set_dont_use`. This is because `setDontUse` applies to all operating modes while `set_dont_use` only affects the operating mode(s) to which it is applied.

- Allowing M1 routing can help to reduce congestion and, therefore, remove some pessimism during RC extraction:

```
setTrialRouteMode -useM1 true
```

- Monitor the density increase. If needed, reduce or remove extra margins on setup target and DRV. Usually the density starts blowing up when trying to fix the last tens of picoseconds and the better timing seen in preCTS will lead to flow divergence later on. Properly setting the setup target slack will reduce area and improve the runtime. For example, if the target is -0.5ns, it might be helpful to set "setOptMode -setupTargetSlack -0.2". The same applies to DRV fixing by using the -drcMargin option:

```
setOptMode -setupTargetSlack slack -drcMargin value
```

## Path Group Optimization

You can focus timing optimization on specific paths using path groups. If path groups are not defined, optDesign will temporarily generate two high-effort path\_groups (reg2reg and reg2clkgate).

- A path\_group can be set as “low” or “high” effort.
  - A high-effort path\_group will receive a higher focus from the optimization engine than a low-effort one
- All high-effort path groups defined are optimized at the same time.
- You can add slack adjustment and priority to any path group using the [setPathGroupOptions](#) command
  - By default, optDesign will use the slack adjustment value that leads to the worst slack
  - The priority is used when an endpoint is part of several path groups so the software can choose which adjustment value to use

The flow to create and optimize path groups is as follows:

```
group_path -name path_group_name -from from_list -to to_list -through through_list  
setPathGroupOptions...  
optDesign -preCTS -incr
```

Creating path groups is not mandatory to achieve the best results:

- Too many custom path groups may impact the run time significantly.
- Too many overlapping or nested path groups may impact TNS timing closure.

## Clock Tree Synthesis

The traditional goal of CTS is to buffer clock nets and balance clock path delays. From the software's 14.2 release onwards, the default engine for performing this is CCOpt-CTS. CCOpt-CTS can automatically generate a clock tree specification from multi-mode timing constraints and then synthesize and balance clock trees to that specification. CCOpt extends CCOpt-CTS by adding Clock Concurrent Optimization, which simultaneously optimizes clock and datapath to achieve better performance, area, and power.

For details on the capabilities and configuration of CCOpt-CTS and CCOpt, an overview of CTS engines in Innovus, and an introduction to the concepts and terminology used, see the [Clock Tree Synthesis](#) chapter. For the legacy FE-CTS capabilities, see the [Legacy FE-CTS Capabilities](#) chapter.

The [CCOpt Rapid Adoption Kit \(RAK\)](#) provides a self-contained example with design data and with instructions and scripts for running CCOpt-CTS and CCOpt.

## Configuring CCOpt-CTS or CCOpt

For complete command examples, see the [Flow and Quick Start](#) section in the Clock Tree Synthesis chapter.

The key steps and commands for a typical setup are as follows:

1. Load post-CTS timing constraints. This is essential for CCOpt since CCOpt combines CTS with post-CTS style optimization. This is also recommended for CCOpt-CTS. Note that because CCOpt-CTS and CCOpt compute latency adjustments to maintain correct inter-clock and I/O timing over the transition from ideal to propagated mode timing, you must arrange for postCTS timing constraints which do not contain the `set_propagated_clock` commands. For details, see the [Flow and Quick Start](#) section in the Clock Tree Synthesis chapter.
2. Configure non-default routing rules (NDRs) and route types using the `create_route_type` and `set_ccopt_property route_type` commands.
3. Set a target maximum transition time, and for CCOpt-CTS a target skew, using `set_ccopt_property target_max_trans` and `set_ccopt_property target_skew` commands.

4. Configure which library cells CTS should use using the `set_ccopt_property buffer_cells, inverter_cells, clock_gating_cells, and use_inverters` properties.
5. Create a clock tree specification from active timing constraints using the `create_ccopt_clock_tree_spec` command.

For more recommendations on settings, see the [Configuration and Method](#) section in the Clock Tree Synthesis chapter.

## Running CCOpt-CTS or CCOpt

To run CCOpt-CTS to perform CTS with global skew balancing, use the following command:

```
ccopt_design -cts
```

To run CCOpt to perform CTS with Clock Concurrent Optimization, use the same command but without the `-cts` parameter:

```
ccopt_design [-outDir <reports directory>] [-prefix <reports prefix>]
```

CCOpt and CCOpt-CTS automatically do the following:

- Detail route clock nets using NanoRoute
- Switch timing clocks to propagated mode and update source latencies to maintain correct I/O and inter-clock timing. There is no need to use the `update_io_latency` command after the `ccopt_design` command and doing so will not give valid results. For details, see the [Source Latency Update](#) section in the Clock Tree Synthesis chapter.

## Reporting after CCOpt-CTS or CCOpt

To report timing after CTS, use the `timeDesign -postCTS` command. The `-outDir`, `-prefix` and other parameters can be used as with earlier flow steps.

Reports on clock trees and skew groups can be obtained using these CCOpt reporting commands:

- `report_ccopt_clock_trees -filename clock_trees.rpt`
- `report_ccopt_skew_groups -filename skew_groups.rpt`

For more information on clock trees and skew groups, see the [Concepts and Clock Tree Specification](#) section in the Clock Tree Synthesis chapter. For more information on reporting capabilities, see the [Reporting](#) section in the Clock Tree Synthesis chapter.

## Visualization of Clock Trees after CCOpt-CTS or CCOpt

The CCOpt Clock Tree Debugger (CTD) permits interactive visualization and debugging of clock trees. Choose the *Clock* menu in the main Innovus window and select *CCOpt Clock Tree Debugger*. A graphical representation of the clock tree alongside an insertion delay scale will appear. The CCOpt CTD permits a variety of functions including path highlighting and cross probing with the placement view. For more information, see the [CCOpt Clock Tree Debugger](#) section in the Clock Tree Synthesis chapter of the *Innovus User Guide* and the CCOpt Clock Tree Debugger section in the [Clock Menu](#) chapter of the *Innovus Menu Reference*.

## PostCTS Optimization

The goals of postCTS optimization include:

- Fixing remaining design rule violations (DRVs)
- Optimizing remaining setup violations
- Correcting timing with propagated clocks
- Optimizing Hold timing violations

## PostCTS SDC Constraints

**Note:** For flows which deploy full CCOpt, and not just CCOpt-CTS, additional postCTS setup optimization is not normally required. Furthermore, the recommend flow for CCOpt is to load postCTS timing constraints before invoking CCOpt. CCOpt is discussed further in [Clock Tree Synthesis](#).

At this point in the design flow, the clocks are inserted and routed. Since timing analysis uses the actual clock delays, you should adjust the timing constraints as follows. Typically, designers have separate SDC files for preCTS and postCTS timing analysis. Use the `update_constraint_mode` command to update the SDC files for each operating mode.

- Set the clocks to propagated by adding the following to your SDC file(s):  
`set_propagated_clock all_clocks`
- Adjust the clock uncertainty (`set_clock_uncertainty SDC`) to model only jitter. You need to update the constraints after clock tree synthesis to adjust clock jitter according to the design. Modeling only the jitter avoids making the timing appear worse than it is. Remember that, since the actual clock skew data is now available, it is possible that the critical path timing will be worse.

- Remove or change the SDC constraints that are not valid postCTS like `clock_uncertainty` or `clock_latency`. You may need to adjust the clock latencies on the IOs so that IO timing does not become the critical path by adjusting the virtual clock source latencies:

```
set_clock_latency -source xxx clock
```

**Note:** CCOpt and CCOpt-CTS automatically adjust source latencies. If FE-CTS is used, you can adjust IO constraints directly (`set_input_delay`/`set_output_delay`) or use the `update_io_latency` command.

- Adjust derating applied per each delay corner.
- Make sure RC scaling factors are tuned properly.
  - Clock routing will use a different scale factor than the signal nets.
    - Signal nets use `-preRoute_cap`
    - Clock nets use `-postRoute_clkcap` (if defined) or `-postRoute_cap`.

- Run timing analysis to check the timing:

```
timeDesign -postCTS -outDir ctsTimingReports
```

- If the timing is not similar to preCTS timing results, check the following:
  - Was CTS able to achieve the skew constraints?
  - Are these skew constraints within the `set_clock_uncertainty` set during preCTS timing?
  - Was the clock uncertainty adjusted after CTS to only model jitter?

## PostCTS Setup Optimization Command Sequences

**Note:** For flows which deploy full CCOpt, and not just CCOpt-CTS, that additional postCTS setup optimization is not normally required. CCOpt is discussed further in [Clock Tree Synthesis](#).

Typically the same options applied during preCTS optimization are used for postCTS optimization.

- Use the `timeDesign` command to check the postCTS timing:

```
timeDesign -postCTS -outDir postctsTimingReports
```

The timing at this point should be similar to the preCTS timing results. If there is a large jump in negative slack, investigate whether the end points are clock gating cells. A jump in negative slack may occur for clock gating cells now that their real skew is used for timing analysis. If the large slack is occurring at other points, you should use GTD to analyze the paths. Double-check that the clocks

are propagated and that the skew is within your specifications.

- To optimize timing after the clock tree has been built, use the following commands:  
`optDesign -postCTS -outDir postctsOptTimingReports`
- To take advantage of useful skew when optimizing timing in postCTS mode using incremental mode, use the following commands:  
`setOptMode -usefulSkew true`  
`optDesign -postCTS -incr`

## Hold Optimization

At this point run timing analysis to report hold violations:

```
timeDesign -postCTS -hold -outDir postctsHoldTimingReports
```

Timing optimization to fix hold violations can be performed at this point. This is recommended if your design has a significant number of hold violations because they are easier to fix prior to routing. If the number of hold violations is low you can wait until after routing the signal nets. To perform hold optimization:

```
optDesign -postCTS -hold -outDir postctsOptHoldTimingReports
```

Hold optimization will insert cells and perform resizing to fix hold violations while minimizing the effect on setup timing. It will minimize the number of cells added by inserting buffers at the common points that fix violations at multiple end points.

Following are recommendations to achieve closure for hold timing:

- Ensure that the Hold timing uncertainty is realistic
  - Too large a value can cause the insertion of thousands more buffers
- ensure that delay cells are allowed to be used and avoid providing very weak buffers for Hold fixing because they are more sensitive to routing detour and SI
- For Multi-Vth design, ensure that you run leakage optimization (through optDesign or optLeakagePower) before running Hold fixing since leakage reduction improves hold timing.
- Add cell padding during placement to leave more space for hold fixing.
  - Cell padding must be removed before postCTS Hold fixing
- Ensure that timing constraints are correctly in sync between setup and hold (especially multicycle paths)
- Ensure that the skew in hold is also good.

- A good clock tree for hold is as important as for setup
- For a design with many hold violated paths, it is highly recommended to run hold fixing at the postCTS stage already (although there is no need to achieve 0ns slack at this stage). Then, use postRoute hold fixing to fix the remaining violations. You can use a negative hold target slack to focus hold fixing on the paths with large violations and fix the remaining hold violations after routing. For example, the following sets a hold target slack of -200ps:  
`setOptMode -holdTargetSlack -0.2`
- By default, Hold fixing can degrade setup TNS (but not Setup WNS). This can be changed through the following:  
`setOptMode -fixHoldAllowSetupTnsDegrade true | false`
- To exclude some path\_group/clockDomains from hold fixing you can apply the following:  
`setOptMode -ignorePathGroupsForHold {groupA groupB...}`
- In the `innovus.logv` file hold fixing will print a detailed report where each net that is not buffered will be sorted through several categories.
  - This will allow the user to identify what are the most critical issues to resolve.
  - It will help the user in understanding why a given net was not buffered.

Below is an example:

---

---

```
=====
Reasons for remaining hold violations
=====
*info: Total 1 net(s) have violated hold timing slacks.

*info:      1 net(s): Could not be fixed as the violating term's net is
marked IPO ignored.
```

- The `setOptMode -fixHoldAllowOverlap` command controls if hold fixing is limited to purely legal moves (no overlaps). When set to "true", hold optimization allows initial cell insertion to overlap cells and then refinePlace legalizes the cells placement. This provides optimization more opportunity to fix violations. When set to the default, the value of "auto" hold optimization is allowed to create overlaps during postCTS optimization but not during postRoute optimization. To allow overlaps during postRoute optimization as well set the following:  
`setOptMode -fixHoldAllowOverlap true`

- Control hold time margins
  - Beyond certain hold margin, delay buffer addition rate increases exponentially
  - Try useful skew optimization for RAM and Register files

## Checking Timing

You can use the following commands to report setup and hold time violations after postCTS optimization.

```
timeDesign -postCTS -outDir postctsOptTimingReports
```

```
timeDesign -postCTS -hold -outDir postctsOptTimingReports
```

## Detailed Routing

After postCTS optimization, there should be few, if any, timing violations left in the design. The goals of detailed routing include the following:

- Routing the design without DRC or LVS violations. NanoRoute performs a DRC and cleans up violations.
- Routing the design without degrading timing or creating signal integrity violations.
- Using DFM techniques such as multi-cut via insertion, wire widening, and wire spacing to optimize the yield.

NanoRoute performs timing-driven and SI-driven routing concurrently. NanoRoute routes the signals that are critical for signal integrity appropriately to minimize cross-coupling between these nets, which would lead to postRoute signal integrity issues. Additionally, it can perform multi-cut via insertion, wire widening, and spacing to optimize the yield.

For 65nm and below, abstracts for MACRO's (RAM/ROM) should have cover blockages that overlap the pins rather than pin cut-outs from OBS (blockage). When the blockage is cut around the pins, it triggers a number of spacing checks that NanoRoute is forced to check. To avoid these checks, cover the pins with the blockage. NanoRoute is allowed to access pins covered by a blockage.

## Routing Command Sequence

The following example shows the use of NanoRoute to perform detailed routing. If filler cells containing metal obstruction other than the followpins are to be used, ensure that they are inserted prior to the initial route.

```
routeDesign
```

The `routeDesign` command automatically sets "`setNanoRouteMode -routeWithTimingDriven true -routeWithSiDriven true`". If you are using `globalDetailRoute` in place of `routeDesign` make sure you manually set these options to `true`. `routeDesign` will automatically unfix clock nets (`setNanoRouteMode -routeDesignFixClockNets false`) so it can ECO route the clock nets after postCTS optimization and resolve DRC violations.

PostRoute wire spreading significantly reduces SI impact. It is enabled by default when "`setDesignMode -flowEffort high`" is set. To run it separately after `routeDesign`:

```
setNanoRouteMode -routeWithTimingDriven false  
setNanoRouteMode -droutePostRouteSpreadWire true  
routeDesign -wireOpt  
setNanoRouteMode -droutePostRouteSpreadWire false
```

For many RC corners at 40nm and below, vias are extremely expensive for resistance. Achieving the highest possible double cut coverage helps reduce the resistance for vias that cannot be eliminated:

```
setNanoRouteMode -routeConcurrentMinimizeViaCountEffort medium  
setNanoRouteMode -drouteUseMultiCutViaEffort medium | high
```

## Improving Timing during Routing

The following tips can help achieve better timing results during the routing phase of the design.

- Make sure the LEF file contains a sufficient number and variety of vias (hammer-head, stacked, and so forth).
  - Check with your library provided or foundry for the latest technology LEF to use.
- Check the definition of tracks in the DEF file. If the tracks are poorly defined, regenerate tracks with the `generateTracks` command.
- If timing is way off and/or there is local or global congestion, return to postCTS optimization and optimize further or run non-timing-driven routing.

- Make sure the top max routing later is set appropriately.
- Specify the required NonDefaultRules (NDRs) and/or shield routing.

## PostRoute Extraction

All nets are now routed. It is important to now set the extraction mode to postRoute and specify the extractor to use.

- Specify the engine to postRoute:

```
setExtractRCMode -engine postRoute
```

- Set the `-effortLevel` so that `extractRC` uses your desired extractor:

```
setExtractRCMode -effortLevel low | medium | high | signoff
```

- **low** - Invokes the native detailed extraction engine. This is the same as specifying the "`-engine postRoute`" setting.
- **medium** - Invokes the tQuantus extraction mode. tQuantus performance and accuracy falls between the native detailed extraction and IQRC engine. This engine supports distributed processing. tQuantus is the default extraction mode for process nodes 65nm and below whenever Quantus techfiles are present. **Note:** This setting does not require a Quantus QRC license.
- **high** - Invokes the Integrated QRC (IQRC) extraction engine. IQRC provides superior accuracy compared to tQuantus. IQRC is recommended for extraction after ECO. In addition, IQRC supports distributed processing. **Note:** IQRC requires a Quantus QRC license.
- **signoff** - Invokes the Standalone Quantus QRC extraction engine. Quantus QRC provides the highest level of accuracy. It can be used if postRoute optimization TAT is not a concern. **Note:** Quantus QRC obviously requires a Quantus QRC license.

## Checking Timing

Use the following command to do a postRoute timing check on non-SI timing to compare with the preRoute timing. Remember to reset the `-SIAware` parameter to `true` to enable SI optimization in the next steps:

```
setDelayCalMode -SIAware false  
timeDesign -postRoute -outDir postrouteTimingReports
```

```
timeDesign -postRoute -hold -outDir postrouteTimingReports
```

If timing jumps at this point compared to timing before routing, check the following:

- One reason for timing jumps is that the extractors are not correlated properly. RC Scaling factors are recommended to correlate the parasitic extractors of Innovus with your signoff extractor. This provides more accurate timing and predictability throughout the flow. Use Ostrich to obtain the parasitic measurements to determine the appropriate scaling factors. This command calculates the capacitance factors by comparing the Innovus extraction with the results of either Quantus Extraction or a SPEF file. Please see the solution [How to Generate Scaling Factors for RC Correlation](#) for the steps to generate the correlation factors with Ostrich.

**Note:** Once the scaling factors are determined, specify them in your MMMC setup using the `create_rc_corner` or `update_rc_corner` commands.

- Is the routing topology similar between trialRoute and NanoRoute? Compare the same paths between postCTS and postRoute databases and for large differences in loads.

## PostRoute Optimization

During postRoute optimization, there should be minimal violations that need correction. The primary sources of these timing violations include:

- Inaccurate prediction of the routing topology during preRoute optimization due to congestion-based detour routing
- Incremental delays due to parasitics coupling

Since the violations at this stage are due to inaccurate modeling of the final route topology and the attendant parasitics, it is critical at this point not to introduce any additional topology changes beyond those needed to fix the existing violations. Making unnecessary changes to the routing at this point can lead to a scenario where fixing one violation leads to the creation of others. This cascading effect creates a situation where it becomes impossible to close on a final timing solution with no design rule violations.

One of the strengths of postRoute optimization is the ability to simultaneously cut a wire and insert buffers, create the new RC graph at the corresponding point, and modify the graph to estimate the new parasitics for the cut wire without re-doing extraction.

In addition to the timing violations caused by inaccurate route topology modeling, the parasitics cross-coupling of neighboring nets can cause the following problems that need to be addressed in high speed designs:

- An increase or decrease in incremental delay on a net due to the coupling of its neighbors and their switching activity.
- Glitches (voltage spikes) that can be caused in one signal route by the switching of a neighbor resulting in a logic malfunction.

These effects need to be analyzed and corrected before a design is completed. They are magnified in designs with small geometries and in designs with high clock speeds.

## Data Preparation for SI analysis

SI optimization requires the following preparation:

- Make sure ECSM/CCS noise models or cdB libraries are provided for each cell for each delay corner.
- You must be in the On Chip Variation (OCV) mode to see simultaneous clock pushout / pullin and enable Clock Path Pessimism Removal (CPPR). Enable this using:  
`setAnalysisMode -analysisType onChipVariation -cppr both`
- Enable SI CPPR through "`set_global_timing_enable_si_cppr true`" (this is the default)

Additionally, consider the following techniques if you have difficulty achieving signal integrity closure on your design.

- Watch for routing congestion during floorplanning and especially after detailed routing.
  - Consider running the `congRepair` command on the design at the `preRoute` stage to eliminate local hot spots or adjust your floorplan
- Use NanoRoute advanced timing with SI-driven routing options during detailed routing. These are automatically enabled when running `routeDesign`:

```
setNanoRouteMode -routeWithTimingDriven true  
setNanoRouteMode -routeWithSiDriven true
```

- Fix transition time violations. This is automatically done as one of the first steps when the `optDesign -postRoute` command is run.
  - Slow transitions introduce a larger delay penalty or incremental delay.

## PostRoute Optimization Command Sequences

The command sequence to fix postRoute setup and hold violations is:

```
optDesign -postRoute
```

```
optDesign -postRoute -hold
```

## Analysis and Debug of PostRoute Optimization Results

Use Global Timing Debug to debug any remaining violations. If violations remain, consider the following suggestions:

- For multi-VT designs, you can perform a LEF-Safe Optimization with only cell swapping by setting the following:  

```
setOptMode -allowOnlyCellSwapping true
```

```
optDesign -postRoute
```

  - Doing so after a normal `optDesign -postRoute` may help close timing if the design is congested and those final few paths were detoured by NanoRoute's ECO routing.
  - Doing so prior to a normal `optDesign -postRoute` may speed up closure in terms of turnaround time.
  - Only works for multi-VT libraries
  - Does not have much impact when design has already mainly LVT cells.
- The local density by default is limited to 98%. If optimization is prematurely exiting due to local placement hotspots this can be increased to 100% using:  

```
setOptMode -maxLocalDensity 1.0
```
- Make sure your extraction filters correlate to your signoff extraction
  - When using IQRC the filters typically can be set to the exact signoff values
    - Make sure to use `setExtractRCMode -capFilterMode relAndCoup`
    - StarRC has no `total_c_th` equivalent so set this to 0 for these flows

Below are suggestions to help achieve SI Closure:

- SI prevention in the preRoute flow for data path can be achieved by adding more pessimism to force optimization to work harder. This can be done by increasing the clock uncertainty.
  - Choose reasonable values to avoid over fixing and increasing area/power too much
- You can apply a targeted approach for cases with the timing paths with very large depth (> 40)

- Sum of small SI push-out on long paths leads to large timing penalty.
- Solution is to add a pessimism only on nets that are part of the large depth path.
- Check the global max transition and ensure that wire spreading is enabled during detailed routing.

## Optimizing With Third-Party SPEF or SDF

If you are using a third-party tool for extraction or timing analysis, the most important step is to make sure they correlate with the corresponding function in Innovus. For example, if you are using a third-party extractor, make sure the RC scaling factors are set properly within Innovus. If you are using a third-party timing analysis tool, make sure it correlates with Innovus by verifying that SDC constraints are applied consistently between the tools. The delay calculation and timing analysis results between Innovus and the third-party tool should also correlate.

If timing violations occur when using SPEF from a third-party extractor you can import the SPEF into Innovus and perform optimization. You must import a SPEF for each RC corner. The flow is:

```
spefIn rc_corner1.spef -rc_corner rc_corner1
spefIn rc_corner2.spef -rc_corner rc_corner2
...
optDesign -postRoute [-hold] -outDir spefFlowTimingReports
```

The `-hold` option is optional in the above command. Use `-hold` if you need to perform hold fixing based on the SPEF.

The `optDesign` command will use the SPEF for initial timing to determine the best location to optimize the paths.

You can use the SDF data generated by an external tool in Innovus to do limited SI fixing. SDF data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

Use the `read_sdf` command to load an SDF file for each view. For each view, two SDFs are required from the external tool; one with base timing only and the other with full timing with base and SI Timing.

Use the following commands to load separate SDF files for two setup views:

```
setDelayCalMode -SIAware false
read_sdf -view <viewname1> -reset_preexisting_derate <view1_base_timing>.sdf
read_sdf -view <viewname1> -reset_preexisting_derate -infer_delta_delay
```

```
<view1_full_timing>.sdf
read_sdf -view <viewname2> -reset_preexisting_derate <view2_base_timing>.sdf
read_sdf -view <viewname2> -reset_preexisting_derate -infer_delta_delay
<view2_full_timing>.sdf
```

Enable the external SDF based SI setup fixing flow:

```
optDesign -postRoute -useSDF
```

Note that in order to use SDF information in MMMC analysis mode, you should provide an SDF file for each active view in the design. If a view was not given an SDF file, Innovus will run detailed extraction and delay calculation for that specific view.

The `optDesign` command will run extraction to calculate slew values because SDF does not contain slews. The delays in the SDF will be used, but the slews must be generated. You can also use `spefIn` to avoid the extraction.

## Chip Finishing

After postRoute optimization is performed, you may need to add filler cells and metal fill shapes to meet the DRC rules. The filler cells are sometimes added as part of postRoute optimization, but not always. They are used to fill the gaps between standard cells in the standard cell rows to fill in the device layers like pwells or nwells, and to meet any implant layer width or spacing DRC rules. The normal cells have implant layers but they may not meet the width and spacing rules of the implant layers without abutting filler cells that use the correct implant layers. An example of the command to insert filler cells is provided below:

```
addFiller -cell filler_cell_list -prefix FILLER -doDRC true -honorPrerouteAsObs true
```

Sometimes, the library provides filler cells with built in decoupling capacitors (decap). You can also insert these fillers to improve the voltage or IR drop, normally at the expense of higher leakage currents. An example of the command to insert decap filler cells is provided below:

```
addFiller -cell decap_filler_list -prefix FILLER_DECAP -doDRC true -honorPrerouteAsObs true -area {x1 y1 x2 y2}
```

Metal fill, also called dummy metal, is used to make the metal density more uniform by adding small, floating, metal-fill shapes in empty areas. You can either use Innovus to add metal fill, or use a physical verification tool. The Innovus metal fill can meet the DRC rules for older process nodes, but for newer process nodes such as 28nm and below, the Innovus metal-fill rules are generally not sufficient and you must use physical verification tools. Metal fill has some special DRC rules in addition to the normal metal shapes that are defined in the technology LEF file. These are listed below.

```
[PROPERTY LEF58_FILLTOFILLSPACING "FILLTOFILLSPACING spacing ;"]  
[FILLACTIVEESPACING spacing ;]  
[MINIMUMDENSITY minDensity ;]  
[MAXIMUMDENSITY maxDensity ;]  
[DENSITYCHECKWINDOW windowLength windowHeight ;]  
[DENSITYCHECKSTEP stepValue ;]
```

You can use the `setMetalFill` command to overwrite the metal-fill rules in the technology LEF file. The `setMetalFill` command can define different metal-fill settings. If you do not specify the iteration name (using the `-iterationName` parameter), the setting will be saved to a "default" iteration name.

Use the `addMetalFill` command to insert metal fill based on the rules defined by the `setMetalFill` command. In Innovus, the `addMetalFill` command also supports the metal-fill connect to power/ground (PG) mesh. The metal fill can carry current to improve the IR drop. Use `addMetalFill -net` to specify the PG net to be connected.

After metal fills are inserted, use the `verifyMetalDensity` command to verify the metal density rules defined in the technology LEF file and by the `setMetalFill` command. The `verifyMetalDensity` command only honors the default iteration name setting. Ensure that you have defined the default iteration name before running the `verifyMetalDensity` command.

There are two flows that can be used to insert metal fill. Use the commands listed below to insert metal fills in Innovus:

```
setMetalFill -activeSpacing value -gapSpacing value -maxWidth value -maxLength value -  
windowSize x y -windowStep x_step y_step -minDensity value -maxDensity value  
  
addMetalFill  
  
verifyMetalFill
```

Use the commands listed below to insert sign-off metal fill in the design. This flow calls the Cadence Physical Verification System (PVS) application for sign-off metal-fill insertion. This is the recommended flow.

```
streamOut -mapFile gds_map -outputMacros -units unit gds_file  
  
run_pvs_metal_fill -ruleFile PVS_RULE_DECK -defMapFile map_file -gdsFile gds_file -cell  
gds_top_cell
```

In Innovus, you can run the `timeDesign` command to check the timing. If the timing has degraded, you can trim the metal fill from critical nets for timing closure. Use the command below to trim the

metal fill around critical nets:

```
setMetalFill -windowStep x_step y_step -windowSize x y  
trimMetalFillNearNet -slackThreshold $slack1 -spacing value -spacingAbove value -  
spacingBelow value -minTrimDensity value
```

## Timing Sign Off

The goal of timing sign off is to verify that the design meets the specified timing constraints. This is accomplished by first using Quantus to generate detailed extraction data, and then using the specified timing analysis engine for a final analysis of setup and hold data.

At this point in the design process, final routing and postRoute optimization is complete.

The following command sequence generates the reports needed to verify timing by calling the Tempus timing analyzer. A Tempus license is required:

```
timeDesign -signoff -outDir signOffTimingReports  
timeDesign -signoff -hold -reportOnly -outDir signOffTimingReports
```

These commands perform the following operations:

1. Run Quantus to generate detailed parasitics (Quantis QRC license required).
2. Use the detailed parasitics and generate the setup timing reports using Tempus.
3. Generate the hold timing reports.

If the timing degrades compared to postRoute timing, check whether the postRoute RC scaling factors that correlate to the signoff extractor are properly set. If you are using a third-party extractor, use `spefIn` to read the SPEF for each RC corner, then run `timeDesign` using the `-reportOnly` option:

```
spefIn rc_corner1.spef -rc_corner rc_corner1  
spefIn rc_corner2.spef -rc_corner rc_corner2  
...  
timeDesign -signoff -reportOnly -outDir signOffTimingReports  
timeDesign -signoff -hold -reportOnly -outDir signOffTimingReports
```

## Final Timing Analysis and Optimization using Tempus/Quantus

Although Innovus has a mode to time the design in the signoff mode, it does not always have the environment that corresponds exactly to what the final signoff timing analysis will use. For example, a signoff tool may propagate full waveforms for delay calculation where an optimization tool would model the waveform as a linear ramp; and a signoff tool will use path-based AOCV where an optimization tool would use graph-based AOCV.

These differences between the implementation tool and signoff tool on timing are because of the runtime consideration during timing optimization. The implementation tool cannot always work with the full timing accuracy level otherwise TAT would be a concern. You can use the `signoffTimeDesign` and `signoffOptDesign` commands to analyze and optimize the timing generated by Tempus Signoff STA and Quantus Signoff extraction while remaining in the implementation tool cockpit.

For more information about how the commands are used and the template scripts that are available, see [Using Signoff Timing Analysis to Optimize Timing](#) in the *Innovus User Guide*.

## Additional Resources

Following are additional application notes, training and documentation related to timing closure. These can be found at <http://support.cadence.com>:

- [Recommended NanoRoute Options with emphasis on 32nm and below advance node/technology](#)
- [Innovus Foundation Flows User Guide](#)
- [How to Generate Scaling Factors for RC Correlation](#)

# Hierarchical and Prototyping Flow

- [Introduction](#)
- [Top-down and Bottom-up Hierarchical Methodologies](#)
  - [Top-down Methodology Steps](#)
  - [Bottom-up Methodology](#)
- [Hierarchical Floorplan Considerations](#)
  - [Hierarchical Methodologies](#)
- [Hierarchical Partitioning Flow and Capabilities](#)
  - [Hierarchical Partitioning](#)
- [Chip Planning](#)
  - [FlexModel](#)
  - [Timing Net Delay Model with Pico-second Per Micron \(psPM\)](#)
  - [Prototyping Flow](#)
- [Supporting Giga-Scale Designs in Planning stage](#)
  - [Active-logic Reduction Technology](#)
- [Top-level Timing Closure](#)
  - [Using Interface Logic Models \(ILM\)](#)
  - [Using Flexible Interface Logic Models \(FlexILM\)](#)
- [Chip Assembly](#)

## Introduction

Most of the system-on-a-chip devices are designed in a traditional flat flow that avoids the effort to set up a design hierarchy. However, as the design size grows beyond a few million instances, Flat design flow becomes unviable due to huge memory requirements and excessive run time. Hierarchical Flow capabilities are essential to divide and conquer the design process – where the design can be divided into manageable partitions, and each partition can be independently assigned to different design groups to be developed in parallel.

However, Hierarchical Flow presents new challenges – such as increased complexity of Prototyping, Planning and Partitioning the design, Hierarchical timing closure at the top level, and

managing the late ECOs. Design planning and hierarchical timing closure contribute significantly to Hierarchical design turnaround time. These challenges need advanced planning and modeling capabilities.

Hierarchical design can be divided into three general stages: chip planning, implementation, and chip assembly.

- **Chip Planning**

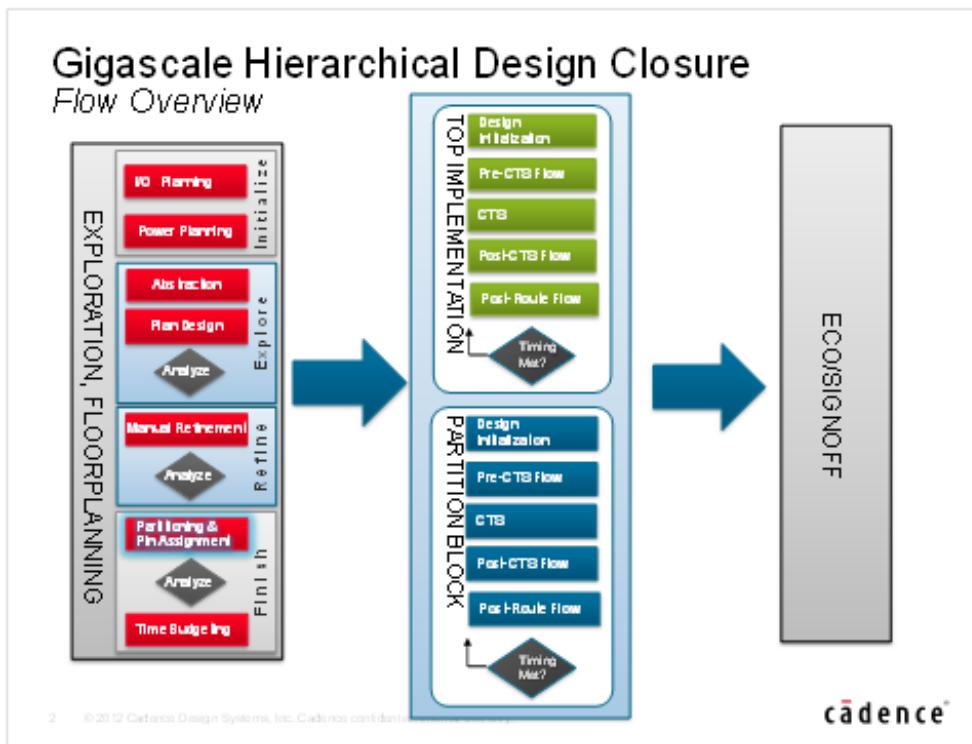
Breaks down a design into block-level designs to be implemented separately.

- **Implementation**

This stage consists of two sub-stages: block implementation for a block-level design and top-level implementation for a design based on block-level design abstracts and timing models.

- **Chip Assembly**

Connects all block-level designs into the final chip.



A few important considerations in choosing the Hierarchical design methodology are:

- Whether to use top-down or bottom-up methodology
- Whether to use traditional channel-based designs, channel-less designs, or a variation that provides benefits of both the approaches
- Whether advanced tool support exists, such as floorplanning and pin-assignment for multi-

level hierarchical designs, single-pass timing closure for interface paths

## Top-down and Bottom-up Hierarchical Methodologies

The top-down methodology usually consists of the top-down planning, implementation, and chip assembly stages. Use this methodology to create a top-level or hierarchical floorplan from a flat floorplan based on fenced modules. In this approach, the die size, shape, and I/O pads locations drive block and partition placement. Block-level design size and pins are generated based on the top-level floorplan. This approach is conducive to early and efficient planning of resources, pins, feedthrough paths and timing, and aims to deliver better performance, area, timing and power planning. It also enables concurrent implementation of blocks and top-level design. However, it requires advanced tool support to handle the full design at early stages and focus on early design planning.

The bottom-up methodology consists of the implementation and assembly stages. In the bottom-up methodology, the size, shape, and pin position of block-level designs drives the top-level floorplanning. It does not need planning ahead, but runs the risk of performance, area, timing, and power not being fully optimal and long design iterations between block and top-level implementations.

## Top-down Methodology Steps

### Chip Planning

Refer to the "Chip Planning" section of the [Partitioning the Design](#) chapter. The Chip Planning section describes the for the most common flow for chip planning which includes specifying partitions and blackboxes.

### Top and Block Implementation

After the chip planning, the next stage is to implement the individual blocks. The detail of each block is implemented using the constraints for timing, size, and pin assignment determined during the planning stage. Block implementation should be done at a block directory that is generated by the `savePartition` step. At the completion of this step, block abstracts, timing models, a DEF file, and a GDSII file should be generated to use in top-level implementation and chip-assembly. Refer to the “Design Implementation” section of the user guide for more information.

The next step is to implement the top-level designs with block model data such as LEF, timing

model such as .lib and ILM, and FlexILM (Refer to top level timing closure section for more information), power model, and noise model.

## Chip Assembly

For the top-down approach, see the "Chip Assembly" section of the [Partitioning the Design](#) chapter for information on how to bring together all the top-level and block-level netlists and routing information.

## Bottom-up Methodology

### Implementation

Each block in the design must be fully implemented. This includes place and route as well as clock, power, and I/O.

### Block Implementation

The size of a block-level design can be derived or adjusted using the *Floorplan - Specify Floorplan* menu command or the `floorPlan` text command. The Innovus software can support a rectilinear block-level design. You can use the same procedure to create a rectilinear partition to create a rectilinear block-level design using the following steps. Refer to the "Design Implementation" section of the user guide for more information.

### Top-level Implementation

After block implementation, physical and timing abstract models (ILM/FlexILM) should be developed for each block-level design that will be used in the top-level implementation. For the bottom-up approach, create a top-level floorplan where block-level abstracts are referenced in the top-level design.

## Chip Assembly

For the top-down approach, see the "Chip Assembly" section of the [Partitioning the Design](#) chapter to understand how to bring together all the top-level and block-level netlists and routing information.

**Note:** For the bottom-up approach, do not use the `-topFP` option of the `assembleDesign` command.

## Hierarchical Floorplan Considerations

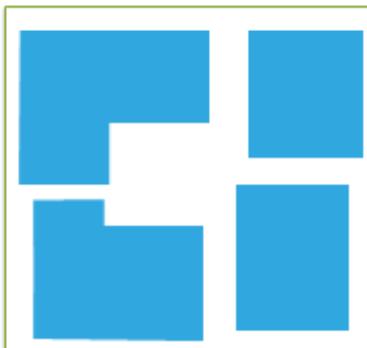
- Prototype the design to plan up-front as much as possible
- Pick a partition size that is optimal for a flat block implementation
- Match logical and physical hierarchy
- Find better partitioning in terms of:
  - Minimal top-level timing paths or logics
  - Simple partition interface timing
  - Minimal pin count
- Create a hierarchical Verilog module wrapper if feedthrough insertion is needed
- Do register-bounded on the partition interface
  - Interface latches are not good for any modeling, for example ILM

## Hierarchical Methodologies

Following are the methodologies that can be used for hierarchical designs.

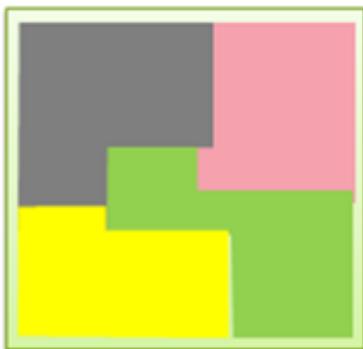
### Channel-based Methodology

Channel-based methodology is a well-established methodology. It is simple, can accommodate hardened IPs easily, and does not generally need the Feedthrough buffers through partitions. However, it may lead to sub-optimal results in terms of area and timing.



## Channel-less Methodology

Channel-less methodology may give 5-7% die-area saving and has the advantage of not having any top-level logic. However, it requires careful planning for multi-fanout nets leaving a partition boundary, precise alignment of pins for abutted partitions, complex master-clone (repeated blocks) configurations, highly-customized clock trees and iterative process, and pin connections for hardened IPs. It also needs Verilog module wrappers for LEC validation due to the netlist changes owing to imminent Feedthrough paths.



## Narrow Channel Methodology

A good balance between the Channel-based and Channel-less methodologies is the proven Narrow (or Thin) Channel Methodology. It does not have strict requirements for pin locations or Feedthrough paths, can work easily with hardened IPs, offers more flexible clock-tree topology choices, and leads to faster convergence. It does have a minor area penalty compared with channel-less designs.



## Hierarchical Partitioning Flow and Capabilities

The foundational Innovus capabilities for Hierarchical Flow are given below. For more information on these capabilities, see [Partitioning the Design](#) chapter of the *Innovus User Guide*.

### Hierarchical Partitioning

#### Capabilities

This is about partitioning the design logically and physically into the top-level design and the various partition blocks, and pushing down all the relevant design data to the partition blocks. It provides capabilities for defining and handling partition blocks, blackboxes, various orientations, rectilinear shapes, multiply-instantiated partitions, multiple-levels of partitions, partition guard-bands, and so on. It creates different directories for the top-level and the blocks, which can then be independently implemented by different design teams.

#### Pin Assignment

Innovus offers very strong automatic pin-assignment capabilities for partitions and blackboxes, which include:

- Abutted (channel-less) designs
- Multiple levels of hierarchical design
- Master and clone designs
- Flip chip (area IO designs)
- Rectilinear partition shapes

- Route-based pin assignment
- Placement-based pin assignment

Pin assignment allows users to set a variety of powerful constraints to guide the pin assignment for optimizing as per the specific needs of the design. The constraints can be set at the level of the design, individual partition(s), or specific pins or their groups. Any contentions are handled by a neatly-defined precedence rule.

A very useful capability for pin assignment is the Interactive Pin-Editor, which allows custom pin placement for specialty usage, such as spreading a group of pins in any of the variety of schemes across layers in a given area, placing pins of non-preferred layers, while still conforming to the user-specified pin constraints.

It also offers a host of capabilities for checking the assigned pins, legalizing any violating pins, and aligning pins across a routing channel.

## Feedthrough Insertion

The `insertPtnFeedthrough` command inserts feedthrough nets and buffers into partitions on the way, to avoid routing those nets over partition areas (to avoid conflict with partition's routing resources), or detouring to avoid the partition (to avoid longer routes and resulting delays). It is absolutely necessary in channel-less designs, and may be useful in channel-based designs too. Innovus can insert Feedthroughs, both based on just placement or along the routes if design has been routed. It has advanced capabilities such as inserting correct buffers for Multi-Supply Voltage designs, optimally reusing buffers across even the abutted multiply-instantiated (Master/Clone) partitions, and reusing buffers for multi-fanout nets.

For more information, see "Multiple Supply Voltage Top-Down Hierarchical Flow" in the [Low Power Design](#) chapter of the *Innovus User Guide*.

## Timing Budgeting

In hierarchical design flows, chip-level timing constraints must be mapped correctly to corresponding block-level constraints while partitioning the design. Innovus automatically apportions budgets to blocks using a path-based method. Since the budgeting is run on an early-stage non-optimized design, Innovus performs a virtual design optimization to derive more accurate timing budgets for the partitions that allows better design convergence. It also has a faster mode that avoids this virtual optimization.

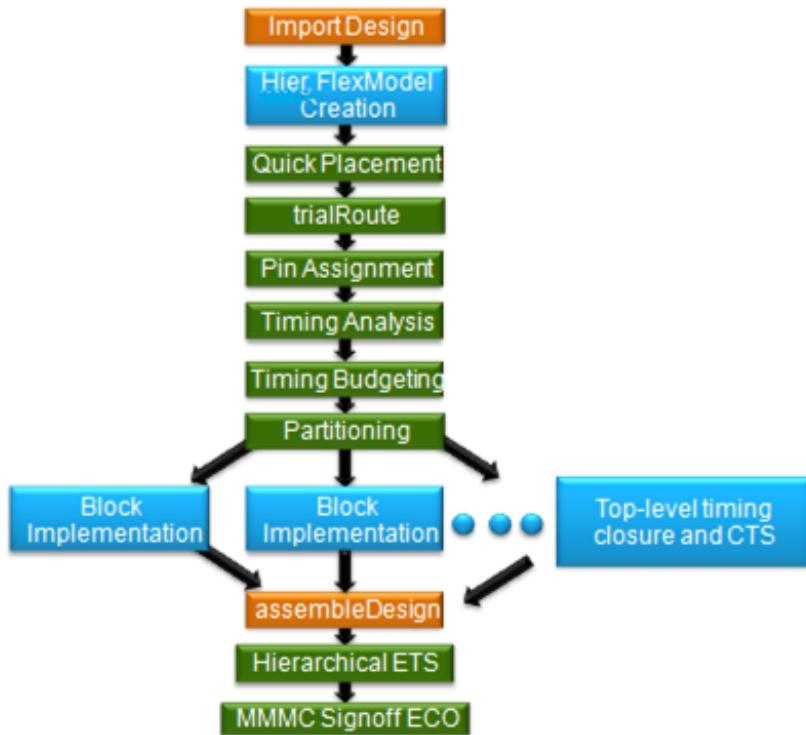
## Assembling Partitions

The assemble design capability brings back partition data for nested partitions. It first restores the top design, assembles the parent partitions, and then brings back all child nodes partitions. It ensures that all references of master and clones (which may be at different levels of hierarchy in different partitions) are assembled properly.

## Hierarchical Partitioning Flow

The flow chart below shows the most common flow for chip planning.

For more information on these flow steps, see the [Partitioning the Design](#) chapter of the *Innovus User Guide*.



For more information on these flow steps, see the [Partitioning the Design](#) chapter of the *Innovus User Guide*.

**Note:** The use of FlexModel is optional in the flow diagram above. If FlexModels are not used, the FlexModel creation step is not needed. Please refer back to the FlexModel section for detailed

information. The main difference between the prototyping flow and the hierarchical partition flow is the automatic floorplan synthesis step ([proto\\_design](#)).

## Chip Planning

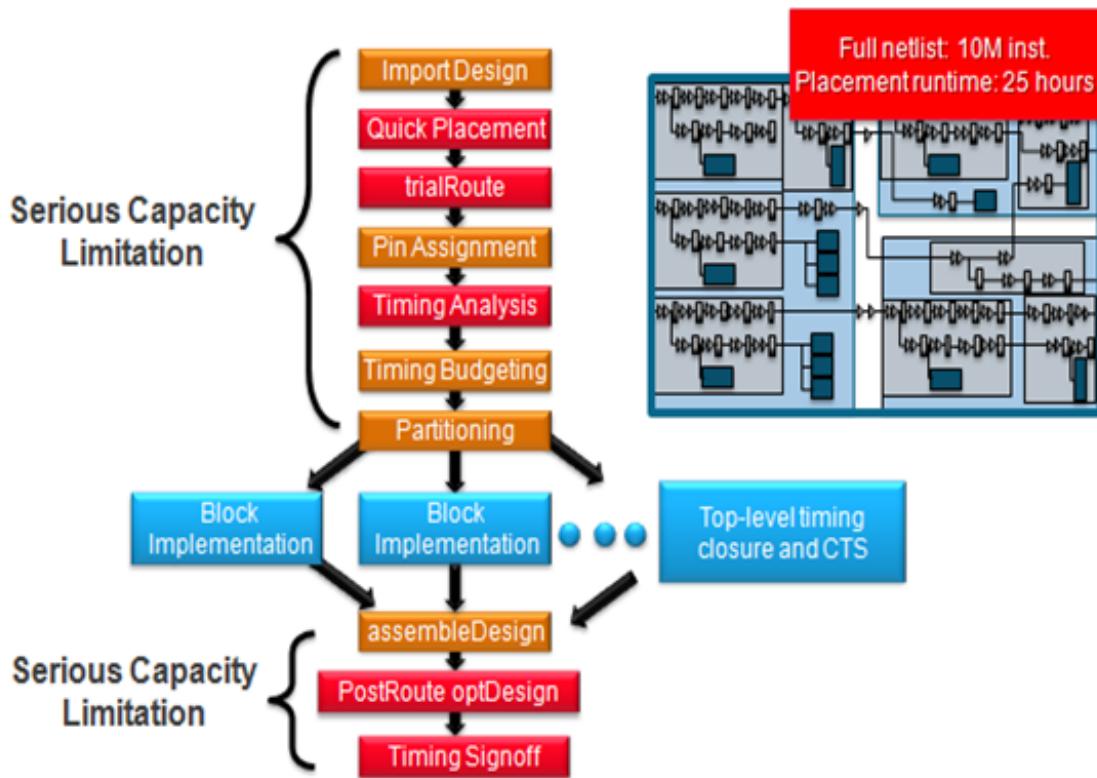
Innovus allows you to do productive chip planning and concurrently handle multiple design objectives (Capacity / Turn Around Time / Abstract models / Optimization) with the following flow methodologies.

## FlexModel

FlexModel can be used in hierarchical partition implementation flow for reducing netlist size and/or prototyping flow for design exploration and planning. With FlexModel abstraction, netlist can be reduced up to 20x. This netlist reduction allows all Innovus applications to run up to 20x faster, while still enabling fairly accurate timing, area, and congestion analysis. The accuracy helps the resulting floorplan to be “implementable” in nearly one pass, rather than going through expensive iterations to converge on the floorplan.

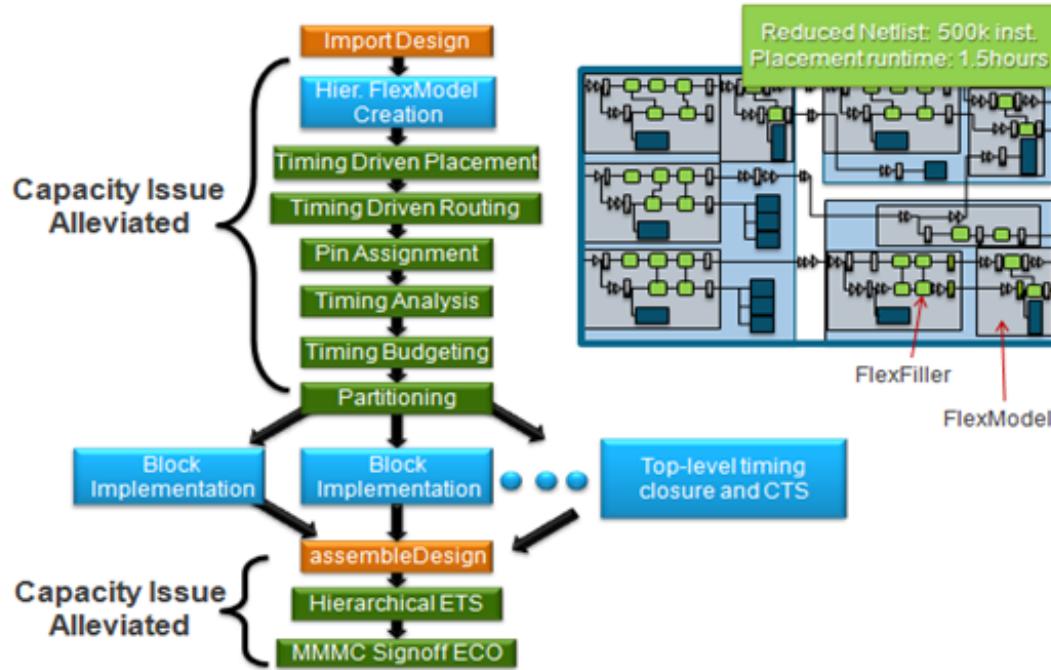
Challenges of implementing a Gigascale design are the capability limitation and long run time issue.

## Bottlenecks for a Big Design Flow



To address these challenges, FlexModel can be used to fasten the overall process. The FlexModel flow requires an additional FlexModel generation step after restoring the design. Other than that, the flow is almost the same as the normal hierarchical flow.

## Removing Bottlenecks for a Big Design Flow

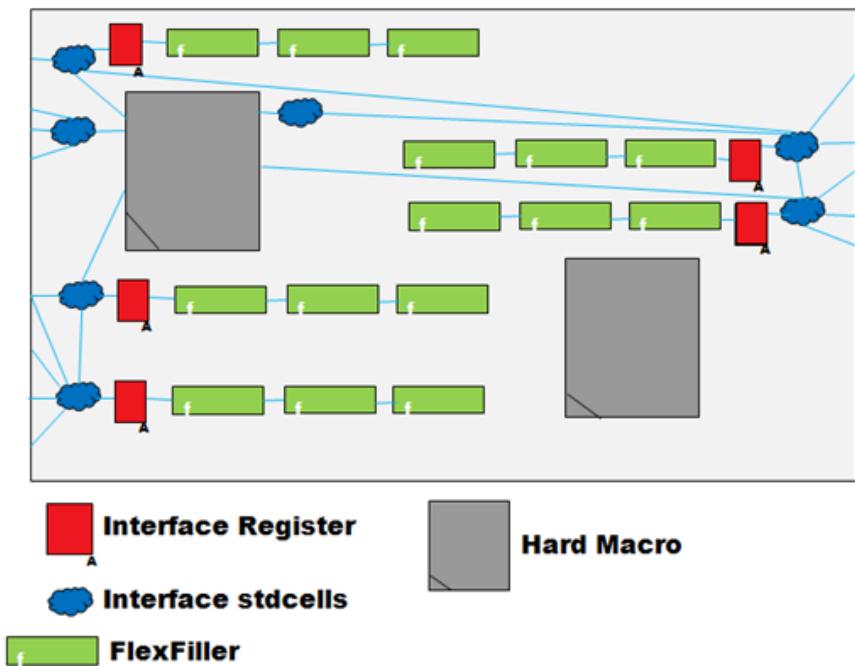


## FlexModel - Introduction

FlexModel can be a Verilog module or an instance group. It contains macros, interface standard cells, and FlexFillers. FlexFillers fill in the space for the internal register-to-register logic. They do not have timing models associated with them, and connect such that the placer will place them close together in one group. This helps in accurate timing and area estimation.

A flexModel netlist is usually one-tenth the number of instances of its full netlist. It is used during early design planning to reduce the run time and memory while accurately modeling the timing and area.

A flexModel can be created even in early stages of the design where the netlist is not complete.



## FlexModel Prototyping Flow Stages

Following are the prototyping flow stages:



- Create FlexModels
  - Identify models
  - Create models
  - Create timing net delay model: Pico-second-Per-Micron model (psPM)
- Debug Constraints and PlanDesign
  - Debug constraints
  - Generate a good initial floorplan that can be used as starting point for manual modification
- Analyze Floorplan and Adjust
  - Manually move FlexModels to shorten timing paths
  - Re-analyze floorplan
- Define Partitions
  - Define partitions based on FlexModel placement
  - Optionally re-place macros within partition fences
  - Optionally manually re-shape and position FlexModel regions within partition partitions
  - Generate partition fences
- Finish and Save Partitions
  - Placement and timing driven trial route
  - Commit/Save partitions
  - Pin assignment, feedthrough insertion, budgeting
  - Power, bus and pipeline planning

For more information on creation of FlexModel, model generation, see the [Prototyping Methodologies](#) chapter of the *Innovus User Guide*.

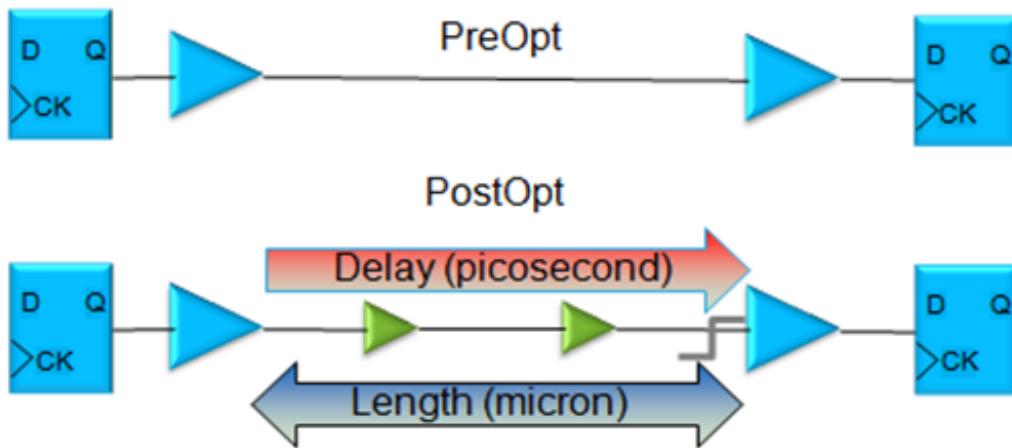
## Timing Net Delay Model with Pico-second Per Micron (psPM)

Pico-second per micron model (psPM model) is the timing net delay model that is used for fast timing estimation without requiring the users to optimize the design. This timing model can be used for the hierarchical implementation flow and/or for the prototyping flow.

Pico-second per micron model is for modeling virtual buffering effect that dominates for long nets used for quick timing estimation. It is characterized based on net length, routing layer, and the number of fanouts. Innovus uses the current loading library technology to create a sample design with 2 pin nets and multiple fanout nets. GigaOpt is used to optimize these nets to derive net delay values that are called pico-second per micron (psPM).

With prototype timer that uses psPM models, timing can be accurately estimated with a quick TAT. Since the same gigaOpt engine (optDesign) is used for characterizing and generating psPM

models, timing results with psPM models are correlated well with optDesign –preCTS results (~10%)



- psPM = (total delay including virtual buffer delay) / (net length)
- Input transition is also considered
- Delay and slew are optimized using “`optDesign -preCTS`”

psPM models can be created using the `create_ps_per_micron_model` command. They can also be generated automatically during the partition based FlexModel generation step.

## Prototyping Flow

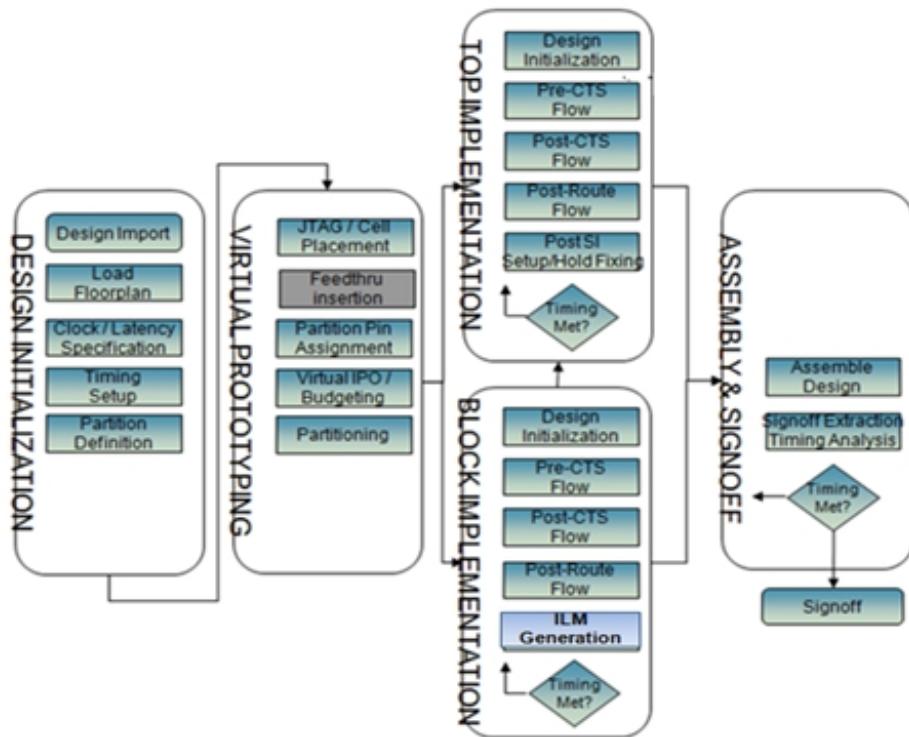
### Overview

For a GigaScale design, it may take many weeks and/or months to generate an implementable floorplan. Innovus has provided a comprehensive prototyping flow that allows designers to find real problems in minutes rather than days so an implementable floorplan can be obtained quickly. Prototyping flow is a subset of the Innovus hierarchical flow. During prototyping an early flow methodology is built upon, which then serves to provide specification for a real implementation flow.

Prototyping flow allows you to do productive chip planning and concurrently handle multiple design objectives. This flow provides:

- **Capacity:** It can handle more than 100 million instances in concurrent timing and congestion driven mode.
- **Turnaround Time:** This is a progressively converging flow and enables you to run:

- Global placement of modules
- Incremental macro placement
- Detailed standard cell placement
- Fast accurate timing analysis using the estimated net delay model called pico-second per micron model

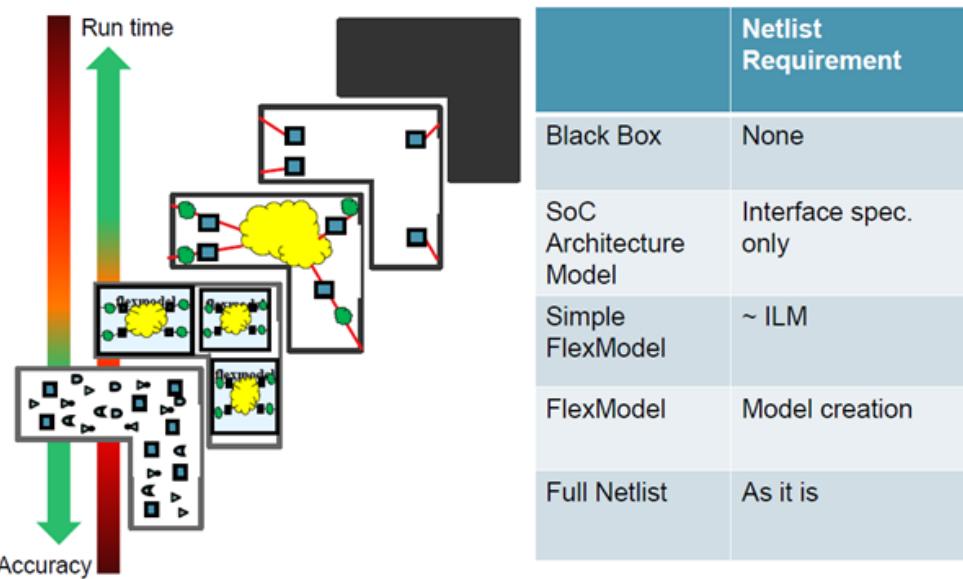


## Prototyping Methodologies

Innovus prototyping flow supports different abstract models such as:

- Black box
- Soc Architecture Information (SAI)
- FlexModel

## Floorplanning Block(Partition) Modeling



As shown in the above diagram, there is a tradeoff between accuracy and run time. As the details in the models increase, the accuracy improves. However, it comes at the cost of increased run time and memory.

For example, usage of the BlackBox model during prototyping enables quicker run time but lesser accuracy.

Next in line is the SoC Architecture Model, where some minimum details are covered till the boundary flops. It provides more accuracy in terms of Quality of results, but with a marginal increase in run time.

FlexModel improves the modeling accuracy even further, leading to better accuracy but with slight increase in runtime.

Partition can be defined as FlexModel or it can have more than one FlexModel per partition.

## BlackBox

Normally a blackbox is a module with content that is not well defined. However, a well-defined module can also be defined as a blackbox. A blackbox is similar to a hard block, but like a fence, a blackbox can be resized, reshaped, and have pins assigned. After a blackbox has its pins assigned and is partitioned, it behaves like a hard block. The blackbox feature can be used only with a partitioned design.

After the netlist has been loaded, you can further specify which modules or cells will be regarded as blackboxes, or modify the existing blackbox sizes.

A blackbox size can be specified in terms of an estimated area (an actual value or an area value in terms of gate count), or a fixed block width and height.

## Blackbox-based Flow

The following flow specifies blackboxes with an original netlist that has modules with content that is not well-defined:

- Import the design.  
**Note:** Users can import their blocks even if they are partially defined, or undefined, and then convert them into Blackboxes to continue with prototyping flow. During importing, the Innovus software by default would keep the empty modules.  
Specify the blackboxes or load a floorplan file with blackbox information.
- Floorplan the design.
- (Optional) Save the design, which saves the blackbox information.
- Run placement.
- (Optional) Run `trialRoute`. When you run `trialRoute` with this parameter, `trialRoute` determines near-optimal location for blackbox pins with respect to top channel congestion and places blackbox pins at these locations. `trialRoute` then creates routes to the blackbox pins without crossing over blackboxes.

## Saving Blackbox

Proceed with the normal hierarchical flow for the design.

To save blackbox information, use the `saveDesign` command or the File - Save Design menu command.

## Reshaping Blackbox

During `planDesign`, a blackbox can be reshaped (within specified aspect ratio range) to minimize overlaps. This reshape is based on the minimum and maximum values for the aspect ratio range while maintaining the current area.

The master and clone blackboxes are reshaped such that the clone blackbox take the same size and shape as its master while meeting orientation constraints.

## Deleting Blackbox

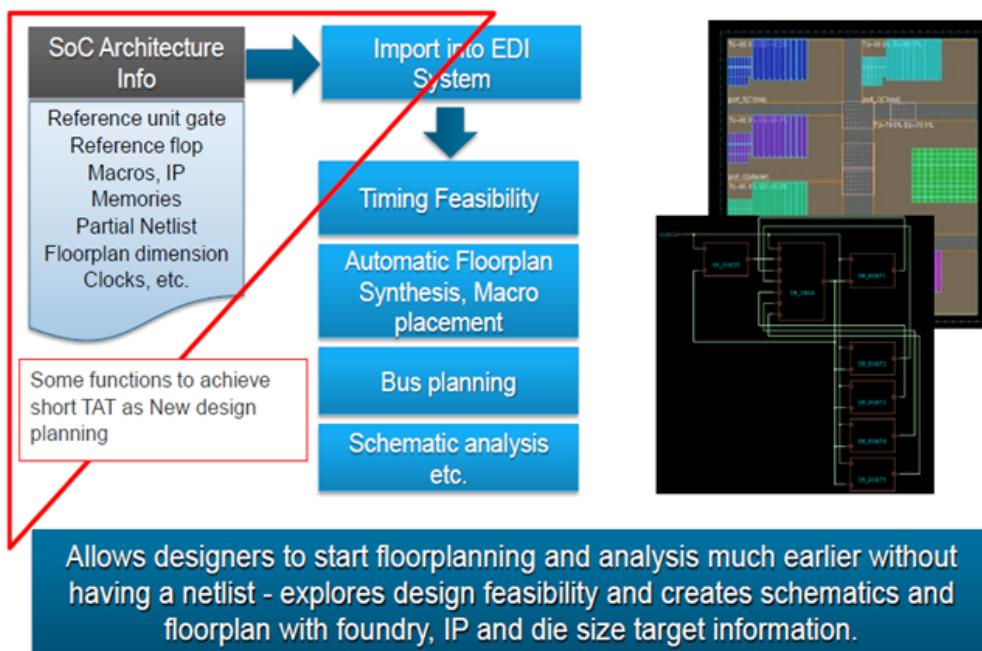
If a blackbox is an empty module in the netlist, or a cell without a physical macro definition, you must modify the netlist before you can delete it.

## Soc Architecture Info (SAI)

Soc Architecture Information (SAI) is new methodology for prototyping that is available from the 14.1 version of the software onwards. SAI allows design exploration floorplan creation and analysis at a much earlier stage in the design flow.

It is very useful in very early chip size study, hierarchical floorplan analysis. Using this methodology you can begin design feasibility without a complete netlist and get to enable Innovus floorplanning.

## Early Floorplanning and Exploration using SoC Architecture Information



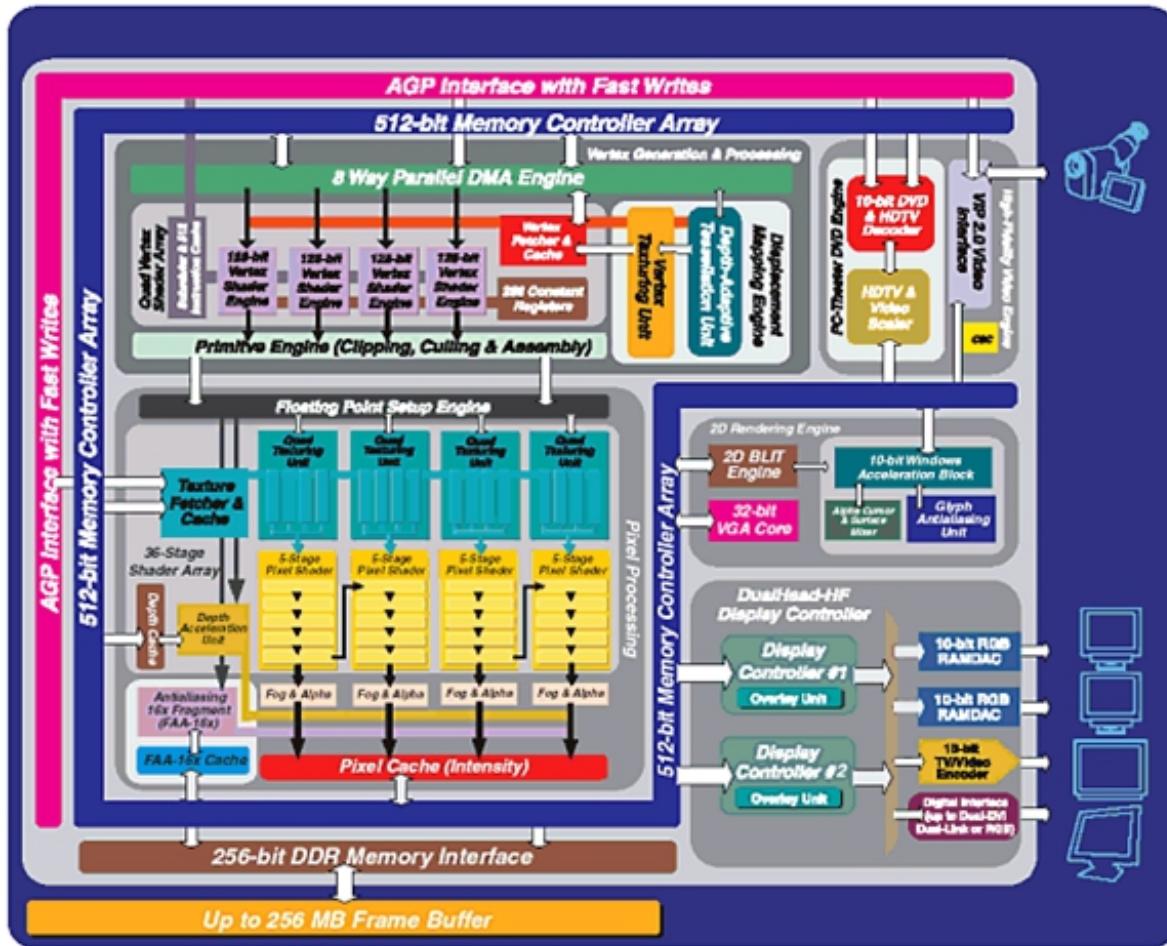
### SoC Architecture Info (SAI) File

SAI flow is simple and is mainly driven by the way the Chip Architecture and IPs information is captured:

- Reference unit “gate” from the foundry specification
- Reference flop
- Macros or special IPs
- Memory with different sizes and ports
- Bus connection
- Soft modules (partitions)
- Clocks
- Floorplan dimension

SAI architecture as seen can contain the details, such as reference gate, IP information, reference flop, partitions, partial netlist, clock, and floorplan information.

Basically one can build upon the design by providing this content information using the SAI commands.



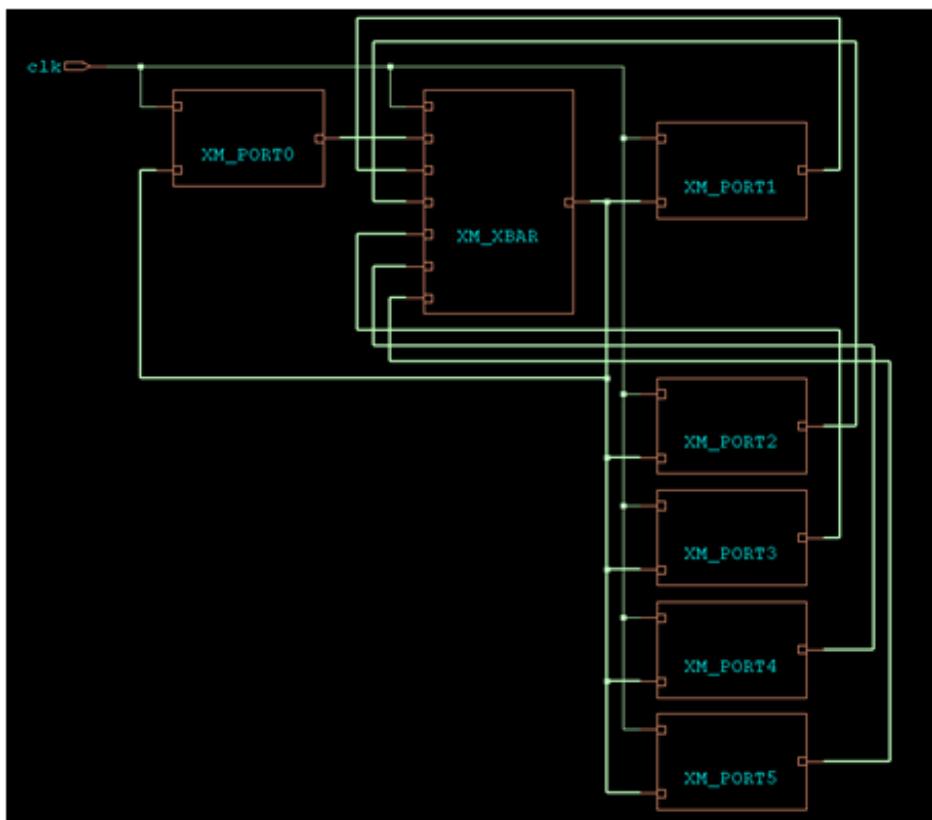
Through SAI, you can create an ideal mechanism for communicating between front-end and back-end designers.

## Netlist Creation

SAI can be used to generate an initial netlist for floorplanning. This netlist can be generated in a few minutes and be ready for Innovus floorplanning.

## Innovus Schematics Display

You can also visualize the content that has been built upon from SAI using the Innovus schematic display.



### Automatic Netlist Creation Flow with SoC Architecture Info(SAI)

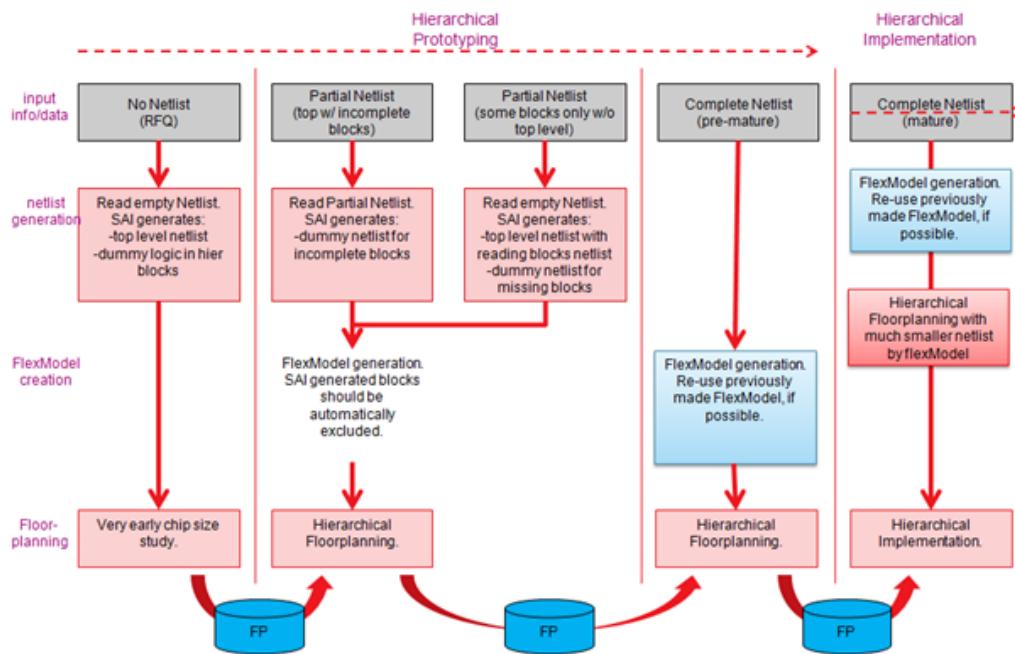
Based on specified SAI architecture information, Innovus creates a netlist to enable Innovus floorplanning as follows:

- Parse a partial netlist and the SAI file to create a netlist
- Add dummy cells to mimic the size of the define modules (RFQ)
- Basically add gut information to the module
- Add a dummy flop to mimic the boundary connection from module to module
- Add a dummy memory or the real memory in order to assist subchip floorplan
- Add pipe line stage registers as per the specification in the connection file
- Create a sdc timing constraints file for the defined boundary connection and read into Innovus
- Define the die size and read into Innovus
- Create all the net groups and pipeline net groups
- Handle master-clone where connection model is single-driver but multiple receiver

Once the netlist has been created, you should go thru the same flow as Innovus floorplanning/prototyping flow.

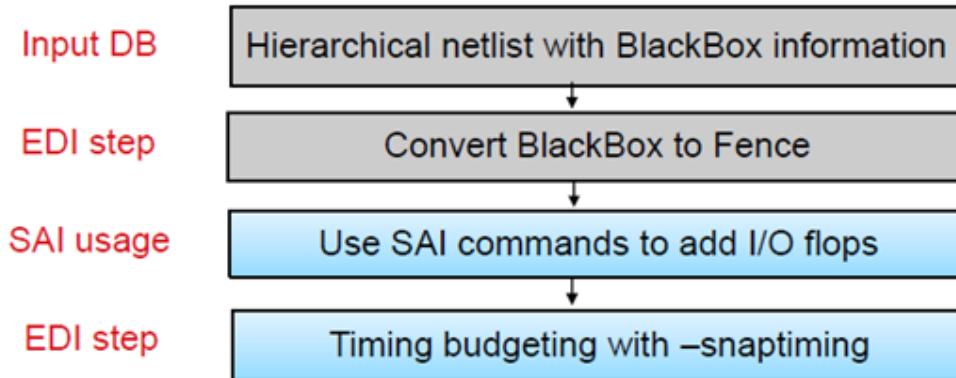
### Possible Application of SAI/FlexModel Flows

SAI can be used along with FlexModel in different phases of prototyping. The following diagram shows the possible application of SAI with FlexModel.



## SAI based Black-Box Time Budgeting Flow

SAI features is helpful in scenarios where you implement a time budgeting flow from a hierarchical netlist consisting of BlackBox.



## Flex Model

For more information, refer to the [FlexModel](#) section.

## Supporting Giga-Scale Designs in Planning stage

As the design size grows to 10s or 100s of millions of instances, serious capacity and run time limitations start occurring in various parts of the Hierarchical Planning stages. Addressing these limitations need advanced modeling techniques and capabilities in the flow.

FlexModels provide a very light-weight abstraction for the partitions that can help reduce the design netlist up to 95%, leading to higher capacity and faster runtimes through the Planning stages.

Besides using FlexModel technology, Active-logic Reduction Technology can be used to reduce timing graph.

## Active-logic Reduction Technology

Active-logic Reduction Technology (ART) is a technique that is used to activate a certain portion of logic in a design and masking the other logic, while maintaining full physical design database in memory. In ART, an active logic view contains only the active portion of the logic.

ART can be applied to any timing-related command, such as timing budgeting or timing optimization to reduce run time and memory usage. In timing operations, an active logic view contains only the set of timing paths exposed to the specific operation. When applied to timing optimization, active logic views enable cross-hierarchical optimization while preserving the full hierarchical view of the design after optimization is complete.

## Top-level Timing Closure

Innovus provides strong block modeling capabilities for efficient closure of top-level design.

## Using Interface Logic Models (ILM)

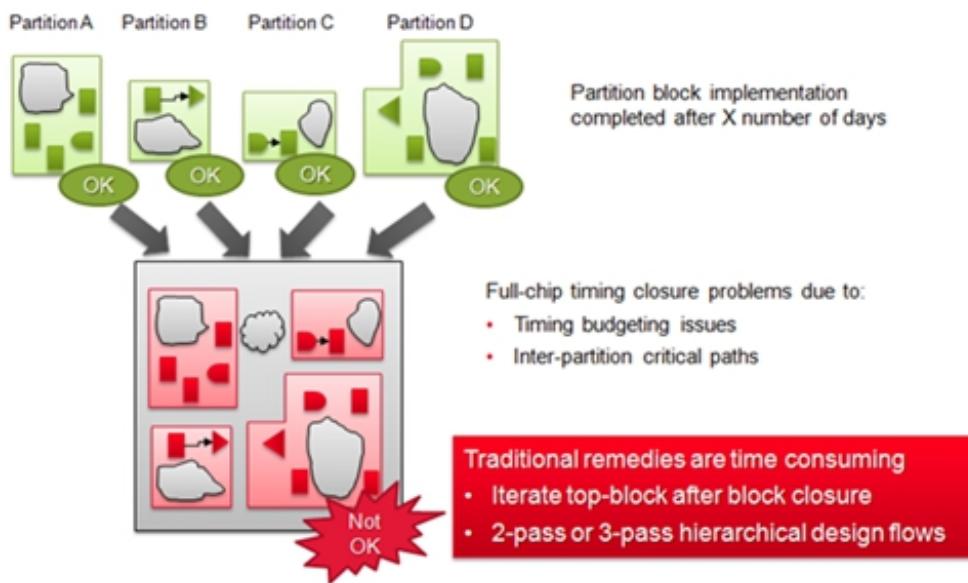
An Interface Logic Model (ILM) is a structural representation of a block, specifically a subset of the block's structure including instances along the I/O timing paths, clock-tree instances, and instances or net coupling affecting the signal integrity (SI) on I/O timing paths. It is a compact and accurate representation of timing characteristics of a block. Instead of using a blackbox at the top level, you create an ILM at the block level and use it as you would use a blackbox.

The advantages of using ILMs are as follows:

- More accurate analysis than a blackbox flow
- More SI aware than combined .lib or .cdb approach
- Can model clock generator inside block

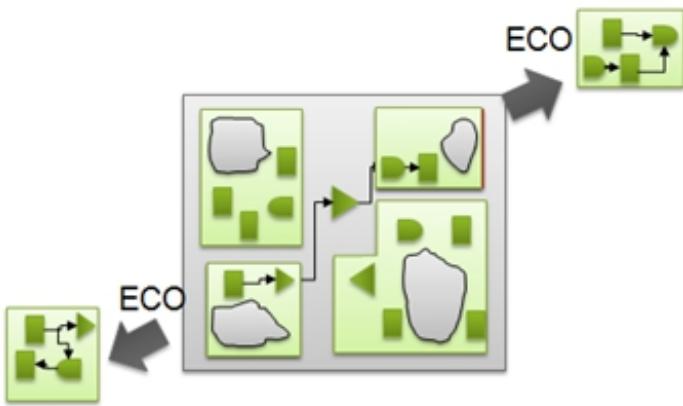
- More accurate timing and SI reduces the number of design iterations to close timing and SI
- No need to characterize blocks
- Works on actual design data
- Can be used in the initial prototyping stage for very big designs, when loading full design data is not feasible
- Allows you to modify only top-level data
- Fully preserves implemented partitions
- Uses the original constraint file for top-level analysis
- No abstraction for timing exceptions

While ILMs serve well for accurately modeling Partition timing characteristics to drive faster design convergence, they cannot be modified to close timing at the top-level, if needed, to avoid iterations between block and top-level optimizations. This is an important challenge of Hierarchical implementation.



## Using Flexible Interface Logic Models (FlexILM)

Designers may need to do at least two or three passes of hierarchical flow to close timing. To address this challenge, a single-pass hierarchical solution with Flexible Interface Logic Models (FlexILM) can be used. FlexILM is a reduced netlist where logic on interface paths are kept and logic on internal paths are removed. FlexILM also reduces memory in timing graph and physical data where removed instances are replaced by placement blockages to avoid violations with new optimized logics. Additionally, routing of removed nets is replaced by RC grids to improve RC extracted correlation. At the top-level design, interface paths of FlexILMs can be optimized, and netlist and placement changes can be ECO back to partition blocks automatically.



For more information, see the "[Top-level Timing Closure Methodologies](#)" chapter of the Innovus User Guide.

## Chip Assembly

Chip assembly is the last stage in the top-down process and consists of bringing together the detailed information for the top-level and all of the blocks for full chip extraction, power, timing, and crosstalk analysis. Chip assembly is done using the `assembleDesign` command.

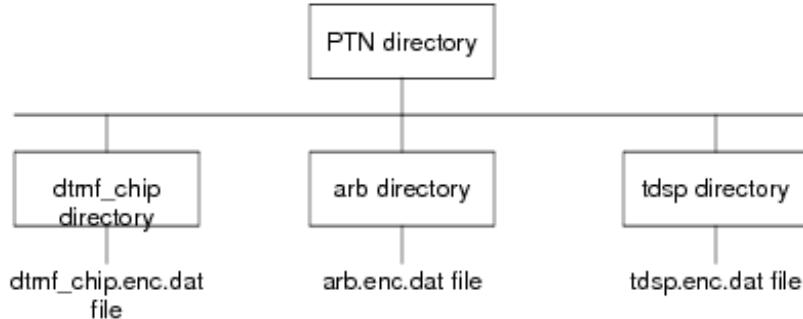
Note: Before using the `assembleDesign` command, for each design, save the top-level and the block-level designs using the `saveDesign -def` command.

As an example, consider a design called dtmf that has two partitions: arb and tdsp. After running the `partition` command, the partition directories are saved under the PTN directory. You would, therefore, implement the following:

- top-level design dtmf\_chip
- arb block

- **tdsp block**

The design files are arb.enc.dat and tdsp.enc.dat for the arb and tdsp blocks, respectively. The following figure shows the directory structure:



**Note:** Chip assembly now supports using “defMerge” capability as well.

With this feature being available in 14.2 the user can now use the direct block DEF transform-and-merge approach during assembleDesign.

Advantages with the use of defMerge option are:

- Faster than read\_partition assembly
- No partition LEF files are required
- Lesser peak memory requirement

Example: `assembleDesign -blockDir b1 -blockDir b2 ... -defMerge`

You can now perform chip assembly using the `assembleDesign` command. This command does the following:

- Concatenates the Verilog netlist files from the partitions back to the top level  
Note: The partition netlists and top-level netlist are changed from the time the save partition step was performed.
- Merges the design data with the original top design level. By default, data from DEF files is used. However, you can use the -fe parameter to specify that Innovus data should be used. You can also use data in the OpenAccess database format.
- Brings back the row information if the -row parameter is specified.
- Preserves scan chain information at partition block-level design, therefore minimizing the floorplan data loss during partition and assemble design cycle. The start and stop scan chain

points at partition block I/O pins are adjusted back to instances that connect to scan chain points. Top-level scan chains are not connected to block-level scan chains.

Run this command from the directory that contains the full chip-level floorplan for the top-down hierarchical flow.

For details on chip assembly see the [Partitioning the Design - Chip Assembly](#) section in the *Innovus User Guide*.

For details of the syntax and the parameters, see the description of the `assembleDesign` command in the *Innovus Text Command Reference*.



# Infrastructure Related Capabilities

---

- Getting Started
- Customizing the User Interface
- Accelerating the Design Process By Using Multiple-CPU Processing
- Data Preparation
- Importing and Exporting Designs

# Getting Started

- Product and Installation Information
- Setting the Run-Time Environment
- Temporary File Locations
- OpenAccess
- Launching the Console
- Tab Completing Command Names, Parameter Names, Global Variable Names and Enum Values
- Command-Line Editing
- Setting Preferences
- Starting the Software
- Interrupting the Software
- Using the Log File Viewer
- Accessing Documentation and Help

## Product and Installation Information

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

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

For information about Innovus™ Implementation System licenses, see "About Innovus Licenses" in the [Product and Licensing Information](#) chapter.

## Setting the Run-Time Environment

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

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

- Use the following command to add the man pages for the Innovus Tcl commands to your MANPATH. This enables you to bring up the man pages for a command from the UNIX prompt without launching the tool.

```
setenv MANPATH <existing_man_path>:<install_dir>/share/innovus/man
```

Here, <install\_dir>/share/innovus/man is the directory in the Innovus installation tree where the `man` pages reside.

Example:

```
setenv MANPATH
```

```
$ {MANPATH} :/icd/flow/INNOVUS/INNOVUS151/latest.main.lnx86/lnx86/share/innovus/man
```

## Supported and Compatible Platforms

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

## 64-Bit Version of Innovus Applications

Innovus software only has a 64-bit mode. A 32-bit version of the software is no longer supported.

## Temporary File Locations

Each Innovus session creates its own temporary directory to store temporary files at the beginning of the run.

By default the `tmp_dir` is created in `/tmp`. If the Unix envar `TMPDIR` is set, then the `tmp_dir` is created inside `$TMPDIR`.

The name of the `tmp_dir` will look like:

```
innovus_temp_[pid]_[hostname]_[user]_xxxxxx
```

Where the `_xxxxxx` is a string added to make the directory unique. For example:

```
innovus_temp_10233_farm254_bob_nfp9ez
```

The temporary directory is automatically removed on exit or if the run terminated with a catchable signal (e.g. SIGSEGV).

## OpenAccess

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

The various OpenAccess Unix utilities, such as `def2oa`, `oa2def`, `verilog2oa`, `oaGetVersion`, and so on are all linked into the `<Cadence_install_dir>/bin` directory.

For more information on the version of OpenAccess supported with this release, see the OpenAccess installation directory or use `oaGetVersion`.

## Launching the Console

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

`innovus>`

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

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

## Tab Completing Command Names, Parameter Names, Global Variable Names and Enum Values

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

After you type a partial text command name and press the `Tab` key, the software displays the exact command name that completes or matches the text you typed (if the string is unique to one text command) or all the commands that match the text you typed. For example, if you type `setP1` and

press the `Tab` key, the software displays the following commands:

```
innovus 3> setPl<Tab>
setPlaceMode setPlanDesignMode
```

If you type `setPlace<Tab>`, the software completes the command name as follows:

```
innovus 3> setPlaceMode
```

**Note:** This function supports all Tcl commands, and the `man` first argument also supports it (e.g. `man setPlace<Tab>`).

## Tab Completing Parameter Names

The Tab completion capability is also available for parameter names starting with `-` for Innovus commands or for user commands registered with `define_proc_arguments`.

For example,

```
innovus 3> set_verify_drc_mode -check<Tab>
-check_implant -checkImplant_across_rows -check_ndr_spacing -check_only -
check_same_via_cell
```

If you type `set_verify_drc_mode -check_s<Tab>`, the software completes the parameter name as follows:

```
innovus 3> set_verify_drc_mode -check_same_via_cell
```

To view all parameters of a command, type the command name followed by `-` and press `Tab`. For example:

```
innovus 3> verify_drc -<Tab>
-area -check_implant_across_rows -check_ndr_spacing -check_only -check_same_via_cell -
exclude_pg_net -layer_range
-limit -report
```

## Tab Completing Global Variable Name

The Tab completion capability is also available for Tcl variable names. The `set` and `man` commands understand them for their first argument, and other commands match names following a `$`.

For example, if you type `set lefDefOut<Tab>` (or `man lefDefOut<Tab>`) the software completes the name:

```
innovus 5> set lefDefOutVersion
```

Other commands use a leading \$ to look for Tcl variable names, like this:

```
innovus 6> puts $delaycal_default<Tab>  
delaycal_default_net_delay delaycal_default_net_load  
delaycal_default_net_load_ignore_for_ilm
```

## Tab Completing Enum Type Values for Parameters

Tab completion can be used to view or complete enum values.

For example, if you type `verify_drc -check_only` followed by a space and press the Tab key, the software displays the following:

```
innovus 5> verify_drc -check_only <Tab>  
all regular special
```

If you type `verify_drc -check_only r<Tab>`, the software completes the value as follows:

```
innovus 5> verify_drc -check_only regular
```

## Tab Completing Unix file and directory names

If you type a <Tab> in any other context, the Tcl shell will look for Unix file and directory names that match the string.

For example,

```
innovus 5> defIn test<Tab>  
test.tcl test.def test.lef
```

## Command-Line Editing

Innovus provides a multiline editing interface. This means, you can move the cursor to any position and edit any character of a multiline command before execution. For example, in the following command, if your cursor is at '.' located at the beginning of the second line and you press the Left arrow key, the cursor will go to '\' at the end of the previous line.

```
innovus 5> dbGet top\  
+ .name
```

Several hotkeys are provided for command-line editing, similar to `emacs` hotkeys. Using these hotkeys, you can quickly move the cursor within and between the lines of a command before execution. Hotkeys can be independent, control characters, or escape sequences. A control character is typed by holding down the Control (`Ctrl`) key when typing the character. Escape sequences are used by pressing the Escape (`Esc`) key before pressing the other key(s) in the sequence.

## Notes

- You can type an editing command anywhere on the line, not just at the beginning. You can press `Enter` anywhere on the line, not just at the end.
- Editing commands are case sensitive.

| Independent keys |                                                                                                  | Result |
|------------------|--------------------------------------------------------------------------------------------------|--------|
| Home             | Goes to the start of the current line. If already there, goes to the start of the previous line. |        |
| Down             | In a multiline command, moves to the next line.                                                  |        |
| End              | Goes to the end of the current line. If already there, goes to the end of the next line.         |        |
| Tab              | Completes the command.                                                                           |        |
| Up               | In a multiline command, moves to the previous line.                                              |        |

| Control characters  |                                                                        | Result |
|---------------------|------------------------------------------------------------------------|--------|
| <code>Ctrl+a</code> | Goes to the beginning of the line.                                     |        |
| <code>Ctrl+b</code> | Moves the cursor left by one character.                                |        |
| <code>Ctrl+c</code> | Exits from editing mode, returning the console to normal Innovus mode. |        |

|        |                                                                                                                                                                                                      |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ctrl+d | Deletes the next character if the cursor is in the middle of a line. Lists the files in the current directory beginning with the word just before the cursor, if the cursor is at the end of a line. |
| Ctrl+e | Goes to the end of the current line.                                                                                                                                                                 |
| Ctrl+f | Move the cursor one character to the right.                                                                                                                                                          |
| Ctrl+h | Deletes one character before the cursor.                                                                                                                                                             |
| Ctrl+i | Completes filename or displays all possible options in the given context.                                                                                                                            |
| Ctrl+j | Submits the line; Same as <code>Enter</code> .                                                                                                                                                       |
| Ctrl+k | Deletes characters from the cursor to the end of the line.                                                                                                                                           |
| Ctrl+l | Clears the screen and redisplays the last line.                                                                                                                                                      |
| Ctrl+m | Same as <code>Ctrl+j</code> .                                                                                                                                                                        |
| Ctrl+n | Goes to the next line in history; Same as <code>Ctrl+Down</code> .                                                                                                                                   |
| Ctrl+o | Accepts the line, moves the history pointer to the next position.                                                                                                                                    |
| Ctrl+p | Goes to the previous line in history; Same as <code>Ctrl+Up</code> .                                                                                                                                 |
| Ctrl+r | Searches backward through history for text; must start line if text begins with an Up arrow.                                                                                                         |
| Ctrl+t | Transposes characters, that is exchanges the character before the cursor with the character at the cursor,<br>and then moves the cursor one character right.                                         |
| Ctrl+u | Deletes the line.                                                                                                                                                                                    |
| Ctrl+w | Deletes the characters between the cursor and the marked position set by <code>Esc+space</code> .                                                                                                    |
| Ctrl+x | Moves the cursor to the marked position set by <code>Esc+space</code> .                                                                                                                              |
| Ctrl+y | Pastes yanked string before the cursor.                                                                                                                                                              |

| Ctrl+z           | Suspends the tool (System hotkey)                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------|
| Ctrl+]           | Moves to the next character; Equals to the next input character.                                               |
| Ctrl+Down        | Goes to the next line in history; Same as Ctrl+n.                                                              |
| Ctrl+Up          | Goes to the previous line in history; Same as Ctrl+p.                                                          |
| Ctrl+?           | Deletes the character before cursor.                                                                           |
| Escape sequences | Result                                                                                                         |
| Esc+Ctrl+h       | Deletes the previous word.                                                                                     |
| Esc+Delete       | Deletes the previous word.                                                                                     |
| Esc+space        | Marks a position.                                                                                              |
| Esc+.            | Inserts the last argument of the last command before the cursor.                                               |
| Esc+<            | Displays the first command in history.                                                                         |
| Esc+>            | Displays the last command in history.                                                                          |
| Esc+?            | Displays all possible file names.                                                                              |
| Esc+b            | Moves the cursor to the beginning of the word to the left.                                                     |
| Esc+d            | Deletes the word to the right of the cursor.                                                                   |
| Esc+f            | Moves cursor to the beginning of the next word.                                                                |
| Esc+l            | Changes the characters from the cursor to the end of the word to lowercase.                                    |
| Esc+u            | Change the characters from the cursor to the end of the word to uppercase.                                     |
| Esc+y            | Pastes the yanked string before the cursor.                                                                    |
| Esc+w            | Saves the strings between the marked position (set by Esc+space) and the cursor position into the yank buffer. |
| Esc+p            | Starts a backward search in history.                                                                           |
| Esc+Up           | Moves the cursor up to the previous line.                                                                      |

|           |                                         |
|-----------|-----------------------------------------|
| Esc+Down  | Moves the cursor down to the next line. |
| Esc+Left  | Same as Esc+b.                          |
| Esc+Right | Same as Esc+f.                          |

#### Notes

- The `Ctrl+[` and `Ctrl+v` key sequences are not currently supported. These will be implemented in a subsequent release.

## Setting Preferences

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

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

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

For information on setting design preferences, see *Set Preference* in the [View Menu](#) chapter of the *Innovus Menu Reference*.

## Initialization Files

Innovus uses the following initialization files for setting preferences:

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .encrc       | <p>Used for setting Tcl parameters or adding user-defined Tcl commands. If different versions of this file exist in the home, or working directories, the file in the working directory takes precedence.</p> <p><b>Note:</b> Usage of this file is no longer recommended, but is allowed for backward compatibility. Use <code>enc.tcl</code> instead. This file is processed before the GUI is created, so it cannot be used to customize the GUI.</p>                                                                             |
| enc.tcl      | <p>Used for setting Tcl parameters, customizing the GUI, or adding user-defined Tcl commands or global variables. If different versions of this file exist in the installation, home, or working directories, the file in the working directory takes precedence.</p> <p><b>Note:</b> The software does not create or modify this file. You must create the file and then put a copy of the file in the installation directory (<code>&lt;installation_path&gt;/tools/innovus/etc</code>), home directory, or working directory.</p> |
| enc.pref.tcl | <p>Contains design preferences set using the Design, Display, Floorplan, and Selection tabs in the Preferences form in the GUI (see Set Preference in the <a href="#">View Menu</a> chapter of the <i>Innovus Menu Reference</i>).</p> <p><b>Note:</b> By default, Innovus saves changes that you make to your preferences to the <code>enc.pref.tcl</code> file in the working directory.</p>                                                                                                                                       |
| .enc         | <p>Contains design preferences set using the Windows tab in the Preferences form in the GUI (see Set Preference in the View Menu chapter of the <i>Innovus Menu Reference</i>).</p>                                                                                                                                                                                                                                                                                                                                                  |
| cds.lib      | <p>Contains library path information. The <code>cds.lib</code> entries are available from Tcl in <code>cds_lib vars</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                      |

The initialization files are read in the following sequence:

1. `.encrc` in the home directory
2. `.encrc` in the working directory
3. `enc.pref.tcl` in the working directory
4. `.enc` in the home directory
5. `enc.tcl` in the installation/etc directory

6. `enc.tcl` in the home directory
7. `enc.tcl` in the working directory

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

## Starting the Software

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

For a detailed description of the `innovus` command, see `innovus` in the *Innovus Text Command Reference*.

The `innovus` command starts one of the following products:

- Innovus™ Implementation System
- Virtuoso® Digital Implementation
- Virtuoso® Digital Implementation XL
- First Encounter® L
- First Encounter® XL

For an overview of the products and product licensing, see [Product and Licensing Information](#). For a detailed description of the `innovus` command, see `innovus` in the *Innovus Text Command Reference*.

## Interrupting the Software

Like most Unix programs, an Innovus session is interrupted by the interrupt signal (SIGINT). You can send this signal to the Innovus process by using the `Ctrl+C` key combination.

## Interrupt Behavior When Tool Is Idle

If you press `Ctrl+C` while the tool is idle, the following message is printed:

```
**INFO (INTERRUPT): One more Ctrl-C to exit Innovus ...
```

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

## Interrupt Behavior in Interactive Mode

When you press `Ctrl+C` during an interactive Innovus process, an Interrupt menu is displayed. All threads other than the main thread (that is, the thread that is handling the display of the menu) are immediately suspended until the menu action is resolved. The Interrupt menu displayed is as follows:

```
**INFO (INTERRUPT): The interrupted design can be viewed in its current state but  
should not be used to continue the flow.
```

- 1) Ignore and continue.
- 2) Quit.
- 3) Finish current command but interrupt Tcl script.
- 4) Interrupt current command and Tcl script and return to prompt.
- 5) Suspend current command and return to prompt. Use command 'resume' to continue.

Type a number 1-5 and press ENTER:

Options 1 and 2 are always available. Option 3 is available if the interrupt happens while a command is running. Options 4 and 5 are displayed only if the command that is currently running is registered to support interruption.

**Note:** If you press `Ctrl+C` while running a short-duration command, the Interrupt menu may not be visible. This is because the command's runtime may be shorter than the time taken to press `Ctrl+C`.

The Interrupt menu makes debugging easier for long-running commands as it can help you trace the underlying causes of problems. Some examples of long-running commands where the Interrupt menu can be useful are:

- `routeDesign`
- `globalDetailRoute`

- optDesign
- verifyGeometry
- verifyConnectivity
- verifyPowerVia
- verifyMetalDensity
- verifyProcessAntenna
- verifyACLimit

## Interrupting the Execution of Batch Files

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

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

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

## Suspending the Execution of a Script

If you want to debug your script, you can use the `suspend` command to suspend your script and return to the Innovus prompt. You can then type any command required for debugging. Whenever you want to resume your script, just type `resume` at the Innovus prompt.

## Stopping the Software

Use one of the following methods to stop the software:

- In the main Innovus window, select *File - Exit*.
- On the text command line, type the following command:  
`exit`

## Using the Log File Viewer

Innovus provides the following methods to view the log file:

- Integrated Log File Viewer

## Integrated Log File Viewer

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

- Ability to expand and collapse command information.
- Ability to view multiple log files in separate console windows simultaneously.
- Color coding of error, warning, and information messages.
- String matching through the *Edit - Find>Select Object* menu.  
For more information, see "*Find>Select Object*" in the [Edit Menu](#) chapter of the *Innovus Menu Reference*.
- Access to the documentation in the *Innovus Text Command Reference*.  
When a log file is displayed, click on any of the underlined commands to open an HTML window that displays the documentation for that command.

Use one of the following methods to use the viewer:

- Select Tools - Log Viewer on the main menu.  
The Log File window is displayed. Select the log file to view. The software opens a separate console window and displays the log file. For more information, see "Log Viewer" in the [Tools Menu](#) chapter of the *Innovus Menu Reference*.
- On the text command line, type the following command in the console window where the software is running:

```
viewLog [-file logFileName]
```

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

## Accessing Documentation and Help

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

- [Launching Cadence Help From the Command Prompt](#)
- [Accessing Documentation and Help from the GUI](#)
- [Using the man and help Commands on the Command Line](#)
- [Using the Integrated Log File Viewer](#)

### Launching Cadence Help From the Command Prompt

You can type the Unix command `cdnshelp` (which is inside the `<install_dir>/bin` directory) to launch the Cadence Help tool. It includes access to all the documents in the installation, along with Search functions.

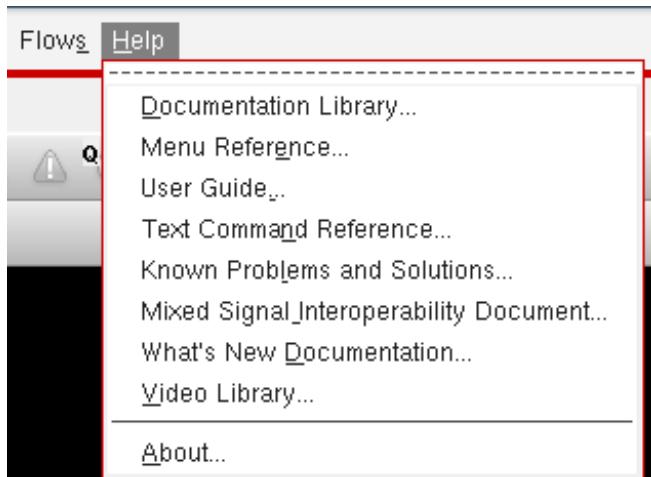
After launching Cadence® Help, press `F1` or choose *Help - Contents* to display the help page for Cadence Help.

### Accessing Documentation and Help from the GUI

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

- [Select Help from the Main Menu](#)
- [Select Help Button on a Form](#)

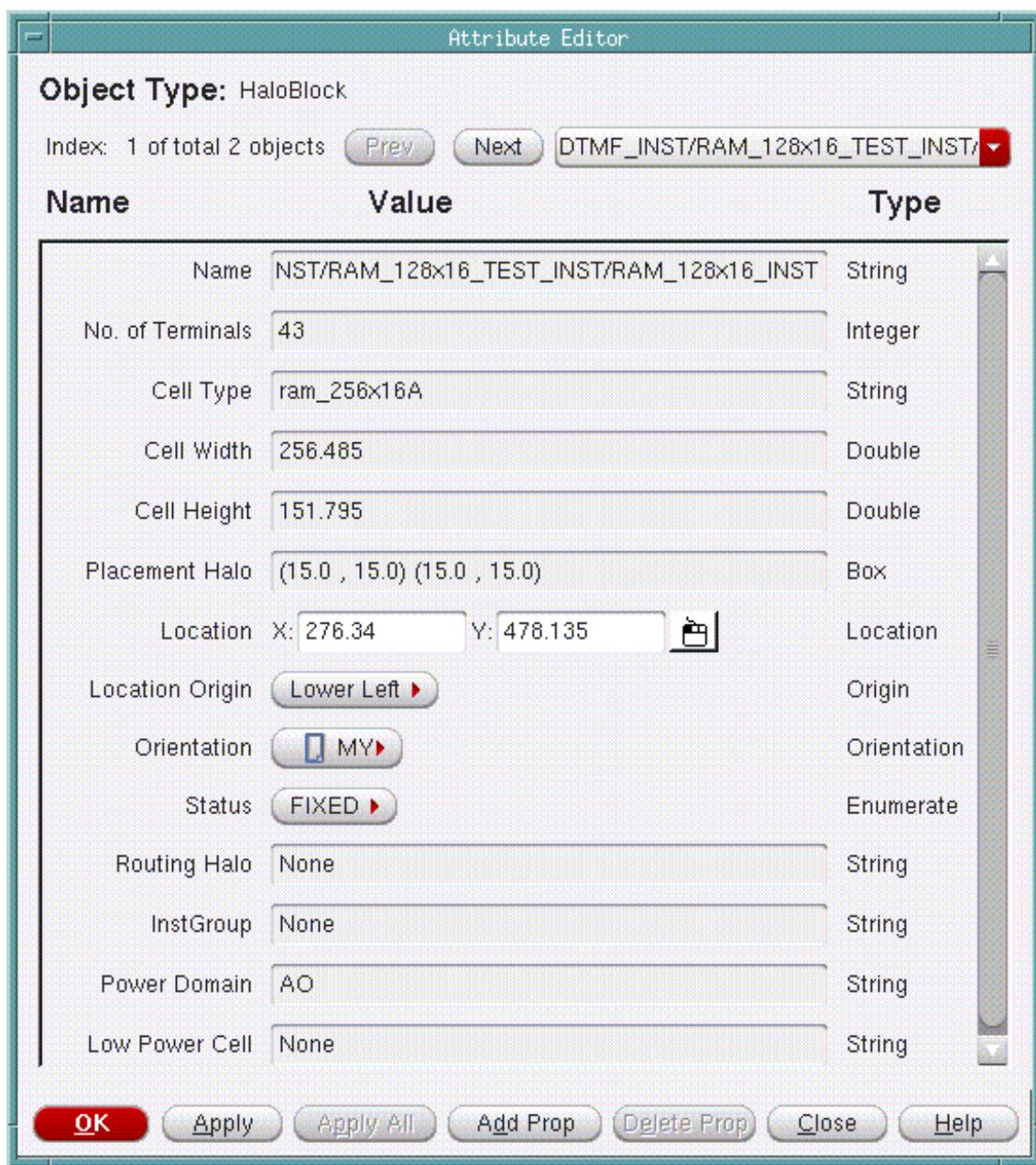
## Select Help from the Main Menu



- Select *Help*, and then any of the following options:
  - *Documentation Library*  
Opens the Cadence Help window, which provides access to all the documentation shipped with the release.
  - *Menu Reference*  
Opens the Table of Contents page of the menu reference.
  - *User Guide*  
Opens the Table of Contents page of the user guide.
  - *Text Command Reference*  
Opens the Table of Contents page of the text command reference.
  - *Known Problems and Solutions*  
Opens the Table of Contents page of the known problems and solutions document.
  - *Mixed Signal Interoperability Document*  
Opens the Table of Contents page of the mixed-signal interoperability document.
  - *What's New Documentation*  
Opens the Table of Contents page of the what's new document.

## Select Help Button on a Form

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



Clicking the *Help* button opens the *Innovus Menu Reference* entry for the form in the Cadence Help window.

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

## Using the help Command to View the Command Syntax

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

```
help command_name
```

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

```
help getAllLayers
```

The software displays the following text:

```
Usage: getAllLayers [-help] [<type>]  
-help # Prints out the command usage  
<type> # <Type of layer> (string, optional)
```

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

```
help
```

## Using the man Command to View the Command Description

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

```
man command_name
```

For example, to see the complete information for the `unsetMessageLimit` command, type the following command:

```
man unsetMessageLimit
```

The software displays the following text:

**Name**

```
unsetMessageLimit
```

**Synopsis**

```
unsetMessageLimit [-help] [prefix [ID1, ID2, ...]]
```

### Description

Removes the default or user-specified limit on error and warning messages.  
All errors and warnings are output to log.

### Parameters

**-help** Outputs a brief description that includes type and default information for each unsetMessageLimit parameter.  
For a detailed description of the command and all of its parameters, use the man command: man unsetMessageLimit.

#### **prefix [ID1, ID2...]**

Removes the default or user-specified limit on error and warning messages with prefix.

### Example

The following command removes the limit on the number of error or warning messages to display:  
unsetMessageLimit  
(END)

## Using the help Command to View Message Summary

- To see the message summary of a particular message ID, type the following command in the software console:

```
help msg_id
```

For example, to see the message summary for the TAMODEL-302 message ID, type the following command:

```
help TAMODEL-302
```

The software displays the following text:

Data signal arrives at clock pin '%s'. This data/clock conflict may be due to missing or incomplete clock definitions. Trigger arcs and check arcs associated with '%s' are being removed to prevent data signal from propagating to clock paths.

## Using the man Command to View Message Detail

- Some error messages have extended help to provide more detailed information or solution. To see the message detail of a particular message ID, type the following command at the software console:
- To see the message detail of a particular message ID, type the following command at the software console:

```
man msg_id
```

For example, to see the message summary for the TAMODEL-302 message ID, type the following command:

```
man TAMODEL-302
```

The software displays the following text:

NAME

TAMODEL-302 (warning)

SYNOPSIS

Data signal arrives at clock pin '%s'. This data/clock conflict may be due to missing or incomplete clock definitions. Trigger arcs and check arcs associated with '%s' are being removed to prevent data signal from propagating to clock paths.

#### DESCRIPTION

Usually data signals arrive at clock pins of sequential elements because clock source is not defined properly. Please trace clock sources backward from the clock pins of sequential elements to make sure that clock waveforms are associated with clock sources. This can be done by using `create_clock` or `create_generated_clock` command.

-  The detailed description is not available for all active message IDs.

## Using the Integrated Log File Viewer

You can also access the command documentation by using the integrated log file viewer. The command to start the viewer is `viewLog`. For more information, see "Integrated Log File Viewer" section of this chapter or `viewLog` in the "General Commands" chapter of the *Innovus Text Command Reference*.

# Customizing the User Interface

- [Overview](#)
- [Creating a New Menu](#)
- [Modifying an Existing Menu](#)
  - [Adding a Menu Element to an Existing Menu](#)
  - [Replacing an Existing Menu Element](#)
- [Adding a New Toolbar and Toolbutton](#)
  - [Supported Image Formats for Icons](#)
- [Querying and Configuring Interface Elements](#)
  - [Iterating, Querying, and Configuring a Menu](#)
  - [Updating the Message on the Status Bar](#)
  - [Setting the Main Window's Size and Title](#)

## Overview

Innovus™ Implementation System provides a GUI development kit comprising five APIs that let you customize the menus, toolbars, status bar, main window, and other interface elements. The kit comprises the following five APIs:

- [uiAdd](#)
- [uiDelete](#)
- [uiSet](#)
- [uiGet](#)
- [uiFind](#)

For more information on these commands, see the [GUI Commands](#) chapter of the *Innovus Text Command Reference*.

Using the commands in the GUI development kit, you can:

- Add a new menu to the main window menu bar. This includes adding a submenu, menu commands, separators, checks and radio buttons. For more information, see [Creating a New Menu](#).
- Modify an existing menu. For more information, see [Modifying an Existing Menu](#).

- Add a new toolbar and toolbutton. For more information, see [Adding a New Toolbar and Toolbutton](#).
- Query and configure interface elements, including menus, status bar, and the main window. For more information, see [Querying and Configuring Interface Elements](#).

This chapter provides a suite of simple examples with annotated comments to familiarize you with the development kit and shorten the learning curve.

## Creating a New Menu

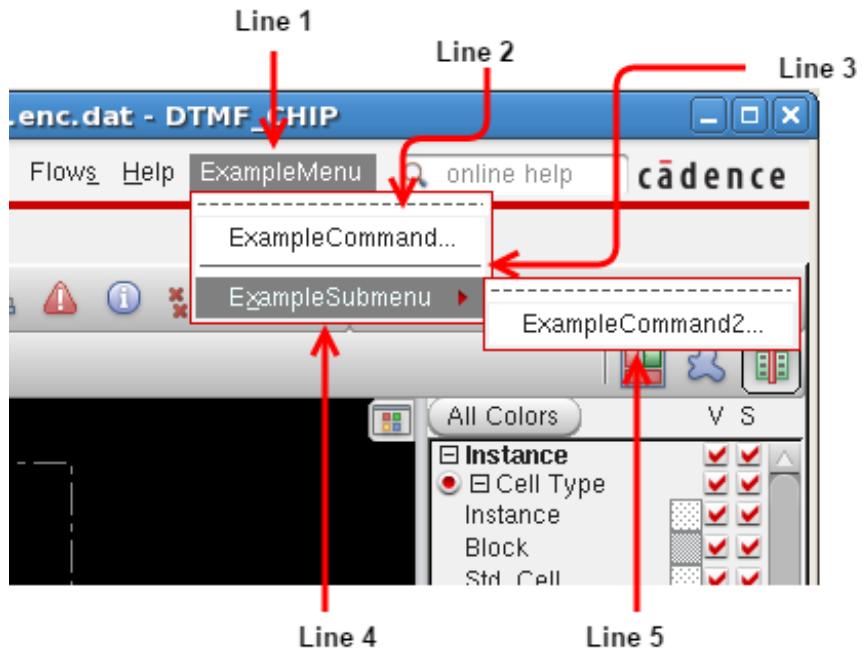
Using the `uiAdd` command, you can create a new menu and add it to the main window menu bar. You can then add menu elements, such as command, submenu, separator, radio button and check box, to the new menu using the same `uiAdd` command.

The following script adds a new menu, labeled *ExampleMenu*, to the main window menu bar:

```
uiAdd expMenu -type menu -label ExampleMenu -in main  
  
uiAdd expCommand -type command -label "ExampleCommand..." -command [list puts "Example  
Command"] -in expMenu  
  
uiAdd expSep -type separator -in expMenu  
  
uiAdd exp_submenu -type submenu -label "ExampleSubmenu" -underline 1 -in expMenu  
uiAdd expCommand2 -type command -label "ExampleCommand2..." -command [list puts  
"Example Command"] -in exp_submenu
```

By default, the new *ExampleMenu* is appended to the end of the menu bar. By specifying the `-before` option in Line 1 of the script, you can insert the new menu before a specified menu.

Lines 2 to 5 of the script add three types of elements to the menu, including command, separator and submenu.



Similarly, you can add items of type `radio` and `check` using the `uiAdd` command.

For more information on the syntax and parameter of the `uiAdd` command, see the "GUI Commands" chapter of the *Innovus Text Command Reference*.

## Modifying an Existing Menu

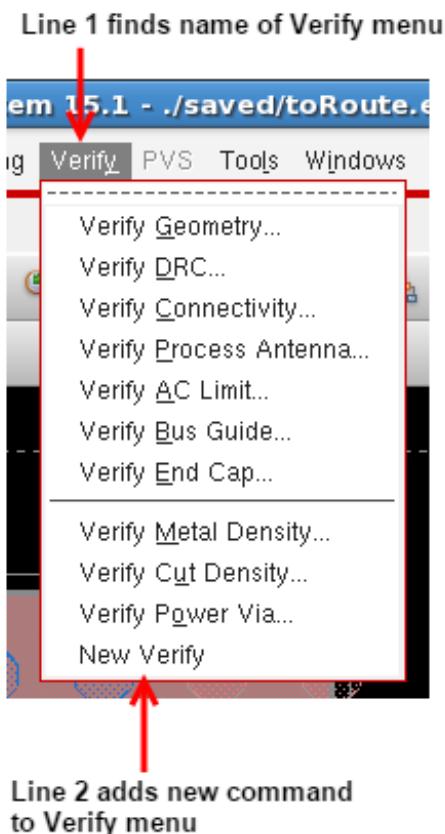
You can also use the `uiAdd` command to add or replace menu elements in an existing menu.

## Adding a Menu Element to an Existing Menu

The following script adds a new command to the existing *Verify* menu:

```
set vMenu [uiFind main -type menu -label "Verify"]  
  
uiAdd newVerify -type command -label "New Verify" -command [list puts "New Verify"] -in  
$vMenu
```

Line 1 of the script retrieves the *name* of the *Verify* menu and assigns it temporarily to the variable *vMenu*. Line 2 adds a new command labeled *New Verify* to *vMenu*, which represents the *Verify* menu.



## Replacing an Existing Menu Element

The following script finds an existing menu element and replaces it with a new one:

```
set toolMenu [uiFind -type menu -label "Tools"]

set oldMenu [uiFind $toolMenu -type command -label "Design Browser..."]

set before [uiGet $oldMenu -before]

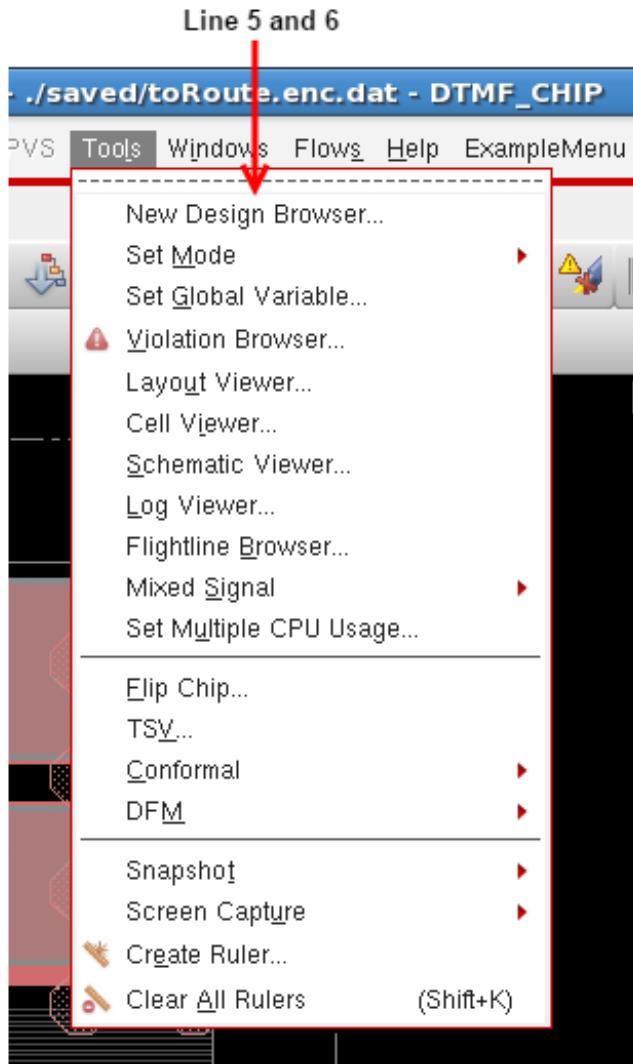
uiDelete $oldMenu

set newMenu ${oldMenu}_new

uiAdd $newMenu -type command -label "New Design Browser..." -before $before -command
"puts {New Design Browser}" -in $toolMenu
```

In this script:

- Line 1 finds the `name` of the *Tools* menu.
- Line 2 finds an existing command, *Design Browser*, in the *Tools* menu by its label.
- Line 3 finds its neighbor using the `uiGet` command.
- Line 4 deletes the *Design Browser* command using the `uiDelete` command.
- Line 5 and 6 create a new menu labeled *New Design Browser* in the same location.



## Adding a New Toolbar and Toolbutton

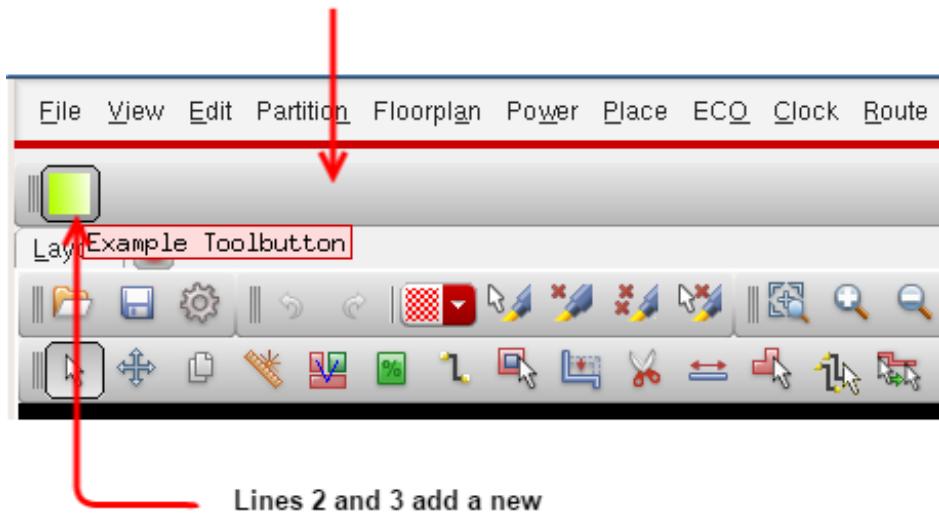
Using the `uiAdd` command, you can add a new toolbar and toolbuttons as shown in the following script:

```
uiAdd expToolbar -type toolbar -in main -label "Example Toolbar" -newline true
set ICON_DIR "./"
```

```
uiAdd expToolbar -type toolbar -in expToolbar -label "Example Toolbar" -  
tooltip "Example Toolbutton" -icon [file join $ICON_DIR example.png]
```

Line 1 adds a new toolbar in the main window. As the `-newline` option is set to `true`, the toolbar is added as a new row. Lines 2 and 3 add a new toolbutton, which uses a .png file as its icon.

Line 1 adds a toolbar in a new row



## Supported Image Formats for Icons

The following image formats are supported for icon files:

**Table 3-1**

| Format    | Description                           |
|-----------|---------------------------------------|
| BMP       | Windows Bitmap                        |
| GIF       | Graphic Interchange Format (optional) |
| JPG, JPEG | Joint Photographic Experts Group      |

|     |                         |
|-----|-------------------------|
| PNG | Portable Networks Group |
| XBM | X11 Bitmap              |
| XPM | X11 Pixmap              |

# Querying and Configuring Interface Elements

Using the `uiGet`, `uiFind`, and `uiSet` commands in the GUI development kit, you can query and configure various interface elements, including menus, status bar, and the main window.

## Iterating, Querying, and Configuring a Menu

The following script finds and sets the *File* menu's state.

```
set menus [uiGet main -menu]

foreach menu $menus {

    if {[uiGet $menu -label] == "File"} {

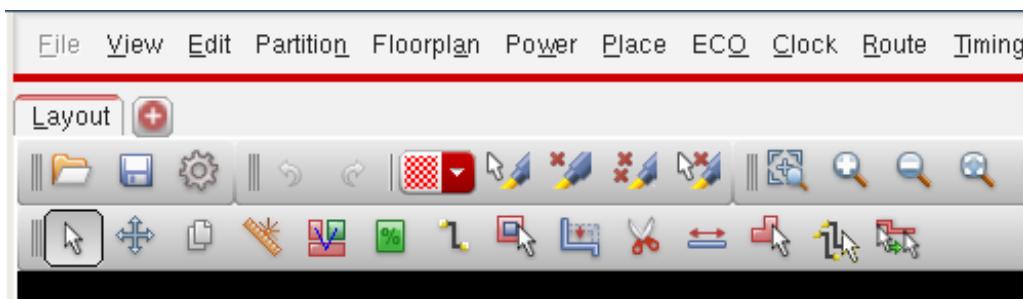
        uiSet $menu -disabled true

    }

}

}
```

This script iterates all the menus in the main window to find the *File* menu. It disables the *File* menu with the `uiSet` command.



The same thing can also be done using the script below:

```
set menu [uiFind main -type menu -label "File"]  
uiSet $menu -disabled true
```

## Updating the Message on the Status Bar

With the help of the `uiGet` and `uiSet` commands, you can also update the message displayed on the status bar of the main window as shown in the following script:

```
set e_statusbar [uiGet main -statusbar]  
uiSet $e_statusbar -message "Example Message"
```

This script first finds the status bar name with the `uiGet` command. It then sets its message using the `uiSet` command.



## Setting the Main Window's Size and Title

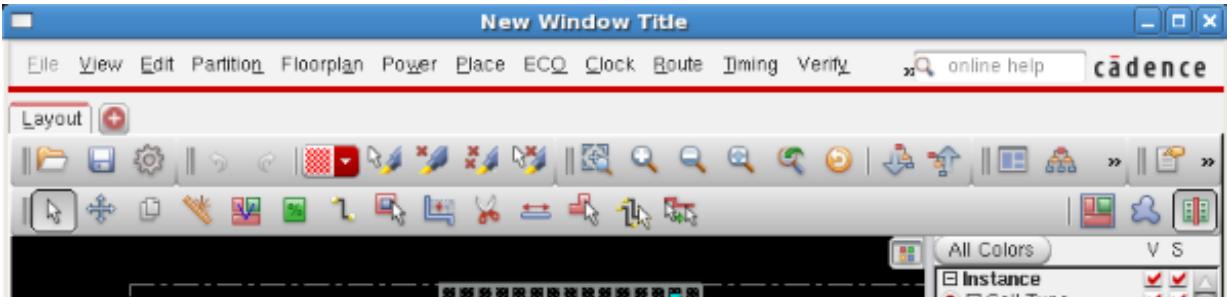
You can use the `uiSet` command to set the size of the main window as desired. For instance, you can set the main window size to 800x600 as follows:

```
uiSet main -geometry 800x600
```

In addition, `uiSet` can be used to set the main window's coordinates and title as in the following script:

```
uiSet main -geometry 780x686+232+0
```

```
uiSet main -title "New Window Title"
```



Line 1 of the script sets main window size to `780x686` and its coordinates to `232,0`. Line 2 sets the main window's title to `New Window Title`.

# Accelerating the Design Process By Using Multiple-CPU Processing

- [Overview](#)
- [Running Distributed Processing](#)
- [Running Multi-Threading](#)
- [Running Superthreading](#)
- [Memory and Run Time Control](#)
- [Checking the Distributed Computing Environment](#)
- [Setting and Changing the License Check-Out Order](#)
- [Limiting the Multi-CPU License Search to Specific Products](#)
- [Releasing Licenses Before the Session Ends](#)
- [Controlling the Level of Usage Information in the Log File](#)
- [Where to Find More Information on Multi-CPU Licensing](#)

## Overview

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

- **Multi-threading**  
In this mode, a job is divided into several threads, and multiple processors in a single machine process them concurrently.
- **Distributed processing**  
In this mode, a job is processed by two or more networked computers running concurrently.
- **Superthreading**  
In this mode, a job runs in the distributed processing mode but each distributed job can also run threads, that is, one or more networked computers, each with multiple processors, work concurrently to complete a job.

You configure multiple-CPU processing by using the commands described in the [Multiple-CPU Processing Commands](#) chapter of the *Innovus System Text Command Reference* or the "Multiple

CPU Processing" form in the [Options Menu](#).

The following table shows the Innovus System features that support multiple-CPU processing:

Table 4-1

### Innovus System features that support multiple-CPU processing

| Feature                               | Command                                                                                                                                                    | Limitations/Notes                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Capacitance table generation          | <a href="#">generateCapTbl</a>                                                                                                                             | For more information, see <a href="#">Capacitance Table Generation Flow</a> in the "RC Extraction" chapter.                                                                                                                                                                                                                                |
| Global placement                      | <a href="#">placeDesign</a><br><a href="#">addFiller</a>                                                                                                   | <ul style="list-style-type: none"> <li>Supported in default mode only (-modulePlan is true)</li> </ul> For more information, see <a href="#">Running Placement in Multi-CPU Mode</a> in the "Placing the Design" chapter.                                                                                                                  |
| Optimization                          | <a href="#">optDesign {-preCTS<br/> <br/>-postCTS   -<br/>postRoute}</a>                                                                                   | For more information, see <a href="#">Distributed Timing Analysis for Hold Fixing</a> in the "Optimizing Timing" chapter.                                                                                                                                                                                                                  |
| Clock concurrent Optimization (CCOPT) | <a href="#">ccopt_design</a><br><a href="#">ccopt_design -cts</a>                                                                                          |                                                                                                                                                                                                                                                                                                                                            |
| Automatic floorplan synthesis         | <a href="#">multiPlanDesign</a>                                                                                                                            | For more information, see <a href="#">Creating Multiple Alternative Floorplans</a> in the "Creating an Initial Floorplan Using Masterplan" chapter.                                                                                                                                                                                        |
| Metal fill                            | <a href="#">addMetalFill</a>                                                                                                                               | For more information, see <a href="#">Adding Metal Fill in Multiple-CPU Processing Mode</a> in the "Optimizing Metal Density" chapter.                                                                                                                                                                                                     |
| NanoRoute router                      | <a href="#">globalRoute</a><br><a href="#">globalDetailRoute</a><br><a href="#">detailRoute</a><br><a href="#">routeDesign</a><br><a href="#">ecoRoute</a> | <ul style="list-style-type: none"> <li>Superthreading is supported for detailed routing only.</li> <li>Superthreading options take precedence over multi-threading options.</li> </ul> For more information, see <a href="#">Accelerating Routing with Multi-Threading and Superthreading</a> in the "Using the NanoRoute Router" chapter. |

|                                           |                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tQuantus, IQRC, and Standalone extraction | <code>setExtractRCMode</code><br><code>extractRC</code>                                                                                                                      | For more information, see <a href="#">Distributed Processing</a> in the "RC Extraction" chapter.                                                                                                                                                                                                                                                   |
| Signal integrity analysis                 | <code>optDesign -postRoute -si</code><br><br><code>timeDesign -si</code><br><br><ul style="list-style-type: none"> <li>● In MMMC mode</li> <li>● In non-MMMC mode</li> </ul> | <ul style="list-style-type: none"> <li>● For backward compatibility, distributed processing options take precedence.</li> <li>● Superthreading options take precedence over multi-threading options.</li> </ul> <p>For more information, see <a href="#">Multi-CPU Processing Settings</a> in the "Analyzing and Repairing Crosstalk" chapter.</p> |
| Verify connectivity                       | <code>verifyConnectivity</code>                                                                                                                                              | For more information, see <a href="#">Verifying Connectivity</a> in the "Identifying and Viewing Violations" chapter.                                                                                                                                                                                                                              |
| Verify geometry                           | <code>verifyGeometry</code>                                                                                                                                                  | For more information, see <a href="#">Verifying Geometry in Multi-Thread Mode</a> in the "Identifying and Viewing Violations" chapter.                                                                                                                                                                                                             |
| Verify metal density                      | <code>verifyMetalDensity</code>                                                                                                                                              | For more information, see <a href="#">Verifying Metal Density in Multi-Thread Mode</a> in the "Identifying and Viewing Violations" chapter.                                                                                                                                                                                                        |
| Delay calculation                         | All commands that require timing data and invoke a full delay calculation.                                                                                                   | For more information, see <a href="#">Calculating Delay in Multi-Thread Mode</a> in the "Calculating Delay" chapter.                                                                                                                                                                                                                               |
| Timing Budgeting                          | <code>deriveTimingBudget</code><br><code>saveTimingBudget</code>                                                                                                             | For more information, see <a href="#">Support for Distributed Processing in Budgeting</a> in the "Timing Budgeting" chapter.                                                                                                                                                                                                                       |
| Power Planning                            | <code>addStripe</code>                                                                                                                                                       | For more information, see <a href="#">Planning Power</a> in the "Low Power Design" chapter.                                                                                                                                                                                                                                                        |

## Running Distributed Processing

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

- [setDistributeHost](#)

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

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

- [setMultiCpuUsage](#)

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

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

For example, to run the `multiPlanDesign` command in distributed processing mode on four machines within an existing LSF environment, and each machine has 4 GB of memory, specify the following commands:

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

## Running Multi-Threading

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

- [setMultiCpuUsage](#)

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

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

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

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

```
setMultiCpuUsage -localCpu 4  
placeDesign
```

## Running Superthreading

To run the Innovus software in super threading mode, the following two commands are required:

- [setDistributeHost](#)
- [setMultiCpuUsage](#)

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

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

```
setDistributeHost -lsf -queue myQueue -resource "mem>4000 OS=RH4"  
setMultiCpuUsage -remoteHost 2 -cpuPerRemoteHost 3  
detailRoute
```

## Memory and Run Time Control

Use the [report\\_resource](#) command to report memory/run time in multiple-CPU processing. This command allows you to determine how much memory is being used at any time and of what form (physical vs. virtual), and to determine real time and CPU time. You can use the [-verbose](#) parameter of the [report\\_resource](#) command to get detailed memory usage information.

When you run [report\\_resource -verbose](#), the following detailed memory information is displayed:

|                                                                                  |
|----------------------------------------------------------------------------------|
| Current (total cpu=0:00:12.9, real=0:05:48, peak res=275.8M, current mem=383.9M) |
| Cpu(s) 2, load average: 4.63                                                     |
| Mem: 16443800k total, 16378412k used, 65388k free, 105704k buffers               |
| Swap: 16777208k total, 17460k used, 16759748k free, 12528212k cached             |
| Memory Detailed Usage:                                                           |

|                | Data Resident Set (DRS) | Private Dirty (DRT) | Virtual Size (VIRT) | Resident Size (RES) |
|----------------|-------------------------|---------------------|---------------------|---------------------|
| Total current: | 383.9M                  | 275.8M              | 854.1M              | 358.9M              |
| peak:          | 383.9M                  | 275.8M              | 854.1M              | 358.9M              |

- Cpu (s) is the number of available processors in the machine.
- Load average is the system load averages for the past 1 minute.
- Mem and Swap are the current memory information of the machine.

The value of `MEM` in the LSF report corresponds to the value of `RES` in the `report_resource` report, and the value of `SWAP` in the LSF report corresponds to the value of `VIRT` in the `report_resource` report.

- Data Resident Set (DRS) is the amount of physical memory devoted to other than executable code. "current mem" shows this value (Total current DRS).
- Private Dirty (DRT) is the memory which must be written to disk before the corresponding physical memory location can be used for some other virtual page. "peak res" shows this value (Total peak DRT). This is the minimum number that you must reserve to run the program.
- Virtual Size (VIRT) is the total amount of virtual memory used by the task. It includes the swapped and non-swapped memory.
- Resident Size (RES) is the non-swapped physical memory a task has used. The number of "Total Peak RES" is the recommended physical memory to reserve.

The `-verbose` parameter also works in conjunction with the `-peak` and `-start/-end` parameters of the `report_resource` command. When you run the local distributed host (`setDistributeHost -local`) command, the memory information will include the memory consumed by master and clients. Otherwise, the master and client details are not displayed.

The following command script specifies to display detailed memory information during the `optDesign -postRoute` command:

```
report_resource -start opt_postroute
setDistributeHost -local
setMultiCpuUsage -localCpu 8
optDesign -postRoute
```

```
report_resource -end opt_postroute -verbose
```

 For `-start/-end` parameters, use `-verbose` with the `-end` parameter only.

The following message is displayed:

```
Ending "opt_postroute" (total cpu=0:57:18, real=0:33:24, peak res=6493.1M, current mem=5305.0M)
```

Memory Detailed Usage:

|                 | Data Resident Set (DRS) | Private Dirty (DRT) | Virtual Size (VIRT) | Resident Size (RES) |
|-----------------|-------------------------|---------------------|---------------------|---------------------|
| Total current:  | 5305.0M                 | 4012.1M             | 5919.2M             | 4255.4M             |
| peak:           | 10712.8M                | 6493.1M             | 15090.3M            | 7312.5M             |
| Master current: | 5305.0M                 | 4012.1M             | 5919.2M             | 4255.4M             |
| peak:           | 5565.5M                 | 4055.1M             | 6064.5M             | 4298.4M             |
| Task peak:      | 748.8M                  | 368.7M              | 1219.4M             | 456.4M              |

Task `peak` reports peak value of each item from all clients, therefore, it is possible that eight values come from eight different clients.

## Checking the Distributed Computing Environment

To check if distributed processing can work in the software environment, use the `checkMultiCpuUsage` command. This command checks if the specified CPUs can be accessed.

## Setting and Changing the License Check-Out Order

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

```
setMultiCpuUsage -licenseList {vdi eds1 edsxl fexl }
```

For information on the default check-out order, see [Innovus System Packaging and Licensing](#).

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

Each base license allows a set of specific licenses to be used for multi-CPU processing. This list can be obtained from the `getMultiCpuUsage` command after invoking the software.

```
[DEV] innovus 1> getMultiCpuUsage  
  
Total CPU(s) Enabled: 2  
Current License(s): 1 Encounter_Digital_Impl_Sys_XL  
keepLicense: true  
licenseList: enccpu eds1 edsxl
```

This license list can be customized from among the available choices by using the `setMultiCpuUsage -licenseList` command.

## Releasing Licenses Before the Session Ends

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

- Before running any multi-CPU applications, specify the following command to keep the acquired multiple CPU-licenses until the current session ends:

```
setMultiCpuUsage -keepLicense true
```

To display the current setting for `setMultiCpuUsage -keepLicense`, use the `getMultiCpuUsage -keepLicense` command.

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

```
setMultiCpuUsage -releaseLicense
```

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

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

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

By default, the software does not write starting and ending information for threads or timing details

to the log file, but you can change this behavior by specifying 1 or 2 for the `-threadInfo` parameter.

- Specify 1 to write the final message to the log file.
- Specify 2 to write additional starting/ending information for each thread.

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

See [Innovus System Packaging and Licensing](#) for more information on multi-CPU licenses.

## Data Preparation

- Generating a Technology File
- Preparing Physical Libraries
- Unsupported LEF and DEF Syntax
- Generating the I/O Assignment File
- Preparing Timing Libraries
- Encrypting Libraries
- Preparing Timing Constraints
- Preparing Capacitance Tables
- Preparing Data for Delay Calculation
- Preparing Data for Crosstalk Analysis
- Checking Designs
- Preparing Data in the Timing Closure Design Flow
- Converting iPRT Format to LEF

## Generating a Technology File

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

## Creating Technology Information Using LEF

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

## Creating Technology Information Using OpenAccess

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

## Preparing Physical Libraries

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

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

## Using LEF to Create Physical Libraries

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

- Use the Abstract Generator.  
For more information, see the *Cadence Abstract Generator User Guide*.
- Create LEF MACROS manually.  
For more information, see the [LEF/DEF Language Reference](#) .

## Creating OpenAccess Physical Libraries

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

## Unsupported LEF and DEF Syntax

The software supports most of the syntax statements in the 5.7 versions of LEF and DEF with the exception of the ones listed below.

### Unsupported LEF 5.7 Syntax

The software parses but ignores the following LEF 5.7 syntax:

| LEF Statement   | Unsupported Syntax                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Layer (Routing) | [DIAGWIDTH <i>diagWidth</i> ;]<br>[DIAGSPACING <i>diagSpacing</i> ;]<br>[DIAGMINEDGELENGTH <i>diagLength</i> ;]<br>[SLOTWIREWIDTH <i>minWidth</i> ;]<br>[SLOTWIRELENGTH <i>minLength</i> ;]<br>[SLOTWIDTH <i>minWidth</i> ;]<br>[SLOTLLENGTH <i>minLength</i> ;]<br>[MAXADJACENTSLOTSPACING <i>spacing</i> ;]<br>[MAXCOAXIALSLOTSPACING <i>spacing</i> ;]<br>[MAXEDGESLOTSPACING <i>spacing</i> ;]<br>[SPLITWIREWIDTH <i>minWidth</i> ;]<br>[HEIGHT <i>distance</i> ;]<br>[SHRINKAGE <i>distance</i> ;]<br>[CAPMULTIPLIER <i>value</i> ;] |
| Macro Pin       | [TAPERRULE <i>ruleName</i> ;]<br>[NETEXPR " <i>netExprPropName defaultNetName</i> " ;]                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

|                   |                                                                                        |
|-------------------|----------------------------------------------------------------------------------------|
| Nondefault Rule   | [DIAGWIDTH <i>diagWidth</i> ;]<br>[HARDSPACING ;]<br>[USEVIARULE <i>viaRuleName</i> ;] |
| Via Rule Generate | [DEFAULT]                                                                              |

The following LEF 5.7 syntax causes an error message in the Innovus software:

| <b>LEF Statement</b> | <b>Unsupported Syntax</b>      |
|----------------------|--------------------------------|
| Layer (Routing)      | DIRECTION {DIAG45   DIAG135} ; |

## Unsupported DEF 5.7 Syntax

The Innovus software parses but ignores the following DEF 5.7 syntax:

| <b>DEF Statement</b> | <b>Unsupported Syntax</b>                     |
|----------------------|-----------------------------------------------|
| Blockages            | [+ SLOTS]                                     |
| Groups               | [+ PROPERTY { <i>propName propName</i> } ...] |
| Extensions           | All BEGINEXT syntax                           |
| History              | All HISTORY syntax                            |

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nets                 | <ul style="list-style-type: none"> <li>[+ SYNTHESIZED]</li> <li>[+ VPIN <i>vpinName</i> [LAYER <i>layerName</i>] <i>pt pt</i><br/> [PLACED <i>pt orient</i>   FIXED <i>pt orient</i>   COVER <i>pt orient</i> ]]</li> <li>[+ SUBNET <i>subNetName</i><br/> [ ( {compName <i>pinName</i>   PIN <i>pinName</i>   VPIN <i>vpinName</i>} ) ]<br/> [NONDEFALTRULE <i>ruleName</i> ]]</li> <li><b>Note:</b> SUBNET NONDEFALTRULE is ignored; routing uses rule for NET.</li> <li>[+ USE {RESET   SCAN   TIEOFF}]</li> <li><b>Note:</b> Supports ANALOG, CLOCK, GROUND, POWER, and SIGNAL.</li> <li>[+ PATTERN {STEINER   WIREDLOGIC}]</li> <li>[+ ESTCAP <i>wireCapacitance</i>]</li> <li>[+ SOURCE {DIST   NETLIST   TEST   USER}]</li> </ul> |
| Pins                 | <ul style="list-style-type: none"> <li>[+ USE {TIEOFF   SCAN   RESET}]</li> <li><b>Note:</b> Supports SIGNAL, POWER, GROUND, ANALOG, and CLOCK.</li> <li>[+ DIRECTION FEEDTHRU]</li> <li>[+ NETEXPR " <i>netExprPropertyName defaultNetName</i> "]</li> <li>[+ SUPPLYSENSITIVITY <i>powerPinName</i>]</li> <li>[+ GROUNDSENSITIVITY <i>groundPinName</i>]</li> </ul>                                                                                                                                                                                                                                                                                                                                                                     |
| Pin Properties       | All PINPROPERTIES syntax                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Property Definitions | The object types: GROUP, REGION, and ROW                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Regions              | [+ PROPERTY { <i>propName propVal</i> } ...]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Rows                 | [+ PROPERTY { <i>propName propVal</i> } ...]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Slots                | All SLOTS syntax                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Special Nets</b> | <pre>[+ SYNTHESIZED] [+ VOLTAGE <i>volts</i>] [+ SOURCE {DIST   NETLIST   USER}] [+ USE {RESET   SCAN   TIEOFF}]  <b>Note:</b> Supports ANALOG, CLOCK, GROUND, POWER, and SIGNAL.  [+ PATTERN {STEINER   WIREDLOGIC}] [+ ESTCAP <i>wireCapacitance</i>] [+ WEIGHT <i>weight</i>]  <b>Note:</b> + WEIGHT only supported in NETS section.  Special Wiring Statement:  [+ STYLE <i>styleNum</i>]  <b>Note:</b> If included in the DEF file, the software displays an error message stating that only the default style is supported, ignores the specified style, and replaces it with the default one.</pre> |
| <b>Styles</b>       | All STYLES syntax                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

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

| DEF Statement                      | Unsupported Syntax                                 |
|------------------------------------|----------------------------------------------------|
| Nets<br>(Regular Wiring Statement) | <pre>[<i>orient</i>] [STYLE <i>styleNum</i>]</pre> |

## Generating the I/O Assignment File

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

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

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

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

This section discusses the following topics:

- [Creating an I/O Assignment File](#)
- [Creating a Rule-Based I/O Assignment File](#)
- [I/O Pad and Pin Assignment Examples](#)
- [Performing Area I/O Placement](#)

## Creating an I/O Assignment File

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

```
( globals
    version = 3
    space = 0
    io_order = default
)
```

```
( row_margin
  ( top
    ( io_row ring_number = 1 margin = 0)
    ( io_row ring_number = 2 margin = 630)
    ( io_row ring_number = 3 margin = 830)

  )
  ( bottom
    ( io_row ring_number = 1 margin = 0)
    ( io_row ring_number = 4 margin = 0)
    ( io_row ring_number = 5 margin = 0)

  )
  ( left
    ( io_row ring_number = 1 margin = 0)
    ( io_row ring_number = 6 margin = 200)
    ( io_row ring_number = 7 margin = 0)

  )
  ( right
    ( io_row ring_number = 1 margin = 0)
    ( io_row ring_number = 8 margin = 200)

  )
)

( iopad
  ( topleft
    (locals ring_number = 1)
    ( inst name="ins_0")
  )
)
```

```
( left  
  ( locals ring_number = 6)  
  
( inst name="ins_5"    offset = 5896.2 )  
  
( inst name="ins_7"    space = 5)  
  
( inst name="ins_9"    place_status = placed)  
  
( inst name="ins_10"   orientation = R0)  
  
( inst name="ins_11"   )  
  
( inst name="ins_12"   )  
  
( inst name="ins_14"   space = 0)  
  
  ( locals ring_number = 7)  
  
( inst name="ins_6"    offset = 5826.4 )  
  
( inst name="ins_8"   )  
  
( inst name="ins_13"   )  
  
( inst name="ins_15"   )  
  
( inst name="ins_17"   )  
  
( inst name="ins_19"   )  
  
( inst name="ins_21"   )  
  
( inst name="ins_23"   )  
  
( inst name="ins_25"   )  
  
  
( bottomleft  
  (locals ring_number = 1)  
  ( inst name="ins_1")  
 )  
  
( bottom
```

```
( locals ring_number = 4      )

( inst name="ins_167" offset = 3946.8)
( inst name="ins_168" )
( inst name="ins_169"  space = 5)
( inst name="ins_170" )
( inst name="ins_171" )
( inst name="ins_172" space = 0)
( inst name="ins_173" )
( inst name="ins_174" )
( inst name="ins_175" )
( inst name="ins_176" )

( locals ring_number = 5)

( inst name="ins_261"    offset = 11812.3 )
( inst name="ins_262" )
( inst name="ins_263" )
( inst name="ins_264" )
( inst name="ins_265" )
( inst name="ins_266" )
( inst name="ins_267" )

( bottomright

(locals ring_number = 1)

( inst name="ins_2" orientation=R0 )
)

( right

( locals ring_number = 8      )

( inst name="ins_313"    offset = 200 )
( inst name="ins_315" )
```

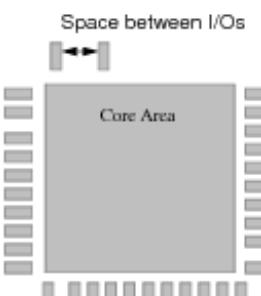
```
( inst name="ins_316" )
( inst name="ins_318" )
( inst name="ins_320" )
( inst name="ins_322" )
( inst name="ins_324" )
( inst name="ins_326" )
( inst name="ins_328" )
( inst name="ins_330" )
( inst name="ins_332" )
( inst name="ins_334" )
( inst name="ins_336" )
( inst name="ins_337" )
( inst name="ins_338" )
( inst name="ins_339" )
( inst name="ins_341" )
( inst name="ins_343" )
( inst name="ins_344" )
( topright
    (locals ring_number = 1)
( inst name="ins_3")
)
( top
    ( locals ring_number = 2      )
( inst name="ins_610"      offset = 100 )

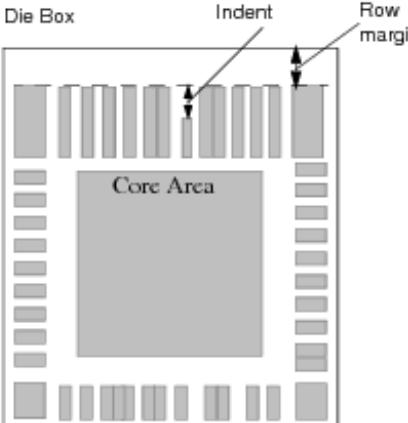
    ( locals ring_number = 3)
( inst name="ins_611"      offset = 200 )
( inst name="ins_612" )
```

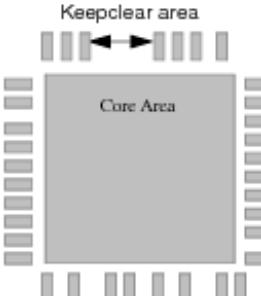
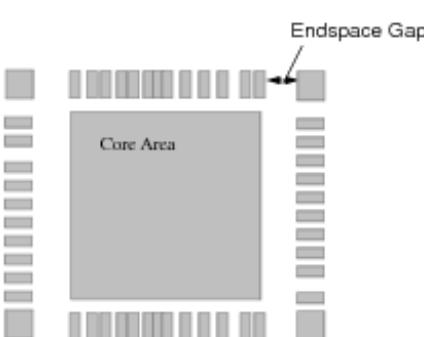
```
( inst name="ins_614" )  
( inst name="ins_616" )  
( inst name="ins_617" )  
( inst name="ins_619" )  
)  
)
```

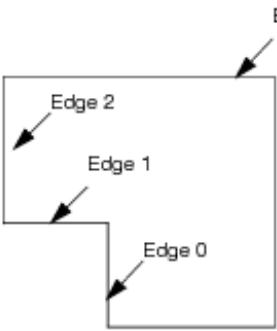
The following entries are included in the template:

| globals                  |                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>version = 3</code> | Specifies the beginning of a new I/O format.                                                                                                                                                                                                                                                                                                                              |
| <code>io_order</code>    | <p>Specifies the order of the I/O pads and pins. This can be:</p> <ul style="list-style-type: none"> <li>● clockwise</li> <li>● counterclockwise</li> <li>● default</li> </ul> <p><b>Note:</b> The default I/O order for a vertical edge is from the bottom to the top, and for a horizontal edge, it is from the left to the right.</p>                                  |
| <code>total_edge</code>  | <p>Specifies the number of edges for the rectilinear block design. The edges are numbered starting from 0. For example, if the <code>total_edge</code> is 4, then the edges are numbered as edge 0, edge 1, edge 2, and edge 3.</p> <p><b>Note:</b> You must verify that the total number of edges that you specify matches with the value in the destination design.</p> |
| <code>space</code>       | Specifies the global I/O pin spacing, in $\mu$ meters.                                                                                                                                                                                                                                                                                                                    |
| iopad locals             |                                                                                                                                                                                                                                                                                                                                                                           |
| <code>space</code>       | <p>Specifies the local I/O pad spacing, in <math>\mu</math>meters.</p> <p><b>Note:</b> This space setting is honored by the first cell on one edge, when <code>xy</code> or <code>offset</code> is not specified.</p>                                                                                                                                                     |
| <code>ring_number</code> | Specifies the ring number in which the I/O pad is placed.                                                                                                                                                                                                                                                                                                                 |
| <code>row_name</code>    | Specifies the I/O <code>row</code> name.                                                                                                                                                                                                                                                                                                                                  |
| iopad instance           |                                                                                                                                                                                                                                                                                                                                                                           |
| <code>name</code>        | Specifies the name of the I/O instance.                                                                                                                                                                                                                                                                                                                                   |

|                                                                                                                                                                                                                                                                                                                                           |               |                                                                                                                                                                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                           | <i>x, y</i>   | <p>Specifies the absolute <i>x, y</i> location of the I/O pad instance, starting from the lower left corner.</p> <p><b>Note:</b> Specifying <i>x, y</i> location for sides and edges of I/O pads is not supported in the I/O file.</p>                                     |
|                                                                                                                                                                                                                                                                                                                                           | <i>skip</i>   | <p>Specifies the distance, in micrometers, of the I/O pad from the previously defined I/O pad.</p> <p>The value that you specify here is valid only for this cell.</p>                                                                                                     |
|                                                                                                                                                                                                                                                                                                                                           | <i>space</i>  | <p>Specifies the spacing, in micrometers, between the pad being defined and the previously defined pad.</p> <p>The value that you specify here, overrides the global space setting.</p>  |
|                                                                                                                                                                                                                                                                                                                                           | <i>offset</i> | <p>Specifies the distance in microns from the IO ring edge to the pad edge based on <i>io_order</i> constraint.</p> <p>The value that you specify here is valid only for this cell.</p>                                                                                    |
| <p><b>Note:</b> For one I/O pad, you can specify only one of the following parameters:</p> <ul style="list-style-type: none"> <li>● <i>skip</i></li> <li>● <i>space</i></li> <li>● <i>offset</i></li> </ul> <p>If you specify all the three parameters, only the last parameter that you define, is considered for I/O pad placement.</p> |               |                                                                                                                                                                                                                                                                            |

|  |                                                                                                                                                                                                                                                                                                                          |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <p><i>indent</i></p> <p>Specifies the offset, in <math>\mu</math>meters, from the row margin.</p>  <p>However, for designs with single I/O ring, row margin is 0. Hence, indent is the offset of the I/O pad from the die boundary.</p> |
|  | <p><i>orientation</i></p> <p>Specifies the orientation of the I/O.</p>                                                                                                                                                                                                                                                   |
|  | <p><i>place_status</i></p> <p>Specifies the placement status of the I/O pad instance. This can be:</p> <ul style="list-style-type: none"> <li>• placed</li> <li>• covered</li> <li>• fixed</li> </ul> <p><i>Default : fixed.</i></p>                                                                                     |

|                                                                                                                                                                                                                                                                                                                                                                                                                              |  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <p><i>keepclear</i></p> <p>Specifies an area on the chip where you cannot place pins or pads. Specify a range between <i>begin</i> and <i>end</i>, in <math>\mu</math>meters, on the chip side in which pins and pads cannot be placed.</p> <p><b>Note:</b> You must define pad cells in the order in which they appear in the design.</p>  |  |
| <p><i>cell</i></p> <p>Specifies the physical I/O cell.</p>                                                                                                                                                                                                                                                                                                                                                                   |  |
| <p><i>endspace gap</i></p> <p>Specifies the space, in <math>\mu</math>meters, between the corner pad and the last I/O pad for the specified side of the design.</p>                                                                                                                                                                       |  |
| <p><i>iopin locals</i></p>                                                                                                                                                                                                                                                                                                                                                                                                   |  |

|              |                     |                                                                                                                                                                                                                                                                           |
|--------------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | <i>side</i>         | Specifies the side of the I/O pin. This can be:<br><ul style="list-style-type: none"> <li>● top   north</li> <li>● left   west</li> <li>● right   east</li> <li>● bottom   south</li> </ul>                                                                               |
|              | <i>edge num = 0</i> | Specifies the edge number of the I/O pin, with <i>edge num = 0</i> starting from the left side of the lowest y coordinate and the left most corner, in the clockwise direction.<br><br> |
|              | <i>space</i>        | Specifies the spacing, in $\mu$ meters, between the previously defined pin and the pin being defined.<br><br>The value that you specify here, sets the global space setting.                                                                                              |
| <b>iopin</b> |                     |                                                                                                                                                                                                                                                                           |
|              | <i>pin name</i>     | Specifies the name of a pin. Specify I/Os as pins for block designs.                                                                                                                                                                                                      |
|              | <i>layer</i>        | Specifies the metal layer on which the pin must be placed.                                                                                                                                                                                                                |
|              | <i>width</i>        | Specifies the width of the pin in $\mu$ meters. It is the length of the edge that is centered at the reference point.                                                                                                                                                     |
|              | <i>depth</i>        | Specifies the length of the pin in $\mu$ meters.                                                                                                                                                                                                                          |
|              | <i>up</i>           | Specifies the details of internal I/O pins.                                                                                                                                                                                                                               |

|  |             |                                                                                                                                                                         |
|--|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <i>x, y</i> | Specifies the absolute <i>x,y</i> location of the internal I/O pin.<br><br><b>Note:</b> The I/O file supports specifying <i>xy</i> location for internal I/O pins only. |
|  | #           | Specifies the incremented I/O pin edge number.                                                                                                                          |

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

| Row Margin |                    |                                                                                                                                                                                                                            |
|------------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | <i>side</i>        | Specifies the side of the I/O row margin. This can be: <ul style="list-style-type: none"><li>● top</li><li>● north</li><li>● left</li><li>● west</li><li>● right</li><li>● east</li><li>● bottom</li><li>● south</li></ul> |
|            | <i>ring_number</i> | Specifies the I/O ring number on which the I/O rows are placed, with ring 1 being the outer most ring.                                                                                                                     |
|            | <i>margin</i>      | Specifies the distance, in microns, from the die boundary edge to the I/O row edge.                                                                                                                                        |

**Note:** You can use the *Edit I/O Ring* form to specify I/O pad rings and row margins for multiple rows. Alternatively, to achieve the same using text commands, you must first use the [setIoRowMargin](#) command to set the distance from the die boundary edge to start of each row and then use the [placePadIO](#) command to place the I/O pads evenly between these rows.

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

Example for margin/offset/space usage:

```
(globals
  version = 3
```

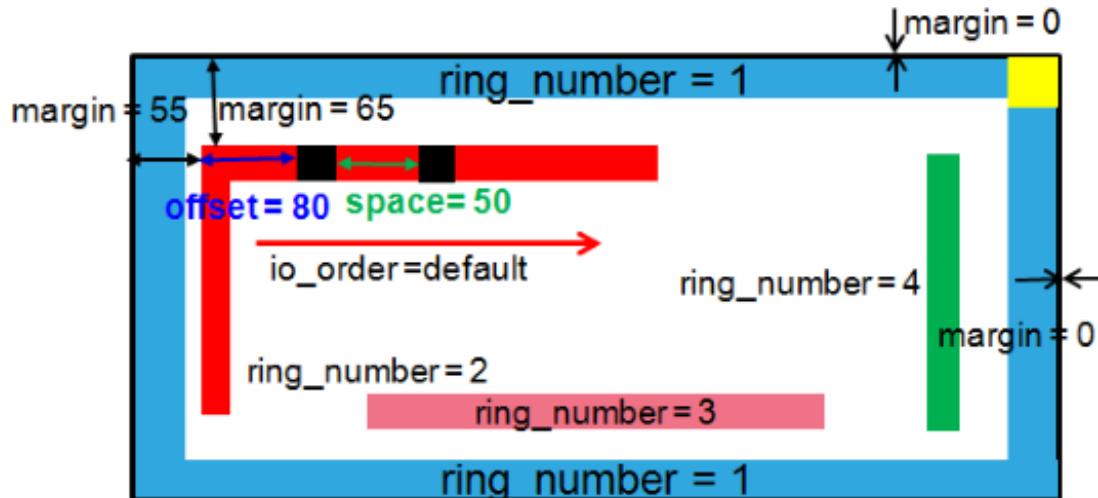
```
    space= <value>
    io_order = default
)

(row_margin
  (top
    (io_row ring_number=1 margin=0.0000)
    (io_row ring_number=2 margin=55.0000)
  )
  (left
    (io_row ring_number=1 margin=0.0000)
    (io_row ring_number=2 margin=65.0000)
  )
  (bottom
    (io_row ring_number=1 margin=0.0000)
    (io_row ring_number=3 margin=75.0000)
  )
  (right
    (io_row ring_number=1 margin=0.0000)
    (io_row ring_number=4 margin=85.0000)
  )
)

(iopad
  (topright
    (locals ring_number=1)
    (inst name="CORNER_TL" orientation=R0)
  )
  (top
    (locals ring_number=2)
    (inst name="ESD_3"      offset=80.0000 orientation=R0)
    (inst name="PAD_1"      space=50 place_status=placed )
  )
  (topleft
  )
  (left
  )

(bottomleft
)
(bottom
)
```

```
(bottomright
)
(right
)
)
```



## Specifying Area I/O Information

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

- **Bump**

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

- To define signal bumps, use the following syntax:

```
bump name="bump_name" cell="bumpcell" x=llx y=llg signal="net_name"
```

For example:

```
bump name="Bump_89_8_8" cell="BUMPCELL" x=855.7200 y=855.7200
signal="port_pad_data_in[1]"
```

- To define power bumps, use the following syntax:

```
bump name="bump_name" cell="bumpcell" x=llx y=llg
signal="net_name" type=power/ground
```

For example:

```
bump name="Bump_90_9_8" cell="BUMPCELL" x=955.7200 y=855.7200 array="array_0"  
signal="VDD" type=power
```

- IOInst

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

```
inst name="inst_name" offset=value place_status=placed/fixed/covered
```

For example:

```
inst name="IOPADS_INST/Pibiasip" offset=35.2800 place_status=placed
```

## Example of an Area I/O file

```
(globals  
version = 3  
io_order = default  
)  
(iopad  
(top  
(inst name="IOPADS_INST/Pibiasip" offset=35.2800 place_status=placed )  
(inst name="IOPADS_INST/Ppllristip" offset=108.8050 place_status=placed )  
)  
(left  
(inst name="IOPADS_INST/Ptdspip03" offset=35.2800 place_status=placed )  
(inst name="IOPADS_INST/Ptdspip04" offset=106.8500 place_status=placed )  
)  
(bottom  
(inst name="IOPADS_INST/Pscanout1op" offset=35.2800 place_status=placed )  
(inst name="IOPADS_INST/Pvcopop" offset=108.8050 place_status=placed )  
)  
(right  
(inst name="IOPADS_INST/Ptdspop04" offset=35.2800 place_status=placed )  
(inst name="IOPADS_INST/Ptdspop05" offset=112.3550 place_status=placed )  
)  
(bumps
```

```
(bump name="Bump_90_9_8" cell="BUMPCELL" x=955.7200 y=855.7200 signal="VDD" type=power )
(bump name="Bump_89_8_8" cell="BUMPCELL" x=855.7200 y=855.7200 signal="port_pad_data_in[1]" )
(bump name="Bump_88_7_8" cell="BUMPCELL" x=755.7200 y=855.7200 signal="scan_en" )
(bump name="Bump_58_7_5" cell="BUMPCELL" x=755.7200 y=555.7200 signal="VSS" type=ground )
)
```

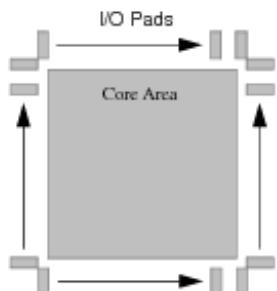
## Creating a Rule-Based I/O Assignment File

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

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

## I/O Pad and Pin Assignment Examples

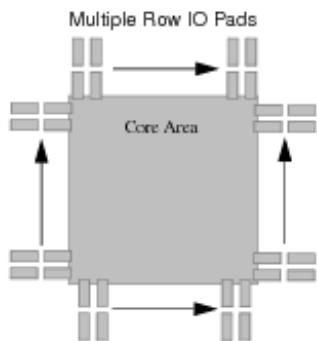
The following example shows statements in a sample I/O assignment file for I/O pads as shown in the figure below:



```
version = 3  
  
io_order = clockwise  
  
total_edge = 4  
  
space = 1.06  
  
(inst  
  
    name = IOPADS_INST/pad1 W  
  
    offset = 235.0000  
  
    orientation = R0  
  
    place_status = fixed  
  
)  
  
(inst  
  
    name = IOPADS_INST/pad2 W  
  
    offset = 296.1250  
  
    orientation = R0  
  
    place_status = fixed  
  
)
```

## Assigning Pads for Multiple Rows

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

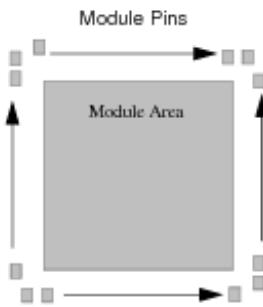


```
version = 3
io_order = clockwise
total_edge = 4
space = 1.06

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

## Assigning Module Pins

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



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

```
)  
)  
)
```

## Recognizing Multiple Corner Cells

The following example shows multiple corner cells defined in I/O file. The `loadIoFile` command recognizes the multiple corner cells defined in I/O file and place them in the right corner with right orientation.

```
version = 3  
  
(iopad  
  (topright  
    (instname = CNR@0001  
  
      orientation = RO  
  
      cell = ZMGACS101N  
  
    )  
    (instname = CNR@0002  
  
      orientation = RO  
  
      cell = ZCGLSNEIS1A  
  
    )  
  )  
)
```

## Performing Area I/O Placement

Before you begin area I/O placement, you must first specify `CLASS PAD AREAIO`, `CLASS PAD` or `CLASS BLOCK` with `CLASS BUMP` in a LEF file. See the "*Data preparation*" section in the [Flip Chip](#)

[Methodologies](#) chapter.

Additionally, a SITE or region must be defined for the `placeAIO` command to place the CLASS PAD AREAIO macro in the required location. The SITE must be referenced in the AREAIO macro.

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

```
SITE IO CLASS PAD ; SIZE 210 BY 100.8 ; END IO
```

```
MACRO INBUF
```

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

```
END PAD
```

**Note:** The bump status can be saved in the DEF file only if the bump status is FIXED or COVER. See [Defining BUMP CELL Placement Status](#).

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

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

```
MACRO PVDD1DGZ

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

  PIN VDD
    DIRECTION OUTPUT ;
    USE POWER ;
    PORT
    CLASS BUMP ;
    LAYER METAL8 ;
    RECT 5.0 25.0 15.0 35.0 ;
  END

  END VDD

END PVDD1DGZ
```

For more information, see "Macro Pin Statement" in the LEF/DEF Language Reference and the "CLASS BUMP Attribute" section in the [Flip Chip Methodologies](#) chapter.

## Defining BUMP CELL in LEF

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

```
MACRO BUMPCELL

    CLASS COVER BUMP ;

    ORIGIN 0 0 ;

    SIZE 80.0 BY 80.0 ;

    SYMMETRY X Y ;

    PIN PAD

        DIRECTION INPUT ;

        USE SIGNAL ;

        PORT

            LAYER M6 ;

            RECT 0.0 0.0 80.0 80.0 ;

            #POLYGON 23.0 0.057.0 0.0 80.0 2

    END

END PAD

END BUMPCELL
```

## Defining BUMP CELL Placement Status

You can define the bump cell placement status, `FIXED` | `COVER` for a bump object in the design, in a DEF/IN file or using the Attribute Editor in Innovus. The bump placement status, `FIXED` or `COVER`, could be saved to DEF file.

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

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

```
Bump: Bump_83_2_8 BUMPCELL 255.720 855.720 refclk -fixed -placeStatus placed
Bump: Bump_82_1_8 BUMPCELL 155.720 855.720 pllrst -fixed -placeStatus cover
```

```
Bump: Bump_81_0_8 BUMPCELL 55.720 855.720 ibias -fixed -placeStatus fixed
```

## Importing LEF Files

To import the LEF files, use the following procedure:

1. Select *File - Import Design*.  
The Design Import form appears.
2. On the Design page, enter the names of the Verilog files, and choose a top cell assignment option.
3. In the LEF Files field, type the LEF file names to import, and include the file that contains the CLASS PAD AREAIO statement. Or, you can click on the ... icon to the right of the field to select files.
4. Click *OK*.  
The Design Import form closes and Innovus imports the data.

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

1. Select *File - Load - Floorplan* or run the `loadFPlan` text command.
2. Select *File - Load - I/O File* or run the `loadIoFile` text command.

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

```
IOFile: iofile_name
```

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

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

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

1. Select *File - Save - Floorplan*. Fill out the form and click *Save*.  
As an alternative, you can specify the `saveFPlan` text command.
2. Select *File - Save - I/O File*. Fill out the form and click *Save*.  
As an alternative, you can specify the `saveIoFile` text command.

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

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

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

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

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

## Preparing Timing Libraries

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

## Encrypting Libraries

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

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

## Parameters

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

## Preparing Timing Constraints

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

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

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

Or, you can use the following command:

```
write_sdc
```

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

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

## Preparing Capacitance Tables

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

For more information on preparing capacitance tables, see chapter [RC Extraction](#).

## Preparing Data for Delay Calculation

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

## Preparing Data for Crosstalk Analysis

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

## Checking Designs

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

To perform these checks, use the following command:

[checkDesign](#)

You can check for the following data:

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

Cadence recommends that you check libraries and data as follows:

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

## Preparing Data in the Timing Closure Design Flow

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

## Converting iPRT Format to LEF

The `iprt2lef` translator converts DRC rules, place-and-route technology data, and RCX data from iDRC, iPRT and iRCX format to the technology LEF format.

For more information about this translator, refer to the *iPRT to LEF Translator Application Note* on Cadence Online Support.

## Importing and Exporting Designs

- Overview
- Verifying Data before Importing a Design
- Preparing the Design Netlist
- The `init_design` Import Flow
  - `init_design` Simple Data Flow
  - Supported `init_design` Invocation Methods
- Importing Designs using the GUI
  - Importing an OpenAccess Design
  - Importing a Design with LEF and Verilog
- Loading a Previously Saved Global Variables File
- Handling Verilog Assigns
- Configuring the Setup for Multi-Mode Multi-Corner Analysis
  - Creating Library Sets
  - Creating Virtual Operating Conditions
  - Creating RC Corner Objects
  - Creating Delay Calculation Corner Objects
  - Adding a Power Domain Definition to a Delay Calculation Corner
  - Creating Constraint Mode Objects
  - Creating Analysis Views
  - Setting Active Analysis Views
  - Checking the Multi-Mode Multi-Corner Configuration
  - Saving Multi-Mode Multi-Corner Configurations
- Saving Designs
  - Saving an OpenAccess Design

- Transferring OpenAccess Data between Innovus and Virtuoso for ECO
- Loading and Saving Design Data
  - Loading a Partition
  - Loading Floorplan Data
  - Loading an I/O Assignment File
  - Loading an FSDB File
  - Saving a Partition
  - Saving Floorplan Data
- Converting an Innovus Database to GDSII Stream or OASIS Format
  - Creating Cells and Instances
  - Renaming LEF Vias
  - Merging GDSII Stream or OASIS Files
  - Merge Examples
- About the GDSII Stream or OASIS Map File
  - Map File Format
  - Map File Columns
  - Specifying Object Subtypes
  - Using Multiple Layers and Data Types
- Updating Files During an Innovus Session
- SKILL to TCL Mapping

## Overview

The Innovus® Implementation System (Innovus) software provides the following options for saving, restoring, importing, and exporting design data:

|                              |                                                                                                         |
|------------------------------|---------------------------------------------------------------------------------------------------------|
| Starting (importing) designs | Allows you to specify data for starting or initializing a design.                                       |
| Saving designs               | Allows you to save the work you complete on designs during a design session for access at a later date. |
| Restoring designs            | Allows you to load saved data from a previous design session.                                           |

|                                  |                                                                                                                                                                                             |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Loading design data              | Allows you to load design data saved in various stages of the design process, and to bring data from specific formats (DEF, PDEF, SPEF, SDF, and OA Cellview) into the Innovus environment. |
| Saving and exporting design data | Allows you to save design data in various stages of the design process, and to export data in specific formats (DEF, PDEF, GDS, and OASIS) from the Innovus environment.                    |

## Verifying Data before Importing a Design

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

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

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

```
getCheckMode
```

## Preparing the Design Netlist

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

- To ensure that the design is uniquified automatically after the top cell is flattened, set the following global variable to 1:

[init\\_design\\_uniquify](#)

## The [init\\_design](#) Import Flow

All designs are saved in Innovus using the [init\\_design](#) import model. In this design import model, all analyses are configured the same way using the multi-mode/multi-corner (MMMC) style of configuration, and the configurations are used directly for initialization. This section introduces the basics of the [init\\_design](#)-based data flow. This section has the following subsections:

- [init\\_design Simple Data Flow](#)

- [Supported init\\_design Invocation Methods](#)

## init\_design Simple Data Flow

In the `init_design`-based data flow:

- Global variables that store data explicitly required for the initialization process are prefixed with `init_`.
- All `init_*` global variables have `help`, can be queried, and are stored by `saveDesign` in the `.globals` file.
- Since MMMC syntax can be used to configure one mode or corner as well as many, `init_design` relies on a valid MMMC specification to provide the necessary timing, SI, constraint, and extraction related data for the system

Note: The Innovus `init_design` style configuration cannot be restored by 10.x or earlier releases of the software directly.

`init_*` style variables are used to store design-level and physical data. For example, the `init_mmmc_file` variable is the pointer to the file containing the MMMC configuration. In addition, the `init_cpf_file` provides a pointer to the design's Common Power Format (CPF) file. This is significant for initialization since an MMMC configuration can be derived from CPF. So while a valid MMMC configuration must be available for `init_design`, it is not required that it come specifically from the `init_mmmc_file` pointer.

The Tcl global variables used by `init_design` are:

- [init\\_abstract\\_view](#)
- [init\\_cpf\\_file](#)
- [init\\_design\\_netlisttype](#)
- [init\\_design\\_settop](#)
- [init\\_gnd\\_net](#)
- [init\\_import\\_mode](#)
- [init\\_io\\_file](#)
- [init\\_layout\\_view](#)
- [init\\_lef\\_file](#)
- [init\\_mmmc\\_file](#)

- `init_oa_default_rule`
- `init_oa_design_cell`
- `init_oa_design_lib`
- `init_oa_design_view`
- `init_oa_ref_lib`
- `init_oa_search_lib`
- `init_oa_special_rule`
- `init_pwr_net`
- `init_top_cell`
- `init_verilog`

For more information on the `init_*` globals, see the [Import and Export Global Variables](#) section in the *Innovus Text Command Reference*. Several possible `init_design` scenarios are discussed in a later section of this chapter.

## Supported `init_design` Invocation Methods

You have seen how to get all the data required to bring up an Innovus session with `init_design`. Let us now look at different examples of actually invoking the `init_design` command:

- [Using a Pointer to an MMMC Configuration File](#)
- [Using a Pointer to a CPF File](#)
- [Using `init\_design` with an Inline MMMC Script](#)
- [Using Physical-Only Flow](#)

## Using a Pointer to an MMMC Configuration File

One of the most common ways of invoking `init_design` is to first use initialization variables to define where to find the key pieces of data. The `init_mmmc_file` variable is used to point to a functioning MMMC configuration. Here, functioning is defined as follows:

- The MMMC configuration must include a `set_analysis_view` command and be complete and correct enough to initialize the specified `-setup` view
- At a minimum, timing library information is required.

The following example uses a pointer to an MMMC configuration file before invoking `init_design`:

```
set init_verilog "top.v"  
set init_top_cell "top"  
set init_mmmc_file "viewDefinition.tcl"  
init_design
```

Instead of having the init globals asserted one-by-one, you can also source the file containing the variable settings and then initialize the design as follows:

```
source test.globals
```

```
init_design
```

**Note:** Here, it is assumed that `test.globals` is configured in MMMC mode.

## • Using a Pointer to a CPF File

In the following example, a CPF file is used in place of an explicit `viewDefinition.tcl` file. The MMMC configuration is derived from the CPF:

```
set init_verilog "top.v"  
set init_top_cell "top"  
set init_cpf_file "top.cpf"  
init_design
```

Here:

- The CPF must be a MMMC style-CPF, which means it must contain at least one analysis view definition.
- The design is initialized based on the default power domain's library information.

## • Using `init_design` with an Inline MMMC Script

If you have a script which is creating the MMMC configuration on-the-fly rather than having a pointer to static file, you can still use the `init_design` flow successfully. However, there is a circular dependency problem that needs to be resolved. `set_analysis_view` cannot be issued until the design has been initialized by `init_design`, but `init_design` requires a complete MMMC configuration including the requisite `-setup` and `-hold` view information. The solution is to use the `-setup` and `-hold` options of the `init_design` command itself, instead of using `set_analysis_view` in

this scenario.

```
set init_verilog "top.v"  
  
set init_top_cell "top"  
  
create_delay_corner -name my_delay_corner_max  
    -library_set my_max_library_set  
    -rc_corner    my_rc_corner_worst  
  
create_delay_corner -name my_delay_corner_min  
    -library_set my_min_library_set  
    -rc_corner    my_rc_corner_worst  
  
create_analysis_view -name my_analysis_view_setup  
    -constraint_mode my_constraint_mode  
    -delay_corner    my_delay_corner_max  
  
create_analysis_view -name my_analysis_view_hold  
    -constraint_mode my_constraint_mode  
    -delay_corner    my_delay_corner_min  
  
init_design -setup my_analysis_view_setup  
    -hold  my_analysis_view_hold
```

## • Using Physical-Only Flow

You can also run `init_design` in the absence of an MMMC configuration. This initializes the system into physical-only mode. No access to the timing part of the system is provided under this mode. To reinitialize, you would need to exit the software or run the `freeDesign` command.

## • Importing Designs using the GUI

Before you import a design, you must first prepare the data. For more information, see the [Data Preparation](#) chapter in the [Innovus User Guide](#).

## • Importing an OpenAccess Design

To import an OpenAccess design, complete the following steps:

- Select *File - Import Design*.
- Select *OA*.

- Specify the following information:
  - *Library*  
Specifies the OpenAccess database library.
  - *Cell*  
Specifies the OpenAccess database cell.
  - *View*  
Specifies the OpenAccess database view.
- Specify the following OpenAccess technology and physical library information:
  - *OA Reference Libraries*  
Specifies the OpenAccess reference libraries to import. The first OpenAccess reference library listed in this field must contain the technology information for the leaf cells.  
Each reference library is processed using the abstract view name list (*Abstract View Names*).  
  
For example, if the reference library is "lib1 lib2", and the abstract view name list is "abstract abstract2", LEF MACRO information is processed for lib1 with the abstract view. Then, for any cells in lib1 that did not have abstract, but did have abstract2, that view is processed for MACRO information. If a cell has both views, the first one is used.  
The process then is repeated for lib2.
  - *OA Abstract View Names*  
Specifies the OpenAccess view names that the software should examine to find the equivalent LEF MACRO information (for example, PINS, OBS, FOREIGN).
  - *OA Layout View Names*  
Specifies the layout view names (separated by spaces) to import.
- Click *Save* or *OK*.
  - *Save* saves your settings to a configuration file. The design is not imported.
  - *OK* uses the current settings to import the design. The configuration file is not updated.

## ● Importing a Design with LEF and Verilog

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

- Select *File - Import Design*.

- Specify the gate-level Verilog netlist files to import in the *Files* text field.
- Select one of the following options to specify the top cell:
  - *Auto Assign*  
Automatically extracts the top cell name from the netlist, provided the netlist contains only one design.
  - *By User*  
(Default) Specifies the name of the top cell when a netlist contains more than one design (more than one top design name). The top cell name specified is the design the software imports and processes.
- Specify the LEF files to import. You must specify the technology LEF file first, then specify the standard cell LEF and block LEF in any order.  
The LEF file provides technology information, such as metal layer and via layer information and via generation rules, which is used in the Add Rings and Add Stripes forms. The router also uses the technology information contained in the LEF file.

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

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

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

## ● Loading a Previously Saved Global Variables File

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

- Select *File - Import Design*.
- Click *Load*. The Load Global Variables form is displayed.
- Select the directory of the file you want to load.
- Select *Global Variable files (\*.globals)* in the *Files of type* field.
- Specify a file name or click on the filename in the window. The filename suffix is *.global*.
- Click *Open*. The Load Global Variables form closes. The global variable file is loaded.
- In the Design Import form, continue to specify data you want to import into the design.

- Click *Save* or *OK*.
  - *Save* saves your settings to the global file. The design is not imported.
  - *OK* saves your settings to the global file and starts the design import process. This might take several minutes to complete, depending on the size of your design. When the design is loaded, the *Design Import* form closes and the design displays in the Innovus main window.

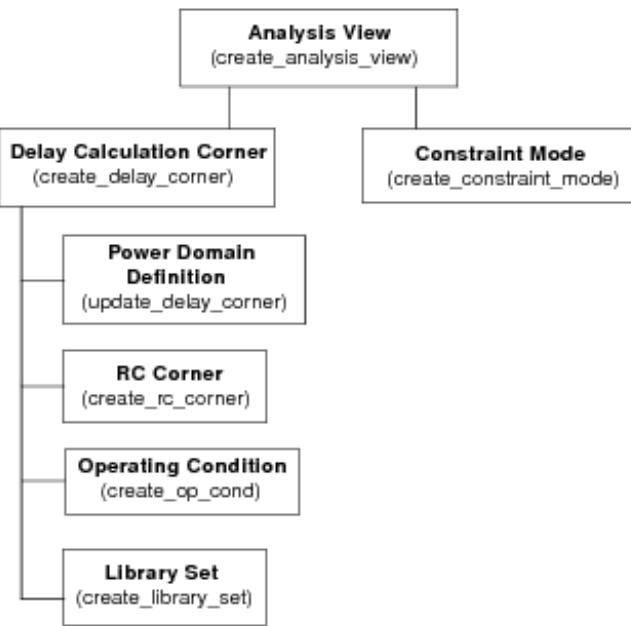
- **Handling Verilog Assigns**

Verilog assign statements may be added, removed, or replaced with buffers automatically by Innovus. All Innovus applications, including GigaOpt, CTS, CCopt, Place, Route, Hierarchy/ILM Flow, MSV, and manual ECO, can handle verilog assign nets natively.

- **Configuring the Setup for Multi-Mode Multi-Corner Analysis**

Multi-mode multi-corner analysis uses a tiered approach to assemble the information necessary for timing analysis and optimization. Each top-level definition (called an analysis view) is composed of a delay calculation corner and a constraint mode. The active analysis views defined in the software represent the different design variations that will be analyzed.

**Figure 6-1 Hierarchical Analysis View Configuration**



## • Creating Library Sets

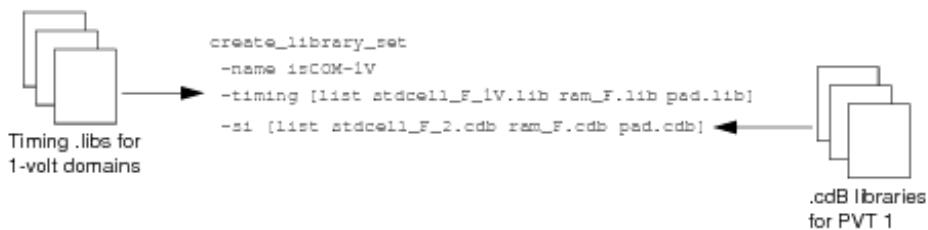
Complex designs can require the specification of multiple library files to define all of the standard cells, memory devices, pads, and so forth, included in the design. Different library sets can be defined to provide uniquely characterized libraries for each delay corner or power domain.

Library sets allow a group of library files to be treated as a single entity so that higher-level descriptions (delay calculation corners) can simply refer to the library configuration by name. A library set can consist of just timing libraries, or it also can include cdB libraries to keep timing and signal integrity libraries in sync throughout a multi-corner flow.

The same library set can be referenced multiple times by different delay calculation corners. To create a library set, use the following command:

`create_library_set`

The following figure shows the creation of a library set that associates timing libraries and cdB libraries with a nominal voltage of 1 volt with the library name `ISCOM-1V`:



## Editing a Library Set

To change the timing and cdB library files for an existing library set, use the following command:

[update\\_library\\_set](#)

You also can make changes to a library set using the Edit Library Set form:

- Choose File - Import Design. Click the Create Analysis Configuration button under Analysis Configuration. In the MMMC Browser form, double click on the name of the library set you want to edit.  
or:
- Choose Timing - Configure MMMC, and double click on the name of the library set you want to edit.

**i** You can use the `update_library_set` command or the Edit Library Set form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## • Creating Virtual Operating Conditions

Generally in most user environments, the process, voltage, and temperature (PVT) point is specified by referring to a predefined operating condition definition in a specific timing library. The library operating condition provides the system with values for P, V, and T, and these then are used to calculate derating parameters and other aspects of the analysis. However, there are situations when there are no predefined operating conditions in the user timing libraries, or the pre-existing operating conditions are not consistent with the user's operating environment.

Instead of actually modifying the timing libraries to add or adjust operating condition definitions, you

can create a set of virtual operating conditions for a library, to define a PVT operating point. These virtual operating conditions can then be referenced by a delay corner as if they actually existed in the library.

To create a virtual operating condition for a library, use the following command:

`create_op_cond`

For example, the following command creates a virtual operating condition called `PVT1` for the library `IsCOM-1V`:

```
create_op_cond -name PVT1
  -library_file IsCOM-1V.lib
  -P 1.0
  -V 1.2
  -T 120
```

### Editing a Virtual Operating Condition

You can add, delete, or change attributes for a defined virtual operating condition using the *Edit Operating Condition* form.

**i** You can edit a virtual operating condition before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

- Choose *File - Import Design*. Click the *Create Analysis Configuration* button under *Analysis Configuration*. In the MMMC Browser form, double click on the name of the library set you want to edit.  
or:
- Choose *Timing - Configure MMMC*, and double click on the name of the operating condition you want to edit.

## • Creating RC Corner Objects

An RC corner object provides the software with all of the information necessary to properly extract, annotate, and use the RCs for delay calculation. RC corner objects also control the attributes for

running sign-off extraction sequentially on each RC corner.

For each active RC corner in the design, the software extracts and stores a unique set of parasitics. You must use the RC corner attributes to control RC scaling when running the software in multi-mode multi-corner analysis mode.

RC corner objects are referenced when creating delay calculation corner objects.

To create an RC corner, use the following command:

[create\\_rc\\_corner](#)

For example, the following command creates an RC corner called `rc-typ` that uses the capacitance table `myTech_nc.CapTbl`, and derates the resistance values based on the temperature of 50 Celsius:

```
create_rc_corner -name rc-typ -cap_table myTech_nc.CapTbl -T 50
```

### Editing an RC Corner Object

To add or change attribute values for an existing RC corner object, use the following command:

[update\\_rc\\_corner](#)

You also can make changes to an RC corner object using the Edit RC Corner form:

- Choose File - Import Design. Click the Create Analysis Configuration button under Analysis Configuration. In the MMMC Browser form, double click on the name of the library set you want to edit.  
or:
- Choose Timing - Configure MMMC, and double click on the name of the RC corner object you want to edit.

**i** You can use the `update_rc_corner` command or the Edit RC Corner form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

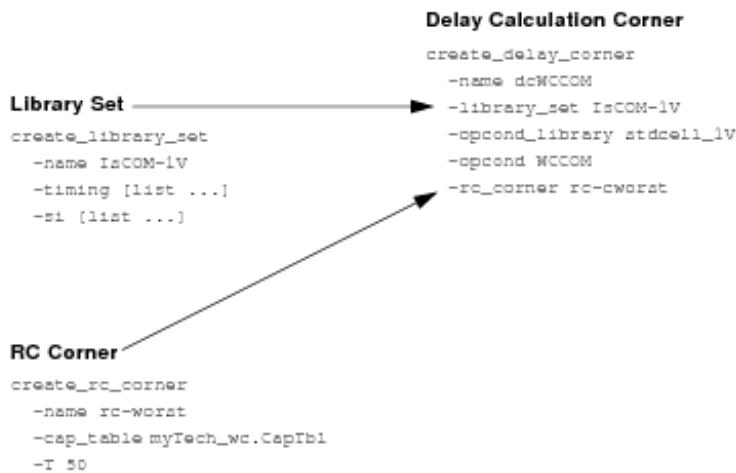
## • Creating Delay Calculation Corner Objects

A delay calculation corner provides all of the information necessary to control delay calculation for a specific analysis view. Each corner contains information on the libraries to use, the operating conditions with which the libraries should be accessed, and the RC extraction parameters to use for calculating parasitic data. Delay corner objects can be shared by multiple top-level analysis views. To create a delay calculation corner, use the following command:

`create_delay_corner`

- Use separate delay calculation corners to define major PVT operating points (for example, Best-Case and Worst-Case).
- Use the `-early_*` and `-late_*` parameters within a single delay calculation corner to control on-chip variation.

The following figure shows the creation of a delay calculation corner called `dcWCCOM`. This corner uses the libraries from `IsCOM-1V`, sets the operating condition to `wccom`, as defined in the `stdcell_1v` timing library, and uses the `rc-cworst` RC corner:



## Editing a Delay Corner Object

To add or change attribute values of an existing delay calculation corner object, use the following command:

`update_delay_corner`

You also can make changes to a delay calculation corner object using the *Edit Delay Corner* form:

- Choose File - Import Design. Click the Create Analysis Configuration button under Analysis Configuration. In the MMMC Browser form, double click on the name of the library set you want to edit.  
or:

- Choose Timing - Configure MMMC, and double click on the name of the delay calculation corner you want to edit.

**i** You can use the update\_delay\_corner command or the Edit Delay Corner form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## • Adding a Power Domain Definition to a Delay Calculation Corner

A single delay calculation corner object specifies the delay calculation rules for the entire design. If a design includes power domains, the delay calculation corner can contain domain-specific subsections that specify the required operating condition information, and any necessary timing library rebinding for the power domain.

To create a power domain definition for a delay calculation corner, use the following command:

[update\\_delay\\_corner](#)

For example, the following command adds a definitions for the power domain `domain-3V` to the `dcWCCOM` delay calculation corner:

```
update_delay_corner -name dcWCCOM
  -power_domain domain-3V
  -library_set libs-3volt
  -opcond_library delayvolt_3V
  -opcond slow_3V
```

### Editing a Power Domain Definition

To add or change attribute values for an existing power domain definition, use the following command:

[update\\_delay\\_corner](#)

You also can make changes to a power domain definition using the Edit Power Domain form:

- Choose *File - Import Design*. Click the *Create Analysis Configuration* button under *Analysis Configuration*. In the MMMC Browser form, click the + next to *Delay Corners* to list the available delay corners, click the + next to the delay corner to which the power domain

definition belongs, and double click on the name of the power domain definition.

or

- Choose *Timing - Configure MMMC*, click the + next to Delay Corners to list the available delay corners, click the + next to the delay corner to which the power domain definition belongs, and double click on the name of the power domain definition.

**i** You can use the update\_delay\_corner command or the Edit Power Domain form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## • Creating Constraint Mode Objects

A constraint mode object defines one of possibly many different functional, test, or Dynamic Voltage and Frequency Scaling (DVFS) modes of a design. It consists of a list of SDC constraint files that contain timing analysis information, such as the clock specifications, case analysis constraints, I/O timings, and path exceptions that make each mode unique. SDC files can be shared by many different constraint modes, and the same constraint mode can be associated with multiple analysis views.

To create a constraint mode object, use the following command:

`create_constraint_mode`

The following figure shows the grouping of the SDC files `io.sdc`, `mission1-clks.sdc`, and `mission1-except.sdc` to create a mode object named `missionSetup`:



### Editing a Constraint Mode Object

To change the SDC constraint file information for an existing constraint mode object, use the following command:

#### [update\\_constraint\\_mode](#)

You also can make changes to a constraint mode object using the Edit Constraint Mode form:

- Choose File - Import Design. Click the Create Analysis Configuration button under Analysis Configuration. In the MMMC Browser form, double click on the name of the library set you want to edit.  
or:
- Choose Timing - Configure MMMC, and double click on the name of the constraint mode object you want to edit.

**i** You can use the `update_constraint_mode` command or the Edit Constraint Mode form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

### • Entering Constraints Interactively

You can use the `set_interactive_constraint_modes` command to update assertions for a multi-mode multi-corner constraint mode object, and have those changes take immediate effect.

Specifying `set_interactive_constraint_modes` puts the software into interactive constraint entry mode for the specified constraint mode objects. Any constraints that you specify after this command will take effect immediately on all active analysis views that are associated with these constraint modes. By default, no constraint modes are considered active.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all views in the current session that are associated with the constraint mode `func1`:

```
set_interactive_constraint_modes func1
```

```
set_propagated_clock [all_clocks]
```

The software stays in interactive mode until you exit it by specifying the `set_interactive_constraint_modes` command with an empty list:

```
set_interactive_constraint_modes { }
```

All new assertions are saved in the SDC file for the specified constraint mode when you save the design (`saveDesign`).

The `all_constraint_modes` command can be used to generate a list of constraint modes as the argument for this command.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all active analysis views in the current session.

```
set_interactive_constraint_modes [all_constraint_modes -active]
```

```
set_propagated_clock [all_clocks]
```

Use the `get_interactive_constraint_modes` command to return a list of the constraint mode objects in interactive constraint entry mode.

**Note:** Interactive constraint mode only works when the software is in multi-mode multi-corner timing analysis mode. In min/max analysis mode, constraints are always accepted interactively.

- **Constraint Support in Multi-Mode and Multi-Mode Multi-Corner Analysis**

The Innovus software isolates the following SDC constraints from conflicting with each other, in both multi-mode and multi-mode multi-corner analysis modes:

- `create_clock`
- `create_generated_clock`
- `set_annotated_check`
- `set_annotated_delay`
- `set_annotated_transition`
- `set_case_analysis`
- `set_clock_gating_check`
- `set_clock_groups`
- `set_clock_latency`
- `set_clock_sense`
- `set_clock_transition`

- `set_clock_uncertainty`
- `set_disable_timing`
- `set_drive`
- `set_driving_cell`
- `set_false_path`
- `set_fanout_load`
- `set_input_delay`
- `set_input_transition`
- `set_load`
- `set_max_delay`
- `set_max_time_borrow`
- `set_max_transition`
- `set_min_delay`
- `set_min_pulse_width`
- `set_multicycle_path`
- `set_output_delay`
- `set_propagated_clock`
- `set_resistance`

**Note:** Path groups defined with `group_path` are considered to be global definitions across all analysis views.

- **Creating Analysis Views**

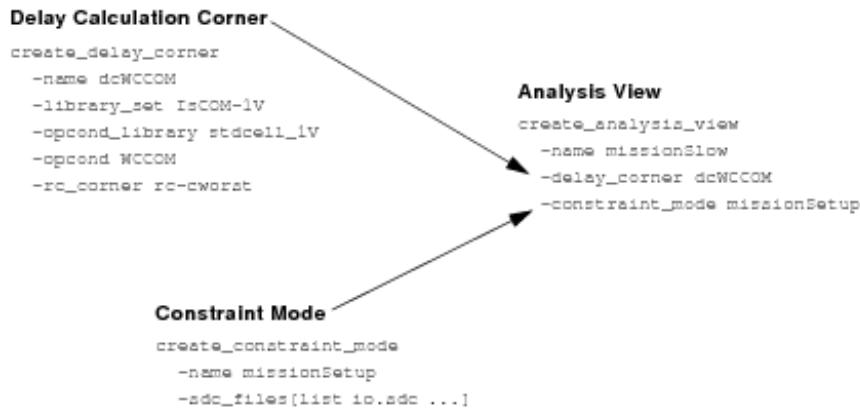
## - Creating Analysis View -

An analysis view object provides all of the information necessary to control a given multi-mode multi-corner analysis. It consists of a delay calculation corner and a constraint mode.

- To create an analysis view, use the following command:

[create\\_analysis\\_view](#)

The following figure shows the creation of the analysis view `missionSetup`:



## Editing an Analysis View Object

To change the attribute values for an existing analysis view, use the following command:

[update\\_analysis\\_view](#)

You also can make changes to an analysis view using the Edit Analysis View form:

- Choose File - Import Design. Click the Create Analysis Configuration button under Analysis Configuration. In the MMMC Browser form, double click on the name of the library set you want to edit.  
or:
- Choose *Timing - Configure MMMC*, and double click on the name of the analysis view you want to edit.

**i** You can use the `update_analysis_view` command or the Edit Analysis View form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## • Setting Active Analysis Views

After creating analysis views, you must set which views the software should use for setup and hold analysis and optimization. These "active" views represent the different design variations that will be analyzed. Active views can be changed throughout the flow to utilize different subsets of views. Innovus applications can handle the views concurrently or sequentially, depending on their individual capabilities. Libraries and data are loaded into the system, as required to support the selected set of active views.

To set active analysis views, use the following command:

```
set_analysis_view
```

**Note:** You must define at least one setup and one hold analysis view.

For example, the following command sets `missionSlow` and `mission2Slow` as the active views for setup analysis, and `missionFast` and `testFast` as the active views for hold analysis:

```
set_analysis_view
  -setup {missionSlow mission2Slow}
  -hold {missionFast testFast}
```

## • Guidelines for Setting Active Analysis Views

- The order in which you specify views using the `set_analysis_view` command is important. By default, the first views defined in the `-setup` and `-hold` lists are the default views. Certain Innovus applications that do not support multi-mode multi-corner can only process the data defined for a single view. These applications use the information defined for the default view.
- Concurrent analysis of views for timing optimization costs memory and CPU.

## • Changing the Default Active Analysis View

Some Innovus applications can function only on a single analysis view at a time. By default, these single-view applications use the default analysis view. If an application or flow step does not provide a native option for specifying which view to use, you can use the `set_default_view` command to temporarily change the default view to a different active view that is better suited to that

flow step.

For example, if the analysis view `missionSlow` is currently the default active setup view in the design, the following command temporarily changes the default view to `mission2Slow`:

```
set_default_view -setup mission2Slow
```

**Note:** Using the `set_default_view` command does not affect software performance because it only uses views that are already active in the design. If you use the `set_analysis_view` command to change the default views, the existing timing, delay calculation, and RC data is reset.

## • Checking the Multi-Mode Multi-Corner Configuration

Use the following command to generate a hierarchical report of your current MMMC configuration:

```
report_analysis_views
```

You can customize the report to show only the active setup or hold analysis views, all of the active views, or all of the defined views in the design, including those that are currently inactive.

You can also use the Innovus GUI to review the MMMC configuration:

- Select *File* -> *Import Design* -> *Create Analysis Configuration*
- or
- Select *Timing* -> *Configure MMMC*.

For more information, see the [File Menu](#) chapter in the *Innovus Menu Reference Guide*.

## • Changing How the MMMC Browser Displays Configuration Information

By default, the MMMC Browser displays configuration information in two columns: one displays the different existing analysis views, and the other displays the different existing multi-mode multi-corner objects.

You can change how the MMMC Browser displays configuration information using the MMMC Preferences form. You can use this form to change the number of columns displayed, and rename the titles of the columns.

You also can use this form (and the Add Object form) to add and delete objects from columns, and rearrange the order in which the objects are listed, by clicking and holding the left mouse button on an object name, and dragging it to a different position in the list.

- Choose *File* - *Import Design*. Click the *Create Analysis Configuration* button under *Analysis*

*Configuration.* In the MMMC Browser form, click the *Preferences* button.

or:

- Choose *Timing - Configure MMMC*, and click the *Preferences* button.

## • Saving Multi-Mode Multi-Corner Configurations

The software stores the multi-mode multi-corner configuration as Tcl commands in a view definition file. The view definition file contains all of the library set, RC corner, delay calculation corner, constraint mode, and analysis view definitions that you created. When you specify the `saveDesign` command, the software saves the file to the save directory, and updates the configuration file with a pointer to the file. This multi-mode multi-corner configuration will be reloaded automatically by the subsequent use of the `restoreDesign` command.

Updated SDC files for each mode are saved to the save directory, if ECO changes were made that affect pins that have constraint assertions.

## • Saving Designs

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

- Use the text command as follows:

```
saveDesign sessionName
```

or

- In the Innovus GUI, click the *Save Design* command on the *File Menu* and then click the *Innovus* option button in the *Save Design* form.

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

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

### • Saving an OpenAccess Design

## • **Saving an OpenAccess Design**

A Verilog based design that was loaded from *File - Import Design* can only be saved into OpenAccess if it uses OpenAccess reference libraries. It cannot be saved in OpenAccess if LEF files were used.

## • **Transferring OpenAccess Data between Innovus and Virtuoso for ECO**

- From an Innovus session, save the OpenAccess design.

```
saveDesign sessionName -cellview { lib cell view }
```

- Exit the Innovus session.
- Open the OpenAccess database in Virtuoso XL (VXL) and edit the design.  
Note: You must use VXL rather than the Virtuoso Layout Editor.
- Save the design.
- Exit the VXL tool.
- Start an Innovus session.
- Restore the OpenAccess design.

```
restoreDesign sessionName topCell -cellview { lib cell view }
```

## • **Loading and Saving Design Data**

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

### • **Loading a Partition**

To load a partition, you can use the *Load - Partition* command from the [File Menu](#). Before you load a partition, perform the following tasks:

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

When you load a partition design, the Innovus software rebuilds the individual partition and the top level, so that the entire chip can be analyzed. When you load a saved partition, the software loads

all the files that are selected in the *Load Partition File* form.

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

## • Loading Floorplan Data

To load floorplan data, use the following command from the *File* menu:

*Load > Floorplan*

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

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

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

- **Placement File Requirement**

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

- **Loading an I/O Assignment File**

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

- **Loading an FSDB File**

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

- **Saving a Partition**

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

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

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

- **Saving Floorplan Data**

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

- Floorplan dimensions

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

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

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

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

## • **Converting an Innovus Database to GDSII Stream or OASIS Format**

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

- For GDSII Stream format:
  - [setStreamOutMode](#)
  - [streamOut](#)  
Create GZIP files by appending .gz to the filename. The `streamOut -merge` command can read files with the .gz extension.

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

- *Mode Setup - StreamOut* from the [Options Menu](#).
- *Save - GDS/OASIS* from the [File Menu](#).

- For OASIS format:
  - [setOasisOutMode](#)
  - [oasisOut](#)

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

- *Mode Setup - OasisOut* from the [Options Menu](#).
- *Save - GDS/OASIS* from the [File Menu](#).

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

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

- **Related Topics**

For more information, see [Merging GDSII Stream or OASIS Files](#).

- **Creating Cells and Instances**

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

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

## • Renaming LEF Vias

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

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

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

`topStructureName_VIA index`

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

For more information, see `setStreamOutMode` or `setOasisOutMode` in the *Innovus Text Command Reference*.

## • Merging GDSII Stream or OASIS Files

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

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

### • Merging Files Using the Command Line

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

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

If you specify the `-uniquifyCellNames` parameter, you must list the top-level file first, as the software uses the first name in the search path when renaming cells. For more information,

see "Merge Examples".

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

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

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

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

## • **Merge Examples**

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

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

### • **Case 1**

Most cases are similar to the following:

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

The design instantiates cells X and Y.

- For examples of cases where hierarchical cells are involved and the contents of a hierarchical cell is different from another cell with the same name, see **Case 2**.

- **Example 1**

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

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

- **Example 2**

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

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

- **Example 3**

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

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

- **Example 4**

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

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

- **Results**

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

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

- **Case 2**

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

- GDS1 contains cells x, y (y is instantiated in x).
- GDS2 contains cell y.

The y cells in the files contain different information.

The design instantiates cells x and y.

- **Example 5**

- ```
streamOut -merge {GDS1 GDS2}
```
- GDS1 processed: x and y are copied from GDS1.
  - GDS2 processed: y from GDS2 is dropped.

- **Example 6**

- ```
streamOut -merge {GDS2 GDS1}
```
- GDS2 processed: y from GDS2 is copied from GDS2.
  - GDS1 processed: x is copied from GDS1, y is dropped (references from x to y now use the one copied from GDS2).

- **Example 7**

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

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

- **Example 8**

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

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

- **Results**

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

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

For more information, see the following commands:

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

- **Merging GDS/OASIS Files Using the GUI**

Use the GDS/OASIS Export form.

- Choose *File - Save - GDS/OASIS*.
- Fill in the appropriate fields on the form.

For more information, see *Save - GDS/OASIS* in the [File Menu](#) chapter of the *Innovus Menu*

Reference.

- **About the GDSII Stream or OASIS Map File**

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

- **Map File Format**

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

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

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

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

```
#This comment is the first argument on a line#
METAL1      NET          1          0
METAL1      SPNET        999        0
#This comment is preceded by white space#
METAL1      PIN          1000       0
#This comment is preceded by a tab#
METAL1      LEFPIN       2000       0
METAL1      FILL          3000       0
METAL1      VIA           4000       0 #This comment is at the end of a line#
METAL1      VIAFILL       5000       0
METAL1      LEFOBS        10000      0
NAME        METAL1/NET    20000      0
```

## • Map File Columns

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

|                 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 | NAME         | <p>Specifies a text label for the layer name and associated object type. If you do not want to output text labels, remove the NAME lines from the file.</p> <p>There is no limit on the length of a structure (cell) name. Because some GDS/OASIS readers have a 32-character limit, the Innovus software issues a warning message when a structure name is longer than 32 characters.</p> <p>If <i>layerObjName</i> is NAME, <i>layerObjType</i> can be a composite layer name/object type (LEFPIN, NET, PIN, or SPNET), or COMP.</p> <ul style="list-style-type: none"> <li>● LEFPIN places the label on the LEF MACRO PIN shape.<br/>(Applies only when the -outPutMacros parameter is specified. For more information, see <a href="#">streamOut</a> or <a href="#">oasisOut</a>.)</li> <li>● NET places the label on the NET.</li> <li>● PIN places the label on the PIN or I/O abstract pad.</li> <li>● SPNET places the label on the SPECIALNET.</li> <li>● COMP places the label on the placed DEF component.</li> </ul> |
| <i>datatype</i> |              | <p>Specifies an object type.</p> <p>You can specify a subtype for some <i>layerObjTypes</i>. For more information, see <a href="#">Specifying Object Subtypes</a>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|                 | ALL          | <ul style="list-style-type: none"> <li>● In routing layers, ALL is equivalent to NET, SPNET, VIA, PIN, LEFPIN, FILL, FILLOPC, LEFOBS, VIAFILL, and VIAFILLOPC.</li> <li>● In cut layers, ALL is equivalent to VIA, VIAFILL, and VIAFILLOPC.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|                 | BLOCKAGE     | Equivalent to DEF BLOCKAGES without + FILLS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|                 | BLOCKAGEFILL | Equivalent to DEF BLOCKAGES with + FILLS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|  |         |                                                                                                                                                                                                                                                                                                                                                                  |
|--|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | CUSTOM  | Applies to text labels created via the <a href="#">add_gui_text</a> command                                                                                                                                                                                                                                                                                      |
|  | FILL    | <p>Equivalent to DEF FILLS without + OPC or DEF SPECIALNETS with + SHAPE FILLWIRE.</p> <p>You can separate FILL into floating and connected fill by specifying the FLOATING subtype. For more information, see "Fill Subtype" in the <i>Specifying Object Subtypes</i> section of this chapter.</p>                                                              |
|  | FILLOPC | <p>Equivalent to DEF FILLS with + OPC or DEF SPECIALNETS + SHAPE FILLWIREOPC.</p> <p>You can separate FILLOPC into floating and connected fill by specifying the FLOATING subtype. For more information, see "Fill Subtype" in the <i>Specifying Object Subtypes</i> section of this chapter.</p> <p><b>Note:</b> DEF 5.6 does not support this object type.</p> |
|  | LEFOBS  | Equivalent to LEF OBS. (Applies only when the -outPutMacros parameter is specified. For more information, see <a href="#">streamOut</a> or <a href="#">oasisOut</a> .)                                                                                                                                                                                           |
|  | LEFPIN  | Equivalent to LEF PIN. (Applies only when the -outPutMacros parameter is specified. For more information, see <a href="#">streamOut</a> or <a href="#">oasisOut</a> .)                                                                                                                                                                                           |
|  | NET     | Equivalent to DEF NETS wiring. For more information, see "Net Name Subtype" in the <i>Specifying Object Subtypes</i> section of this chapter.                                                                                                                                                                                                                    |
|  | PIN     | Equivalent to DEF PINS.                                                                                                                                                                                                                                                                                                                                          |
|  | SPNET   | Equivalent to DEF SPECIALNETS without + SHAPE FILLWIRE or + SHAPE FILLWIREOPC. For more information, see "Net Name Subtype" in the <i>Specifying Object Subtypes</i> section of this chapter.                                                                                                                                                                    |

|                    |            |                                                                                                                                                                                                                                                                                         |
|--------------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | TEXT       | Applies to text labels created via the <a href="#">add_text</a> command.                                                                                                                                                                                                                |
|                    | VIA        | For via master creation for regular vias. For more information, see "Net Name Subtype" in the <i>Specifying Object Subtypes</i> section of this chapter.                                                                                                                                |
|                    | VIAFILL    | You can separate VIAFILL into floating and connected fill by specifying the FLOATING subtype. For more information, see "Fill Subtype" in the <i>Specifying Object Subtypes</i> section of this chapter.                                                                                |
|                    | VIAFILLOPC | You can separate VIAFILLOPC into floating and connected fill by specifying the FLOATING subtype. For more information, see "Fill Subtype" in the <i>Specifying Object Subtypes</i> section of this chapter.<br><br><b>Note:</b> DEF 5.6 does not support this object type."Fill Syntax" |
| <i>layerNumber</i> |            | Specifies the GDSII Stream/OASIS single layer number, comma separated list of layer numbers (for example, 21, 31, 99), or a range of layer numbers (for example, 31-35). The numbers must be integers between 1 and 65535.                                                              |
| <i>dataType</i>    |            | Specifies the GDSII Stream/OASIS single data types, comma separated list of data types (for example, 1, 2, 4), or a range of data types (for example, 1-4). The data type must be an integer between 0 and 65535.                                                                       |

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

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

## Specifying Object Subtypes

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

You can specify the following subtypes:

- Floating and non-floating metal and via fill  
For more information, see "Fill Subtype".
- Net names  
For more information, see "Net Name Subtype".
- Voltage levels  
For more information, see "Voltage Subtype".
- VIA cut sizes  
For more information, see "SIZE Subtype".
- **Fill Subtype**

Use the following syntax to specify metal and via fill:

*layerObjName layerObjType [:FLOATING] layerNumber dataType*

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

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

For example, to divide the output of metal fill shapes, so that non-floating fill on **METAL1** is output to *layerNumber 8 dataType 0* and floating fill to *layerNumber 8 dataType 51*, the map file would have the following statements:

```
METAL1 FILL 8 0
METAL1 FILL:FLOATING 8 51
```

To output the connected metal fill shapes on *METAL1* to *layerNumber 8 dataType 0* and floating fill to both *layerNumber 8 dataType 0* and to *layerNumber 8 dataType 51*, the map file would have the following statements:

```
METAL1 FILL 8 0
METAL1 FILL:FLOATING 8 0,51
```

## • Net Name Subtype

Use the following syntax to specify layers for nets.

For special nets, use the following syntax:

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

For regular nets, use the following syntax:

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

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

For example, to output special net named *vss* on LEF layer *METAL3*, include the following lines in the map file:

```
METAL3 SPNET:VSS 111 0
```

```
METAL3 SPNET 11 0
```

To output via named *vss* on LEF layer *METAL3* to GDS layer *111*, via *vss* on LEF layer *VIA34* on GDS layer *112*, and via *vss* on LEF layer *METAL4* to GDS layer *113*, include the following lines in the map file:

```
# routing/metal layers
METAL3 NET,SPNET 11 0
METAL4 NET, SPNET 13 0
# via cell layers
METAL3 VIA 11 0
VIA34 VIA 12 0
METAL4 VIA 13 0
```

```
# routing/metal layers - net name sub-types
METAL3 SPNET:VSS 111 0
METAL4 SPNET:VSS 113 0
# via cell - net name sub-types
METAL3 VIA:VSS 111 0
VIA34 VIA:VSS 112 0
METAL4 VIA:VSS 113 0
```

## • **Net Name Subtype**

The specified nets can be streamed out into multi layers:

```
layerName NET layerNumber dataType
layerName NET:netName layerNumber dataType
layerName NET:netName layerNumber1 dataType1
```

### **Example:**

```
M2 NET 32 0
M2 NET:clk1 32 10
M2 NET:clk1 132 0
```

## • **Voltage Subtype**

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

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

For example, to output nets on LEF layer *METAL1* with a minimum voltage of 1.8 to *layerNumber 31 dataType 3*, use the following syntax:

```
METAL1 NET:VOLTAGE:1.8 31 3
```

To output nets on LEF layer *METAL1* with a minimum voltage of 1.8 and a maximum voltage of 2.499 to *layerNumber 31 dataType 3*, use the following syntax:

```
METAL1 NET:VOLTAGE:1.8:2.499 31 3
```

If you specify both net names and voltages in the file, the net name overrides the voltage (because

the net name is more specific than the voltage). In the following example, VDD nets are output to `layerName /dataType 31 4`, even whose voltage is between 1.8 and 2.499.

```
METAL1 NET:VDD 31 4
METAL1 NET:VOLTAGE:1.8:2.499 31 1
```

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

```
METAL1 NET 31 0
METAL1 NET:VOLTAGE:1.8:2.499 31 0,1
```

## • SIZE Subtype

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

The map file syntax is as follows:

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

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

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

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

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

```
VIA12 VIA:SIZE:0.1X0.2 41 1
```

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

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

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

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

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

## • MASK Subtype

Use the following syntax to specify `MASK` attribute for designs using processes that require multiple masks per layer:

```
layerObjName layerObjType [:MASK:#] layerNumber dataType
```

### Example

```
Met1 NET,SPNET,PIN 31 0 # collector for "gray/uncolored" data
```

```
Met1 NET,SPNET,PIN:MASK:1 31 1 # output for "MASK1" wiring shapes
```

```
Met1 NET,SPNET,PIN:MASK:2 31 2 # output for "MASK2" wiring shapes
```

```
Met1 VIA 31 0 # collector for "gray/uncolored" data
```

```
Met1 VIA:MASK:1 31 1 # output for "MASK1" via shapes
```

```
Met1 VIA:MASK:2 31 2 # output for "MASK2" via shapes
```

```
Via12 VIA 41 0 # no support for cut layer coloring in Phase 1
```

```
Met2 VIA 32 0 # collector for "gray/uncolored" data
```

```
Met2 VIA:MASK:1 32 1 # output for "MASK1" via shapes
```

```
Met2 VIA:MASK:2 32 2 # output for "MASK2" via shapes
```

...  
...

**Note:** From the 14.1 release, a cut layer in the map file (for a via) can have MASK, SIZE, and VOLTAGE at the same time. For example, you can specify the size of cuts and the mask attribute for design as follows:

```
layer VIA:SIZE:value1xvalue2:MASK:maskvalue layerNumber dataType
```

**Example:**

```
vial VIA:SIZE:0.24x0.24:MASK:2 196 2
```

- **Using Multiple Layers and Data Types**

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

| Use the Following Syntax           | To ...                                      | To Output Layer/Datatype(s) |
|------------------------------------|---------------------------------------------|-----------------------------|
| METAL1 NET 31 0                    | Single layer, single data type              | 31:0                        |
| METAL1 NET 31 0,1                  | Single layer, two data types                | 31:0, 31:1                  |
| METAL1 NET 31,32 0                 | Two layers, single data type                | 31:0, 32:0                  |
| METAL1 NET 31,32 0,1               | Two layers, two data types                  | 31:0, 31:1, 32:0, 32:1      |
| METAL1 NET 31 0<br>METAL1 NET 32 1 | Two layers, each with a different data type | 31:0, 32:1                  |

- **Updating Files During an Innovus Session**

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

| Type | Replace | Update | How                        |
|------|---------|--------|----------------------------|
| ILM  | Y       | Y      | specifyilm<br>unspecifyilm |
| LEF  | N       | Y      | loadLefFile -incremental   |

|                     |   |   |                                                                      |
|---------------------|---|---|----------------------------------------------------------------------|
| QRC Tech File       | Y | N | update_rc_corner -qx_tech_file                                       |
| Timing Libraries    | N | Y | update_library_set                                                   |
| Timing Constraints  | Y | Y | update_constraint_mode<br>set_interactive_constraint_modes<br>source |
| I/O Assignment File | Y | N | loadIoFile                                                           |
| Partition File      | Y | N | specifyPartition                                                     |
| Floorplan File      | Y | N | loadFPlan                                                            |
| Placement File      | Y | N | restorePlace                                                         |
| Routing File        | Y | N | restoreRoute                                                         |
| Special Route File  | Y | Y | Use loadSpecialRoute to replace                                      |
| DEF                 | Y | Y | defIn<br>(use the -scanChain option to update scan chains)           |
| PDEF                | Y | Y | pdefIn                                                               |

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

## • SKILL to TCL Mapping

The following table shows the mapping of Virtuoso SKILL functions to Innovus TCL functions while using the `setOaxMode -bindkeyFile` parameter.

| Virtuoso Key (Default) | SKILL Function   | Innovus Key (Default) | Innovus Command |
|------------------------|------------------|-----------------------|-----------------|
| Shift-k                | leHiClearRuler() | K                     | cleanRuler      |
| Shift-m                | leHiMerge()      | M                     | mergeWire       |

|                      |                                               |      |                 |
|----------------------|-----------------------------------------------|------|-----------------|
| Shift-q              | leEditDesignProperties()                      | Q    | summaryReport   |
| Shift-r              | leHiReShape()                                 | R    | resizeMode      |
| Shift-s              | leHiSearch()                                  | S    | getWireInfo     |
| Shift-u              | hiRedo()                                      | U    | redo            |
| Shift<br><DrawThru3> | hiZoomOut()                                   | Z    | zoomOut         |
| a                    | geSingleSelectPoint()                         | a    | selectMode      |
| c                    | leHiCopy()                                    | c    | copySpecialWire |
| e                    | leHiEditDisplayOptions()                      | e    | popUpEdit       |
| f                    | hiZoomAbsoluteScale<br>(hiGetCurrentWindow()) | f    | fit             |
| k                    | leHiCreateRuler()                             | k    | createRuler     |
| m                    | leHiMove()                                    | m    | moveWireMode    |
| o                    | leHiCreateVia()                               | o    | addViaMode      |
| q                    | leHiEditProp()                                | q    | attributeEditor |
| Shift-o              | leHiRotate()                                  | r    | rotateInstance  |
| s                    | leHiStretch()                                 | s    | stretchWireMode |
| u                    | leUndo()                                      | u    | undo            |
| w                    | hiPrevWinView<br>(hiGetCurrentWindow())       | w    | previousView    |
| z                    | hiZoomIn()                                    | z    | zoomIn          |
| 4 - Down arrow key   | geScroll(nil \\\"n\\\" nil)                   | Up   | panUp           |
| 5 - Down arrow key   | geScroll(nil \\\"s\\\" nil)                   | Down | panDown         |
| 4 - Down arrow key   | geScroll(nil \\\"w\\\" nil)                   | Left | panLeft         |

|                  |                                         |        |                |
|------------------|-----------------------------------------|--------|----------------|
| 5-Down arrow key | geScroll(nil \\\"e\\\" nil)             | Right  | panRight       |
| F2               | geSave()                                | F2     | saveDesign     |
| Delete           | leHiDelete()                            | Delete | deleteSelected |
| Escape           | cancelEnterFun()                        | Escape | cancel         |
| Ctrl-d           | geDeselectAllFig()                      | Ctrl-d | deselectAll    |
| Ctrl-n           | leSetFormSnapMode<br>(\\\"90XFirst\\\") | Ctrl-n | snapFloorplan  |
| Ctrl-r           | hiRedraw()                              | Ctrl-r | redraw         |
| Ctrl-s           | leHiSplit()                             | Ctrl-s | splitWire      |

**Note:** If the `setOaxMode -bindkeyFile` parameter is used, then the Virtuoso Key column applies to Innovus for all of the equivalent commands in the mapping.

# Design Planning Capabilities

---

- Floorplanning the Design
- Using Structured Data Paths
- Bus Planning
- Power Planning and Routing

# Floorplanning the Design

- Overview
- Common Floorplanning Sequence
- Viewing the Floorplan
- Module Constraint Types
  - Target Utilization Display
  - Effective Utilization Display
  - Calculating Density
  - Standard Row Spacing
- Grouping Instances
  - Defining the Bounding Box
  - Adding Logical Hierarchy Without Creating Additional Hierarchy
  - Logical Hierarchy Manipulation
- Creating and Editing Rows
- Using Vertical Rows
- Using Multiple-height Rows
  - Using Integer Multiple-height Rows
  - Using Non-Integer Multiple-height Rows
  - Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows
  - Merging Hierarchical Floorplans from Partitions
- Performing I/O Row Based Pad Placement
  - Prerequisites
  - Enabling the I/O Row Flow in Innovus
  - Use Models
  - Resizing Rectilinear block-level floorplan
- Editing Pins
  - Pin Snapping on Resized Boundaries
  - Moving Pins
  - Swapping Pins
  - Using the Pin Editor
  - Spreading Floating Pins
- Running Relative Floorplanning

- Orientation Key
- Instance Place Example
- Pre-Route Examples
- Saving and Restoring Relative Floorplan
- Saving and Loading Floorplan Data
  - Specific Floorplan Section TCL Export/Import
- Snapping the Floorplan
- Resizing the Floorplan
  - Resize Floorplan Options
  - Setting Resize Lines
  - Specifying Resize Directions
  - Snapping Resize Values
  - Viewing Resize Lines using Color Preferences
  - Distributing I/Os using Resize Floorplan
  - Resizing Floorplan Bounding Box in GUI
- Checking the Floorplan
- FinFET Technology
  - FinFET Support in Innovus
- Related Topics

## Overview

Floorplanning a chip or block is an important task of physical design in which the location, size, and shape of soft modules, and the placement of hard macros are decided. Depending on the design style or purpose, floorplanning can also include row creation, I/O pad or pin placement, bump assignment (flip chip), bus planning, power planning, and more.

For example, floorplanning is very important when preparing the design for timing closure and detailed routing. Floorplanning, in conjunction with placement and trial routing, can be an iterative design process.

The Innovus Implementation System (Innovus) software provides a rich set of commands and GUI functions to floorplan your design interactively. There are also commands for creating an initial floorplan automatically, or, resize a finished floorplan while keeping relative placement of objects.

- For information on floorplan commands, see the [Floorplan Commands](#) chapter, in the *Innovus Text Command Reference*.

- For information on floorplan GUI, see the [Floorplan Menu](#) chapter, in the *Innovus Menu Reference*.

Innovus includes several keyboard shortcuts for use with the floorplanning feature. Make sure you type the bindkey while the main Innovus window is active and the cursor is in the design display area. The [Binding Key](#) form contains a complete list of bindkeys.

To display this form, select *Options - Set Preference* from the Innovus menu, then click the *Binding Key* button on the *Design* tab of the *Preferences* form, or use the default `b` binding key.

## Common Floorplanning Sequence

Floorplanning usually starts by preplacing blocks, modules, and submodules according to the prepared floorplan. All other modules or blocks not in the prepared floorplan are left outside the chip area. The following steps describe the most common sequence for floorplanning:

1. Importing the design.
2. Studying the design's connectivity.
3. Performing the minimum amount of floorplanning based on the chip design floorplan, or do no floorplanning at all.
4. In some cases, no floorplanning is required. For example, a front-end designer might want to predict the quality of the design's netlist by initially placing the entire design without any floorplanning. This iteration provides a good indication of how the blocks should be located and arranged together with the larger modules. After a few iterations, it should be clear how to position the blocks and modules in the floorplan.
5. Running placement and Trial Route to view placement and routing congestion.  
Optionally, running `resizeFP` to enlarge or shrink the die after placement and routing.
6. In this case, floorplanning is done to detail the pre-placement of all blocks, most likely done by a back-end designer to gauge the feasibility of a prepared floorplan.
7. The placer places all remaining blocks that were not preplaced in the floorplan, and also recognizes the floorplan object, such as power and ground routes.
8. If you are at the design's top-level in the display area and want to generate a guide for a submodule, ungroup the top module until you have reached the submodule.
9. Using the full chip placement to refine block (hard macro and blackbox) locations.  
(Optional) Based on the full chip placement results - placement density and routing congestion, running `resize` floorplan to enlarge or shrink the die.
10. View the placements of blocks to determine if you need to change the alignment or orientations.

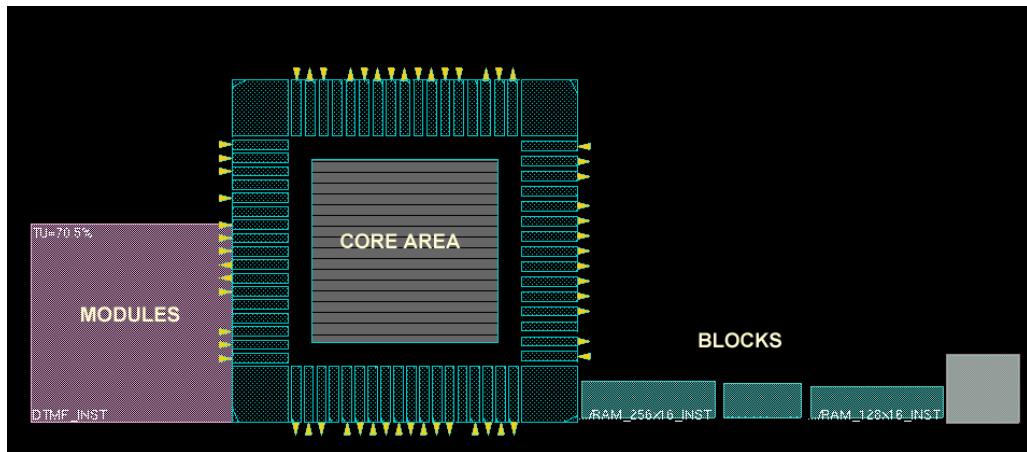
11. Looking for congestion in modules and change heavily congested modules' placement density to a lower percentage (using the `createDensityArea` command).

12. (Optional) If you made any changes in step 5, or especially step 6, rerun placement.

**Note:** To place macros that were not pre-placed during floorplanning, it is recommended that you run `planDesign` first and set the status `fixed', before proceeding to `placeDesign`. This is because, `placeDesign` may not be able to place unfixed hard macros optimally.

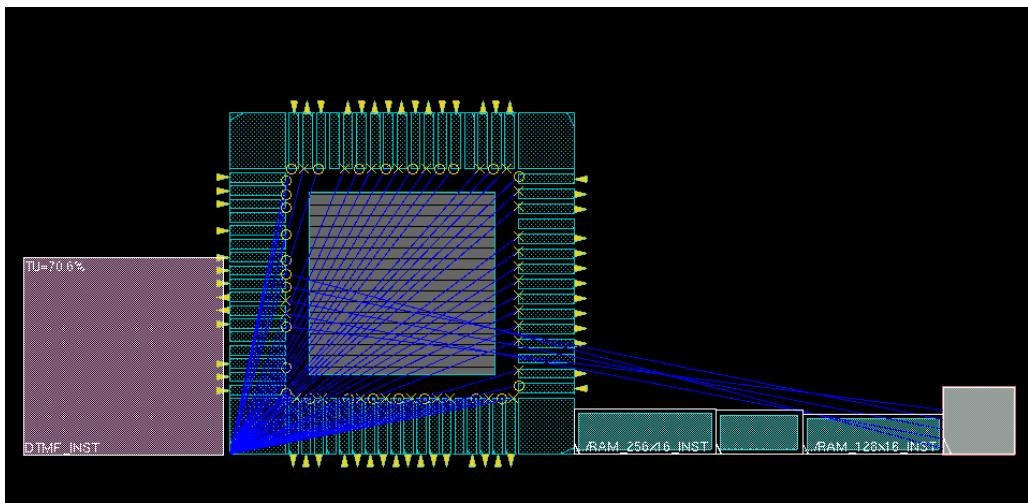
## Viewing the Floorplan

In the design display area, the objects to the left of the core area are the top-level modules, which can be moved and reshaped. The objects to the right of the chip area are the blocks, which can be moved but not reshaped.



Use the `G` key (ungroup), or click the *Hierarchy Down* icon, to display the submodules for a selected module guide. Each time you use the `G` key, you move further down the hierarchy. Use the `g` key (group), or click the *Hierarchy Up* icon, to move up the hierarchy.

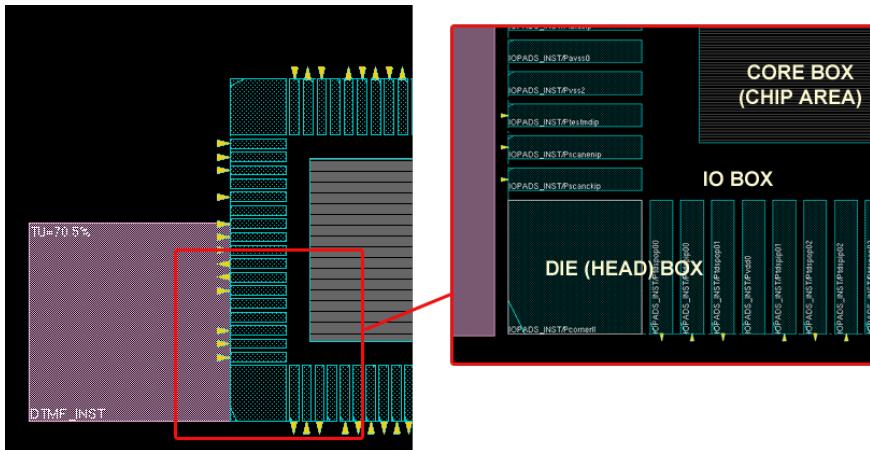
In Floorplan view, you can view the block pins and connection flight lines by clicking on a block or module. Flight lines show the connections and number of connections between the selected module or block to any other modules and blocks.



The pins for blocks are displayed where the flight lines terminate to help you orient the blocks so that the block pins face in the direction that best reduces routing congestion.

To set options for displaying flight lines in the design, select *Options - Set Preference* from the Innovus menu, then click the *Flightline* tab on the *Preferences* form in the GUI.

You can change the die or core size; the margins between the core box and I/O pad instances; and the individual die (head), I/O, or core box sizes. These boxes are shown in the following figure.



You can move module or instance groups outside the core area.

**Note:** Descendant macros and standard cells can move with their ancestor modules when the module is moved in and out of the core area.

## Module Constraint Types

The entire design size is initially calculated during design import, and each module size is calculated. The size of the modules is determined by either the core utilization or the core width and height specifications. The imported design modules can have one of the following constraint types:

### None

The module is not pre-placed in the core design area. The contents of the module are placed without any constraints.

### Guide

The module is preplaced in the core design area. A module guide represents the logical module structure of the netlist. The purpose of a module guide is to guide placement to place the cells of the module in the vicinity of the guide's location. The preplaced guide is a soft constraint, which is discussed later in this section. After the design is imported, but before floorplanning, you can locate module guides on the left side of the core area, which appear as pink objects (by default) in the Floorplan view.

When a module is preplaced in the core design area, it snaps to a standard cell row in the vertical direction and to a metal 2 pitch in the horizontal direction (the default). This default can be changed to snap to the manufacture grid (in the *Preferences* form's *Floorplan* page).

To create a guide for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the [createGuide](#) command or select *Guide* from the Attribute Editor's *Constraint Type* pulldown menu.

You can use the `useGuideBoundary` feature of the [setPlanDesignMode](#) command to define how to handle guide constraint during [planDesign](#). The `useGuideBoundary` feature places the macros that belong to a guide constraint inside the guide boundary during `planDesign`. If the guide is large enough to place all macros, all macros are placed inside the guide boundary. If the space inside the guide is enough, other macros are also allowed to be placed in the guide.

## Fence

The module is a hard constraint in the core design area. After specifying a hierarchical instance as a partition, the constraint type status of a module guide is automatically changed to a fence. The physical outline of a fence module is rigid, and the design for the module is self-contained within the rigid outline. Only child instances must be contained within the partition physical outline; non-child blocks or modules that do not belong to the partition are excluded, and should not be pre-placed within another partition. This restriction is a hard restriction for third party back-end tools where the placement file for a partition does not match the partition netlist.

To create a fence for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the [createFence](#) command or select *Fence* from the Attribute Editor's *Constraint Type* pulldown menu.

**Note:** Fence groups can potentially cause overlaps that cannot be corrected because the Innovus software cannot move the cells out of the group.

## Region

This constraint is the same as a fence constraint except that instances from other modules can be placed within its physical outline by placement. A module guide is changed to a status of Region when preplaced in the core design area.

To create a region for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the [createRegion](#) command or select *Region* from the Attribute Editor's *Constraint Type* pulldown menu.

**Note:** Region groups can potentially cause overlaps that cannot be corrected because the Innovus software cannot move the cells out of the group.

## Soft Guide

This constraint is similar to a guide constraint except there are no fixed locations. This provides stronger grouping for the instances under the same soft guide. The soft guide constraint is not as restrictive as a fence or a region constraint, so some instances might be placed further away if they have connections to other modules.

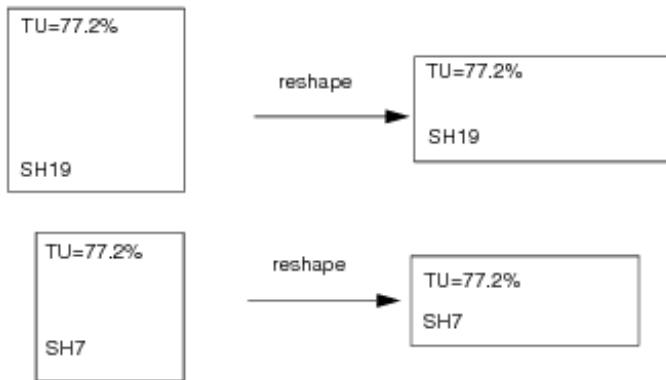
To create a soft guide for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, select *SoftGuide* from the Attribute Editor's *Constraint Type* pulldown menu.

**Note:** Now the module constraints, guide, fence, region, and soft guide, are allowed to be out of core area but must be in the die area.

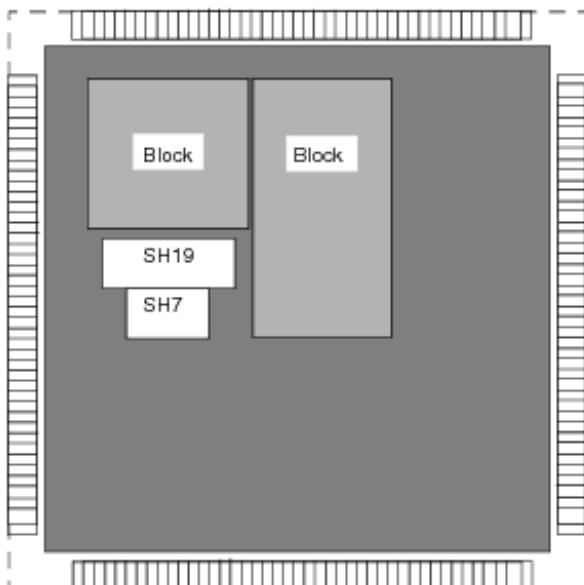
## Target Utilization Display

Module constraints display a target utilization (TU=%) value to represent their physical design size. This is an estimation of module utilization for the given size of the module where only standard cell and hard macro areas are considered; floorplan constraints, such as placement blockages, are not considered. This value is calculated by the standard cells area plus the hard macros area, divided by the module area. The initial TU values are calculated during design import.

The TU percentage helps judge the physical size of a module guide to customize the shape of the module in the floorplan. For example, modules SH19 and SH7 have a TU values of 77.2%. If the modules are reshaped with the same area, they retain their TU values, as shown in the following figure:



You can place them in the core area so they are preplaced close to one another, as shown in the following figure:



The position of the module guide is a placement constraint, and the final definition of the module is determined by several factors. The most important factor – the highest priority of constraint – is the connectivity between itself and other modules. Other floorplan constraints, such as neighboring preplaced module guides, preplaced blocks, placement blockages, and routing blockages, are also considered, but at a lower priority than connectivity.

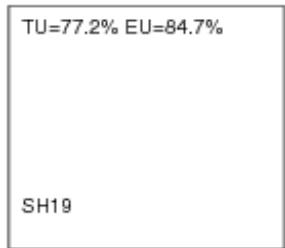
**Note:** You can use a stronger constraint for keeping modules SH19 and SH7 close together using the Group Instances form, and even a stronger constraint by saving the regrouped netlist.

Unlike module guides, the position of fences and regions is a hard placement constraint and are not moved by the same factors.

## Effective Utilization Display

For fences and regions, you can display the effective utilization (EU=%) value. The EU value takes into account the actual cells and hard macros in the fence or region, placement or routing blockages, partition cuts, and other floorplan constraints. It is a good practice to update the EU value before running placement.

Click the *Display/Calculate Effective Utilization* toolbar widget (the % button above the design display area) to display the EU value for each fence and region, as shown in the following figure.



**Note:** The displayed EU values are not automatically updated. You must click the *Display/Calculate Effective Utilization* toolbar widget each time you want to display the updated EU value. This calculation could be time consuming, especially for larger designs.

**Note:** If the EU value is at or exceeds 100% for a fence or region, placement changes the fence or region to a guide. To avoid this, before you run placement, make sure to check and update the EU value, if necessary.

## Calculating Density

When specifying the floorplan, you can determine the core and module sizes by total density or standard cell density using the *Core Utilization* or *Cell Utilization* options, respectively, in the [Specify Floorplan](#) form.

Core Utilization determines the initial size of the core area and the initial size of the pink module guides off to the left of the die area. The total density is calculated as follows:

$$\text{Core Size} = (\text{standard cell area} + \text{macro area} + \text{halo}) / \text{core utilization}$$

In determining the size of the core area and module guides, standard cells and hard macros are treated the same. However, you can determine how densely objects can be packed by weighing the standard cell density separately from the hard macro density. The standard cell density is calculated as follows:

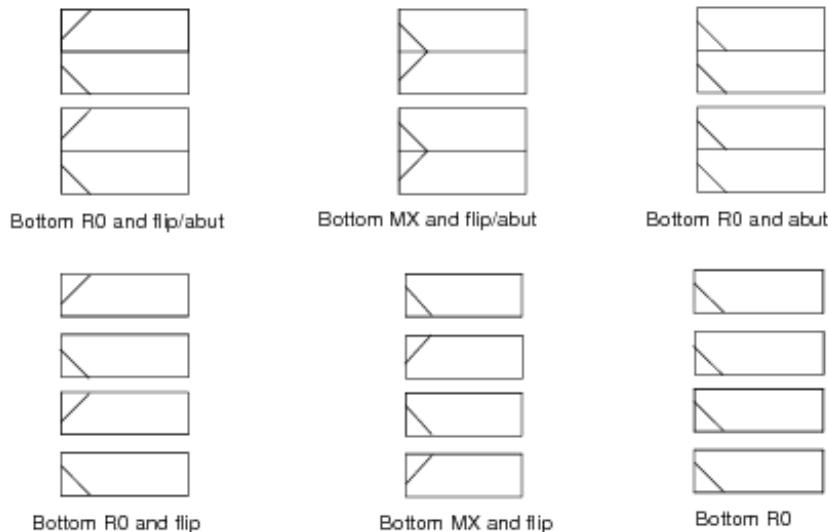
$$\text{Core Size} = (\text{standard cell area} / \text{cell utilization}) + \text{macro area} + \text{halo}$$

The size of the core is smaller once you specified your floorplan by using *Cell Utilization*.

## Standard Row Spacing

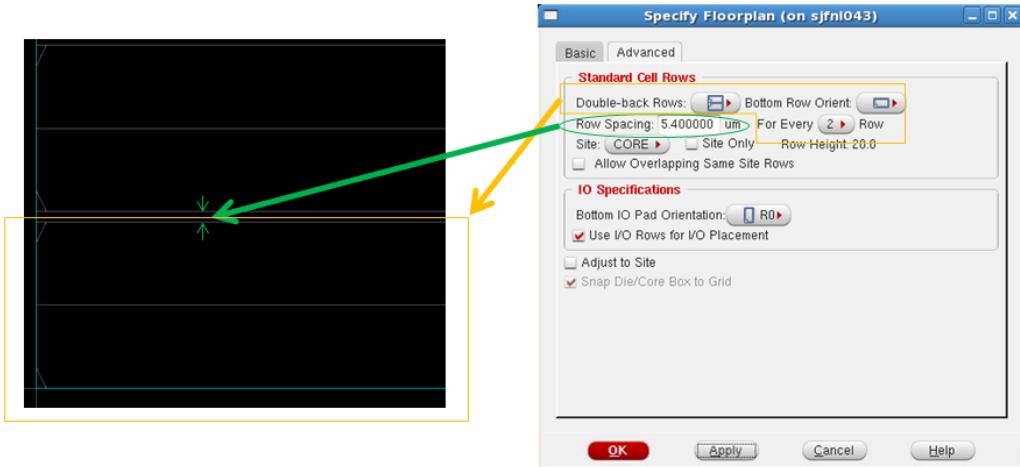
To configure the rows, use the [setFPlanRowSpacingAndType](#) command, the [createRow](#) command or the options from the *Standard Cells Rows* panel of the *Specify Floorplan* form.

The following row configurations are supported:



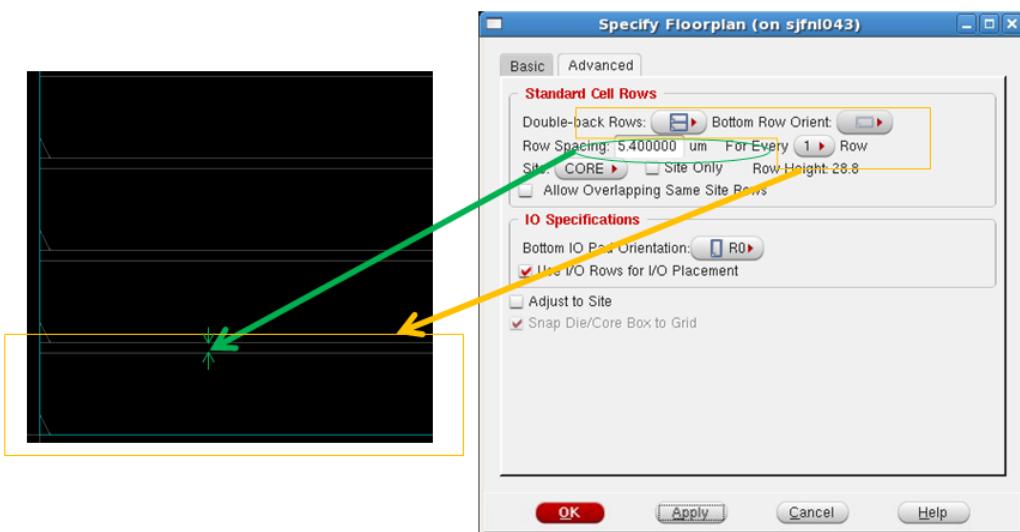
### Example 1: Bottom R0 and flip/about,

```
innovus 1> createRow -limitInCore -site CORE -spacing 5.4
```



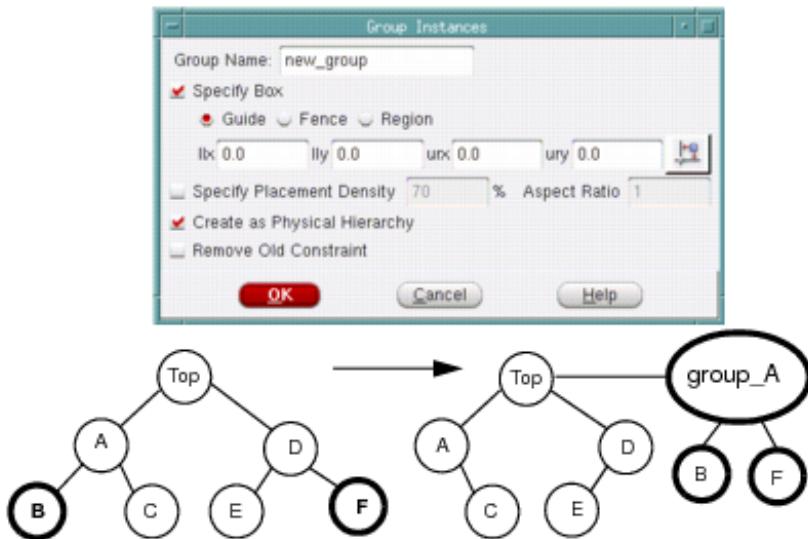
### Example 2: Bottom R0

```
innovus 1> createRow -limitInCore -noAbut -noFlip -site CORE -spacing 5.4
```



## Grouping Instances

The hierarchy of the new instance group is formed at the common point of the modules and submodules. The following example shows how the hierarchy is changed from the common point if submodules B and F are added to a new group called group\_A.



To delete an instance from an instance group, complete the following steps:

1. Choose *Tools - Design Browser*.
2. In the Design Browser, click on and highlight the module or submodule guide(s) to be deleted from the instance group.
3. Click the *Delete Group/Group Member* icon.

To add an instance to an existing group name, complete the following steps:

1. Click on and highlight the module or submodule guide(s) to be added to an instance group.
2. Choose the *Floorplan - Instance Group* submenu to select the group name.

To save the instance group back to the netlist, use the *Generate Regrouped Netlist* form (*Floorplan - Generate Regrouped Netlist*).

## Defining the Bounding Box

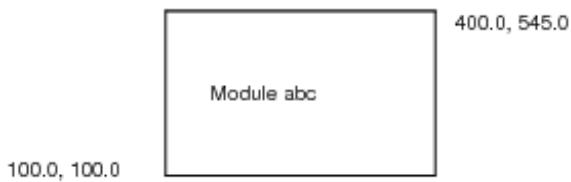
During floorplanning, you can use the `setObjFPlanBox` command to define a bounding box of a specified object, and the `setObjFPlanBoxList` command to define rectilinear shape of an object, which is comprised of two or more boxes.

This section provides graphical information to illustrate some of the command examples in the [Floorplan Commands](#) chapter of the *Innovus Text Command Reference*.

## setObjFPlanBox

The following command specifies a bounding box for Module `abc` at a lower left x coordinate of 100.0, a lower left y coordinate of 100.0, and upper right x coordinate of 400.0, and an upper right y coordinate of 545.0:

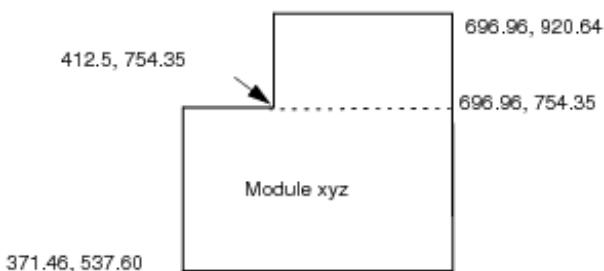
```
setObjFPlanBox Module abc 100.0 100.0 400.0 545.0
```



## setObjFPlanBoxList

The following command defines a rectilinear boundary for Module `xyz`. The rectilinear boundary is made up of two bounding boxes: (371.46, 537.60) (696.96, 754.35), and (412.5, 754.32) (696.96, 920.64):

```
setObjFPlanBoxList Module xyz 371.46 537.60 696.96 754.35 412.5 754.35 696.96 920.64
```



## Adding Logical Hierarchy Without Creating Additional Hierarchy

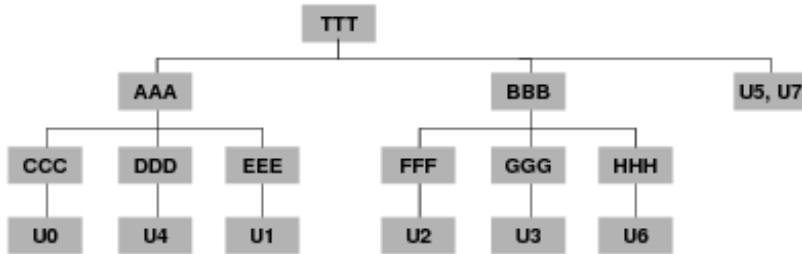
The Innovus software enables you to add logical hierarchy without creating additional hierarchy. For example:

```
createInstGroup /TTT -isPhyHier  
addInstToInstGroup /TTT U5
```

```
addInstToInstGroup /TTT U7
runRcNetlistRestruct
```

**Note:** The leading slash character (/) in /TTT is required for the software to create a temporary group named /TTT.

After restructuring, the result looks like this:



For more information, see the [runRcNetlistRestruct](#) command in the *Innovus Text Command Reference*.

## Logical Hierarchy Manipulation

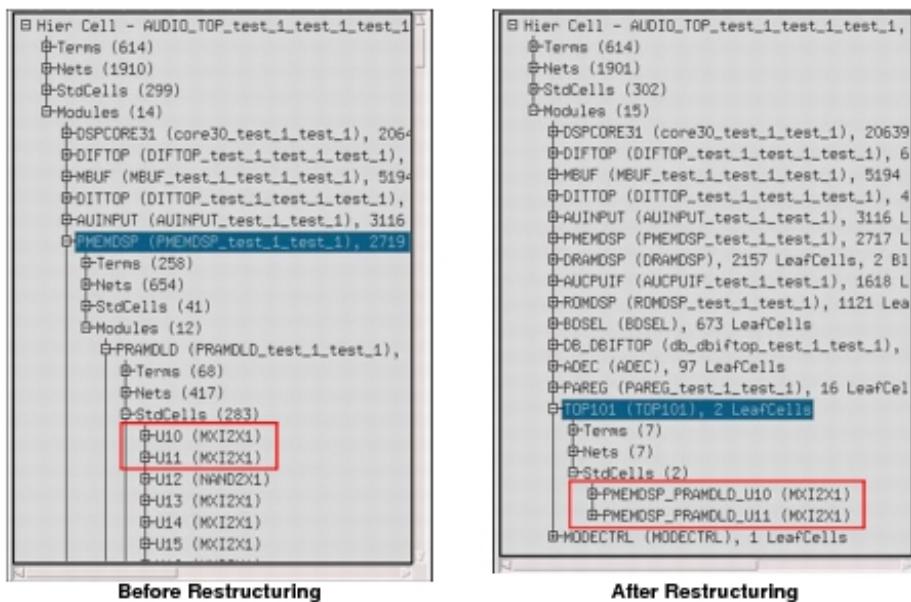
In addition to Adding Logical Hierarchy Without Creating Additional Hierarchy, you can also manipulate the logical hierarchy as follows:

- [Moving Instances to a New Top Module](#)
- [Moving Instances to an Existing Module](#)
- [Moving Instances to the Top Root Level](#)

## Moving Instances to a New Top Module

To move an instance to a new top module named `TOP101`, you can do the following:

```
createInstGroup TOP101 -isPhyHier
addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U10
addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U11
runRcNetlistRestruct
```

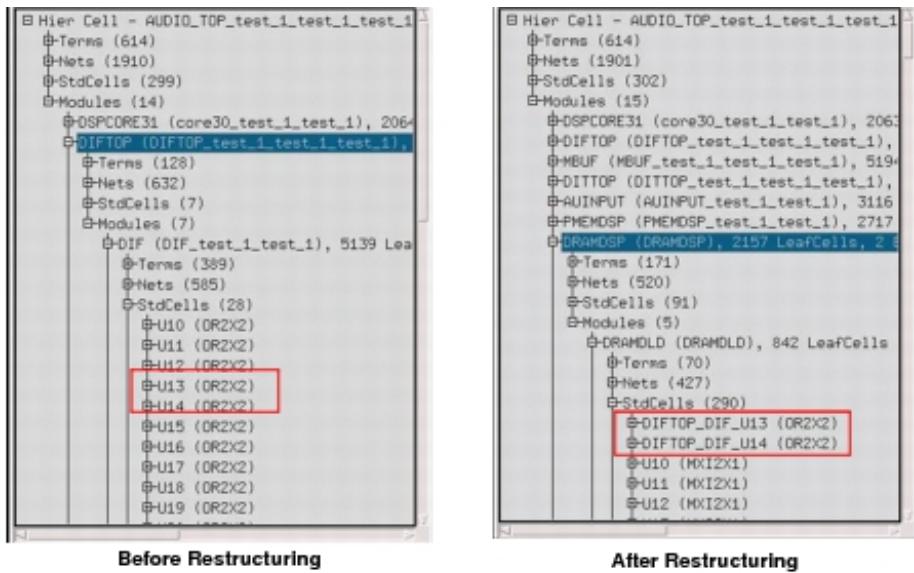


## Moving Instances to an Existing Module

To move an instance to an existing module named DRAMDSP/DRAMDLD, you can do the following:

```
createInstGroup /DRAMDSP/DRAMDLD -isPhyHier  
addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U13  
addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U14  
runRcNetlistRestruct
```

**Note:** The leading / (slash) is required for an existing module.

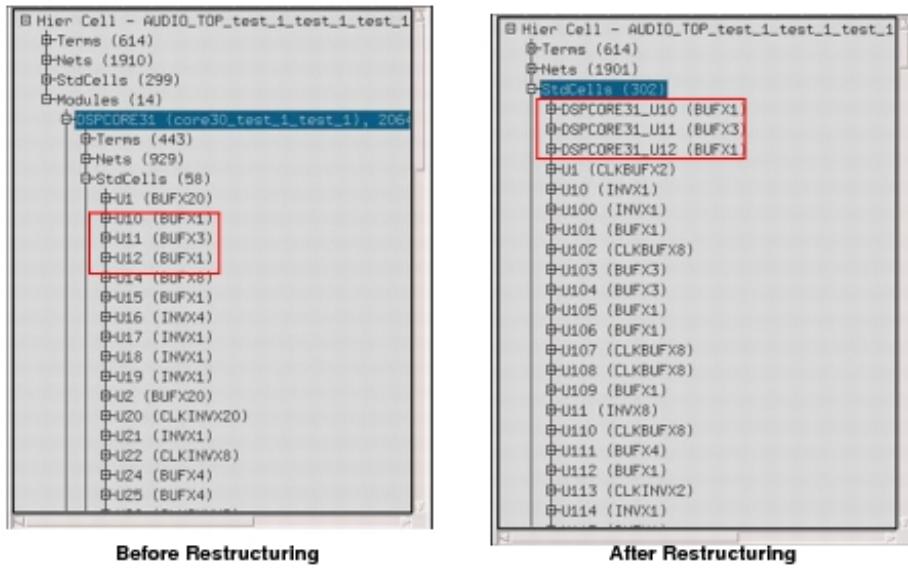


## Moving Instances to the Top Root Level

To move an instance to the top root level, you can do the following:

```
createInstGroup /AUDIO_TOP_test_1_test_1_test_1 -isPhyHier  
addInstToInstGroup /AUDIO_TOP_test_1_test_1_test_1 DSPCORE31/U10  
addInstToInstGroup /AUDIO_TOP_test_1_test_1_test_1 DSPCORE31/U11  
addInstToInstGroup /AUDIO_TOP_test_1_test_1_test_1 DSPCORE31/U12  
runRcNetlistRestruct
```

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



## Creating and Editing Rows

You can create and edit rows in different regions. The following table lists the commands that you can use to create and cut rows.

|                          |                                                                                                                                                                                                                                                                     |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>createRow</code>   | Creates rows for the specified site. The row boundary can be defined by core area or the area that you specify. This command supports the creation of overlapping rows. This command can create only horizontal rows. By default, the rows are flipped and abutted. |
| <code>cutRow</code>      | Cuts site rows that intersect with the specified area or object.                                                                                                                                                                                                    |
| <code>deleteRow</code>   | Deletes the specified row(s).                                                                                                                                                                                                                                       |
| <code>stretchRows</code> | Stretches selected rows.<br>For example, you can specify that the left edge of all selected rows should be aligned to the left-most edge among all selected rows.                                                                                                   |

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

You can also use the following forms available through the GUI:

- [Create Core Rows](#), available through *Floorplan - Row - Create Core Row*.

- [Cut Core Rows](#), available through *Floorplan - Row - Cut Core Row*.
- [Stretch Core Rows](#), available through *Floorplan - Row - Stretch Core Row*.

## Using Vertical Rows

**Note:** Support for vertical rows is a beta feature. Usage and support of this beta feature are subject to prior agreement with Cadence. Contact your Cadence representative if you have any questions.

In addition to horizontal rows, Innovus also supports vertical rows. You can import designs with vertical rows and output design data containing the layout of vertical rows. Vertical rows appear vertically in the display. You can select and query vertical rows and delete them with the delete key. The commands that snap rows to row grid also support vertical rows. These commands are [snapFPlan](#), [relativeFPlan](#), and the interactive [move](#) command.

Horizontal and vertical rows can co-exist, but at different layers of hierarchy. You cannot create horizontal and vertical rows together on the same level of hierarchy.

The global variable `fp_vertical_row` is used to specify whether the rows are horizontal or vertical.

The variable can be set to 0 (for horizontal rows) or 1 (for vertical rows) as follows:

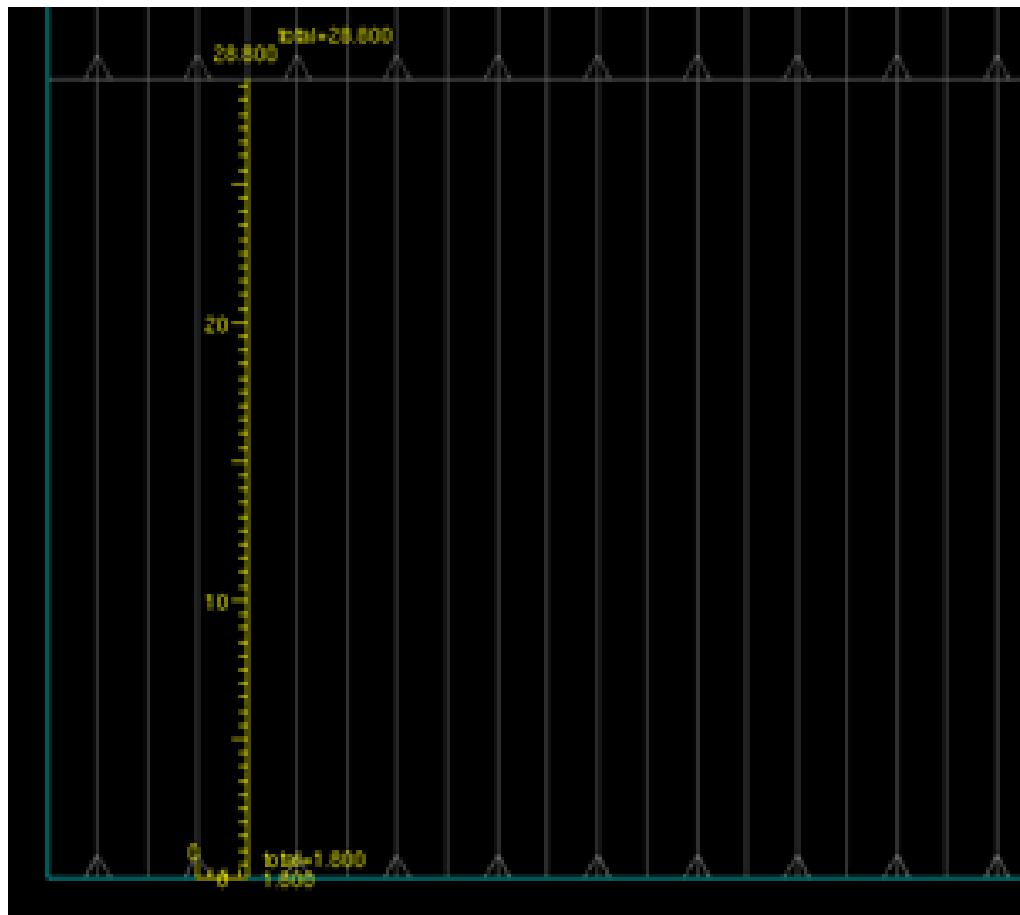
```
set fp_vertical_row {0|1}
```

The Innovus software generates vertical rows, provided the design data or the library meets the following prerequisites:

- Each SITE is stacked one above the other, when rows are created.
- MY and R0 row orientations are supported.
- The cells are stacked vertically, when they are placed and their power rails run vertically.
- The preferred direction of M1 is vertical.

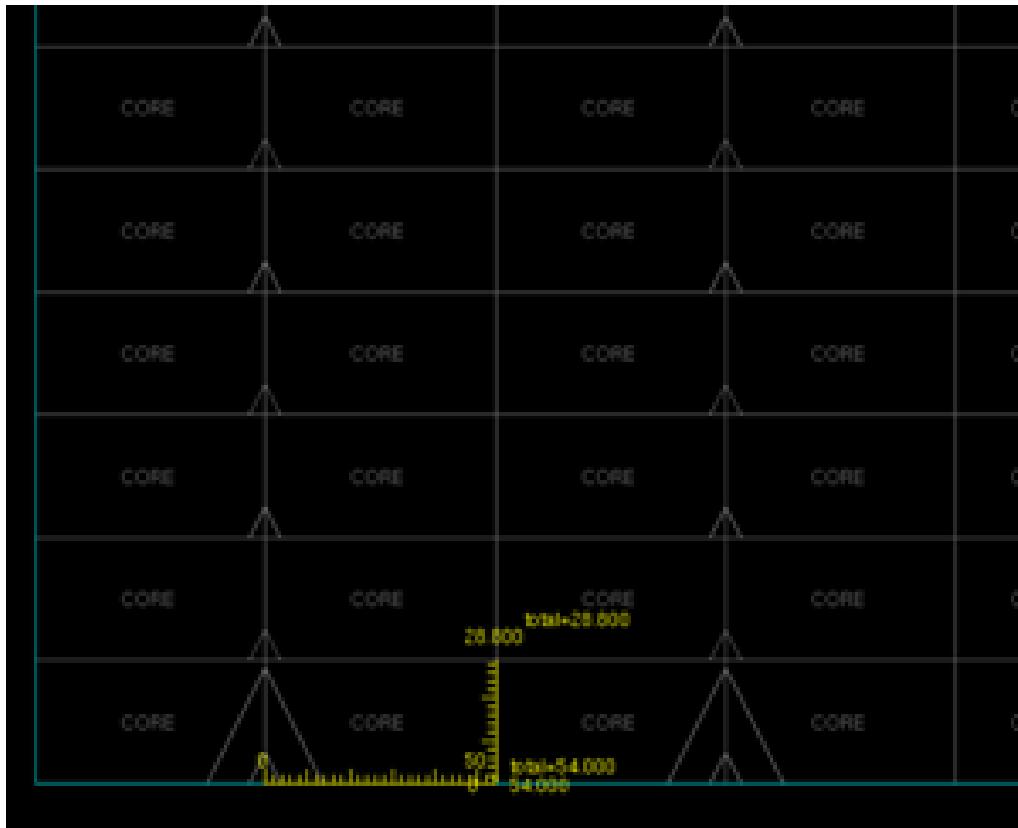
**Example 1:**

```
SITE CORE  
CLASS CORE;  
SIZE 1.800 BY 28.800;  
END CORE
```



## Example 2:

```
SITE CORE
CLASS CORE;
SIZE 54.000 BY 28.800;
END CORE
```



## Limitations

The following limitations apply to vertical rows:

- Vertical rows cannot be created inside power domains.
- Non-integer and multiple integer vertical rows are not supported.
- Vertical rows cannot be created or edited directly. That is, the `createRow`, `cutRow`, and `stretchRows` commands are not supported for vertical rows.

**Note:** You can, however, select rows to query, and delete rows with the delete key or

`deleteRow` command.

## Using Multiple-height Rows

In many cases, designs contain cells with different standard cell heights. For example, a design might utilize multiple standard cell libraries-possibly from different foundries or library vendors-which might have different standard cell heights.

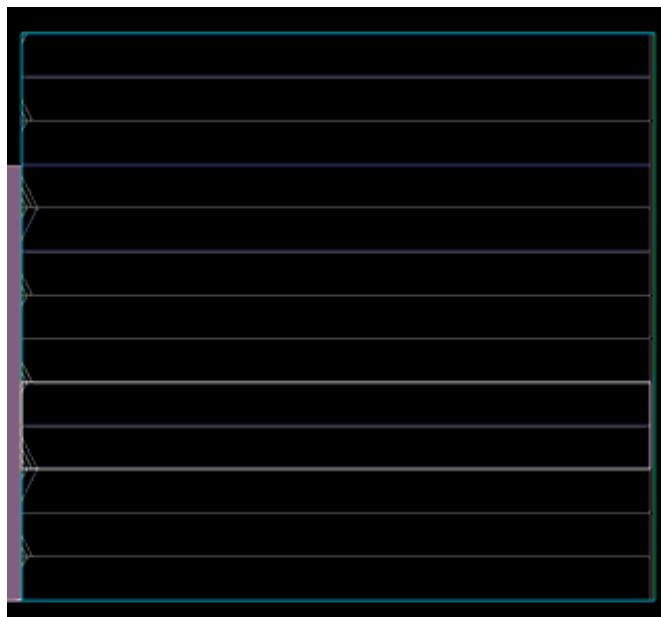
Standard cell designers create multiple-height standard cells for improving performance. Also, in a design with multiple power domains, standard cells with different voltages will probably have different footprints and different heights.

The Innovus software supports multiple-height standard cells by supporting:

- A combination of integer multiple-height rows
- A combination of non-integer multiple-height rows

## Using Integer Multiple-height Rows

The Innovus software automatically generates integer multiple-height rows overlapping the single-height core rows provided the design data or the library meets the following prerequisites:



- The LEF file contains integer multiple-height SITE definitions and MACROS that use the SITE.
- The netlist includes at least one instantiation of such an integer multiple-height cell.

After you import the design or specify the floorplan, the core area is automatically populated with default rows and multiple-height rows are automatically generated.

Here is an extract from a sample LEF file that contains integer multiple-height SITE definitions and a MACRO that uses a SITE:

```
SITE coreSite
  SYMMETRY X Y;
  CLASS CORE;
  SIZE 0.660 BY 5.040;
END coreSite

SITE doubleHeightSite
  SYMMETRY X Y;
  CLASS CORE;
  SIZE 0.660 BY 10.080;
END doubleHeightSite

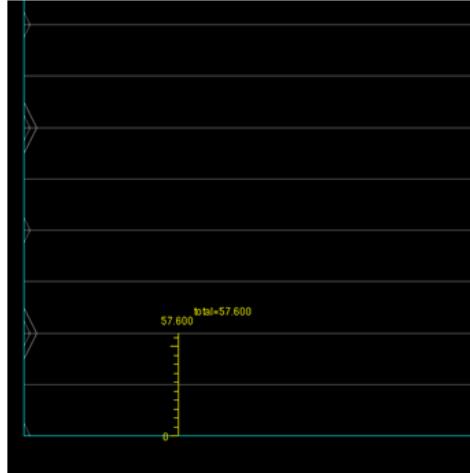
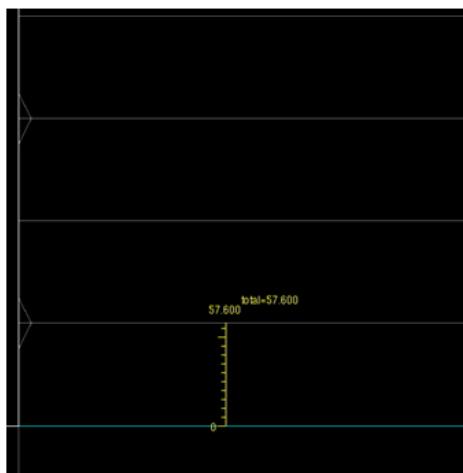
MACRO DFFX64
  CLASS CORE;
  FOREIGN DFFX64 0 0;
  ORIGIN 0 0;
  SIZE 21.12 BY 10.080;
  SYMMETRY X Y;
  SITE doubleHeightSite;
...
END DFFX64
```

When you create integer multiple-height rows, the rows are automatically aligned with the single-height row. You cannot create unaligned integer multiple-height rows.

For information on creating and editing rows, see [Creating and Editing Rows](#)

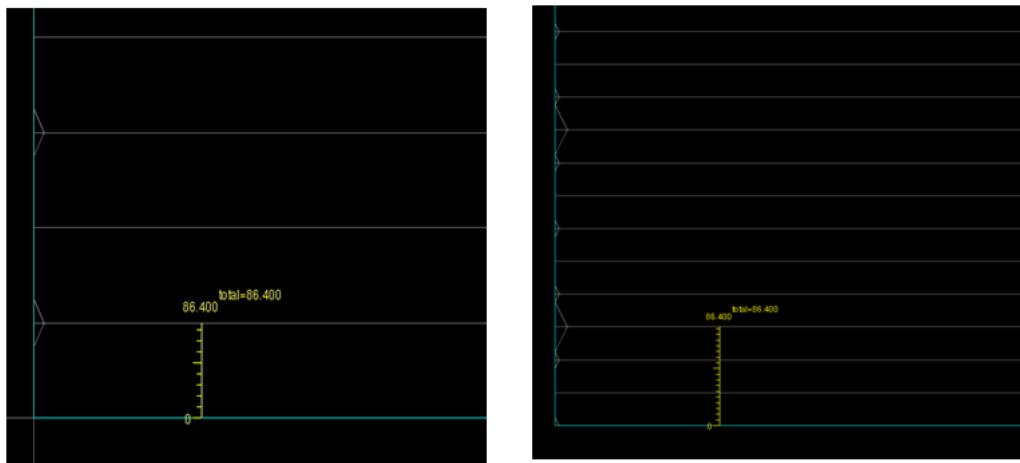
The following left figure illustrates a double-height row and the right figure illustrates a double-height row flipped to align with the orientation of the single row.

```
SITE CORE
  CLASS CORE;
  SIZE 1.800 BY 28.800;
END CORE
SITE CORE_double
  CLASS CORE;
  SIZE 1.800 BY 57.600;
END CORE_double
```



The following left figure illustrates a triple-height row and the right figure illustrates a triple-height row flipped to align with the orientation of the single row.

```
SITE CORE
  CLASS CORE;
  SIZE 1.800 BY 28.800;
END CORE
SITE CORE_triple
  CLASS CORE;
  SIZE 1.800 BY 86.4;
END CORE_triple
```



## Using Non-Integer Multiple-height Rows

You can also use non-integer multiple-height (NIMH) rows in your designs.

While creating NIMH rows, ensure the following:

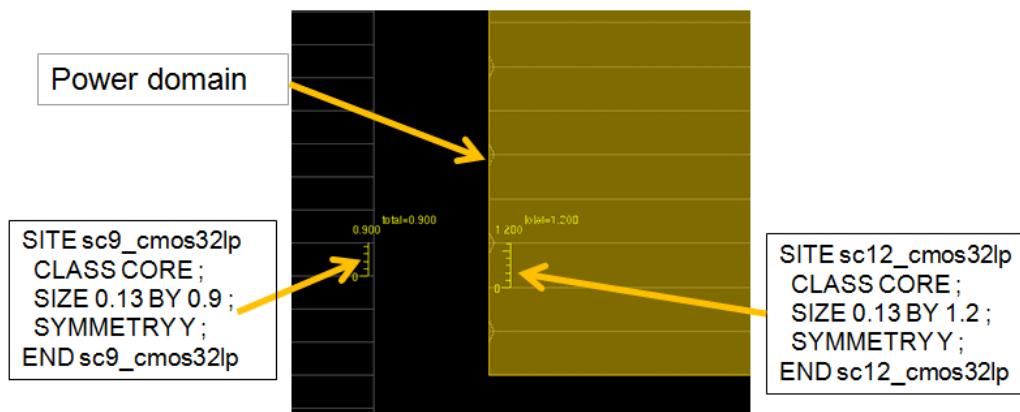
- NIMH rows must be created *only* in those areas that have power domains associated with them.
- Any newly created NIMH rows in an area must be an integer multiple of any existing rows in the area.

Each hierarchical instance to be declared as a power domain can only have one type of NIMH standard cells. In other words, NIMH rows inside any particular power domain must have the same height. Multiple types of NIMH standard cells require multiple power domains to be created. For example, if you want to use standard cells with heights that are respectively 2.5, 3.25, and 4.1 times

the height of a standard single-height cell, you should create three power domains, with each power domain containing one type of NIMH row.

When you create a power domain, Innovus automatically detects the site that is common to all the cells and creates the rows inside the power domain.

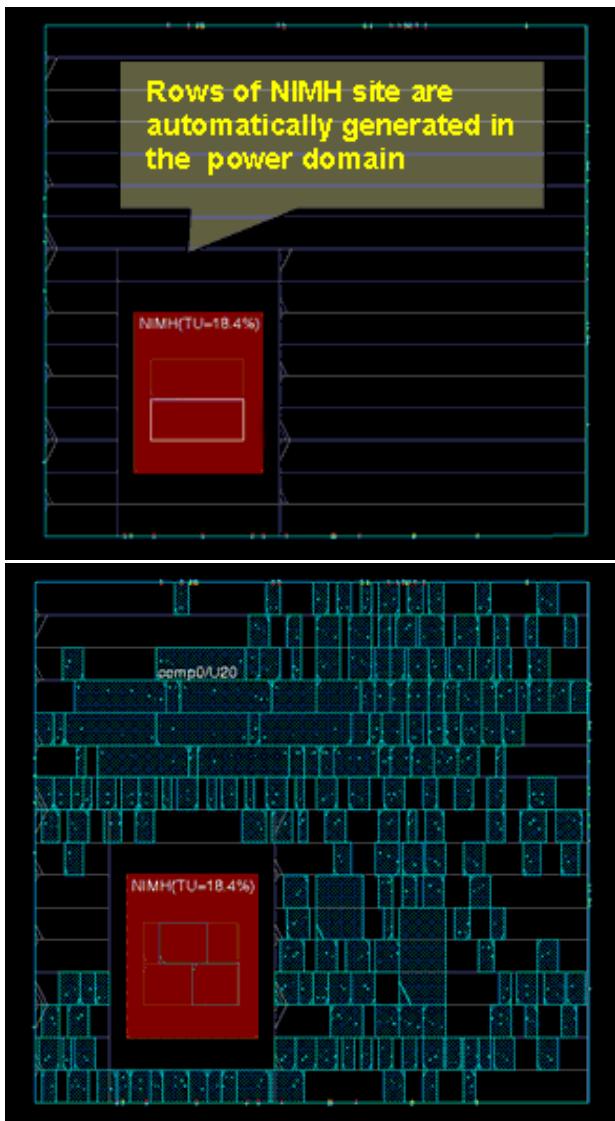
The following diagram illustrates how rows are automatically generated within the power domain for standard cells of the hierarchical instance.



You can use the `fpHonorDefaultPowerDomainSite` global variable to control which tech site is used for creating rows:

- The default value of this variable is 0. With the default value, Innovus uses the shortest tech site in the design as the default tech site. All other tech sites must be compatible with this default tech site.
- If the `fpHonorDefaultPowerDomainSite` value is set to 1, Innovus uses the default tech site of the default power domain as the whole design's default tech site. In other words, rows are created based on the default power domain's site. This setting is useful if you want to treat the default power domain as the 'whole area outside other power domains' in an MSV design.

The following diagram illustrates how the standard cells of the hierarchical instance are all placed on the row inside the power domain.



The modules and/or instance groups can be moved outside the core boundary but within the die. As new rows are not created automatically in this area, you can use the `createRow` command to create new rows. Manual editing of rows might not be preserved by `floorPlan`, `resizeFP`, and `initCoreRow` commands.

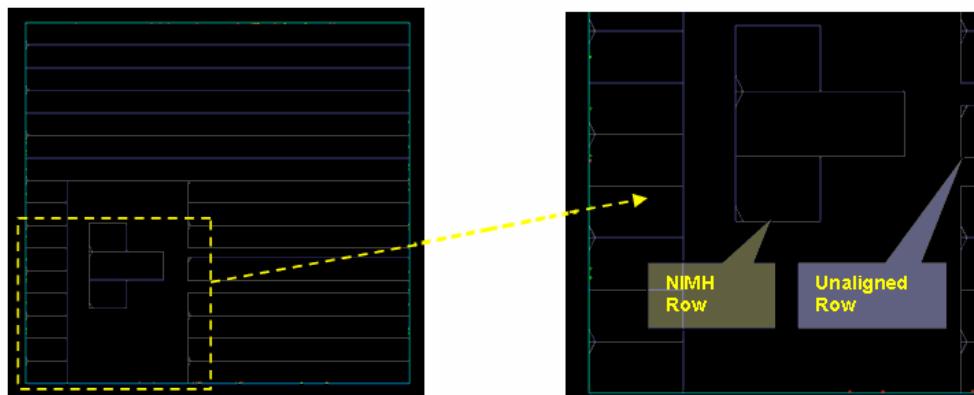
Innovus displays a warning message if you try to move a power domain outside the core boundary, and snaps the power domain inside the core.

## Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows

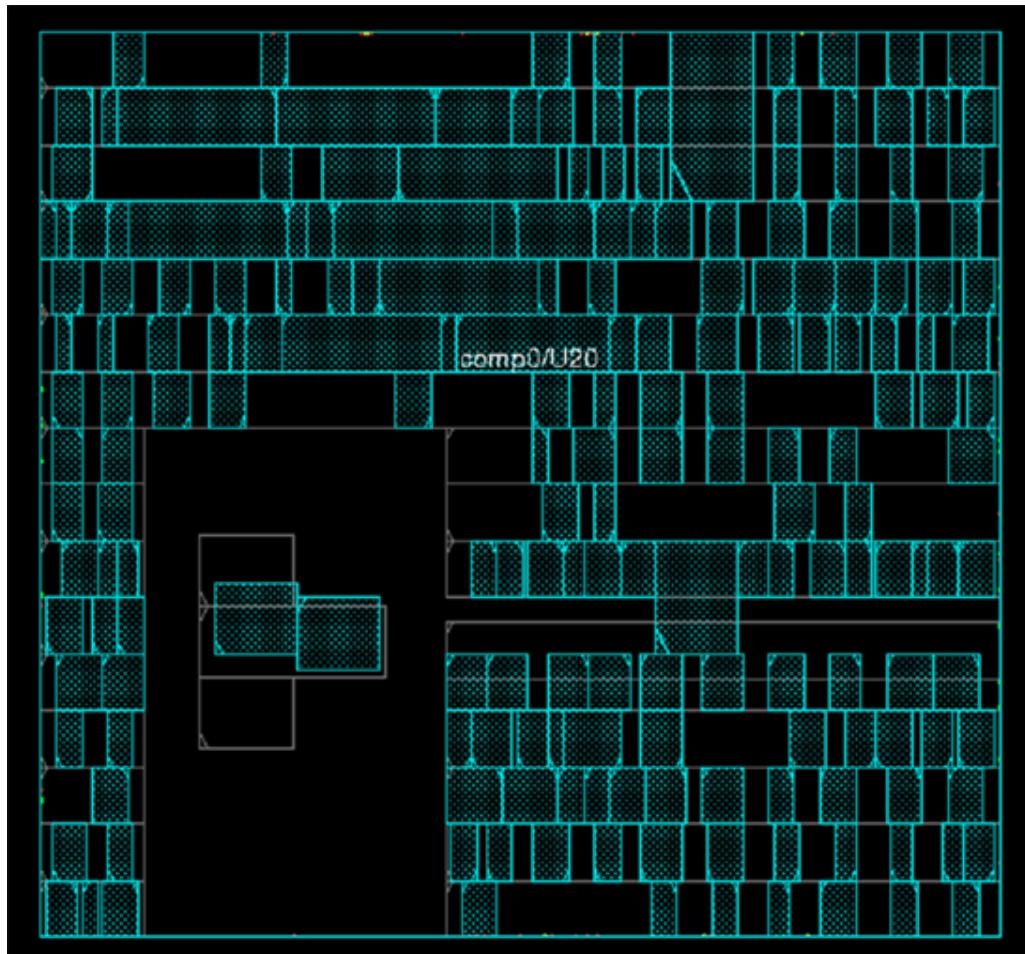
In case of integer multiple-height rows, as long as the rows are overlapping the single-height rows, standard cells will by default be legally placed on their corresponding site or row definition.

However, if you import a user-defined plan through a DEF file that contains NIMH rows and unaligned rows, you need to define power domain(s) for each of these disjoint special row style. Otherwise, these rows might not be correctly placed.

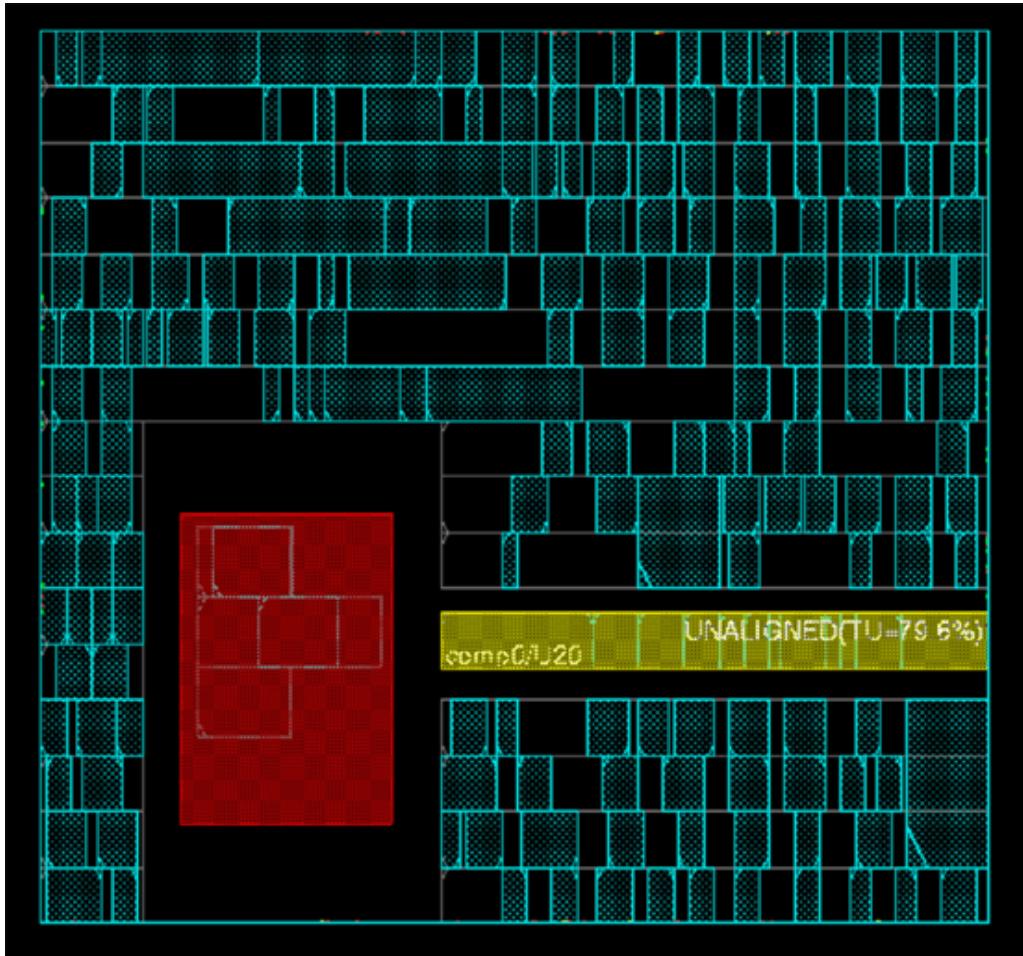
The following figure illustrates a user-defined floorplan brought in through a DEF file.



The following figure illustrates how placement without power domain association results in illegal placement.



The following figure illustrates how placement with the correct power domain association results in legal placement.



By default, when a power domain is moved or (re)located in the main core row area, rows are initialized.

To keep the rows brought in by the DEF file, you should pre-place and pre-size the power domains that cover the NIMH rows and the unaligned rows.

## Merging Hierarchical Floorplans from Partitions

While flattening partitions with the `flattenPartition` command, you can bring back row information, including NIMH rows and unaligned rows, from an existing floorplan. You can then run placement and routing to further improve the design performance.

Use the `flattenPartition` command can preserve the row information by default. The `flattenPartition` command also supports rotated partitions.

Power domains are automatically created and associated with each of the partition that have NIMH or unaligned rows. These automatically created power domain have the following characteristics:

- The `minGap` value is the same as the placement halo defined for the partition at the full-chip level.
- The timing library is the same as that specified at the full-chip level.
- The global net connection is the same as that for the full-chip level, which is the same as the partition floorplan global net connection.
- The value of the `RouteSearchExt` field is set to the default value of 0.0.
- The core-to-edge distances are not preserved as an attribute of the power domain, but are preserved in the merged floorplan.
- Row parameters are not translated or detected.
- The power domain is set to `alwaysOn`.

## Use Model

The recommended use model for bringing back non-integer multiple-height rows or unaligned rows is as follows:

### Top-down flow

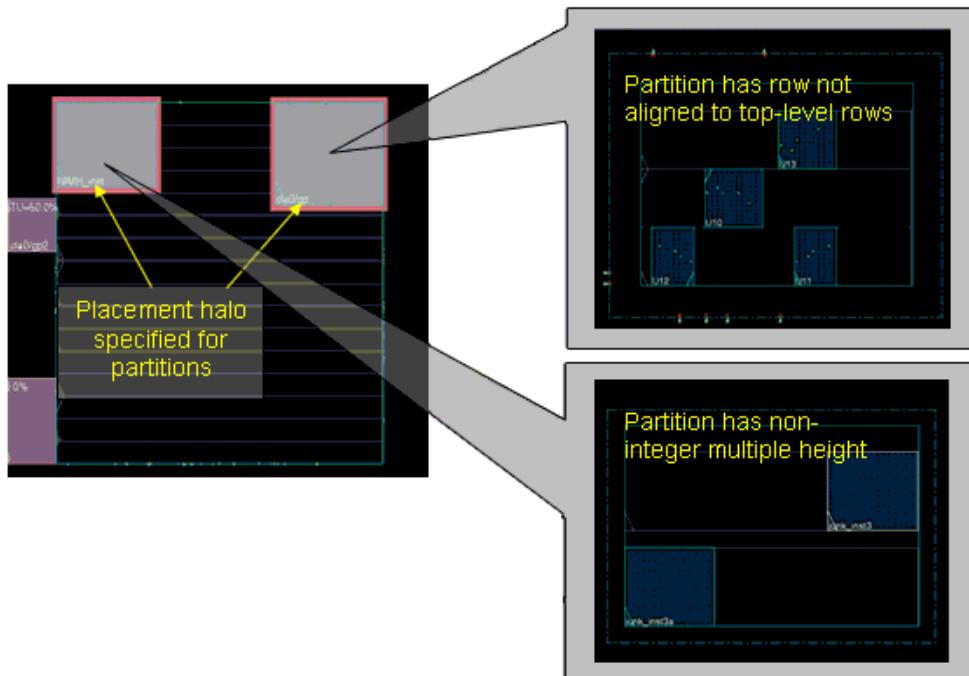
1. Create power domains at full-chip level design.
2. Specify the same hierarchical instance for power domain as defined for the partition.

### Bottom-up flow

1. Create power domains at top-level design.
2. Create one PD for each of the partitions.
3. Assign instance blocks as a member of the created power domain.

For more information on the `flattenPartition` command, see the *Innovus Text Command Reference*.

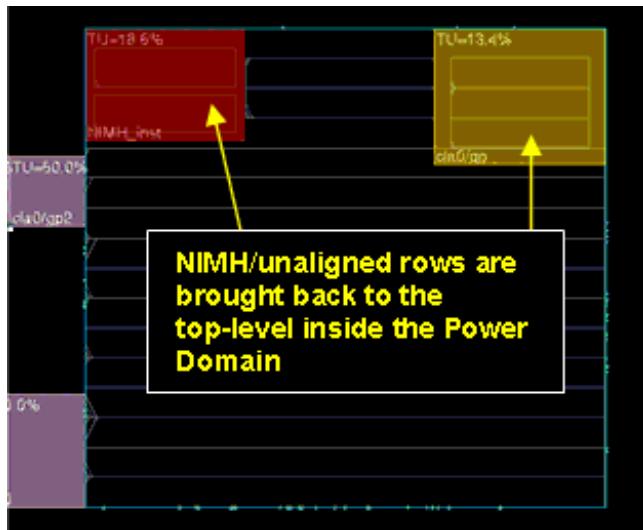
The following figure shows a full-chip view with the partition halos specified.



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



The following figure illustrates the result with you use the `flattenPartition` command with the `-bringbackRow` parameter. In this case, the NIMH rows and the unaligned rows are brought back to the top-level inside the power domain.



## Performing I/O Row Based Pad Placement

In many cases, designs contain multiple-height I/O pads or asymmetric I/O rings, for example, a design might have a single I/O ring on one side and double rings or no rings on the other side, or no rings on part of a certain side. For such designs, the Innovus software enables you to create, edit, save, and restore I/O rows and perform pad placement based on the I/O rows.

You can create I/O rows anywhere in the die – within the core or in the periphery and use the I/O row flow for both, pad and area I/Os.

## Prerequisites

1. LEF technology file should contain I/O SITE definition.

Before you begin the I/O row flow in Innovus, you must first define I/O SITE for each type of I/O cell in the LEF I/O macro (LEF technology file).

2. Each I/O cell LEF must have correct CLASS and SITE type specified.

Consider the following examples that define LEF I/O SITE and CLASS PAD MACRO in the I/O assignment file, each I/O CLASS PAD macro is referenced with the I/O SITE:

**Example 1:**

The I/O SITE IOPFC is referenced from the I/O CLASS PAD MACRO pn1\_qdr\_vp:

```
SITE IOPFCSYMMETRY Y;  
    CLASS PAD ;  
    SIZE 0.1 BY 321.94 ;  
END IOPFC  
  
MACRO pn1_qdr_vp CLASS PAD ;  
    FOREIGN pn1_qdr_vp ;  
    ORIGIN 0.000 0.000 ;  
    SIZE 35.000 BY 321.940 ;  
    SYMMETRY X Y R90 ;  
    SITE IOPFC ;  
    ...  
END pn1_qdr_vp
```

**Example 2:**

The corner site, IOPFCCRN, is referenced from the CLASS PAD MACRO pn1\_qdr\_iocrnr:

```
SITE IOPFCCRN  
    SYMMETRY Y ;  
    CLASS PAD ;  
    SIZE 321.94 BY 321.94 ;  
END IOPFCCRN  
  
MACRO pn1_qdr_iocrnr  
    CLASS PAD ;  
    FOREIGN pn1_qdr_iocrnr ;  
    ORIGIN 0.000 0.000 ;  
    SIZE 32.940 BY 321.940 ;  
    SYMMETRY X Y R90 ;
```

```
SITE IOPFCCRNR ;  
...  
END pnl_qdr_iocrnr
```

For more information, see "[Generating the I/O Assignment File](#)" in [Data Preparation](#) chapter of the *Innovus User Guide*.

3. Each I/O cell LEF must have correct CLASS and SITE type specified. If the design contains multiple I/O SITES, the gap between the core boundary and the die boundary must be greater than the biggest I/O SITE; Otherwise, the Innovus software issues a warning and no I/O rows are created. Innovus does not automatically expand the core or die boundary to accommodate all the I/O pads.

## Enabling the I/O Row Flow in Innovus

To start using the I/O row flow in Innovus, you must enable the I/O row flow by doing one of the following:

- Specify `setUserDataValue conf_use_io_row_flow 1` in the Innovus global variable file.
- Select *Use I/O Row for I/O Placement* check box in the *Floorplan - Specify Floorplan - Advanced* GUI form.
- Set the [`setIoFlowFlag`](#) command to 1.

To stop using the I/O row flow in Innovus, you must disable it by setting the [`setIoFlowFlag`](#) command to 0.

**Note:** By default, the I/O row flow is Off. Use the [`getIoFlowFlag`](#) command to check the current flow. A value of "0" means traditional I/O flow where as a value of 1 means I/O row flow.

## Use Models

### Starting a new design

1. Import the design in Innovus
2. Set the I/O row flow by selecting the *Use I/O Row for I/O Placement* check box in the [Specify Floorplan - Advanced](#) form.
3. By default, one I/O row is created on each side of the chip, between the core boundary and the die boundary. If the design has multiple I/O sites in the library, then rows are created for each side based on the number of I/O sites used in the design.  
By default, the I/O pads are placed randomly on the I/O rows.
4. In the resulting design, you can create I/O rows using the [Create I/O Row](#) form.
5. Edit (move/stretch/rotate/flip) the I/O rows using [Floorplan - Row](#) form or using the text commands.

The text commands that can be used for creating and editing I/O rows are described in the following table.

| Commands                        | Usage                                                                                                                                    |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <code>createIoRow</code>        | Creates an I/O row.                                                                                                                      |
| <code>flipOrRotateObject</code> | Flips or rotates the selected objects.                                                                                                   |
| <code>stretchRows</code>        | Stretches the selected I/O rows.                                                                                                         |
| <code>deleteRow</code>          | Deletes the selected I/O row.<br>You can also delete the row by selecting the row and pressing the <code>Del</code> key on the keyboard. |
| <code>setIoRowMargin</code>     | Sets the distance from the die boundary edge to the I/O row starting edge location.<br>You can use this command for multiple I/O rows.   |
| <code>snapFPlanIO</code>        | Snaps the I/O pads onto the correct side of the I/O rows if the pads are not already on the rows.                                        |
| <code>spaceIoInst</code>        | Spaces the selected I/O pads on the I/O rows, horizontally or vertically by a specified distance value.                                  |

Optionally, you can specify the I/O row constraints in the I/O assignment file. For more information, see "Generating the I/O Assignment File" in the [Data Preparation](#) chapter of the *Innovus User Guide*.

**Note:** To add I/O filler cells to the new I/O rows, use the `addIoRowFiller` command. You can delete the filler cells using `deleteIoRowFiller` command.

After designing the rows, save the design in a floorplan file (\*.fp) using the `saveDesign` or `saveFPlan` command.

## Reading an old design

If you already have a design with placed pads, created using an earlier version of the software (v6.2 and above) and you want the design to be read into current version of Innovus to use the new I/O row flow:

1. Restore the design with placed pads, using the `restoreDesign` command or load the floorplan information from the file, using the *Load FPlan File* form or the `loadFPlan` command.  
     Optionally, load the I/O constraint file using the `loadIoFile` command.
2. Turn on the I/O row flow by setting the I/O flow flag ( `setIoFlowFlag` ) to 1.
3. Create the initial I/O rows using the `createIoRow -deriveByCells` command. This command creates I/O rows based on the existing pad placement.  
    Once you have rows and pads placed in the design, you can continue to create more rows or edit the rows.  
    Use the `changeIoConstraints` command to change the constraints of the I/O row read from the I/O constraints file.  
    You can also use the [Attribute Editor](#) to change the I/O pad location or pad orientation from the GUI.
4. After editing the I/O rows and the I/O row constraints, run the `snapFPlanIO` command to snap the I/O pads and area I/O's onto the legal sites/rows.
5. Save the design in a floorplan file (\*.fp) using the `saveDesign` or `saveFPlan` command.

## Resizing Rectilinear block-level floorplan

Given an initial rectilinear block-level floorplan, Innovus automatically resizes its bounding box by enlarging or shrinking the edges of the box proportionally in the X and Y directions, ensuring that the specified target utilization is met.

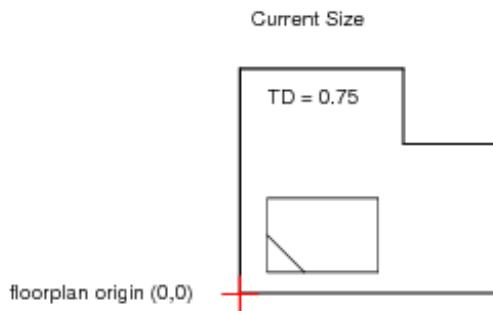
During this process, to retain the original shape of the rectilinear block-level floorplan, you must specify the `-keepShape` parameter in the `floorPlan` command. Consider analog designs where you have digital blocks that need to be fit into the analog chip and the shape of the block is already pre-defined. In such mixed-signal designs, you can retain the block shape during resizing, and also meet the specified target utilization value by shrinking or expanding the floorplan using the `floorplan -keepShape util` value, where `util` is the target core utilization value.

## Use Models

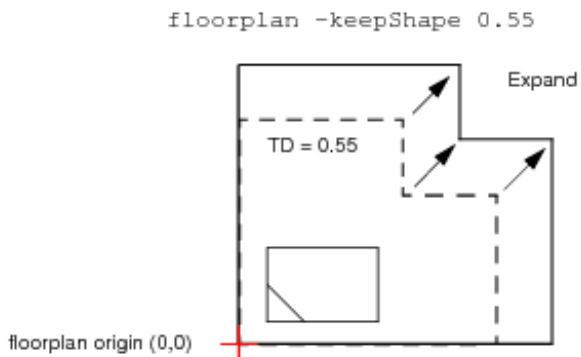
1. Import a DEF file or a floorplan file (\*.fp) that contains a rectilinear block-level floorplan boundary or you cut one corner of a rectangular block. This results in the core utilization missing the target value.
2. Run the `floorplan -keepShape util` command to automatically shrink or expand the block-level floorplan boundary, proportionally, in the X and Y directions, trying to meet the specified target core utilization.
3. Use the `checkFPlan -reportUtil` command to report the final core utilization.

### Example:

Consider the following example of a rectilinear block-level floorplan whose current target core utilization value is 0.75.

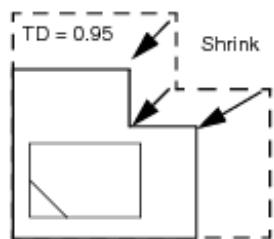


To meet a target core utilization of 0.55, expand the rectilinear block-level floorplan.



To meet a target core utilization of 0.95, shrink the rectilinear block-level floorplan.

floorplan -keepShape 0.95



In both the cases, the shape of the block-level floorplan is retained, and the required target core utilization is met.

For I/O pins, prior to resize, Innovus saves the I/O file sequence internally and loads the file back after resize. The side and sequence of the I/O pin remains the same as in the old block, but the pins get distributed evenly. To redistribute the I/O pins, you must edit the pins manually in the resize block.

## Assumptions

The automatic resizing of rectilinear block-level floorplan is based on the following assumptions:

1. The rectilinear block-level floorplan are L-shaped.
2. The floorplan origin, (0,0) remains unchanged during the resize.
3. The instances inside the block move proportionally or stay fixed during the resize.

## Results

The results of automatic resizing of rectilinear block-level floorplan are as follows:

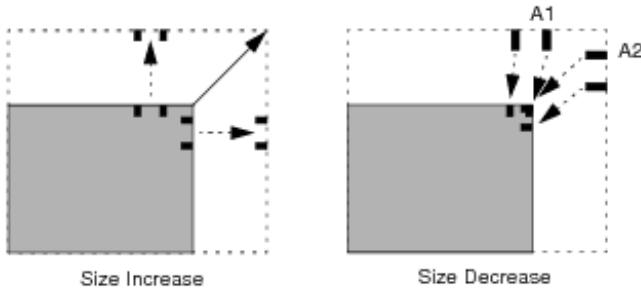
1. The shape of the block-level floorplan is preserved during the resize.
2. Pre-placed macros are adjusted to the new size as much as possible.
3. Pre-routed wires are removed after the resize.
4. The core rows and block pins are automatically adjusted after the resize.

## Editing Pins

This section describes how you can move and manipulate pins in your design. For information on blackbox and partition pins, see the [Assigning Pins](#) section in the "Partitioning the Design" chapter of the *Innovus User Guide*.

## Pin Snapping on Resized Boundaries

As the boundary size increases, the pins maintain their exact horizontal and vertical coordinates, depending on the modified edge. As the boundary size decreases, the pin snap retains its relative position on the modified edge. This following figure illustrates this capability. For the size decreasing example, pins A1 and A2 are both snapped to the upper right corner.



**Note:** This feature is limited to rectangular edges.

## Moving Pins

To move a pins or a group pins, they should be at the same block and same side of the block. By default, all pins will move together relatively and the layer will be changed to the appropriate layer if the side was changed. For example, layer M2 is changed to M1 when moving pins from top to left.

Moving pins from top to bottom does not change the layer.

To move a selected pin or group of pins in the design display area from one edge to another edge (including rectilinear edges) on a module, complete the following steps:

**Note:** For pin groups, this will preserve the relative position between pins.

1. Click the *Move/Resize/Reshape* widget.
2. Select (left-click) the pin in the design display area.  
For a group of pins, press the `Shift` key to highlight each pin.
3. Left click on the pin(s) and move them to the new location.

**Note:** To zoom out on the design display area while dragging the pins, press the `Shift-Z` key combination.

You can use the `moveGroupPins` command for moving the pin(s) to the new location.

## Swapping Pins

You can swap pins using the `swapPins` command or the *Swap Pins* option in the design display area by completing the following steps:

1. Select two pins of the same block.
2. With the cursor over one of the selected pins, right-click the mouse to bring up the context menu.
3. Select *Swap Instances*.

## Using the Pin Editor

You can use the *Pin Editor* to display and edit pins and pin groups. To open the *Pin Editor*, choose *Edit - Pin Editor*.

For information in the fields and options, see *Pin Editor* in the "Edit Menu" chapter of the *Innovus Menu Reference*.

Here are the main features of the *Pin Editor*:

- Works for all type of pins such as partition pins, blackbox pins, and I/O pins
- When moving pins, associated pin geometry is moved or updated accordingly

- Can be used to move a single pin and/or a group of pins
- Supports pin editing by various criteria such as location, layer, side/edge, and so on
- Provides pin spreading capabilities. For more information, see [Using the Pin-Spreading Feature](#)
- Provides pin snapping capabilities such as to manufacturing grid, user grid, and layer track.
- Supports non-preferred routing layers for all supported snapping grids.
- Honors pin constraints at partition-level and pin-level, as well as constraints defined through the GUI.
- Supports pre-assigned pins.
- Supports rectilinear edges (multiple edges per side).

The `editPin` command provides the equivalent functionality of the *Pin Editor*.

The following sections describes some of the features that you can use with the *Pin Editor*.

## Using the Pin-Spreading Feature

The *Pin Editor* includes a utility to spread pins along the edges of a block. There are four different methods of spreading pins:

- Use a pin as the starting point (anchor) and provide a pin spacing distance.
- Use the center of a side or edge as the starting point and provide a pin spacing distance.
- Space the pins evenly along the side or edge, using the ends of the side or edge as the starting and ending points. The Pin Editor calculates the pin spacing distance.
- Space the pins evenly using explicit starting and ending points on the side or edge. The Pin Editor calculates the pin spacing distance.

## Basic Concepts for Pin Spreading

Two basic concepts underlie the pin-spreading functionality of the Pin Editor:

- Pin ordering affects the starting point for pin spreading.  
Use the Pin Editor's Group Bus, Reverse Order, or Reorder Pin List functions to specify the first pin in a group. The coordinates of the first pin in a group provide the starting point from which to spread pins.
- Pin spacing distances can be expressed in either positive or negative values:

Positive spacing values spread pins to the right along a horizontal block edge, or up along a vertical block edge.

Negative spacing values spread pins to the left along a horizontal block edge, or down along a vertical block edge.

**Note:** You cannot specify pin spacing distances with spacing methods that rely on the Pin Editor to determine the spacing.

## Pin Spreading Methods Supported by the Pin Editor

The following sections provide details on the four pin-spreading methods supported by the Pin Editor.

- [Using a Pin as the Starting Point for Spreading Pins](#)
- [Using the Center of a Side or Edge as the Starting Point for Spreading Pins](#)
- [Spacing Pins Evenly Along an Edge or Side](#)
- [Spacing Pins Evenly Using Explicit Starting and Ending Points](#)

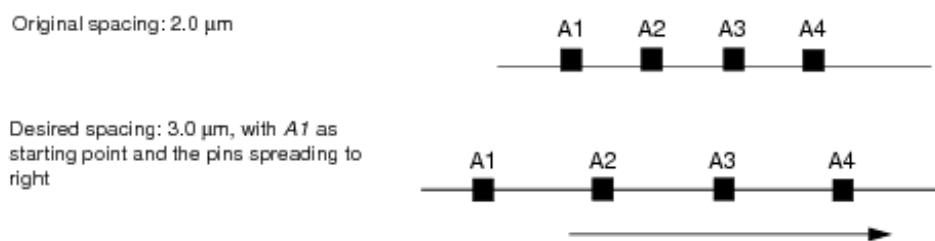
## Using a Pin as the Starting Point for Spreading Pins

For this method, you select a group of pins and sort them in the desired order. The first pin in the list serves as the starting point (anchor) for spreading the other pins in the group. You must also provide the pin spacing distance if you are spreading more than one pin.

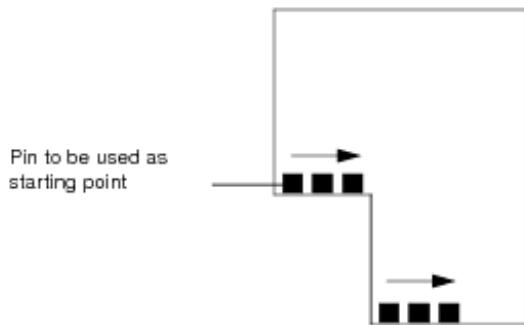
Assume that your design contains four pins (*A1*, *A2*, *A3*, and *A4*) that are currently spaced 2.0  $\mu\text{m}$  apart. You want to spread the pins to the right with 3.0  $\mu\text{m}$  spacing, using *A1* as the starting point. To do this you must

1. Sort the pins so that *A1* is the first pin in the list. The coordinates of *A1* appear in Starting X/Y.
2. Select *Spread - From Starting Point* on the *Pin Editor* form. Specify a positive spacing value: 3.0.

The following figure illustrates this situation:



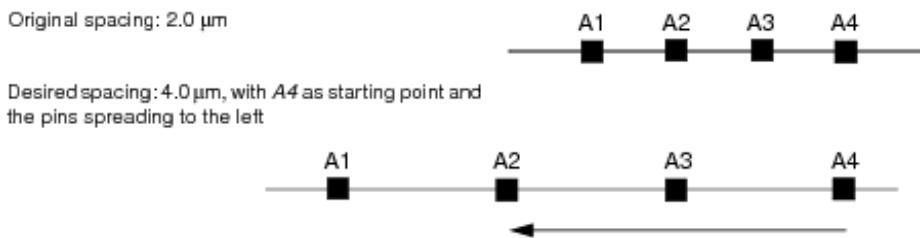
The following figure shows how pins are spread from a pin as starting point in case of rectilinear partitions for a side with multiple edges. In this case, the specified side is bottom, and the pins are therefore spread along the edges of the bottom side.



Now assume that you want to spread the pins to the left with  $4.0 \mu\text{m}$  spacing, using A4 as the starting point. To do this you must:

1. Sort the pins so that A4 is the first pin in the list. The coordinates of A4 appear in the *Starting* field of the *Pin Editor* form.
2. Specify a negative spacing value: -4.0.

The following figure illustrates this situation:



## Using the Center of a Side or Edge as the Starting Point for Spreading Pins

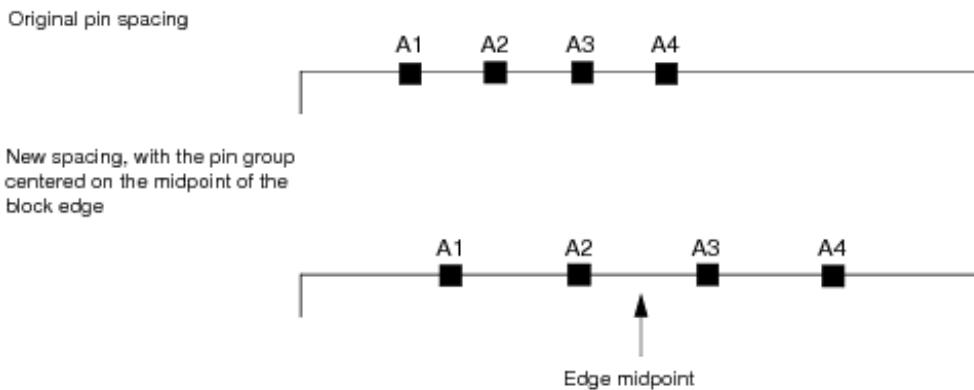
For this method, you select a group of pins and sort them in the desired order. You must also provide the pin spacing distance.

Assume that your design contains four pins (A1, A2, A3, and A4). You want to define new spacing and then group the pins so that the group is centered on the midpoint of the block edge. To do this you must

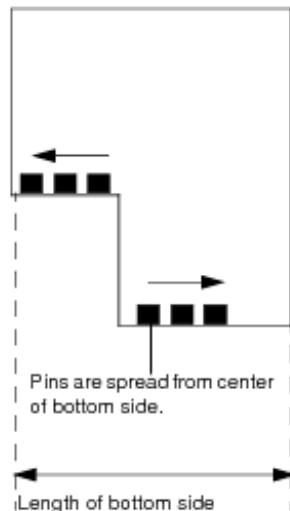
1. Sort the pins in the desired order (optional).

2. Select *Spread - From Center* on the *Pin Editor* form.
3. Specify a positive spacing value: 3.0.

The following figure illustrates this situation:



The following figure shows how pins are spread from a center point in case of rectilinear partitions where a side with multiple edges has been specified. In this case, the specified side is bottom, and the pins are therefore spread from the center of the bottom side.



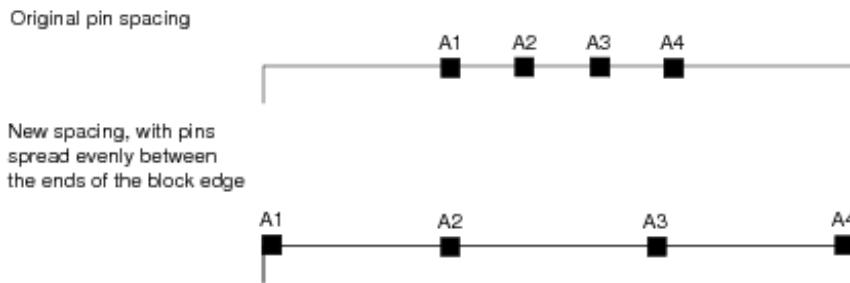
## Spacing Pins Evenly Along an Edge or Side

For this method, you select a group of pins and sort them in the desired order. You do not specify a pin spacing distance because the *Pin Editor* calculates the appropriate distance, based on the length of the block edge or side, and spaces the pins evenly along the block edge or side.

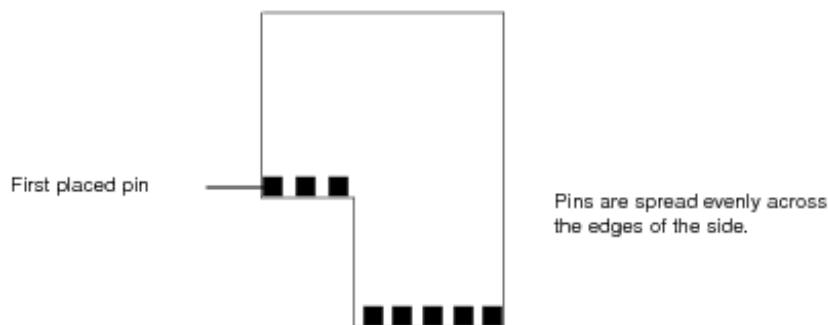
Assume that one edge of your design contains four pins (A1, A2, A3, and A4). You want to spread the pins evenly along the block edge. To do this you must

1. Sort the pins in the desired order (optional).
2. Select *Spread - Along Entire Edge* on the *Pin Editor* form.

The following figure illustrates this situation:



The following figure shows how pins are spread along a side in case of rectilinear partitions where a side with multiple edges has been specified. In this case, the specified side is bottom, and the pins are, therefore, spread along the edges of the bottom side.



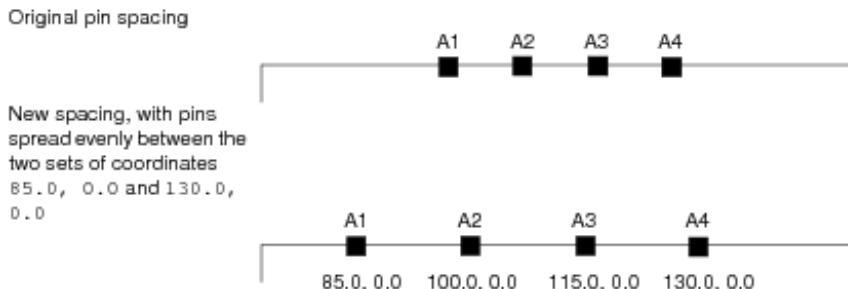
## Spacing Pins Evenly Using Explicit Starting and Ending Points

For this method, you select a group of pins and sort them in the desired order. You do not specify a pin spacing distance because the *Pin Editor* calculates the appropriate distance, based on the specified starting and ending points, and spaces the pins evenly along the edge or side.

Assume that one edge of your design contains four pins (A1, A2, A3, and A4). You want to spread the pins evenly along the block edge between two sets of coordinates. To do this you must

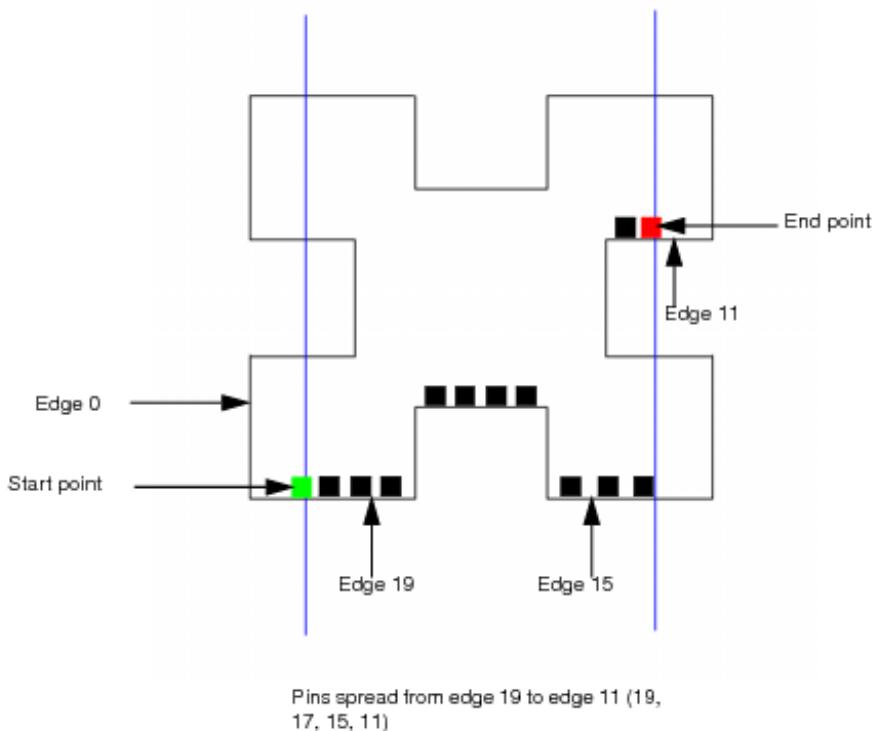
1. Sort the pins in the desired order (optional).
2. Select *Spread - Between Points* on the *Pin Editor* form .
3. Revise the starting and ending coordinates as desired.

The following figure illustrates this situation:

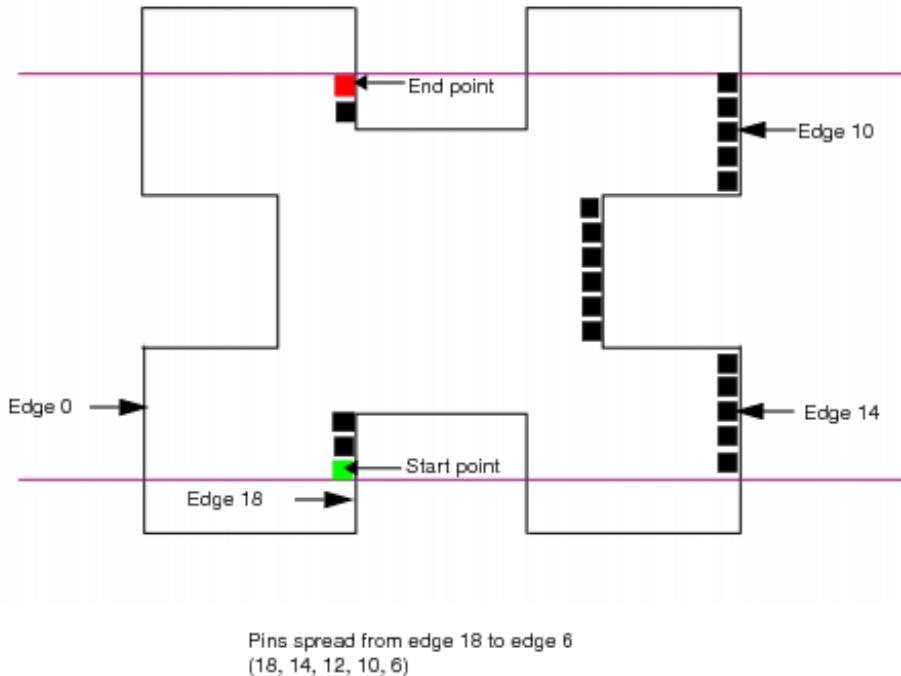


The following two figures shows how pins are spread along a side in case of rectilinear partitions where a side with multiple edges has been specified.

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



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



## Spreading Floating Pins

Floating pins are spread across all edges over the whole partition. The number of pins on each edge depends on the pin density. An edge with higher pin density will have lesser pins.

The following considerations are taken into account while spreading the floating pins:

- If some bits of buses have connections and the rest are floating, then the floating pins are kept together as a bus.
- If a bus is floating then it is placed together. If not then its pins are placed on the same edge.
- If some bits of a pin group have connections and the rest are floating, then the floating pins are not spread and the order of the pin group is maintained.
- In the master and clone scenario, both pair of nets (for pin connecting master and pin connected clone) should be floating to consider it as a floating pin pair.
- In the nested scenario, floating pins are not aligned across hierarchies.

## Running Relative Floorplanning

This section describes how to use the *Floorplan* menu's *Relative Floorplan* form to capture and define the placement relationship of floorplan objects independently from the actual coordinates in a floorplan.

The Relative Floorplan program provides a more flexible way to place objects, such as modules, blocks, groups, blockages, pin guides, pre-routed wires, and power domains. Block I/O pins can be used as reference objects but they cannot be relative objects. You can capture and define the placement relationship of floorplan objects independently from the actual coordinates in a floorplan. You can also resize a module or blackbox based on other floorplan objects, while maintaining its current area based on a specified width and height, a given dimension (width or height), and a target utilization value. You can also specify a wire's start or end point relative to the reference object's reference corner, or specify a wire's start or end point directly.

Before relative floorplanning, the design must be loaded into the current Innovus session.

## Orientation Key

The following table is a key to the orientation of placed objects:

| Value | Definition                                                    |
|-------|---------------------------------------------------------------|
| R0    | No rotation                                                   |
| MX    | Mirror through X axis                                         |
| MY    | Mirror through Y axis                                         |
| R180  | Rotate counter-clockwise 180 degrees                          |
| MX90  | Mirror through X axis and rotate counter-clockwise 90 degrees |
| R90   | Rotate counter-clockwise 90 degrees                           |
| R270  | Rotate counter-clockwise 270 degrees                          |

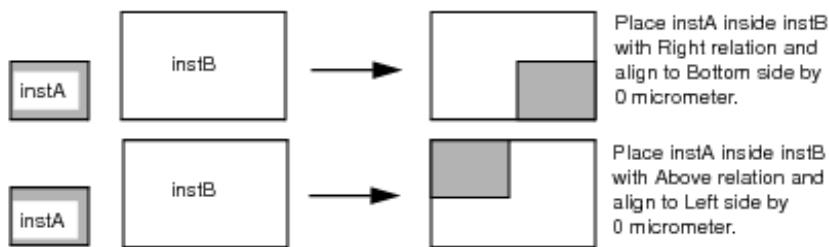
MY90



Mirror through Y axis and rotate counter-clockwise 90 degrees

## Instance Place Example

The following figure shows an example of instance placement



## Pre-Route Examples

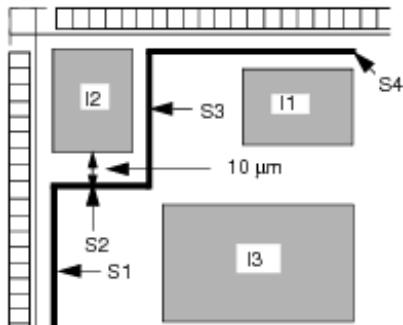
You can generate pre-routed relative floorplan information automatically after you have pre-routed a relative floorplan. The following example and illustrations show how the relative floorplan pre-route feature operates.

The following commands specify that S3, S2, and S1 are relative to object I2 and the core:

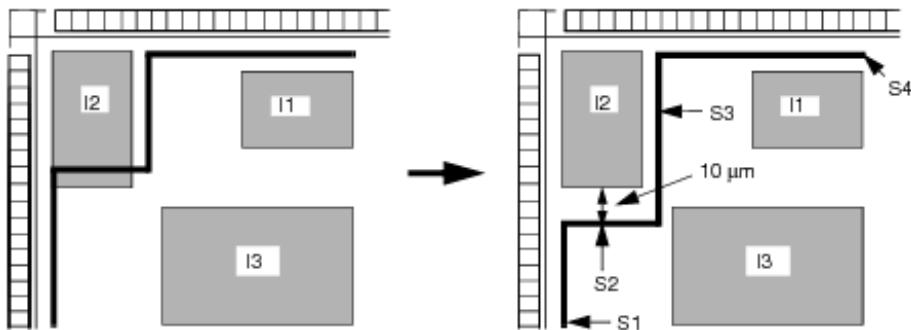
```
relativeFPlan --preRoute VDD Metal1 Metal12 0.44 0.44 I2 BR X2 Y2 I2 TR X3 Y3
```

```
relativeFPlan --preRoute VDD Metal1 Metal12 0.44 0.44 I2 BL X1 Y1 I2 BR X2 Y2
```

```
relativeFPlan --preRoute VDD Metal1 Metal12 0.44 0.44 core BL X0 Y0 I2 BL X1 Y1
```



When I2 is stretched southward, the original distance between its south side and the S2 side of the wire is readjusted, but maintains its distance from the south side at 10 μm.



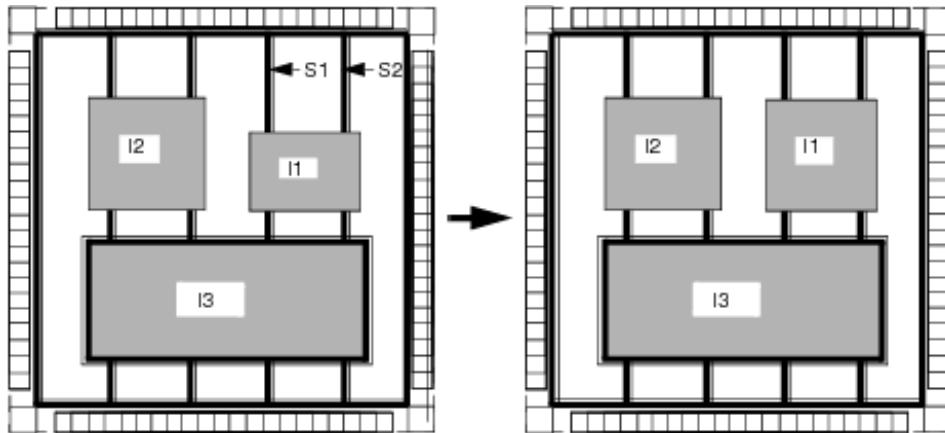
The following commands specify that S1 and S2 are relative to the object I2 and the Core\_Boundary:

```
relativeFPlan --preRoute VSS Metal1 Metal2 0.44 0.44 I1 TL x1 y1 Core_Boundary TR x2 y2
```

```
relativeFPlan --preRoute VDD Metal1 Metal2 0.44 0.44 I1 TR x3 y3 Core_Boundary TR x4 y4
```

```
relativeFPlan --preRoute VSS Metal1 Metal2 0.44 0.44 I1 TL x5 y5 Core_Boundary TR y6 y6
```

```
relativeFPlan --preRoute VDD Metal1 Metal2 0.44 0.44 I1 TR x7 y7 Core_Boundary TR x8 y8
```



In the above example, when I1 is stretched northward, the S1 and S2 wires are shortened.

## Saving and Restoring Relative Floorplan

The Innovus software automatically saves all the executed menu and text commands for the relative floorplanning actions in the innovus.cmd file.

To save all the relative floorplan commands that were executed during a session, click the Save button on the *Relative Floorplan* form. This saves a script that can be used later for updating or adjusting an existing floorplan based on the new blocks' size and position. You can also save the constraints associated with an object through the [saveRelativeFPlan](#) command.

To restore the relative floorplan information to the design display area, use the [restoreRelativeFPlan](#) command.

## Saving and Loading Floorplan Data

You can save and load floorplan data at any time during a session.

- To save the floorplan information to a file, use the *Save FPlan File* form or the [saveFPlan](#) command.

- To load the floorplan information from a file, use the *Load FPlan File* form or the [loadFPlan](#) command.

**Note:** You can save and load floorplan data in the XML format.

The `saveFPlan` command saves net attributes including non-default attributes of a net, like weight, bottomPreferredRoutingLayer, or detour. This capability is especially useful in the prototyping flow where you need to save out the state of the floorplan, and timing-driven proto\_design sets net weights to guide planDesign to a timing driven result.

For large designs, as the designer iterates through different versions of the floorplan, they often check-point by doing a `saveFPlan`. A saved .fp file is also excellent for two people working separately on the same floorplan, so that .fp can be passed on to another designer who has a slightly different netlist. By saving the net attributes the `saveFPlan` and `loadFPlan` commands are able to bring back the entire prototyping/floorplanning picture.

## Specific Floorplan Section TCL Export/Import

You can use the `writeFPlanScript` command to write out specified floorplan sections and source the output file after `init_design`. With this command Innovus writes a file that you can examine, edit, and then source to selectively (or completely) define your floorplan. The `writeFPlanScript` command writes out a file with TCL commands to recreate the floorplan. It can optionally write out specific floorplan sections listed with option `-sections`.

The benefit of this command is selective export/import of concise TCL for better command discovery and easy partial export/import of a portion of a floorplan between two designs. The human readable TCL floorplan file format completely captures the floorplan specification and is cut and paste sourceable.

## Snapping the Floorplan

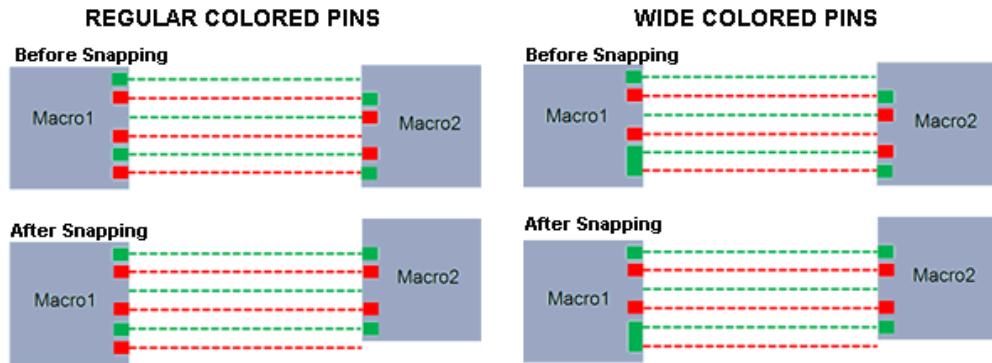
The snap floorplan feature enables you to snap objects to the grid. You can use the `snapFPlan` command or alternatively use the *Snap Loaded Floorplan* form to align objects to the grid. Additionally, you can use the `snapFPlanIO` command to snaps I/O cells to a user-defined grid.

**Note:** You specify the user-defined grid in the *Preferences - Floorplan* form in the *Options* menu.

In floorplan flow, after CCE (colorizeGeometry) assigns color to macro pins and other objects. Using the `snapFPlan -macroPin` command you can snap macro pins to the correct color

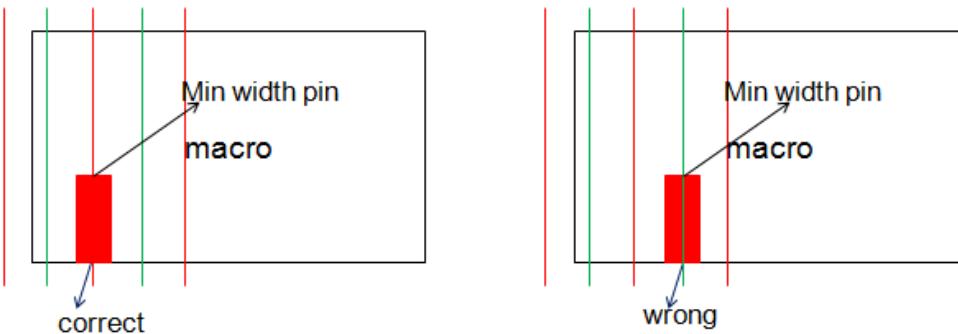
track. The `snapFPlan` command only snaps signal pins and places a macro with macro pins on color track. Innovus snaps the port on the lowest metal layer which is close to macro origin to the color track.

In addition to the macro pin color-aware snapping, Innovus also checks that the pins have default width and it uses the first default wide pin to snap. The non-default-width pin shapes are ignored while snapping on colored tracks and only the pins with default width are snapped.

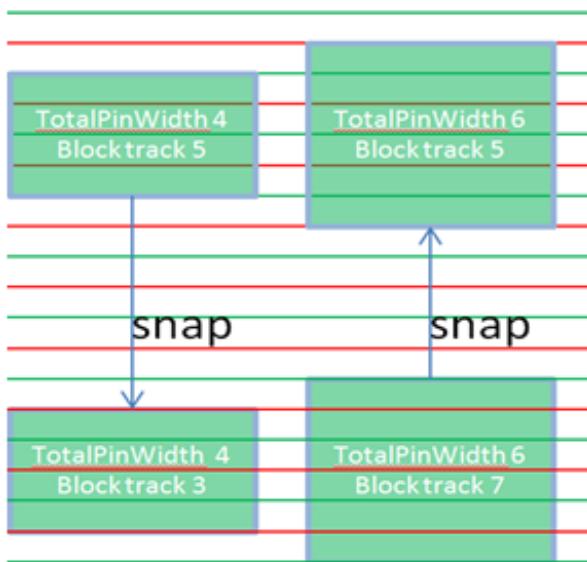


The `checkFPlan` command verifies if the macro pins are snapped to the correct color track. It reports a violation if the macro placer cannot find legal location or when macro pins have color violation.

- For min width pins, the `checkFPlan` command checks whether the pin center is snapped to the routing track with the same color.



- For fat pins, the `checkFPlan` command checks whether the pin center is snapped to the routing track and decides if the pin center should snap to the same or opposite colored routing track.



Innovus chooses the result which saves maximum routing track. It reports an error if a pin is not snapped to any track.

It displays a warning if the fat pin snap is not correct. It gives details of:

- The violated fat pin layer and shape.
- The number of tracks the pin has occupied.
- How to modify the LEF file to save routing track.

## Resizing the Floorplan

The resize floorplan feature enables you to resize a floorplan while maintaining the relative locations of the floorplan objects.

Normally, floorplan resizing is done

- to reduce the die size on a finished floorplan.
- to expand or shrink the die during floorplan creation, based on the full chip placement results.

You can set the global parameters for the `resizeFP` command by using the `setResizeFPlanMode` command. The parameters that you specify with the `setResizeFPlanMode` command are then used automatically whenever you run the `resizeFP` command to resize the floorplan. Alternatively, you can use the [\*Floorplan - Resize Floorplan\*](#) form in the Innovus GUI to perform the resize functions in the design.

**Note:** You can use the `getResizeFPlanMode` command to view the information about a specified

`setResizeFPlanMode` command in the Innovus log file and in the Innovus console.

## Resize Floorplan Options

The space among floorplan objects can be resized in three ways:

### Proportional Spacing

Distributes the space among floorplan objects proportionally (`setResizeFPlanMode -proportional`). It can shrink or expand the space in both, X and/or Y directions. However, you cannot adjust pre-routed wires using proportional spacing.

See [Resize Floorplan Proportional mode](#) in the "Floorplan Menu" chapter of the *Innovus Menu Reference* for more information.

### Shift-based Spacing

Shifts floorplan objects at the right/upper (x/y resize) sides of the resize line and keeps the location of the rest of the floorplan objects. (`setResizeFPlanMode -shiftBased`).

You can perform automatic resizing or resize the floorplan based on resize lines defined using the `setResizeLine` command. The shift-based resize maintains the existing pre-routed wire topology and automatically adjusts bus guides during resizing.

See [Resize Floorplan Shift Based mode](#) in the "Floorplan Menu" chapter of the *Innovus Menu Reference* for more information.

### Congestion-based Spacing

Resizes and shifts the floorplan objects by estimating the congestion for the floorplan and automatically deciding where to draw a resize line to avoid the congested area.

(`setResizeFPlanMode -congAware`)

See [Resize Floorplan Congestion Based mode](#) in the "Floorplan Menu" chapter of the *Innovus Menu Reference* for more information.

## Setting Resize Lines

For performing shift-based resizing, you can specify one or multiple resize lines.

For example,

```
setResizeLine -direction H (x1 y1) (x2 y1) (x2 y3) (x4 y3) (x4 y5) (x6 y5) -width 20
```

When specifying a resize line, if you do not specify a coordinate for the resize line, the `setResizeLine` command automatically derives the missing coordinate.

For example, the following command automatically derives the missing coordinate for setting the resize line:

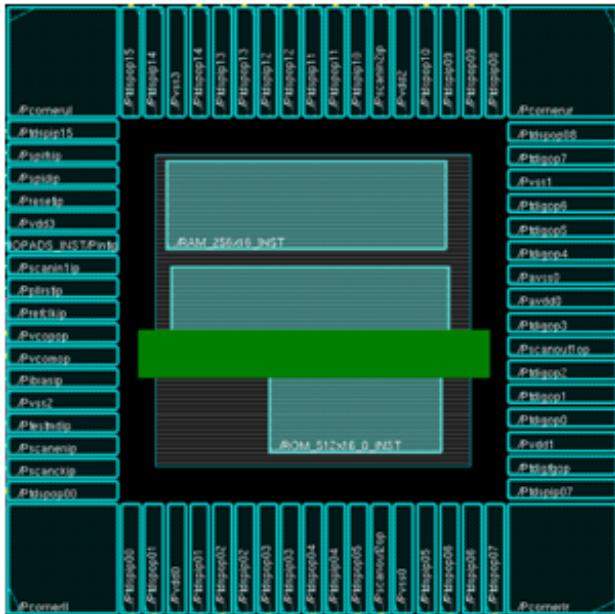
```
setResizeLine -direction H (x1 y1) (x2 *) -width 20
```

## Specifying Resize Directions

Resizing can be done in X and Y directions. Positive values mean expanding the space and negative values indicate shrinking. However, Innovus does not support a scenario where resizing line by expanding and shrinking, both occur on the same direction.

For example, the following command specifies a resize line in horizontal direction with a resize width of 100 microns:

```
setResizeLine -direction H -width 100 (279.2845 555.443) (1030.896 555.443)
```



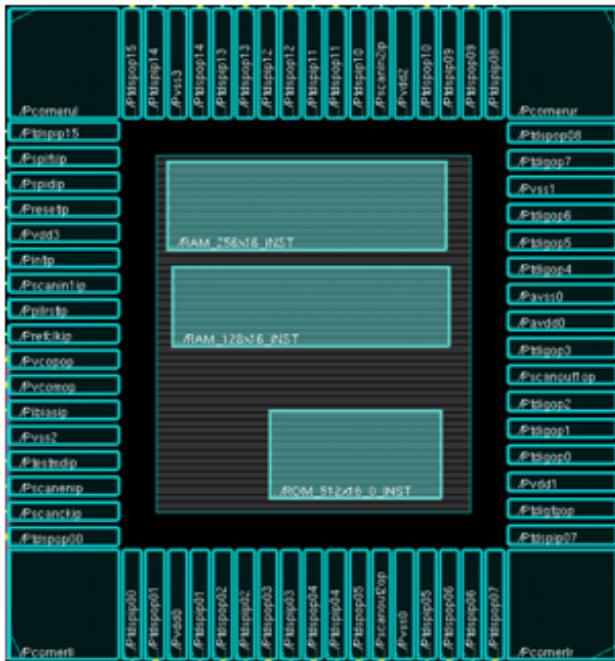
The following command again resizes the floorplan in the same X direction with a negative value of -200 microns:

```
resizeFP -xSize -200
```

Innovus displays a warning message in such a situation.

The following command resizes the floorplan in the same Y direction with the same positive value of +100 microns:

```
resizeFP -shiftBased -ySize +100
```



## Snapping Resize Values

The resize values (shrink/expand) can be snapped to a multiple integer of the metal layer pitch.

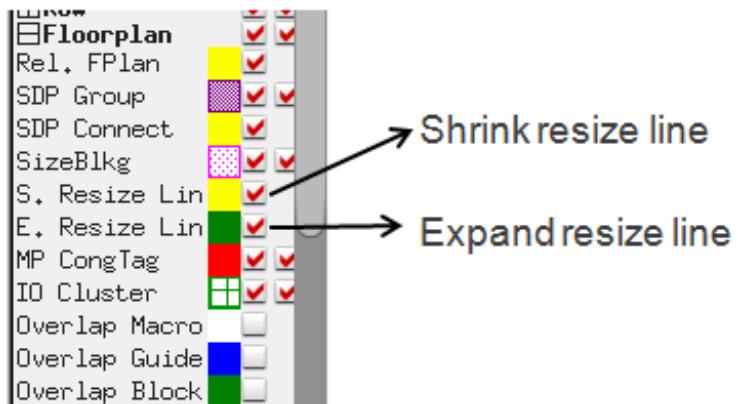
**Note:** Specify the `setResizeFPlanMode -snapToTrack` option to snap resize values.

For example, if the horizontal metal pitch is 1.5 microns and you want to shrink the floorplan by 8 microns in y direction, the actual shrink value is 7.5 microns, the nearest multiple integer of the metal pitch.

See [Resize Floorplan](#) in the "Floorplan Menu" chapter of the *Innovus Menu Reference* for more information.

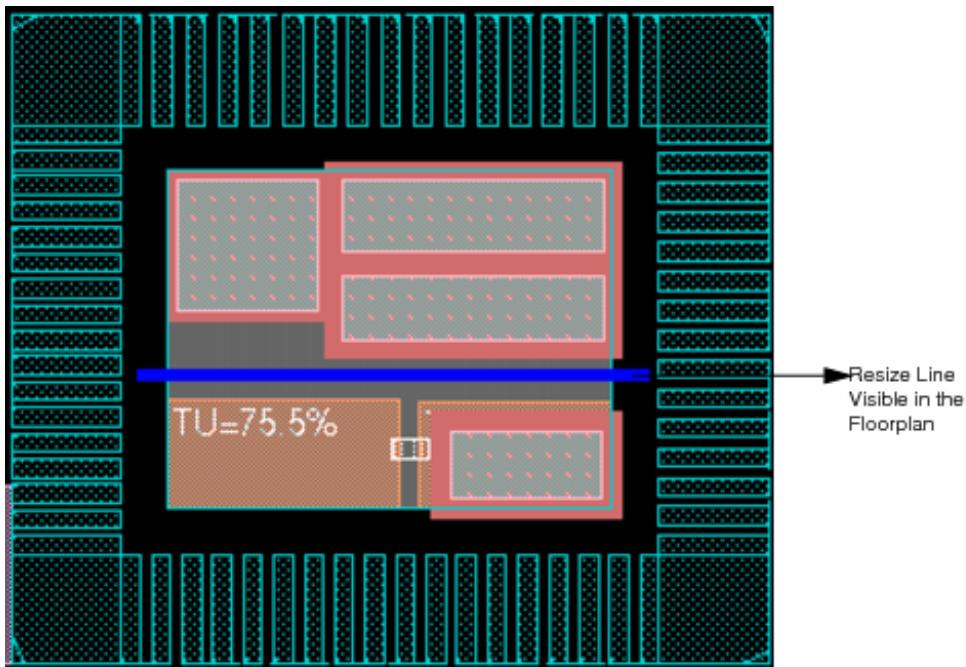
## Viewing Resize Lines using Color Preferences

Once you specify the resize lines to perform shift-based resizing, you can display the resize lines in Innovus by setting the *Resize Line* option in the *Floorplan*.

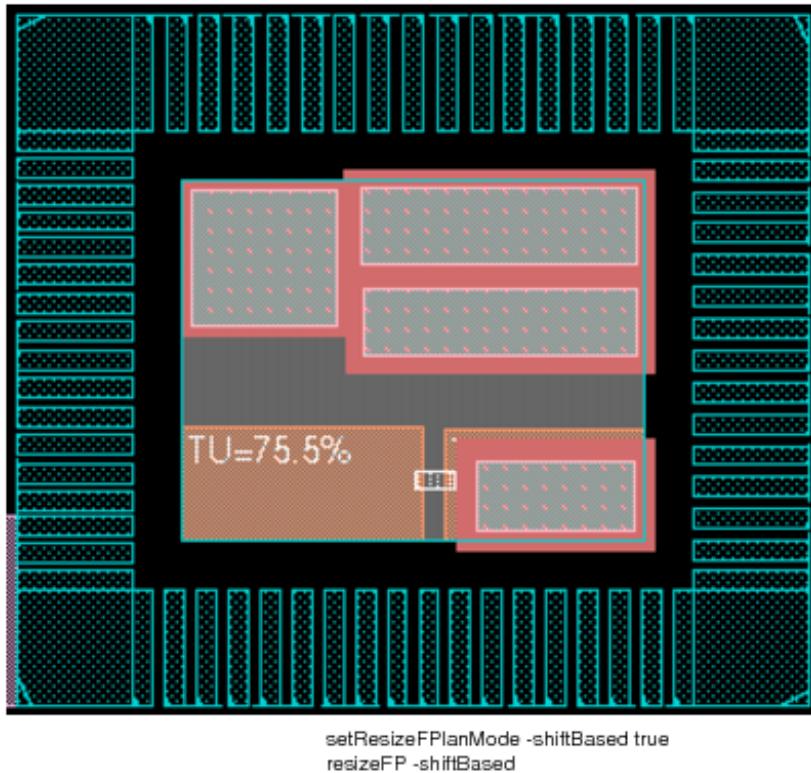


However, the resize lines disappear once you run the `resizeFP` command.

### Example: Setting a Resize Line Before running `resizeFP`



### Example: Resize Line Disappears After Running resizeFP in Shift Based Mode



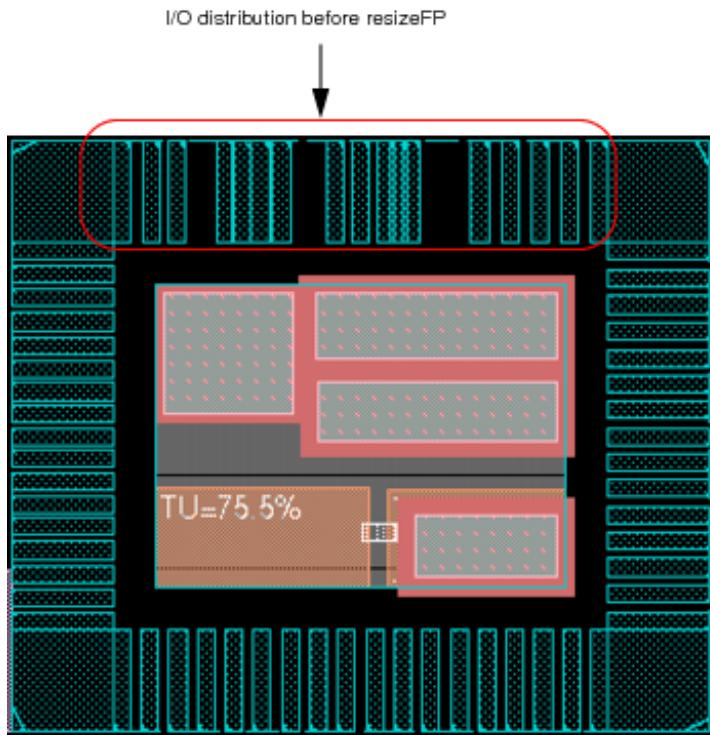
**Note:** During resizing, the target size may not be achievable. You have to force resize to meet the target size as much as possible, using the `resizeFP -forceResize` option.

## Distributing I/Os using Resize Floorplan

You can use the `setResizeFPlanMode -ioproportional` to specify how I/Os are handled during floorplan resize. When distributing the I/O's using resize floorplan,

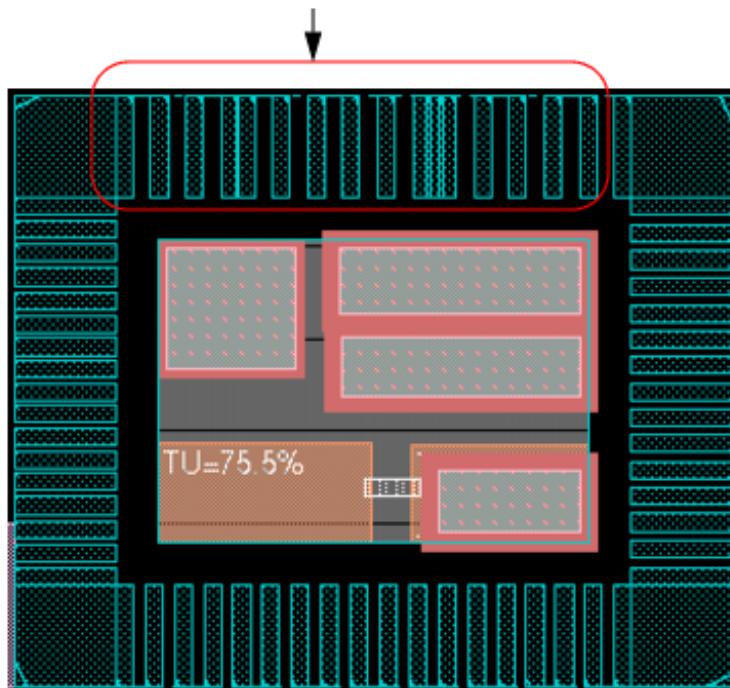
- The space between I/O pads can be adjusted evenly or proportionally. By default, the I/O's are distributed proportionally.
- The I/O side constraints and order constraints are honored.
- The offset that exists between I/O's and the design boundary is preserved.

## Example: Distribution of I/Os Before Resize Floorplan



## Example: Evenly Distribution of I/O's After Resize Floorplan

Evenly I/O distribution after  
setResizeFPlanMode -ioProportional false  
resizeFP -xSize 50



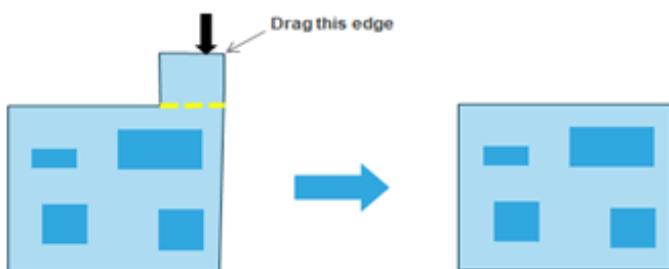
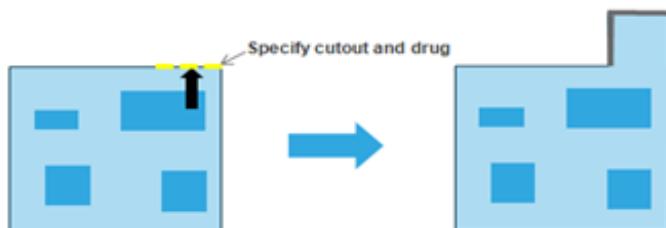
## Resizing Floorplan Bounding Box in GUI

You can create a rectilinear floorplan by dragging edges of the bounding box. You can drag and edit all edges of a floorplan including cutouts without moving the other objects. In case of new violations, such as placed objects out of core or I/O pins out of core or routing grid, Innovus will display a violation and suggest a solution to user.

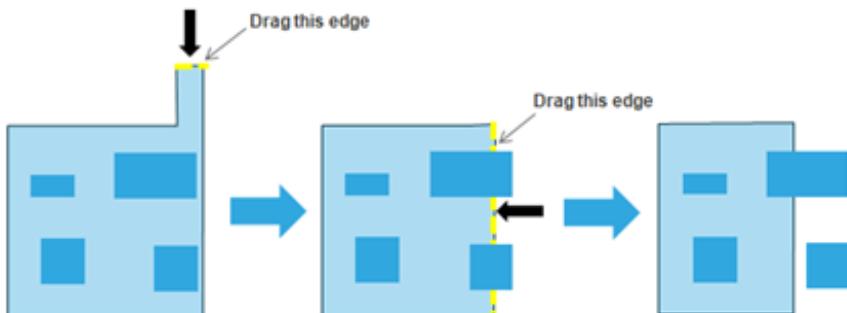
### Selecting and Dragging an Edge

You can change the top-cell bounding box and cutting rectangle floorplan to rectilinear shape. IO box /die box/core box will be automatically updated after floorplan reshaping. Innovus cuts an edge at the point you specify.

1. Innovus allows you to create/manipulate cutouts. You can cut rectangle floorplan to rectilinear shape and Innovus will automatically cut an edge at the point specified by you. After cutting, the edge will be divided into two segments and the new segment is editable. You can drag this segment to shrink or expand freely.
2. If you move the selected edge, you can make existing cutout to disappear or “reverse” edge direction.
3. You can create cutouts and then specify the length of each new edge, the cutouts may disappear you merge two edges as one.



You can drag any edge of the floorplan boundary and edit each edge to reshape the rectilinear floorplan.

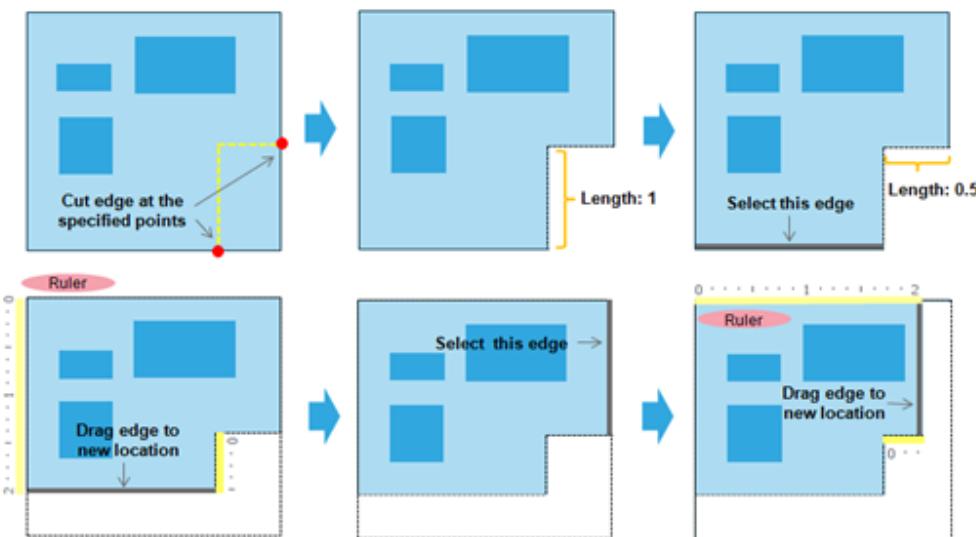


## Moving Edge and Setting Length of the Specified Edge

Innovus displays the length of the selected edge in the ruler when you drag the edge to reshape. This allows you to change the top-cell bounding box and cutting rectangle floorplan to rectilinear shape.

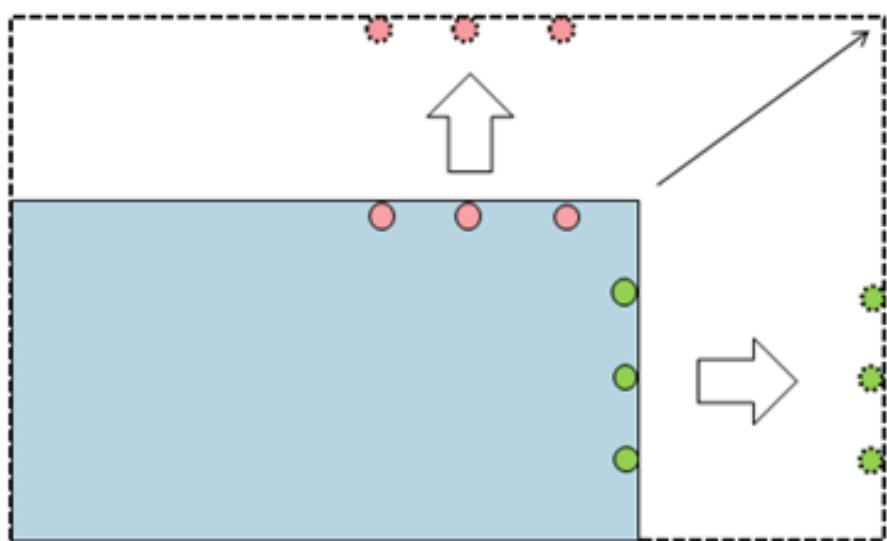
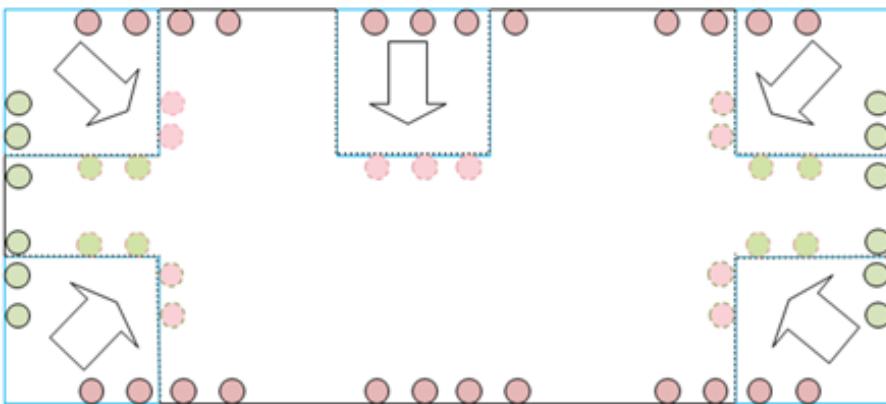
To set the length for an edge:

1. Specify a point at the boundary edge, and cut the edge to make a cutout.
2. The length of the new edge (vertical) in the figure below is 1.
3. You can select an edge and drag the edge to a new location. The length of the new edge (horizontal) in the figure below is 0.5.
4. Using the ruler, you can drag the specified edge and set its length according to the ruler.
5. You can select another edge to edit and using the ruler, you can drag the specified edge and set its length.



## Movement of I/O Pins with Edge

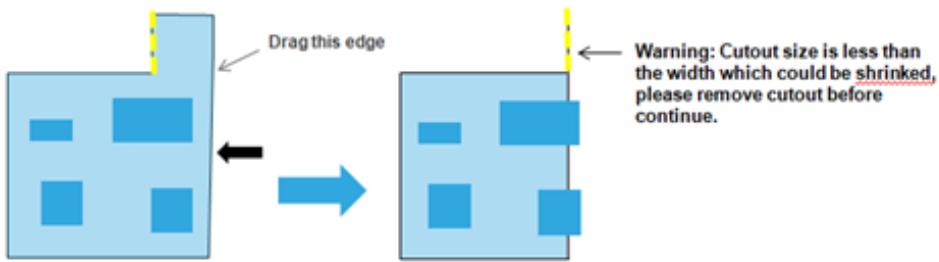
When you move an edge while reshaping a floorplan, I/O pins are moved with the edge. Innovus keeps the same position of the pin (like a rectilinear cut) on the new position of the edge (snapping pin). Also, the side and the sequence of the I/O pin remains the same as in the old floorplan, while the pins get distributed according to the new position of the edge.



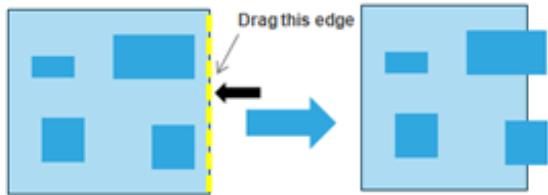
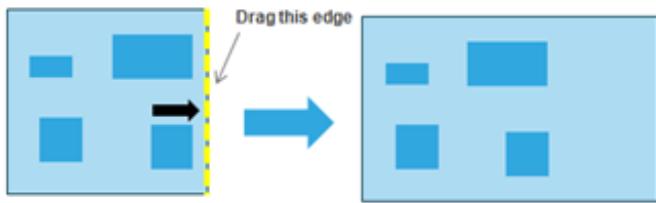
Innovus does not honor pin constraints. The I/O pins which cannot find a legal location, after floorplan is reshaped, are unplaced.

## Automatic Checking

Innovus stops resizing when the cutout size gets lower than the predefined threshold.



When you drag an edge of a floorplan, all objects remain untouched. You can shrink or expand the edge. However, Innovus creates violations if the object is outside the floorplan boundary after resizing. It issues a warning message to let you fix the violation. You can run [checkFPlan](#) before/after and capture the delta.



## Undo/redo Capability

You can always reverse your changes after reshaping a floorplan.

## Checking the Floorplan

After creating the floorplan, you can check the floorplan to identify problems before a design is passed to downstream tools. You can check the bus guide, routing grid, placement, pins, power domains, partition clone alignment, feedthrough buffer insertion and narrow channel etc.

## checkFPlan

Use the `checkFPlan` command to check the quality of the floorplan to detect potential problems before the design is passed on to other tools. This highlights the cells in the design display area with violation markers, where applicable. Use this command after specifying the floorplan data or running an initial floorplan. Run the `checkFPlan` command on your final floorplan file. Alternatively, you can use the *Check Floorplan* form.

## checkDesign

Use the `checkDesign -floorplan` command to check the floorplan, generate error or warning messages, and report the following information:

- Off-grid horizontal and vertical tracks
- Instances not snapped to row site
- Unplaced I/O pins
- Off Grid Power/Ground Pre-routes
- Instances not on the manufacturing grid
- Preroutes not on the manufacturing grid (error)
- Regular preroutes not on tracks

## checkFPlanSpace

Use the `checkFPlanSpace` command to check the floorplan for spacing rule violations and save the violation information in a report. A marker is displayed at each violation spot. Alternatively, use the *Check Floorplan Space* form to check the floorplan for spacing rule violations.

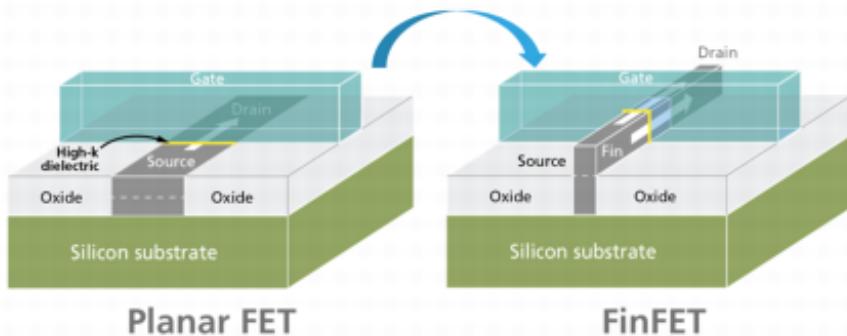
## adjustFPlanChannel

Use the `adjustFPlanChannel` command to automatically adjust the channel width between objects to prevent potential routing congestion. You can use this command after importing the design, loading a completed floorplan, and then running the `placeDesign` and `trialRoute` commands.

## FinFET Technology

FinFETs are 3D structures that rise above the planar substrate, giving them more volume than a planar gate for the same planar area. Given the excellent control of the conducting channel by the gate, which wraps around the channel, very little current is allowed to leak through the body when the device is in the off state. This allows the use of lower threshold voltages, which results in optimal switching speeds and power.

In a FinFET, the FET gate wraps around three sides of the diffusion fin as shown below. This forms conducting channels on three sides of the vertical fin structure. This approach provides much more control over channel current compared to planar transistors. Multiple fins can be used to provide more current.



FinFETs also promise to alleviate problematic performance versus power tradeoffs. Designers can run the transistors faster and use the same amount of power, compared to the planar equivalent, or run them at the same performance using less power. This enables design teams to balance throughput, performance and power to match the needs of each application.

## FinFET Support in Innovus

Ideally origin of all placeable objects (standard cells, I/o pads and blocks) must be aligned to FinFET girds to be manufacturable for the FinFET technology. FinFET library rules are used to define FinPITCH in the technology LEF file to support the FinFET grid. The FinFET pitch is used to define the legal Y/X values of the placement grid in a design. Innovus honors the FinFET grid and automatically checks for the FinFET definition. If it detects a FinFET definition it enables support and sets the placement grid as FinFET grid by default. It also displays a message informing you about the pitch, offset, and the direction values.

## Defining FinFET Rule using LEF 5.8

FinFET pitch can be defined in LEF 5.8 to support FinFET grid. User can define fin pitch, offset and the direction of fin grid in PROPERTYDEFINITIONS of technology LEF file. The FinFET pitch is used to define legal Y (or X) values of placement grid in a design, while the other placement grid is typically derived from M1 grid. The FinFET grid must be a multiple of manufacture grid and one FinFET grid per design.

You can define a FinFET rule using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_FINFET STRING
"FINFET
    PITCH pitch [OFFSET offset] {HORIZONTAL|VERTICAL}
    ;"
END PROPERTYDEFINITIONS
```

Where:

```
FINFET
    PITCH pitch [OFFSET offset] {HORIZONTAL | VERTICAL}
```

Specifies the FinFET pitch to be `pitch`, which starts at the `offset`, if specified, or zero from the origin of the design.

The HORIZONTAL/VERTICAL keywords mean that the FinFET pitch is used to define the legal Y/X values of the placement grid that all cells, blocks, and IOs must align to (specifically, the origin of every cell must be aligned to the legal Y/X values), in order to guarantee all the FinFETs inside are aligned properly. The X/Y value of the placement grid is derived from the standard cell site width and only applies to standard cells. The cell row height should typically be multiple of the FinFET pitch.

**Type:** Floats, specified in microns

**Note:** [OFFSET offset] is not supported yet.

### Finfet Rule Examples

The following example indicates that the FinFET y pitch is 0.108 μm:

```
PROPERTYDEFINITIONS
    LIBRARY LEF58_FINFET STRING "
    FINFET
        PITCH 0.108 HORIZONTAL ;
    END PROPERTYDEFINITIONS
```

## Snapping Physical and Floorplan Objects to FinFET Grid

With the introduction of FinFET grid, all of the placeable objects such as I/O pads, macros and standard cells (cell rows) must be placed on the FinFET grid defined in technology LEF file. The height of all placeable objects must be a multiple of FinFET grid and the origin of these objects must be snapped to the FinFET grid. If FinFET PITCH is defined in technology LEF file, Innovus sets the placement grid as FinFET grid by default and checks for any violations.

**Note:** The *FinFET Grid* options are available on the *Layer Control* bar. You can use it to control the visibility of the FinFET grid. Additionally, you can specify the color preference for the FinFET grid using the *FinFET Grid* option on the *Color Preferences* form.

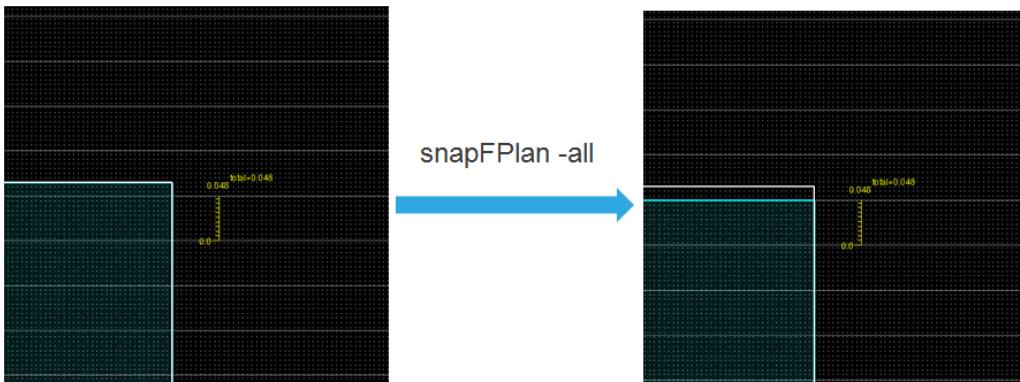
The following floorplan commands support this functionality:

### snapFPlan

The `snapFPlan` command snaps blocks, I/O pads, and standard cells to FinFET grid. It honors user specified snapping rule for x direction. For y direction, it honors the FinFET grid (when the FinFET grid is horizontal in regular horizontal row design).

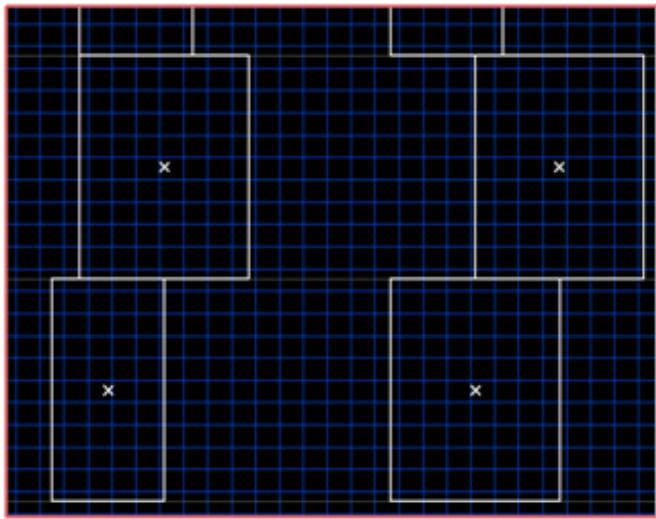
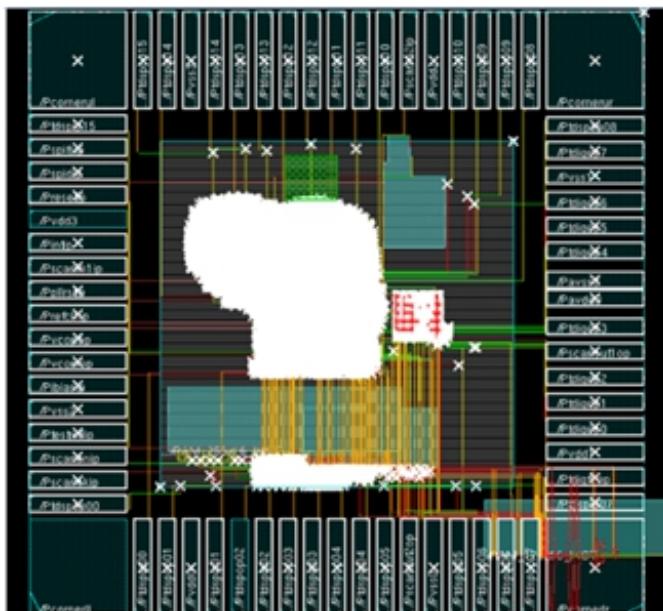
The SITE definition is as follows:

```
LIBRARY LEF58_FINFET STRING "
    FINFET
    PITCH 0.048 HORIZONTAL ; " ;
END PROPERTYDEFINITIONS
```



## checkFPlan

The `checkFPlan` command checks that all supported objects (die box, core box, standard cell rows, I/O rows, blocks, IO pads, and standard cells) are aligned with the FinFET grid and reports DRC violations. All supported objects which are not aligned to FinFET grid are marked in the GUI and reported in the Violation Browser.



## checkDesign

The `checkDesign -floorplan` command checks the floorplan and reports any objects not aligned with the FinFET grid correctly.

**Note:** In a top-down flow, for any partition shape (example, fence and power domain), the height of die box/core box must be a multiple of FinFET grid and all the corners of partition shape must be snapped to the grids.

## Related Topics

- [Floorplanning the Power Domain](#)

# Using Structured Data Paths

- [Overview](#)
- [Benefits of Using SDP](#)
- [General SDP Flow](#)
- [Support for High-Speed Flip Flop Columns](#)
- [SDP Placement Flow](#)
  - [placeDesign with SDP Placement](#)
  - [Placement with Flop Clustering](#)
- [Implementing SDP Capability](#)
- [SDP Relative Placement File](#)
  - [SDP File Examples](#)
  - [SDP File Format](#)
  - [Reusing SDP Instantiations](#)
- [Aligning SDPs by Pins](#)
- [Setting SDP Options](#)
- [Optimizing a Design with SDPs](#)
- [Checking SDP Placement](#)

## Overview

Innovus™ Implementation System provides the Structured Data Path (SDP) capability, which allows you to specify data path information to get better performance, power, and area. You can specify data path information by either importing an SDP relative placement file or sourcing an SDP TCL script. Correct SDP placement ensures uniform routing.

SDP capability should be used when:

- Design is data path intensive. That is, the design contains standard cell columns and rows that require alignment.
- Performance increases are required.
- Time to market does not allow for full custom flow.

SDP is a semi-custom methodology that requires manual intervention so you need to have detailed

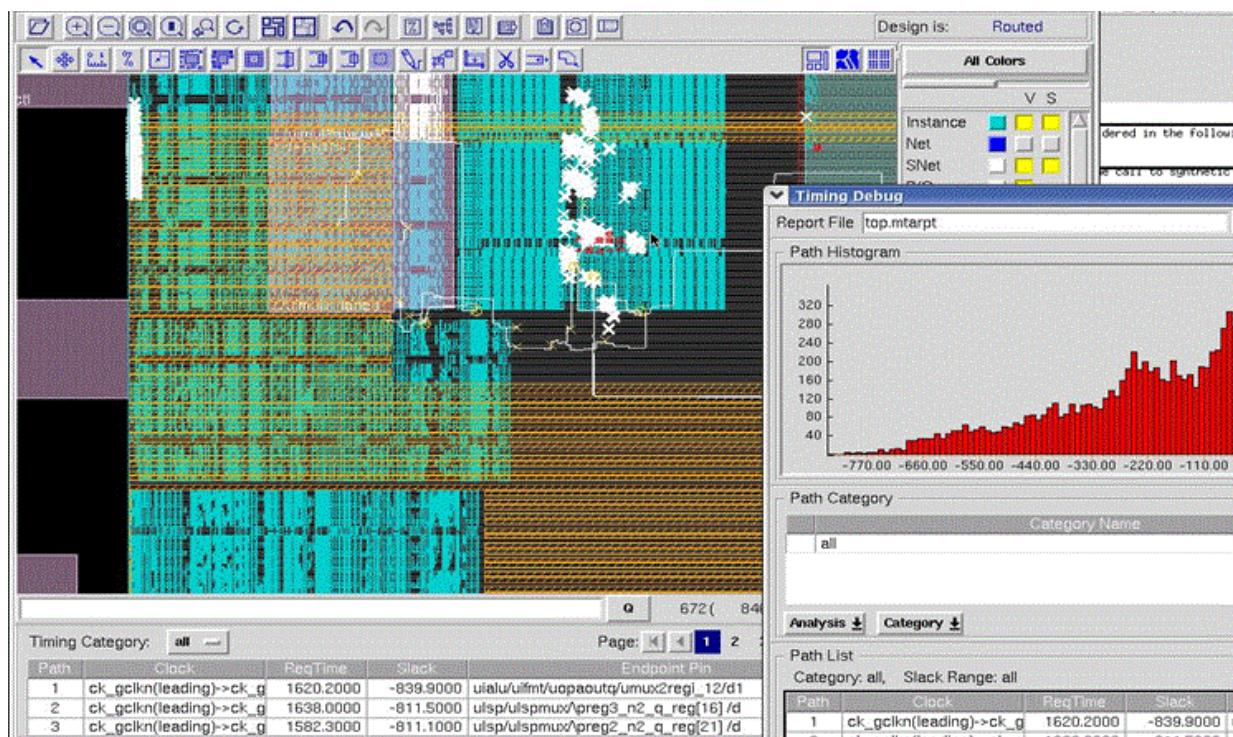
design knowledge in order to get better speed, power, and area. The Innovus tool makes it easy for you to try different SDP experiments and evaluate their impact on congestion, timing, and power. However, the tool still relies on the relative placement information you specify for placing and handling SDP elements.

Furthermore, currently Innovus does not identify SDP elements automatically. You must script them based on naming conventions and detailed design knowledge.

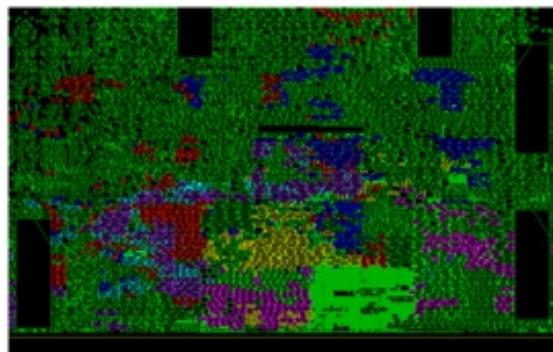
In addition, SDP capabilities can be used for high speed register or flip flop (FF) column methodology to help reduce clock latency and/or skew.

## Benefits of Using SDP

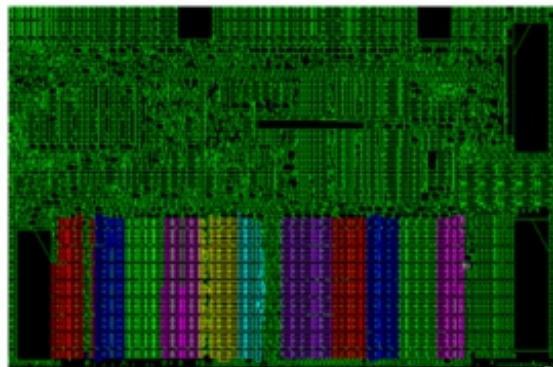
SDP provides a uniform environment for data path and control logic for placement, routing, and timing analysis.



SDP cells can be placed concurrently with other standard cells to get the optimal placement

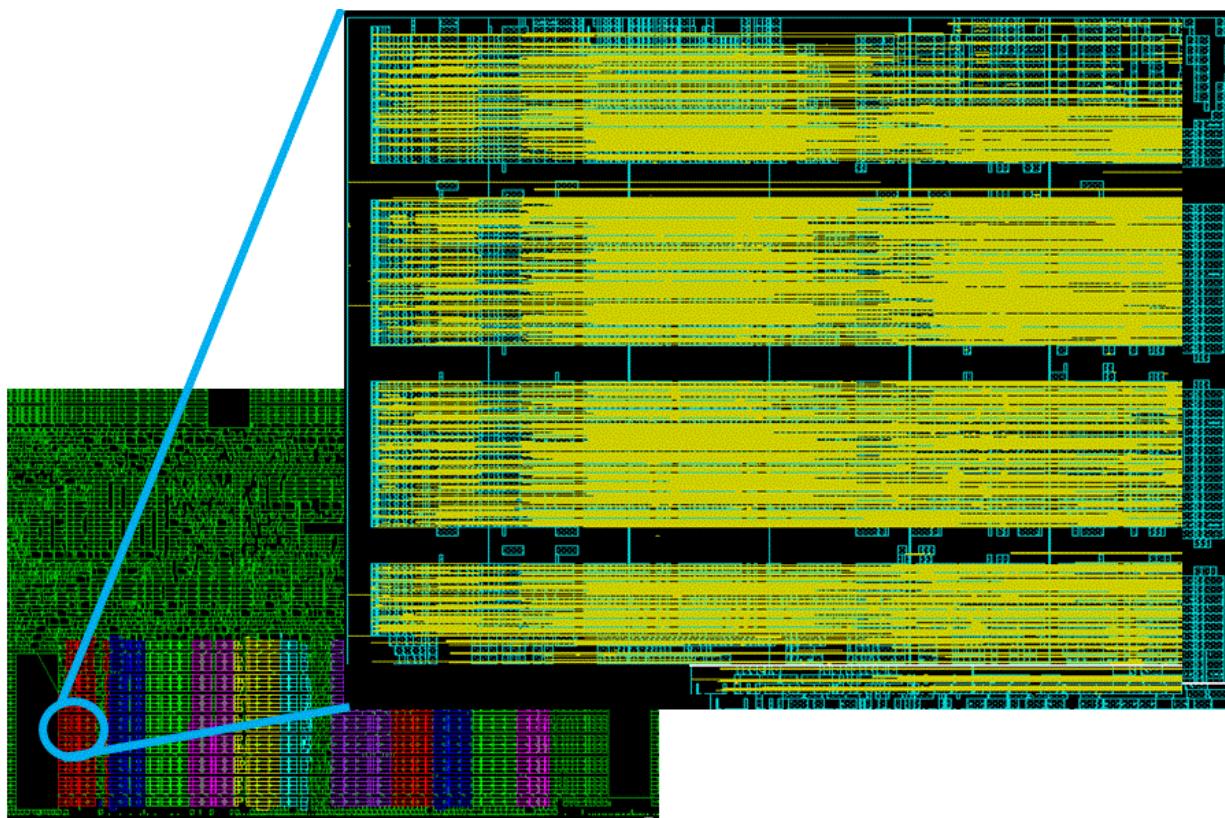


Traditional Placement



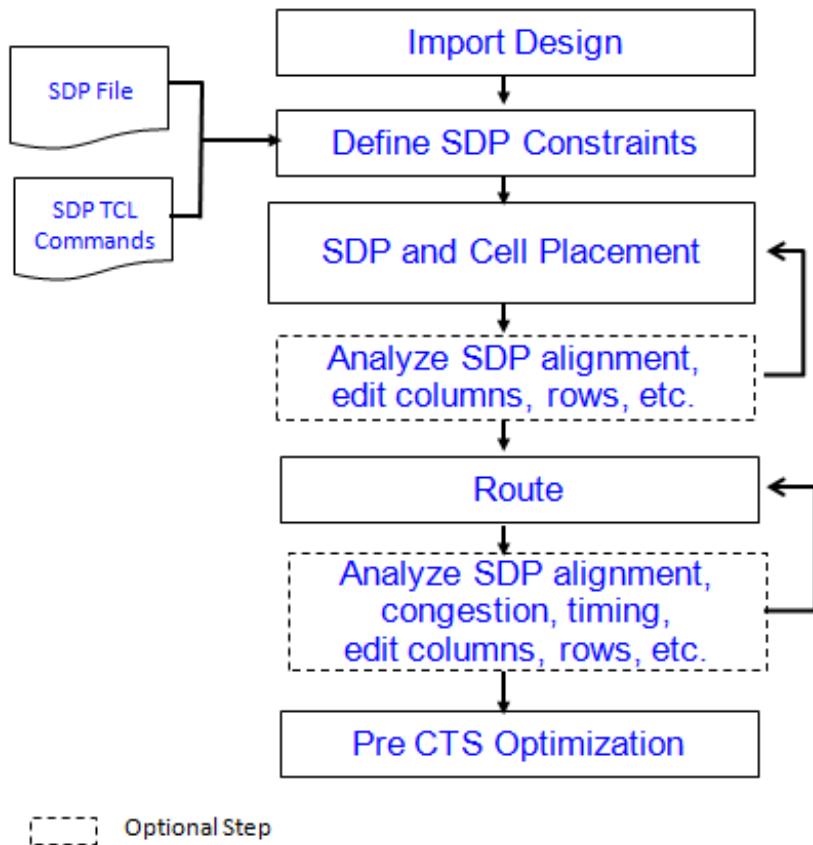
SDP Placement

The main advantage of this SDP placement is that it ensures uniform routing.



## General SDP Flow

The general SDP flow is as follows:

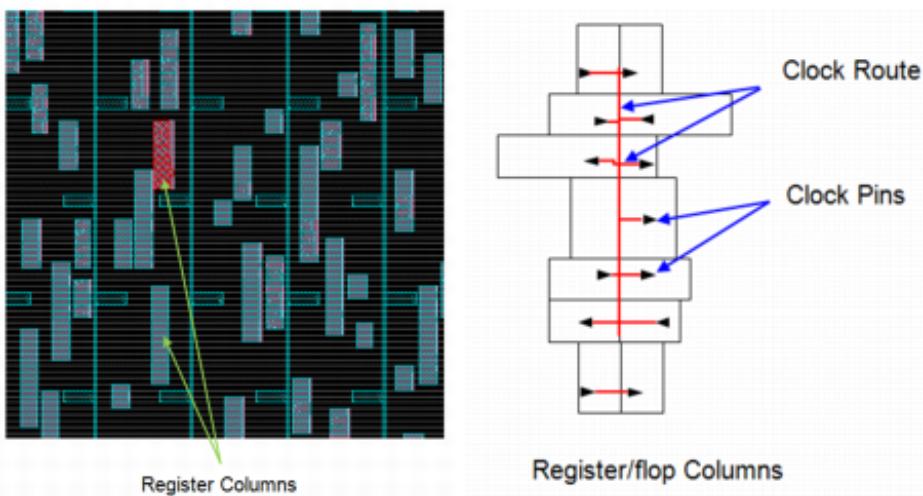


As shown in the flow diagram, after importing the design, you can read in an SDP file to define SDP constraints. Alternatively, you can source a TCL file that has SDP TCL commands, such as `createSdpGroup`, `addSdpGroupMember`, `addSdpObject`, and so on to define relative SDP placement. All SDP TCL commands can be used during the Analyze SDP alignment, edit columns and rows flow step.

Once SDP and standard cells have been placed, you can follow the normal flow

## Support for High-Speed Flip Flop Columns

High-speed register or flip flop columns help reduce clock latency and/or skew. In high-speed register methodology, flops that are driven by same clock are grouped together by placing them either one on top of the other in one column or side-by-side in two columns.

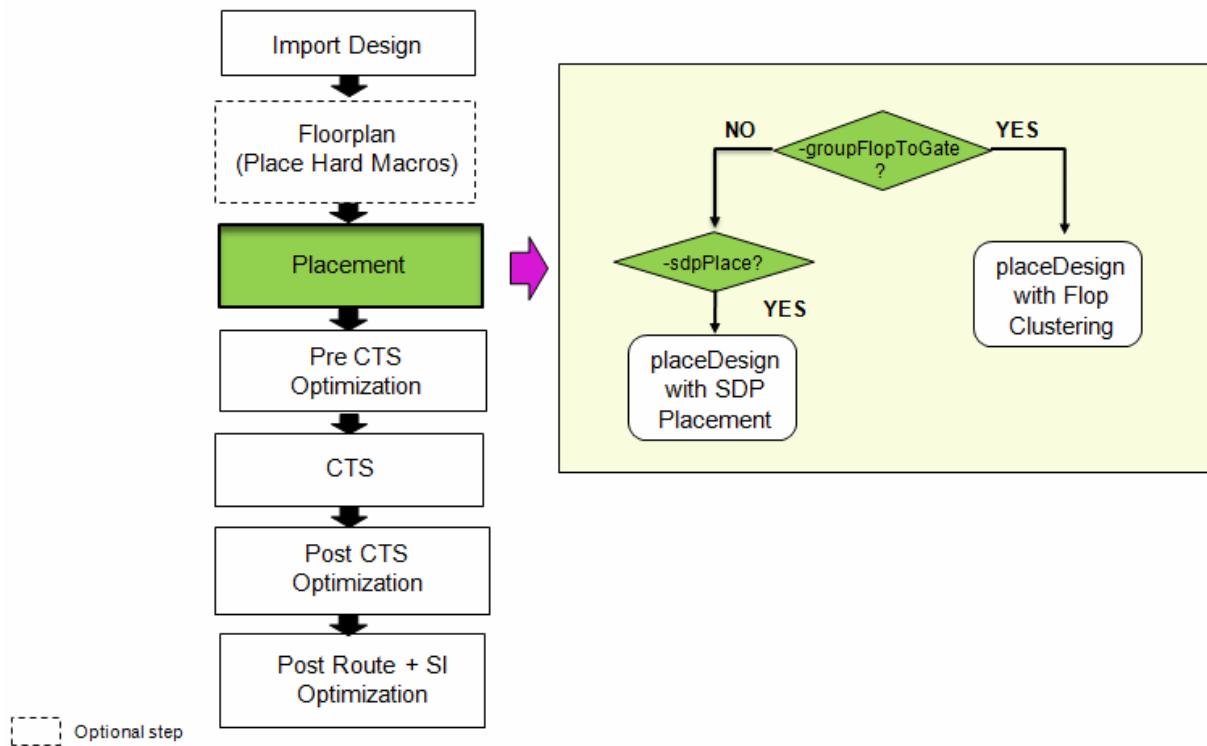


Some advantages of using high-speed register methodology are:

- Reduced latency as a result of tighter control of clock skew because of:
  - Short routing length from clock-driver to flip flops
  - Precise control over clock-driver to match known flip flop and routing load
  - Use of CTS or clock-mesh with flip flop column drivers as targets
- Reduced dynamic power as a significant fraction of power is often at the bottom of the clock tree.
- Reduce electromigration and voltage drop (IR-drop) issues on power-rails as flip flops are clustered into columns.

In high-speed register methodology, structured data paths are used to specify the order of the flops in the column.

## SDP Placement Flow

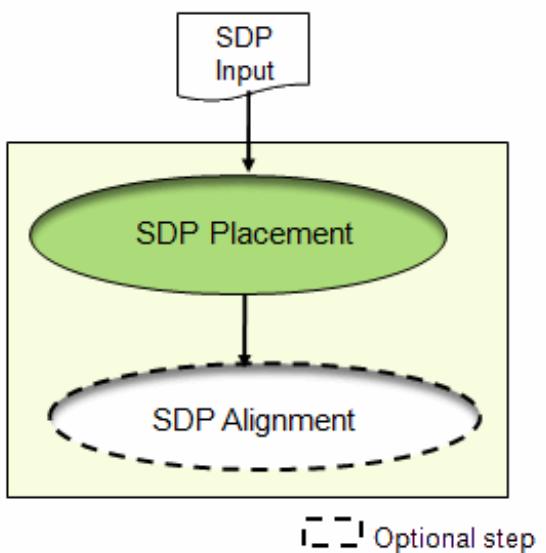


According to this flow, placement can be done in two ways, depending on whether or not you require flop clustering:

- [placeDesign with SDP Placement](#)
- [Placement with Flop Clustering](#)

## placeDesign with SDP Placement

[placeDesign](#) with SDP placement honors existing SDP groups and places SDP cells close together as a group. To enable [placeDesign](#) with SDP placement, set the `-sdpPlace` option of [setPlaceMode](#) to `true`. Optionally, you can set the `-sdpAlignment` option of [setPlaceMode](#) to `true` to enable SDP alignment.



The following pictures show the difference that `-sdpAlignment` can make:

| placeDesign with <code>-sdpPlace</code> | placeDesign with <code>-sdpPlace</code> and <code>-sdpAlignment</code> |
|-----------------------------------------|------------------------------------------------------------------------|
|                                         |                                                                        |

**Note:** Cadence recommends that you use `placeDesign` with `setPlaceMode -sdpPlace` and `-sdpAlignment` options instead of the existing `-useSdpGroup` option. The `-useSdpGroup` option will be obsolete in a future release.

## Sample Use Model

Here's a sample use model for `placeDesign` with SDP placement and alignment:

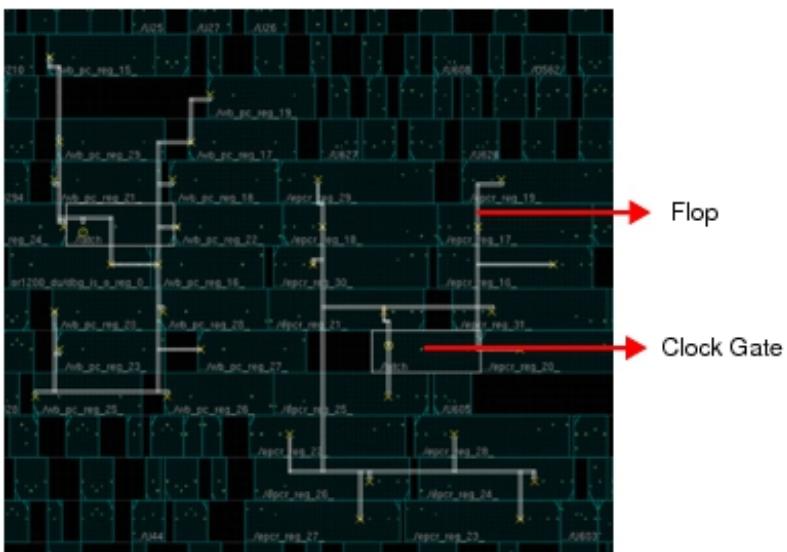
```
#Restore the design.  
  
restoreDesign dtmf.enc.dat dtmf_recv_core  
  
# Specify SDP relative placement information  
  
readSdpFile ff.sdp  
  
# Enable SDP placement.  
  
setPlaceMode -sdpPlace true  
  
# Enable SDP alignment  
  
setPlaceMode -sdpAlignment true  
  
# Limit SDP movement from its original cluster location to 30 um for resolving  
overlaps. Delete SDP groups if their overlap cannot be resolved. This step is optional.  
  
set_sdp_mode -max_move_distance 30 -max_move_action delete_sdp_group  
  
# Place the design  
  
placeDesign
```

## Placement with Flop Clustering

- [Overview](#)
- [Flop Clustering Flow](#)
- [SDP Alignment](#)
- [Sample Use Model](#)

## Overview

In placement with flop clustering, the flops that are driven by the same clock are clustered together based on clustering information you specify, such as the half-perimeter bounding box and the minimum and maximum fanout values on the bottom Integrated Clock Gate (ICG) fanout nets. Bottom level ICGs are the clock gates that directly drive flops but do not drive any other ICGs.



## Flop Clustering Flow

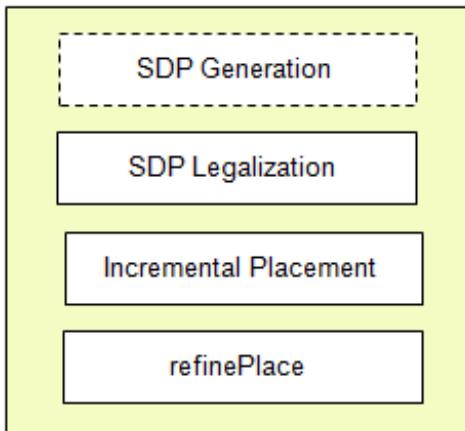
Placement with flop clustering is done using the super command `placeDesign`. When flop clustering is enabled, `placeDesign` applies a higher net weight to the ICG driving net so that the flops driven by the ICG are placed closer to the ICG cells. As a result, flops and clock gates are clustered together. This prevents wire length increase, improves clock skew, and reduces clock gating violations.

1. To enable flop clustering placement, set the `-groupFlopToGate` option of `setPlaceMode` to `true`.
2. Specify the size of the bounding box for each ICG-flop cluster by using the `setPlaceMode -groupFlopToGateHalfPerim` parameter to set the box's half perimeter. The default bounding box's half perimeter is 100 um.
3. Specify the maximum and minimum fanout values for clock gates using the `-groupFlopToGateMaxFanout` and `-groupFlopToGateMinFanout` parameters of `setPlaceMode`. The default minimum fanout value is 2 whereas the default maximum fanout value is 32. This means that by default `placeDesign` clusters those clock gates that drive between 2 to 32 leaf flops. These flops are pulled closer to the gating cell and placed within the bounding box you specify (default half-perimeter = 100).
4. Set `setPlaceMode -sdpAlignment` to `true` if you want to convert the flop clustering information into SDP format after placement with flop clustering is complete.
5. Run `placeDesign`.

## SDP Alignment

To enable SDP alignment, you must set `setPlaceMode -sdpAlignment` to `true`. The SDP alignment process comprises four sub-steps:

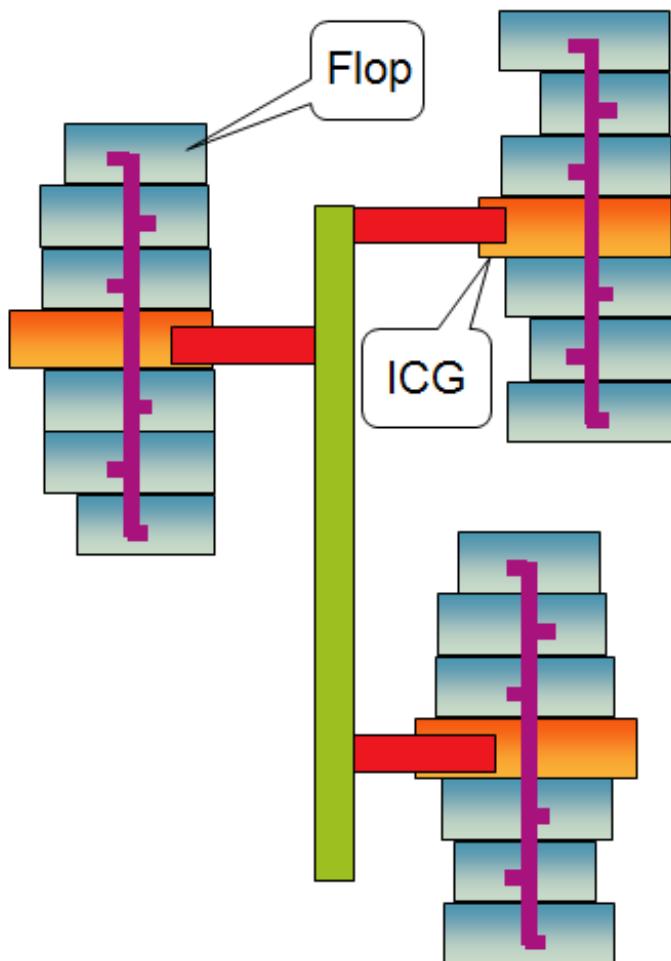
1. **SDP Generation:** If `-groupFlopToGate` is enabled, `placeDesign` automatically generates SDP groups in the current design based on their physical locations while honoring fanout limit and bounding box clustering limit.
2. **SDP Legalization:** After SDP generation, `placeDesign` internally calls SDP legalization to resolve overlaps between the flop columns (SDPs) and preplaced cells/macros.
3. **Incremental Placement:** `placeDesign` then runs incremental placement to further improve QOR.
4. **Refine Placement:** `placeDesign` then internally calls `refinePlace` to resolve overlaps between standard cells.



You can use the following SDP modes to control SDP alignment. You must set these options before running `placeDesign`:

- Use `set_sdp_mode -max_move_distance` to specify the maximum distance that an SDP group can be moved away from its original cluster placement location to resolve overlaps. If you do not specify the `-max_move_distance` option, the tool uses `100* pitch` as the default value, where `pitch` is the first vertical routing pitch.
- Use `set_sdp_mode -max_move_action` to specify whether an SDP group will be deleted, converted to an instance group, or left at its original cluster placement location if overlaps cannot be resolved due to high utilization. The default option is to convert the SDP group to an instance group.

- During SDP generation, the flops with clock gate are placed in a column style to reduce bottom level net cap loading. By default, the flops are placed in two columns. Use `set_sdp_mode -num_column` to control the number of flip flop columns for each SDP group.
- Use `set_sdp_mode -clock_location` to specify where the ICG cell will be placed in an SDP register column. By default, the gate cell is placed at the center.



## Sample Use Model

Here's a sample use model for placement with flop clustering and SDP alignment:

```
#Restore the design.
```

```
restoreDesign dtmf.enc.dat dtmf_recv_core
```

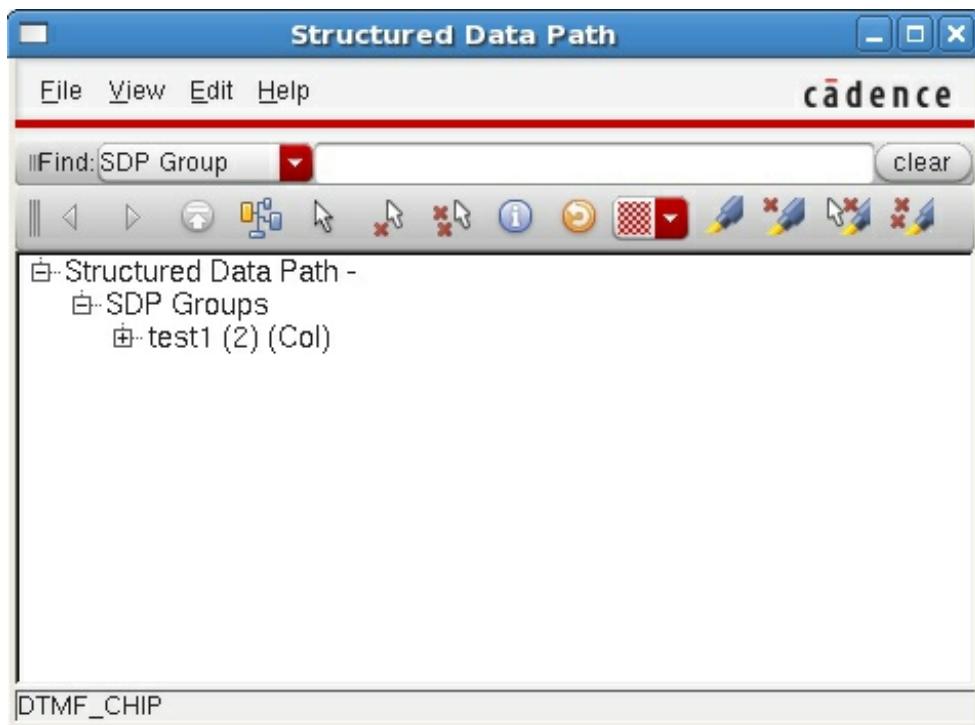
```
# Enable flop clustering placement.  
  
setPlaceMode -groupFlopToGate true  
  
# Specify flop clustering limit. This is an optional step. Default limit is 100.  
  
setPlaceMode -groupFlopToGateHalfPerim 200  
  
# Enable SDP alignment  
  
setPlaceMode -sdpAlignment true  
  
# Limit SDP movement from its original cluster location to 30 um for resolving  
overlaps. Delete SDP groups if their overlap cannot be resolved. This step is optional.  
  
set_sdp_mode -max_move_distance 30 -max_move_action delete_sdp_group  
  
# Place the design  
  
placeDesign
```

## Implementing SDP Capability

The SDP capability can be implemented using the SDP TCL commands and the SDP browser. Innovus provides a number of TCL commands for defining and manipulating data path structures and groups.

For more information, see the "[Structured Data Path Commands](#)" chapter of the *Innovus Text Command Reference*.

Innovus also provides the Structured Data Path browser (SDP browser) to help you manipulate SDP groups and/or elements and define data path and control logic using the graphical interface. To launch the browser, select *Floorplan - Structured Data Path* from the menu bar.



The SDP browser enables you to move around SDP rows and columns easily, simply by dragging and dropping.

For more information on using the SDP browser, see the [Floorplan Menu](#) chapter of the *Innovus Menu Reference*.

## SDP Relative Placement File

You can bring SDP information into the Innovus environment through an SDP relative placement file. The SDP file:

- Specifies relative placement information of data paths
- Supports hierarchical constructs, such as rows within a column or columns within a row
- Supports alignment, flipping, and orientation constraints
- Supports creation of empty rows and columns
- Supports wildcards (\*) and (?) for instance names
- Supports numeric bus bit range as part of an instance name. For example, specifying

`dataPath_reg[0:2]` is equivalent to specifying:

`dataPath_reg[0]`

`dataPath_reg[1]`

`dataPath_reg[2]`

Similarly, specifying `dataPath_reg<0:2>` is equivalent to specifying:

`dataPath_reg0`

`dataPath_reg1`

`dataPath_reg2`

The SDP file format also supports bit order sequence. So, specifying `dataPath_reg<2:0>` is equivalent to specifying:

`dataPath_reg2`

`dataPath_reg1`

`dataPath_reg0`

- Allows pre-place location

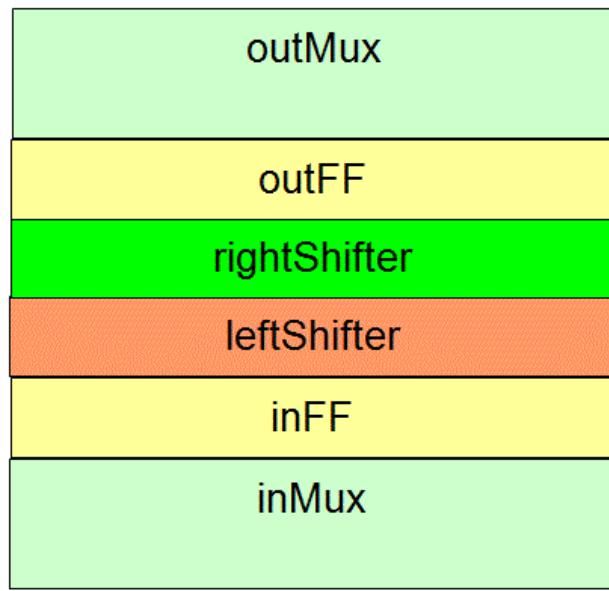
After design import, you can define relative placement information by reading in an SDP relative placement file using the `readSdpFile` command or by sourcing an SDP TCL script.

The SDP format provides you the ability to create rows or columns.

## SDP File Examples

Following is an example of an SDP file for creating a column of rows:

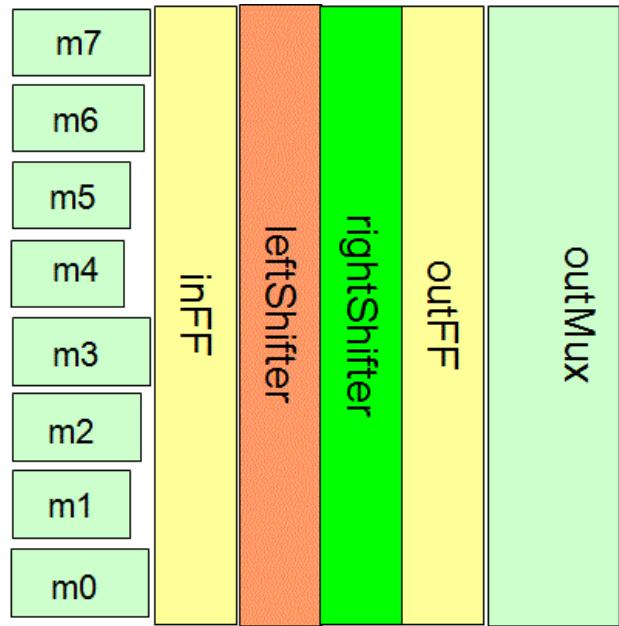
```
datapath sdp {  
Column adder {  
    justifyBy SW  
    row inMux { ... }  
    row inFF { ... }  
    row leftShifter { ... }  
    row rightShifter { ... }  
    row outFF { ... }  
    row outMux { ... }  
}  
}
```



SDP also supports hierarchical construction (SDP within SDP, rows within a column and vice versa), as shown below.

```

datapath sdp_row {
    row adder {
        justifyBy SW
        column inMux {
            inst m0
            inst m1
            inst m2
            inst m3
            inst m4
            inst m5
            inst m6
            inst m7
        }
        column inFF { ... }
        column leftShifter { ... }
        column rightShifter { ... }
        column outFF { ... }
        column outMux { ... }
    }
}
    
```



## SDP File Format

The SDP relative placement file has the following format:

```

alias var1 var2
datapath name {
    hierPath name
    origin X Y
    
```

```
row name {  
  
    [ orient R0 | MX | MY | R180 | MX90 | R90 | R270 | MY90]  
  
    [ justifyBy NW | SW | SE | NE | W | E | N | S | MID ]  
  
    [ flip X | Y | XY ]  
  
    [ skipSpace value [siteWidth siteName | micron]  
  
        | inst instanceName [orient R0|R90|..] [justifyBy ...] [flip X|Y|XY]  
  
        | column name { ... } ]...  
  
    [ spreadGroup value instanceName [sideWidth siteName | micron] ]  
  
}  
}  
  
}  
  
datapath name {  
  
    hierPath name  
  
    origin X Y  
  
    column name {  
  
        [ orient R0 | MX | MY | R180 | MX90 | R90 | R270 | MY90]  
  
        [ justifyBy NW | SW | SE | NE | W | E | N | S | MID ]  
  
        [ alignByPinName pin_list{W | E | MID} ]  
  
        [ flip X | Y | XY ]  
  
        [ skipSpace value [siteWidth siteName | micron]  
  
            | inst instanceName [orient R0|R90|..] [justifyBy ...] [flip X|Y|XY]  
  
            | row name { ... } ]...  
  
        [ spreadGroup value instanceName [sideWidth siteName | micron] ]  
  
    }  
}  
  
# is used for comment
```

```
# Keywords like row, column can be redefined using alias command.
```

The format uses the following keywords:

- **alias**: Can be used to redefine keyword name of **row**, **column**, **justifyBy**, **skipSpace**, **hierPath**, **origin**, **flip**, **datapath**, and **inst**.
- **datapath**: Specifies the name of a data path structure.
- **hierPath**: Specifies the hierarchical path name of a data path structure.
- **origin**: Specifies the lower left location of a data path structure.
- **row**: Specifies the name of an SDP row group. The name should be unique within same data path group or across different data path group. If the name is not unique, the tool automatically adds an index to the specified name to make it unique. For example, if the specified name is `SdpGroup`, the modified name will be `SdpGroup_id`.
- **orient**: The orientation of an SDP group or an SDP element. Values can be `R0`, `R90`, `MY`, `MX`, etc. If this orientation is specified at SDP group level, the orientation will be applied to instances that belonged to this SDP group.
- **justifyBy**: Specifies the anchor point that will be used for aligning a SDP group/element. If the information is not specified then the default value `SW` will be used. If the `justifyBy` constraint is not specified at that level, this constraint will be inherited from its parent level.

The following examples illustrate `justifyBy`.

Example 1:

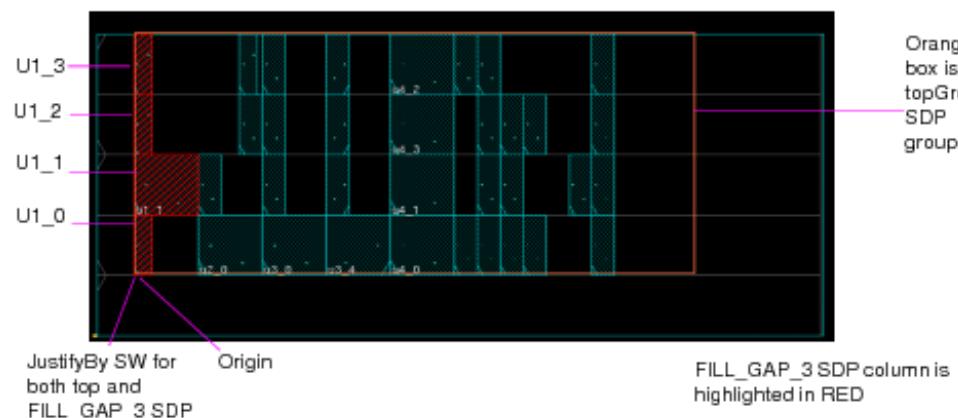
```
datapath topGroup {  
    origin 2.56 4.8  
    row top {  
        justifyBy SW  
        column FILL_GAP_3 {  
            justifyBy SW  
            inst u1_<0:3>  
        }  
        ...  
    }  
}
```

```

        }
        ...
    }
    ...
}

```

SDP topGroup has the origin at {2.56 4.8}. This topGroup SDP has more than one row with anchor point for alignment is SW. First row is a column with anchor point for alignment is SW.

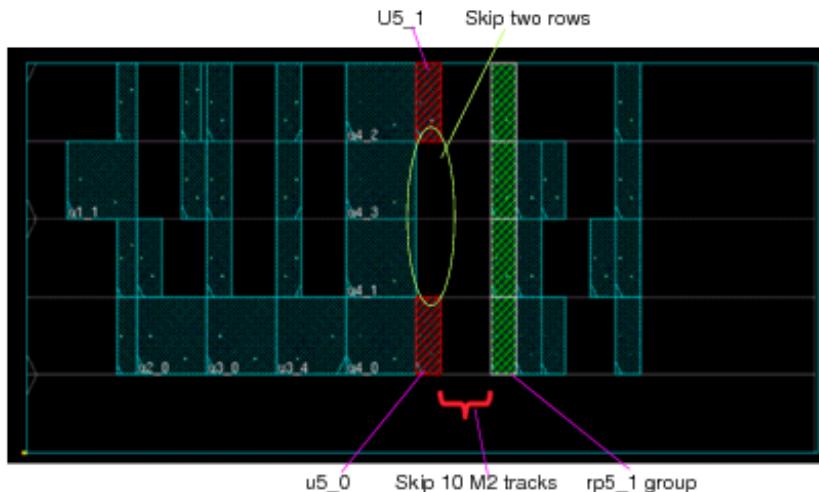


- **alignByPinName:** Specifies the pin names by which SDPs are to be aligned. If SDP instances are aligned by pins, the router can chalk a straight route to connect all instances and thus minimize routing. Pin alignment control is typically used for clock pins of high speed register columns.
- **flip:** Flips the SDP group or an SDP instance element in vertical, horizontal, or both directions. Possible values can be **x**, **y**, or **xy**.
- **column:** Specifies the name of an SDP column group. Name should be unique within same data path group or across different data path group. If the name is not unique, the tool automatically adds an index to the specified name to make it unique.
- **skipSpace:** Specifies a space value to be skipped. By default if the `skipSpace` value is defined in a column, then this value is for row skipping and represents the number of skipped rows. If the `skipSpace` value is defined in a row, then this value is for column skipping and represents the number of M2 tracks (pitch of first vertical layer). `skipSpace` value can also be

specified in micron units or as number of widths of a specified row site. However, row site width should be specified only for an SDP row, and not for an SDP column.

Following is the example of skipSpace:

```
...
column FILL_GAP_9 {
    justifyBy SW
    row main_5 {
        justifyBy SW
        column rp5_0 {
            justifyBy SW
            inst u5_0
            skipSpace 2 # Skip 2 rows
            inst u5_1
        }
        skipSpace 10 # skip 10 M2 tracks
        column rp5_1 {
            justifyBy SW
            inst u5_<2:5>
        }
    }
}
...
```



- **siteWidth:** Specifies the name of the tech site. Use the number of widths of the specified site as skip unit in horizontal direction.
- **micron:** Specifies that the specified **skipSpace value** is to be interpreted as distance in microns (um).
- **inst:** Specifies one or more instance names. Use this keyword if you want to create the SDP group from specified instances.
- **spreadGroup:** Inserts space between group of instances. If **spreadGroup** is specified in a column, spacing will be the number of added rows. If **spreadGroup** is specified in a row, spacing will be the number of placement grids added between the columns in that row. **spreadGroup** value can also be specified in micron or number of widths of a specified site.

In the following example, **spreadGroup** is defined inside a column SDP so that a row will be skipped between specified instances:

```
column sdp_grp1 {
    justifyBy SW
    spreadGroup 1 reqPath/reg_busBit<0:17>
}
```

This is equivalent to:

```
column sdp_grp1 {
```

```
justifyBy SW

reqPath/reg_busBit0

skipSpace 1

reqPath/reg_usBit1

skipSpace 1

...
}

...
...
```

## Reusing SDP Instantiations

Starting from Release 12, the SDP file format will support reuse capability. You must specify the data path macro definition with the **define** keyword before it is used anywhere. The SDP macro definition can be specified in a SDP file different than the data path that references it. The content of the macro definition must have the same syntax as the existing row or column data path.

```
define SDP_macro_name {

    normal_dataPath_syntax

}
```

Once specified, you can instantiate or use the macro definition in a data path specification by using the **use** keyword:

```
use S DP_macro_name row_or_column_name [hierPathName]
```

Here:

- *SDP\_macro\_name*: Specifies the name of a data path macro definition.
- *hierPathName*: Specifies the path name. If a data path has specified the **hierPath** information, then, this specified path name is concatenated to the data path hierarchical path.
- *row\_or\_column\_name*: Specifies the user-specified name of the instantiated row or column. If a column is instantiated inside a column, the SDP reader automatically creates a row between these two columns and vice versa.
- *normal\_dataPath\_syntax*: Specifies a row or column data path:

```
row name {

    [ orient R0|R90 |... ]
}
```

```

[ justifyBy NW|SW|SE|NE|W|E|N|S|MID ]
[ flip X|Y|XY ]
[ skipSpace rowVal x colVal
  | inst instanceName [orient R0|R90|...] [justifyBy ...]
    [flip X|Y|XY]
  | column name { ... }
  | use SDP_macro_name column_name [ hierPathName ] ]
}

```

Or

```

column name {
  [orient R0|R90|...]
  [ justifyBy NW|SW|SE|NE|W|E|N|S|MID ]
  [ flip X|Y|XY ]
  [ skipSpace rowVal x colVal
    | inst instanceName [orient R0|R90|...] [justifyBy ...]
      [flip X|Y|XY]
    | row name { ... }
    | use SDP_macro_name row_name [ hierPathName ] ]
}

```

The following example illustrates how data path instantiations can be reused.

```

define entry1 {
    row row_0 {
        inst g1 orient R0
        inst CKGA orient MY
    }
}

define entry2 {

```

```
row row_1 {  
    inst CKGA_dcap0 orient MX  
    inst CKGA_dcap1 orient MX  
    column nested_col {  
        use entry1 row_0_1 inst1  
        use entry1 row_0_2 inst2  
    }  
}  
}
```

```
datapath DP_one {  
    hierPath PTN1  
    column col_arr {  
        justifyBy SW  
        skipSpace 1  
        inst trn_1 orient R0  
        use entry1 row_3_4 top0  
        use entry2 row_3_5 top_1  
    }  
}
```

Here, data path *DP\_one* is equivalent to:

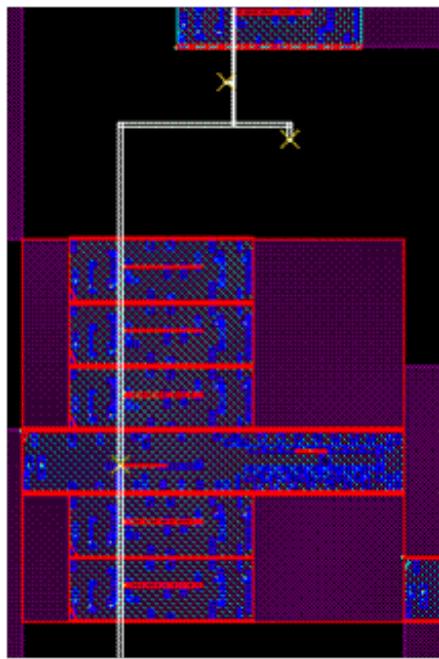
```
datapath DP_one {  
    hierPath PTN1  
    column col_arr {  
        justifyBy SW  
        skipSpace 1  
        inst trn_1 orient R0  
        row row_3_4 {
```

```
inst top0/g1 orient R0
inst top0/CKGA orient MY
}

row row_3_5 {
    inst top_1/CKGA_dcap0 orient MX
    inst top_1/CKGA_dcap1 orient MX
    column nested_col {
        row row_0_1{
            inst top_1/inst1/g1 orient R0
            inst top_1/inst1/CKGA orient MY
        }
        row row_0_2{
            inst top_1/inst2/g1 orient R0
            inst top_1/inst2/CKGA orient MY
        }
    }
}
```

## Aligning SDPs by Pins

In addition to controlling the alignment of SDP columns by using the **justifyBy** constraint, you can align SDP instances by pins so that the router can chalk a straight route to connect all instances and thus minimize routing. Pin alignment control is typically used for clock pins of high speed register columns.



Pin alignment control is supported by the SDP file format as well as the TCL command:

- SDP file format: The pin alignment constraint can be specified in an SDP file by introducing the **alignByPinName** keyword to the column SDP syntax as follows:

```
column name {  
    [justifyBy NW | SW | SE | NE | W| E | N | S | MID]  
  
    [alignByPinName pin_list{W | E | MID}]  
  
    ...}
```

**Note:** You can specify alignment by pin only for a column SDP group.

- TCL command: The existing [createSdpGroup](#) command has also been enhanced to support the align by pin name feature. Use the new `-alignByPinName {pinList}` and `-side w|e|mID` parameters to specify the pin names by which you want to align SDPs and the pin edge to be used for alignment.

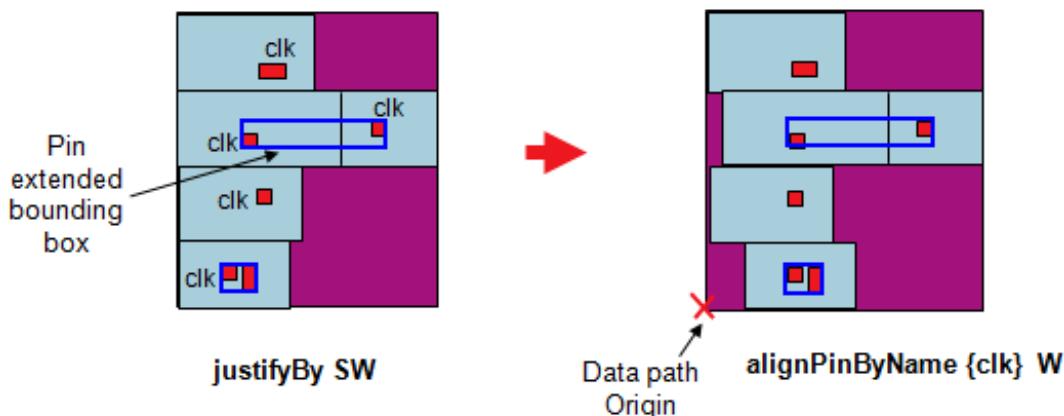
**Note:** You can specify `-alignByPinName` only for a column SDP group (`-type col`).

Pin alignment control:

- If specified pin has more than one physical shape, the pin bounding box that is used for

alignment is the extended bounding box of all matched pin shapes.

- If more than one pin is specified, the extended bounding box that covers all pin shapes of specified pins is used.
- If a row within a column has more than one instance, the extended bounding box of all matched pin shapes of all instances within the row is used.



The existing `readSdpFile` and `writeSdpFile` commands read and save the new **alignByPinName** syntax, respectively. The `readSdpFile` command issues a warning message if the **alignByPinName** constraint is specified for an SDP row group and ignores the constraint for row groups.

The `placeSdpGroup` command also honors the **alignByPinName** constraint.

## Setting SDP Options

The `set_sdp_mode` command allows you to set SDP related sticky options before using commands like `placeSdpGroup` and/or `placeDesign`. You can use the `set_sdp_mode` command to:

- Disable extension of core boundary  
If you set the `-disable_extended_core` parameter of `set_sdp_mode` to `true` before running the `placeSdpGroup` command, the software does not adjust the core boundary to accommodate all SDP placements. Instead, the software places the SDP elements outside the core boundary if they do not fit within the boundary.
- Resolve overlaps between SDP members and specified row/column cells  
In traditional SDP flow, if you specify the rows and columns to be skipped by setting the -

`resolveOverlapRowCell` and `-resolveOverlapColumnCell` parameters, `placeSdpGroup` automatically resolves overlaps in both X and Y direction while placing SDP groups.

- Maintain SDP alignment during design optimization

The `-honor_alignment` parameter is used to control the `refinePlace` step of the `optDesign` command. By default, `refinePlace` ignores the `justifyBy` constraints of SDP when finding a legal location for new added/modified instances. As a result, SDP instances may no longer be aligned after optimization. However, if you set the `-honor_alignment` parameter, `refinePlace` is forced to honor existing `justifyBy` constraints.

- Generate a detailed SDP placement report

If you specify `set_sdp_mode -place_report file_name`, the software generates a detailed SDP placement report in the following cases:

- On running `placeSdpGroup`
- On running `planDesignWith -useSdpGroup`:

```
setPlanDesignMode -useSdpGroup true  
planDesign
```

- On running `placeDesign` with `-sdpAlignment` of `setPalceMode` set to `true`

The detailed SDP placement report contains the following information:

- Standard output file header
- Summary report at the end of the report file such as:
  - Total number of SDPs in the design
  - Total number of placed SDPs
  - Total number of unplaced SDPs
  - Number of overlapped SDPs.

Following is an example of an SDP placement report file:

```
#####  
# Generated By: Cadence Innovus 15.10-b024_1  
# Generated on: Tue Apr 7 08:48:11 2015  
# HostName: rlno-leenap  
# Design: DTMF_CHIP
```

```
# Command: placeSdpGroup
#####
SDP Name      Location
=====
```

```
Inst1,      (276.340, 276.340)
```

```
Inst2,      (276.340, 276.340)
```

```
Total Number of SDPs: 2
```

```
Total Number of placed SDPs: 2
```

```
Total Number of Unplaced SDPs: 0
```

**Note:** You can use the `get_sdp_mode` command to retrieve the current values of `set_sdp_mode` options.

## Optimizing a Design with SDPs

After importing a placed SDP design and setting SDP-related sticky options, you need to optimize the design. During design optimization with `optDesign`, the `refinePlace` command honors SDP instances and handles them properly. This enables you to optimize a design with SDPs while honoring SDP order and alignment constraints.

To better support SDPs, the size-only file specified with the `setOptMode -sizeOnlyFile` option has been enhanced to support two additional attributes for a specific instance. The two attributes are:

- `-noMoveInst`: Use the `-noMoveInst` attribute to disable move instance transform during optimization. This option is recommended for an SDP instance.
- `-preserveCellHeight`: Use `-preserveCellHeight` to maintain existing cell height.

Here's a sample of a size-only file:

```
tdsp_core/inst1 -noMoveInst -preserveCellHeight
tdsp_core/inst2 -noMoveInst -preserveCellHeight
tdsp_core/inst3 -noMoveInst -preserveCellHeight
tdsp_core/inst4 -noMoveInst
tdsp_core/inst5 -noMoveInst
tdsp_core/inst6 -noMoveInst -preserveCellHeight
tdsp_core/inst7 -noMoveInst -preserveCellHeight
tdsp_core/inst8 -noMoveInst -preserveCellHeight
```

Following is a sample of the optDesign flow for traditional SDP:

```
restoreDesign init.enc.dat dtmf_recv_core
```

```
readSdpFile -file dtmf.sdp
```

```
# Run planDesign to place SDPs and blocks in the design
```

```
setPlanDesignMode -useSdpGroup true
```

```
planDesign
```

```
# Run placeSdpGroup to resolve overlaps between SDPs and tap/power switch cells. This step is optional.
```

```
setSdpMode -resolve_overlap_column_cell {tapCell1 tapCell2} -resolve_overlap_row_cell
switch_cell
```

```
placeSdpGroup
```

```
# Place the rest of standard cells
```

```
placeDesign
```

```
# Run pre-CTS optimization
```

```
setSdpObjectStatus -status placed -all
```

```
setOptMode -sizeOnlyFile dtmf.sizeOnly
```

```
optDesign -preCTS
```

## Checking SDP Placement

After you perform any manual editing and/or optimization step, you may want to check for any resulting SDP violations before you move on to the next step in your flow. Innovus provides the `check_sdp_group` command that enables you to check current SDP placement against the SDP constraints that you may have originally specified via an SDP relative placement file or a set of TCL commands.

Using `check_sdp_group`, you can check whether you need to resolve SDP overlapping or re-run SDP placement before moving to the next step in the flow. `check_sdp_group` checks the following:

- SDP group-instance orientation
- SDP group-instance `justifyBy` constraint
- Alignment by pin name constraint
- SDP group-instance flip constraint
- Skip space constraint
- SDP group-instance overlapping

Using the `-file` option of `check_sdp_group`, you can generate a detailed report that contains the following information for each of the above checking categories:

- Total number of violations
- List of SDP group-instance names that have that violation

By default, the report file name is `designName.checkSdp.rpt`.

```
# Generated By: Cadence Innovus 15.10-b024_1# Generated on: Tue Apr 7 08:48:11 2015#
HostName: rlno-leenap# Design: DTMF_CHIP
```

# Bus Planning

- [Overview](#)
- [Bus Planning Flow in Innovus](#)
- [Creating a Bus Guide](#)
  - [Using the Edit Bus Guide GUI](#)
    - [Drawing a Bus Guide](#)
  - [Using Text Commands](#)
  - [Example](#)
- [Moving and Stretching a Bus Guide](#)
- [Cutting, Splitting, and Merging Bus Guides](#)
- [Customizing the Bus Guide Display](#)
  - [Highlighting and Dehighlighting the Bus Guide](#)
- [Saving and Restoring Bus Guide Information](#)
- [Verifying Bus Guide](#)
- [Limitations of Bus Planning](#)

## Overview

The Bus Planning feature in the Innovus software enables you to plan and create bus guides which are used to guide the path of busses for floorplanning, partition pin optimization, feedthrough insertion, congestion prediction in trialroute, and final routing in nanoroute.

Most designs need bus planning for estimating the design size and routing channel widths. Without bus guides, the routers do not route all the bus bits together on the desired path. Routing the bus bits outside the desired path can have high cost implications. Hence it is very important to accurately plan the bus guide layouts.

Bus planning is critical in the prototyping stage of the hierarchical flow. Use the bus planning capability to guide the path of bus routing for feedthrough insertion, partition pin optimization, and congestion prediction. If you are in the implementation stage, use bus planning to guide the path of busses for detailed routing.

## Bus Planning Flow in Innovus

For hierarchical designs, you create bus guides before or after assigning the partition/black box pins. For flat or top-level designs, you create bus guides before routing. Normally, you create bus guides before pin assignment.

The following steps describe the bus planning flow in Innovus:

1. Importing the design

Import the design into the Innovus environment.

2. Floorplanning the design

If the design is a partition design, then specify partitions. For more information, see "Specifying Partitions and Blackboxes" section in the [Partitioning the Design](#) chapter of the *Innovus User Guide*.

If it is a black box design then define black boxes and specify their sizes. You can manually preplace black boxes/macros or run `planDesign` to automatically place them. Further, adjust the floorplan if needed.

3. Defining net groups

Group the bus bit nets together as net groups using `createNetGroup` and/or `addNetToNetGroup` commands.

4. Creating bus guides

Create bus guides associated with the net groups, to guide routing for all the nets of the specified net group. Bus guides can be created using the Edit Bus Guide form, which can be accessed from the [Edit Menu](#) and/or the `createBusGuide` command. See [Creating a Bus Guide](#).

5. Placing the design

Place the standard cells. If you do not want the Innovus placer (`placeDesign`) to move your macros and/or black boxes, set their placement status to `fixed` before running placement.

**Note:** This is an optional step for designs that do not have standard cells at full-chip level.

6. (Optional) Routing the design

Run `trialRoute` to route the design.

7. (Optional) Inserting feedthrough buffers

Feedthrough can be inserted based on routing or placement. If `trialRoute` was run before this step, then feedthroughs are inserted based on routing. For more information, see the "Inserting Routing Feedthroughs" section of the [Partitioning the Design](#) chapter of the *Innovus User Guide*.

## 8. Assigning pins

Assign pins using `assignPtnPin` command.

## 9. Committing partition

Commit partitions using `partition` command.

## 10. Saving Partition

Save the partition information using `savePartition` command.

## 11. Running NanoRoute at the top-level design

Perform detailed routing using the `NanoRoute router` at the top-level design.

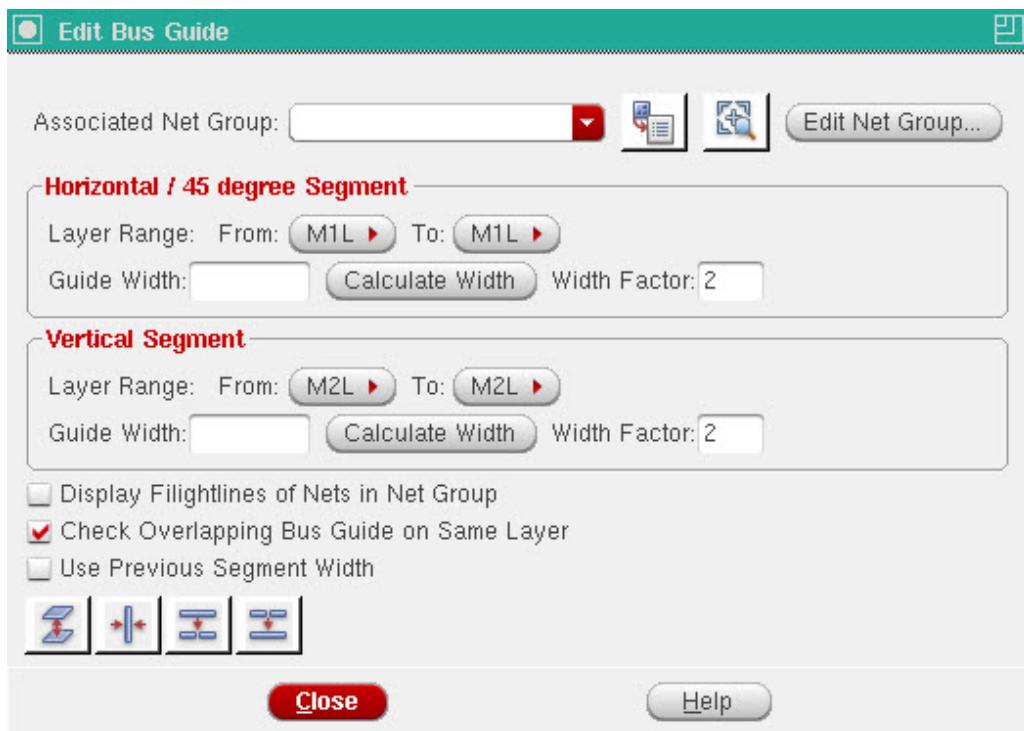
# Creating a Bus Guide

A bus guide consists of one or more overlapping segments. It must always be associated with a net group. So, before creating a bus guide you must define a net group. Remember that a net group can either be assigned to a bus guide or a pin guide, but not to both. For each bus guide segment that you create, you must specify a layer or a layer range.

You can create a bus guide [Using the Edit Bus Guide GUI](#) and/or [Using Text Commands](#).

## Using the Edit Bus Guide GUI

The bus guide editor in Innovus allows you to create bus guides before or after assigning the bus pins. Using the *Edit Bus Guide* form, you can edit the bus guide properties and interactively create the bus guide for a specific net group.



You can specify the net group associated with the bus guide you are creating in *Associated Net Group* drop-down list. This is an editable drop-down list, which lists all net groups by default. You can either select the required net group from the list or type the name of the net group in the box. As you start typing the name of the net group, the drop-down list changes to show only the net group names that match the pattern.

In addition to the net group, you can specify the layer or layer range on which the bus guide is to be created and the width of the bus guide segment.

By default, the bus guide editor derives the default minimum guide width required to hold all the nets assigned to the bus guide. If the bus guide connects to placed pins on block edges, the bus guide editor automatically adjusts the width of the guide segment to cover all the pins of nets in the net group. The bus guide editor provides options to enable overlapping check for bus guides created on a specific layer and display flight lines of nets in the net group, when creating the bus guides.

The Edit Bus Guide form can also be used for net group to bus guide cross-probing:

- Finding the net associated with the bus guide selected in the floorplan: Use the *Get Selected* ( button to find the name of the net group associated with the bus guide segment selected in the floorplan. When you click the *Get Selected* button, the *Associated Net Group* drop-down list jumps to the net group of the selected bus guide segment.
- Finding the bus guide of a specific net group: Use the new *Zoom Selected* ( button to zoom to the bus guide of the net group selected in the *Associated Net Group* drop-down

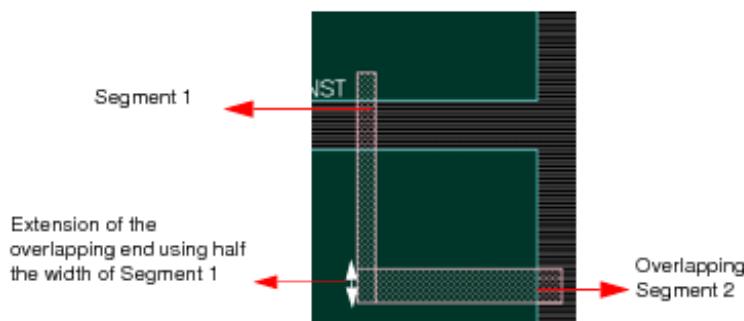
list. When you click the *Zoom Selected* button, the view window will zoom to fit the bus guide of the net group.

For more information on the *Edit Bus Guide* form, see *Edit - Object - Edit Bus Guide* in the [Edit Menu](#) chapter of the *Innovus Menu Reference*.

## Drawing a Bus Guide

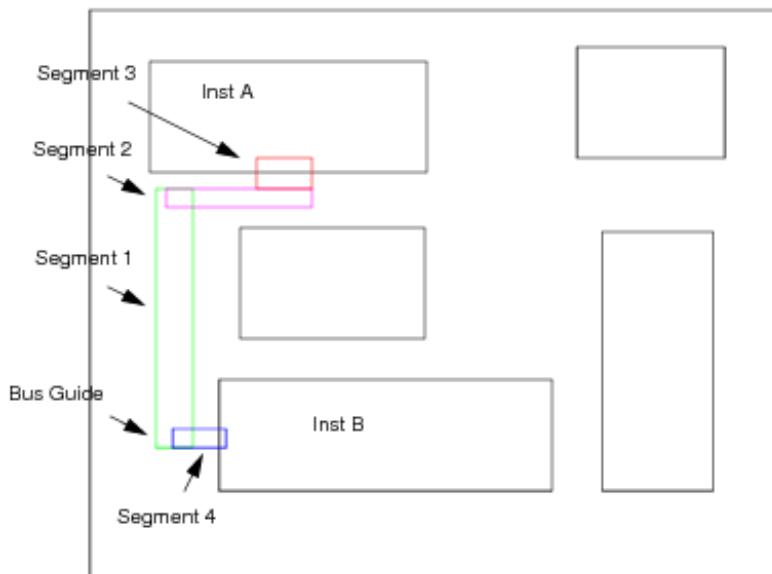
To draw a bus guide in Innovus, you must first click the *Edit Bus Guide* icon in the toolbar.

Once you are in the bus planning mode, you can draw the bus guide segment by clicking the left mouse button and dragging it along the points of center line for the guide segment. To end a bus guide segment, double-click the left mouse button. By default, the bus guide extends half width for the overlapping end of the created segment. However, if the guide segment overlaps with another segment that has bigger or smaller width, the bus guide editor uses half the width of the other segment for the extension of the overlapping end.



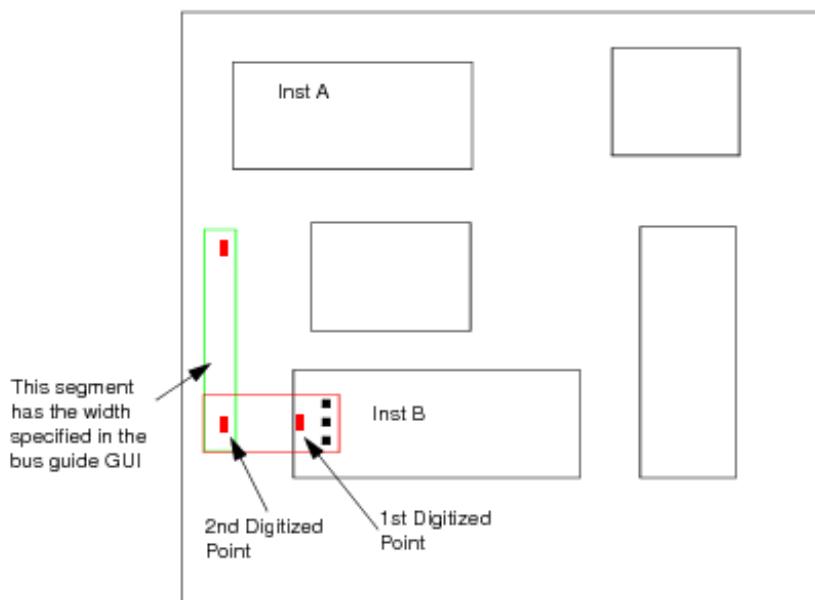
**Note:** All the segments of the bus guide should overlap to ensure continuity; Otherwise, the router (nanoroute) may create routing problems or may take longer time to run.

You can specify a new segment connected to an existing segment as shown in the following image where segment 4 overlaps with segment 1:



**Figure 1**

You can also draw a bus guide segment that connects to the placed pins of the associated net group.



**Figure 2**

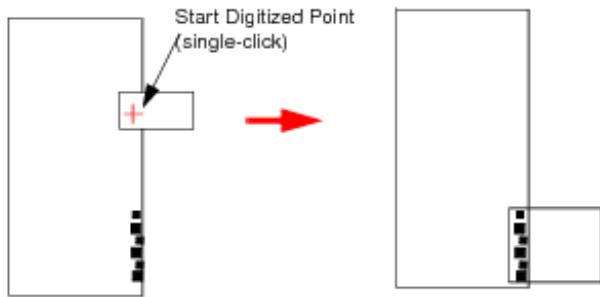
If you click on the partition boundary side where the pins are placed, the bus guide editor automatically snaps to these pins. If the width value specified in the bus guide editor is smaller than the width required to fully cover all these pins, the bus guide editor derives new width for the guide segment such that all the associated physical pin geometries are covered. If the width value is bigger than the width that needs to cover all pins, the editor will use the current width value without adjusting it.

In Figure 2, the width of the segment defined by the first and the second digitized points is derived

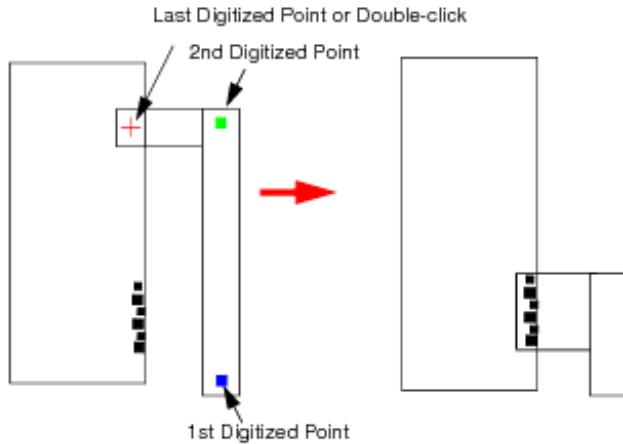
based on the placed pin information such that the segment width can fully cover the all the pins. The width of the next segment (defined by second and third points) is the width that is specified in the bus guide editor.

The snapping of bus guides to pins (partition or black box pins) occur at the start or at the end of the bus guide, when you double-click to end the bus guide.

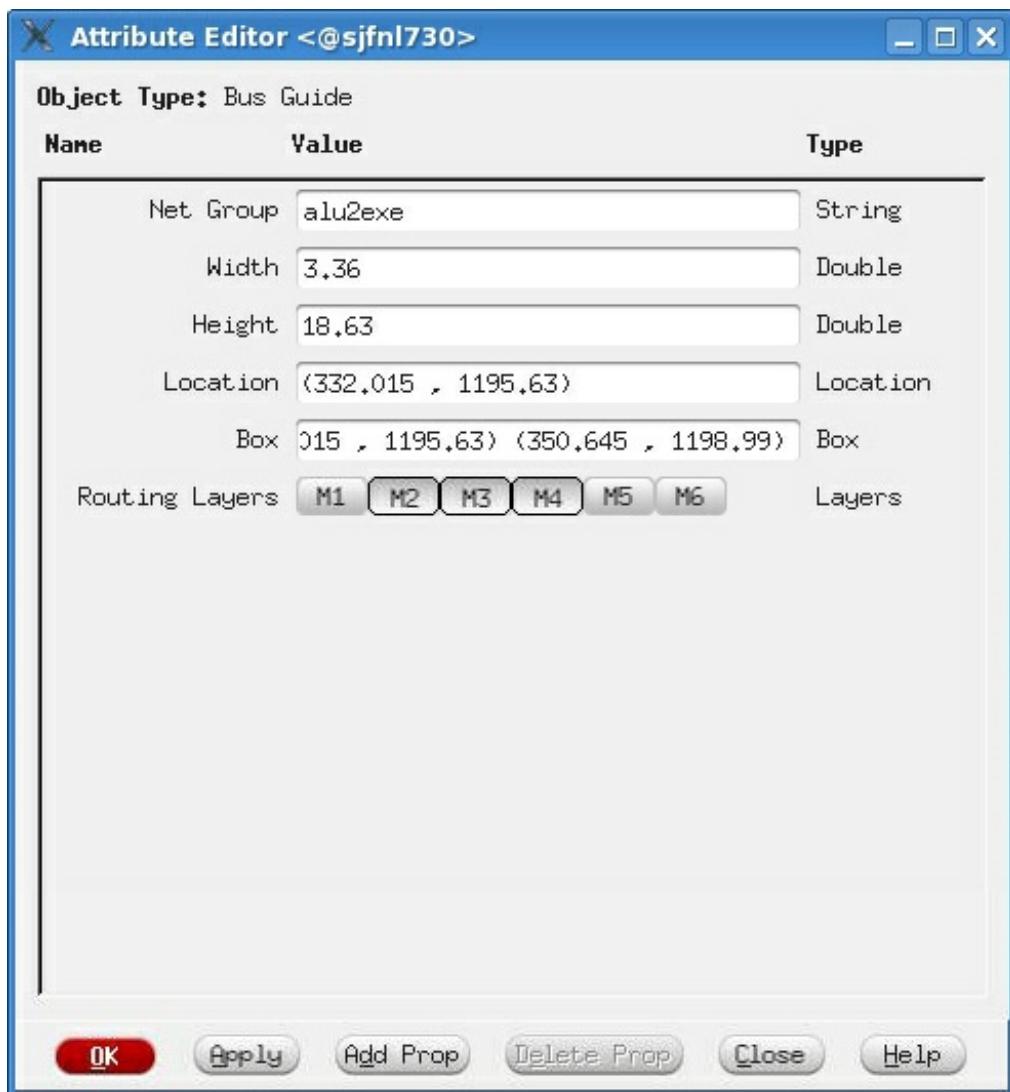
The following example illustrates the snapping behavior at the starting digitized point. The snapping occurs before you specify the second point:



The following example illustrates the snapping behavior at the end of a bus guide.



To view the attributes of a bus guide that you created, double-click the bus guide segment to display the Attribute Editor as shown in the following example:



A bus guide gets deleted when you delete its associated net group.

## Using Text Commands

You can create and edit bus guides using the following text commands:

| Commands       | Use                           |
|----------------|-------------------------------|
| createBusGuide | To create a bus guide segment |

|                                    |                                                                                                                                                                   |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>deleteBusGuide</code>        | To delete a bus guide<br><br><b>Note:</b> You can also delete a bus guide segment by selecting the segment and pressing the <code>Del</code> key on the keyboard. |
| <code>deselectBusGuide</code>      | To deselect a bus guide segment                                                                                                                                   |
| <code>editCutWire</code>           | To cut a bus guide segment                                                                                                                                        |
| <code>editMerge</code>             | To merge two bus guide segments                                                                                                                                   |
| <code>editSplit</code>             | To split a bus guide segment                                                                                                                                      |
| <code>selectBusGuide</code>        | To select a bus guide segment                                                                                                                                     |
| <code>selectBusGuideSegment</code> | To select a bus guide segment with its specified bounding box                                                                                                     |
| <code>update_bus_guide</code>      | To modify the layer or width information of an existing bus guide                                                                                                 |

For more information on the commands, see the "Bus Plan Commands" chapter in the *Innovus Text Command Reference*.

The following example describes the steps to create bus guides using text commands.

## Example

This sample script creates two bus guides for two bus nets, `abcBusNet` and `cdeBusNet`. The `abcBusNet` bus has 32 bus bits and `cdeBusNet` has 100 bus bits. Two net groups, `abcNetGroup` and `cdeNetGroup` are defined for `abcBusNet` and `cdeBusNet` busses, respectively. Two bus guides are used to guide routing for these two busses for feedthrough insertion:

```
#Restore the bBoxFP.enc.dat design of top cell Test that is already being floorplanned
restoreDesign bBoxFP.enc.dat Test

#Create net groups for busses abcBusNet and cdeBusNet
createNetGroup abcNetGroup -net abcBus*
createNetGroup cdeNetGroup -net cdeBus*
```

```
#Create bus guide for bus net abcBusNet[0..31]. This bus guide has 4 segments.

createBusGuide -netGroup abcNetGroup -centerLine 4421.8 10749.36 4960.8 10749.36 -width 90 -layer
Metal4:Metal8

createBusGuide -netGroup abcNetGroup -centerLine 4900.8 10809.36 4900.8 9470 -width 90 -layer
Metal3:Metal7

createBusGuide -netGroup abcNetGroup -centerLine 4840.8 9530.0 11525.4 9530.0 -width 90 -layer
Metal4:Metal8

createBusGuide -netGroup abcNetGroup -centerLine 11465.4 9590.0 11465.4 9203.5 -width 90 -layer
Metal3:Metal7
```

```
#Create bus guide for net cdeBusNet[0..99] that has only one vertical segment.
```

```
createBusGuide -netGroup cdeNetGroup -centerLine 15300.7 7061 15300.7 11230 -width 300 -layer
Metal5:Metal7
```

```
#Place the design since design has some top-level cells
```

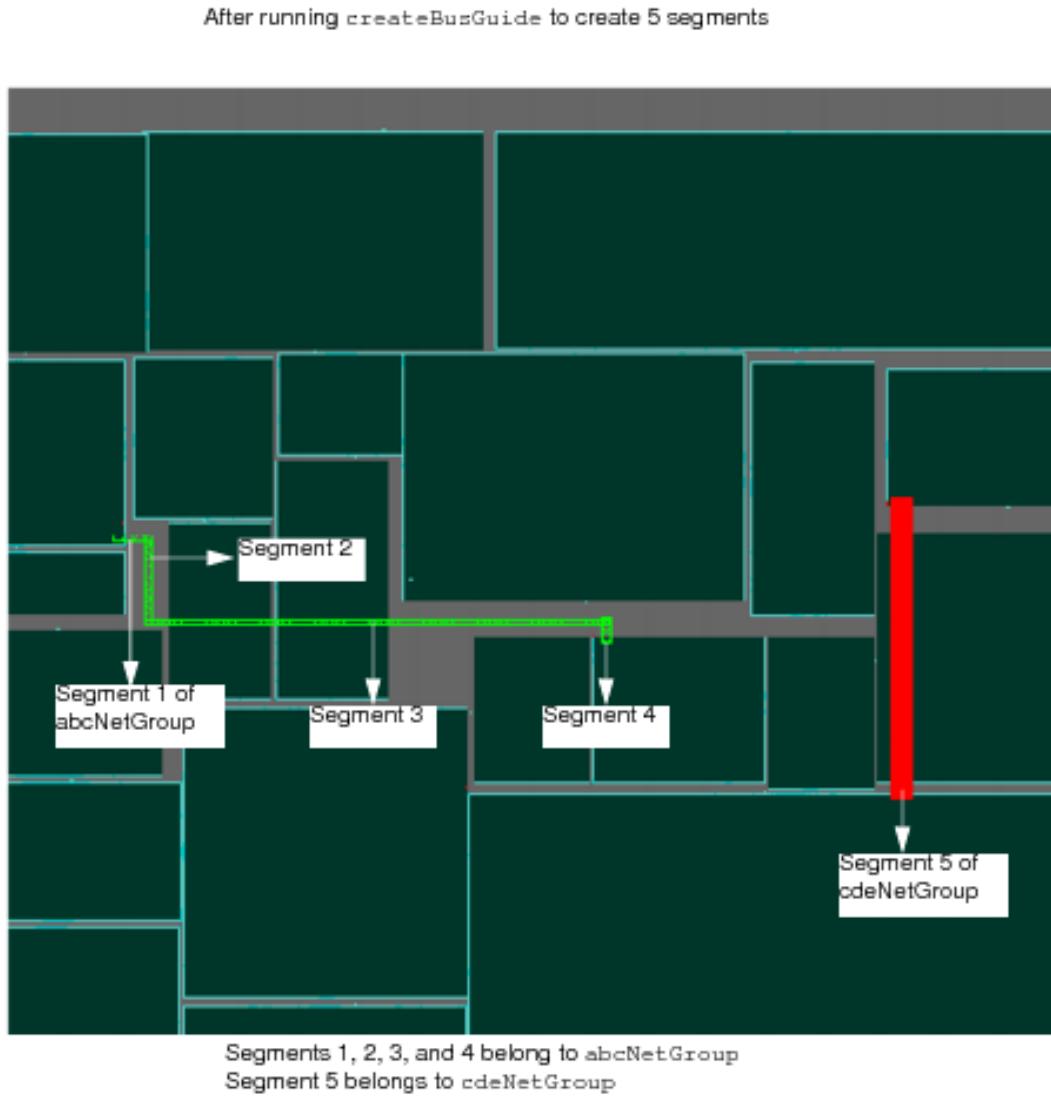
```
placeDesign
```

```
#Run trialRoute with option -printWiresOutsideBusguide to report any nets that are routed outside
specified bus guide areas
```

```
trialRoute -printWiresOutsideBusguide
```

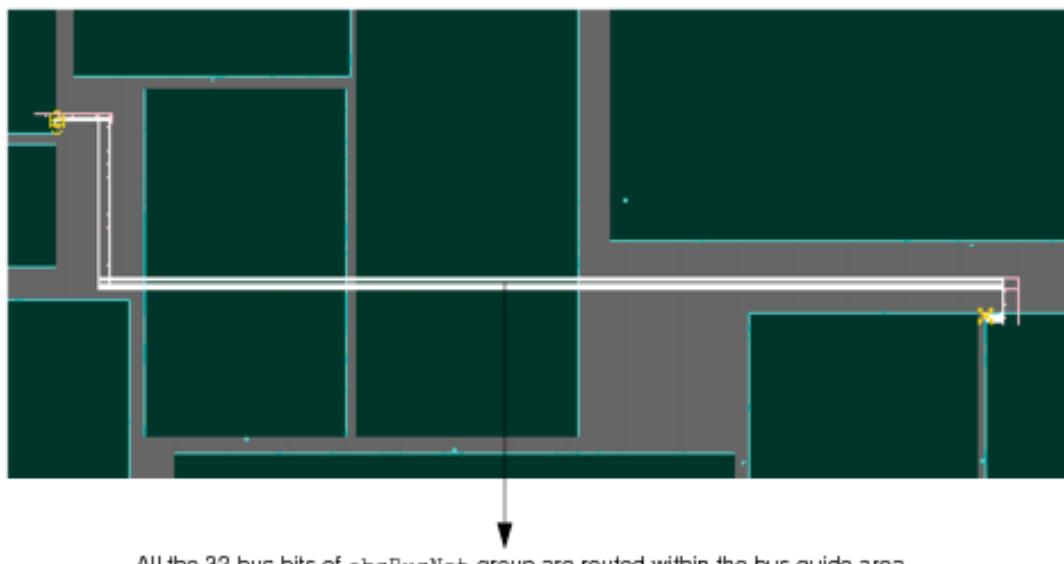
```
#Continue with the normal flow, invoking feedthrough insertion, pin assignment, and so on...
```

The following figure displays the bus guide associated with the net group `abcNetGroup`, highlighted in *green*, and the bus guide associated with the net group `cdeNetGroup`, highlighted in *red*:



The following figure displays the routing of the bus `abcBusNet [0...31]`, routed within the bus guide area:

After running the `placeDesign` and `trialRoute`



All the 32-bus bits of `abcBusNet` group are routed within the bus guide area.

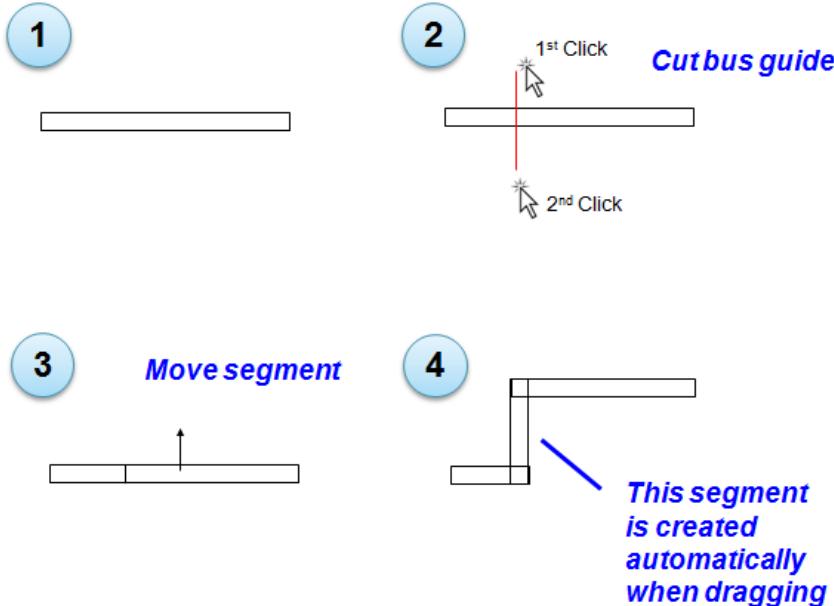
## Moving and Stretching a Bus Guide

You can use the *Move Wire* ( ) widget to move a bus guide segment, in the same way as you move a wire. The bus guide connection is maintained while moving a segment. Similarly, you can use the *Stretch Wire* ( ) widget to stretch a bus guide. When you do so, it auto snaps to connect fully with other segments in the bus guide.

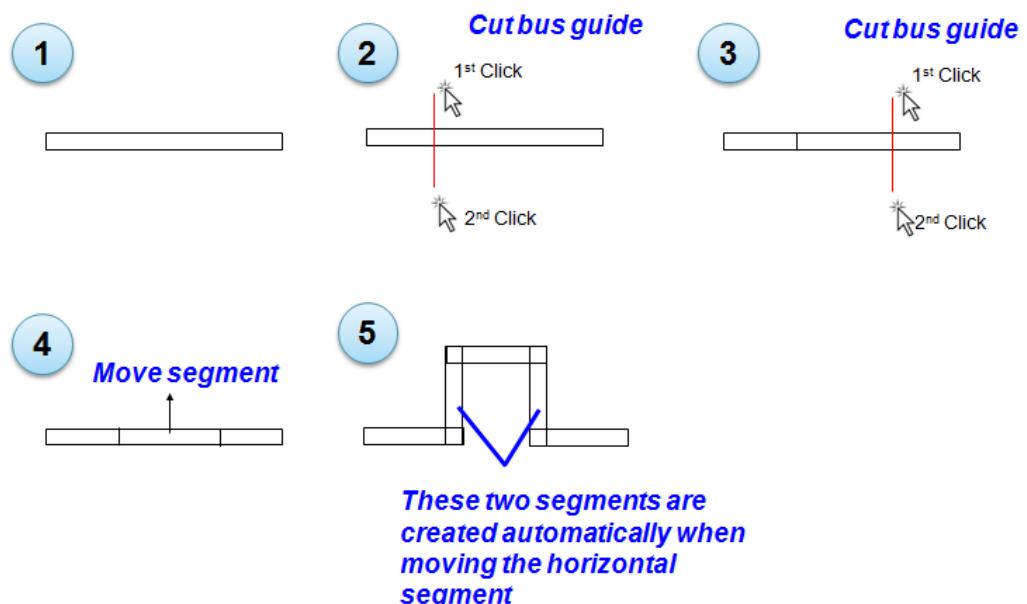
## Cutting, Splitting, and Merging Bus Guides

While editing bus guides, you might sometimes need to flip a corner of a bus guide or make a jog or detour on the bus guide. Jogs and detours are needed when the bus guide runs into a obstruction or you want to make a bus guide longer. To make jogs and detours, you would need to adjust bus guides by cutting and merging bus guide segments. Cutting and merging bus guides is possible through the `editCutWire`, `editSplit`, and `editMerge` wire edit commands. These commands support cutting, splitting, and merging of bus guides by default, in addition to wires. You can also use the *Split Selected Bus Guides* and *Merge Selected Bus Guides* buttons on the Edit Bus Guide form to split and merge bus guides.

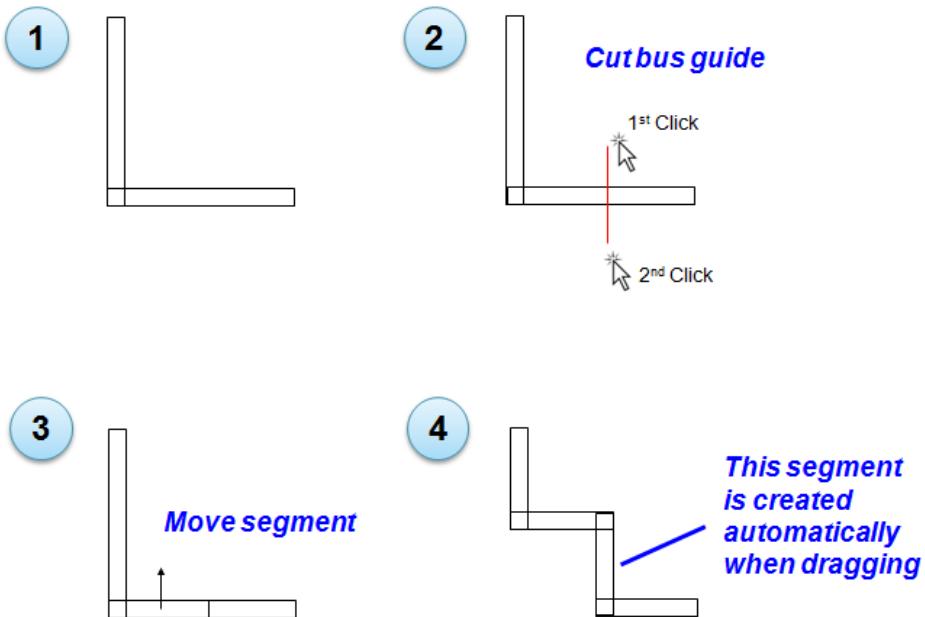
## Creating a jog



## Creating a detour



## Flipping a corner



## Customizing the Bus Guide Display

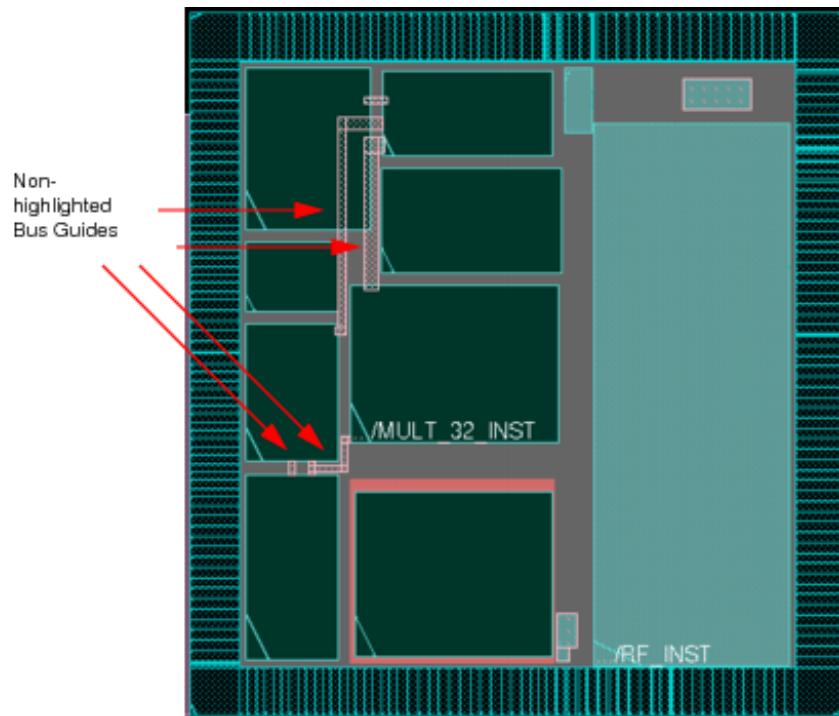
You can specify multiple colors for bus guide objects in the design, using the *Bus Guide Color Selection* form. (*Color Preferences -- Objects -- Bus Guide -- Bus Guide Color Selection*)

## Highlighting and Dehighlighting the Bus Guide

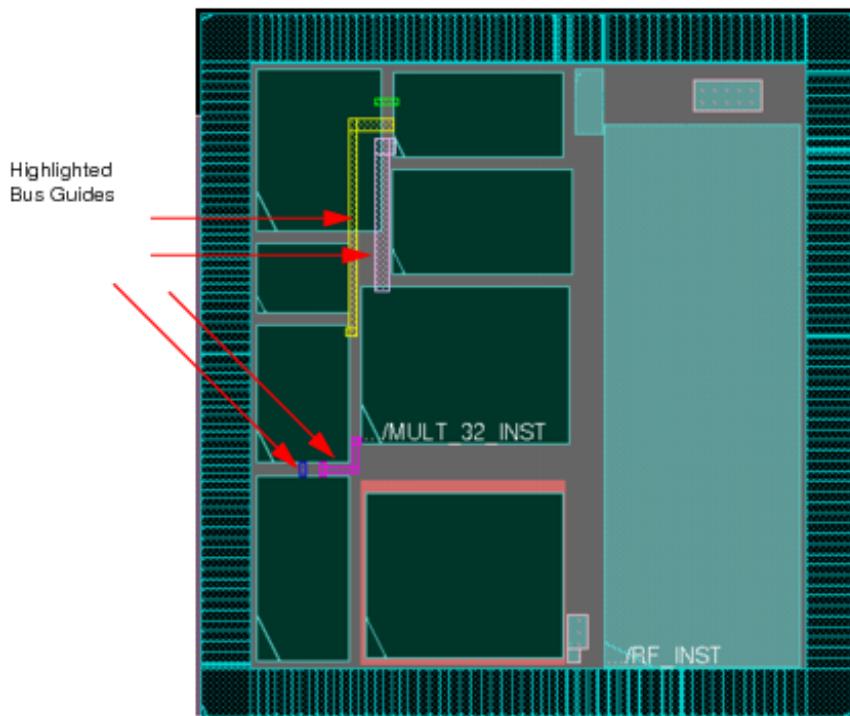
After specifying colors for bus guides, you can highlight the bus guides in the design using the *Edit - Bus Guide -- Color* menu command.

Alternatively, you can run the `setBusGuideMultiColors` command to color the bus guides and `resetBusGuideMultiColors` command to clear the bus guide colors.

The following example displays the bus guides before you run the `setBusGuideMultiColors` command:



The following example displays the bus guides after you ran the `setBusGuideMultiColors` command:



## Saving and Restoring Bus Guide Information

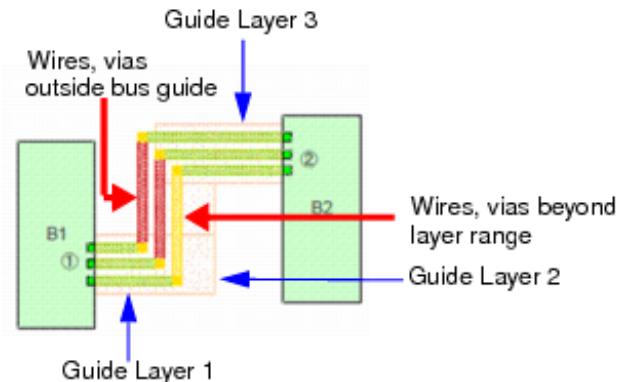
The bus guide data is stored in the floorplan spr file (.fp.spr file). You can save and restore this information using the `saveFPlan` and `loadFPlan` commands.

However, you cannot load the .fp.spr file having bus guide information from the 8.1 version, into an older version of Innovus.

## Verifying Bus Guide

You can check for bus guide violations in all or specified net groups. Bus guide violations include:

- Wires, vias, and pins outside the bus guide
- Wires, vias, and pins partially outside the bus guide
- Wires and vias beyond the layer range



You can check for such violations using the Verify Bus Guide form in the [Verify Menu](#) or the `verifyBusGuide` command.

## Limitations of Bus Planning

- Feedthrough insertion does not honor bus planning. Once you insert feedthroughs in the design, the existing bus guides will no longer be valid.
- The software currently does not provide checks to detect the following:
  - Overlapping bus guide segments on different layers
  - Complete bus guide coverage from source to sink

- Complete coverage of placed pins
- Enough room for routing.

# Power Planning and Routing

- [Overview](#)
- [Before You Begin](#)
- [Results](#)
- [Loading, Saving, and Updating Special Route](#)
- [Global Net Connections](#)
  - [globalNetConnect Command and Connections for Signal Pins and Power/Ground Pins](#)
- [Creating a Ring with User Defined Coordinates](#)
- [Fixing LEF Minimum Spacing Violations](#)
- [Adding Stripes to Power Domains](#)
- [Adding Stripe in Multi-CPU mode](#)

## Overview

Power planning and routing is composed of the following components:

- Adding a core ring
- Adding block rings
- Adding stripes to the core area
- Adding stripes over blocks within the design
- Creating a pad ring
- Connecting pad pins
- Routing standard cell pins
- Connecting block pins
- Connecting unconnected stripe
- Routing to power bumps

Use the Innovus power planning features to create power structures such as rings and stripes for the design. To create power structures, complete the following steps:

1. Load a LEF file that contains technology information before you add power rings and power stripes. If the LEF file is not loaded, you will not be able to select metal layers on the power planning forms.
2. Specify the floorplan.
3. Establish logical power connectivity. You can issue the [globalNetConnect](#) command.
4. Add power rings around the core of the design, and block rings around blocks, row clusters,

and power domains.

5. Add power stripes within the overall design, within a specific area, or over specified blocks or power domains.
6. Save special route data.

After your design is placed, you can use the Innovus power routing features to make the final power connections. The SRoute software creates pad rings and routes power and ground nets to the following power structures:

- Block pins
- Pad pins
- Standard cell pins
- Unconnected stripes

Use the SRoute form to specify routing to any or all of these structures. In addition, you can specify whether or not the software should make the connections by changing layers or allowing jogs.

## Before You Begin

Before you can begin power planning, the following conditions must be met:

- The design must be loaded into the current Innovus session.

The following input file is required:

- The design file

Before you can begin power routing, the following conditions must be met:

- The design must have power rings and stripes.

The following input file is required:

- A LEF file that contains technology and macro information.

## Results

After using the power planning software, your design has preliminary power structures that provide the foundation for hooking up each cell to a power source.

After using the power routing software, your design has power connections between pins of

specified nets on the blocks and pads to nearby rings or stripes. Your design is ready for combined power and rail analysis to determine whether power structures and connections provide sufficient power to the design.

## Loading, Saving, and Updating Special Route

To load special route data into a design, use the Load Special Route form. When loading the floorplan, the `.fp` and `.fp.spr` files are included. The filename extension entered in the Load FPlan form is `.fp`, not `.fp.spr`.

When creating special routes, use the Save Special Route form or the Save Design form to save the special route data plus vias.

Floorplan files are saved with the following extensions:

- `.fp` -- Contains the general floorplan information.
- `.fp.spr` -- Contains the special route data.

You can also use the Save DEF form to save special route information in the DEF file.

## Global Net Connections

Global net connections connect terminals and nets to the appropriate power and ground nets so that power planning, power routing, detail routing, and power analysis functions operate correctly for the entire design. Some of these terminals and nets are contained in the Verilog® netlist, and others are contained in the LEF file.

From the Verilog netlist, you can connect the following type of nets to power and ground nets:

- Power and ground nets  
Connect between the power and ground nets to the appropriate power and ground nets. These power and ground nets are `wire` keywords in the Verilog netlist.
- Tie-hi and tie-lo nets  
Connect between the tie-hi and tie-lo nets to the appropriate power and ground nets. These are keywords in the Verilog netlist, such as `1'b0`, `1'b1`, `supply0`, and `supply1`.
- Local nets  
Connect between the local nets to the global nets. These local nets are `wire` keywords in the

Verilog netlist.

From the LEF file, you can connect the following type of terminals and nets to power and ground nets:

- Power and ground terminals

Connect between the power pins to the appropriate power and ground nets. `vdd!` and `gnd!` are examples of these power and ground pin and net names in the LEF file.

- Filler cell nets

Connect between the power pins to the appropriate power and ground nets. You can specify these connections before or after adding filler cells.

To assign pins or nets to a global net, use the Global Net Connections form (*Floorplan - Global Net Connections*).

**i** The order of global net connections are important, especially when the *Apply All* or the *Override prior connection* options are selected in the Global Net Connections form. *Apply All* connects all pins or nets in the design to the specified global net. *Override prior connection* first disconnects pins and local nets that are already connected to a global net, then reconnects them to the specified global net specified in the form.

You can also use the `globalNetConnect` text command to assign global net connections. For more information, see "Power Planning Commands" in the *Innovus Text Command Reference*.

## globalNetConnect Command and Connections for Signal Pins and Power/Ground Pins

The `globalNetConnect -type net` command automatically connects *only* signal pins to the global net--*not* power or ground pins.

To reconnect power or ground pins to the global net after using the `init_design` command, use the command `globalNetConnect -type pgpin -pin pin_name -all -override`. All global net connection rules and rules are defined/derived in the CPF file. The global file in the 12.x version of the software maps to the configuration file in the 10.x and earlier software versions.

## Creating a Ring with User Defined Coordinates

To create a block ring or a core ring in a specific location, complete the following steps:

- Choose *Power - Power Planning - Add Rings*. This opens the *Basic* page of the Add Rings form. Specify the net names for power rings to be created.
- Select *User defined coordinates*. Select either *Core ring* or *Block ring*.
- If you select *Core ring*, the shape of the wires is ring. If you select *Block ring*, the shape of the wires is block ring.
- Specify a set of coordinates.
- You must specify at least four pairs of coordinates. Specify the x coordinate followed by the y coordinate for each corner of the ring. Separate each coordinate with a space.  
For example, to define a 100 x 100 square ring at the bottom left corner of the design, specify the coordinates as follows: 0 0 0 100 100 100 100 0
- You can create a rectilinear ring by specifying an even number of coordinate pairs. For example, specify the following set of coordinates to create an L-shaped ring: 0 0 0 100 50 100 50 50 100 50 100 0
- You must specify the coordinates in a linear sequence. For example, if you specify the coordinates in the following sequence, the ring is not created because the sequence of the coordinates defines the bottom left, top right, top left, and top right corners: 0 0 100 100 0 100 100 0
- You must also specify coordinates that create perpendicular wires. For example, if you specify the following coordinates, the ring is not created because the coordinates define an edge that slants: 0 0 0 100 100 100 50 0
- Click *Apply* or *OK*. The ring is created in the exact location specified by the coordinates.

### Fixing LEF MINIMUMCUT Violations

If your design contains violations to the LEF MINIMUMCUT rule, use the [fixVia](#) command with option `-minCut` as a post-processing step. This command replaces power vias flagged by a violation marker because of a violation of the LEF MINIMUMCUT rule. The via is replaced with a via that contains the minimum number of cuts based on the LEF rule, and the violation marker is removed. If the via cannot be replaced, the violation marker is not removed. User could call command fixVia several times to fix the violation.

See the [Power Route Commands](#) chapter in the *Innovus Text Command Reference* for more information.

## Fixing LEF Minimum Spacing Violations

If your design contains violations of the LEF minimum spacing rule, use the `fillNotch` command as a post-processing step. The command fills gaps, which removes same net violations between generated vias and wires or pins where these violations are flagged by the Verify Geometry software. Using this command, you do not have to sacrifice via size by trimming vias in order to fix same net spacing violations.

See the [Verify Commands](#) chapter in the *Innovus Text Command Reference* for more information.

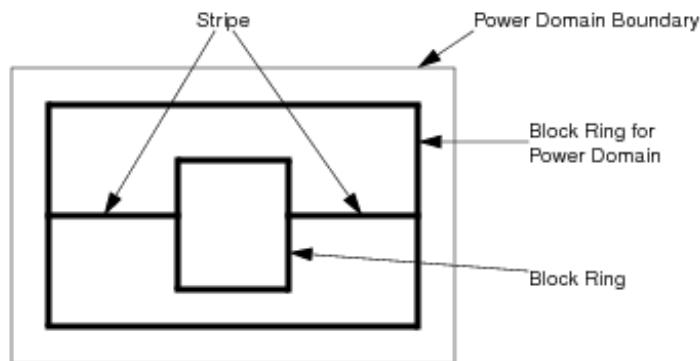
## Adding Stripes to Power Domains

When you use the *Each selected block/domain/fence* option on the [\*Basic\*](#) page of the Add Stripes form, stripes are placed according to the location of the power domain ring. Sometimes the location of the power domain ring is not specified at the time the power domain was created. In this case, you must indicate the location of power domain rings to the power planning software by issuing the following command:

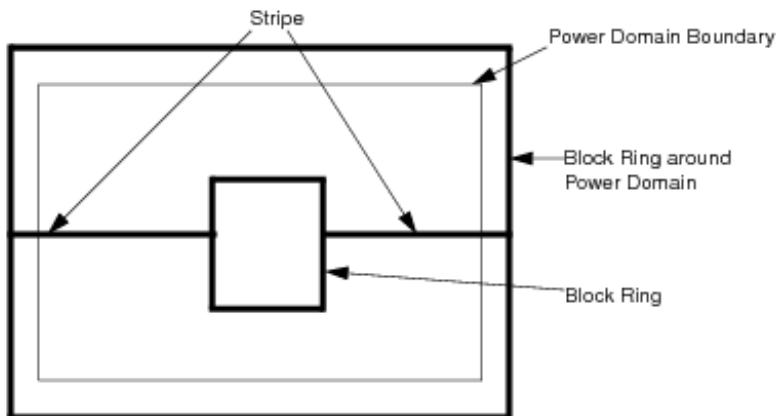
```
modifyPowerDomainAttr -rsExts top bottom left right
```

The `top`, `bottom`, `left`, and `right` values specify a distance from the edge of the power domain boundary. Rings between the power domain boundary and the specified distance are considered power domain rings.

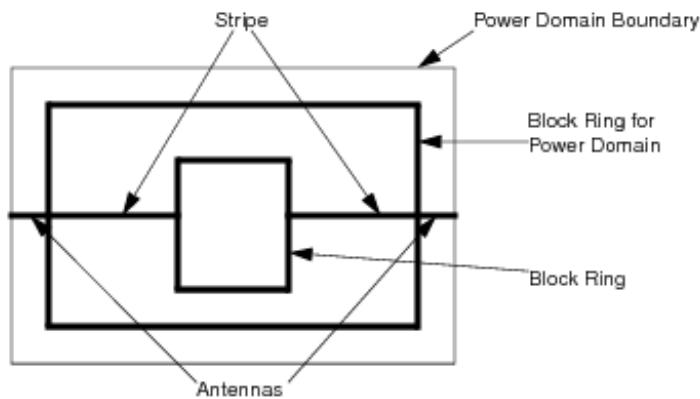
- Specify negative values if the power domain ring is inside the power domain boundary. The power planning software considers any ring from the power domain boundary to the specified distance *inside* the boundary to be a power domain ring. When you add stripes, the software trims the stripes at the ring. The stripe also correctly recognizes block rings within the power domain and breaks at those rings if you specify the *Omit stripes inside block rings* option on the [\*Advanced\*](#) page of the Add Stripes form. The following illustration shows how the stripe is created when you specify negative values.



- Specify positive values if the power domain ring is outside the power domain boundary. The power planning software considers any ring from the power domain boundary to the specified distance *outside* the boundary to be a power domain ring, and extends the stripes to that ring. The stripe also correctly recognizes block rings within the power domain and breaks at those rings if you specify the *Omit stripes inside block rings* option on the *Advanced* page of the Add Stripes form. The following illustration shows how the stripe is created when you specify positive values.



If the location of the power domain ring is not specified, the stripe begins and ends at the power domain boundary. If the power domain ring is inside the power domain, the stripe recognizes it as a block ring. This can cause the stripe to break in the wrong locations if you specify the *Omit stripes inside block rings* option on the *Advanced* page of the Add Stripes form, and can create antennas, as shown in the following illustration.



## Adding Stripe in Multi-CPU mode

In 12.x release of the software, [addStripe](#) has been enhanced to support multi-threading to improve the addStripe runtime on a large design. Multi-threading accelerates addStripe by splitting a job into several tasks that run concurrently on a single machine that has multiple processors.

Multi-threading addStripe has the following limitations:

- It is not available for the following options: `-over_pins`, `-master`, `-over_bumps` or `-between_bumps`.
- When viaGen is invoked internally in addStripe, multi-threading may not be honored if viaGen detects the design is not suitable for it to work properly using multi-cpu.

Use the following command to specify the number of CPUs on the local machine:

`setMultiCpuUsage –localCpu cpuNum`

Also, you can set or change the multi-CPU setting from *Multi-CPU Processing* GUI form of Innovus.

**Note:** This will affect any application that can run in multiple-CPU processing mode

# Design Implementation Capabilities

---

- Low Power Design
- Placing the Design
- Clock Tree Synthesis
- Legacy FE-CTS Capabilities
- Working with Clock Mesh Structures
- Optimizing Timing
- Using the NanoRoute Router
- Optimizing Metal Density
- Flip Chip Methodologies

# Low Power Design

- Overview
- Power Domain Shutdown and Scaling
- Support for the Common Power Format (CPF)
  - CPF Version Support
  - Innovus Commands Supporting CPF
  - Loading and Committing a CPF File
  - Saving a CPF Database
  - Load the design (init\_design)
  - CPF Documentation
- Support for IEEE1801
  - Enable IEEE1801 Flow in 13.2 Release of the Software
  - Low Power Cell Definition
  - Timing Information
  - Load the design (init\_design) for IEEE1801
  - Innovus Commands Supporting IEEE1801
  - Innovus IEEE1801 Low Power Flow
  - Innovus IEEE1801 Command Set Support
  - IEEE1801 Documentation
- Flow Special Handling for Low Power
  - Low Power Cells and Usage
  - Specifying Power Intent
  - The Innovus Low Power Flow
  - Low Power Planning and Routing
  - Low Power Optimization
  - Secondary Power Pin Routing
  - Low Power Design Verification
  - Low Power Debugging Commands
- Multiple Supply Voltage Top-Down Hierarchical Flow
  - Overview
  - Always-On Feedthrough Handling
  - Chip Partitioning
  - Block-level CPF Generation
  - Top-Level CPF Generation

- Block-Level Implementation
- Top-Level Implementation
- Chip Assembly
- Example of Block-Level CPF Generated by Innovus
- Example of Top-Level CPF Generated by Innovus
- Multiple Supply Voltage Bottom-Up Hierarchical Flow
  - Block-Level Implementation
  - Top-Level Implementation
  - Chip Assembly
- Leakage Power Optimization Techniques
  - Multi-Vth Optimization
  - Substrate Biasing
- Power Shutdown Techniques
  - Data Preparation
  - Buffer Styles
  - Adding Column Switches
  - Attaching the Acknowledge Receiver Pin
  - Enable Chaining
  - Controlling the Maximum Enable Chain Depth
  - Synthesizing Acknowledge Trees
  - Adding Power Switch Rings
  - Ring Conventions
  - Using Pitch Control and Offsets
- Power Switch Optimization
  - Power Switch Reduction
  - Power Switch ECO
- Power Switch Prototyping
  - Power Domain Parameters and Specification
  - Options Summary - Switch and Power Domain
  - Options Summary - Prototyping Features
  - Chain Style Impacts on Ramp Up Time and Rush Current
  - Prototyping Results
  - Optimal Switch Results
  - Switch Number Enumeration Results
  - Ramp Up Switch Enumeration Results

- Number of Switches Given Current Maximum Ramp Up
- Switch Delay Given Current Maximum Ramp Up Current
- Ramp Up Time

## Overview

This chapter describes how the multiple supply voltage (MSV) feature can help you save power in your design.

There are two types of MSV designs:

- Multiple Supply Single Voltage (MSSV)  
Core logic runs at a single voltage, but some portions of the logic are isolated on their own power supply.
- Multiple Supply Multiple Voltage (MSMV)  
Supplies of different voltages are used for core logic.

A power domain (also known as voltage island) is a floorplan object in the Innovus™ Implementation System (Innovus) software. A non default and non virtual power domain has a fence constraintant physically; each power domain has a specific library (.lib, .lef) associated with it. Standard cell Instances that belong to a power domain can be placed only within that power domain. The exception to this rule is Macros, IP blocks and IOs. By constraining the design this way, a complete place and route flow can be used on an MSV design. You can automatically place level shifters, perform timing optimization, run clock tree synthesis (CTS) across domain boundaries, and obtain DRC-clean power routing.

## Power Domain Shutdown and Scaling

You can reduce power consumption either by shutting down a power domain or operating it at a reduced voltage (voltage scaling).

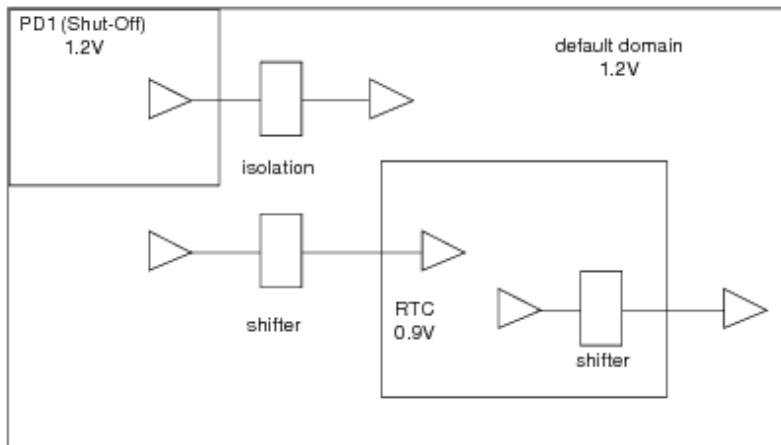
Power domain shutdown is a technique in which an entire power domain is shut down during a specific mode of operation. This results in both leakage power and dynamic power savings because the transistors are isolated from the supply and ground lines. You must use isolation cells when shutting down domains in order to drive the interface signals to predetermined known states. In many cases, a design in the shutdown mode operates at a single voltage throughout the design (an MSSV design); however, the portion of the design that is shut off must be in a different power domain. This is necessary because this portion must be isolated from the rest of the system so that it can be shut off independently from the rest of the core logic. For more information on power shutdown, see [Power Shutdown Techniques](#).

In power domain scaling (also known as voltage scaling), one or more domains operate at a lower

voltage than that of the other core logic. Power domain scaling provides dynamic power savings, and can provide leakage power savings, depending on the threshold characteristics of the library for the scaled domain.

**Note:** These techniques can be used separately or together in the same design.

The following figure shows three power domains: RTC, PD1, and the default power domain, which contains PD1 and RTC.



- PD1 and default domains can share libraries since PD1 and default domains operate at the same voltage.
- Power switches enable PD1 to shut down.
- Power domain RTC operates at a different voltage than PD1 and the default domain.
- RTC can remain always on.
- You must insert voltage level shifters between the default domain and RTC, and between PD1 and RTC.
- Isolation cells (clamps) drive outputs of a power domain to known states when that power domain is shut down.

## Support for the Common Power Format (CPF)

Cadence provides a Common Power Format (CPF) that enables you to freely exchange data between Cadence tools supporting the low-power design flows, and most importantly, capture low power design intent early in the design process rather than late in the back-end cycle. A CPF file captures all design and technology-related power constraints, which can be used throughout the design flow. Users need to create CPF file for their power constraint and read it in to Innovus for the low power design.

CPF commands perform functions such as the following:

- Creating power domains and specifying their power/ground connections
- Specifying timing libraries (Optional; users can define timing libraries in Innovus viewDefinition.tcl)
- Creating analysis view and defining library set for each power domain (Optional: users can define them in Innovus viewDefinition.tcl)
- Defining operating conditions (Optional; users can define them in Innovus viewDefinition.tcl)
- Defining low power cells
- Creating low power rules: isolation rules, level shifter rules, SRPG rules, power switch rules)

**Note:** If there is a minor CPF change during the flow, you can either perform CPF ECO (requiring CLP license) or a view-related update during the flow, without running the flow from the beginning.

## CPF Version Support

The Innovus software supports the following versions of CPF:

- CPF 1.0
- CPF 1.0e
- CPF 1.1 (default)
- CPF 2.0

## Innovus Commands Supporting CPF

- [loadCPF](#)
- [commitCPF](#)
- [saveCPF](#)

## Loading and Committing a CPF File

A GUI enables you to load and commit a CPF file:

- [Power - Multiple Supply Voltage - Load/Commit CPF](#)

This GUI corresponds to the following text commands:

- [loadCPF](#)  
Reads a CPF file into Innovus for error checking
- [commitCPF](#)  
Executes (commits) the CPF commands within the Innovus environment

## Saving a CPF Database

The [saveDesign](#) command prevents you from saving an Innovus database from obsolete Innovus MSV commands.

By default, if the design was created with MSV commands only and no CPF file, then the design cannot be saved to the Innovus database with [saveDesign](#).

Complete the following steps to save a CPF database:

1. Save the CPF file from an Innovus database [saveCPF](#)
2. Exit the current session
3. Start a new session
4. Load the CPF file into Innovus [loadCPF](#)
5. Execute the commands in the CPF file [commitCPF](#)
6. Save the design [saveDesign](#)

## Load the design (`init_design`)

The design data is read through the `init_design` procedure. In case of CPF, `init_design` calls a special `loadCPF` to generate the view definition file. The `init_design` then loads the libraries and sets up MMMC based on the `viewDefinition.tcl`. Since there is no minimum/maximum analysis, the CPF creates the analysis view. Also, since there is no analysis view in CPF, `commitCPF` does not generate the script `viewdefinition.tcl` and `init_design` gives an error.

The command `init_design` works as follows:

- If there is a *viewDefinition.tcl* for Low Power design, *init\_design* does not call special *loadCPF* to re-generate *viewDefinition.tcl*.
- If there is no *viewDefinition.tcl* for Low Power design, *init\_design* calls a special *loadCPF* to generate *viewDefinition.tcl* based on the CPF.
- If both *viewDefinition.tcl* and CPF are provided and *viewDefinition.tcl* has higher priority, *init\_design* does not call the special *loadCPF*.
- If there is no *viewDefinition.tcl* and CPF does not define analysis view, *init\_design* gives an error.

For design portability, you can set the Library Primary Path in tcl. For example,

```
set libDir "libs/";
viewDefinition.tcl will refer to the tcl variable (libDir)
```

For all the library specifications like: `create_library_set -library $libDir/...`

If there is any network change, you can re-define the Library Primary Path and do not need to change anything else. To do this, the special *loadCPF* generates the *viewDefinition.tcl*.

**Note:** Since 11.1 release of the software supports only MMMC, Low Power supports only the MMMC Flow.

- If design does not have *viewdefinition.tcl*, CPF should contain *create\_analysis\_view*.
- If design has *viewDefinition.tcl*, CPF does not require timing library information. The CPF is library-less, and only specifies the Power intent. All the timing information is specified in *viewDefinition.tcl*. **This setting is recommended from 14.1 and above releases of the software.**
- If CPF does not have *create\_analysis\_view* and design does not have *viewdefinition.tcl*, *commitCPF* gives an error.

## CPF Documentation

For more information about CPF, see the following documents:

- [Innovus Text Command Reference](#)  
Documents the following Innovus commands supporting the CPF flow. Descriptions on obsolete native Innovus commands are provided.
- [Innovus Menu Reference](#)

Documents the `loadCPF/saveCPF` GUI.

- [Innovus User Guide](#)

- Provides a list of supported CPF 1.0 commands in the [Supported CPF 1.0 Commands](#) chapter.
- Provides a list of supported CPF 1.0 commands in the [Supported CPF 1.0e Commands](#) chapter
- Provides a list of supported CPF 1.0 commands in the [Supported CPF 1.1 Commands](#) chapter
- Provides a sample CPF 1.0 script in the [CPF 1.0 Script Example](#) chapter
- Provides a sample CPF 1.0e script in the [CPF 1.0e Script Example](#) chapter.
- Provides a sample CPF 1.1 script in the [CPF 1.1 Script Example](#) chapter.

## Support for IEEE1801

The IEEE1801 standard is supported from 13.2 and above releases of the software to specify the design's power intent.

### Enable IEEE1801 Flow in 13.2 Release of the Software

- Set “`setLimitedAccessFeature msvSupportIEEE1801 1`” at the beginning of the flow.

## Low Power Cell Definition

- All the LP cells and related power pin information need to be defined in the Liberty file with the Liberty LP attributes.

## Timing Information

- Define Timing Information in *Innovus MMMC viewDefinition.tcl* and Use “`set init_mmmc_file viewDefinition.tcl`” to specify MMMC timing information. The power domain library binding must be specified through the `update_delay_corner` command for each power domain.

## Load the design (`init_design`) for IEEE1801

The **IEEE1801 specifies** only the Power intent like library-less CPF. All the timing information is specified in `viewDefinition.tcl`.

## Innovus Commands Supporting IEEE1801

- [read\\_power\\_intent](#)
- [commit\\_power\\_intent](#)
- [write\\_power\\_intent](#)

## Innovus IEEE1801 Low Power Flow

Innovus IEEE1801 low power flow is similar to the Innovus CPF low power flow. All the Innovus implementation steps such as floorplan, placement, optimization, clock tree synthesis and routing have been enhanced to handle IEEE1801 so that the IEEE1801 low power flow can run smoothly. The IEEE1801 flow looks like:

```
setLimitedAccessFeature msvSupportIEEE1801 1
init_design #use "set init_mmmc_file viewDefinition.tcl" to specify mmmc setting
read_power_intent -1801 <IEEE1801File>
commit_power_intent
# The rest of IEEE1801 low power flow (same as CPF low power flow).
```

## Innovus IEEE1801 Command Set Support

Innovus mainly supports a selected set of IEEE1801 2.0 commands and options. For the compatibility purpose of users, Innovus also supports some IEEE1801 1.0 commands and options.

**Currently, the following IEEE commands/options are supported.**

| IEEE1801 Commands/Options                                                                             | IEEE Version |
|-------------------------------------------------------------------------------------------------------|--------------|
| <code>add_port_state port_name<br/>{-state {name &lt;nom   min max   min nom max   off&gt;} }*</code> | 1.0          |

|                                                                                                                                                                                                                                |     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| add_power_state object_name<br>{-state state_name {<br>[-supply_expr {boolean_function}]<br>[-logic_expr {boolean_function}]<br>[-update]}}                                                                                    | 2.0 |
| apply_power_model power_model_name<br>[-elements instance_list]<br>[-supply_map {{lower_scope_handle upper_scope_supply_set}}]                                                                                                 | 2.1 |
| add_pst_state state_name<br>-pst table_name<br>-state supply_states                                                                                                                                                            | 1.0 |
| associate_supply_set supply_set_ref<br>-handle supply_set_handle                                                                                                                                                               | 2.0 |
| begin_power_model power_model_name<br>[-for model_list]                                                                                                                                                                        | 2.1 |
| connect_logic_net net_name<br>-ports port_list                                                                                                                                                                                 | 2.0 |
| connect_supply_net net_name<br>[-ports list]<br>[-pg_type {pg_type_list element_list}] *<br>[-pins list]<br>[-domain domain_name]                                                                                              | 1.0 |
| connect_supply_set supply_set_ref<br>{-connect {supply_function {pg_type_list}}} *                                                                                                                                             | 2.0 |
| create_logic_net net_name                                                                                                                                                                                                      | 2.0 |
| create_logic_port port_name<br>[-direction <in   out   inout>]                                                                                                                                                                 | 2.0 |
| create_power_domain domain_name<br>[-elements element_list]<br>[-exclude_elements exclude_list]<br>[-include_scope]<br>[-supply {supply_set_handle [supply_set_ref]} *]<br>[-update]<br>[-available_supplies {supply_set_ref}] | 1.0 |
|                                                                                                                                                                                                                                | 2.1 |

|                                                                                                                                                                              |     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| create_power_switch switch_name<br>{-output_supply_port {port_name}<br>{-input_supply_port {port_name}<br>{-control_port {port_name [net_name]} } *<br>[-domain domain_name] | 1.0 |
| create_pst table_name<br>-supplies list                                                                                                                                      | 1.0 |
| create_supply_net net_name<br>-domain                                                                                                                                        | 1.0 |
| create_supply_port port_name<br>[-domain domain_name]<br>[-direction <in  out>]                                                                                              | 1.0 |
| create_supply_set set_name<br>[-function {func_name [net_name]} ] *<br>[-update]                                                                                             | 2.0 |
| end_power_model                                                                                                                                                              | 2.1 |
| load_upf upf_file_name<br>[-scope instance_name]                                                                                                                             |     |
| map_isolation_cell isolation_name<br>-domain domain_name<br>-lib_cells lib_cell_list                                                                                         | 1.0 |
| map_level_shifter_cell level_shifter_name<br>-domain domain_name<br>-lib_cells lib_cell_list                                                                                 | 1.0 |
| map_power_switch switch_name<br>-domain domain_name<br>-lib_cells lib_cell_list                                                                                              | 1.0 |
| map_retention_cell retention_name_list<br>-domain domain_name<br>[-lib_cells lib_cell_list]<br>[-lib_cell_type lib_cel_type]                                                 | 1.0 |
| name_format<br>[-isolation_prefix string]<br>[-level_shifter_prefix string]                                                                                                  | 1.0 |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| set_scope instance                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 2.0                                                                                                                        |
| set_domain_supply_net domain_name<br>-primary_power_net supply_net_name<br>-primary_ground_net supply_net_name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 1.0                                                                                                                        |
| set_isolation isolation_name<br>-domain ref_domain_name<br>[-elements element_list]<br>[-source source_supply_ref]<br>[-sink sink_supply_ref]<br>[-applies_to <inputs   outputs   both>]<br>[-isolation_power_net net_name]<br>[-isolation_ground_net net_name]<br>[-no_isolation]<br>[-isolation_supply_set supply_set_list]<br>[-isolation_signal signal_list]<br>[-isolation_sense {<high   low>}*]<br>[-name_prefix string]<br>[-clamp_value {<0   1   any   Z   latch   value>}*]<br>[-location <automatic   self   other   fanout   fanin  <br>-diff_supply_only <TRUE   FALSE>]<br>[-update] | 2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0 |
| set_isolation_control isolation_name<br>-domain domain_name<br>-isolation_signal signal_name<br>[-isolation_sense <high low>]<br>[-location <self parent sibling fanout automatic>]                                                                                                                                                                                                                                                                                                                                                                                                                 | 1.0                                                                                                                        |
| set_level_shifter level_shifter_name<br>-domain domain_name<br>[-elements element_list]<br>[-no_shift]<br>[-source source_domain]<br>[-sink sink_domain]<br>[-applies_to <inputs outputs both>]<br>[-rule <low_to_high high_to_low both>]<br>[-location <self   parent   sibling   fanout   automatic><br>[-name_prefix string]<br>[-input_supply_set supply_set_name]<br>[-output_supply_set supply_set_name]<br>[-update]                                                                                                                                                                         | 2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0                                    |

|                                                                                                                                                                                                                                                                                                                                                                                                               |                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| set_pin_related_supply library_cell<br>-pins list<br>-related_power_pin supply_pin<br>-related_ground_pin supply_pin                                                                                                                                                                                                                                                                                          | 1.0                                                  |
| set_port_attributes<br>[-ports {port_list}] [-exclude_ports {port_list}]<br>[-receiver_supply supply_set_ref]<br>[-driver_supply supply_set_ref]                                                                                                                                                                                                                                                              | 2.0                                                  |
| set_retention retention_name<br>-domain domain_name<br>[-elements element_list]<br>[-exclude_elements [-retention_power_net net_name] exclude_list]<br>[-retention_power_net net_name]<br>[-retention_ground_net net_name]<br>[-retention_supply_set ret_supply_set]<br>[-no_retention]<br>[-save_signal logic_net <high   low   posedge  <br>[-restore_signal logic_net <high   low   posedge  <br>[-update] | 1.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0<br>2.0 |
| set_retention_control retention_name<br>-domain domain_name<br>[-save_signal net_name]<br>[-restore_signal net_name]                                                                                                                                                                                                                                                                                          | 1.0                                                  |
| upf_version [string]                                                                                                                                                                                                                                                                                                                                                                                          | 1.0                                                  |
| use_interface_cell interface_implementation_name<br>[-strategy list_of_isolation_level_shifter_strategies]<br>[-lib_cells lib_cell_list]                                                                                                                                                                                                                                                                      | 2.0                                                  |
| find_objects scope<br>-pattern search_pattern<br>[-object_type <inst   port   net   process>]<br>[-direction <in   out   inout>]<br>[-transitive <TRUE   FALSE>]<br>[-non_leaf -leaf_only]                                                                                                                                                                                                                    | 2.0                                                  |
| set_related_supply_net power_net<br>[object_list]<br>[-ground]<br>[-power]                                                                                                                                                                                                                                                                                                                                    | SNPS Special                                         |

## IEEE1801 Documentation

For more information about IEEE1801, see the following documents:

- [Innovus Text Command Reference](#)

Documents the Innovus commands supporting the IEEE1801 flow. Descriptions on native Innovus commands are provided.

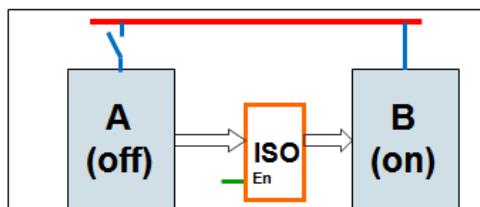
## Flow Special Handling for Low Power

### Low Power Cells and Usage

In the low power design, several different types of Low Power cells are required and used. These cells are defined in CPF or Liberty. A brief description of each type of cell is as below:

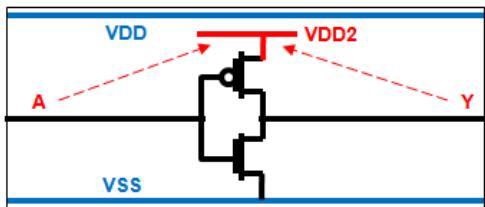
#### Isolation Cell

Isolation cell is inserted between two domains to prevent signals from floating when the driving domain is powered off. The Isolation logic types can be *high*, *low* or *keep last value*.



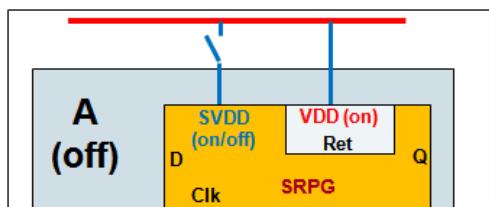
#### Always-On Cell

The Always-On cell is powered by its second power pin. If it is in a switchable power domain and powered by always-on power, it can stay on when switchable power domain is off.



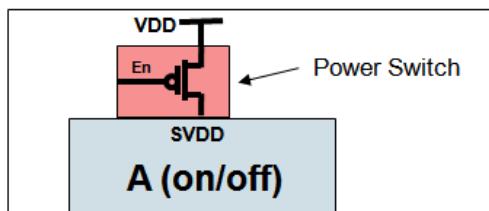
## State Retention DFF

A special flop in a switchable domain that can retain its state value when the switchable domain's power supply is turned off. It has secondary power pin to power the retention logic.



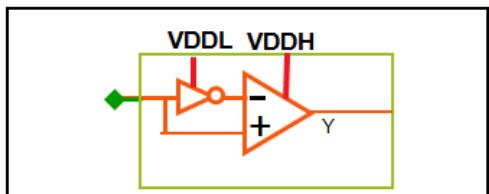
## Power Gate or Power Switch Cell

This is a cell used to turn on/off the power supply of a domain.



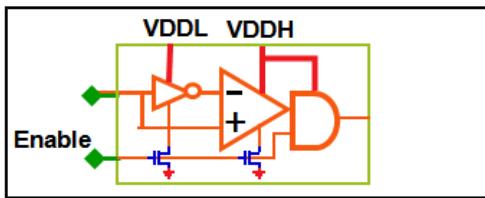
## Level Shifter Cell

The Level Shifter cell shifts mainly from lower voltage signal to higher voltage signal or from higher voltage signal to lower voltage signal. It may have a significant delay impact.



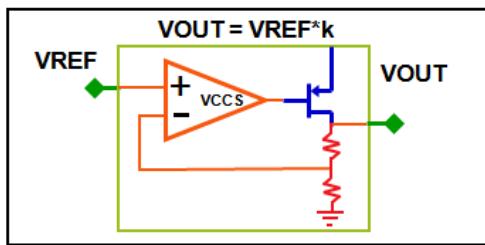
## Level Shifter/Isolation Combo Cell

This is a combination of level shifter and isolation cell and is commonly used in designs having both MSV and PSO.



## Voltage Regulator Cell (Optional on-chip)

This provides different voltage supply on a chip. Factors like Area, IR-drop and noise need special handling.



## Specifying Power Intent

As the low power design becomes more and more complex, the **Power Intent File** is required to capture the low power information such as power domain and low power cell usages.

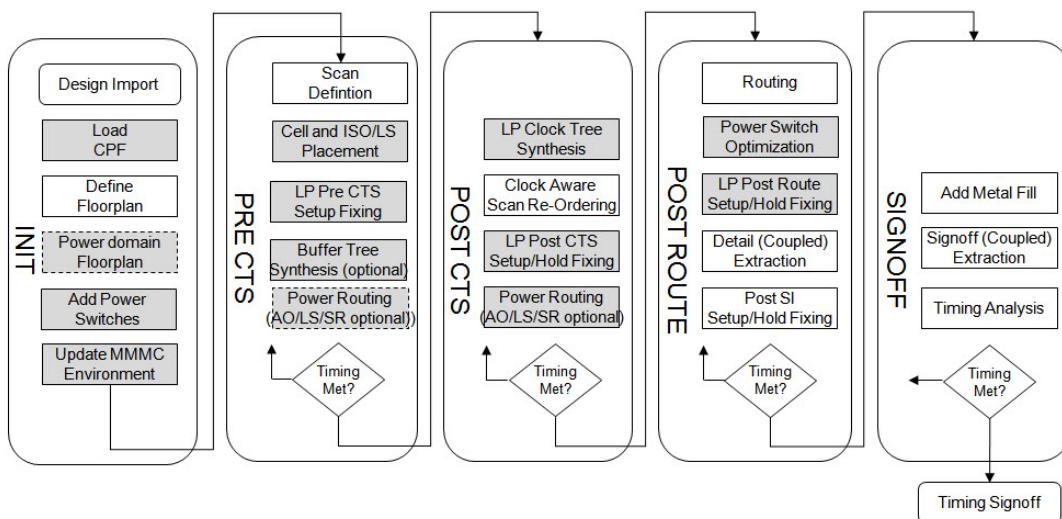
## Power Intent File

The Power Intent File mainly specifies the following in TCL-based CPF or IEEE1801:

- Definition of Low Power cells (Can also be defined in Liberty)
- Definition of power nets and nominal condition (PD working voltage)
- Creation and updation of power domains
- Creation and updation of power modes/power state table
- Creation and updation of rules such as iso/ls rule, srpg and power switch rules

## The Innovus Low Power Flow

The Low Power flow mainly includes the following steps:



**Note:** The steps in grey boxes in the above diagram need special handling for Low Power.

## Setting up the Low Power Flow

- Write power intent file in either CPF or IEEE1801  
Separate MMMC from Power Intent file
- Write Innovus MMMC (**viewDefinition.tcl**) separately  
Make sure each power domain has library binding

## Example

### Innovus viewDefinition.tcl

```

create_library_set -name wc_0v81\
-timing\
[list ./timing/tcbn45gsbwpc.lib\
./timing/tcbn45pbwp_c070208wc0d720d9_modified.lib\
./timing/tcbn45pbwp_c070208wc0d90d9_modified.lib\
./timing/tcbn45pbwp_c070208wc_modified.lib\
./timing/tpzn651pgv2wc.lib]
create_library_set -name wc_0v81_1\
-timing\
[list ./timing/tcbn45gsbwpc_1.lib\
list ./timing/tcbn45pbwp_c070208wc0d90d9_modified.lib]
create_library_set -name bc_0v81\
-timing\
[list ./timing/tcbn45gsbwpc.lib\

```

```

./timing/tcbn45lpbwp_c070208bc0d881d1_modified.lib\
./timing/tcbn45lpbwp_c070208bc1d11d1_modified.lib\
./timing/tcbn45lpbwp_c070208bc_modified.lib\
./timing/tpzn651pgv2bc.lib]
create_op_cond -name PM_wc_virtual -library_file \
./timing/tcbn45gsbwpsc.lib -P 1 -V 0.81 -T 125
create_op_cond -name PM_bc_virtual -library_file \
./timing/tcbn45gsbwpsc.lib -P 1 -V 0.99 -T 0
create_rc_corner -name rc_cworst \
-cap_table worst.CapTbl
create_delay_corner -name AV_PM_on_dc\
-library_set wc_0v81\
-opcond_library tcbn45gsbwpsc\
-opcond PM_wc_virtual -rc_corner rc_cworst
update_delay_corner -name AV_PM_on_dc -power_domain PDdefault\
-library_set wc_0v81_1\
-opcond_library tcbn45gsbwpsc\
-opcond PM_wc_virtual
update_delay_corner -name AV_PM_on_dc -power_domain PD1\
-library_set wc_0v81\
-opcond_library tcbn45gsbwpsc\
-opcond PM_wc_virtual
update_delay_corner -name AV_PM_on_dc -power_domain PD2\
-library_set wc_0v81\
-opcond_library tcbn45gsbwpsc\
-opcond PM_wc_virtual
update_delay_corner -name AV_PM_on_dc -power_domain PD3\
-library_set wc_0v81_1\
-opcond_library tcbn45gsbwpsc\
-opcond PM_wc_virtual

```

**Note:** Specify each domain binding by the `update_delay_corner -power_domain` command.

## Low Power DB and ENV Creation

`read_power_intent/commit_power_intent` mainly does the following:

- Check the CPF/IEEE1801 syntax
- Create power domains (groups) and site list for each domain
- Read and commit CPF/IEEE1801 rules (if required)
- Synthesize the simple logic for low power enabled signals
- Replace the assign statement with buffer (PD-aware)
- Generate global and tie connection specifications

- Create the implicit ISO/LS rules for optimization/CTS/*verifyPowerDomain*

Once the above is done, Lower power DB (called \*.cpfdb), is generated to keep the low power information such as power domain, PG nets and low power rules. The DB is saved by *saveDesign* and can be restored by *restoreDesign*.

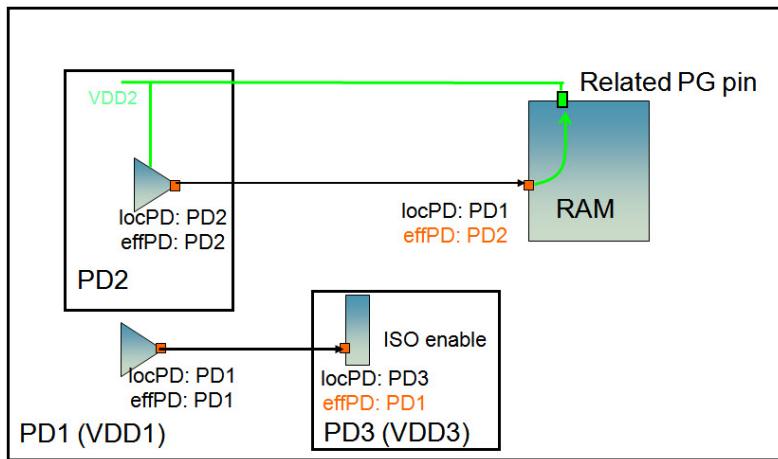
## Pin/Term Power Domain Assignment

Each LEAF signal pin/term and boundary port are assigned to a power domain based on its location, or its related PG pin, or its driving instance, or the definition in CPF/IEEE1801.

The Low Power cell enable pin is assigned to its driving pin domain. Low Power cells and Macros signal pins are assigned according to its “related PG pin”. Some signal pins can also be assigned in CPF or IEEE1801. Regular standard cell leaf instance pin/or pin without “related PG pin” is assigned to its instance power domain (location domain).

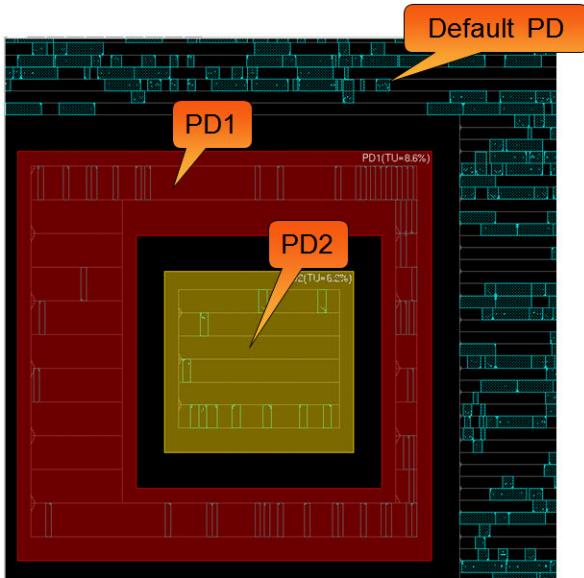
In the following diagram, locPD is the location power domain. effPD is the pin's effective power domain whose primary PG nets drive the pin in reality. locPD can be different from effPD.

The pin's Effective power domain is used to determine if there is domain crossing from the driver to receiver during low power cell insertion, optimization, BTS/CTS and *verifyPowerDomain*.



## Logical vs. Physical Power Domain

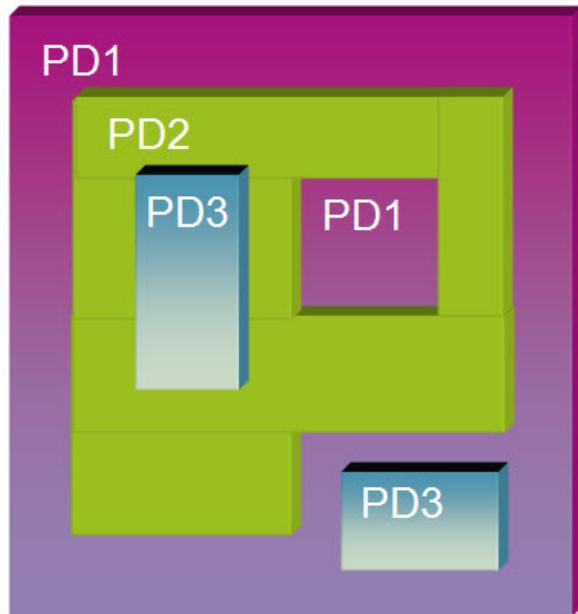
Apart from logical power domain definition in CPF or IEEE1801, power domain in Innovus also means that all of the standard cells (members) in the same domain share the same follow-pin net and standard cell row structure.



In the above example, LEF's "Site" is used to control which cells are allowed in this row. Note that different power domain may have different standard cell height, and the non-default power domain with standard cell members has fence constraint.

## Complex Power Domain Floorplan

Innovus supports various types of the power domain fence shapes such as rectangular, rectilinear, donut and disjoint.



- Ring shape power domain (PD1); Non-default power domain in the middle

- Donut shape power domain (PD2)
- Nested power domain (PD3): It can be either physically nested (PD2), or logically nested, or both
- Disjoint power domain (PD3)

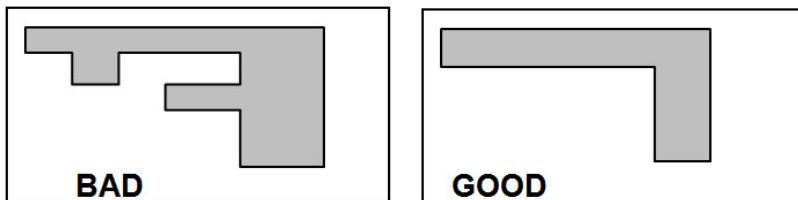
## Power Domain Floorplanning Guideline

- The power domain floorplan (size, shape and location) greatly affects the Cell and ISO/LS placement QoR, Optimization QoR and Routing congestion.
- You should **try** to create the rectangular or simple rectilinear shapes with mini PD boundary edges. This will minimize the crossing-PD route patterns and help ISO/LS placement.
- **Avoid** dividing/blocking power domain fence into pieces, avoid creating the narrow channel between domains or abutted domain.

### Example

#### Minimize the number of PD boundary edges

- Reduce the feedthrough routes
- Help ISO/LS placement



### Example

#### Avoid narrow channel

- Reduce the feedthrough routes and congestion in the channel



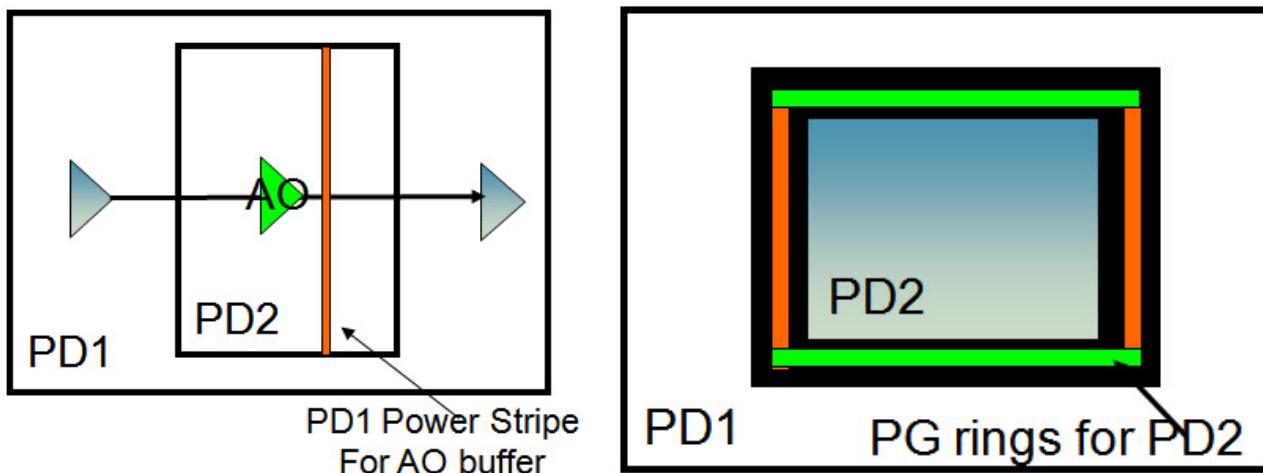
## Low Power Planning and Routing

### Power planning

In the Power Planning process, you need to plan power stripes for AO feedthrough buffering inside the power domain. It is better to create PG rings for power domain to reduce IR drop.

### Power routing

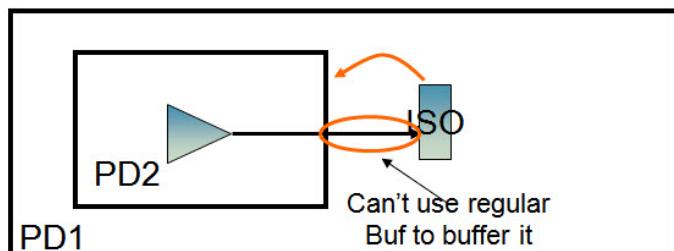
In the Power Routing process, you need to use `addStripe` command to route power switch always-on power over the switches, and the `routePGPinUseSignalRoute` command to route secondary PG pins of the Low Power cells such as AO, SRPG, isolation and level shifter.



### Cell and ISO/LS placement

The cell placement is Power Domain (PD) aware and honors the PD fence constraints.

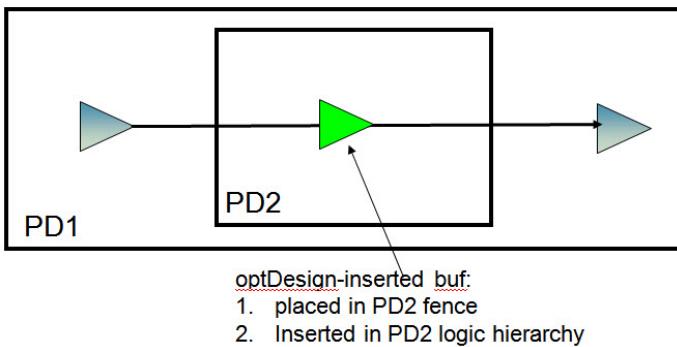
The goal of ISO/LS placement is to increase the chance that optimization can buffer either input or output net using as many regular buffers as possible. The placement needs to place ISO/LS close to power domain boundary or near its driver/receiver.



You can control ISO/LS placement or fix ISO/LS by using `setPlaceMode`, creating ISO/LS group constraints, setting the ISO/LS cell padding, creating the routing blockage along domain boundary and creating the pin guide.

# Low Power Optimization

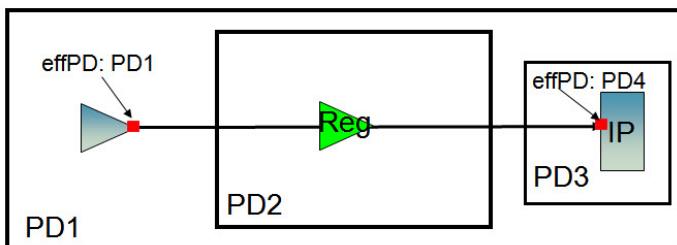
- *optDesign* is Power Domain (PD) aware and inserts buffers along the route
  - Extra physical and logic constraints must be applied in LP Optimization
    - Physical constraint: Buffers needs to be placed in its domain fence
    - Logic constraint: Buffer needs to be pushed down/pulled up to its domain logic hierarchy



## Example

# Regular buffer insertion

- Determine the location: Green
  - Get the location PD: PD2
  - Power (domain) coverage analysis
    - Can PD1 drive PD2 and PD2 drive PD4 (no off->on or low->high)
    - Yes, Use Reg buffer
  - Pick up Req buffer from PD2 lib binding; insert it into PD2 logic hierarchy

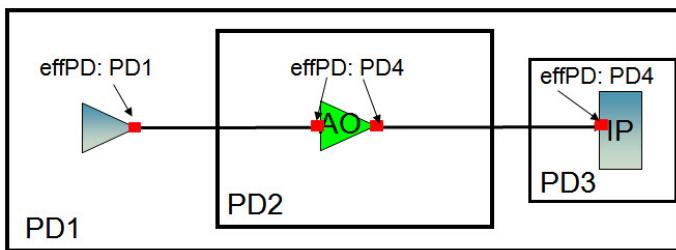


PD1 can drive PD2; PD2 can drive PD4

## Example

### Always On (AO) buffer insertion

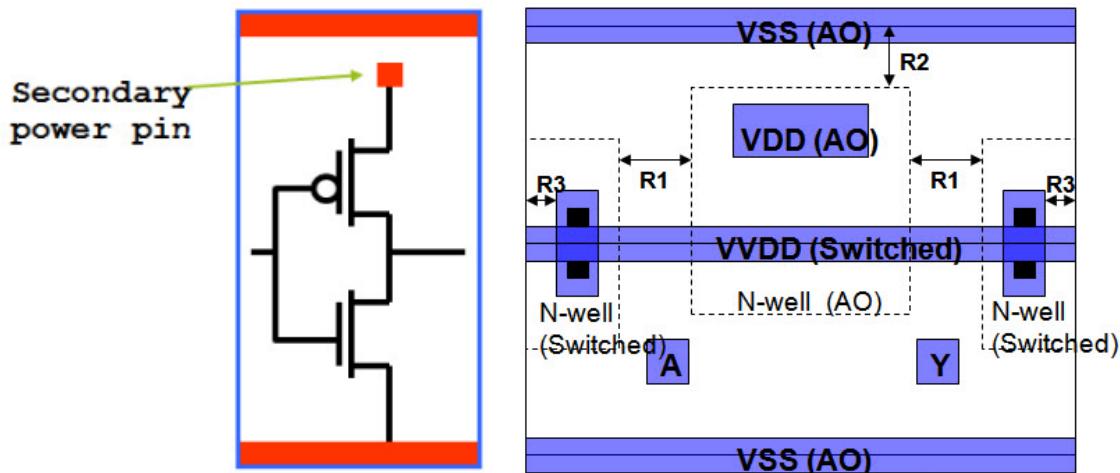
- Determine the location: Green
- Get the location PD: PD2
- Power (domain) coverage analysis
  - Can PD1 drives PD2 and PD2 drives PD4 (no off->on or low->high)
  - No, Use AO buffer
- Pick up AO buffer from PD2 lib binding; insert it into PD2 logic hierarchy
  - Connect AO 2nd power pin to PD4's power
  - Assign AO input and output pin to PD4



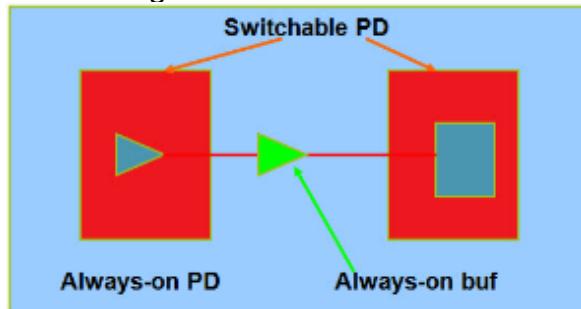
PD1 can drive PD2; PD2 can NOT drive PD4

## LP Optimization Using Always-on Buffers

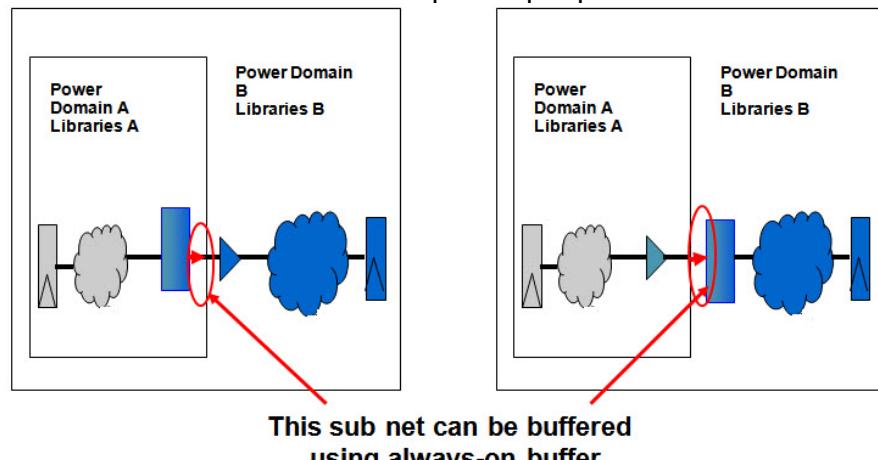
Always-on (AO) buffer is a buffer with two power/ground pins where the secondary power pin is used to supply the power for the always-on buffer. The cell can be mixed into a switched power domain because its primary power pin is compatible with the row's power rail (follow pins), has built-in Nwell for secondary and primary, powers for abutment with other cell. AO Buffers are required to maintain power domain compatibility when the local power domain is not compatible with its sink or receiver.



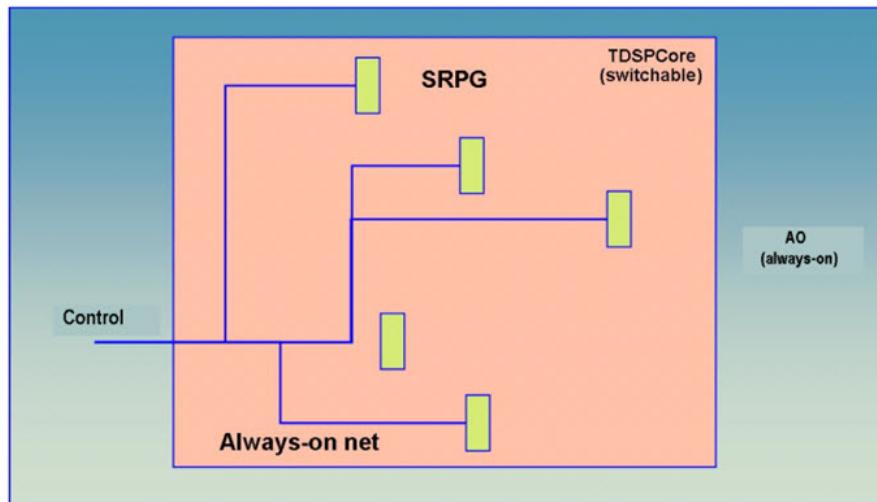
- Feedthrough Net



- The subnet between ISO/LS input/output pin and domain boundary



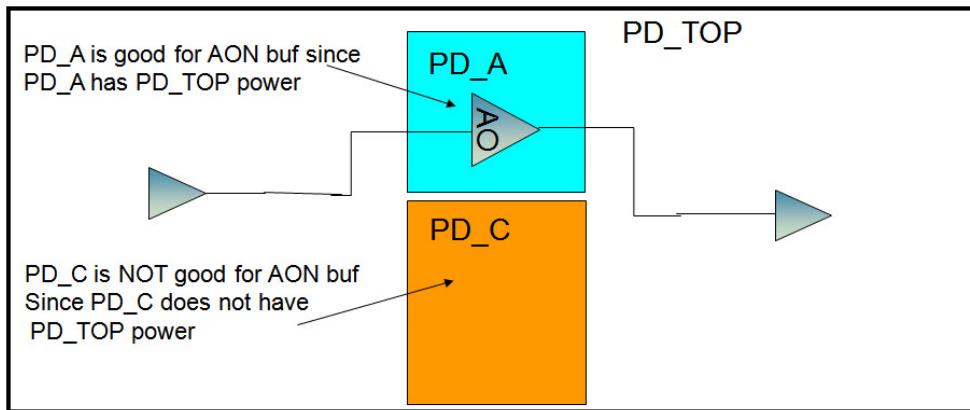
- AO net in switchable domain such as ISO, PS and SRPG enabled



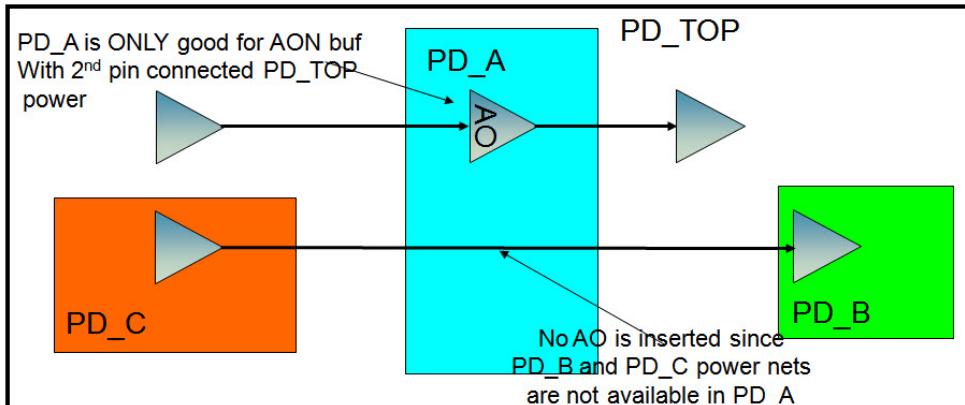
## Disable AO Buffering Based On The Unavailable Power Nets In The Specific Domain

### CPF Command:

```
update_power_domain -name PD_C -user_attribute {{disable_secondary_domains {PD_TOP}}}
```



## Enable AO Buffering Based On The Available Power Nets In The Specific Domain



### CPF Command:

```
update_power_domain -name PD_A -user_attribute {{enable_secondary_domains {PD_TOP}}}
```

PD\_A can ONLY insert AO with its second power pin connected to PD\_TOP's power

### UPF Command:

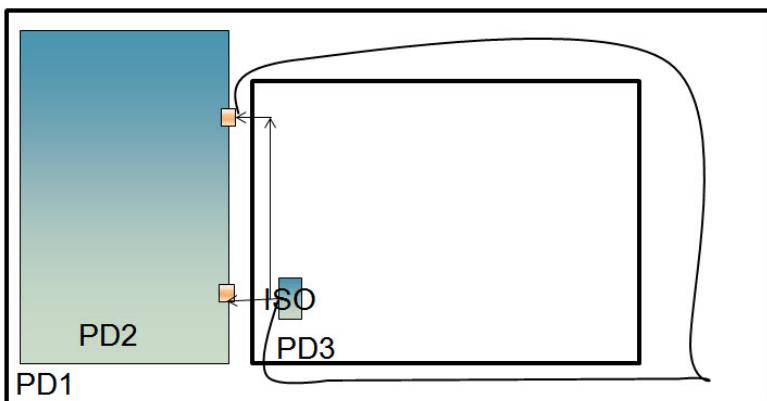
```
create_power_domain PD_A -available_supplies <PD_TOP's supply set>
```

PD\_A can ONLY insert AO with its 2nd power pin connected to PD\_TOP's power

## Regular Buffer vs. AO buffer (cost-based buffering)

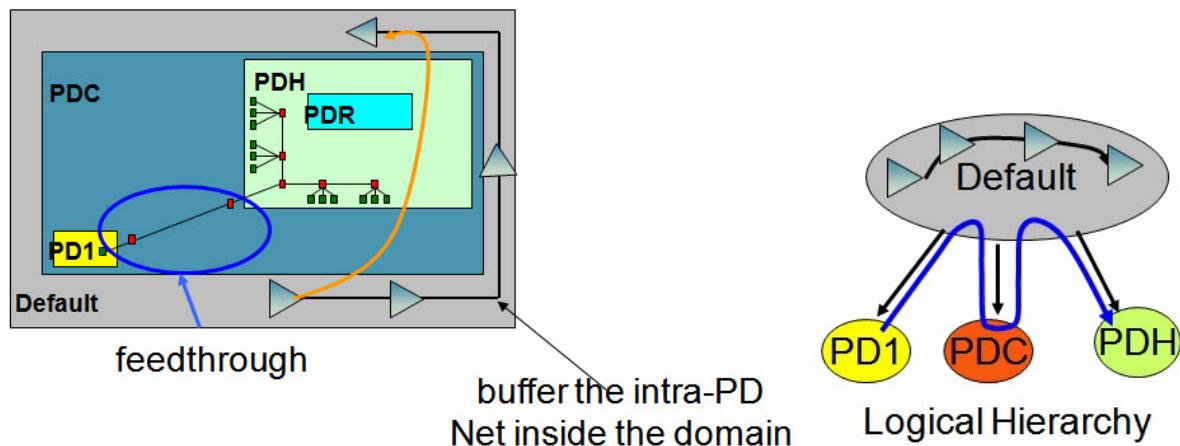
Minimize the number of AO buffering. The AO buffer is set at a higher cost.

Optimization is able to generate different topologies/routes for regular or AO buffering. It can choose which topologies/route for buffering according to a pre-defined cost function.



## bufferTreeSynthesis (Optional)

Buffer Tree Synthesis uses the CTS engine. It can be used to buffer the high fanout nets or long nets or feedthrough nets and is not required to follow the routing path.

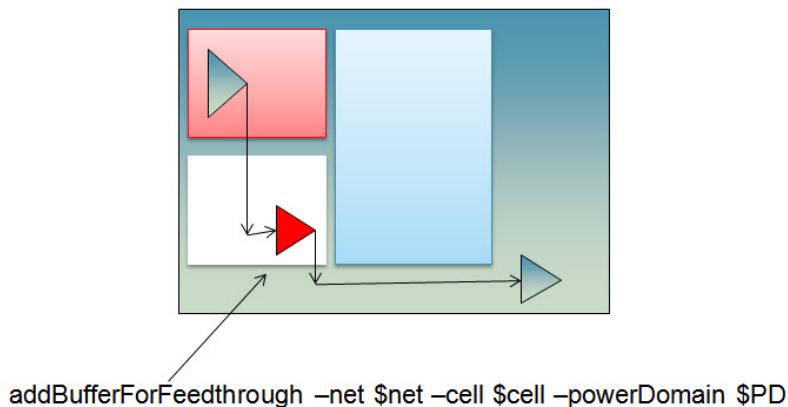


## ***addBufferForFeedthrough - A Handy Interactive Command***

The *addBufferForFeedthrough* command provides flexibility to buffer the crossing domain nets or build buffer trees in any arbitrary logic hierarchy.

### Example

**Addition of *guide* buffer for the crossing domain net to help choose the buffer path**



## Secondary Power Pin Routing

- The following LP cells have the secondary power pins:
  - Always-on buffers, SRPG, Level Shifters, Isolation Cells, Comb Cell and Power Switch Cell
  - The cell's secondary power pin is defined in the CPF or \*lib
  - The second power pin connection is defined in CPF or IEEE1801 or floorplan file
  - This command can be used after Low Power cells are inserted and placed
- Route second power pin
  - The power switch second power pin is routed by power planning command *addStripe*
  - The second power pins for the other cells are routed by nanoroute commands *routePGPinUseSignalRoute* and *setNanoRouteMode*
- Power planning for second power pin route
  - Must add the second power stripes for the switchable domain
  - Need to add some other power stripes for the feedthrough AO buffering
- ECO routing in `optDesign -postRoute` can automatically do secondary power pin ECO route. ECO routes for the cells and pins are defined in `setPGPinUseSignalRoute cell1:pin1 cell2:pin2`
- Route secondary power pins before signal route

## Low Power Design Verification

- The design must match the power intent such as:
  - Isolation cell is required from Off to On
  - Level shifter is required from Low to High
- To verify Low Power design against CPF/IEEE1801 by CPF/IEEE1801 rules, use the low Power verification command *verifyPowerDomain*
- To verify Low Power design against CPF/IEEE1801 by tracing PG connection in the physical netlist, use the command *runCLP (physical)*

## Low Power Debugging Commands

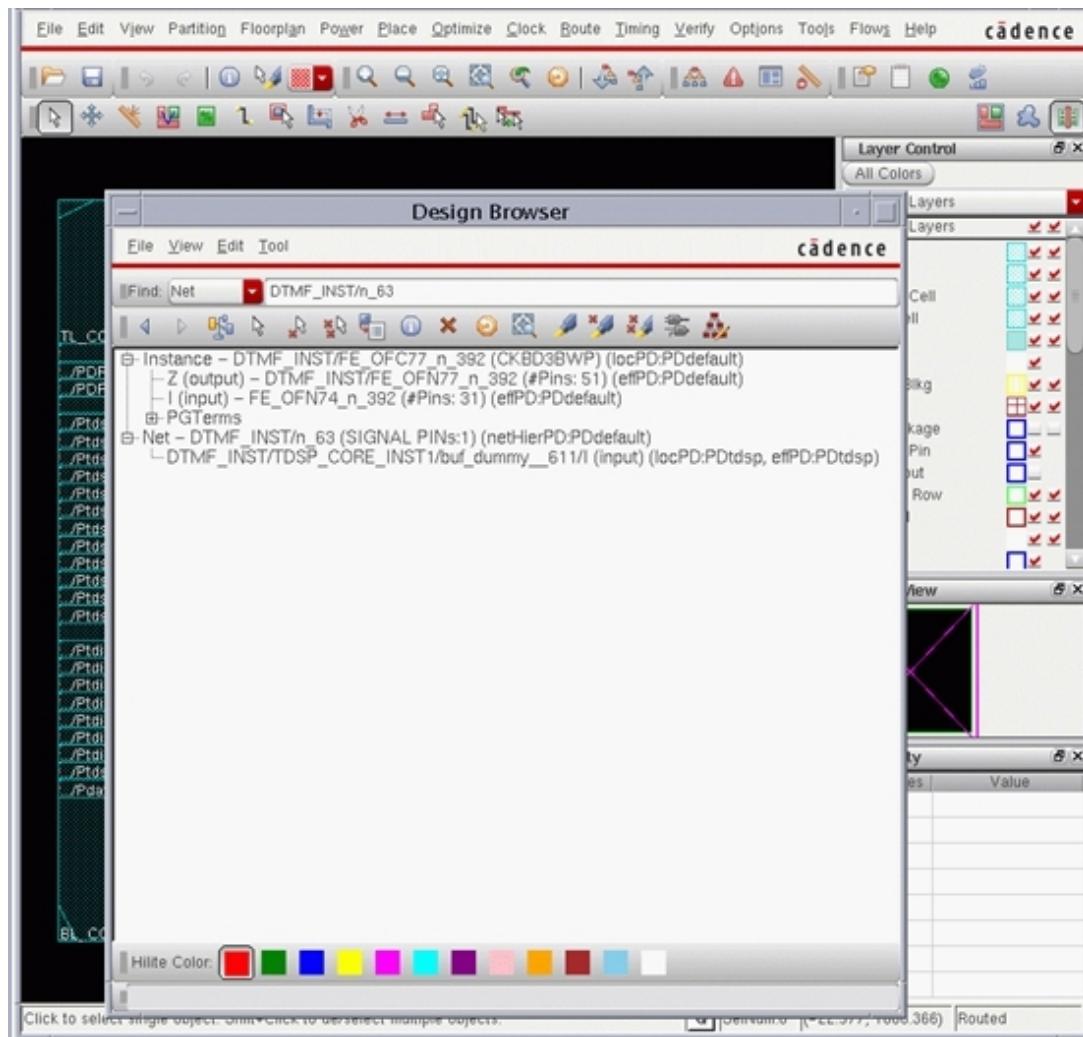
To get information about driver/receiver domain, domain coverage, pin power domain, related pg pin ...  
`reportPowerDomain -inst|-net|-pin|-powerDomain -verbose`

**Tcl db access:**

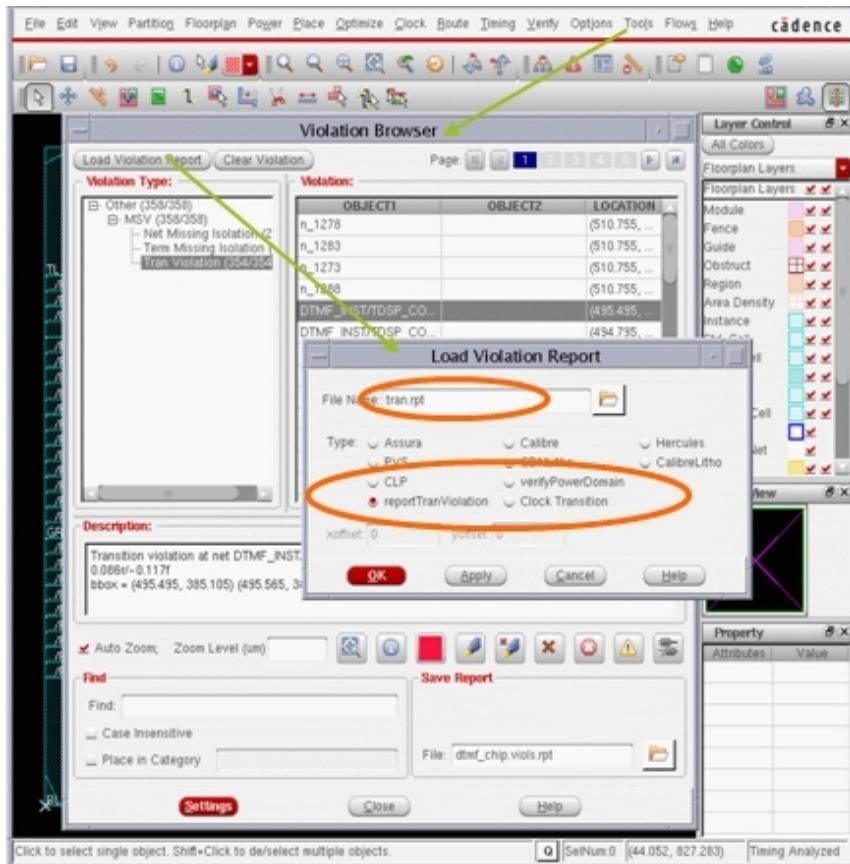
```
dbGet top.pds.??
```

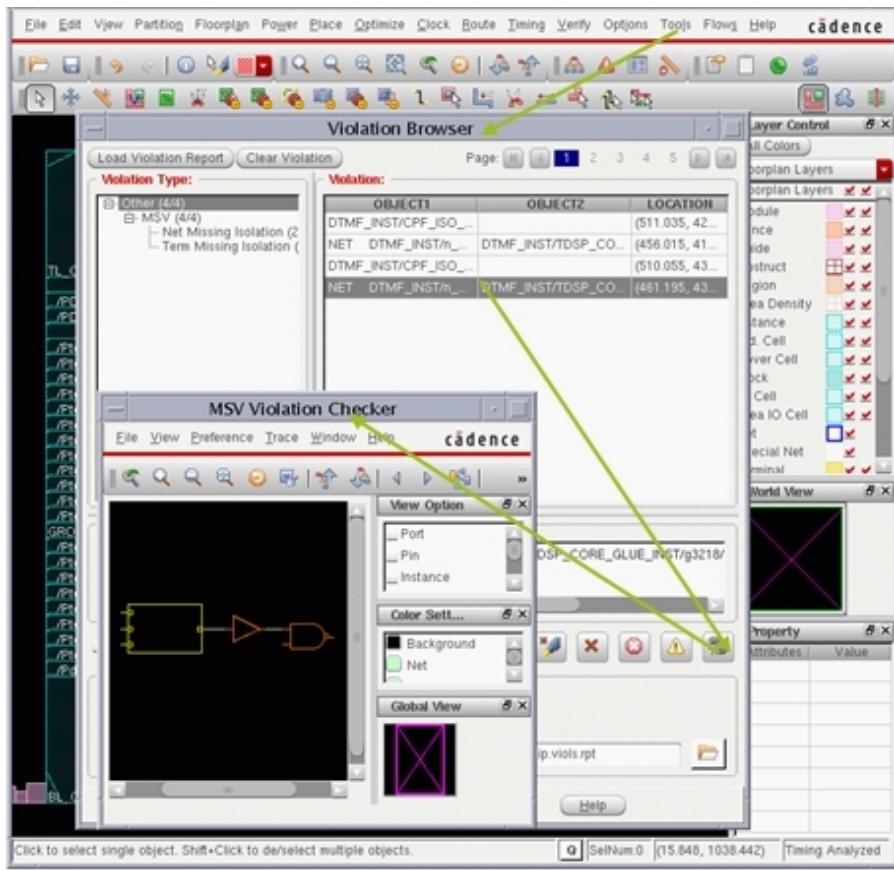
## Low Power GUI Debugger

### Design Browser



## Violation browser and Schematic Viewer Screen Capture





## Multiple Supply Voltage Top-Down Hierarchical Flow

This section discusses the following topics:

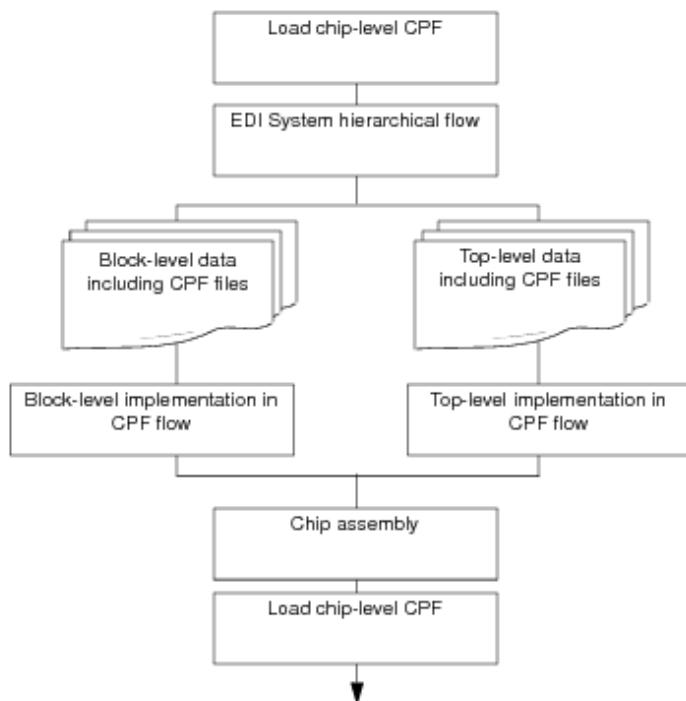
- Overview
- Always-On Feedthrough Handling
- Chip Partitioning
- Block-level CPF Generation
- Top-Level CPF Generation
- Transition to OpenAccess for Low Power Flow
- Block-Level Implementation
- Top-Level Implementation
- Chip Assembly

## Overview

The Innovus tool supports the low power top-down CPF-based hierarchical flow built on the regular Innovus hierarchical flow. The difference is that, in the CPF-based hierarchical flow, you partition the chip-level CPF file into block-level CPF files and a top-level CPF file. You then use those CPF files to implement the block level and top level designs.

The CPF-based hierarchical flow supports the following scenarios:

- The partition is physically the same as the power domain. The hierarchical instance is logically the same for power domain and partition.
- The power domain is physically inside partition. The power domain hierarchical instance is logically a sub hierarchical instance of partition.
- Partition is physically inside power domain. The partition hierarchical instance is logically one (and only one) of the hierarchical instances of power domain.



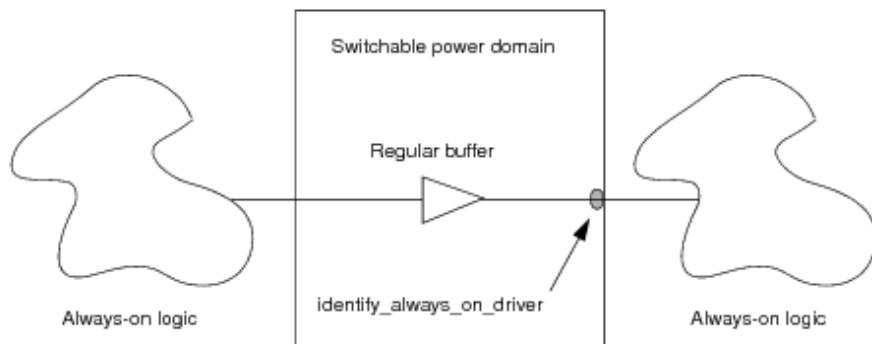
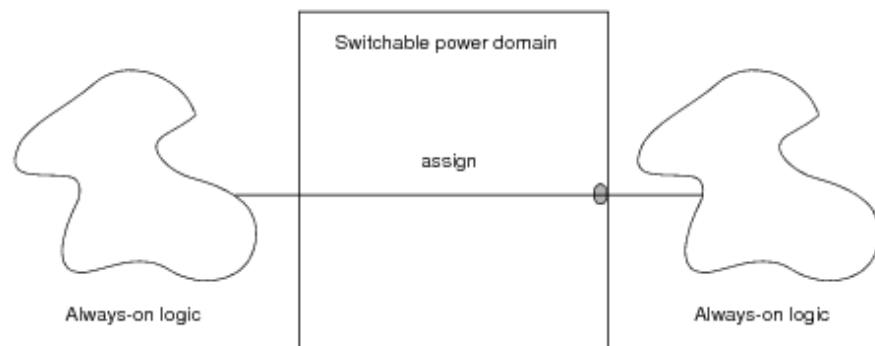
## Always-On Feedthrough Handling

In the low power flow, when you commit CPF, the tool replaces assign statements with always-on buffers for the net connecting to always-on logic or a net whose driver is always-on. The tool identifies these nets automatically as follows:

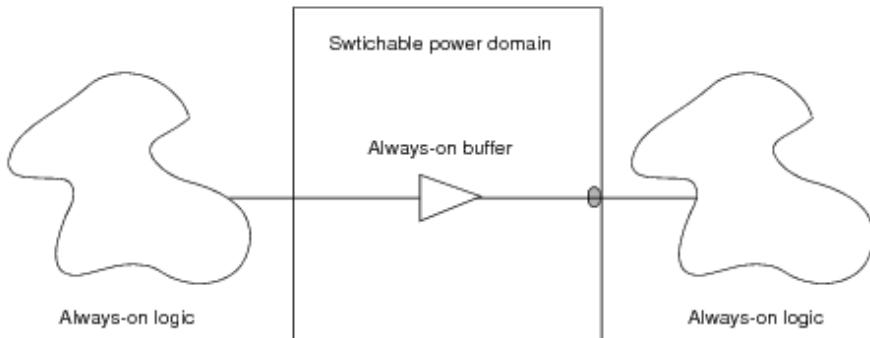
- The driver of the feedthrough and the feedthrough path are always on
- The output of the feedthrough is defined by `identify_always_on_driver` in the chip-level CPF file

The tool can also insert a feedthrough buffer in a partition that resides in a shut-off power domain.

If a feedthrough already exists in an input netlist (for example, a netlist for some reused blocks), the feedthrough may use the assign statements and regular buffers. The tool identifies the always-on feedthroughs and replaces the regular buffers or assign statements with always-on buffers defined in the chip-level CPF when you commit the CPF file.



If you use `insertPtnFeedthrough` to insert a feedthrough in the hierarchical flow, the tool can pick up always-on buffers defined in CPF for the scenario shown in the following figure:



## Chip Partitioning

Low power hierarchical partitioning with CPF is a combination of the Multi-Mode Multi-Corner (MMMC) and CPF flows. Do the following:

1. Derive the MMMC timing budget by `deriveTimingBudget` and `saveTimingBudget`. For more information, see the "[Timing Budgeting](#)" chapter of the *Innovus User Guide*.
2. Use `savePartition` to generates CPF files for each block-level design (partition), and a top-level CPF file for top-level design.
3. (Optional) If the power domain is inside the partition, save the floorplan file for chip assembly.

## Block-level CPF Generation

Block-level CPF is used to implement block-level design and determine the power domain attribute for the partition boundary pin at top-level implementation. When you commit CPF, the tool generates block-level CPF from the chip-level CPF file as follows:

- Low power information in CPF
  - Pushes down naming style, hierarchy separator, CPF version, and library set definitions
  - Pushes down low power cell definitions
  - Creates the power domains referenced by the block-level CPF files
  - Creates the power domains' power/ground nets and connections
  - Pushes down the scope-related rules or commands such as state retention rules, power switch rules, and `identify_always_on_driver` rules
- For level shifter and isolation rules:
  - If the rules specify the shifter and isolation are added into a block, the tool pushes down those rules into block-level CPF

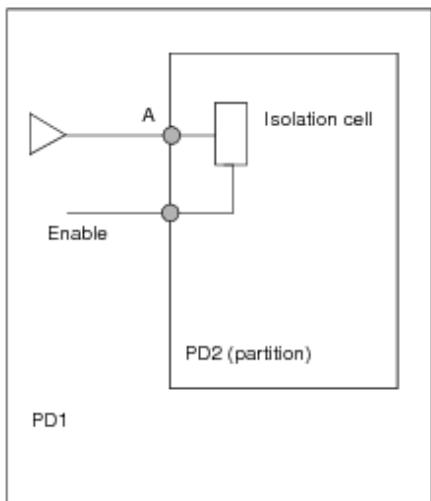
- If the rules specify the shifter and isolation are added outside a block, the tool does not push down those rules into block-level CPF.
- Assigns the power domain attribute to each partition boundary pin
- Pushes down all the nominal conditions and power modes
- Creates virtual ports to control low power logic by using `set_design -ports`

**Note:** Virtual ports do not exist in the netlist, but are needed to enable power switch, isolation, state-retention logic, and so on, in the block level

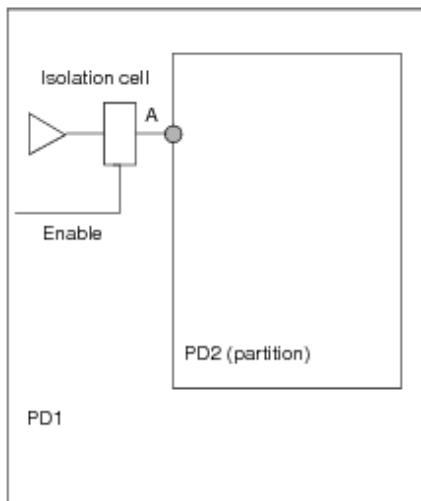
- MMMC information in CPF

The analysis views, operating conditions and power domain library binding are written into the `viewdefinition.tcl` file by timing budget commands as part of MMMC setup. Block-level CPF does not include this information.

- i** The tool marks the partition boundary pin power domain attribute that will determine whether there is a domain-crossing for the net connecting to the pin. The following example shows how the tool marks the power domain attribute for the partition boundary pin and determines whether to push down the isolation rule:



1. Port A is marked as PD1 in the block-level CPF
2. The isolation rule is pushed down



1. Port A is marked as PD2 in the block-level CPF
2. The isolation rule is kept at the top level

For always-on feedthroughs, the tool automatically traces through the feedthrough and assigns both the input and output pins of the feedthrough as always on.

For the net connecting to the partition boundary pin without an isolation or level shifter cell, the tool assigns the pin to the power domain of its driver domain.

## Top-Level CPF Generation

The tool generates top-level CPF from the chip-level CPF file as follows:

- Define each block-level boundary power domain information through `create_power_domain -boundary_ports`.
- Retains isolation or level shifter rules when the isolation or level shifter is inserted at top level.
- Discards rules in block-level scope such as state retention and power switch rules.
- Retains library sets, cell definitions, nominal condition, power mode and MMMC views at the chip level.

## Block-Level Implementation

1. Implement the block-level design with an MMMC flow, using the block-level CPF file generated at the partitioning step and the `viewdefinition.tcl` file created by `deriveTimingBudget`.
2. After completing block-level implementation, use `saveDesign -def` to save the block-level design in `def` format for chip assembly.

## Top-Level Implementation

1. Implement the top-level design with an MMMC flow, using the top-level CPF file generated at the partition step and the `viewdefinition.tcl` file created by `deriveTimingBudget`.
2. After completing top-level implementation, use `saveDesign -def` to save the top-level design in `def` format for chip assembly.

## Chip Assembly

Chip assembly assembles the physical data you generated at the block level and block level.

1. If there is a power domain inside partition, specify the chip-level floorplan with `assembleDesign -`

chipFP.

2. Load the chip-level CPF after the assembly to restore the low power settings and continue to chip-level verification and chip finishing.

## Example of Block-Level CPF Generated by Innovus

**Note:** A special construct is used to avoid duplicate definition for timing-related CPF files when sourced by top-level CPF. If the `[set_instance] == [set_hierarchy_separator]` condition is True, then the tool recognizes the implementation is at the block level, and loads the related timing information. If the condition is False, then the tools sources the block-level CPF file at top level, and does not load the timing-related information.

```
set_design tdsp_core \
-ports {n_41}

set_hierarchy_separator "/"

create_power_domain -name TDSPCore -default \
-shutoff_condition {n_41}

create_power_nets -nets VDD_TDSPCore \
-voltage 0.792 \
-internal

update_power_domain -name TDSPCore \
-internal_power_net {VDD_TDSPCore}

create_power_nets -nets VDD_TDSPCore_R \
-voltage 0.792

create_global_connection -net VDD_TDSPCore_R \
-domain TDSPCore \
-pins TVDD

create_global_connection -net VDD_TDSPCore \
-domain TDSPCore \
-pins VDD

create_ground_nets -nets VSS

create_global_connection -net VSS \
-domain TDSPCore \
-pins VSS

create_power_domain -name AO
-boundary_ports { clk reset SRPG_PG_in SRPG_PG_in_1 DFT_sen n_41, ...}
```

```
create_power_nets -nets VDD -voltage 0.792
update_power_domain -name AO -internal_power_net {VDD}
if {[set_instance]==[set_hierarchy_separator]} {
define_library_set -name ao_wc_0v99
-libraries { ../../LIBS/N45/timing/wc.lib}

define_library_set -name ao_wc_0v792 \
-libraries { ../../LIBS/N45/timing/AOwc0d72.lib}

define_library_set -name tdsp_wc_0v792 \
-libraries { ../../LIBS/N45/timing/wc0d72.lib}

create_operating_corner -name WC08COM_AO \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set ao_wc_0v792

create_operating_corner -name WC08COM_TDSP \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set tdsp_wc_0v792

create_nominal_condition -name low_ao -voltage 0.792
update_nominal_condition -name low_ao -library_set ao_wc_0v792
create_nominal_condition -name off -voltage 0

create_power_mode -name PM_LO_FUNC \
-domain_conditions { AO@low_ao TDSPCore@off} \
-default

update_power_mode -name PM_LO_FUNC \
-sdc_files { ../../RELEASE/mmmc/dtmf_recvr_core_dull.sdc}

create_analysis_view -name AV_LO_FUNC_MAX_RC1 \
-mode PM_LO_FUNC \
-domain_corners { AO@WC08COM_AO TDSPCore@WC08COM_TDSP}
}

define_isolation_cell -cells {LVLLH} \
-ground {VSS} \
-enable {NSLEEP} \
-valid_location {to} \
-power {VDD}
```

```
define_level_shifter_cell -cells {LVLH} -valid_location {to} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099}
-ground {VSS} -direction {down}

define_level_shifter_cell -cells {PTLVLH} \
-valid_location {to} \
-direction {down} \
-output_power_pin {TVDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} -input_voltage_range {0.792:0.99:0.099}

define_level_shifter_cell -cells {LVLLH} \
-valid_location {to} \
-input_power_pin {VDDL} \
-output_power_pin {VDD}
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-output_voltage_input_pin {NSLEEP} \
-ground {VSS} \
-direction {up}

define_level_shifter_cell -cells {LVLLHD} \
-input_voltage_range {0.792:0.99:0.099} \
-valid_location {to} \
-direction {up} \
-output_power_pin {VDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_power_pin {VDDL}

define_state_retention_cell -cells {RSDF} \
-ground {VSS} \
-save_function {SAVE} -power {TVDD} \
-restore_function {!NRESTORE} \
-clock_pin {CP} \
-power_switchable {VDD}

define_power_switch_cell -cells {HDRDID HDRDIA}
-stage_2_enable {!NSLEEPIN2} \
-stage_1_output {NSLEEPOUT1} \
-power {TVDD} \
-stage_2_output {NSLEEPOUT2} \
-power_switchable {VDD} \
-stage_1_enable {!NSLEEPIN1} \
```

```
-type {header}

define_always_on_cell -cells {PTBUFF PTLVLH} \
-ground {VSS} \
-power {TVDD} \
-power_switchable {VDD}

create_level_shifter_rule -name LSRULE_H2L \
-to {TDSPCore} \
-from {AO} \
-exclude {n_41 SRPG_PG_in SRPG_PG_in_1}

update_level_shifter_rules -names LSRULE_H2L \
-cells {LVLH} \
-location {to}

create_level_shifter_rule -name LSRULE_H2L_AO
-from {AO} \
-to {TDSPCore} \
-pins {n_41 SRPG_PG_in SRPG_PG_in_1}

update_level_shifter_rules -names LSRULE_H2L_AO
-location {to} \
-cells {PTLVLH}

create_state_retention_rule -name SRPG_TDSP \
-save_edge {SRPG_PG_in} \
-domain {TDSPCore} \
-restore_edge {!SRPG_PG_in_1}

update_state_retention_rules -names SRPG_TDSP \
-cell {RSDF} \
-library_set {tdsp_wc_0v792}

create_power_switch_rule -name TDSPCore_SW \
-domain {TDSPCore} \
-external_power_net {VDD_TDSPCore_R}

update_power_switch_rule -name TDSPCore_SW \
-prefix {CDN_SW_} \
-cells {HDRDID} \
-acknowledge_receiver {switch_en_out}

end_design
```

## Example of Top-Level CPF Generated by Innovus

```
.  
set_cpf_version 1.0  
set_design dtmf_recv_core  
set_hierarchy_separator "/"  
create_ground_nets -nets Avss  
create_ground_nets -nets VSS  
create_power_nets -nets VDD \  
-voltage 0.792  
create_power_nets -nets Avdd \  
-voltage 0.990  
create_power_nets -nets VDD_TDSPCore_R \  
-voltage 0.792  
create_power_nets -nets VDD_TDSPCore \  
-voltage 0.792 \  
-internal  
define_library_set -name ao_wc_0v99 \  
-libraries { ../../LIBS/N45/timing/tcbn45lpbwp_c060907wc.lib}  
define_library_set -name ao_wc_0v792 \  
-libraries { ../../LIBS/N45/timing/tcbn45lpbwp_c060907wc0d72.lib}  
define_library_set -name tdsp_wc_0v792 \  
-libraries { ../../LIBS/N45/timing/tcbn45lpbwp_c060907wc0d72.lib}  
source cpf_1.0_hierarchical_PD.tcl  
set_instance TDSP_CORE_INST \  
-port_mapping { {n_41 PM_INST/power_switch_enable} } \  
-domain_mapping { {TDSPCore TDSPCore} {AO AO} }  
source TDSP_CORE_INST.cpf  
create_power_domain -name TDSPCore \  
-shutoff_condition {PM_INST/power_switch_enable}  
create_global_connection -net VDD_TDSPCore_R \  
-domain TDSPCore \  
-pins TVDD  
create_global_connection -net VDD_TDSPCore \  
-
```

```
-domain TDSPCore \
-pins VDD

create_global_connection -net VSS \
-domain TDSPCore \
-pins VSS

create_power_domain -name AO \
-default

update_power_domain -name AO \
-internal_power_net {VDD}

create_global_connection -net VDD_TDSPCore \
-domain AO \
-pins VDDL

create_global_connection -net VDD \
-domain AO \
-pins VDD

create_global_connection -net VSS \
-domain AO \
-pins VSS

create_power_domain -name PLL \
-instances { PLLCLK_INST}

update_power_domain -name PLL \
-internal_power_net {Avdd}

create_global_connection -net VDD \
-domain PLL \
-pins VDDL

create_global_connection -net Avdd \
-domain PLL \
-pins avdd!

create_global_connection -net Avdd \
-domain PLL \
-pins VDD

create_global_connection -net Avss \
-domain PLL \
-pins VSS

create_global_connection -net Avss \
-domain PLL \
-pins agnd!
```

```
create_operating_corner -name WC08COM_AO \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set ao_wc_0v792

create_operating_corner -name WC08COM_TDSP \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set tdsp_wc_0v792

create_nominal_condition -name low_ao \
-voltage 0.792

update_nominal_condition -name low_ao \
-library_set ao_wc_0v792

create_nominal_condition -name off -voltage 0

create_power_mode -name PM_LO_FUNC \
-domain_conditions { AO@low_ao PLL@high_ao TDSPCore@off }

update_power_mode -name PM_LO_FUNC \
-sdc_files {../RELEASE/mmmc/dtmf_recv_core_dull.sdc}

create_analysis_view -name AV_LO_FUNC_MAX_RC1 \
-mode PM_LO_FUNC \
-domain_corners { AO@WC08COM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP }

define_isolation_cell -cells {LVLLH} \
-ground {VSS} \
-enable {NSLEEP} \
-valid_location {to} \
-power {VDD}

define_level_shifter_cell -cells {LVLH} \
-valid_location {to} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-direction {down}

define_level_shifter_cell -cells {PTLVLH} \
-valid_location {to} \
-direction {down} \
-output_power_pin {TVDD} \
-output_voltage_range {0.792:0.99:0.099}
```

```
-ground {VSS} \
-input_voltage_range {0.792:0.99:0.099}

define_level_shifter_cell -cells {LVLLH} \
-valid_location {to} \
-input_power_pin {VDDL} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-output_voltage_input_pin {NSLEEP} \
-ground {VSS} -direction {up}

define_level_shifter_cell -cells {LVLLHD} \
-input_voltage_range {0.792:0.99:0.099} \
-valid_location {to} \
-direction {up} \
-output_power_pin {VDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_power_pin {VDDL}

define_state_retention_cell -cells {RSDF} \
-ground {VSS} \
-save_function {SAVE} \
-power {TVDD} \
-restore_function {!NRESTORE} \
-clock_pin {CP} -power_switchable {

define_power_switch_cell \
-cells {HDRDID HDRDIAO} \
-stage_2_enable {!NSLEEPIN2} \
-stage_1_output {NSLEEPOUT1} \
-power {TVDD} \
-stage_2_output {NSLEEPOUT2} \
-power_switchable {VDD} \
-stage_1_enable {!NSLEEPIN1} \
-type {header}

define_always_on_cell -cells {PTBUFF PTLVLH} \
-ground {VSS} \
-power {TVDD} \
-power_switchable {VDD}

create_isolation_rule -name ISORULE \
-from {TDSPCore} \
-isolation_condition {!PM_INST/isolation_enable} \
-isolation_output {high}
```

```
update_isolation_rules -names ISORULE \
    -location {to} \
    -cells {LVLLH}

create_level_shifter_rule \
    -name LSRULE_H2L_PLL \
    -from {PLL} \
    -to {AO}

update_level_shifter_rules -names LSRULE_H2L_PLL \
    -location {to} \
    -cells {LVLHLD}

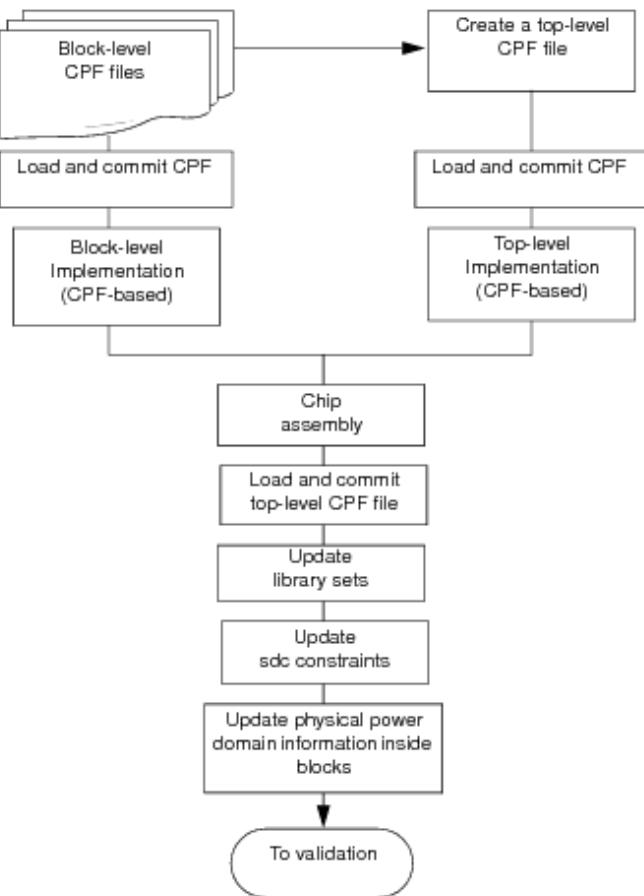
end_design
```

## Multiple Supply Voltage Bottom-Up Hierarchical Flow

This section discusses the following topics:

- Block-Level Implementation
- Top-Level Implementation
- Chip Assembly

The Innovus tool supports the low power bottom-up CPF-based hierarchical flow built on the regular Innovus bottom-up hierarchical flow. The difference is that you use the existing block-level CPF files to construct the top-level hierarchical CPF file, and implement the design using the CPF flow.



## Block-Level Implementation

You can use any combination of hard and soft blocks.

For the hard blocks (that are already implemented),

- Place the blocks in top-level floorplan.

For the soft blocks,

- Load and commit the block-level CPF files.
- Implement the blocks using the block-level CPF implementation flow.

After completing block-level implementation,

- Save the block-level design in def format for chip assembly.  
`saveDesign -def`
- If a power domain exists inside a block, use the following command to obtain the physical

information about the power domain.

```
saveCPF tmp.cpf
```

The tool restores the physical information for the power domain inside block (partition) after chip assembly.

## Top-Level Implementation

Before top-level implementation,

- Manually build the top-level CPF file by reusing the block-level CPF files as follows:  

```
Set_instance HinstOfBlock -domain_mapping { {..} } -port_mapping { {..} }  
Source block.cpf
```

The CPF file you create contains the same type of information as in the previous example file: [Example of Top-Level CPF Generated by Innovus](#).

To implement the design,

- Use the CPF implementation flow using the hierarchical top-level CPF file.

After completing top-level implementation,

- Use the following command to save the top-level design in DEF format:

```
saveDesign -def
```

## Chip Assembly

To assemble the design's physical data, use the following command:

```
assembleDesign
```

After chip assembly,

- To restore the lower power setup for chip-level verification and chip finishing, load and commit the top-level hierarchical CPF file.
- (Optional) Update power domain physical information inside block.
  - To update power domain shape, use the following command:  

```
setObjFPlanBox
```
  - To update the power domain attribute, use the following command:  

```
modifyPowerDomainAttr
```

**Note:** These steps are necessary only if you have a power domain inside a partition.

- To update the library set, use the following command:

```
update_library_set -name -timing {..}
```

- To apply chip-level timing constraints (sdc files), use the following command:

```
update_constraint_mode -name -sdc_files
```

- Proceed to design verification.

## Leakage Power Optimization Techniques

- Multi-Vth Optimization
- Substrate Biasing

### Multi-Vth Optimization

You can optimize non-critical path logic for leakage, while preserving the critical timing slack (WNS).

**Note:** Run multi-Vth optimization only after your design meets timing.

- Report the total leakage power in the design.

```
reportPower -leakage
```

- If you want to obtain a report file, run `reportPower -leakage` with the `-outfilefileName` option.

The following example is a leakage report showing the total leakage power in microwatts, along with cell usage statistics. For each library, the number of cells used in the design and the total leakage power dissipated by the cells are listed.

```
Total leakage power = 785.079708uW
Cell usage statistics:
Library normalVt, 49265 cells (64.855650%), 733.007529uW (93.367269%)
Library highVt, 26696 cells (35.144350%), 52.072179uW (6.632725%)
```

Optimize leakage power.

```
optLeakagePower
```

This command resizes low voltage threshold gates in the design to gates with a higher voltage threshold, while maintaining timing. This command only resizes cells that have positive slack. Cells that belong to any library are candidates for swapping. The `-highEffort` parameter overrides effort levels set by `setOptMode`.

**Note:** The `optLeakagePower` is a standalone command that can be used when a netlist is already

optimized in timing and on which you want to reclaim as much leakage power as possible. It is recommended that you use `optLeakagePower` without any options.

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

```
reportPower -leakage -outfile fileName
```

## Optimizing Leakage Power While Running `optDesign` (Recommended)

You can optimize non-critical path logic for leakage power by using the `setOptMode` command in the following ways:

- If leakage power optimization is a second priority after timing convergence, then use `setOptMode -powerEffort low` after `placeDesign` along with the default setting of `-leakageToDynamicRatio 1.0`. The leakage power optimization is automatically done by `optDesign`.
- If leakage power optimization is as critical as timing convergence, then use `setOptMode -powerEffort high` after `placeDesign` along with the default setting of `-leakageToDynamicRatio 1.0`. The leakage power optimization is done by each `optDesign` step in high-effort mode.

**Note:** When `setOptMode -powerEffort` is set to `low` or `high`, the hold fixing steps ensure that no low-voltage threshold gates are inserted. Only high-voltage threshold gates are used to fix the hold-time violations.

## Substrate Biasing

Substrate biasing is another technique for reducing leakage power. Changing the body voltage of the field effect transistor (FET) affects both the threshold voltage and the static leakage current.

To bias the substrate, insert biasing cells into a region of the design. In Innovus, you can do this in either of two ways:

- Use the `addWellTap` command to add bias cells at regular intervals.

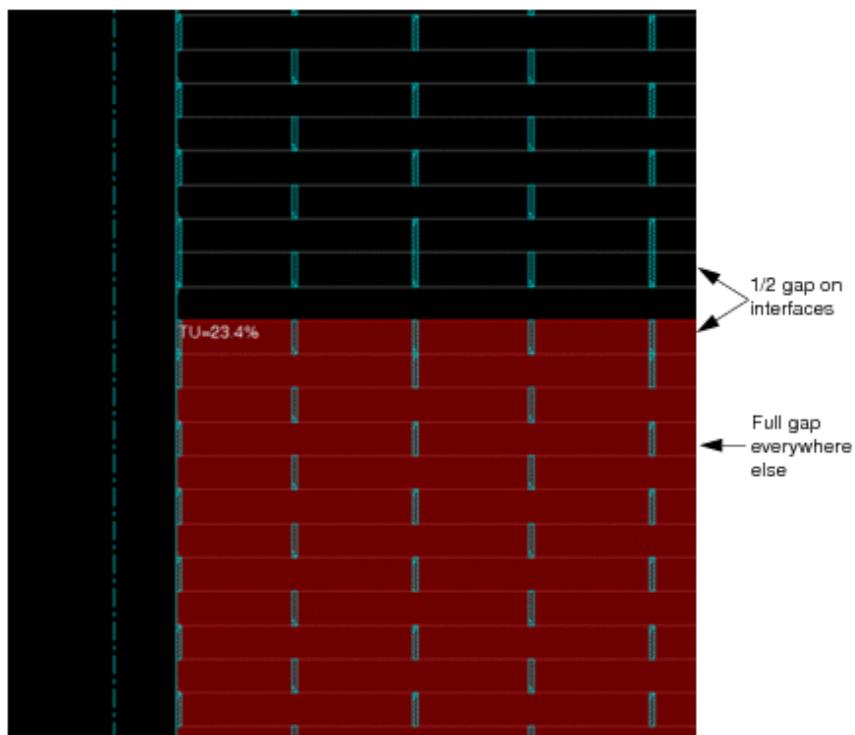
```
addWellTap -maxGap
```

- Add well taps in a checkerboard configuration; for example,

```
addWellTap -cellFILL1 -maxGap 20 -checkerBoard -fixedGap
```

```
addWellTap -cellFILL2 -maxGap 20 -powerDomain PD -checkerboard -fixedGap
```

These commands produce results such as those shown in the following figure:

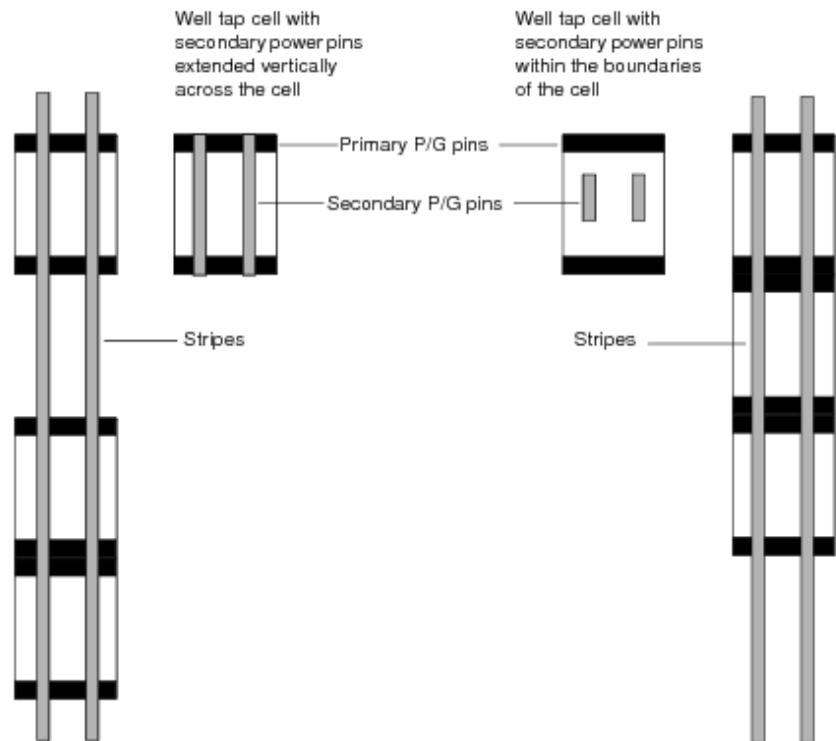


For well tap cells, you must add stripes to connect the secondary power/ground pins in the vertical or horizontal direction.

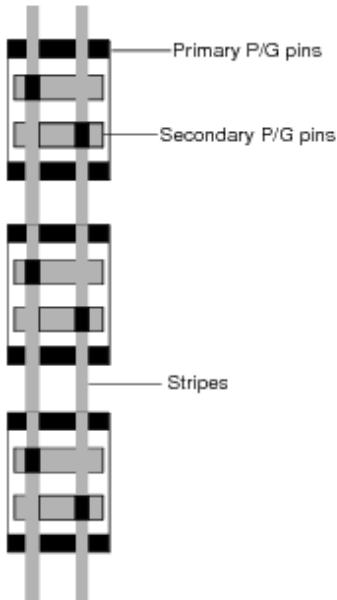
1. Select *Floorplan - Custom Power Planning - Add Stripes*.
2. On the Basic page, select *Over P/G pins*.
3. Click on *Master Name*.
4. Type the master name of the standard cell.
5. Select *Pin Layer*.

The top layer is the default.

The following figures show how well tap cells are connected. The software connects the secondary power/ground pins vertically or horizontally to the nearest secondary power/ground pins, regardless of whether the pins extend fully across the cell.



Well tap cells with secondary power pins extended horizontally across the cell



## Power Shutdown Techniques

Power shutdown is a coarse-grain methodology for performing power gating. This technique shuts off a specific power domain under certain conditions. There are two styles of this methodology:

- Ring style: All switches are inserted outside the domain.
- Column style: All switches are inserted inside the power domain.

## Data Preparation

For ring-style switch insertion, prepare the data as follows:

- Assign CLASS RING to the power switch cell in the LEF file (SYMMETRY X Y R90) . This is recommended, but not required for switch cells. No SITE information is required.
- Ensure that there are enable nets to drive the buffer inside of the power switch cell, and acknowledge nets to exit the power switch cell. These nets are used as input to the `-enableNetIn` and `-enableNetOut` options to `addPowerSwitch`.
- Specify the power/ground net and pin connects of the power switch cell.

For column-style switch insertion, prepare the data as follows:

- Assign CLASS CORE and the correct SITE definition for the switch cell in the LEF file.
- Specify the power/ground net and pin connections of the power switch cell.
- Specify the distance between the columns and switches in microns (horizontal pitch value).

For ring style, you need to know the following:

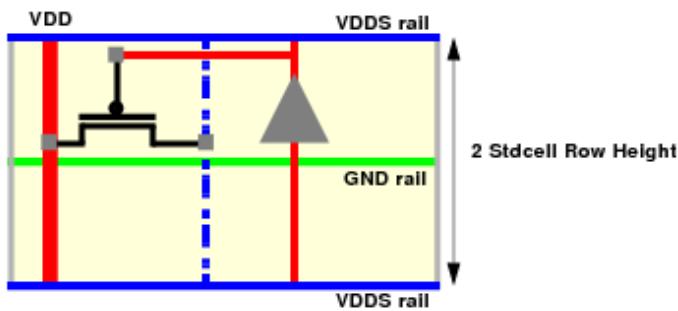
- For power planning, to ensure that the power stripes connect to the power switch cell
- NanoRoute connects enable signals. Abutment depends on the physical layout of the power switch cell.

For column style, you need to know the following:

- In the `addPowerSwitch` command, the `-enableNetOut` option can only specify one net name, which will be the net base name. The tool adds the suffix `_columnNumber` .
- The dimension of the power switch cells must be an integer multiple of a single-height standard cell.

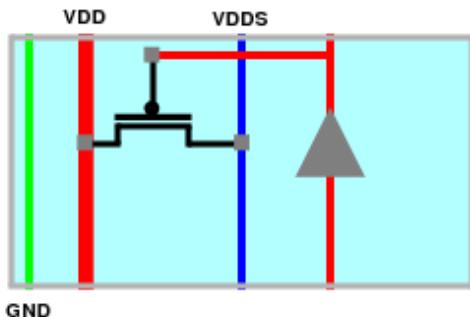
## Buffer Styles

The following figure shows column switch cell, which contains a buffer. This cell has a height of two times the standard cell height.



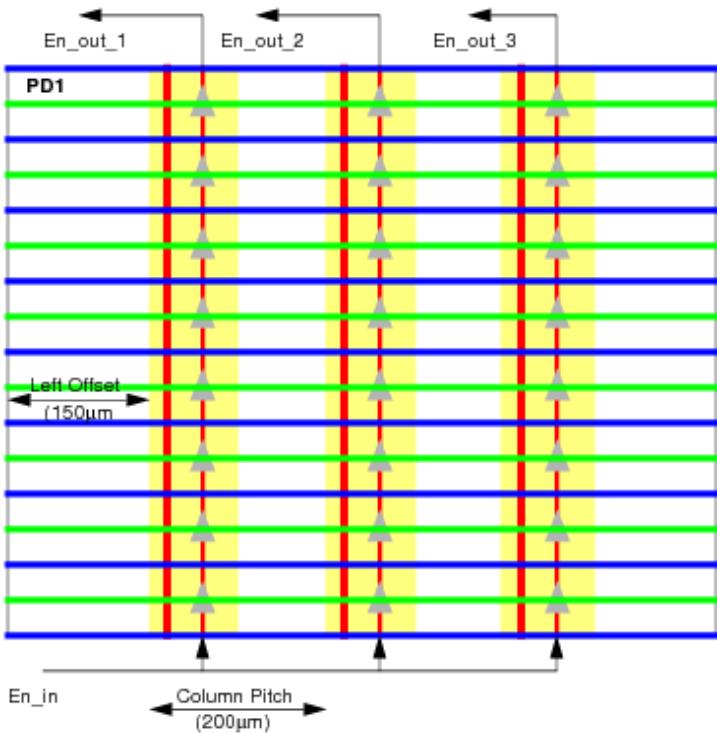
The following figure shows a ring switch cell, which contains a buffer. The cell has the same height as a standard cell. Ring switch cells can also contain two buffers with different directions.

**Ring Switch Cell**



## Adding Column Switches

The column switch methodology adds power switches entirely within the power domain. The following figure shows an example of column switches within a power domain:



To add column switches, use the following command:

```
addPowerSwitch -column
```

The following parameters are required:

- powerDomain
- enablePinIn
- enablePinOut
- enableNetIn
- enableNetOut
- globalSwitchCellName

Optionally, you can specify the following:

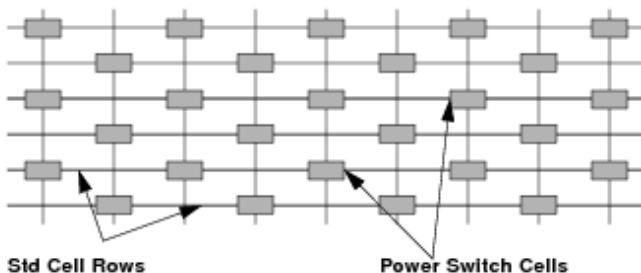
- Offsets from the top, bottom, right, and/or left side of the power domain.
- Area in which the tool can place switches.
- Power/ground pin connections.
- Many other options.

Instead of using the text command, you could use the menu command as follows:

- From the main Innovus window, choose *Power - Multiple Supply Voltage - Power Switch Insertion*, then click on the *Column* button.

- With the text command, you can place the switches in a checkerboard pattern as follows:

```
addPowerSwitch -column -checkerboard
```



This command allows you to reserve space for other uses or reduce the number of switches, for example, and possibly reduce leakage power.

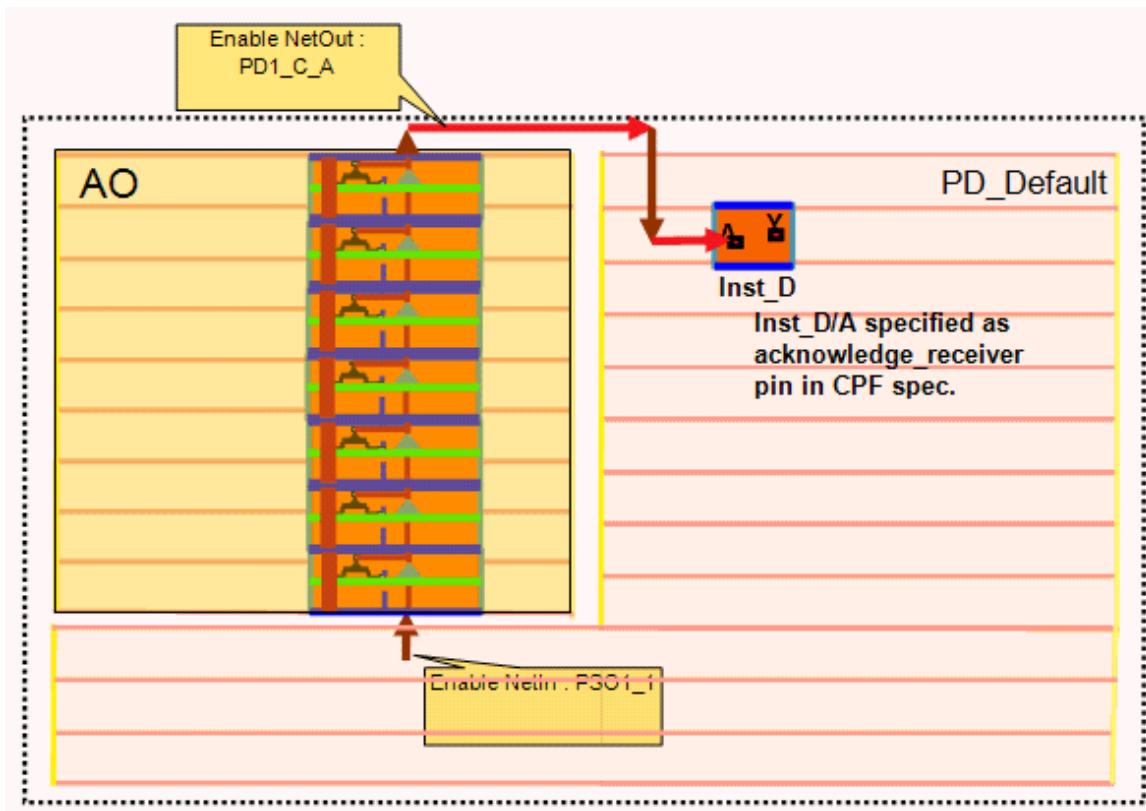
## Attaching the Acknowledge Receiver Pin

In CPF, you can specify an input pin that must be connected to an output pin of the last power switch in the chain. This information is specified in CPF as follows:

```
update_power_switch_rule -name string  
-acknowledge_receiver_pin ...
```

The `addPowerSwitch` command can connect the output pin of the last switch cell to the acknowledge pin specified by `update_power_switch_rule`.

The following figure shows `enableNetOut PD1_C_A` connected to `Inst_D` as specified in CPF:



## Example

In the CPF file:

```
create_power_switch_rule -name sw1 -domain PD1 -external_power_net VDDH
update_power_switch_rule -name sw1 -cells COLUMN_SW -acknowledge_receiver Inst_D/A
```

**Power switch insertion command:**

```
addPowerSwitch -column -powerDomain PD1 -enablePinIn SWIN -enablePinOut SWOUT -enableNetIn
PSO1_1 -globalSwitchCellName COLUMN_SW -leftOffset 3 -horizontalPitch 100
```

**Verilog file after addPowerSwitch is run:**

```
BUFXH Inst_D (.Y(switch_out), .A(PD1_C_A));
...
COLUMN_SW pso1_PD1_1_COLUMN_SW_9_52_3 (.SWOUT(PD1_C_A),
.SWIN(psoPSI_PD1_EnNet_1_3_9_49_0));
```

Inst\_D/A is connected to the PD1\_C\_A net and is connected to the SWOUT pin of the last switch.

- i** Do not specify `-enableNetOut` because this setting interferes with and overrides the `-acknowledge_receiver` specification.

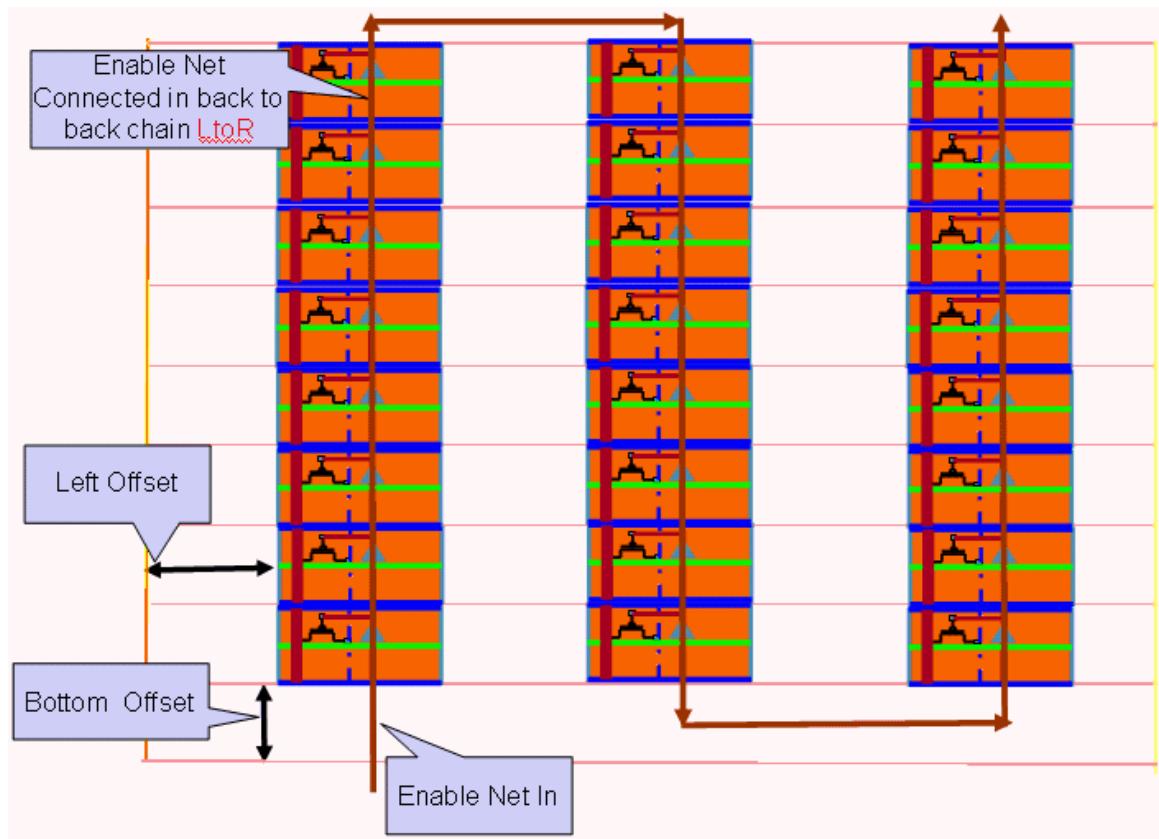
## Enable Chaining

By default, the `enableNetIn` is connected to the bottom of each column and the `enableNetOut` exits from the top of each column, in parallel. The following commands let you create columns with daisy-chain enables:

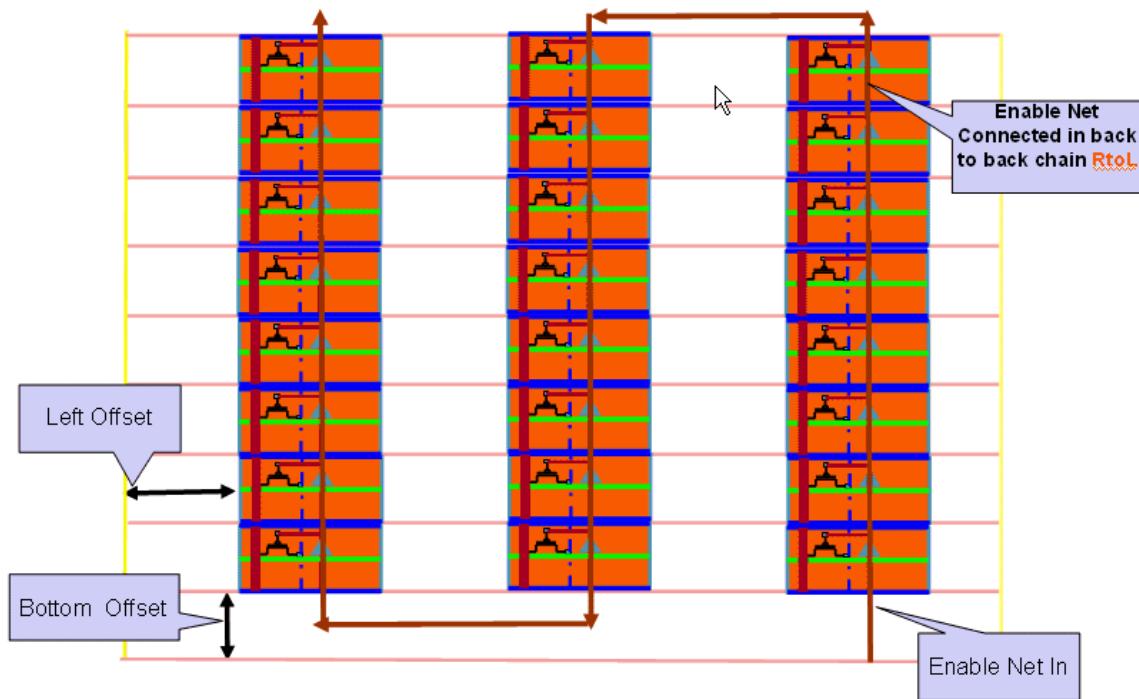
- `-backToBackChain`

Connects the `enableNetOut` at the top of a column to the `enableNetIn` at the top of the next column, and connects the `enableNetOut` at the bottom of a column to the `enableNetIn` at the bottom of the next column.

The following figure shows `-backToBackChain` with the `LtoR` (left-to-right) option:



The following figure shows -backToBackChain with the RtoL (right-to-left) option:



**Example:**

```

addPowerSwitch \
    -column \
    -powerDomain DSP \
    -switchModuleInstance dummy_dsp_1 \
    -enablePinIn {NSLEEPIN2} \
    -enablePinOut {NSLEEPOUT2} \
    -enableNetIn {UNCONNECTED249} \
    -globalSwitchCellName HDRDIHVTD2 \
    -enableNetOut {power_out_ack} \
    -leftOffset 15.0 \
    -bottomOffset 0.0 \
    -horizontalPitch 150.0 \
    -backToBackChain RtoL

● -loopbackAtEnd

```

Connects the `enablePinOut` of the last cell in the chain to the `enablePinIn` of the same cell.

In the following example, two `-enablePinIn` and `-enablePinOut` pins are specified, so you can use

`-loopbackAtEnd`:

```
addPowerSwitch \
-column \
-powerDomain DSP \
-switchModuleInstance dummy_dsp_1 \
-enablePinIn {NSLEEPIN2 NSLEEPIN1} \
-enablePinOut {NSLEEPOUT2 NSLEEPOUT1} \
-enableNetIn {UNCONNECTED249} \
-globalSwitchCellName HDRDIHVTD2 \
-enableNetOut {power_out_ack} \
-leftOffset 15.0 \
-bottomOffset 0.0 \
-horizontalPitch 150.0 \
-topDown \
-backToBackChain RtoL \
-loopbackAtEnd
```

## Controlling the Maximum Enable Chain Depth

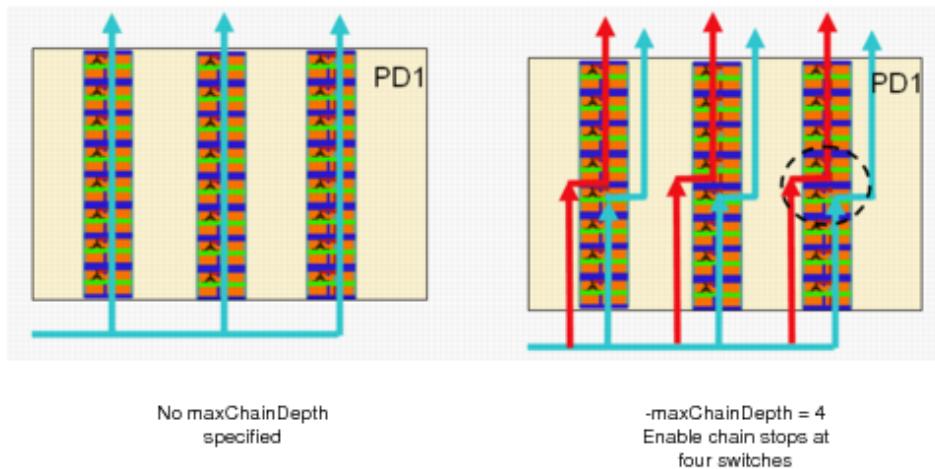
You can control the ramp-up time for the power domain by specifying the number of column switches are allowed in an enable chain before a new chain is started.

- To control the maximum number of switches in an enable chain, use the following parameter:

```
-maxChainDepth integer
```

The software then starts a new enable chain at the next switch, as shown in the following figure:

```
addPowerSwitch -column -powerDomain PD1 -maxChainDepth 4
```



## Synthesizing Acknowledge Trees

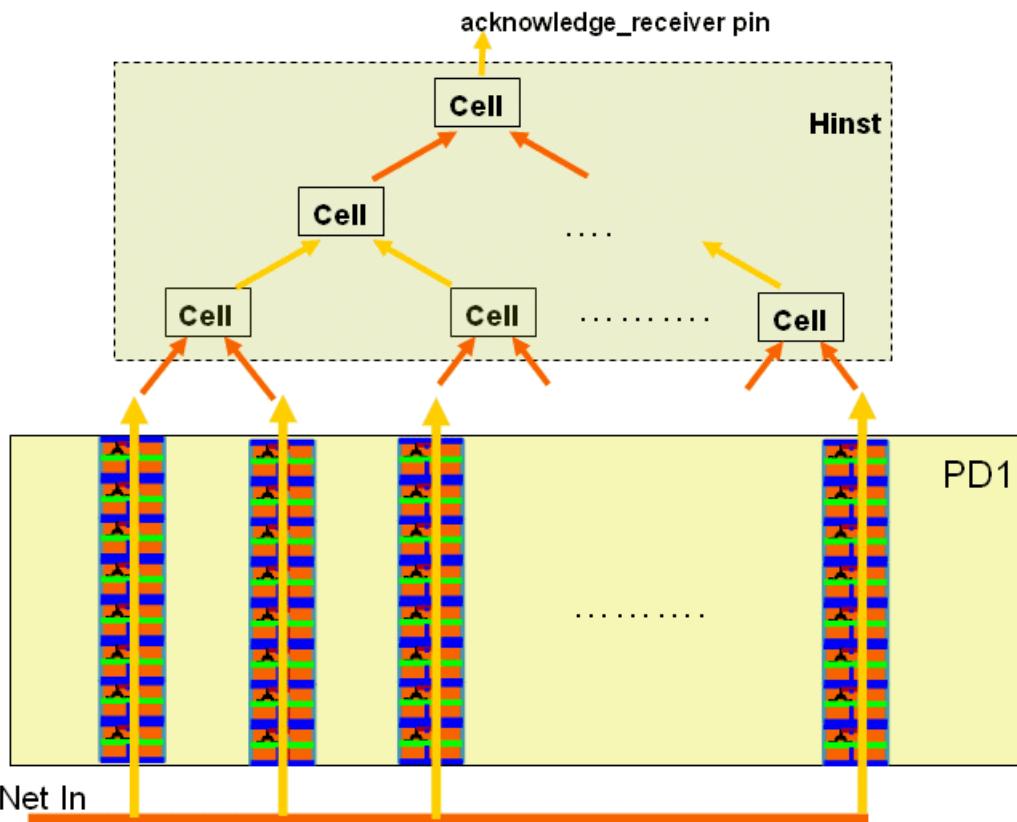
The Innovus tool can automatically create acknowledge trees that you would otherwise build manually. The acknowledge tree collects enable signals exiting the power domain and funnels them to an acknowledge receiver pin.

Use the following parameters to create the acknowledge tree:

- acknowledgeTreeCell
- acknowledgeTreeHierInstance

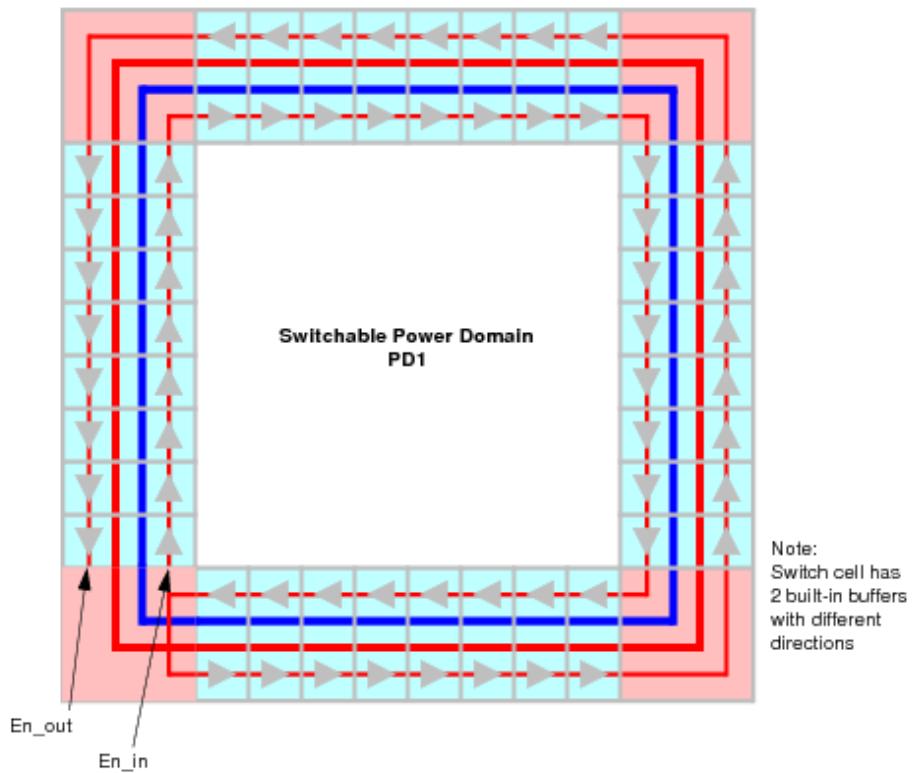
If you do not specify `-acknowledgeTreeHierInstance`, the tool places the cells in the top module.

The following figure shows an acknowledge tree built from cells of type Cell, placed in hierarchical instance HInst.



## Adding Power Switch Rings

You can add power switches in a ring entirely outside the boundary of the power domain as shown below.



In this figure, the switches abut and connect to the next switch. Because the switches in this example contain two built-in buffers with different directions, the enable net loops around the inner side of the ring, connects at the corner, and loops back around to where it becomes the enable net out.

This technique is useful when the power domain is a pre-designed macro.

To create a power switch ring, use the following command:

```
addPowerSwitch -ring
```

The following parameters are required:

```
-powerDomain  
-enablePinIn  
-enablePinOut  
-enableNetIn  
-enableNetOut
```

By default, the tool distributes cells evenly in the ring. To stack cells instead, use the following command:

```
addPowerSwitch -ring -distribute 0
```

Instead of using the text command, you could use the menu command as follows:

From the main Innovus window, select *Power- Multiple Supply Voltage - Add Power Switch*, then click on the Ring tab. Select Distribute Switches on the Switch Cell Count form.

With ring options, you have many ways of configuring the switch ring. Among many possibilities, you can do the following:

- Control how switches are distributed around the ring
- Choose the sides on which you want to add switch cells
- Specify the breaker, buffer, filler, switch, and corner cells you want to use on specified sides
- Specify the distance between the power domain and each side of the ring
- Choose the orientation of cells on specified sides
- Arrange the buffer, breaker, filler, and switch cells in a pattern

## Creating Patterns

When you create rings, you can specify a pattern that customizes switch placement. If you do not specify a pattern, the software adds switches evenly around the power domain.

The following command shows a pattern of cells that repeats on all sides:

```
addPowerSwitch -ring \
-powerDomain TDSP2 \
-enablePinIn {A0} -enablePinOut {Z0} \

enableNetIn swcontrol_2 -enableNetOut swack_2
-specifySides {1 1 1 1 1 1 1 1}
-sideOffsetList {3 3 3 3 3 3 3 3 }
-globalSwitchCellName {{CDN_RING_SW S} {CDN_RING_SW_1 D} {CDS_RING_SW_2 G}} \
-cornerCellList CDN_RING_CORNER_UL \
-globalFillerCellName {{CDN_RING_FILLER F}}
-insideCornerCellList CDN_RING_CORNER_InCell \
-globalPattern {S S D D G G F}
```

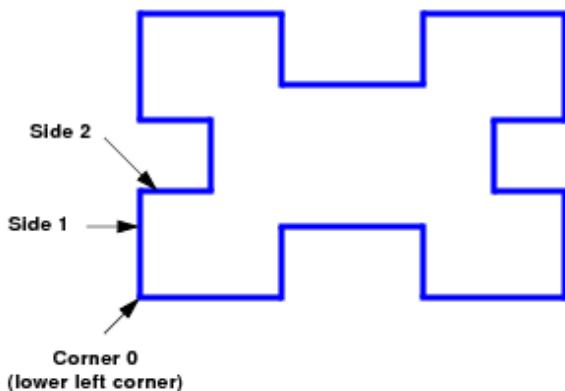
In this example, the command adds power switches in the following pattern:

```
CDN_RING_SW CDN_RING_SW CDN_RING_SW_1 CDN_RING_SW_1 CDN_RING_SW_2 CND_RING_SW_2
CDN_RING_FILLER
```

The command repeats the pattern on each side. If you want to continue the pattern on the next side or edge, use the `-continuePattern` parameter.

## Ring Conventions

The Innovus software supports rectilinear power domains, such as the 20-sided power domain shown below:



The default side/corner numbering is clockwise from the starting corner (Corner 0), which is always the lower left corner of the power domain.

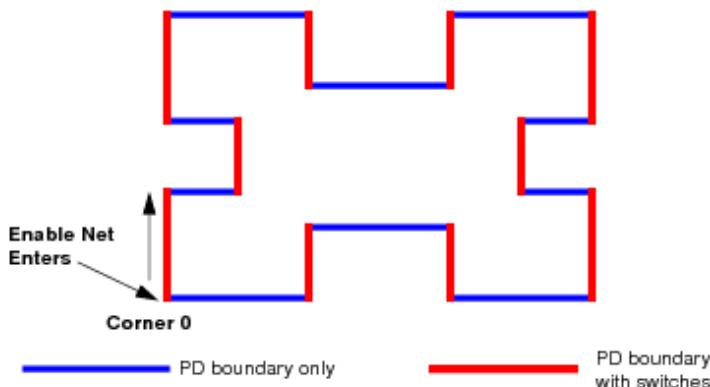
Corner numbering starts with 0. Side numbering starts with 1.

## Specifying Sides in a Switch Ring

Use `addPowerSwitch -specifySides` to add switches around a power domain of any number of sides.

Each value provided in the `-specifiedSides` parameter corresponds to a side of the power domain. The `1` value indicates that the tool should add switches on a side, and `0` indicates that it should not. For example, for switches on all sides of a 4-sided power domain, use `-specifySides {1 1 1 1}`. By default, the tool adds switches on all sides.

The following example shows how you can place switches on every other side of the 20-sided power domain.

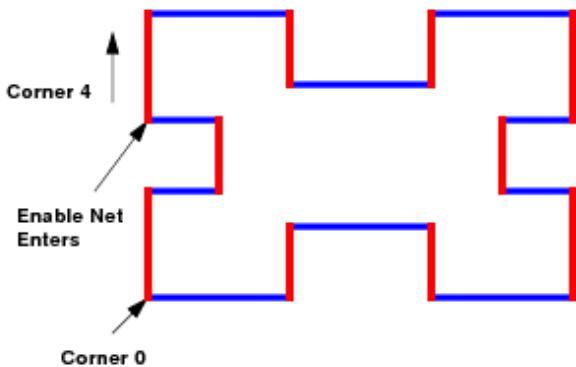


## Starting the Enable Chain at a Different Corner

By default, the enable net enters at Corner 0. To select the corner at which you want the enable net to enter, use the `-startEnableChainAtCorner`. This example does the following:

- Adds power switches on sides 1, 3, 5, 7, 9, 11, 13, 15, 17, and 19
- Starts the enable corner to corner 4.

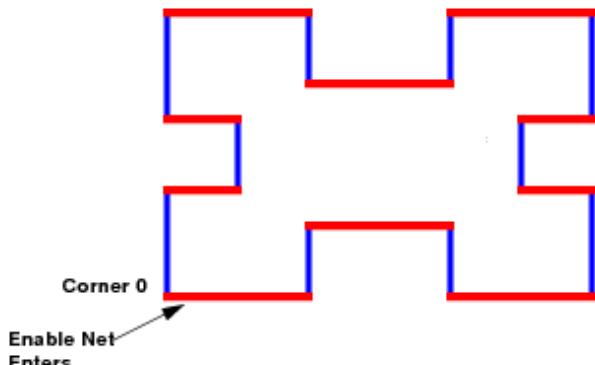
```
addPowerSwitchRing -ring -specifySides {1 0 1 0} -startEnableChainAtCorner 4
```



## Counter-Clockwise

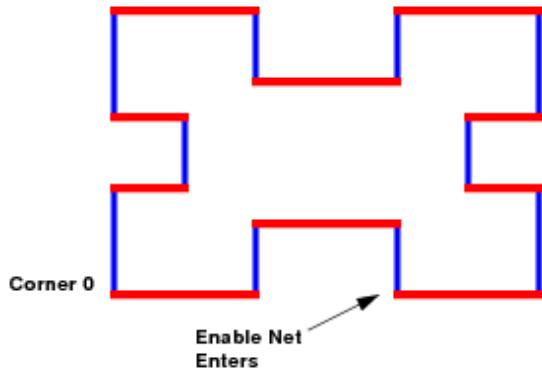
The `-counterclockwise` option reverses the corner/side numbering from Corner 0. What was side 20 in the previous example becomes side 1 in this example.

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



In the following example, side/corner numbering is counterclockwise, and the enable net enters at Corner 4. Note how location of the specified corner differs from the location of the corner specified

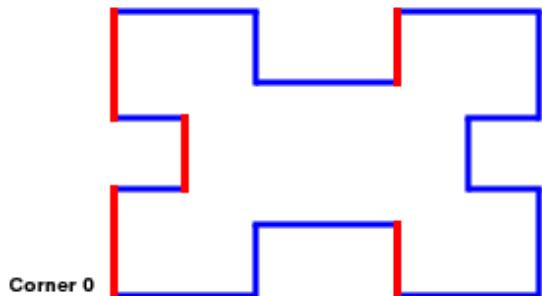
without the  $\text{-counterclockwise}$  parameter.



## **Left Sides**

The following command adds switches only to the left sides of the power domain: Sides 1, 3, 5, 9, 17.

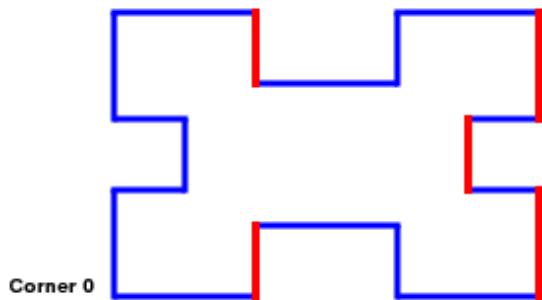
```
addPowerSwitch -ring -leftSide 1
```



## Right Sides

The following command adds switches only to the right sides of the power domain: Sides 7, 11, 13, 15, and 19.

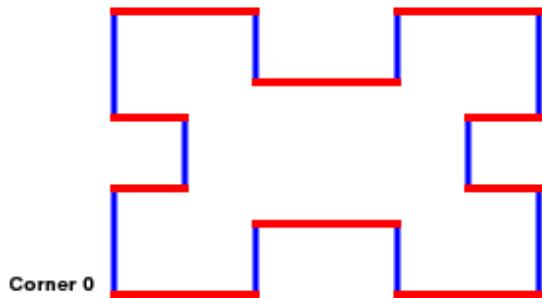
```
addPowerSwitch -ring -rightSide 1
```



## Horizontal Sides

The following command adds switches only to the horizontal sides of the power domain: Sides 2, 4, 6, 8, 10, 12, 14, 16, and 18.

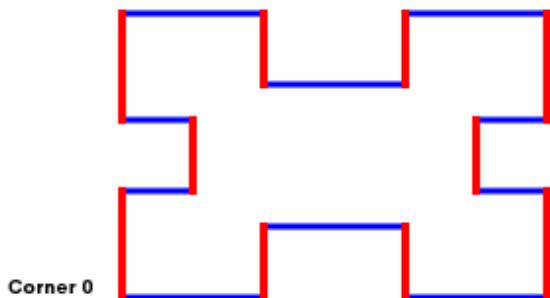
```
addPowerSwitch -ring -horizontalSide 1
```



## Vertical Sides

The following command adds switches only to the vertical sides of the power domain: Sides 1, 3, 5, 7, 9, 11, 13, 15, and 19.

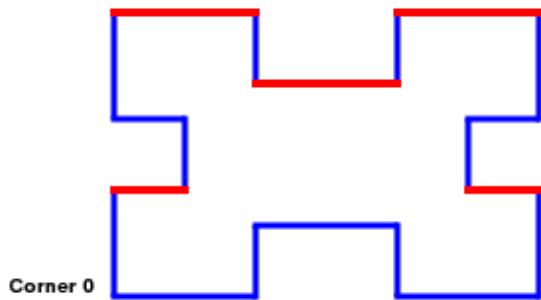
```
addPowerSwitch -ring -verticalSide 1
```



## Top Sides

The following command adds switches only to the top sides of the power domain: Sides 2, 6, 8, 10, and 14.

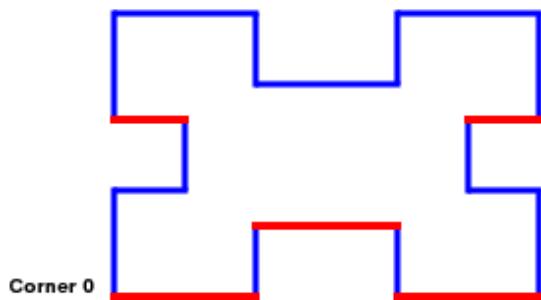
```
-addPowerSwitch -ring -topSide 1
```



## Bottom Sides

The following command adds switches only to the bottom sides of the power domain: Sides 4, 12, 16, 18, and 20.

```
-addPowerSwitch -ring -bottomSides 1
```



## Using Pitch Control and Offsets

You can control switch placement by using the following parameters:

- -globalOffset
- -bottomOffset
- -topOffset
- -leftOffset

- -rightOffset
- -horizontalOffset
- -verticalOffset
- -sideOffsetList { value ...}
- -startOffset
- -startOffsetBottom
- -startOffsetTop
- -startOffsetLeft
- -startOffsetRight
- -startOffsetHorizontal
- -startOffsetVertical
- -sideStartOffsetList { value ...}
- -endOffset
- -endOffsetBottom
- -endOffsetTop
- -endOffsetLeft
- -endOffsetRight
- -endOffsetHorizontal
- -endOffsetVertical
- -sideEndOffsetList { value ...}
- -forceOffset [0|1]
- -switchPitch
- -switchPitchBottom
- -switchPitchHorizontal
- -switchPitchLeft
- -switchPitchRight
- -switchPitchTop
- -switchPitchVertical
- -switchPitchSideList { value ...}

## Forcing Offsets

Offsets you specify are the *minimum* offsets you require to complete the power switch ring. Resulting offsets could be quite different from the ones you specify. To force the tool to comply as much as possible to the offset values, specify `-forceOffset 1`.

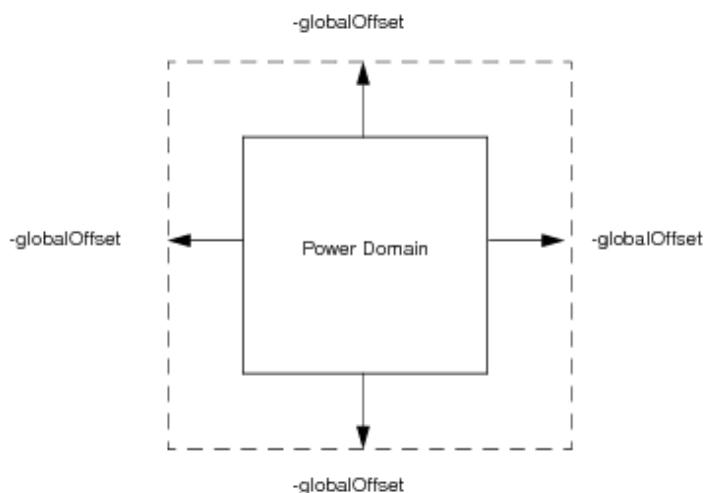
## Setting the Global Offset

Instead of placing switches against the power domain boundaries, you can place them away from the boundaries by the specified distances.

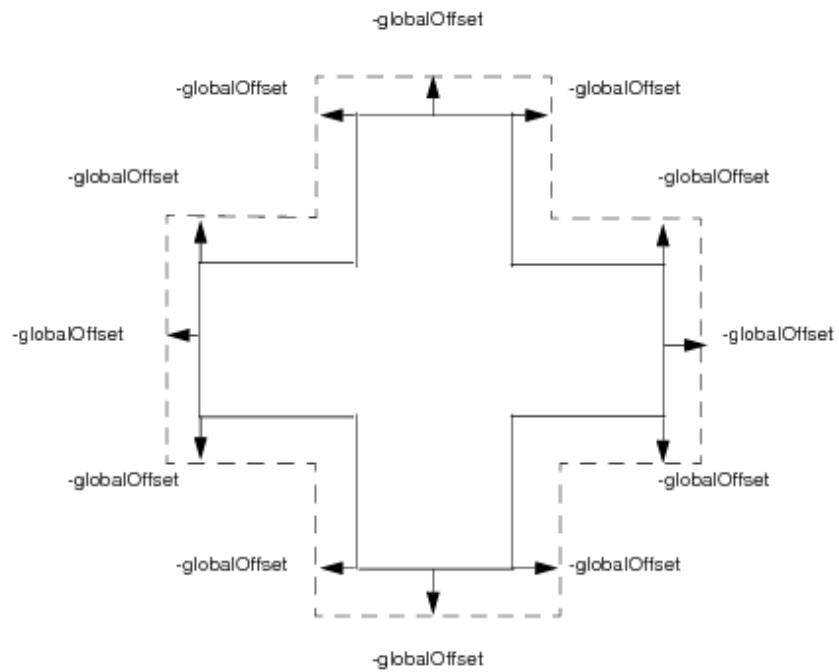
- To specify the same offset for all sides, use the `-globalOffset` parameter.

The default offset value is 0 (no offset).

The following figure shows equal offsets for a rectangular power domain if `-forceOffset 1` is specified:



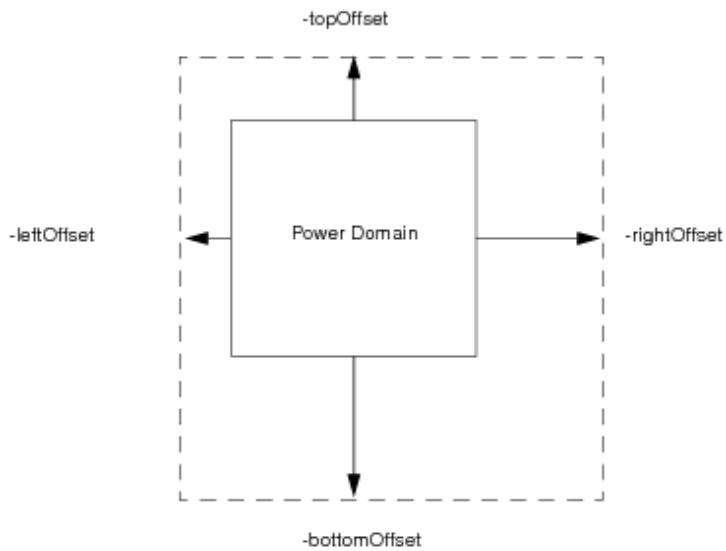
The following figure shows global offsets for a rectilinear power domain:



## Setting Different Offsets for Different Sides

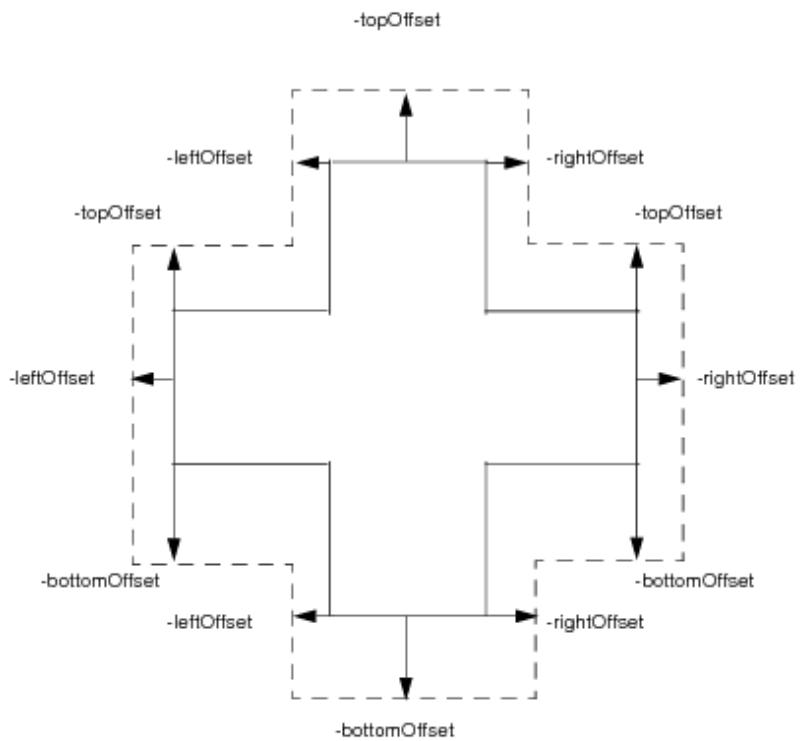
To specify different offsets for different sides, use the `-leftOffset`, `-rightOffset`, `-bottomOffset`, and/or `-topOffset` parameters.

The following example shows a different offset for each side if `-forceOffset` is specified:



- To specify offsets for the top and bottom, use the `-horizontalOffset` parameter.
- To specify offsets for the left and right sides, use the `-verticalOffset` parameter.

The following figure shows how `-leftOffset`, `-rightOffset`, `-bottomOffset`, `-topOffset` parameters affects rectilinear power domains:



- To specify offsets for the horizontal sides, use the `-horizontalOffset` parameter.
- To specify offsets for the vertical sides, use the `-verticalOffset` parameter.

## Specifying Switch Location

You can choose the location for placing the first and/or last cell in a power switch row, and the spacing between switches in a row. Use the following parameters:

- `-startOffset*`: Places the first cell instance on a side a specified distance from the nearest left, right, top bottom, vertical or horizontal offset (if specified) or the nearest power domain edge.
- `-endOffset*`: Places the last cell a specified distance from the nearest \*Offset (if specified) or the nearest power domain boundary at the end of the side.
- `-switchPitch*`: Specifies the distance from switch to switch (not the spacing between switches).

The following table shows the start, end, and pitch parameters:

| Start Offset           | End Offset           | Switch Pitch           | Applies to         |
|------------------------|----------------------|------------------------|--------------------|
| -sideStartOffsetList   | -sideEndOffsetList   | -switchPitchSideList   | Specified side(s)  |
| -startOffsetBottom     | -endOffsetBottom     | -switchPitchBottom     | Bottom side(s)     |
| -startOffsetTop        | -endOffsetTop        | -switchPitchTop        | Top side(s)        |
| -startOffsetRight      | -endOffsetRight      | -switchPitchRight      | Right side(s)      |
| -startOffsetLeft       | -endOffsetLeft       | -switchPitchLeft       | Left side(s)       |
| -startOffsetHorizontal | -endOffsetHorizontal | -switchPitchHorizontal | Horizontal side(s) |
| -startOffsetVertical   | -endOffsetVertical   | -switchPitchVertical   | Vertical side(s)   |
| -startOffset           | -endOffset           | -switchPitch           | All sides equally  |

You can omit offsets because the default values are 0. You can omit pitch because, by default, the cells abut. The software starts placing cells at corner 0.

- You can combine the following:
  - -startOffset with other -startOffset\* parameters
  - -endOffset with other -endOffset\* parameters
  - -switchPitch with other -switchPitch\* parameters

If there is more than one start or end offset, or switch pitch, on a side, the software always uses the most specific parameter for the side.

For example, if both `-startOffset` and `-startOffsetRight` are specified, the tool uses the `-startOffsetRight` value for the right side.
- You can combine the global offsets and pitch with side-specific offsets and pitch.
  - For example, for a rectangular power domain:  
`-startOffsetTop 1 -endOffsetLeft -2 -switchPitch 3`
  - For example, for a rectilinear power domain:  
`-startOffset -1 -switchPitchSideList {3, 4, 3, 0, 0, 0, 0, 0, 0}`
- You can combine any side-specific parameters.
  - For example, for a rectangular power domain:  
`-startOffsetLeft 1 -startOffsetRight 2 -endOffsetTop -2 -switchPitch 3`
  - For example, for a rectilinear power domain:  
`-startOffsetRight -1 -switchPitchSideList {3, 3, 3, 2, 2, 2, 1, 1, 3}`

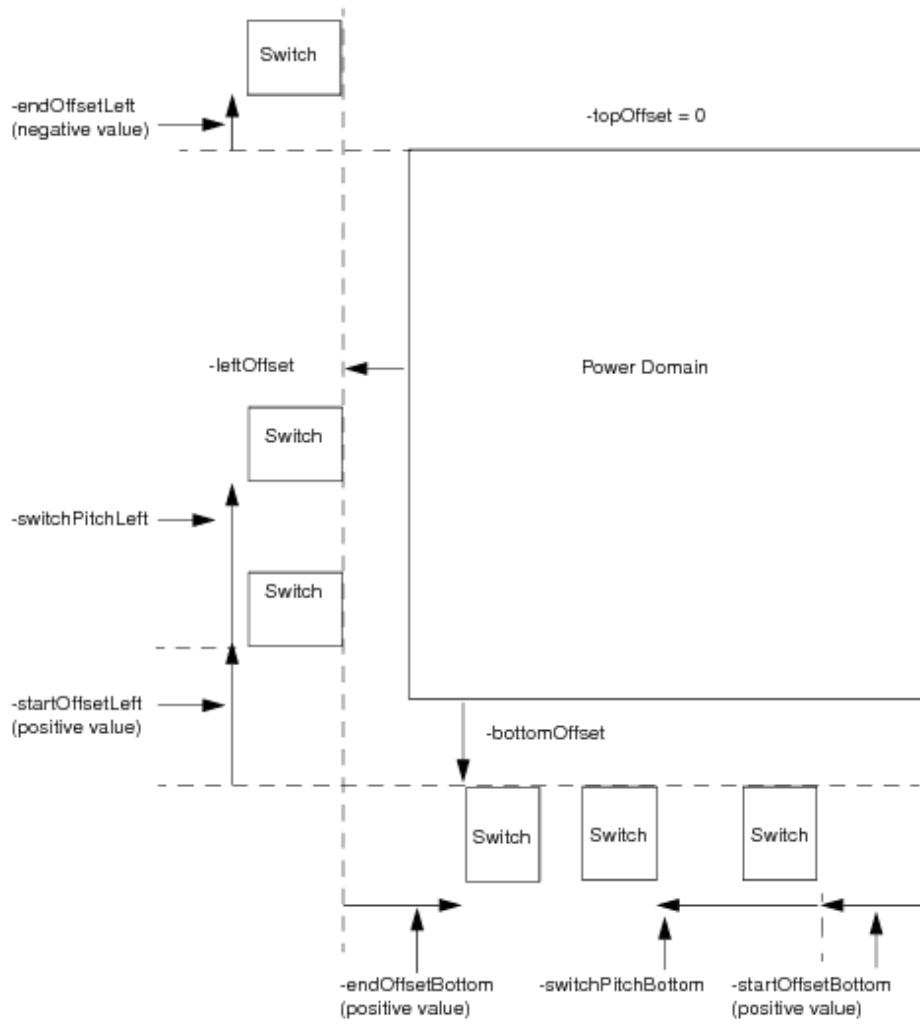
**Note:** All `-startOffset` and `-endOffset` values can be positive or negative, which affect cell placement toward or away from the center of the side. Pitch values can be positive only.

## Example of Offsets

The following example shows a clockwise switch ring with the following parameters:

- -leftOffset
- -bottomOffset
- -startOffsetLeft
- -startOffsetBottom
- -endOffsetLeft
- -endOffsetBottom
- -switchPitchLeft
- -switchPitchBottom

Different switch pitches are specified for the left and bottom sides, and the start and end offsets are positive or negative.



- The first switch on the left side is placed a distance `-startOffsetLeft` from the edge of the `-bottomOffset` boundary.
- The second switch on the left side is placed a distance `-switchPitchLeft` from the bottom edge of the first switch on the side.
- The last switch on the left side is placed a distance `-endOffsetLeft` *above* the `-topOffset` because the `-endOffsetLeft` value is negative. In this case, the `-topOffset` is 0, so the last switch is placed above the top power domain boundary.
- The first switch on the bottom side is placed a distance `-startOffsetBottom` from the right edge of the power domain. There is no `-rightOffset` or `-topOffset` specified.
- The second switch on the bottom side is placed a distance `-switchPitchBottom` from the right edge of the first switch on the side.

- The last switch on the bottom side is placed `-endOffsetBottom` to the *right* of the `-leftOffset` edge. The `-endOffsetBottom` parameter has a positive value.

## Power Switch Optimization

The `optPowerSwitch` command lets you perform power switch optimization in two ways:

- Optimize (reduce) the number of power switches in the ring and columns
- Perform ECOs to replace a filler cell with a switch or replace a switch with a bigger switch

You can use these two features in the following flow:

- Define the floorplan and power domains
- Synthesize the power grid
- Optimize power switches
- Place the design
- Run trialRoute
- Synthesize the clock tree
- Perform power switch ECO
- Perform buffer tree synthesis

## Power Switch Reduction

For power switch optimization (reduction), you can use the following options to `optPowerSwitch`:

- `-readPowerSwitchCell`
- `-commit`
- `-effort`
- `-maxIRDrop`
- `-maxSwitchIRDrop`
- `-net`
- `-padFile`
- `-readInstancePower`
- `-reportFile`
- `-setDontTouchCells`
- `-setDontTouchInstances`

- -totalPower

## Power Switch ECO

For power switch ECO, you can use the following options:

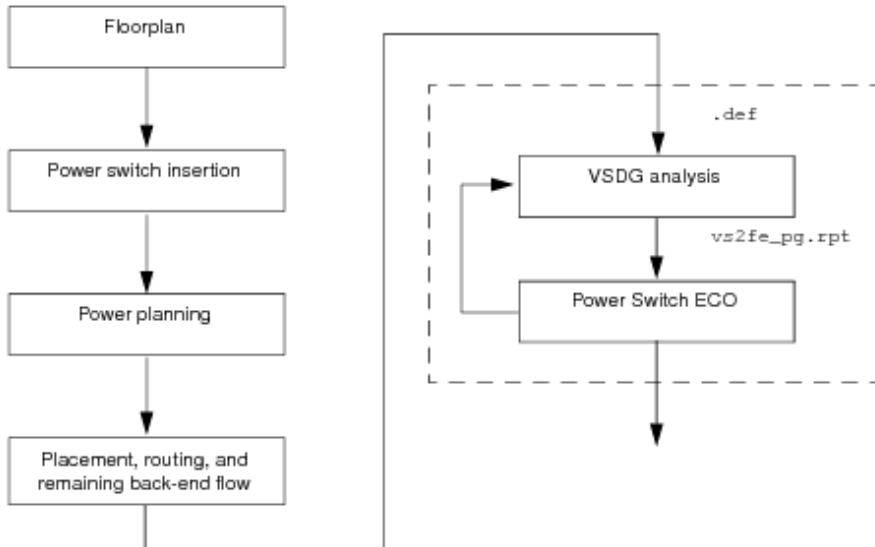
- -vsdgInFile
- -reportFile
- -fixViolations
- -reportViolationsOnly

You can now fix IR violations due to added power switches.

- Add power switches to your design.
- Create a complete power structure.
- Pre-characterize power gating cells in Libgen.
- Run Voltage Storm analysis to detect IR violations.
- Use the new `optPowerSwitch` command to repair these violations.

**Note:** In power switch optimization, the inserted power switches could overlap standard cells. After running `optPowerSwitch`, run `refinePlace` to move the standard cells away from the switches.

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



## Power Switch Prototyping

Innovus provides the facility for prototyping power switches. Power switch prototyping enables you to:

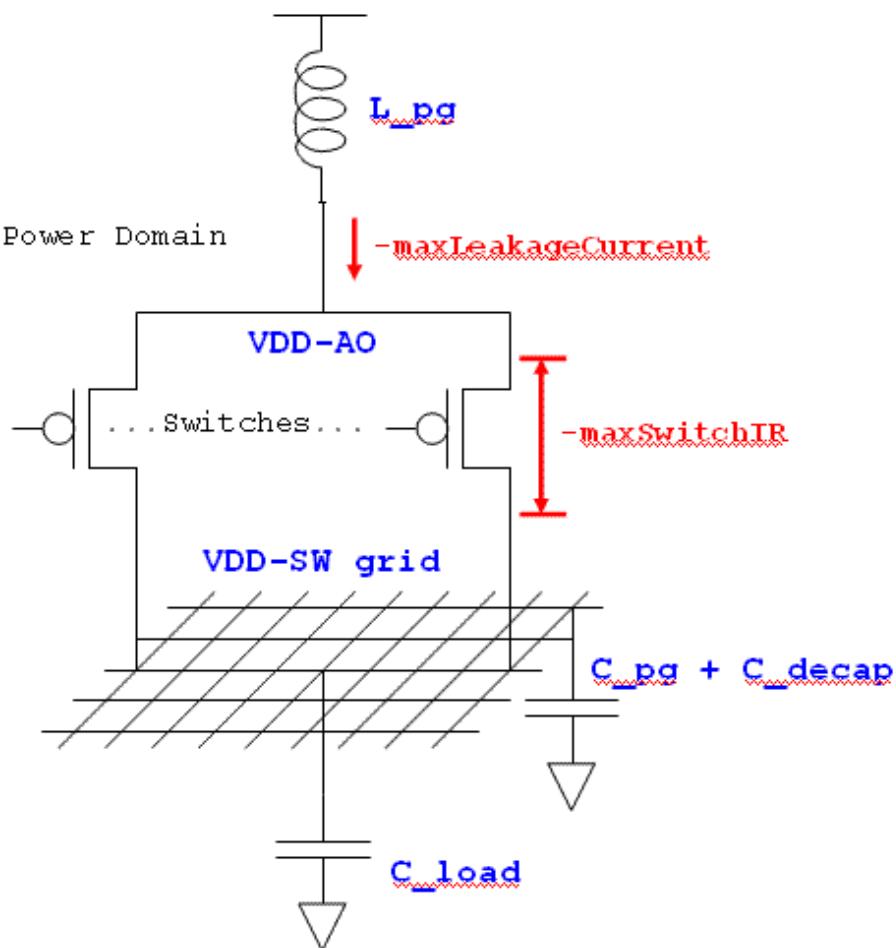
- Find optimal number of switches
- Sweep (Enumerate) number of switches
- Given the ramp up time, find optimal number of ramp up switches
- Given number of ramp up switches, find ramp up time
- Sweep (Enumerate) number of ramp up switches
- Find number of ramp up switches constrained by maximum ramp up current
- Find best switch delay constrained by maximum ramp up current

In this section we will cover:

- Power Domain Parameters and Specification
- Options Summary - Switch and Power Domain
- Options Summary - Prototyping Features
- Chain Style Impacts on Ramp Up Time and Rush Current
- Prototyping Results

## Power Domain Parameters and Specification

The following diagram and description outlines the power domain parameters and specifications:



### Attribute for the domain power:

- Total power: 1.0 mW
- Domain voltage: 1.0 V
- Max switch IR tolerance: 10 mV
- Max domain leakage current allowed: 2 uA
- Domain capacitance: 1 uF
- Package inductance: 0.1 nH

## Attributes for the switch cells:

- Idsat: 1 mA
- Ron: 800 Ohm
- Ileak: 10 nA
- Switch buffer delay: 100ps

## Options Summary - Switch and Power Domain

The following is summary of switch and power domains:

### Switch Cell Characterization

- Idsat
- Ileak
- Ron
- BufferDelay
- CellEM
- readPowerSwitchCell *filename*

### Power Domain Specification

- totalPower
- voltage
- maxSwitchIR
- maxLeakageCurrent
- loadCapacitance
- pgCapacitance
- pgInductance
- rampUpRailVoltagePercent (*0+..100-*)
- numberSimultaneousRampUpChain *num*

## Options Summary - Prototyping Features

The following is the summary of prototyping features:

- To find optimal number of switches enter:

```
-prototypeNumberSwitches 0 | 1
```

- To find Sweep/Enumerate number of switches enter:

```
-prototypeSweepSwitchNumber min max incremental
```

- To find Sweep/Enumerate number of ramp up switches enter:

```
-prototypeSweepChainDepth min max incremental
```

- To find min/max number of ramp up switches given ramp up time enter:

```
-prototypeChainDepth 0 | 1
```

```
-prototypeMinChainDepth 0 | 1
```

```
-prototypeMaxChainDepth 0 | 1
```

```
-rampUpTime min max
```

- To find ramp up time given number of ramp up switches enter:

```
-prototypeRampUpTime 0 | 1
```

```
-rampUpChainDepth num
```

- To find number of ramp up switches constrained by current maximum ramp up enter:

```
-prototypeChainDepthGivenRampUpCurrent 0 | 1
```

```
-maxRampUpCurrent float
```

- Find optimal switch delay constrained by current maximum ramp up:

```
-prototypeDelayGivenRampUpCurrent 0 | 1
```

```
-maxRampUpCurrent float
```

```
-rampUpChainDepth num
```

- To find miscellaneous enter:

```
-protoReportFile filename
```

```
-commitPrototype 0 | 1
```

```
-maxLeakagePercent percent
```

```
-maxIrPercent percent
```

## Chain Style Impacts on Ramp Up Time and Rush Current

The following is the impact of chain style power domains on ramp up time and rush current, per category:

### More simultaneous chain:

- Smaller resistance per time step
- Shorter ramp up time
- Larger rush current

### Longer Chain Depth:

- Longer time to turn on all switches
- Longer ramp up time
- Smaller rush current

### Ideal:

- Balance chain depth and simultaneous chain for optimal rush current control versus ramp up time

## Prototyping Results

The following results are covered in this section:

- Optimal Switch Results
- Switch Number Enumeration Results
- Ramp Up Switch Enumeration Results
- Number of Switches Given Current Maximum Ramp Up
- Switch Delay Given Current Maximum Ramp Up Current
- Ramp Up Time

## Optimal Switch Results

To find optimal switch results, enter:

```
addPowerSwitch -column -powerDomain PD -globalSwitchCellName HEADER \
-prototypeNumberSwitches 1
```

The following type of result is displayed:

```
# -----
# Results :
#
Recommended number of switches : 80
Min number of switches : 80
Max number of switches : 200
Switch area overhead : 3.25 %
Domain leakage current pre-PSO : 8.85e-06 A
Domain leakage current post-PSO : 8e-07 A
Domain percent leakage saving post-PSO : 91 %
Total switch current capacity : 0.08 A
Estimated switch IR drop : 0.01 V (1 %)
Peak ramp up current : 0.08 A
Number of columns : 4
Column left offset : 35.2 um
Column horizontal pitch : 79.3 um
# -----
```

## Switch Number Enumeration Results

To find the switch number results, enter:

```
addPowerSwitch -prototypeSweepSwitchNumber {100 200 10} {min max increment}
```

The following type of result is displayed:

```
# Results:
#
# Number Area      post-PSO Leakage Current Switch          PeakRampUp Num of   Pitch   Left
# Switch Overhead Leakage Saving Capacity IR             Current Columns   (um)    Offset
#       (%)        (A)      (%)     (A)      (V (%))      (A)           (um)      (um)
# -----
 100 :    4.06    1e-06    88.7     0.1    0.008( 0.80)    0.1      4    79.3    35.2
 110 :    4.47    1.1e-06   87.6     0.11   0.00727( 0.73)   0.11     4    79.3    35.2
 120 :    4.88    1.2e-06   86.4     0.12   0.00667( 0.67)   0.12     4    79.3    35.2
 130 :    5.28    1.3e-06   85.3     0.13   0.00615( 0.62)   0.13     4    79.3    35.2
 140 :    5.69    1.4e-06   84.2     0.14   0.00571( 0.57)   0.14     6    52.9     22
 150 :    6.10    1.5e-06   83.1     0.15   0.00533( 0.53)   0.15     6    52.9     22
 160 :    6.50    1.6e-06   81.9     0.16   0.005( 0.50)    0.16     6    52.9     22
 170 :    6.91    1.7e-06   80.8     0.17   0.00471( 0.47)   0.17     6    52.9     22
 180 :    7.31    1.8e-06   79.7     0.18   0.00444( 0.44)   0.18     6    52.9     22
 190 :    7.72    1.9e-06   78.5     0.19   0.00421( 0.42)   0.19     6    52.9     22
 200 :    8.13    2e-06    77.4     0.2    0.004( 0.40)    0.2      6    52.9     22
# -----
* Recommended number of switches.
```

## Ramp Up Switch Enumeration Results

To find ramp up switch enumeration results, enter:

```
addPowerSwitch -prototypeSweepChainDepth {100 200 10} {min max increment}
```

The following type of result is displayed:

```
# -----
# Results:
#
# Number   RampUp      PeakRampUp
# Switch    Time        Current
#          (s)         (A)
# -----
 100 : 9.01e-10      0.1
 110 : 7.92e-10      0.11
 120 : 6.84e-10      0.12
 130 : 6.78e-10      0.13
 140 : 5.72e-10      0.14
 150 : 5.67e-10      0.15
 160 : 5.63e-10      0.16
 170 : 4.59e-10      0.17
 180 : 4.56e-10      0.18
 190 : 4.53e-10      0.19
 200 : 4.5e-10       0.2
# -----
```

## Number of Switches Given Current Maximum Ramp Up

To find number of switches given the current maximum ramp us, enter:

```
addPowerSwitch -prototypeChainDepthGivenRampUpCurrent 1 -maxRampUpCurrent 0.088
```

The following type of result is displayed:

```
# -----
# 
# Results:
#
Number of ramp up switches : 128
Peak ramp up current : 0.0127 A
Ramp up time : 1.06e-08 s
Rate of ramp up current : 2e+05 A/s
# -----
```

## Switch Delay Given Current Maximum Ramp Up Current

To find switch delay given current maximum ramp up, enter:

```
addPowerSwitch -column -prototypeDelayGivenRampUpCurrent 1 -maxRampUpCurrent\ 0.088 -rampUpChainDepth 10
```

The following type of result is displayed:

```
# -----
# Prototype Ramp Up Time:
#
# Results:
# Results:
#
Switch stage-1 buffer delay : 1e-12 s
Peak ramp up current : 0.01 A
Ramp up time : 5.64e-08 s
Rate of ramp up current : 2e+07 A/s
#
# -----
```

## Ramp Up Time

To find ramp up time:

```
addPowerswitch -column -prototypeRampUpTime 1 -rampUpChainDepth 10
```

The following type of result is displayed:

```
# -----
# 
# Results:
#
Ramp up time : 5.64e-08 s
Peak ramp up current : 0.0002 A
# -----
```

# Placing the Design

- [Overview](#)
- [Loading a Design](#)
- [Preparing for Placement](#)
- [Guiding Placement With Blockages](#)
  - [Placement Treatment of Preroutes](#)
- [Adding Well-Tap Cells](#)
  - [Controlling the Distance Between Well-Tap Cells](#)
  - [Adding Well-Tap Cells to MSV Designs](#)
  - [Deleting Well-Tap Cells](#)
- [Adding End-Cap Cells](#)
  - [Adding End Cap Cells to MSV Designs](#)
  - [Adding Different Kinds of End Cap Cells](#)
  - [Deleting End-Cap Cells](#)
- [Placing Spare Cells and Spare Modules](#)
  - [Placing Spare Cells That Are Included in the Netlist](#)
  - [Placing Spare Cells That Are Not Included in the Netlist](#)
  - [Spare Cell Placement Behavior](#)
  - [Running Hierarchy-Aware Spare Cell Placement](#)
- [Adding Padding](#)
  - [Adding Instance or Module Padding](#)
  - [Adding Cell Padding](#)
- [Placing Standard Cells](#)
- [Running Placement in Multi-CPU Mode](#)
  - [Multi-Threading Placement Steps](#)
- [Checking Placement](#)
  - [Using the Amoeba View](#)
  - [Using the Density Map](#)
- [Adding Filler Cells](#)
  - [Adding Fillers to MSV Designs](#)
  - [Deleting Filler Cells](#)
- [Placing Gate Array Style Filler Cells for Post-Mask ECO](#)

- [Adding Decoupling Capacitance](#)
  - [Deleting Decoupling Capacitance](#)
- [Adding Logical Tie-Off Cells](#)
- [Saving Placement Data](#)
- [Specifying and Placing JTAG and Other Cells Close to the I/Os](#)
- [Optimizing and Reordering Scan Chains](#)
  - [Specifying Scan Cells](#)
  - [About Scan Chains](#)
  - [Reordering Scan Chains](#)

## Overview

After floorplanning, place the cells in the design. Placement considers the modules that were placed during floorplanning and takes into account the hierarchy and connectivity of the design. It honors floorplanning constraints, including guides, regions, and fences. For descriptions of the constraint types, see "Module Constraint Types" section in the [Floorplanning the Design](#) chapter of the *Innovus User Guide*.

Placement also follows legalization rules, such as cells cannot overlap each other and takes into account short and spacing DRC rules required by routing.

After the cells are legally placed, next step is preCTS optimization.

## Loading a Design

Load a design by using the `restoreDesign` command or the `init_design` command

For more information, see

- Importing and Exporting Designs chapter of the *Innovus Text Command Reference*.
  - `restoreDesign`
  - `init_design`

## Preparing for Placement

Before placement, run the following commands and correct problems. Some of these commands generate reports that you can use as a baseline for comparisons later in the flow.

- Run `checkDesign` to check the integrity of the library and design data. For more information, see `checkDesign` in the "Import and Export Commands" chapter of the *Innovus Text Command Reference*.
- Run `timeDesign -prePlace` to get an idea of Zero Wire Load timing of the design. For more information, see `timeDesign` in the "Timing Analysis (Common Timing Engine) Commands" chapter of the *Innovus Text Command Reference*.
- Run `createPlaceBlockage` to create blockages (this is usually done during floorplanning). For more information see "Guiding Placement With Blockages" or `createPlaceBlockage` in the "Floorplan Commands" chapter of the *Innovus Text Command Reference*.
- Use one of the following methods to place and fix hard blocks.
  - Run `planDesign`. For information, see `planDesign` in the "Floorplan Commands" chapter of the *Innovus Text Command Reference*.
    - Manually place and fix hard blocks.
  - Run `checkPlace` (or `checkDesign -place`) or use the Violation Browser to check for violations caused by preplaced cells or blocks. For more information, see `checkPlace` in the "Placement Commands" chapter of the Innovus Text Command Reference.

For more information on preparing the design for placement, see [Data Preparation](#).

## Guiding Placement With Blockages

Use placement blockages to help guide placement.

Create the blockages during the floorplanning session by using the following command:

`createPlaceBlockage`

After creating a blockage, assign an attribute to it by using the Attribute Editor.

Alternatively, you can create placement blockages using the *Set Placement Blockage Options* form.

For more information, see *Create Placement Blockage* in the "Floorplan Menu" chapter of the *Innovus Menu Reference*.

A placement blockage has one of the following attributes:

|             |                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>Hard</b> | The area cannot be used to place blocks or cells. By default, <code>createPlaceBlockage</code> creates blockages with this attribute. |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------|

|                   |                                                                                                                                                                                                                                                                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Soft</i>       | The area cannot be used to place blocks or cells during placement but can be used during in-place optimization, clock tree synthesis, ECO placement, or placement legalization ( <code>refinePlace</code> ).                                                                                                                              |
| <i>Partial</i>    | <p>Sets a percentage of the area that is available for placement.</p> <p>For example, a partial placement percentage of 75 percent means that up to 75 percent of placement density is allowed in the area.</p> <p><b>Note:</b> A partial blockage of 100 percent (or 0 percent placement density screen) behaves as a soft blockage.</p> |
| <i>Macro-Only</i> | The area cannot be used to place blocks but can be used to place standard cells as a free area.                                                                                                                                                                                                                                           |

- ➊ If the design has routing congestion issues in the small channels between hard blocks, consider using the `setPlaceMode -autoBlockageInChannel true` which automatically adds soft placement blockages in these areas. Although the blockages obstruct standard cells during placement step, they do not obstruct standard cells during optimization operations. Using soft blockages can help improve both timing and routability.

For more information, see

- `createPlaceBlockage` in the "Floorplan Commands" chapter of the *Innovus Text Command Reference*.

## Placement Treatment of Preroutes

Placement treats preroutes the same way it treats routing blockages: It places standard cell instances at legal locations where there should not be any DRC violations against preroutes or routing blockages.

Typically, you use preroutes for special nets that are floorplanned (pre-designed) before placement, such as power, ground, and clock mesh nets, where you do not want any standard cells placed underneath. Instances placed next to power and ground stripes honor the design spacing rule. Instances placed next to routing blockage objects are set adjoined.

By default, the Innovus placement honors preroutes and routing blockages on *metal2* for a three-metal layer process and on *metal2* and *metal3* for a four or more metal layer process.

You can change this behavior by using the following command before running placement:

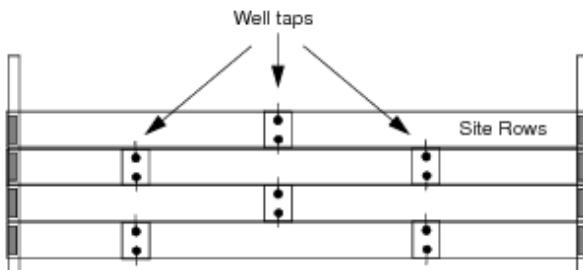
```
setPlaceMode -prerouteAsObs
```

For more information, see `setPlaceMode` in the "Placement Commands" chapter of the *Innovus Text Command Reference*.

## Adding Well-Tap Cells

Well taps are physical-only filler cells that are required by some technology libraries to limit resistance between power or ground connections to wells of the substrate. Well-tap cells are placed in a preplaced status, so future placement commands do not move them.

The following diagram shows an example of well-tap cell placement. In this diagram, the cells are staggered in the site rows.



Add well taps after the floorplan is fixed and hard blocks are placed, but before placing standard cells.

Use one of the following methods to add well-tap cells:

- Add Well Tap Instances form
- `addWellTap` command

## Controlling the Distance Between Well-Tap Cells

Use the `addWellTap -cellInterval` parameter to specify the maximum distance between well-tap cells in the same row.

- `-cellInterval` measures the distance from the center of one well-tap cell to the center of the next well-tap cell in the same row.

By default, the software always leaves a distance that is at least 45 percent of the specified maximum distance between well-tap cells in the same row. For example, if the specified maximum distance between same-row well-tap cells is 48.0 microns, the default minimum distance would be 21.6 microns.

## Adding Well-Tap Cells to MSV Designs

In cases where there are different voltages in the same design, also known as a multi-voltage (MSV) design, specify the power domain in which to insert the well-tap cells by using the following command:

```
addWellTap -powerDomain
```

## Deleting Well-Tap Cells

To remove added well-tap cells, use the Delete Instances form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes only well-tap cells that are completely contained within the area; it does not delete well-tap cells that cross the area boundary.

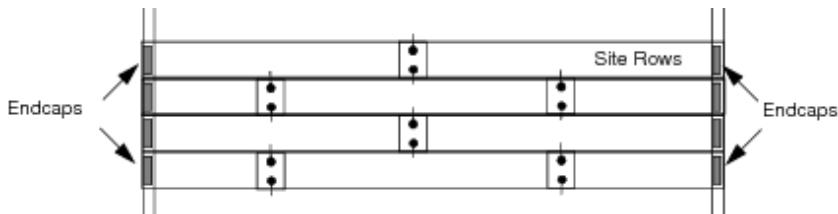
For more information see the following topics:

- [addWellTap](#) and [deleteFiller](#) in the "Placement Commands" chapter of the *Innovus Text Command Reference*.

## Adding End-Cap Cells

End-cap cells are preplaced physical-only cells that are required to meet certain design rules. They are placed at the ends of the site rows, and are used in some technologies for power distribution. Besides of inserting the end-cap cells at the row ends, `addEndCap` also supports to place the end-cap cells at the top and bottom row to constitute an enclosed end-cap ring around the core area, placement blockage and hard macros. End-cap cells are placed in a preplaced status, so future placement commands do not move them. Add end-cap cells to the design before any other standard cells are placed, but after hard blocks are placed in the floorplan.

The following diagram shows an example of end-cap cell placement. The cells are placed at the ends of each site row.



To add end-cap cells, use the Add End Cap Instances form or the `addEndCap` command.

## Adding End Cap Cells to MSV Designs

In cases where there are different voltages in the same design, specify the power domain in which to insert the end cap cells by using the following command:

```
addEndCap -powerDomain
```

## Adding Different Kinds of End Cap Cells

You can add different kinds of end cap as per the site numbers between core or placement blockage or hard macro using the `setEndCapMode` command. There are two kinds of single-height end cap cells for left/right boundaries, for example, A type and B type. If the site number is odd, the end cap type on both sides of the core will be different. This means AB or BA end caps are placed on left/right boundaries. If the site number is even, the end cap type on both sides should be the same. This means AA or BB end caps are placed on left/right boundaries.

In the inner corner of core or blockage, double-height end cap cell is required. The double-height cells are also placed the same way as single-height cells. For A type or B type double-height cell, two kinds of power rail should be defined: VSS-VDD-VSS, VDD-VSS-VDD. So there are 4 types of double-height cell: A(VSS-VDD-VSS), A(VDD-VSS-VDD), B(VSS-VDD-VSS), B(VDD-VSS-VDD). Accordingly, two kinds of double-height sites should be defined: VSS-VDD-VSS and VDD-VSS-VDD. All those end cap cells and sites must be pre-defined in LEF.

Before end cap insertion, you need specify all the end cap cells for each boundary in `setEndCapMode` to constitute an enclosed end cap ring around the core area, placement blockage, and hard macros. For this, you use the following options of `setEndCapMode`.

- -leftBottomCornerEven
- -leftBottomCornerOdd
- -leftBottomEdgeEven
- -leftBottomEdgeOdd
- -leftEdgeEven
- -leftEdgeOdd
- -leftTopCornerEven
- -leftTopCornerOdd
- -leftTopEdgeEven

- -leftTopEdgeOdd
- -rightBottomCornerEven
- -rightBottomCornerOdd
- -rightBottomEdgeEven
- -rightBottomEdgeOdd
- -rightEdgeEven
- -rightEdgeOdd
- -rightTopCornerEven
- -rightTopCornerOdd
- -rightTopEdgeEven
- -rightTopEdgeOdd

The `addEndCap` command will honor these mode settings. Therefore, it is important to specify the correct end cap cell type, especially for the double-height cells.

## Deleting End-Cap Cells

To remove end-cap cells, use the Delete Instances form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes end-cap cells that are completely contained within the area; it does not delete end-cap cells that cross the area boundary.

For more information see

- `addEndCap` and `deleteFiller` in the "Placement Commands" chapter of the *Innovus Text Command Reference*

# Placing Spare Cells and Spare Modules

## Placing Spare Cells That Are Included in the Netlist

If spare cell instances are included in the gate-level netlist, the software places them during preplacement processing; however, you must specify them during the floorplanning session.

-  Cadence recommends that you place clusters of spare cells at different locations within the core area to allow easy access to the cells from different parts of the core.

1. Specify the spare cells by using the following command:

```
specifySpareGate
```

Use the following parameter to identify the module whose hierarchy contains the spare cell instances:

```
-hinst
```

2. If the design contains hierarchical modules, specify the following `setPlaceMode` parameter to ensure that the spare cells within the modules are kept within bounds of the hierarchy, even if no constraint is set on it:

```
-moduleAwareSpare
```

For more information on using this parameter, see "Running Hierarchy-Aware Spare Cell Placement".

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

## Related Topics

For more information, see the following commands in the "Placement Commands" chapter of the *Innovus Text Command Reference*:

- [setPlaceMode](#)

- [specifySpareGate](#)

## Placing Spare Cells That Are Not Included in the Netlist

If the netlist does not include spare cell instances, you must create a spare module and place it before you place the standard cells.

1. Use the following command to create a spare module:

```
createSpareModule
```

2. Use the following command to place the module:

```
placeSpareModule
```

To delete a spare module, use the following command:

```
deleteSpareModule
```

For more information, see the following commands in the "Placement Commands" chapter of the *Innovus Text Command Reference*:

- [createSpareModule](#)
- [placeSpareModule](#)
- [deleteSpareModule](#)

## Spare Cell Placement Behavior

- If there are no floorplanning constraints, or if the design has not been floorplanned, the software places spare cell instances randomly in the core area.
- If a spare cell instance is contained in a fence or a region, the software places the instance randomly in the fence or region that includes the instance.
- If spare cell instances in the netlist are grouped into modules, the software places the modules in a grid fashion in the core area.

**Note:** For information on controlling spare cell placement when hierarchy is an issue, see "Running Hierarchy-Aware Spare Cell Placement".

Spare cell distribution is dependent upon the way spare cells are connected.

- If the spare cells are floating (that is, if they are not connected) or they are connected to power or

ground, they are evenly distributed in the placement area.

- If the spare cells have connections to other spare cells, they are treated as a spare cell group and are placed close to one another in the placement area.
- If the spare cells, or a group of spare cells, have a connection to a non-spare cell instance, they are placed close to that instance.

To instruct the software to disregard spare cell connections and distribute the cells evenly in the placement area, complete one of the following steps before running placement:

- Specify the following command:

```
setPlaceMode -ignoreSpare true
```

- Select the *Ignore Spare Cell Connections* option on the *Placement* page of the Design - Mode Setup form.

For more information, see [setPlaceMode](#) in the "Placement Commands" chapter of the *Innovus Text Command Reference*.

## Running Hierarchy-Aware Spare Cell Placement

To control placement of spare cells or modules in the netlist when hierarchy is an issue, use the following commands:

- `specifySpareGate { -hinst | -inst}`
- `setPlaceMode -moduleAwareSpare {true | false}`

The following examples and figures show how these commands affect placement.

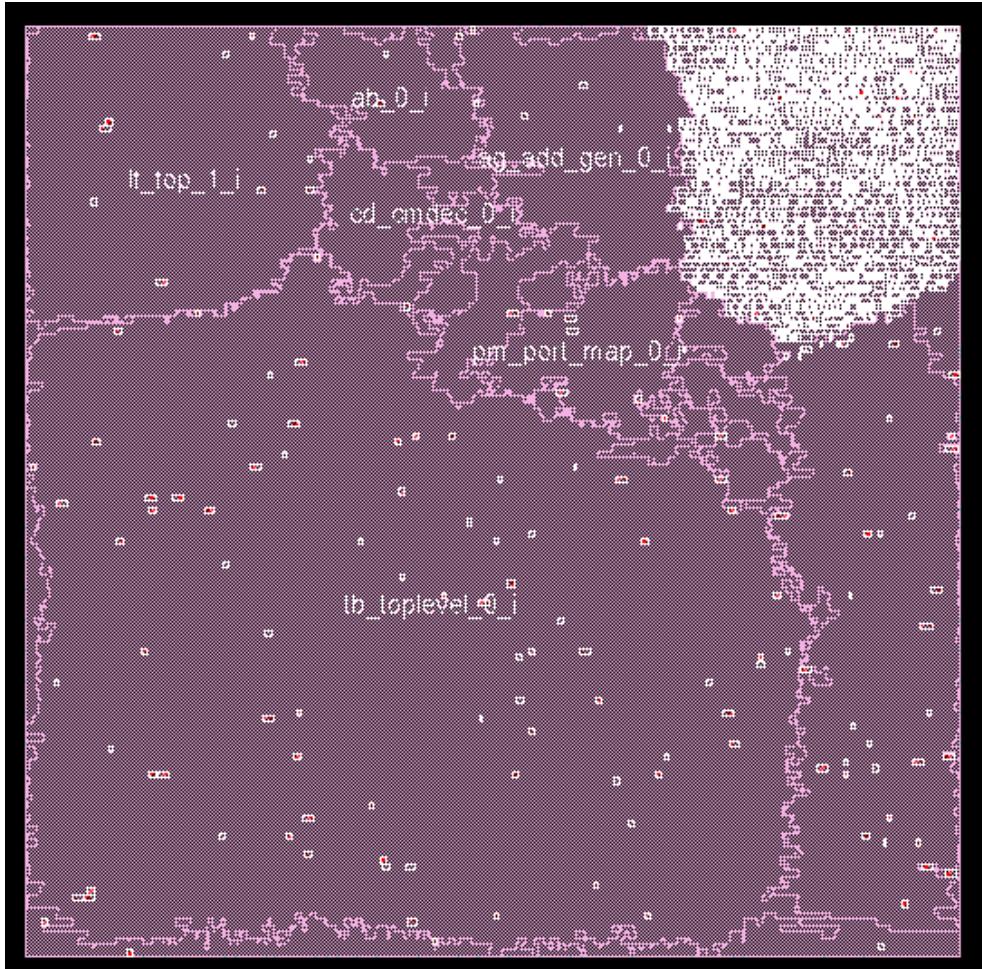
```
specifySpareGate -inst blk1/spare_1/*
# Works also for specifySpareGate -hinst
setPlaceMode -moduleAwareSpare true

placeDesign
```

To place the spare cells evenly in the core, without binding them to the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x <6} {incr x 1} {
    specifySpareGate -inst lt_top_0_i/spare_${x}i/*
}
```

```
placeDesign
```

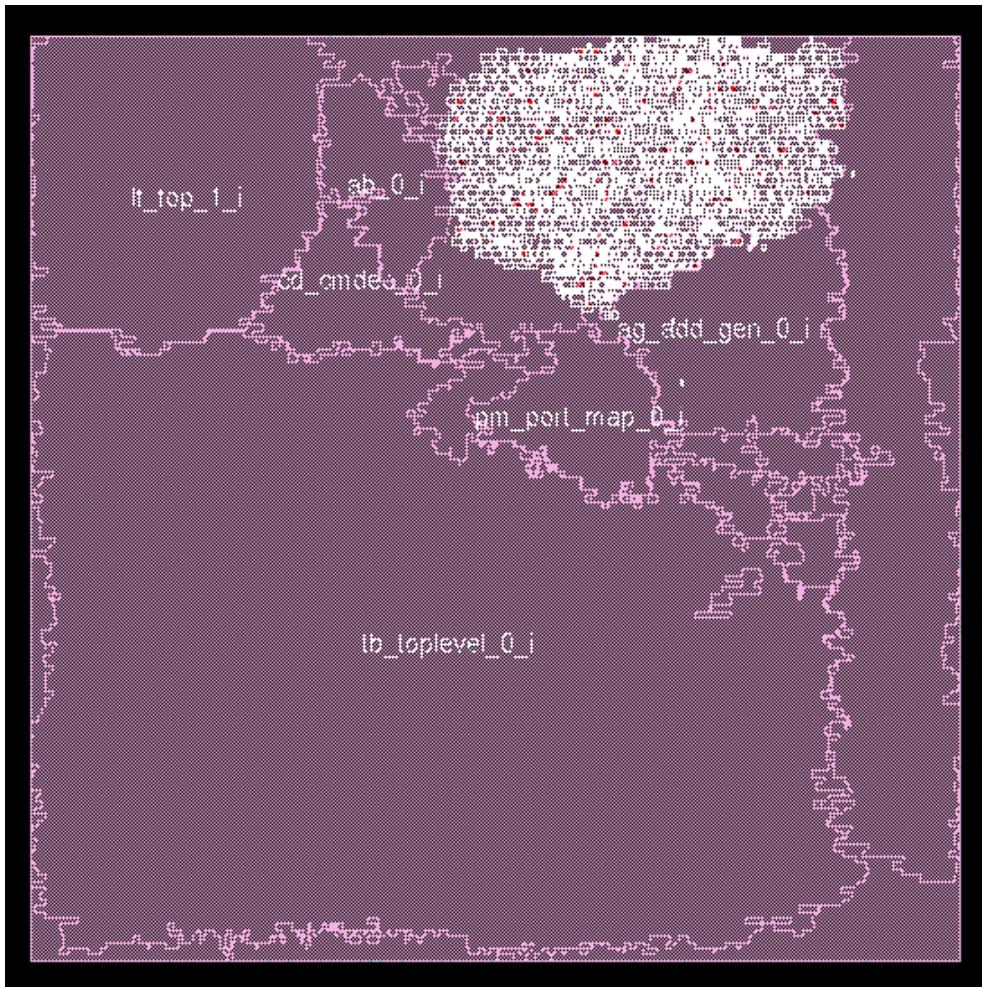


The log file, before Iteration 1, contains the following information: Identified 240 spare or floating instances, with no clusters.

To place the spare cells within the bounds of the `lt_top_0_i` hierarchy, using the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {
    specifySpareGate -inst lt_top_0_i/spare_${x}i/*
}

setPlaceMode -moduleAwareSpare true
placeDesign
```

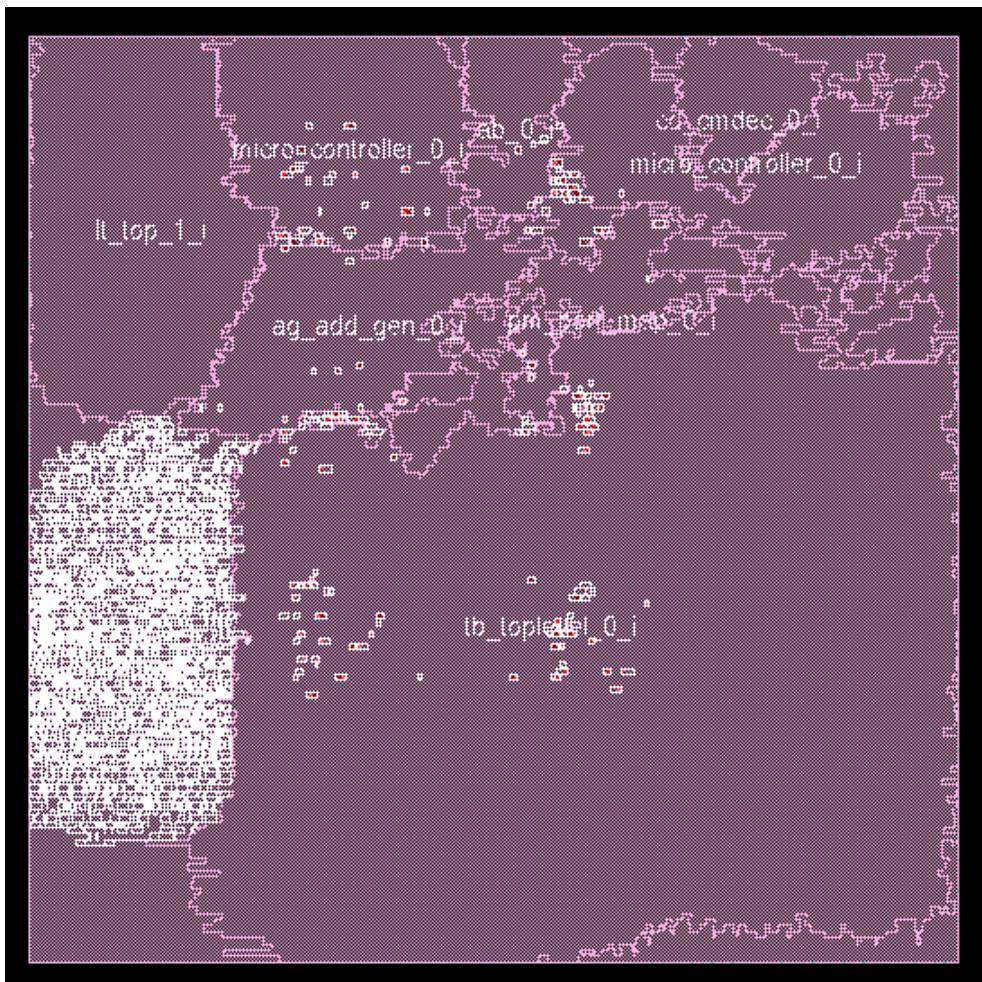


The log file, before Iteration 1, contains the following information: Identified 240 spares within logical modules.

To spread out the spare cells evenly as six clusters in the core, without binding them to the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {  
    specifySpareGate -hinst lt_top_0_i/spare_${x}i/*  
}
```

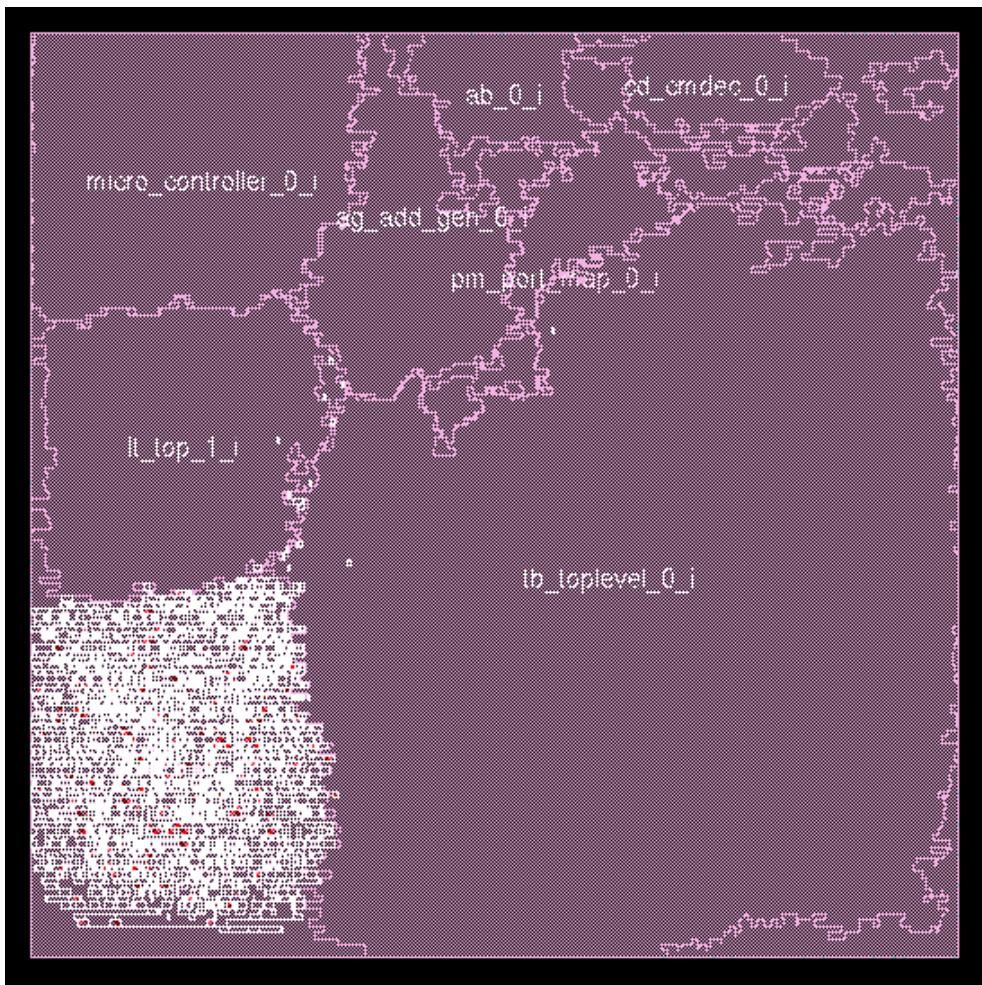
placeDesign



The log file, before Iteration 1, contains the following information: Identified 240 spares or floating instances, where some are grouped into 6 clusters.

To place the spare cells within the bounds of the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {  
    specifySpareGate -hinst lt_top_0_i/spare_${x}i/*  
}  
  
setPlaceMode -moduleAwareSpare true  
setPlaceMode -modulPlan false  
placeDesign
```



The log file, before Iteration 1, contains the following information: Identified 240 spares within logical modules, of which 240 are in 6 spare-only modules.

## Adding Padding

Add padding to reserve placement space for cells or routing added after placement, for example, to make sure there is room to insert clock buffers when running Clock Tree Synthesis (CTS) on a highly localized clock. The software adds the padding on the right side of placed instances at a default *metal2* pitch dimension.

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

- ✓ If the clock in your design is concentrated in a tight area, reserve three to five percent of the targeted final utilization to add clock buffers. If your initial settings do not provide sufficient space for the buffers, add more padding and rerun placement after CTS or placement optimization.

## Adding Instance or Module Padding

Instance padding and module padding reserve space during global placement so it can be used later in the design flow, for cells added during placement legalization, Clock Tree Synthesis (CTS), or timing optimization.

Note: Cell padding is ignored by refinePlace if instances are fixed. Use setPlaceMode [-padFixedInsts {true | false}] if cell padding needs to be honored for fixed instances. For more information, see [setPlaceMode](#).

## Adding Instance Padding

Instance padding is specified in terms of the number of sites occupied by each instance. For example, if a row fits 30 single-site instances without padding, you can specify padding of two sites for each instance in the row. In this case, each instance in the row will then occupy three sites, and the row will fit only 10 instances.

1. Specify the following command:

```
specifyInstPad
```

2. (Optional) Report instance padding by using the following command:

```
reportInstPad
```

To delete instance padding, use the following command:

```
deleteInstPad
```

For more information, see the following commands in the "Placement Commands" chapter of the *Innovus User Guide*:

- [specifyInstPad](#)
- [reportInstPad](#)
- [deleteInstPad](#)

## Adding Module Padding

To reduce localized congestion, add module padding.

1. Specify the following command:

```
setPlaceMode -modulePadding module factor
```

This command adds padding within hierarchical modules by spreading out the standard cell instances within the modules. The padding is specified in terms of a factor that is applied to the instance area of all the cells within the module. For example, a factor of 1.2 increases the area by 20 percent.

**Note:** The software ignores factors that are less than 1.0.

## 2. Run standard cell placement.

For more information, see `setPlaceMode` in the "Placement Commands" chapter of the *Innovus User Guide*.

## Adding Cell Padding

Cell padding adds hard constraints to placement. The constraints are honored by cell legalization, CTS, and timing optimization, unless the padding is reset after placement so those operations can use the reserved space. You can use cell padding to reserve space for routing.

- Specify the following command:

```
specifyCellPad
```

This command adds padding on the right side of library cells during placement. (Padding location is dependent on the orientation of the cell. For example, if the library cell is flipped when it is instantiated, the padding is on the left side.) The padding is specified in unit of placement SITE. For example, if you specify a value 2, the software ensures that there is additional clearance of two placement SITES on the right side of the specified cells.

-  To add padding to a cell if any signal pin is near enough to the border to cause DRC violations with any metal geometry, run the following command before placing standard cells:

```
setPlaceMode -padForPinNearBorder true
```

To delete cell padding, use the following command:

```
deleteAllCellPad
```

For more information, see the following commands in the "Placement Commands" chapter of

the *Innovus User Guide*:

- [specifyCellPad](#)
- [deleteAllCellPad](#)

## Placing Standard Cells

Place standard cells with the `place_opt_design` command. By default, the command runs preplacement optimization and standard cell placement. It also assigns or reassigns IO pins that are not in preplaced status and executes preCTS flow with both placement and preCTS optimization.

If you specified SDC timing constraints, it runs in timing-driven mode by default. If you specified scan information, it performs scan tracing and reordering by default.

**i** If `place_opt_design` does not place the standard cells, for example if all instances are fixed or if there is no placeable area, then `place_opt_design` also skips I/O pin assignment.

This command was designed as a super command; that is, with some exceptions, you can use it to place standard cells without specifying any placement options for your initial placement.

- To reduce the switching power on power-critical nets, consider running the following command before placing standard cells:

```
setPlaceMode -powerDriven true
```

Tune the initial placement by trying the following techniques:

- If the design is congested, try the following command, then rerun placement:

```
setPlaceMode -congEffort high
```

- If the design has local congestion, try the following command, then rerun placement:

```
setPlaceMode -modulePadding module factor
```

## Related Topics

- `placeDesign` and `setPlaceMode` in the "Place Commands" chapter of the *Innovus Text Command Reference*

- [specifyClockTree](#) in the "Clock Tree Synthesis Commands" chapter of the *Innovus Text Command Reference*

## Running Placement in Multi-CPU Mode

The `place_opt_design` command and the `addFiller` command supports multi-threading. Multi-threading accelerates placement by splitting a job into two or more tasks that run concurrently on a single machine that has multiple processors. The placement acceleration is not linear, however, because some set-up and synchronization time is required.

Multi-threading requires additional licenses. The number of additional licenses required is dependent on the "base" Innovus Digital Implementation System license (the base license is the license used to invoke the software) and the number of threads you want to use.

Multi-threading placement has the following limitations:

- It is supported by global placement only, not by placement legalization.
- The maximum number of threads you can use is eight. If you request more, the software issues a warning and sets the number of threads to eight.

**i** To get the greatest benefit from multi-threading, your placement job should be the only job running on your machine--you should avoid all other tasks, even a regular system backup. For example, a machine with four CPUs that is running backup or other system tasks that occupy one CPU might show less speed-up with four threads than a machine running no system tasks that is running global placement with three threads.

## Multi-Threading Placement Steps

To run multi-threading placement, complete the following steps. You can complete these steps before running any commands that run multiple-CPU processing, or before running placement. Because the Innovus software has a common interface for multiple-CPU processing (multi-threading or distributed processing), you need specify these commands only once per session, and any application that can run in multiple-CPU processing mode can use the additional licenses and processors.

1. (optional) Use the following command to specify the number of multiple-CPU licenses to check out and the license check-out order:

```
setMultiCpuUsage [-acquireLicense integer] [-localCpu {integer | max}] \
[-licenseList licenses]
```

If you do not use this command, the software runs it automatically, using a default check-out order and requesting the appropriate number of licenses based on the parameters you set for `setMultiCpuUsage`.

2. Use the following command to specify the maximum number of threads to use:

```
setMultiCpuUsage -localCpu {integer | max}
```

If you request more threads than are available, the software uses the maximum number that are available.

**Note:** It is generally not a good idea to request more threads than the number of CPUs in your machine, as it will slow down the machine and waste licenses.

3. (optional) Use the following command to release the additional licenses after global placement:

```
setMultiCpuUsage -keepLicense true
```

 Alternatively, run the following command after placement to release the additional licenses immediately:

```
setMultiCpuUsage -releaseLicense
```

4. Run global placement.

The log file reports the number of threads used for multi-threading placement just before it shows the global placement iterations, for example:

```
Placement running 2 threads
```

5. (optional) Check the run-time information at the end of the `place_opt_design` section of the log file.

```
(cpu for global=1:25:08) real=0:50:32***  
Placement multithread real runtime: 0:50:32 with 2 threads.
```

```
Core Placement runtime cpu: 1:14:24 real: 0:41:42
Starting refinePlace ...
```

The number to look for in the log is `Placement multithread real runtime`. In the preceding example, the `Placement multithread real runtime` is `0:50:32`.

## Calculating Multi-Thread Speed-Up

The amount of time used for global placement is calculated by using the following formula:

$$\text{Global Placement} = \text{Core Placement} + \text{Timing Analysis} + \text{Congestion Analysis}$$

In some designs, when timing and congestion analysis consume a high percentage of run time, the speed-up factor from multi-threading is not significant.

In the preceding example, if only one thread were used, the log would have reported the following:

```
(cpu for global=1:13:53) real=1:15:09***
Core Placement runtime cpu: 1:04:53 real: 1:05:59
Starting refinePlace ...
```

Comparing the times from the two log segments gives the following calculations:

- `real time (1 thread)` = `1:15:09` = 4509 seconds
- `real time (2 threads)` = `0:50:32` = 3032 seconds

$$4509 / 3032 = 1.49$$

**i** If other jobs are running during multi-threading placement, the `real` time includes the run time for those jobs, so you do not get an accurate speed-up comparison.

## Related Topics

- [Accelerating the Design Process By Using Multiple-CPU Processing](#)
- [Multiple-CPU Processing Commands](#) chapter in the *Innovus Text Command Reference*.

## Checking Placement

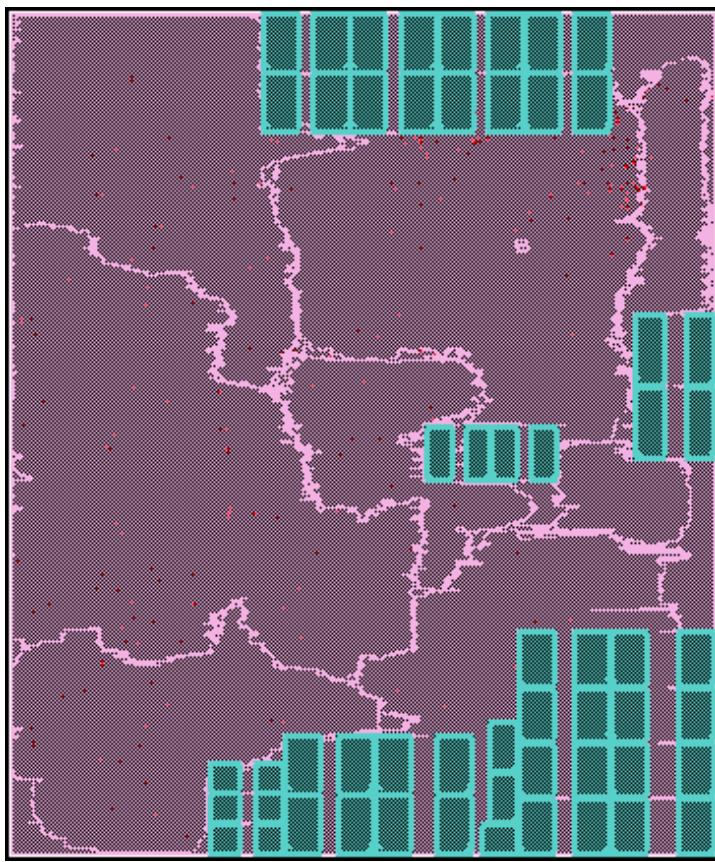
Use the following methods to check placement:

- Amoeba view  
For more information, see [The Main Window](#) chapter in the *Innovus Menu Reference*.
- checkPlace command
- For more information, see [checkPlace](#) in the "Placement Commands" chapter of the *Innovus Text Command Reference*.
- Placement density map  
For information, see the following references:
  - "Placement Commands" chapter of the *Innovus Text Command Reference*
    - [getDensityMapMode](#)
    - [reportDensityMap](#)
    - [setDensityMapMode](#)
  - "Placement Menu" chapter of the *Innovus Menu Reference*
    - Display Density Map
- Violation Browser

**Note:** The Violation Browser does not indicate the layer on which a placement violation occurs.

## Using the Amoeba View

Use the Amoeba view to see the placement of modules and blocks. For example, in the following figure you can see the outlines of the hard blocks and the modules, and that the instances in each of the modules are placed closely together.



To display the Amoeba view, select the *Amoeba view* widget from the *Views* panel in the main Innovus window.

For more information, see [The Main Window](#) chapter in the *Innovus Menu Reference*.

## Using the Density Map

Use one of the following methods to turn the display of the density map on or off:

- Select *Density Map* on the list of Visibility toggles in the main Innovus window.
- Click the All Colors button to open the Color Preferences form, then select the *View Only* tab, and select *Density Map*, in the *Multi-Color Layers* section.

## Adding Filler Cells

The software uses filler cells to fill the gaps between standard cell instances. Filler cells also provide decoupling capacitance to complete the power connections in the standard cell rows and extend N-well and P-well regions. The reason to add them as the last placement step is that you cannot run in-place optimization after they are added. If filler cells are added after routing, the software checks for DRC violations between regular nets and the filler cells.

To add filler cells, use the [addFiller](#) command.

- ❶ Provide a list of filler cells so that at least one filler cell can be used to fill the space without causing DRC violations.

Add Filler recognizes whether a filler cell has implant layer geometries and attempts to add fillers that honor the implant layers' width, spacing and minimum area rules. By judicious selection of filler cells, the software can correct implant layers' minimum spacing errors by putting in same voltage threshold implant layer fillers in spaces between two same implant layer cells. Add Filler also avoids creating implant layer minimum width and minimum area errors by abutting fillers of same implant layer as the adjacent cells, thus extending the implant layer width.

- ❷ Add Filler expects to be provided with cells of all types of implant layers to be able to completely fill the design's core area with fillers. For example, if only a low-voltage implant layer filler is provided, and the abutting logical cell has a high-voltage implant layer, then Add Filler places the provided low-voltage implant filler only if its width satisfies the minimum width rule for that implant layer.

## Adding Fillers to MSV Designs

In cases where there are different voltages in the same design, also known as a multi-voltage (MSV) design, you might need to specify the power domain in which the fillers are to be inserted using the `-powerDomain` parameter of the [addFiller](#) command.

**Default:** If this parameter is not specified, and a list of filler cells is specified with the `-cell` parameter, the [addFiller](#) command tries to add fillers to all power domains.

## Deleting Filler Cells

To remove added filler cells, use the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes only filler cells that are completely contained within the area; it does not delete filler cells that cross the area boundary.

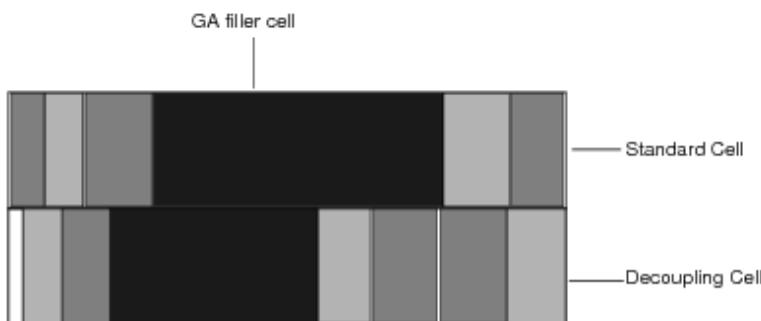
## Placing Gate Array Style Filler Cells for Post-Mask ECO

You can use pre-existing Gate Array (GA) style filler cells in regular `CORE` sites during a post-mask ECO flow. As such, the placer can add GA fillers at any grid location, rather than in a GA CORE grid.

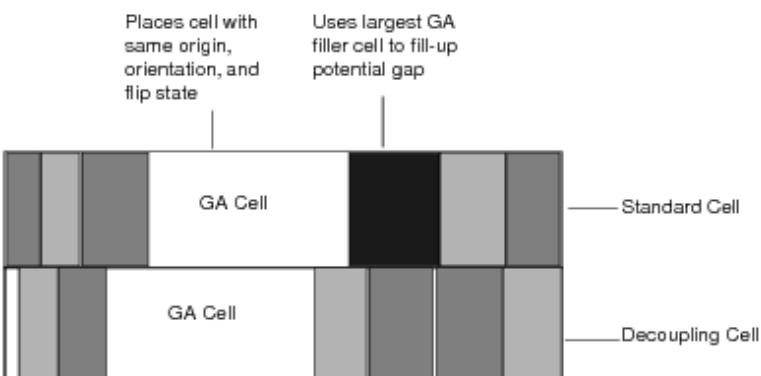
- Specify the following command:

```
ecoPlace -useGAFillerCells GAFillerCells
```

The placer first finds the optimal location each instance based on its connectivity, then searches for the nearest GA filler cell that is equal to or larger in size. A GA cell is placed at the GA filler location, and the original GA filler is deleted.



If the GA filler is larger than the GA cell, the placer creates a new GA filler instance using the list of GA filler cells you provide and places the filler in the gap.



The `ecoPlace` command also contains options that let you map unplaced standard cells to spare cells,

and map GA cells to GA core sites. You can specify that instances that are `PLACED` cannot be moved.

For more information, see the following command in the *Innovus Text Command Reference*:

- `ecoPlace` in the "Interactive ECO Commands" chapter

## Adding Decoupling Capacitance

Adding decoupling capacitance to a design can help maintain a stable voltage between power and ground when signal nets switch. This can reduce IR drop for power nets and limit bouncing on ground nets.

The Innovus software adds decoupling capacitance by choosing from the specified available decoupling capacitance cell candidates, and adding enough cells until their combined total capacitance value equals the user-specified value. You can insert decoupling capacitance homogeneously inside a specified area, or based on the peak current density of the instances in the area.

1. To define the cells to use for decoupling capacitance insertion, use the `addDeCapCellCandidates` command.

For example, the following commands define two decoupling capacitance cell candidates: `DECAP1` has a capacitance value of 10fF, and `DECAP8` has a capacitance value of 5fF.

```
addDeCapCellCandidates DECAP1 10  
addDeCapCellCandidates DECAP8 5:
```

2. To add the specified total decoupling capacitance to the design, use the `addDeCap` command.

For example, the following command adds 1000 fF of capacitance to the design using `DECAP1` and `DECAP8` cells:

```
addDeCap -totCap 1000 -cells DECAP1 DECAP8
```

## Deleting Decoupling Capacitance

- To clear all available decoupling cell candidates, use the `clearDeCapCellCandidates` command.
- To delete all of the decoupling capacitance cells in a design, use the `deleteDeCap` command.

## Adding Logical Tie-Off Cells

Tie-off cell instances provide connectivity between the tie-hi and tie-lo logical input pins of the netlist instances to power and ground. This connectivity does not cross the hierarchy module boundaries. The number of tie-off instances added can be controlled by setting the distance and fanout constraints using the [setTieHiLoMode](#) command.

To add logical tie-off cells to the design after placing the netlist, use the [Place Menu - Add Tie Hi/Lo](#) form or the [addTieHiLo](#) command. To remove added logical tie-off cell instances, you can use the [deleteTieHiLo](#) command.

## Saving Placement Data

You can save placement data in the Innovus place format or in DEF and PDEF placement data formats. This can be done at any time after running placement. To save placement data, use the [savePlace](#) command or the [saveDesign](#) command.

## Specifying and Placing JTAG and Other Cells Close to the I/Os

You can constrain the placement of JTAG cells and other cells so they are placed close to the outer core area. Place these cells before you run placement in the rest of the design.

When the software runs JTAG placement, it creates a temporary blockage over the area where the cells must not be placed and removes it after the placement.

You can constrain the placement of instances, hierarchical instances, or cells.

1. Use the following command to constrain placement:

```
specifyJtag
```

To include instances or cells other than JTAG cells, you must identify them with the [specifyJtag](#) command.

To undo the specification, use the following command:

```
unspecifyJtag
```

2. To place the instances or cells, use the following command:

```
placeJtag
```

3. (optional) To generate a report of the JTAG placement, use the following command:

```
reportJtagInst
```

To undo JTAG placement, use the following command:

```
unplaceJTAG
```

 If you do not want to place regular instances in the JTAG outer core area after running JTAG placement, specify a placement blockage prior to running placement.

## Related Topics

For more information, see the following commands in the "Placement Commands" chapter of the *Innovus User Guide*:

- [specifyJtag](#)
- [unspecifyJtag](#)
- [placeJtag](#)
- [reportJtagInst](#)
- [unplaceJtag](#)
- [traceJtag](#)

## Optimizing and Reordering Scan Chains

The `place_opt_design` command reorders scan chains by default, unless it is in prototyping mode. If you decide not to reorder scan cells with `place_opt_design` but run standalone scan chain reorder flow, use the information provided in this section.

## Related Topics

To see this step in the design flow, see "Place the Design and Run PreCTS Optimization" in the *Innovus Foundation Flows: Flat Implementation Flow Guide*.

## Specifying Scan Cells

Scan cells are usually identified and read automatically from the timing library during design import. Use the `specifyScanCell` command to define scan cells that the software cannot retrieve from the library.

You can specify scan chains in a design by defining them in a DEF file, or by using the `specifyScanChain` command.

If scan chains are specified by reading in a DEF file, the software does a native scan trace. The scan DEF file is stored in the database and, when the `scanReorder` command runs, the software matches the scans and honors the ordered segments. However, if you run `specify setPlaceMode -reorderScan false`, the software does not perform scan chain reordering, so the DEF file will not include the `+ ORDERED` statement in the SCANCHAINS section.

## About Scan Chains

If you do not need to retain the scan chain order in your design, you can change the order of the scan flip-flop connections along any or all scan chains. Changing the connection order eases connection constraints on the scan cells, but does not constrain their placement.

To facilitate reordering of the scan nets, unify the incoming netlist and make sure that it does not contain Verilog assignment statements involving scan nets. A scan net is a net that resides along the scan datapath--that is, a net that connects the scan flip-flops in a scan chain.

When the Innovus software reads the netlist, it outputs the following messages:

```
Reading netlist ...
```

```
Reading verilog netlist ".fileName"
```

```
Inserting temporary buffers to remove assignment statements.
```

## Reordering Scan Chains

Use one of the following approaches to scan chain reordering:

- Native scan reordering

Use this approach in the following conditions:

- Single-clock domain, single-edge chains
- Multiple clock domain chain segments separated by data lockup elements
- Shared functional output signal chains
- ScanDEF-based reordering

Use this approach in the following conditions:

- All simple scan chain architectures (handled by the native approach)
  - Implied domain transition scan chains (without data lockup elements)
  - Scan chains with ordered segments
  - Scan chains generated by LogicVision software

After reordering scan chains, save a netlist of the design using one of the following methods:

- *Save - Netlist* form (*Design - Save - Netlist*)
- `saveNetlist` command

## Native Scan Reordering Approach

Use the native approach to scan chain reordering when you do not have a scanDEF file.

This approach requires that you use the `specifyScanChain` command to identify the `START` and `STOP` signals of the top-level chains, or chain segments, in the netlist. Using this information, the software identifies the scan flip-flops along the scan chain when running the `scanTrace` command to analyze the scan flip-flop connections. You can also auto-detect data lockup latch elements using the `scanTrace -lockup` command.

If the scan cells are not listed in the timing library, you must specify them before tracing the scan chains. You can identify scan cells with the `specifyScanCell` command.

After `scanTrace` has identified the elements along the chain, complete the following steps:

1. (Optional) Ignore the scan connections:

```
setPlaceMode -ignoreScan true
```

2. (Optional) Set scan reorder options:

```
setScanReorderMode -skipMode [skipNone | skipBuffer | skipTwoPinCell]
```

**3. Run placement:**

```
place_opt_design
```

The recommended flow for scan chains that have data lockup latches is as follows:

**1. Specify a scan chain in the design:**

```
specifyScanChain
```

**2. Trace the scan chain connection with the automatic detection lockup latch elements:**

```
scanTrace -lockup [-verbose]
```

**3. (Optional) Ignore the scan connections:**

```
setPlaceMode -ignoreScan true
```

**4. (Optional) Set scan reorder options:**

```
setScanReorderMode [-skipNone | -skipBuffer | -skipTwoPinCell]
```

**5. Run placement:**

```
place_opt_design
```

The recommended flow for scan chains that have data lockup flip-flops is as follows

**1. Specify a scan chain in the design:**

```
specifyScanChain ...
```

**2. Specify a cell or instance as a lockup flip-flop element:**

```
specifyLockupElement ...
```

**3. (Optional) Ignore the scan connections:**

```
setPlaceMode -ignoreScan true
```

**4. (Optional) Set scan reorder options:**

```
setScanReorderMode -skipMode [skipNone | skipBuffer | skipTwoPinCell]
```

**5. Run placement:**

```
place_opt_design
```

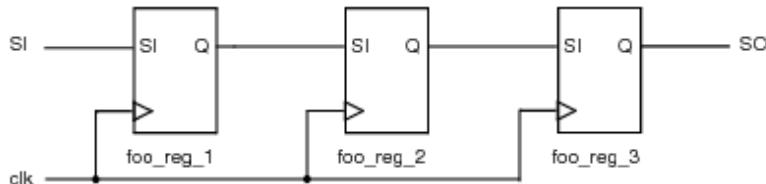
**Note:** The `scanReorder` command automatically calls `scanTrace` internally if you have not previously run `scanTrace`. By default, this internal `scanTrace` run specifies that the tracing will not detect lockup elements (`-noLockup`); therefore, if you have lockup latches, Cadence recommends using the `scanTrace -lockup` command before `scanReorder`, or specify `LockupElement` prior to running `scanReorder`.

## Valid Design Types

You can use the native approach to scan chain reordering on designs comprising a simple scan chain architecture with the following characteristics:

- Single-clock domain, single-edge chains

In the following figure, all `foo_reg` scan flip-flops are triggered by the same clock domain and phase.

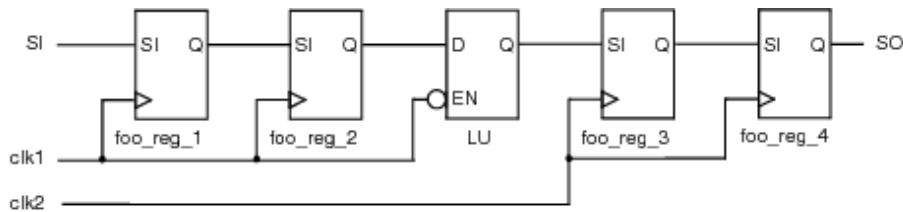


- `foo_reg_1`, `foo_reg_2`, and `foo_reg_3` scan flip-flops are triggered by `clk1` (positive edge).

```
specifyScanChain chain1 -start SI -stop SO
scanTrace [-verbose]
```

- Multiple clock domain chain segments separated by data lockup elements

In the following figure, all domain or edge transitions are separated by a data lockup element.



- `foo_reg_1` and `foo_reg_2` scan flip-flops are triggered by `clk1` (positive edge).
- `foo_reg_3` and `foo_reg_4` scan flip-flops are triggered by `clk2` (positive edge).
  - LU represents a data lockup element of type latch.

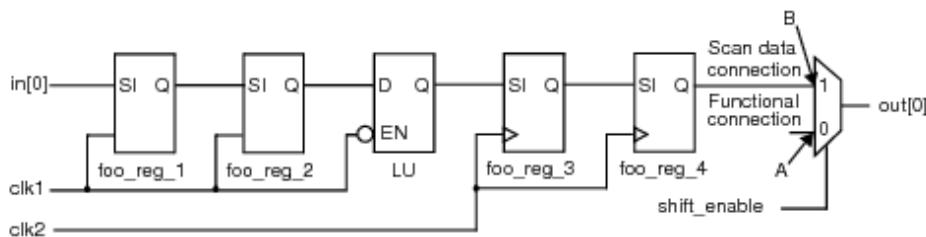
```
specifyScanChain chain1 -start SI -stop SO
scanTrace -lockup [-verbose]
```

All elements along the scan chain are assumed reorderable from the specified START and STOP signals unless there is a data lockup element in the scan data path. The presence of a data lockup element works as a boundary so that the chain segments on either side of the lockup element are individually reordered. For this example, the top-level chain is reordered as two individual scan chain segments:

- reorderable segment 1: SI > LU/D
- reorderable segment 2: LU/Q > SO

- Shared functional output signal chains

If the STOP signal of the scan chain is also a shared functional output, the endpoint of the scan chain must be specified to the scan input (SI) pin of the last register in the scan chain, or to the data input pin of the multiplexer (MUX), which drives the shared functional output signal. This is necessary because `scanTrace` does not perform the forward trace from the last flip-flop in the scan chain through the MUX instance. The following figure is an example of shared functional output:



The following command sequence performs the forward trace from the last flip-flop in the scan chain to the MUX instance:

```
specifyScanChain chain1 -start in[0] -stop MUX/B
scanTrace -lockup [-verbose]
```

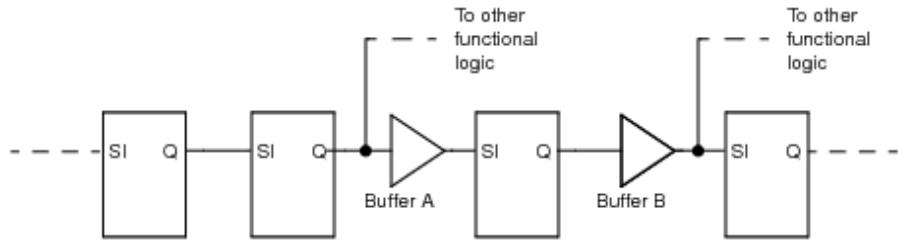
The following command sequence does not perform the forward trace from the last flip-flop through the MUX instance; `scanTrace` will not succeed:

```
specifyScanChain chain1 -start in[0] -stop out[0]
scanTrace -lockup [-verbose]
```

## Scan Chains with Two-Pin Logic Cells

Scan chains often contain two-pin logic cells, usually buffers. The scan tracing algorithm always recognizes and traces through two-pin cells. The `scanReorder` command parameters control whether two-pin cells remain in the scan chain after scan reordering.

In the following scan chain example, buffer A is in the scan chain, but not part of the functional logic of the design, and therefore can be deleted. Buffer B is part of the functional logic, and must not be deleted.



The `scanReorder -skipMode skipNone` command retains all two-pin cells in the scan chain, and reordering changes only the connections to the two-pin cells' outputs. The nets connected to the two-pin cells' inputs will not be modified. In the preceding example, both buffers A and B would be retained, and scan reordering would be performed by rearranging the scan input pin connected to their output nets.

The `scanReorder -skipMode skipBuffer` command reconnects the scan chain so that buffers (as defined by `setBufFootPrint`) are skipped. Buffers that are not part of the functional logic are deleted. This disconnects scan inputs and reconnects them directly to a scan output pin, skipping all buffers. Any two-pin cells that are not buffers are retained in the scan chain in the same manner as `skipNone`.

In the preceding example, buffer A would be deleted and functional buffer B would be retained in the netlist. Scan reordering would disconnect the scan input pin from the output of buffer B, and reconnect some other scan input pin to the input net of buffer B, so buffer B would no longer be in the scan chain.

The `scanReorder -skipMode skipTwoPinCell` command works the same as `scanReorder -skipMode skipBuffer`, except that it is not limited to buffers. Any two-pin cell will be treated as `skipBuffer` would treat a buffer.

- i** Because `scanReorder -skipMode skipTwoPinCell` does not consider the functionality of cells that it removes from the scan chain, it can change the scan chain in unpredictable ways. For example, if buffer A in the preceding example was an inverter, it would be removed, and the test pattern would have to be changed to account for the loss of inversion.

## scanDEF-Based Reordering Approach

If you have a scanDEF file that describes the set of reorderable scan chains in the design, Cadence recommends using the scanDEF approach. To reorder scan chains with the scanDEF approach, complete the following steps:

1. Read in the scanDEF file:

```
defIn -scanChain
```

**Note:** In the case where a DEF file contains a SCANCHAIN section, the `defIn` command automatically reads in the scanDEF file, so the `-scanChain` parameter is not necessary.

2. Run placement:

```
place_opt_design
```

## Using the `scanReorder` Command

When running the `scanReorder` command, the Innovus software uses the begin and endpoints from the scanDEF chains to trace the connectivity of the scan chains in the netlist. This check verifies whether the elements in the netlist scan chains are represented as elements in their respective scanDEF chains. As a result of this check, an internal representation of each scanDEF chain is created in the Innovus database.

When a netlist-to-scanDEF file mismatch occurs, for each instance mismatched, `scanReorder` issues the following WARNING message:

```
WARNING (SOCSC-5003): The scan chain was found to pass through instance <inst> in the netlist, but this instance does not appear in the DEF scan chain.
```

Mismatches of combinational components (buffers or inverters) in the scan data path can be expected if the netlist has undergone pre-placement optimization, or if the scanDEF file is not properly formatted, as described in Netlist-to-scanDEF mismatch section. Sequential mismatches are tolerated if the mismatch

occurs for a scan flop from the `FLOATING` section only of the scanDEF chain. However, sequential mismatches are not expected and indicate a discrepancy between the scan chains in the netlist, and the scanDEF chains. You should investigate the source of the discrepancy before proceeding with reordering. If necessary, revise the scanDEF description of the scan chains.

Using the internal representation of the scanDEF chains, Innovus issues the following message prior to reordering the chains in the netlist:

INFO: Scan reorder based on traced netlist chains.

INFO: Medium effort Scan reorder

INFO: Reordering scan chain <chainName>

## ***Netlist-to-scanDEF Mismatch***

Netlist-to-scanDEF mismatches can occur if a driving scan flip-flop is buffered (or inverted) to the `SI` pin of the next scan flip-flop in the scan chain. In this situation, the driving scan flop and buffer (or inverter) should be captured to the scanDEF file as an `ORDERED` segment, rather than capturing the driving scan flip-flop as a freely reorderable element in the `FLOATING` section of the scanDEF chain. The correct syntax for the `FLOATING` and `ORDERED` sections of the scanDEF file is as follows:

```
- chain X
  + START PIN
  + FLOATING
  ....
  next_scan_flop_reg ( IN SI ) ( OUT SO )
+ ORDERED
  driving_scan_flop_reg ( IN SI ) ( OUT SO )
  buf_instance ( IN A ) ( OUT Y )
+ STOP
```

In previous releases of Innovus, when a scanDEF to netlist mismatch occurred, scan reorder would abort. If the mismatches were due to combinational components (buffers or inverters) in the scan data path, you could still proceed with scan reordering by issuing `scanReorder` with the following parameters:

`scanReorder -defInForce`

For backward compatibility, these options are maintained in this release of the tool. However, in order to leverage the new netlist-to-scanDEF tracing feature, you should remove these parameters from the `scanReorder` command

The `-defInForce` parameter forces reordering to use the scanDEF file.

## scanDEF File Format

The scanDEF file follows a pin-based format that describes the set of scan chains or chain segments which are reorderable in the design. The syntax is as follows:

```
SCANCHAINS numScanChains ;
[- chainName
 [+ COMMONSCANPINS [( IN pin )][( OUT pin )] ]
 [+ START {fixedInComp | PIN} [outPin] ]
 {+ FLOATING {floatingComp [( IN pin )] [( OUT pin )]}...}
 [+ ORDERED
    {fixedComp [( IN pin )] [( OUT pin )]
     fixedComp [( IN pin )] [( OUT pin )]}
    [fixedComp [( IN pin )] [( OUT pin )] ]...]
 [+ STOP {fixedOutComp | PIN} [inPin] ];...]
END SCANCHAINS
```

The logic synthesis tool writes the input scanDEF file after the top-level scan chains are created in the design. Each top-level scan chain can be segmented into multiple scanDEF chains because the elements along each scanDEF chain must belong to the same clock domain, and be triggered by the same active edge of clock. Scan flip-flops that are freely reorderable along the scan chain are captured to the FLOATING section. Fixed segments (a set of connected elements), which are reordered as a fixed entity along the scan chain, are captured to the ORDERED section. Each scan chain must also have a START and STOP signal that defines the reordering start and end points of the scan chain.

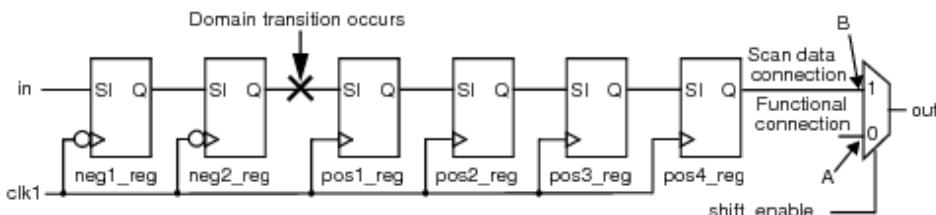
**Note:** You can use the following RTL Compiler command: `write_scandef > fileName`

## Valid Design Types

You can use the scanDEF approach to reorder top-level scan chains. This section provides a reordering example for implied domain transition scan chains, and an example of scan chains with fixed-ordered segments. You can also use this approach with all simple scan chain architectures that can use the native approach, as well as scan chains generated by LogicVision software.

- Implied domain transition scan chains

The scan flip-flops are triggered by alternate active edges of the same clock domain. The negative (positive) edge triggered segment precedes the positive (negative) edge triggered segments, respectively. In the following example, the implied domain transition occurs at `neg2_reg` to `pos1_reg`:



In this example, the two scan chain segments are as follows:

- c1k1 (negative edge) consisting of elements neg1\_reg and neg2\_reg
- c1k1 (positive edge) consisting of elements pos1\_reg, pos2\_reg, pos3\_reg, and pos4\_reg

Because the domain transition is done implicitly (without a data lockup element), the scan chain must be segmented to be properly reordered. In the scanDEF format, the top-level chain becomes two scanDEF chains, segmented by clock domain and clock edge; the pos1\_reg scan flip-flop is sacrificed to anchor the domain transition.

This register becomes an internal end and internal being point of scan DEF chains (chain1 and chain2 respectively):

```
SCANCHAINS 2 ;
- chain1
+ START pin in
+ FLOATING
    neg1_reg ( IN SI ) ( OUT Q )
    neg2_reg ( IN SI ) ( OUT Q )
+ STOP pos1_reg SI
;
- chain2
+ START pos1_reg Q
+ FLOATING
    pos2_reg ( IN SI ) ( OUT Q )
    pos3_reg ( IN SI ) ( OUT Q )
+ STOP pos4_reg SI
;
END SCANCHAINS
```

**Note:** The shared functional output signal (out) is not the STOP signal of the second scan chain segment. Instead, the scan chain is terminated to the IN pin of the last scan flop in the positive-edge triggered segment (BuildGates/PKS), or terminated to the data input pin of the MUX (other third-party tools).

- Scan chains with ORDERED segments

An order segment is a set of connected elements that can be reconnected along the scan

chain based on its placement. Reconnection to the fixed segment occurs using the `IN` pin of the first element and the `OUT` pin of the last element of the ordered segment. The connections of the other elements in the ordered segment are presumed connected and remain as intact connections. When an `ORDERED` segment is reconnected in the scan chain, the location of the `ORDERED` segment appears as a comment in the `FLOATING` section and again in the `ORDERED` section in order to correlate the segment to its location in the `FLOATING` section. The notation is as follows:

```
# ORDERED segment integer ;
```

The integer corresponds to as many `ORDERED` segments as defined in the original scan chain. For example, a `scanDEF` chain with one `ORDERED` segment is as follows:

```
SCANCHAINS 1 ;
- chain0
  + START PIN scan_in
  + FLOATING
    out_reg_0 ( IN SI ) ( OUT Q )
    out_reg_1 ( IN SI ) ( OUT Q )
    out_reg_2 ( IN SI ) ( OUT Q )
    out_reg_3 ( IN SI ) ( OUT Q )
  + ORDERED
    out_reg_4 ( IN SI ) ( OUT Q )
    u_buf ( IN A ) ( OUT Y )
  + STOP PIN scan_out ;
END SCANCHAINS
```

After reordering the output, the `scanDEF` file is as follows:

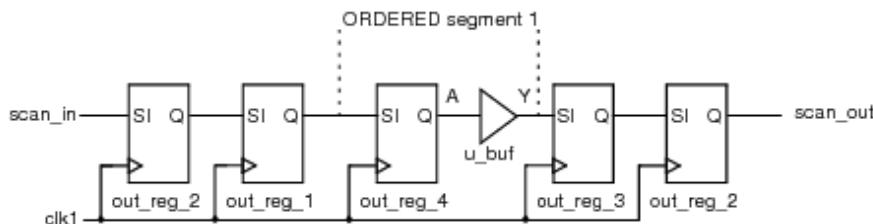
```
SCANCHAINS 1 ;
- chain0
  + START PIN scan_in
  + FLOATING
    out_reg_2 ( IN SI ) ( OUT Q )
    out_reg_1 ( IN SI ) ( OUT Q )
  # ORDERED segment 1
    out_reg_3 ( IN SI ) ( OUT Q )
    out_reg_0 ( IN SI ) ( OUT Q )
  + ORDERED
  # ORDERED segment 1
    out_reg_4 ( IN SI ) ( OUT Q )
```

```

    u_buf ( IN A ) ( OUT Y )
    + STOP PIN scan_out ;
END SCANCHAINS

```

Therefore, the connectivity of the elements along the reordered scan chain is as follows:



## Saving Scan Files

After scan reorder is run, save a DEF file using the following command:

```
defOutBySection -noNets -noComps -scanChains
```

With this command, you can view the new order of elements along the scan chain.

To save scan files, use the *Save DEF* form or the [defOut](#) command.

## Loading Scan Files

To load scan files in DEF format, use the *Load DEF File* form. For DEF, use the [defIn](#) command.

# Clock Tree Synthesis

- [The Clock Tree Synthesis Engines](#)
- [Overview](#)
- [Flow and Quick Start](#)
  - [Quick Start Example](#)
- [Configuration and Method](#)
  - [Properties System](#)
  - [Route Types](#)
  - [Library Cells](#)
  - [Transition Target](#)
  - [Skew Target](#)
  - [Creating the Clock Tree Specification](#)
  - [Configuration Check](#)
  - [CCOpt Effort](#)
  - [Create Preferred Cells Stripes to Control IR Drop](#)
  - [Common Specification Modifications](#)
  - [Restricting CCOpt Skew Scheduling](#)
  - [Method](#)
- [Concepts and Clock Tree Specification](#)
  - [Graph-Based CTS](#)
  - [Clock Trees and Skew Groups](#)
  - [Automatic Clock Tree Specification Creation](#)
  - [Manual Setup and Adjustment of the Clock Specification](#)
  - [Deleting the Clock Tree Specification](#)
  - [Chains](#)
- [Reporting](#)
  - [Get Commands](#)
  - [Skew Groups](#)
  - [Clock Trees](#)
  - [Timing Data for Reports](#)
  - [Worst Chain](#)
  - [Halo Violations](#)
  - [Cell Name Information](#)

- Clock Tree Convergence
- CCOpt Clock Tree Debugger
  - Launching the CCOpt CTD
  - CCOpt CTD Interface
  - Key Features of the CTD
- Additional Topics
  - Source Latency Update
  - Cell Halos
  - Power Management
  - Shared Clock and Data Concerns
- CCOpt Property System
  - Setting Properties
  - Getting Properties
- Migrating from FE-CTS
  - Specification Translation
  - Concept Mapping
- Legacy FE-CTS Flow
  - Creating the `clockDesign` Specification File
  - Synthesizing the Clock Tree with `clockDesign`
  - Analyzing and Debugging the `clockDesign` Results
  - Optimizing a `clockDesign` Built Clock Tree

## The Clock Tree Synthesis Engines

The Innovus™ Implementation System (Innovus) product family offers several types of clock tree synthesis (CTS):

- **CCOpt** – Full Clock Concurrent Optimization (CCOpt). The command `ccopt_design`, without the `-cts` parameter, is used to invoke this.
- **CCOpt-CTS** – Global skew balanced CTS using the CCOpt CTS engine. In the 14.2 and later versions of the software, this is the default CTS engine when the `clockDesign` command is invoked. Existing users of CCOpt-CTS will be familiar with the `ccopt_design -cts` command. CCOpt-CTS can be used with a clock tree specification generated from SDC timing constraints, or for backward compatibility from a FE-CTS clock specification.

- **Legacy FE-CTS** – Global skew balanced CTS using the FE-CTS engine. In the 14.1 and earlier versions of the software, this is the default engine involved by the `clockDesign` command.

The table below summarizes ways in which the different CTS engines can be invoked. The `clockDesign` command's default behavior is changed in the 14.2 release of the software. By default, the 14.2 and later versions of the software will use the CCOpt-CTS engine.

| <b>Command</b>                         | <b>setCTSMODE - engine Setting</b> | <b>Engine Used</b>                                                                                                                                                                                                                  |
|----------------------------------------|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ccopt_design</code>              | ignored                            | CCOpt: Full clock concurrent optimization is performed. Use <code>create_ccopt_clock_tree_spec</code> to create a clock tree specification from SDC constraints.                                                                    |
| <code>ccopt_design -cts</code>         | ignored                            | CCOpt-CTS : Global skew balancing using CCOpt-CTS engine. Use <code>create_ccopt_clock_tree_spec</code> to create a clock tree specification from SDC constraints.                                                                  |
| <code>ccopt_design -cts -ckSpec</code> | ignored                            | CCOpt-CTS : Global skew balancing using CCOpt-CTS engine driven by a FE-CTS specification for backward compatibility.                                                                                                               |
| <code>clockDesign</code>               | <code>auto</code> (default)        | If an FE-CTS specification is not loaded then the engine used is as per the <code>ccopt</code> engine setting. If an FE-CTS specification is loaded then the engine used is as per <code>ccopt_from_edi_spec</code> engine setting. |
| <code>clockDesign</code>               | <code>ccopt</code>                 | Same as <code>ccopt_design -cts</code> , but automatically invokes <code>create_ccopt_clock_tree_spec</code> first to create a clock tree specification from SDC constraints.                                                       |
| <code>clockDesign</code>               | <code>ccopt_from_edi_spec</code>   | Same as <code>ccopt_design -cts -ckSpec</code> .                                                                                                                                                                                    |
| <code>clockDesign</code>               | <code>ck</code>                    | Invokes the legacy FE-CTS engine. This is identical to the behavior of the <code>clockDesign</code> command in software versions prior to 14.2.                                                                                     |

To obtain the `clockDesign` behavior of the software version 14.1 and earlier, use the following commands:

```
setCTSMODE -engine ck
clockDesign ...
```

## Overview

CCOpt-CTS is the CTS engine underpinning CCOpt. The benefits of CCOpt-CTS include the following:

- Automatic creation of the clock tree specification from multi-mode SDC constraints via the `create_ccopt_clock_tree_spec` command. The skew group data model permits complex balancing relationships to be captured.
- The use of a graph-based algorithm avoids the need for sequential CTS between modes and avoids over balancing in complex multi-mode clock networks.
- Graphical CCOpt Clock Tree Debugger to visualize and debug clock trees.

For an introduction to CCOpt concepts including clock trees and skew groups, see the [Concepts and Clock Tree Specification](#) section.

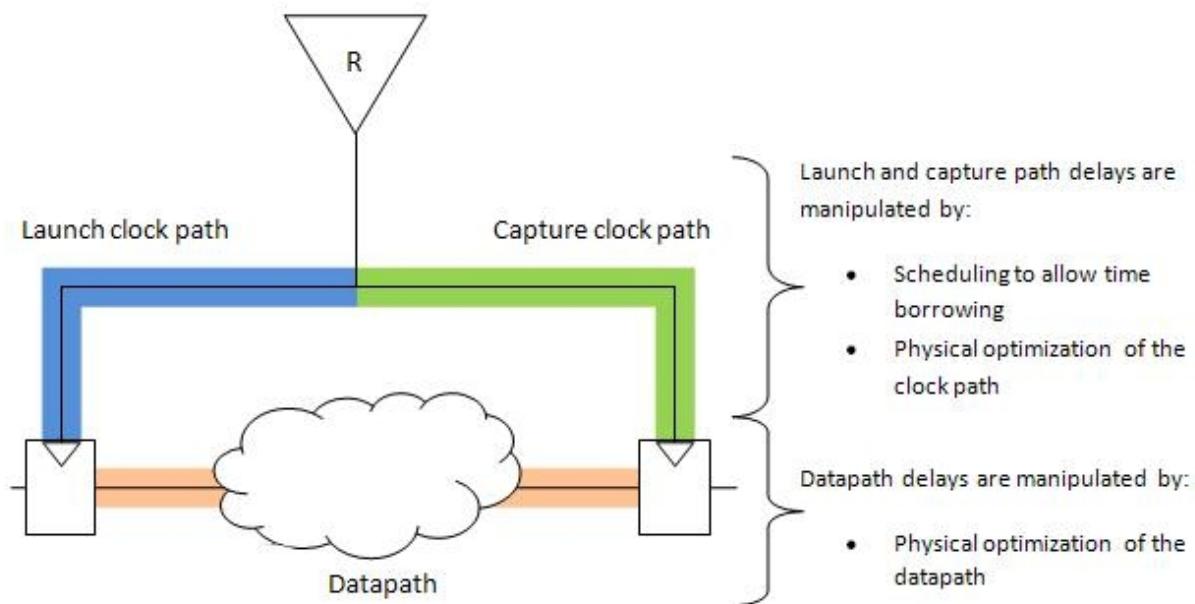
CCOpt extends CCOpt-CTS to replace traditional global skew balancing with a combination of CTS, timing driven useful skew, and datapath optimization. In traditional CTS flows an ideal clock model is used before CTS to simplify clock timing analysis. With the ideal clock model, launch and capture clock paths are assumed to have the same delay. After CTS, the ideal clock model is replaced by a propagated clock model that takes account of actual delays along clock launch and capture paths. In traditional CTS global skew balancing attempts to make the propagated clock timing match the ideal mode clock timing by balancing the insertion delay (clock latency) between all sinks. However, a number of factors combine such that skew balancing does not lead to timing closure. These include:

- **OCV** – On-chip variation means that skew, measured using a single metric such as the ‘late’ configuration of a delay corner, no longer directly corresponds to timing impact because launch and capture paths have differing timing derates. In addition, Common Path Pessimism Removal (CPPR) and per-library cell timing derates mean that it is not possible to accurately estimate clock or datapath timing without synthesizing a clock tree. Advanced OCV (AOCV) further complicates this by adding path and bounding box dependent factors.
- **Clock gating** – Clock gating uses datapath signals to inhibit or permit clock edges to propagate from a clock source to clock sinks. The clock arrival time at a clock gating cell is unknown prior to CTS and this arrival time determines the required time for the datapath control signal to reach the clock gating cell enable input. Therefore the setup slack at a clock gating enable input is hard to predict preCTS. In addition, clock gating cells have an earlier clock arrival time than regular sinks and are therefore often timing critical. Typically, the fan-in registers controlling clock gating may need to have an earlier clock arrival time than regular sinks in order to avoid a clock gating slack violation – which means the fan-in registers need to be skewed early.
- **Unequal datapath delays** – Front end logic synthesis will attempt to ensure logic between registers is roughly delay balanced to optimize the target clock frequency. However, with wire delay dominating many datapath stages it is likely that after placement and preCTS optimization there will exist some combinational paths with unavoidably longer delays than others. Useful

skew clock scheduling permits slack to be moved between register stages to increase clock frequency. In contrast, global skew balancing is independent of timing slack. In addition, CCOpt useful skew scheduling can avoid unnecessarily balancing of sinks where there is excess slack in order to reduce clock area and clock power.

CCOpt treats both clock launch, clock capture, and datapath delays as flexible parameters that can be manipulated to optimize timing as illustrated below.

### Manipulating Clock Delays and Logic Delays



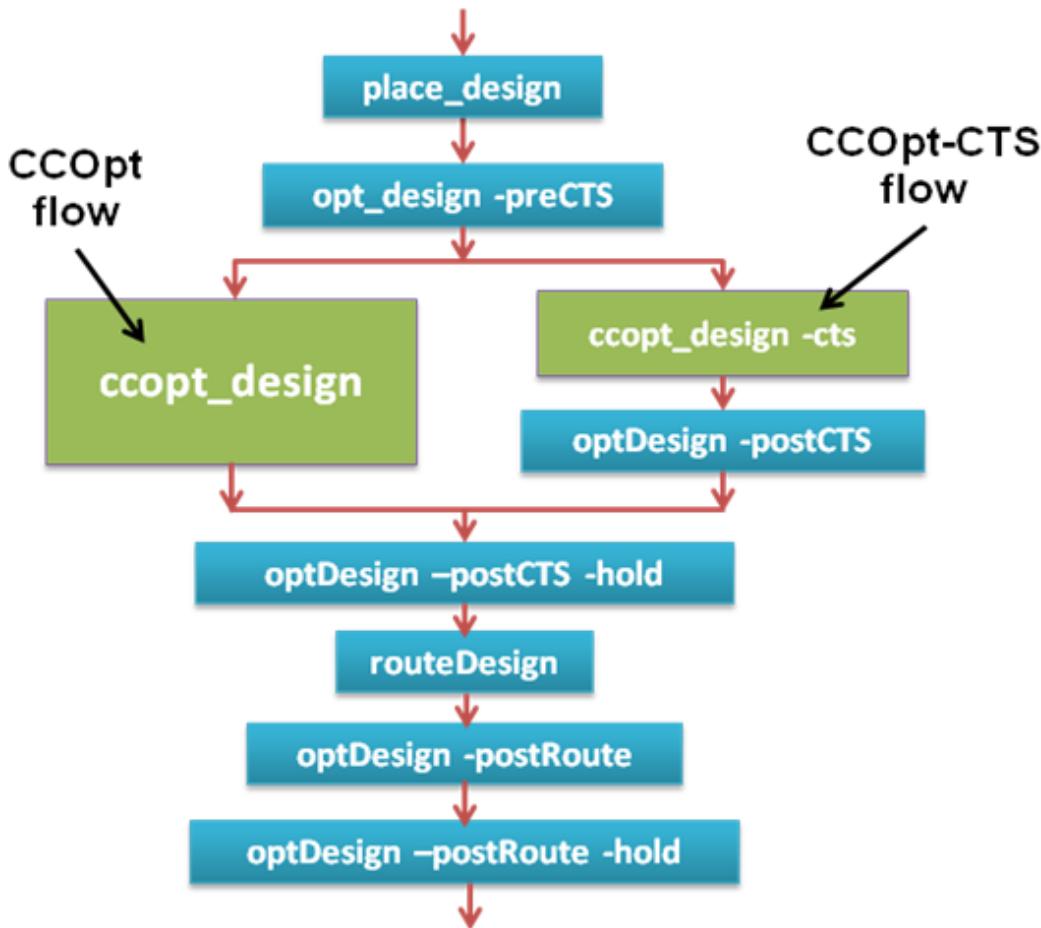
At each clock sink (flip-flop) in the design, CCOpt can adjust both datapath and clock delays in order to improve negative setup timing slack – specifically the high effort path group(s) WNS. This is performed using the propagated clock timing model at all times. Further discussion about the concept of moving slack between register stages and the concept of worst chains is discussed in the [Chains](#) section.

## Flow and Quick Start

**⚠** For users using a FE-CTS clock specification, this guide assumes the use of either `ccopt_design` or `ccopt_design -cts`. For information on using `ccopt_design -cts -ckSpec`, including the default behavior of `clockDesign` with a FE-CTS clock specification loaded, see the [Migrating from FE-CTS](#) section.

The diagram below highlights the CCOpt and CCOpt-CTS steps as part of the standard Innovus block implementation flow. In the CCOpt flow post-CTS optimization is not required because `ccopt_design` includes post-CTS style optimization both as part of clock concurrent optimization and as a further final internal optimization step.

### CCOpt and CCOpt-CTS in the Design Flow

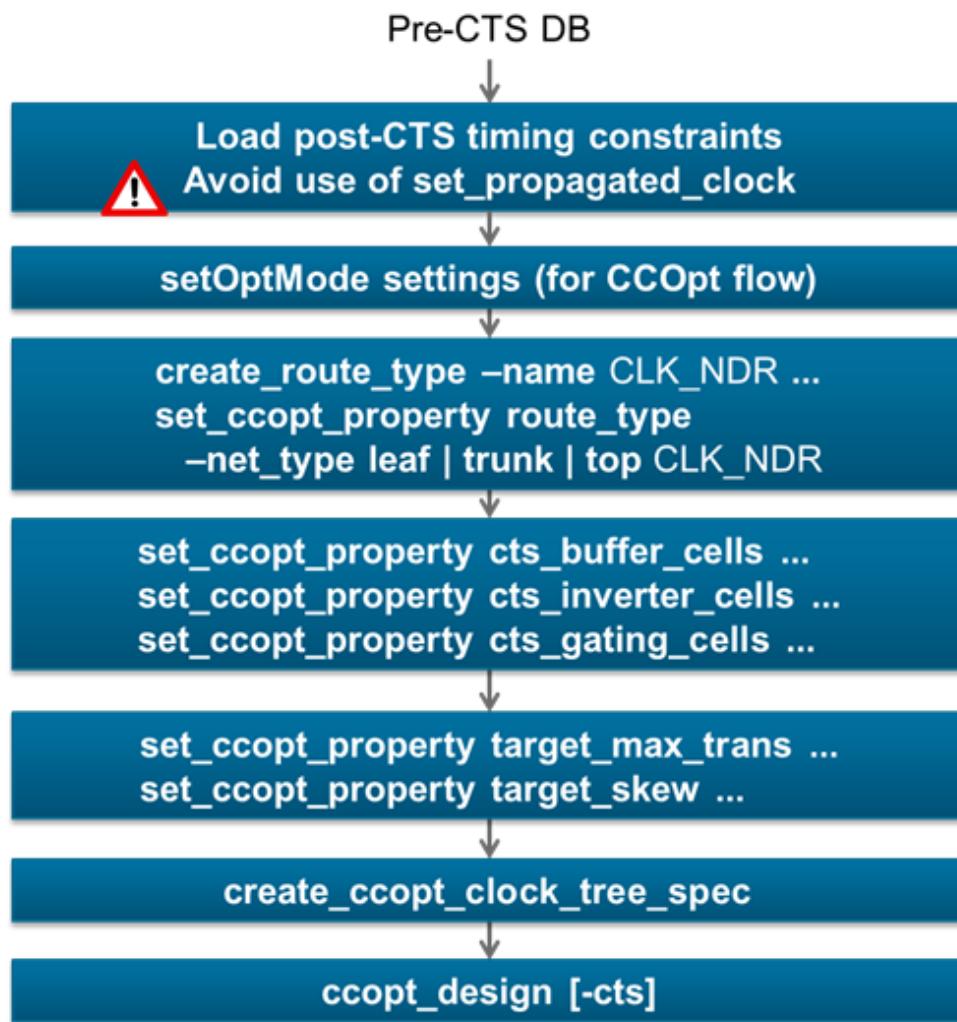


An important consideration for the CCOpt and CCOpt-CTS flows is the need for high quality multi-mode timing constraints. The best strategy is to use post-CTS timing constraints but with clocks in ideal

clocking mode. It is recommended to avoid the use of 'CTS-specific' SDC, which may be encountered in other tool flows or even with FE-CTS based flows.

The diagram below summarizes the key configuration steps.

### CCOpt and CCOpt-CTS Configuration Steps



It is important to apply the intended post-CTS configuration before invoking CCOpt but with clocks still in ideal timing mode. This is also recommended for use with CCOpt-CTS.

**Switch to propagated timing model** – CCOpt and CCOpt-CTS switch clocks to propagated mode and update source latencies of clock root pins such that the average arrival time of clocks after CTS matches the before CTS ideal mode arrival times. This process will not happen for clocks that are initially in propagated mode. The source latency update is important so that optimization of inter-clock and I/O boundary paths during `ccopt_design` operates with correct timing. In contrast, in a traditional flow you would do this as a separate flow step. The source latency update scheme is discussed further in the [Source Latency Update](#) section.

**Post-CTS configuration** – CCOpt combines CTS with datapath optimization replacing the need for a separate post-CTS setup timing optimization step. Before running CCOpt the session should be configured with post-CTS uncertainties, CPPR enabled, OCV timing derates or AOCV enabled, active analysis views and all other settings appropriate for post-CTS optimization. The `update_constraint_mode`, `setAnalysisMode`, and `setOptMode` commands are the most commonly used commands to configure relevant settings.

The example below illustrates a typical CCOpt/CCOpt-CTS configuration and run script. Some of these settings may also be configured via the Foundation Flow environment.

## Quick Start Example

Load post-CTS timing constraints. This example loads functional mode and scan mode constraints intended for post-CTS use. The user has ensured that clocks are not switched to propagated mode in these SDC files.

```
update_constraint_mode -name func -sdc_files func_postcts_no_prop_clock.sdc
update_constraint_mode -name scan -sdc_files scan_postcts_no_prop_clock.sdc
```

Ensure sufficient analysis views are active. Clock specification generation is fully multi-mode and it is important that views utilizing both functional and scan modes are active. CCOpt is hold-slack aware when permitting sinks to be unbalanced to reduce clock area. The first defined setup view is used to determine the primary CTS delay corner – this is the delay corner used by the majority of CTS, but note that CCOpt useful skew scheduling and timing optimization considers setup slack in all active analysis views.

```
set_analysis_view -setup {func_max_setup scan_max_setup} -hold {func_min_hold
scan_min_hold}
```

Define route types. A route type binds a non-default routing rule, preferred routing layers and a shielding specification together. NDRs can be defined via LEF or using the `add_ndr` command.

```
create_route_type -name leaf_rule -non_default_rule CTS_2W1S
-top_preferred_layer M5 -bottom_preferred_layer M4

create_route_type -name trunk_rule -non_default_rule CTS_2W2S
-top_preferred_layer M7 -bottom_preferred_layer M6
-shield_net VSS -bottom_shield_layer M6

create_route_type -name top_rule -non_default_rule CTS_2W2S
-top_preferred_layer M9 -bottom_preferred_layer M8
-shield_net VSS -bottom_shield_layer M8
```

Specify that the route types defined above will be used for leaf, trunk, and top nets respectively. Note

that top routing rules will not be used unless the `routing_top_min_fanout` property is also set.

```
set_ccopt_property -net_type leaf route_type leaf_rule
set_ccopt_property -net_type trunk route_type trunk_rule
set_ccopt_property -net_type top route_type top_rule
```

Specify that top routing rules will be used for any clock tree net with a transitive sink fanout count of over 10000.

```
set_ccopt_property routing_top_min_fanout 10000
```

Configure library cells for CTS to use. In this example, the `logic_cells` property is not defined so when resizing existing logic cell instances CTS will use matching family cells which are not `dont_use`. The specification of a library cell overrides any `dont_use` setting for that library cell.

```
set_ccopt_property buffer_cells { BUFX12 BUFX8 BUFX6 BUFX4 BUFX2 }
set_ccopt_property inverter_cells { INVX12 INVX8 INVX6 INVX4 INVX2 }
set_ccopt_property clock_gating_cells { PREICGX12 PREICG8 PREICGX6 PREICGX4 }
```

Include this setting to use inverters in preference to buffers.

```
set_ccopt_property use_inverters true
```

Configure the maximum transition target.

```
set_ccopt_property target_max_trans 100ps
```

Configure a skew target for CCOpt-CTS (`ccopt_design -cts`). This is ignored by CCOpt (`ccopt_design`).

```
set_ccopt_property target_skew 50ps
```

Create a clock tree specification by analyzing the timing graph structure of all active setup and hold analysis views. The clock tree specification contains `clock_tree`, `skew_group`, and property settings. Alternatively, the specification can be written to a file for inspection or debugging purposes and then loaded.

```
create_ccopt_clock_tree_spec
#create_ccopt_clock_tree_spec -filename ccopt.spec
#source ccopt.spec
```

Run CCOpt or CCOpt-CTS

```
ccopt_design
#ccopt_design -cts
```

Report on timing

```
timeDesign -postCTS -expandedViews -outDir post_cts_report_dir
```

Report on clock trees to check area and other statistics.

```
report_ccopt_clock_trees -filename clock_trees.rpt
```

Report on skew groups to check insertion delay and, if applicable, skew.

```
report_ccopt_skew_groups -filename skew_groups.rpt
```

Open the CCOpt Clock Tree Debugger Window. Alternatively, use the “CCOpt Clock Tree Debugger” entry in the main *Clock* menu.

```
ctd_win
```

For a more detailed explanation and recommendation on each of the above settings, see the [Configuration and Method](#) section. For details of the clock tree specification system, see the [Concepts and Clock Tree Specification](#) section.

## Configuration and Method

### Properties System

Configuration of CCOpt-CTS and CCOpt is performed using a combination of the clock tree specification and CCOpt properties.

To set a property:

```
set_ccopt_property [-object_type <object>] <property name> <property value>
```

To get a property:

```
get_ccopt_property [-<object_type> <object>] <property name>
```

To obtain help on a property, or to find properties matching a wildcard pattern:

```
get_ccopt_property -help <property name or pattern>
```

To obtain a list of all available properties use this command:

```
get_ccopt_property -help *
```

The help for each property indicates which object type(s) the property applies to. Many properties are global properties for which an object type is not specified, but there are also properties that are applicable to specific object types including pins, skew groups, clock trees, and types of nets.

For example:

```
get_ccopt_property -help target_max_trans
```

...

Optional applicable arguments: "-delay\_corner *name*", "-clock\_tree *name*", "-net\_type *name*", "-early" and "-late".

For a summary of all parameters of the `get_ccopt_property` and `set_ccopt_property` commands, see

the *Innovus Text Command Reference*. You can also use the `man` command, for example: `man get_ccopt_property`

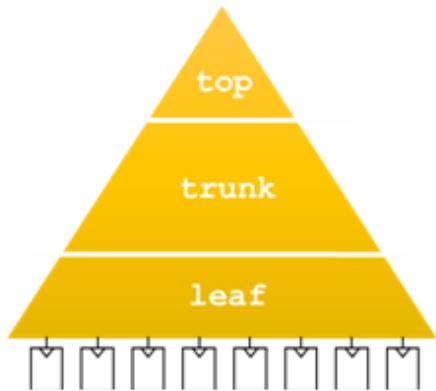
Note that some properties are read-only and can not be set. For further details of property manipulation, see the [CCOpt Property System](#) section.

For descriptions of all public CCOpt properties, see the [CCOpt Properties](#) chapter.

## Route Types

CCOpt-CTS and CCOpt use the concept of top, trunk, and leaf net types as illustrated below.

### Clock Tree Net Types



- **Leaf nets** – Any net that is connected to one or more clock tree sinks is a leaf net. By default, CCOpt-CTS and CCOpt will insert buffers so that no buffer drives both sinks and internal nodes.
- **Trunk nets** – Any net that is not a leaf net is by default a trunk net.
- **Top nets** – If you configure the `routing_top_min_fanout` property, then any trunk net which has a transitive fanout sink count higher than the configured count threshold will instead be a top net. For example, if the property is set to 10,000, then any trunk net that is above (in the clock tree fan-in cone) 10,000 or more sinks will be a top net.

You can define route types. A route type binds together a non-default routing rule (NDR), preferred routing layers, and a shielding specification. For each net type (leaf, trunk, and optionally top) you can specify the route type that should be used. Non-default routing rules can be defined via LEF or using the `add_ndr` command. For example, the following command creates a route type with the name `trunk_rule` that uses the `CTS_2W2S` non-default rule, with preferred routing layers `M6` and `M7`, with shielding in all layers down to `M6` inclusive.

```
create_route_type -name trunk_rule -non_default_rule CTS_2W2S  
-top_preferred_layer M7 -bottom_preferred_layer M6  
-shield_net VSS -bottom_shield_layer M6
```

The following command specifies that the `trunk_rule` route type should be used for the trunk net type:

```
set_ccopt_property -net_type trunk route_type trunk_rule
```

While the above command might seem superfluous, note that route types can be specified per clock tree, or per clock tree and net type combination. The [Quick Start Example](#) section illustrates commands to configure and apply route types for leaf, trunk, and top nets.

## Top Net Configuration

As discussed above, the `routing_top_min_fanout` property can be configured with a sink count threshold to determine which nets are considered top nets instead of trunk nets. This property can be set globally for all clock trees, or per individual clock tree. For example:

```
set_ccopt_property -clock_tree clk500m routing_top_min_fanout 10000
```

By default, each clock tree sink counts as ‘1’ for the purposes of top net thresholds. For macro clock input pins it may be desirable to treat the clock input pin as having a higher count, for example representing the number of internal state elements. The `routing_top_fanout_count` property can be used to configure this.

For example, to specify that the clock input `mem0/clkin` to a memory should count as 1000 sinks, use the following command:

```
set_ccopt_property -pin mem0/clkin routing_top_fanout_count 1000
```

## Routing Rule Recommendations

Clock net routing rules are crucial to obtaining low insertion delay and avoiding signal integrity problems. Especially for small geometry process nodes, the following recommendations should be considered:

- Configure the trunk net type to use double width double spacing and shielding. Prefer middle to higher layers subject to the power grid pattern. Double width is recommended to reduce resistance and permit use of bar shape vias, with double spacing to reduce the capacitance impact of shielding. Shielding is essential to avoid aggressors impacting clock trunk net timing, as impact on clock trunk timing is often significant for both WNS and TNS.
- Configure the leaf net type to use double width and prefer middle layers. Double width is recommended to reduce resistance. Extra spacing is desirable, but extra spacing and/or shielding may consume too much routing resource.

- Try to arrange for each net type (leaf, trunk, top) to use a single layer pair, one horizontal and one vertical, which have the same pitch, width, and spacing. This increases the correlation accuracy between routing estimates before clock nets are routed and actual routed nets.

## Library Cells

The library cells used by CCOpt-CTS and CCOpt are configured with the properties listed below. These cell lists may be configured per-power domain and per-clock tree by using the `-power_domain` and `-clock_tree` keys for these properties respectively.

|                                 |                                                                                                                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>buffer_cells</code>       | Specifies buffer, inverter, and clock gating cells. It is recommended to explicitly configure buffer, inverter, and clock gating cells.                                                                |
| <code>inverter_cells</code>     |                                                                                                                                                                                                        |
| <code>clock_gating_cells</code> |                                                                                                                                                                                                        |
| <code>logic_cells</code>        | Specifies logic cells. If logic cells are not specified, CTS will use any library cell which has the same logic function and is not <code>dont_use</code> when resizing existing logic cell instances. |
| <code>use_inverters</code>      | Specifies that CTS should prefer inverters to buffers.                                                                                                                                                 |

To view detailed information about above properties, run the following command:

```
get_ccopt_property -help propertyname
```

For example:

```
get_ccopt_property -help buffer_cells
```

A Tcl list of symmetric buffer cells. All buffers created in the clock tree under a power domain will be instances of these cells.

Set the global property to specify buffer cells for all clock trees and all power domains:

```
set_ccopt_property buffer_cells {bufA bufB bufC}
```

Set the per-clock tree/power domain property to specify buffer cells for a particular clock tree and ALL power domains:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk
```

Set the per-clock tree /power domain property to specify buffer cells for a particular clock tree and power domain:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk -power_domain pd
```

By default CCOpt automatically selects appropriate cells.

Valid values: `list lib_cell`

*Default:* {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

The [Quick Start Example](#) illustrates commands to configure library cells. Specifying that CTS can use a library cell overrides any user or library dont\_use setting for that library cell.

The following are some recommendations:

- Always specify library cells for buffers, inverters and clock gating.
- Use low threshold (LVT) cells. The resulting insertion delay will be lower leading to less impact of OCV timing derates, therefore reducing the datapath dynamic and leakage power increase from timing optimization.
- For many, but not all, low geometry processes inverters result in lower insertion delay and lower power than buffers. In older technologies, buffers may be more efficient. The exact preference here is technology, library, and design target dependent.
- Permitting the largest library cells to be used may be undesirable for electromigration reasons and can increase clock power.
- Very weak cells, for example, X3 and below in many libraries, are usually undesirable due to poor cross-corner scaling characteristics and are sensitive to detailed routing jogs and changes.
- Limiting the number of library cells to no more than 5 per cell type may help reduce run time.
- Do not exclude small cells, such as X4, as otherwise CTS will be forced to use larger more power and area consuming cells to balance skew or implement the useful skew schedule.
- Include always-on buffers and inverters in designs with multiple power domains.

## Transition Target

The maximum transition target to CCOpt is specified using the following command:

```
set_ccopt_property target_max_trans value
```

The value can be specified in library units, for example "100", or specified in explicit units for example "100ps", "0.1ns".

The transition target can be specified by net type, clock tree, and delay corner. For example, it may be desirable to have a tighter transition target at sink pins to improve flop CK->Q arc timing, but relax the transition target in trunk nets to reduce clock area and power. Shielding and extra spacing could be used for trunk nets to further reduce clock power whilst avoiding signal integrity problems. This example configures trunk nets to have a 150ps transition target whilst leaf nets have a 100ps transition target:

```
set_ccopt_property -net_type trunk target_max_trans 150ps
set_ccopt_property -net_type leaf target_max_trans 100ps
```

If a target max transition is not specified, CCOpt-CTS and CCOpt will examine the `target_max_trans_sdc` property (see the [SDC Transition Targets](#) section), and if that is not defined then an automatically generated target is chosen. It is recommended to set a transition target unless intentionally using per clock tree settings which are to be obtained from the SDC constraints.

## Skew Target

The skew target used by CCOpt-CTS can be configured globally as follows:

```
set_ccopt_property target_skew value
```

Alternatively, the target skew can be specified per skew group, for example:

```
set_ccopt_property -skew_group ck200m/func target_skew 0.1ns
```

If a target max transition is not specified, CCOpt-CTS will auto-generate a target. This may not be optimal. Note that CCOpt (`ccopt_design` without `-cts`) ignores skew targets, except where skew groups have been explicitly configured to restrict useful skew.

## Creating the Clock Tree Specification

The recommended method for generating a clock tree specification is to use the command `create_ccopt_clock_tree_spec`. This will analyze the timing graph of all active setup and active hold analysis views and create the specification. For details on how the specification creation process operates and the commands used in the specification, see the [Concepts and Clock Tree Specification](#) section.

To write the specification to a file, instead of just applying it immediately in memory, add the `-file` parameter. This example writes the specification to a file and then loads the specification.

```
create_ccopt_clock_tree_spec -filename ccopt.spec  
source ccopt.spec
```

## Configuration Check

The command `ccopt_design -check_prerequisites` can be used to perform setup, library, and design validation checks without actually running CTS. It is otherwise identical to the checks normally performed near the start of `ccopt_design`.

## CCOpt Effort

The command `set_ccopt_effort` can be used to specify the optimization effort used by CCOpt. The available options are high, medium, and low. The -high option is used to run the CCOpt flow with high effort. A medium effort is used to lower the optimization effort and with this mode the timing QOR is not expected to be as good as that compared to the high-effort setting but there are run-time benefits. The low effort setting runs the `ccopt_design -cts` and `optDesign` flow. This is the default setting. The `get_ccopt_effort` command can be used to check the current setting of the CCOpt effort level. Note that setting the effort level changes multiple internal properties ‘under the cover’. Advanced users using internal properties advised by their support contact should be aware of this.

## Create Preferred Cells Stripes to Control IR Drop

Large cells can induce a large IR drop when they switch. In some designs, where power switches occur in regular vertical columns across the chip, it is advantageous to try and place large CTS cells, which may switch often, closer to the power switches to control the R part of the IR drop.

To do this, you can create, one or more, preferred cell stripes objects, using the `create_ccopt_preferred_cell_stripes` command. This command lets you define a box in which large, specified, cells should be preferentially placed within the specified stripes; the pitch, width, height, bottom left corner, and the number of stripes is also configured. When these objects are created, CTS will try to place cells in the stripes, wherever possible.

For example, if power switches are arranged in vertical columns at intervals of 50um, are 5um wide, with the first column at 20um from the left of the chip, then you might create preferred cell stripes in the following manner:

```
create_ccopt_preferred_cell_stripes -name stripes_for_IR_drop_control -bbox {0 0 500 1000}
-bottom_edge 0 -height 1000 -left_edge 15 -width 15 -pitch 50 -repeat 10 -cells
{CTS_BUF_X16 CTS_BUF_X20 CTS_INV_X16 CTS_INV_X20 ICG_X16 ICG_X20}
```

Assuming that the chip bounds are {0 0 500 1000}, the above command will create a set of 10 stripes, each 15um wide, so that the large (x16 and x20) CTS cells are preferentially placed within 5um on either side of the columns of power switches.

However, this is not a hard constraint, so, if there is a scenario where in order to be DRV clean or to meet skew constraints, a cell needs to be outside the stripes, then CTS has the freedom to do that. But, wherever possible, it will use the area inside the stripes.

**Note:** These constraint objects should be created before running `ccopt_design`.

You can delete any incorrectly created preferred cell stripes objects and view a list of preferred cell stripes objects matching the supplied pattern by using the `delete_ccopt_preferred_cell_stripes` and `get_ccopt_preferred_cell_stripes` commands, respectively.

You can also specify the preferred cell stripes to which a property applies by using the `-preferred_cell_stripes` parameter of the [set\\_ccopt\\_property](#) command. For related properties, see the [CCOpt Properties](#) chapter in the *Innovus User Guide*. The properties related to preferred cell stripes objects are listed below:

- `stripes_bbox`
- `stripes_bottom_edge`
- `stripes_cells`
- `stripes_height`
- `stripes_left_edge`
- `stripes_pitch`
- `stripes_repeat`
- `stripes_width`

## Common Specification Modifications

### Stop and Ignore Pins

Stop pins and ignore pins both stop the clock tree specification process from tracing beyond a pin. A stop pin is treated as a sink to be balanced whereas an ignore pin is not balanced. For details about stop and ignore pins, see the [Clock Tree Sink Pin Types](#) section.

### Macro Clock Input Pins

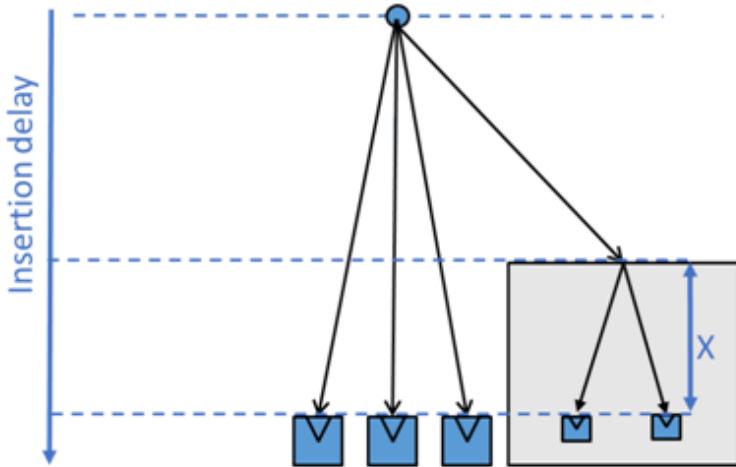
The clock input pins of macros (.lib model) must usually be earlier than other sinks, which means they will have a lesser clock arrival time to take account of the internal clock path inside the macro. If this is represented by a pin specific network latency, [set\\_clock\\_latency](#), command in the SDC timing constraints then the automatically-generated clock tree specification will take this into account. This is discussed further in the [Network Latencies](#) section.

If the clock offset at a macro clock pin is not captured in the timing constraints, then you must add this. For example:

```
set_ccopt_property -pin mem1/CK insertion_delay 1.2ns
```

Note that the property setting is the delay to be assumed inside the macro. Positive numbers will reduce the clock arrival time at the pin, negative numbers will increase it. This is illustrated in the following diagram, where X represents the property setting value.

## Pin Insertion Delay



It is possible to set a pin insertion delay at any clock sink to adjust the skew of the sink relative to other sinks without such a setting. This can be used to implement manual or preCTS useful skew. Note that setting a pin insertion delay on large number of pins is not recommended and may increase clock area and power.

## Architectural Clock Gates

An architectural clock gate is a clock gate typically very early (small insertion delay) in a clock tree that is used to enable and disable entire functions or logical partitions of a design. The flops controlling such a clock gate may also need to be scheduled early to avoid setup slack violations at the clock gate enable input. This can be achieved by adding an additional skew group to balance the flops with the clock gate. For an example of this, refer to the example, [Balancing flops with a clock gate](#), in the [Modifying Skew Groups](#) section. Such additional configuration for architectural clock gates is frequently recommended with CCOpt, and will be essential for timing closure with CCOpt-CTS.

## Restricting CCOpt Skew Scheduling

CCOpt will initially compute the maximum insertion delay over all skew groups. By default, CCOpt skew scheduling is restricted such that the insertion delay of any sink may not exceed some factor multiplied by the initially computed maximum insertion delay. This factor is set by the property, `auto_limit_insertion_delay_factor`, which in the 14.2 version of the software, defaults to 1.5. This permits useful skew scheduling to increase the global maximum insertion delay by up to 50%. Useful skew scheduling is unrestricted by how much it can decrease the insertion delay to a sink.

To change this restriction on late useful skew set the property. For example to lower the restriction to a 20% insertion delay increase:

```
set_ccopt_property auto_limit_insertion_delay_factor 1.2
```

To restrict the skew of a given skew group in CCOpt set the target\_skew property on the skew group and set the constrains property of the skew group to include the keyword 'ccopt'. For syntax details, see the [Defining Skew Groups](#) section. For example, to place a hard limit on the skew of all skew groups to 400ps, irrespective of the impact on timing closure, use the following commands:

```
foreach sg [get_ccopt_skew_groups *] {  
    set_ccopt_property target_skew 400ps -skew_group $sg  
    set_ccopt_property constrains ccopt -skew_group $sg  
}
```

## Method

The recommended method for setting up CCOpt or CCOpt-CTS on a new design is to use the following steps:

1. Configure and create the clock tree specification as per the Quick Start Example and configuration instructions above.
2. Before invoking the [ccopt\\_design](#) command use the CCOpt Clock Tree Debugger in unit delay mode to inspect the clock tree. This will permit examination of the clock tree structure. For more information, see the [Unit Delay](#) section.
3. Invoke only the clustering step of CCOpt or CCOpt-CTS which performs buffering to meet design rule constraints but does not perform skew balancing or timing optimization. Check the maximum insertion delay path looks sensible in the CCOpt Clock Tree Debugger. For designs with narrow channels, many blockages, or complex power domain geometries this is a good time to check for large transition violations caused by floorplan issues. The screenshot, "Cluster Maximum Insertion Delay", below shows the placement view (left) and the CCOpt Clock Tree Debugger view (right) with the maximum insertion path delay highlighted in green.
4. For a CCOpt flow with a simple clock tree, for example a CPU core, switch to using full [ccopt\\_design](#). For a design with a complex clocking architecture consider using trial mode, which will perform clustering and then balancing using virtual delays. The trial balancing can be inspected to look for large skew or insertion delay increases due to conflicting skew group constraints. The design can be timed using [timeDesign -postCTS](#) to check for large timing slack violations, for example, due to incorrect balancing constraints. Virtual delays will appear in timing reports as additional arrival time increments.
5. Run full [ccopt\\_design](#). Inspect the log file for errors and warnings. For CCOpt, a summary table of

timing slack and other metrics at each stage of the `ccopt_design` internal flow is reported.

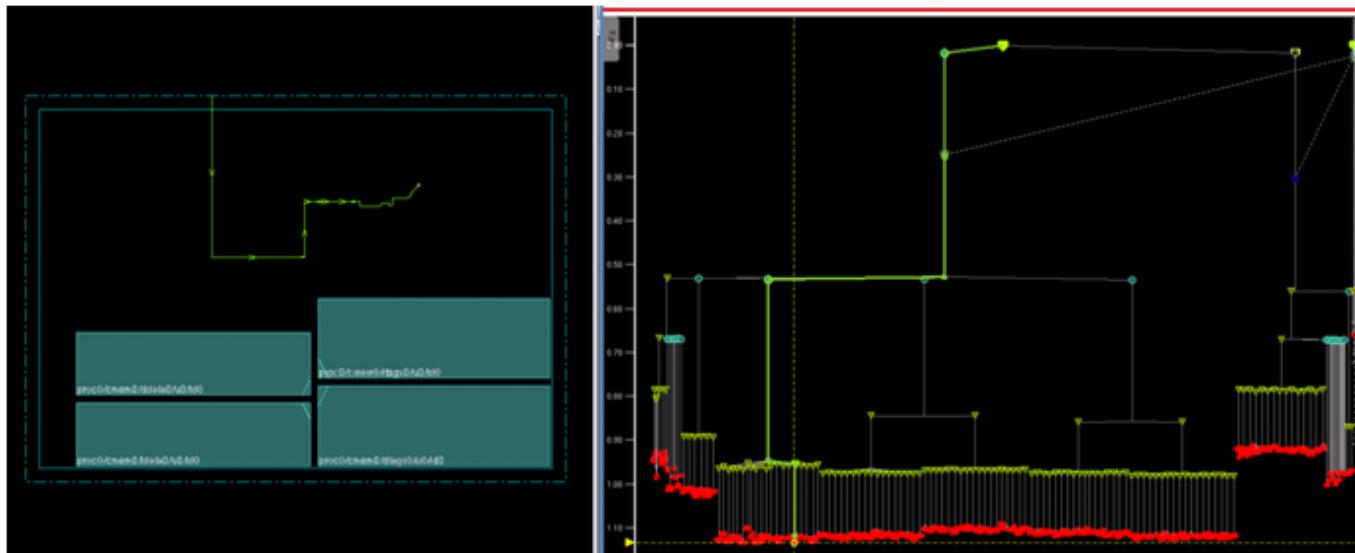
6. For CCOpt, check the worst chain report in the log. Note that there may be multiple worst chain reports in the log. It is recommended to look at the worst chain report after the last occurrence of skew adjustment before any re-clustering steps in the log, this is usually the second last chain report. This report will indicate if useful skew scheduling has hit constraint limits that are limiting optimization. For more information on the worst chain report, see the [Worst Chain](#) section .
7. Report on clock trees and skew groups. For example, it is recommended to check skew group maximum insertion delay and clock tree area even if setup timing slack is closed. For more information, see the [Reporting](#) section.

As mentioned above, CCOpt and CCOpt-CTS can be configured between `cluster`, `trial`, or `full` mode. This is performed using the `cts_opt_type` mode setting, which controls both CCOpt and CCOpt-CTS.

```
set_ccopt_mode -cts_opt_type cluster | trial | full
ccopt_design -cts
```

The default is full mode. The concepts of clustering and trial virtual delay balancing are detailed further in the [Graph-Based CTS](#) section.

### **Cluster Maximum Insertion Delay**



# Concepts and Clock Tree Specification

## Graph-Based CTS

CCOpt-CTS and CCOpt use a graph-based CTS algorithm to perform skew balancing (CCOpt-CTS) or initial seed balancing (CCOpt). The main internal steps in this are as follows:

- **Clustering** – This step groups nearby clock sinks into clusters and buffers clock trees to meet maximum transition, capacitance, and length constraints, such as DRV (Design Rule Violation) constraints. After clustering, the maximum insertion delay is approximately known.
- **Constraints analysis and virtual delay balancing** – Constraints analysis identifies how the balancing constraints (skew and insertion delay constraints) interact and identifies where delay should be added to the clock graph to best meet these constraints. For example, a common scenario is identifying where to add delay to balance test mode clock skew without impacting functional mode clock insertion delay. This happens automatically without any user intervention or need for user-driven sequential steps. Virtual delay balancing is simply the process of annotating clock nodes in the timing graph with additional delay that is added to the propagated clock arrival time to achieve the solution found by constraints analysis. CCOpt uses both clustering and virtual delays to initially balance clocks to obtain initial propagated mode timing and to permit run-time efficient what-if style analysis during useful skew scheduling.
- **Implementation** – This step synthesizes virtual delays using real physical cells. It is followed by a refinement process to account for the difference between virtual delays and those achievable with physical cells. For CCOpt-CTS, this is essentially the last step. For CCOpt, further post-CTS style datapath optimization is performed.

The use of this graph-based CTS approach, combined with the multi-mode clock specification generated by the `create_ccopt_clock_tree_spec` command enables CCOpt-CTS and CCOpt to cope with large complex clock structures, typically, with zero or minimal user intervention.

## Clock Trees and Skew Groups

Clock trees and skew groups are the two key object types used in the CCOpt clock specification. The term object is used here because clock tree and skew group objects can be defined, modified, and deleted using commands. For example, `create_ccopt_clock_tree`, `create_ccopt_skew_group`, `modify_ccopt_skew_group`, and `delete_ccopt_skew_groups`.

Properties can be set per skew group or clock tree instead of globally. For

example, `set_ccopt_property -skew_group name target_skew value`. The `report_ccopt_clock_trees` and `report_ccopt_skew_groups` commands can be used to generate reports on clock trees and skew groups. For more information, see the [Reporting](#) section.

## Clock Trees

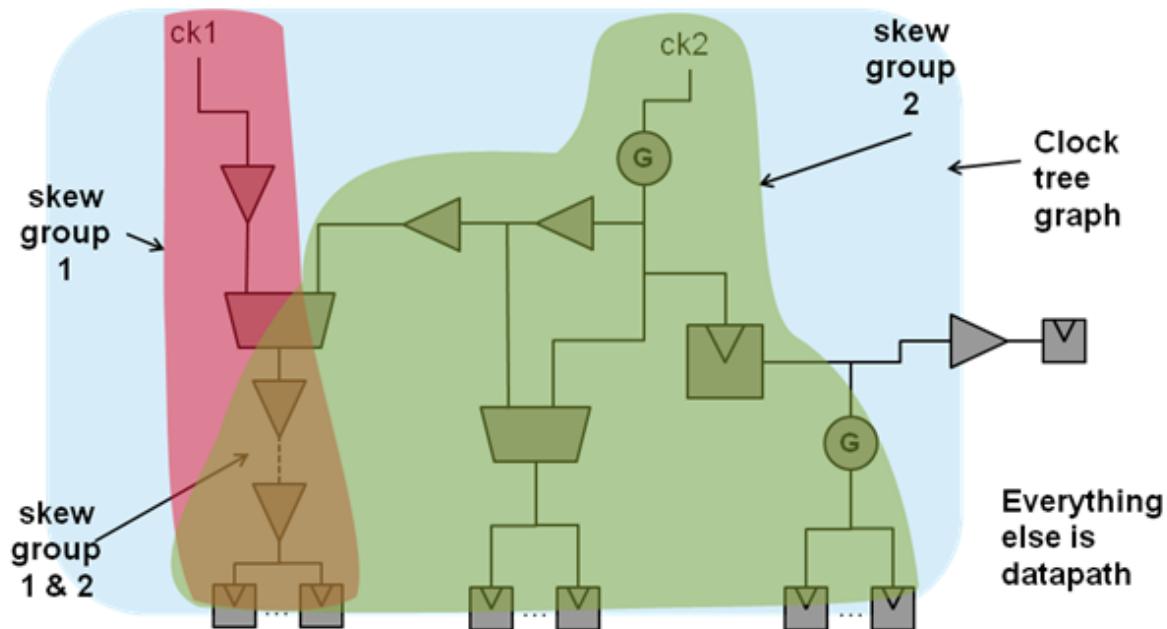
- The union of all clock trees specifies the subset of the circuit graph that CTS will buffer. The circuit subset covered by clock tree definitions is best referred to as a clock tree graph since clock trees may interact, for example via clock logic cells. The clock tree graph is a single physical graph even in a multi-mode timing environment.
- Maximum transition times, route types and other physical properties are associated with the clock tree graph or with individual trees in the clock tree graph.
- In all but rare exceptional circumstances, the clock tree definitions created by `create_ccopt_clock_tree_spec` do not require user modification.

## Skew Groups

- A skew group represents a balancing constraint and is the CTS equivalent of an SDC clock. The automatically generated clock tree specification will create one skew group per SDC clock per mode.
- Each skew group has one or more sources and a number of sinks. Among other properties, a skew target and insertion delay target can be set per skew group. Any pin in the clock tree graph can be a skew group source or sink and pins can be designated a skew group specific ignore pin such that the specific skew group does not propagate beyond the pin.
- CCOpt-CTS global skew balancing aims to achieve an equal delay, subject to the skew target, from all sources to all sinks within each skew group. CCOpt virtually balances skew groups to zero skew to determine initial clock tree timing with propagated clocks before optimization starts.
- A skew group can be viewed as a subset of the clock tree graph superimposed on top of the clock tree graph. Skew groups can overlap, share sources, and/or sinks.
- In complex cases or with CCOpt-CTS where the SDC timing constraints do not fully capture the balancing requirements, user adjustment to the skew group configuration may be required and/or additional skew groups can be defined.

The diagram below illustrates the relationship between the clock tree graph and skew groups. Note the path to the data input of a flip-flop at the right hand side, the clock tree graph is ‘pruned back’ to exclude this path, the input to the right most buffer will be an ignore pin – clock pin types are discussed later.

## Clock Tree Graph and Skew Groups



## Automatic Clock Tree Specification Creation

The `create_ccopt_clock_tree_spec` command is used to automatically create clock tree and skew group definitions from analysis of the active timing constraints, typically those loaded from SDC. It is important to note that the `create_ccopt_clock_tree_spec` command does not perform a text based or any other kind of direct translation from SDC constraint commands but performs an analysis of the clock definitions and clock propagation in the timing graph. This means that whilst there is often a close correspondence between SDC commands and the clock tree specification in some areas there will not be an exact one-to-one mapping.

- **Clock trees** – The automatically generated specification will have clock trees defined for primary clocks at input ports, generated clocks at sequential flip-flop outputs, and will have ignore pins at the edge of the clock tree graph. Typically, the user can ignore the precise clock tree definitions.
- **Skew groups** – The automatically generated specification will have one skew group defined per timing clock in each constraint mode. Skew groups corresponding to generated timing clocks are set to be non-constraining, which means they are ignored by CTS but included for reporting purposes. The sinks of generated timing clocks are balanced with sinks of the appropriate master clock. This is arranged via a single skew group corresponding to the master clock that propagates through the generator. This is illustrated as part of the "Single Mode Example" below.

Skew groups are named according to the timing clock and constraint mode they correspond to. For

example, the timing clock with name `clk1500m` in the `func` constraint mode would be given name `clk1500m/func`.

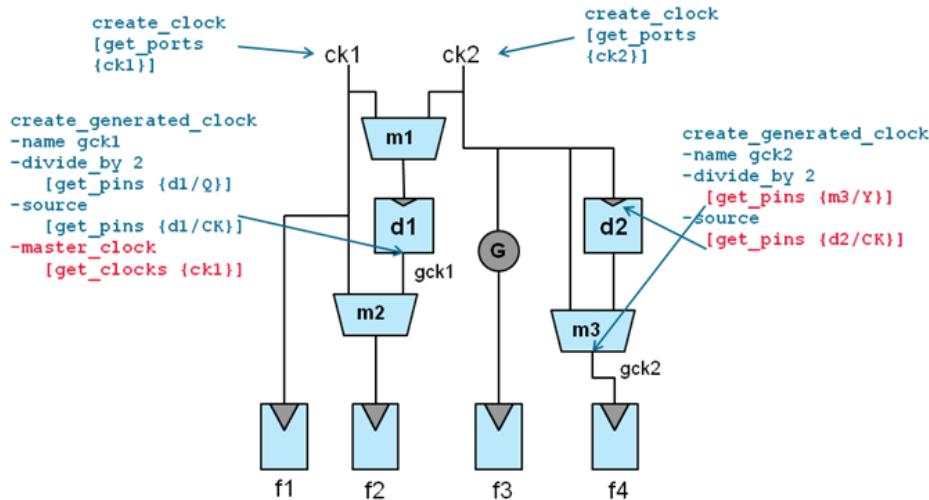
**Note:** In some CCOpt property names, the process of clock specification generation from SDC constraints is sometimes referred to as ‘extraction’ for historic reasons. This is not to be confused with parasitic extraction.

## Single Mode Example

The diagram below shows a single constraint mode example with two clocks, some multiplexers, and two clock dividers. The SDC clock definitions are illustrated.

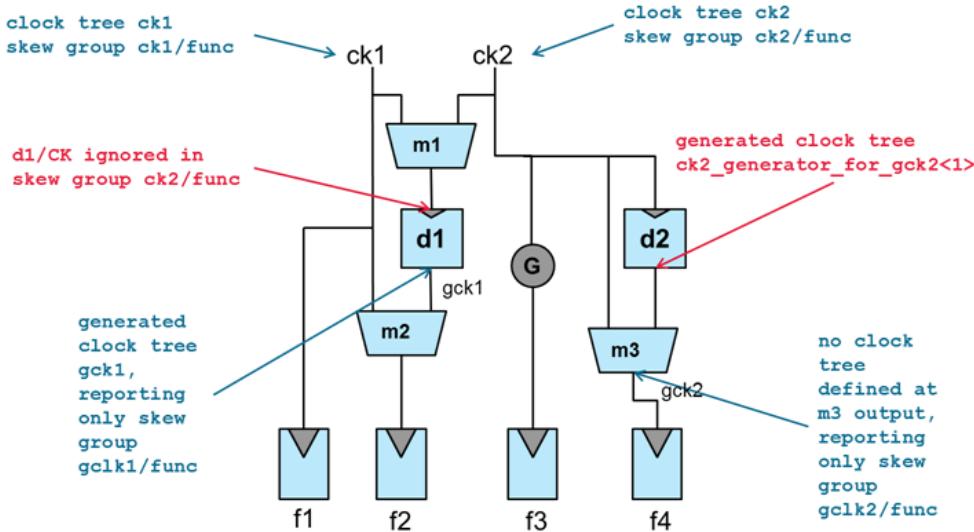
Note the precise definitions of the generated clocks carefully.

### Single Mode Example – SDC Clock Definitions



On the left side, the generated clock `gck1` refers to master `ck1` such that `ck2` does not propagate to `f1` or `f2`. On the right side, the definition of `gck2` is such that the path from `d2/CK` to `m3/Y` is considered part of the clock generator circuit. Both these points have implications for the resulting clock tree specification output that is annotated in the diagram below.

## Single Mode Example – Clock Tree Specification



In the generated specification, a clock tree is defined at each of the primary inputs `ck1` and `ck2`.

On the left side, a generated clock tree is defined at the output of divider `d1` to distinguish `d1` as a sequential element in the clock graph. Without this generated clock tree definition CTS would treat `d1` as a regular sink. Additionally, at the output of divider `d1` a skew group `gck1/func` is defined, but note that this skew group is non-constraining so does not influence CTS. It is present purely for reporting purposes. Sinks `f1` and `f2` are balanced together by skew group `ck1/func`. Skew group `ck2/func` is ignored at the input to `d1`, this corresponds to the `master_clock` specification in the SDC.

On the right side, no generated clock tree is defined at the output of multiplexer `m3`, since `m3` is a combinational cell. However, a non-constraining skew group is defined at the output of multiplexer `m3` for reporting purposes. So that CTS does not treat divider `d2` as a regular clock sink and so that the path from `d2` to `m3` is included in the clock tree graph, a generated clock tree is defined at the output of `d2`.

Key lines from the output of `create_ccopt_clock_tree_spec -filename ccopt.spec` for the example are given below.

## Single Mode Example – create\_ccopt\_clock\_tree\_spec Output

```

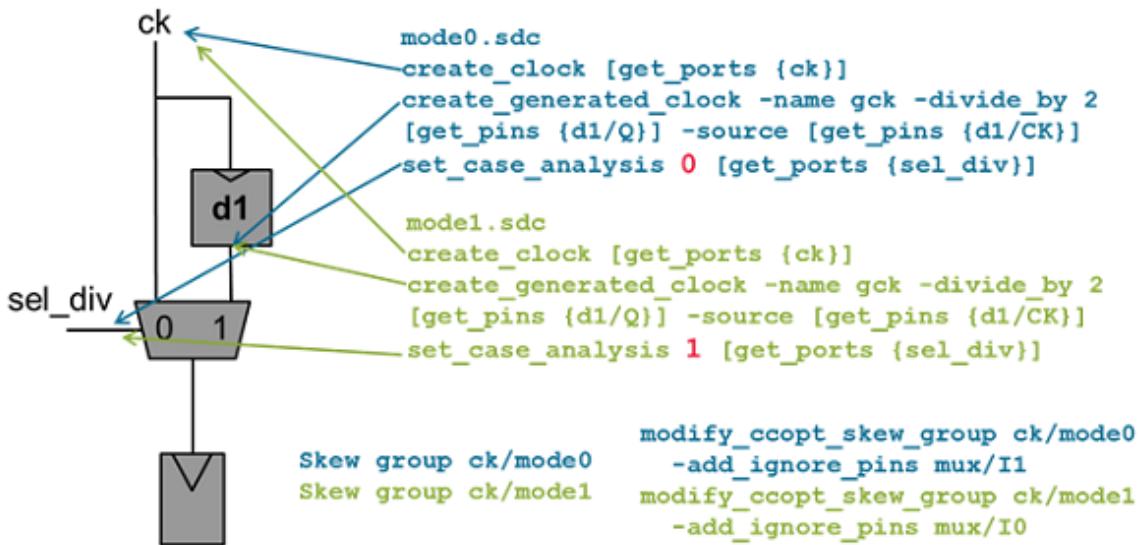
create_ccopt_clock_tree -name ck2 -source ck2 -no_skew_group
create_ccopt_generated_clock_tree -name ck2_generator_for_ck2<1> -source d2/Q -generated_by
d2/CK
create_ccopt_clock_tree -name ck1 -source ck1 -no_skew_group
create_ccopt_generated_clock_tree -name gck1 -source d1/Q -generated_by d1/CK
create_ccopt_skew_group -name ck1/func -sources ck1 -auto_sinks
create_ccopt_skew_group -name ck2/func -sources ck2 -auto_sinks
modify_ccopt_skew_group -skew_group ck2/func -add_ignore_pins d1/CK
create_ccopt_skew_group -name gck1/func -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck1/func none
create_ccopt_skew_group -name gck2/func -sources m3/Y -auto_sinks
set_ccopt_property constrains -skew_group gck2/func none

```

## Multi-Mode Example

The diagram below shows a simple multi-mode example annotated with SDC constraints and skew group information.

### Multi-Mode Example



The resulting specification contains the following:

- Two clock trees, ck and gck. The clock tree definitions tell CTS which parts of the circuit are included in the clock tree graph and are not mode specific.
- Two skew groups, ck/mode0 and ck/model. The skew groups tell CTS how to perform balancing.

- Each skew group has an ignore pin defined at the appropriate multiplexer input. This represents the fact that there is no need to balance the direct clock path with the divided clock path as the paths are never active in the same mode at the same time.

Key commands from the specification are listed below. Some details have been omitted for clarity.

## Multi-Mode Example – `create_ccopt_clock_tree_spec` Output

```
create_ccopt_clock_tree -name ck -source ck -no_skew_group
create_ccopt_generated_clock_tree -name gck -source d1/Q -generated_by d1/CK
create_ccopt_skew_group -name ck	mode0 -sources ck -auto_sinks
create_ccopt_skew_group -name ck	mode1 -sources ck -auto_sinks
modify_ccopt_skew_group -skew_group ck	mode0 -add_ignore_pins mux/I1
modify_ccopt_skew_group -skew_group ck	mode1 -add_ignore_pins mux/I0
create_ccopt_skew_group -name gck	mode0 -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck	mode0 none
create_ccopt_skew_group -name gck	mode1 -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck	mode1 none
```

## SDC Transition Targets

The `create_ccopt_clock_tree_spec` command will translate `set_max_transition` constraints. For example, consider the SDC constraint “`set_max_transition 0.123 [get_clocks {ck1}]`”.

The automatically generated specification will contain the following line:

```
set_ccopt_property target_max_trans_sdc -clock_tree ck1 0.123
```

CCOpt-CTS and CCOpt will look at the `target_max_trans` property. If this is set to the default value of `auto`, then the `target_max_trans_sdc` will be inspected. If `target_max_trans_sdc` is not set, then an automatic default will be computed.

## Network Latencies

The `create_ccopt_clock_tree_spec` command will translate clock network latency settings to an insertion delay target on the corresponding skew group. For example, consider the functional mode SDC constraint, “`set_clock_latency 1.456 [get_clocks {ck1}]`”. The automatically generated specification will contain the following line:

```
set_ccopt_property target_insertion_delay -skew_group ck1/func 1.456
```

Similarly, pin network latency settings are translated to the `insertion_delay` property of a pin. This property is often referred to as a pin insertion delay. A pin insertion delay represents the delay ‘underneath’ a clock sink. For example, for a macro clock input pin, the pin insertion delay would represent the internal clock path delay inside the macro. Continuing the above example, add the constraint “`set_clock_latency 0.234 [get_pins {mem1/CK}]`”. The automatically generated specification will additionally contain the following line:

```
set_ccopt_property insertion_delay -pin mem1/CK 1.222
```

The property setting indicates that the delay internal to the macro clock input `mem1/CK` is 1.222. The value 1.222 is computed as the difference between the clock latency of 1.456 and the pin latency of 0.234. Note that SDC pin-specific latencies override clock latencies, which means they are not added together.

## Clock Tree Convergence

In some circumstances, the clock tree graph undesirably propagates into datapath and includes what should be datapath as part of the clock tree graph. For example, this can happen due to missing `set_case_analysis` or other incorrect SDC constraints. Including significant datapath logic as part of the clock tree graph can result in excessive CCOpt or CCOpt-CTS run time due to large numbers of paths existing between a skew group source and sink due to multiple levels of re-convergent logic. Additionally, such paths would not be optimized by datapath optimization.

To help detect cases where run time would be adversely affected, the automatically generated clock tree specification includes an invocation of the `check_ccopt_clock_tree_convergence` command. This command traces the number of paths to every sink and issues a warning if the number of clock paths to any sink is greater than a default threshold of 100 paths. The `report_ccopt_clock_tree_convergence` command can be used to report sinks with large numbers of clock paths.

To remedy this situation, either correct the SDC constraints, for example by adding `set_case_analysis`, `set_clock_sense -stop_propagation` or other suitable commands, or use a clock tree stop, ignore, or exclude pin as appropriate. These pins are described in the next section.

## Clock Tree Sink Pin Types

Clock tree sink pin types can be manually overridden before invoking the `create_ccopt_clock_tree_spec` command with the following property setting:

```
set_ccopt_property -pin pin_name sink_type ignore | stop | exclude
```

## Ignore pin (ignore)

An ignore pin is considered as a part of the clock tree graph. CTS will perform DRV buffering up to the pin, but the pin will not be considered as a sink in any skew group, which means the latency to an ignore pin is not important. Tracing through and beyond the pin will be disabled. Sometimes such a pin is referred to as a clock tree ignore pin. An alternative strategy to deploying an ignore pin would be to use the SDC constraint, `set_clock_sense -stop_propagation`. This may be preferable since it would keep the timing model in synchronization with the CTS configuration.

## Stop pin (stop)

A stop pin is considered as a part of the clock tree graph. CTS will perform DRV buffering up to the pin and by default the pin will be considered a sink to be balanced in any skew group that reaches the stop pin. Tracing through and beyond the pin will be disabled.

## Exclude pin (exclude)

An exclude pin is a pin that is not part of the clock tree graph but might still be connected to a clock net anyway if the same net has other clock fanout. Specifically, the clock tree graph must not extend beyond an exclude pin but it can be pruned back from an exclude pin. The `create_ccopt_clock_tree_spec` command will prune back from an exclude pin and, if possible, specify an ignore pin earlier in the fanin cone. The [Shared Clock and Data Concerns](#) section discusses how to add buffers to disconnect exclude pins from any clock tree nets they may be connected to. This can be important where clock and datapath overlap.

**Note:** In addition to the above pin types, it is possible to make any pin that is within the clock tree graph a skew group specific ignore or sink pin. This is discussed in the subsequent sections.

## Manual Setup and Adjustment of the Clock Specification

Following are some important recommendations for setting up and adjusting the clock tree specification:

- It is recommended that the `create_ccopt_clock_tree_spec` command is used to create the clock tree specification.
- It is not recommended to edit the specification file generated by the `create_ccopt_clock_tree_spec -file filename` command. A more stable flow is obtainable

either by adjusting the SDC, configuring CCOpt properties, setting clock tree sink pin types before generating the specification, or making skew group adjustments after loading the specification.

- Consider making adjustments to the SDC timing constraints instead of the CTS specification, if applicable. This will ensure that timing analysis uses a clock propagation model consistent with the CTS configuration. For example, setting a clock logic instance input pin to be a clock tree ignore pin will stop CTS tracing through the pin, but will not stop `report_timing` from propagating a clock through the pin. The `create_ccopt_clock_tree_spec` command has been engineered to create a clock tree specification consistent with the active timing constraints.

The table below shows commonly used commands for manipulating clock trees and skew groups:

| Command Name                                                                    | Usage                                                                                                                                                |
|---------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>create_ccopt_clock_tree</code>                                            | Adds a new clock tree definition into the in memory clock tree specification.                                                                        |
| <code>delete_ccopt_clock_trees</code>                                           | Removes a clock tree definition from the in memory clock tree specification.                                                                         |
| <code>create_ccopt_skew_group</code>                                            | Adds a new skew group definition into the in memory clock tree specification.                                                                        |
| <code>delete_ccopt_skew_groups</code>                                           | Removes a skew group definition from the in memory clock tree specification.                                                                         |
| <code>modify_ccopt_skew_group</code>                                            | Permits adjustment to sinks and ignore pins of a skew group.                                                                                         |
| <code>set_ccopt_property -skew_group skew_group_name property_name value</code> | Adjusts skew group specific properties. The most commonly adjusted properties are <code>target_skew</code> and <code>target_insertion_delay</code> . |

**Note:** The above commands do not modify the design or perform any CTS but manipulate the in-memory clock tree specification.

## Defining Clock Trees

Clock trees are defined using the `create_ccopt_clock_tree` and `create_ccopt_generated_clock_tree` commands. For example:

```
create_ccopt_clock_tree -name ck -source ck -no_skew_group
create_ccopt_generated_clock_tree -name gck -source d1/Q -generated_by d1/CK
```

The optional `-name` parameter can be used to specify the name of the clock tree. Alternatively, the source pin name will be used as the clock tree name. The mandatory `-source` parameter specifies the clock tree root pin from which clock tree tracing will be performed. The `-no_skew_group` parameter disables the automatic creation of a corresponding skew group, otherwise a skew group with the same name as the clock tree is automatically created. In addition, the definition of a generated clock tree requires the `-generated_by` parameter to specify the input side of the clock generator, which is typically the clock input pin of a divider flip-flop.

When a clock tree is defined, CCOpt traces the circuit connectivity from the specified source pin, adding the nets and cell instances it encounters to the clock tree graph. Tracing continues until encountering a clock pin (such as a flip-flop, latch, or macro input), a user-defined stop, ignore, or exclude pin. A generated clock tree definition must normally be used at the output of a sequential cell to continue tracing.

## Defining Skew Groups

Skew groups are defined using the `create_ccopt_skew_group` command. The complete syntax of this command is detailed below:

```
create_ccopt_skew_group
[-help]
-name skew_group_name
[-constraints {icts | ccopt_initial | ccopt}]
[-sinks pins | -shared_sinks pins | -exclusive_sinks pins | -auto_sinks | -
filtered_auto_sinks pins | -balance_skew_groups skew_group_names]
[-sources pins]
[-rank rank]
[-target_insertion_delay value]
[-target_skew value]
[-from_clock clock_name]
[-from_constraint_modes constraint_mode_names]
[-from_delay_corners delay_corner_names]
```

The descriptions of the above parameters are in the table below:

| Parameter                                 | Description |
|-------------------------------------------|-------------|
| Name                                      |             |
| <code>-name <i>skew_group_name</i></code> |             |

|                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                  | <p>Specifies the mandatory name for the newly created skew group. The <code>create_ccopt_clock_tree_spec</code> command will use a name in the following format:</p> <pre>clock_name/constraint_mode_name</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|                                                  | <pre>-constraints {cts   ccopt_initial   ccopt}</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|                                                  | <p>Optionally specifies the parts of CTS that the skew group should constrain. A list of keywords is specified: CCOpt-CTS/CCOpt initial balancing (<code>cts/ccopt_initial</code>) or full CCOpt flow (<code>ccopt</code>). Note that <code>cts</code> and <code>ccopt_initial</code> are synonymous. The default is <code>ccopt_initial</code> such that the skew group constrains CCOpt-CTS and influences the initial virtual delay balancing step of CCOpt.</p> <p><b>Note:</b> The initial virtual delay balancing step of CCOpt ignores skew targets. The <code>ccopt</code> setting can be used to limit the range of useful skew performed by CCOpt.</p> |
| <pre>-sinks pins</pre>                           | <p>Specifies sink pins to be balanced within this skew group. If the <code>-rank</code> parameter is not used, then this is equivalent to <code>-shared_sinks</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <pre>-shared_sinks pins</pre>                    | <p>Specifies sink pins to be balanced with this skew group. Sinks will be balanced in any existing skew group that they are sinks in, and will additionally be balanced in the newly created skew group. The newly created skew group is given a rank of 0.</p>                                                                                                                                                                                                                                                                                                                                                                                                  |
| <pre>-exclusive_sinks pins</pre>                 | <p>Specifies sink pins to be balanced with this skew group. The newly created skew group will have a rank number 1 above all existing skew groups. This means that sinks will only be balanced in the newly created skew group and will be ignored in existing skew groups, which by definition will have a lower rank.</p>                                                                                                                                                                                                                                                                                                                                      |
| <pre>-auto_sinks</pre>                           | <p>Automatically traces the clock tree graph from the source pins looking for sinks, for example flip-flop clock input pins.</p> <p><b>Note:</b> If this parameter is not specified and an explicit list of sinks is not specified then the skew group will not have any sinks.</p>                                                                                                                                                                                                                                                                                                                                                                              |
| <pre>-filtered_auto_sinks pins</pre>             | <p>This performs the same tracing as <code>-auto_sinks</code>, but will only select sinks that are also in the specified pins.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <pre>-balance_skew_groups skew_group_names</pre> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                               | Specifies that the newly created skew group should be a merging of existing skew groups. The newly created skew group will have the union of source pins from the existing skew groups and sink pins from the existing skew groups. The effect is as if to balance existing skew groups together as a single skew group. Typically, this parameter is used to balance two skew groups that correspond to different clock trees.                                                                                                                                                                                      |
| -sources pins                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| -rank rank                    | Specifies the source pins for the skew group.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| -target_insertion_delay value | Specifies the exclusive sinks rank for the skew group. This is discussed in a subsequent section.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| -target_skew value            | Specifies a target insertion delay for this skew group. CCOpt-CTS attempts to keep the latency of all sinks within half the target skew earlier or later than this delay. CCOpt will use this target insertion delay during initial virtual delay balancing. The default auto setting permits an increase of 5% from the initial clustering insertion delay to improve clock power. The special value of min can be used to aim for minimum insertion delay even at the expense of some clock power. Alternatively, set the target_insertion_delay property for the skew group using the set_ccopt_property command. |
| -from_*                       | Specifies a target skew for skew between sinks of this skew group overriding the global skew target. CCOpt-CTS will respect this setting. CCOpt ignores it unless the -constraints setting includes ccopt. Alternatively, set the target_skew property for the skew group using the set_ccopt_property command.                                                                                                                                                                                                                                                                                                      |

**Note:** The parameters taking a list of pins operate with either a plain TCL list of hierarchical pin names or with a collection of pins obtained from the [get\\_pins](#) command.

## Skew Group Rank

The rank of a skew group determines whether a sink pin is an active sink in that skew group or not. A pin is only an active sink in the skew group(s) with the highest rank out of all the skew groups to which the pin belongs. An active sink is a pin that will be balanced against other active sinks in the same skew group.

For example, consider the following sequence of commands:

```
create_ccopt_skew_group -name SG1 -sources get_pins top -shared_sinks [get_pins */D]
create_ccopt_skew_group -name SG2 -sources get_pins top -exclusive_sinks [get_pins *XYZ*/D]
create_ccopt_skew_group -name SG3 -sources get_pins top -exclusive_sinks [get_pins
*XYZ_01*/D]
```

After running the first command, a single skew group SG1 is created. Shared sinks are specified so this skew group has a rank of zero. All the "D" pins in the design are members of this skew group. In addition, all the "D" pins are active sinks in skew group SG1. This is because SG1 is the highest ranked skew group so far, even though it has a rank of zero.

The second command defines skew group SG2. Exclusive sinks are specified so this skew group has a rank of 1, which is one higher than the current highest rank of 0. Pins that match the pattern “\*XYZ\*/D” are now members of both SG1 and SG2. However, they are only active sinks in SG2, which is the highest ranked parent skew group so far. Pins that do not match this pattern remain active sinks in SG1.

The third command defines another exclusive skew group SG3. Exclusive sinks are specified so this skew group has a rank of 2, which is one higher than the current highest rank of 1. Pins that match the pattern “\*XYZ\_01\*/D” are now members of skew groups SG1, SG2, and SG3. However such pins are only active sinks in SG3, which is the highest ranked parent skew group. Pins that matched the pattern “\*XYZ\*/D” but not “\*XYZ\_01\*/D” are members of both SG1 and SG2 but only active sinks of SG2. Sinks that do not match the pattern \*XYZ\*/D are active sinks in SG1.

The rank of a skew group can be accessed via the `exclusive_sinks_rank` property.

## Finding Active Skew Group Sinks

Use the following command to find all the sink members of a skew group:

```
get_ccopt_property -skew_group name sinks
```

Use the following command to find all the active sink members of a skew group:

```
get_ccopt_property -skew_group name sinks_active
```

Use the following command to find all the skew groups for which a pin is a sink member:

```
get_ccopt_property -pin name skew_groups_sink
```

Use the following command to find all the skew groups for which the pin is an active sink:

```
get_ccopt_property -pin name skew_groups_active_sink
```

Use the following command to find all the skew groups which are active at a pin, either passing through the pin or for which the pin is an active sink:

```
get_ccopt_property -pin name skew_groups_active
```

**Note:** In debugging CCOpt-CTS skew or CCOpt initial balancing, the ‘active’ properties above should be used, since these reflect the constraints CTS will respect. For example, if a pin is configured as a sink of skew group but the skew group does not propagate to the pin due to a lack of connectivity, the pin will not be an active sink of the skew group. After defining skew groups or modifying existing skew groups it is recommended to invoke the [report\\_ccopt\\_skew\\_groups](#) or [ccopt\\_design](#) command to ensure that the CTS timer is updated before checking the active sinks properties.

## Modifying Skew Groups

The [modify\\_ccopt\\_skew\\_group](#) command is used to make changes to the sink and ignore pins associated with a skew group. The syntax of the command is provided below.

```
modify_ccopt_skew_group
[-help]
-skew_group skew_group_name
[-add_sinks pins | -remove_sinks pins]
[-add_ignore_pins pins | -remove_ignore_pins pins]
```

The `-add_sinks` and `-remove_sinks` parameters are used to add and remove sinks. The `-add_ignore_pins` and `-remove_ignore_pins` parameters are used to add and remove ignore pins, and are discussed below.

The [set\\_ccopt\\_property](#) command can be used to modify properties of a skew group, including the `target_insertion_delay`, `target_skew`, and `constrains` properties.

## Skew Group Ignore Pins

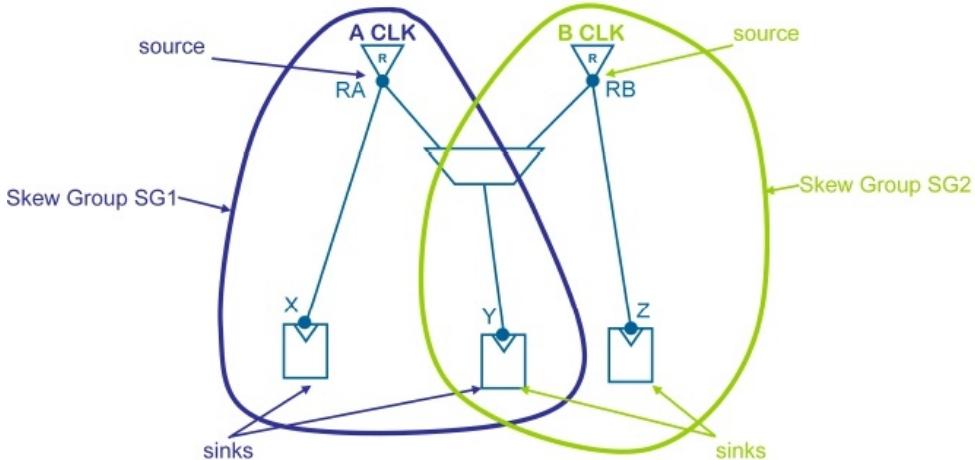
Specifying a pin as an ignore pin of a skew group stops CTS from considering the latency to that pin in that specific skew group, and stops that specific skew group propagating through and beyond that pin. Other skew groups at the pin are not affected. Skew group ignore pins are always applicable regardless of the skew group rank.

For example, if a leaf flip-flop clock pin is specified as a skew group ignore pin, CTS will not balance that flip-flop with other sinks for the same skew group. Balancing of other skew groups, possibly involving the same pins, would not be affected.

If a non-leaf pin is specified as a skew group ignore pin, for example a multiplexer input, CTS will ignore both the latency to and through that multiplexer input in the given skew group. Other skew groups passing through the same multiplexer input would not be affected. In such an example, any flip-flops in the fanout of the multiplexer would cease to be active sinks of the skew group.

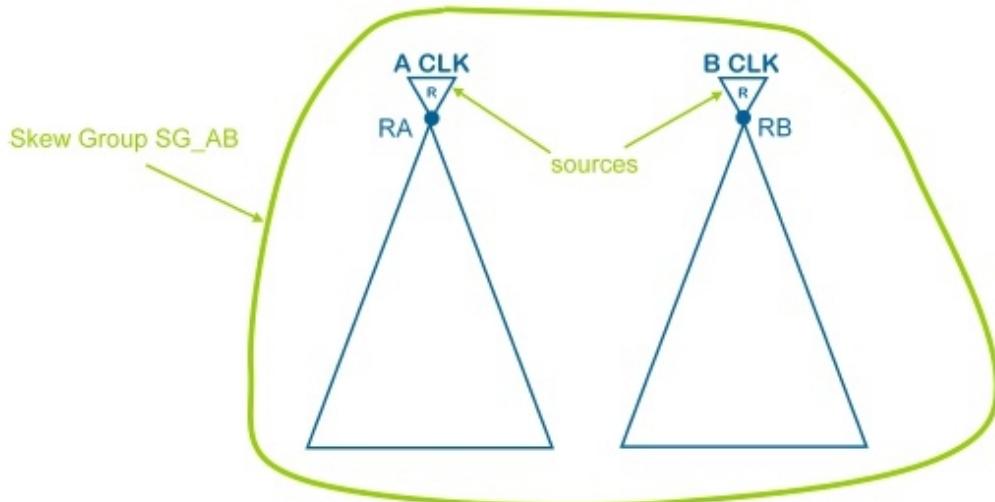
## Example – Overlapping Skew Groups

The diagram below illustrates an example with two clock trees, A and B, with corresponding skew groups, SG1 and SG2. The sink Y is an active sink of both skew group SG1 and skew group SG2. Sink X and Y are balanced together and sink Y and Z are balanced together. The insertion delay difference between X and Z is not constrained. Constraint analysis during CTS will identify the most efficient place to put this delay, which may be at the multiplexer inputs. For example, to add delay to balance the B-Y path with the B-Z path, delay can be added at the right hand multiplexer input without increasing the insertion delay of the A-Y path.



## Example – Balancing Independent Clock Trees

The next example below again has two clock trees, A and B, but with a single skew group, SG\_AB. The skew group has two sources and all the sinks of clock tree A and clock tree B. CTS will balance all paths from A to the sinks and all paths from B to the sinks together.



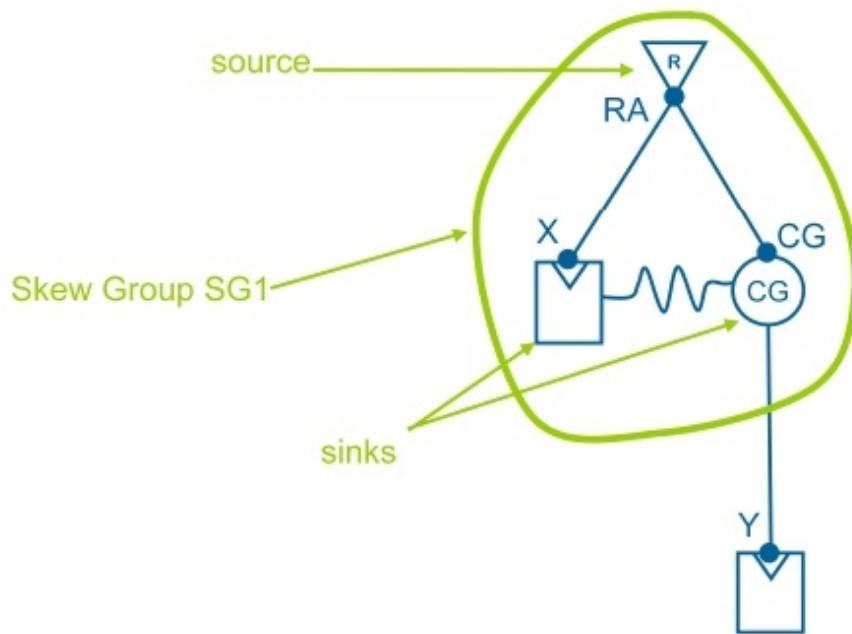
A variant of the above example would also have skew group, SG\_A and skew group, SG\_B corresponding to each of the two clock trees, which would be the default behavior of automatic clock tree specification. The user could then use `create_ccopt_skew_group -name SG_AB -balance_skew_group {SG_A SG_B}` to create a combined skew group.

## Example – Balancing Flops with a Clock Gate

When using CC Opt-CTS it may be necessary to reduce the clock insertion delay to sinks at the source of paths to a clock gate to avoid a setup violation at the clock gate enable input. In the example below, this is done by creating an additional skew group, SG1, to balance the flip-flop pin X with the clock gate pin CG as follows:

```
create_ccopt_skew_group -name SG1 -sources RA -exclusive_sinks {X CG}
```

The pins, X and CG, are made exclusive sinks so that X is no longer an active sink in other existing skew groups.



With CCOpt, rather than CCOpt-CTS, an additional user skew group would not normally be required to do this as useful skew scheduling will automatically adjust the insertion delay of X and CG to optimize the setup slack at the clock gate enable input.

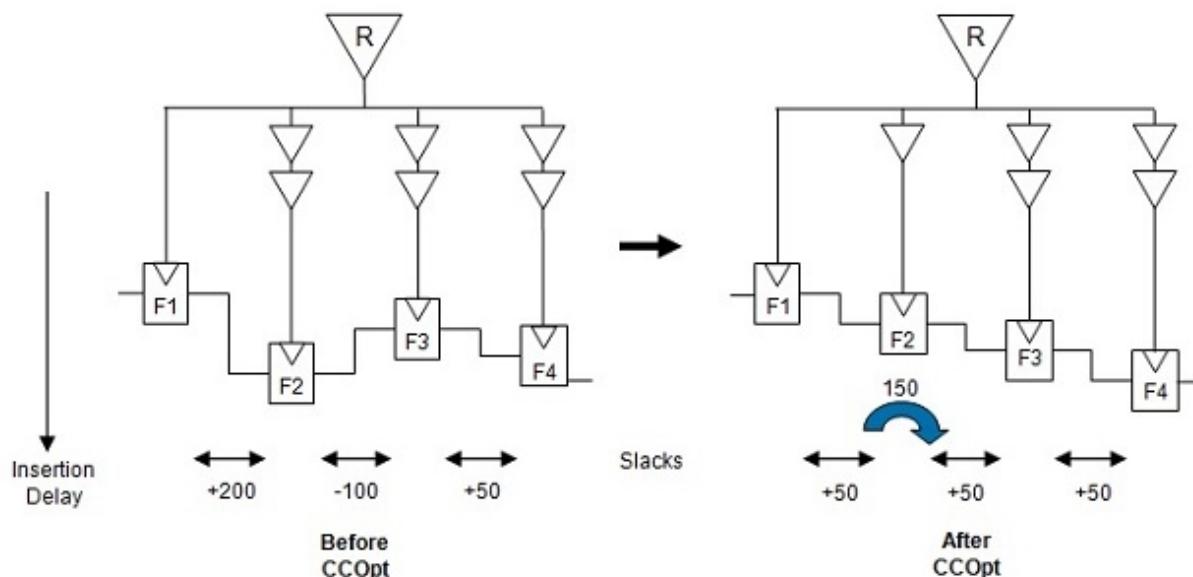
## Deleting the Clock Tree Specification

The `delete_ccopt_clock_tree_spec` command can be used to remove all skew groups, clock trees, and associated data. However, this command does not reset property settings on pins, instances and other database entities. The `reset_ccopt_config` command can be used to remove both the clock tree specification and all CCOpt property settings.

## Chains

CCOpt uses useful skew to adjust clock delays, therefore, moving slack between datapath stages. The limit of WNS optimization is not determined by a single flop-to-flop datapath stage but a chain of such paths. At each flop slack can be shifted from the capture or launch side as illustrated below.

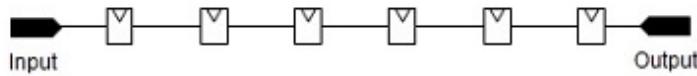
## Moving Slack between Datapath Stages



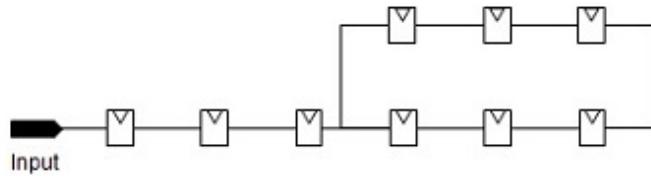
In the example above, CCOpt moves 150ps of slack from the  $F_1 \rightarrow F_2$  path and moves it to the  $F_2 \rightarrow F_3$  path to address the negative slack of -100ps. This is done by reducing the delay on the  $F_2$  clock path, illustrated by the removal of a buffer in the above simplified diagram. However, the ability to move slack between datapath stages is not unlimited. It must stop when the chain of paths either loops back on itself or reaches an input or output port. This gives rise to different types of chains:

- Input-to-output chain – a chain of flops starting at an input pin and ending at an output pin
- Input-to-loop chain – a chain of flops starting at an input pin and ending at a looped path
- Loop-to-output chain – a chain of flops starting at a looped path and ending at an output pin
- Looping chain – a chain of flops starting and ending at a looped path

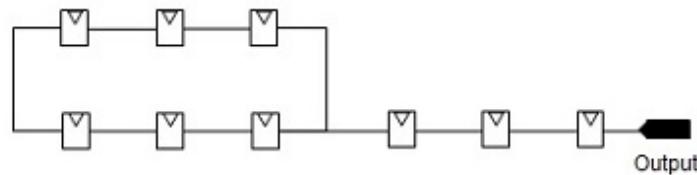
The different types of chains are shown below.



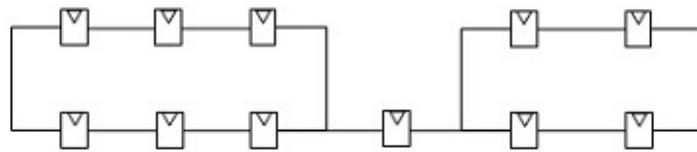
IO Chain



Input-to-loop chain



Loop-to-output chain

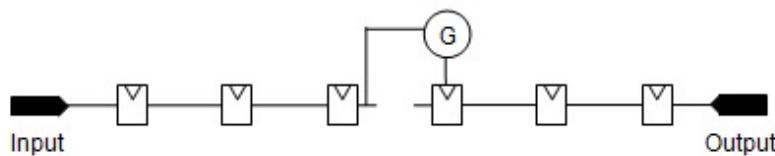


Looping chain

A variant of the input and output chains is a chain containing a flop which either does not launch or does not capture paths, for example if such paths are subject to `set_false_path` exception.

Chains can contain clock gates as illustrated below. CCOpt can adjust the clock insertion delay both to the clock gate and the flops during useful skew scheduling. Adjusting the clock insertion delay to a clock gate may impact the insertion delay to the gated flops, which in turn impacts the slack of timing paths launched by those flops.

#### Clock Gate in a Chain

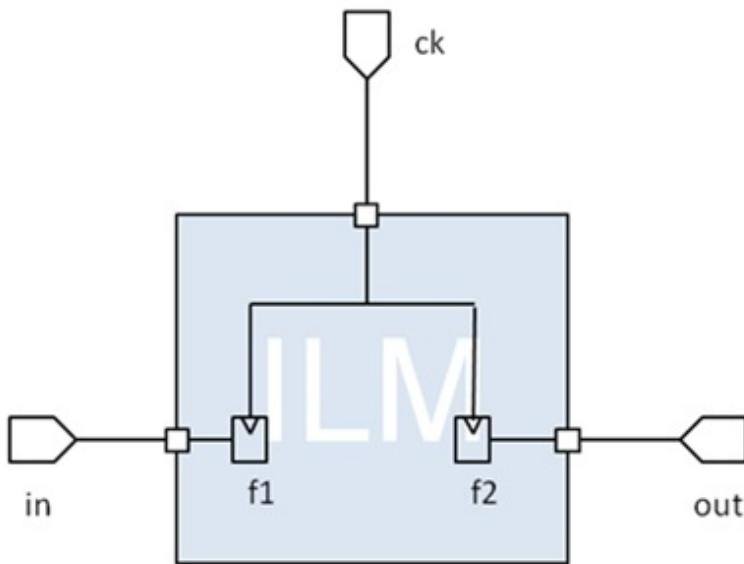


The design worst chain is the chain constructed by taking the global WNS path and expanding the chain around this path such that each path within the chain is a local WNS path. The worst chain is reported in the log during CCOpt design, and the format of this chain report is discussed further in the "Worst Chain" section.

## Disjoint Chains

The worst chain may pass through an ILM partition or “.lib” macro. The example below illustrates an ILM partition in which a single clock input clocks both an input register and an output register inside the ILM. The example contains two timing paths, in-to-f1 and f2-to-out. CCOpt cannot independently adjust the insertion delay of flop f1 and flop f2 because the ILM contents are read-only.

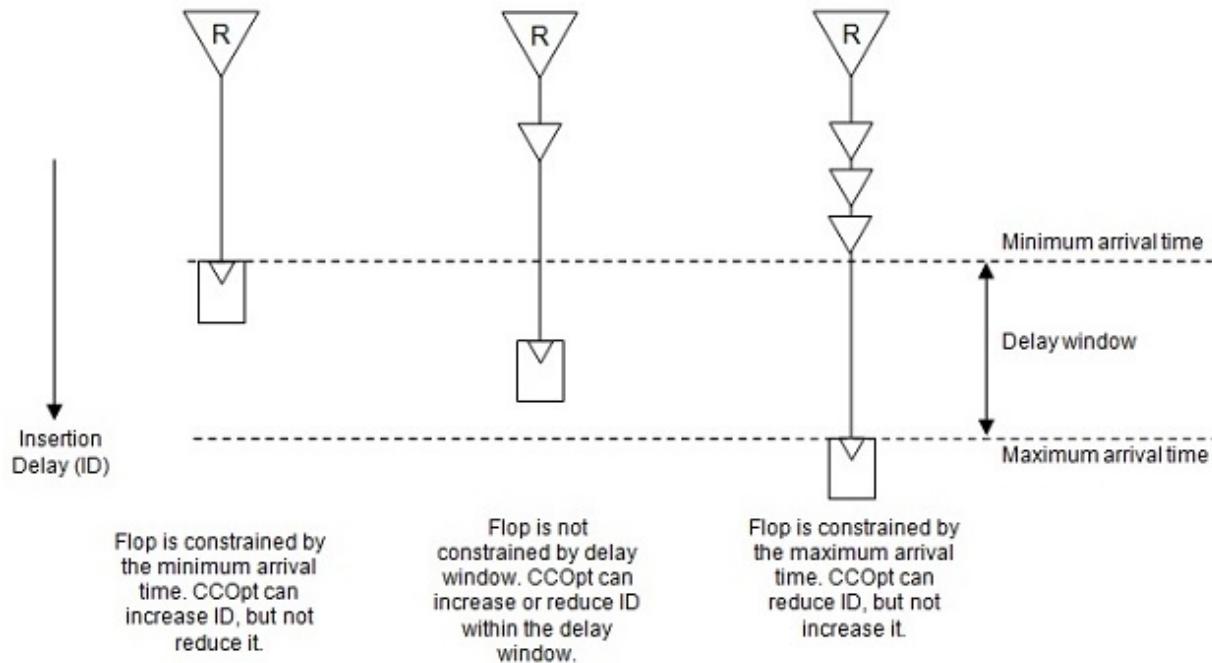
### Partition or Macro in a Disjoint Chain



## Constraint Windows

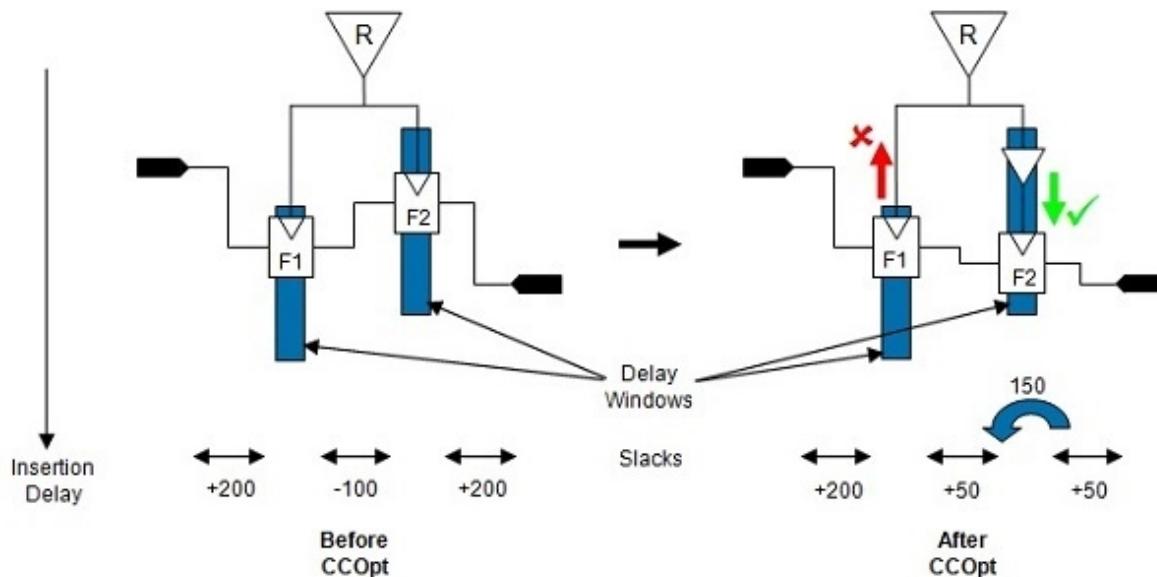
CCOpt determines a delay constraint window for every sink representing the minimum and maximum clock insertion delay (clock arrival time) for the sink. This is illustrated below.

## Constraint Windows



When calculating delay constraint windows, CCOpt considers all the constraints applicable to a particular sink including physical constraints, for example the minimum buffering delay from the clustering step, skew group constraints and insertion delay limit. Useful skew can, therefore, only take place if permitted by the delay constraint window. Consider the example below.

### Skew Scheduling within Constraint Windows

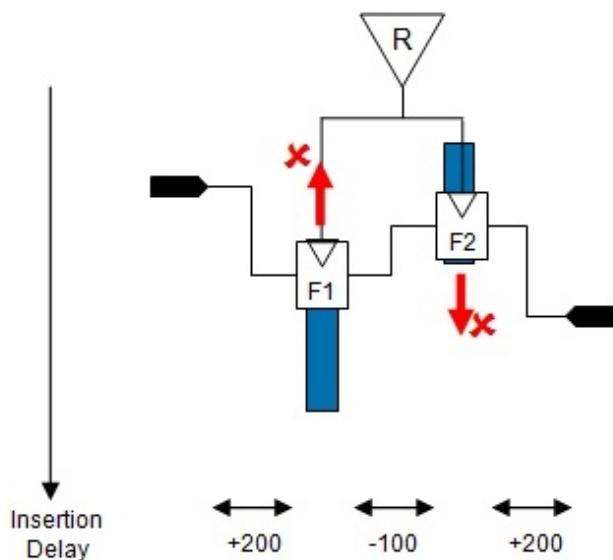


In this example, flop F2 needs to be scheduled later than flop F1 to improve the negative slack of -

100ps. To achieve this, CCOpt could decrease the insertion delay to flop F1 but F1 is already close to the top of the delay constraint window. Alternatively, increasing the insertion delay to flop F2, which is in the middle of its delay constraint window, permits the movement of 150ps of slack from launch side to the capture side of F2.

However, consider the same situation with different constraints and, therefore, different delay constraint windows. In the example below, CCOpt is unable to fix negative slack because of the delay constraint windows.

### **Skew Scheduling Restricted by Constraint Windows**



CCOpt is unable to reduce the insertion delay to flop F1 because it is already at the top of its delay window. Similarly, CCOpt is unable to increase the insertion delay to flop F2 because it is already at the bottom of its delay window. Therefore, the negative slack between F1 and F2 cannot be fixed by useful skew. It might be possible to optimize the datapath between F1 and F2 further, but note that CCOpt will typically only skew clock sinks when datapath optimization is unable to progress.

## **Timing Windows**

Further window types are the "chosen window" that appears in worst chain reports and "timing windows" that are viewable in the CCOpt Clock Tree Debugger. The chosen window represents an insertion delay range that useful skew scheduling would like a sink to be within, in order to progress timing optimization.

The timing window represents the final window used by the implementation step. Each sink is assigned a timing window such that so long as the sink is within the timing window the sink will not be at risk of

degrading the high effort path group(s) WNS and will not adversely impact hold timing. Implementation is able to group sinks together that are physically nearby with overlapping timing windows such that clock tree area and power is reduced by avoiding the need to strictly balance less critical sinks.

For more information on worst chain reporting, both in the log and via the `report_ccopt_worst_chain` command, see the [Worst Chain](#) section.

## Reporting

### Get Commands

There are several `get_ccopt_*` commands that aid TCL scripting and combined with `get_ccopt_property` and other Innovus commands, for example `dbGet`, aid the generation of custom checks and reports. The most commonly used `get_ccopt_*` commands are listed below. For each command, additional parameters may be necessary. For details of these commands, refer to the *Innovus Text Command Reference*.

| Command Name                                     | Usage                                                                                                                                                |
|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>get_ccopt_clock_tree_cells</code>          | Returns all cells which are part of the clock tree graph. Cells that are leaf sinks are excluded.                                                    |
| <code>get_ccopt_clock_tree_nets</code>           | Returns all nets that are part of the clock tree graph.                                                                                              |
| <code>get_ccopt_clock_tree_sinks</code>          | Returns all clock tree sinks, that is automatically determined sinks, stop pins, ignore pins which are part of the clock tree graph.                 |
| <code>get_ccopt_clock_trees</code>               | Returns all clock trees.                                                                                                                             |
| <code>get_ccopt_effective_max_capacitance</code> | Returns the value of the frequency-dependent effective maximum capacitance constraint that the software will apply at a given pin in the clock tree. |
| <code>get_ccopt_skew_group_delay</code>          | Returns the insertion delay for a particular skew group or sink within a skew group.                                                                 |
| <code>get_ccopt_skew_group_path</code>           | Returns a path for a particular skew group or sink within a skew group.                                                                              |

|                                                  |                                                                                 |
|--------------------------------------------------|---------------------------------------------------------------------------------|
| <a href="#">get_ccopt_skew_groups</a>            | Returns all skew groups.                                                        |
| <a href="#">get_ccopt_preferred_cell_stripes</a> | Returns a list of preferred cell stripes objects matching the supplied pattern. |

## Skew Groups

The `report_ccopt_skew_groups -filename file` command creates a report including a summary of all defined skew groups with insertion delay data per delay corner and the maximum and minimum insertion delay paths per skew group and delay corner. The optional `-histograms` parameter can be used to include an insertion delay histogram per delay corner.

The main sections in the skew group report are:

- Skew group structure summary indicating number of active sinks
- Skew group summary indicating maximum and minimum insertion delay per skew group per late/early conditions of each delay corner
- Table of skew group minimum and maximum insertion delay paths per skew group and delay corner with sink pin names. Each path is given an ID number.
- Detailed path listings, using the same path ID number

An example of the skew group summary is illustrated below. A '\*' indicates that a target insertion delay or skew target was not met.

| Timing Corner        | Skew Group                        | ID   | Target | Min ID | Max ID | Avg ID | Skew Target | Type   | Skew Target | Skew  |
|----------------------|-----------------------------------|------|--------|--------|--------|--------|-------------|--------|-------------|-------|
| slow_max:setup.early | div_clk/functional_func_slow_max  | -    |        | 0.651  | 0.809  | 0.737  | ignored     |        | -           | 0.158 |
|                      | my_clk/functional_func_slow_max   | -    |        | 0.922  | 1.139  | 1.096  | ignored     |        | -           | 0.217 |
|                      | test_clk/functional_func_slow_max | -    |        | 0.652  | 0.871  | 0.827  | ignored     |        | -           | 0.218 |
| slow_max:setup.late  | div_clk/functional_func_slow_max  | none |        | 0.651  | 0.809  | 0.738  | explicit    | 0.200  | 0.159       |       |
|                      | my_clk/functional_func_slow_max   | none |        | 0.928  | 1.146  | 1.102  | explicit    | *0.200 | 0.217       |       |
|                      | test_clk/functional_func_slow_max | none |        | 0.663  | 0.880  | 0.836  | explicit    | *0.200 | 0.217       |       |
| fast_min:hold.early  | div_clk/functional_func_slow_max  | -    |        | 0.212  | 0.284  | 0.256  | ignored     |        | -           | 0.072 |
|                      | my_clk/functional_func_slow_max   | -    |        | 0.310  | 0.417  | 0.392  | ignored     |        | -           | 0.106 |
|                      | test_clk/functional_func_slow_max | -    |        | 0.228  | 0.335  | 0.310  | ignored     |        | -           | 0.107 |
| fast_min:hold.late   | div_clk/functional_func_slow_max  | -    |        | 0.213  | 0.284  | 0.256  | ignored     |        | -           | 0.072 |
|                      | my_clk/functional_func_slow_max   | -    |        | 0.316  | 0.422  | 0.398  | ignored     |        | -           | 0.106 |
|                      | test_clk/functional_func_slow_max | -    |        | 0.236  | 0.341  | 0.317  | ignored     |        | -           | 0.105 |

The `report_ccopt_skew_groups` command accepts various parameters to restrict the reporting to specified skew groups or delay corners, or to restrict to particular paths using `-through` and `-to` in a similar manner to the `report_timing` command. The `-summary` parameter can be used just to report the summary. For more details, see the *Innovus Text Command Reference*.

Note that the skew group report uses the same timing model as the CCOpt-CTS engine. To report on

timing clocks, use the [report\\_clock\\_timing](#) command.

## Clock Trees

The [report\\_ccopt\\_clock\\_trees](#) -filename *file* command creates a report including statistics per clock tree and statistics over all clock trees, including transition violations. The *-histograms* parameter can be used to enable histograms of various data and the *-list\_special\_pins* parameter will add a detailed listing of clock tree stop and ignore pins.

## Timing Data for Reports

The timing engine is used to calculate the timing data that is displayed in the clock tree and skew group reports. By default, this is invalidated when you generate the reports, so all the timing data is recalculated. This increases the time taken to generate the reports. Use the *-no\_invalidate* parameter of the [report\\_ccopt\\_clock\\_trees](#) and the [report\\_ccopt\\_skew\\_groups](#) commands to specify that the timing engine should not be invalidated and that the existing timing data, if any, should be used in the reports.

## Worst Chain

For an explanation about the concept of chains, see the [Chains](#) section.

The worst chain is reported from time to time in the log during CCOpt and an examination of the log may help identify reasons limiting timing optimization. In addition, the [report\\_ccopt\\_worst\\_chain](#) command can be used to report the worst timing chain after [ccopt\\_design](#) has completed, but note that this will reflect the current worst chain, not the worst chain during optimization.

The illustration below shows a perfectly balanced worst chain. Each sequential element in the chain is identified by a “cell:name” line with ASCII art on the left representing the chain connectivity. In this example, there is a loop between flops A and B. The data between each sequential element summarizes the combinational path between adjacent sequential elements. For example, the timing slack is identified, and the WNS marked with “\*WNS\*”. In this example, the slack between each stage is identical suggesting that it is not possible to further move slack between stages. Such a chain is balanced.

## Example of Worst Chain

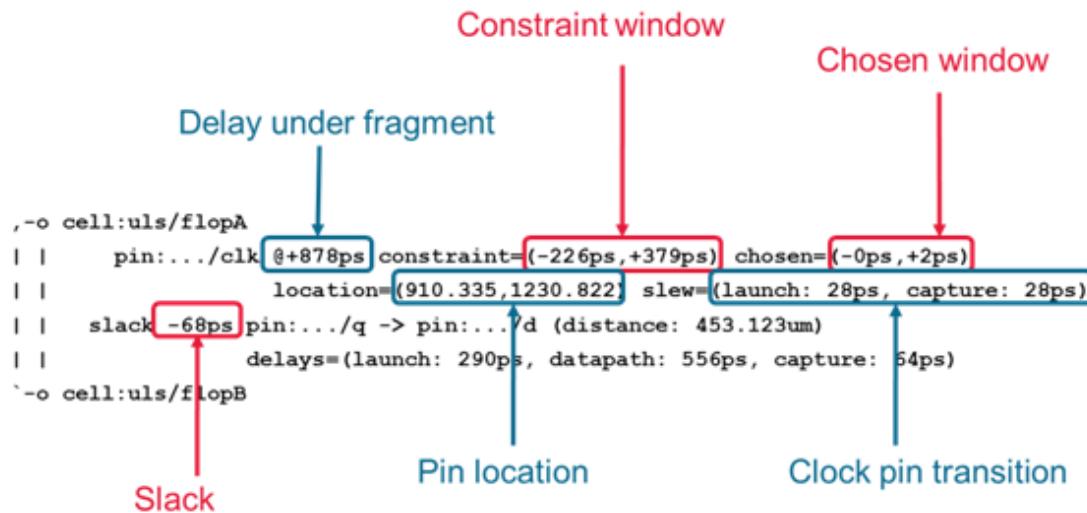
Worst chain (Setup):

```
=====
          Equal slacks
-----
```

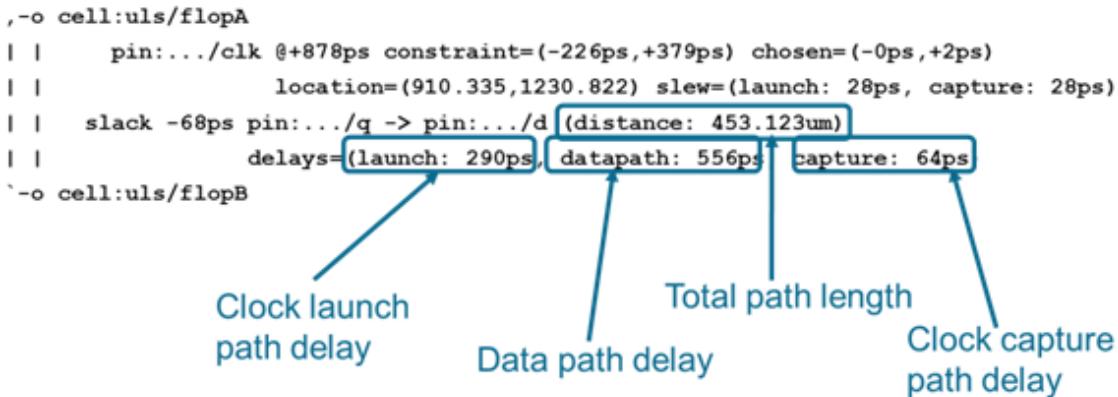
```
,-o cell:uls/flopA
| |   pin:.../clk @+878ps constraint=(-226ps,+379ps) chosen=(-0ps,+2ps)
| |   location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| |   slack -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| |   delays=(launch: 290ps, datapath: 556ps, capture: 64ps)
`-o cell:uls/flopB
|   pin:.../clk @+652ps constraint=(-50ps,+589ps) chosen=(-0ps,+0ps)
|   location=(862.335,1250.822) slew=(launch: 28ps, capture: 28ps)
| *WNS* -68ps pin:.../q -> pin:.../d (distance: 1232.345um)
|   delays=(launch: 64ps, datapath: 782ps, capture: 290ps)
o cell:uls/flopC
|   pin:.../clk @+867ps constraint=(-226ps,+381ps) chosen=(-0ps,+3ps)
|   location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
|   slack -68ps pin:.../q -> pin:.../d (distance: 372.312um)
|   delays=(launch: 290ps, datapath: 556ps, capture: 64ps)
...
...
```

The two diagrams below label the various fields in the worst chain report.

### Worst Chain Data - Diagram 1



## Worst Chain Data - Diagram 2



The above labels are described in detail in the table below.

| Filed Name              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Constraint window       | The constraint window is the range of permissible insertion delay modification subject to the minimum buffering delay from the clustering step, the automatic insertion delay limit and optional user skew constraints. For more information, see the <a href="#">Constraint Windows</a> section. In this example, the -226ps indicates that the sink insertion delay can be scheduled up to 226ps earlier and the +379ps indicates that the sink can be scheduled up to 379ps later. |
| Chosen window           | The chosen window represents the insertion delay range, relative to the current insertion delay, within which useful skew scheduling desires to place the sink.                                                                                                                                                                                                                                                                                                                       |
| Slack                   | The datapath slack between two sequential elements in the chain.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Delay under fragment    | This is an internal number representing the delay between the non-buffer parent of a sink and the sink.                                                                                                                                                                                                                                                                                                                                                                               |
| Pin location            | The placement coordinate of the sink pin.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Clock Pin transition    | The transition time at the sink pin, both for launch and for capture.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Clock launch path delay | The launch clock arrival time.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|                          |                                                                                                        |
|--------------------------|--------------------------------------------------------------------------------------------------------|
| Data path delay          | The datapath delay.                                                                                    |
| Total path length        | The physical length of the path obtained by summing the distance between each pin pair along the path. |
| Clock capture path delay | The capture clock arrival time.                                                                        |

The example below illustrates a worst chain report where the slacks on either side of flopB are unequal. To further improve the WNS, it is desirable to move slack from the launch side of flopB to the capture side of flopB. To do this would require the insertion delay of flopB to be increased but this is not possible because flopB is at the bottom of the constraint window. The source of such a limit is the automatic insertion delay limit as discussed in the [Restricting CCOpt Skew Scheduling](#) section.

#### Example of Worst Chain – sink at bottom of constraint window

```

,-o cell:uls/flopA
| |   pin:.../clk @+600ps constraint=(-226ps,+200ps) chosen=(-0ps,+2ps)
| |           location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| |   *WNS* -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| |           delays=(launch: 64ps, datapath: 556ps, capture: 290ps)
`-o cell:uls/flopB
|   pin:.../clk @+800ps constraint=(-50ps,0ps) chosen=(-0ps,+0ps)
|   location=(862.335,1250.822) slew=(launch: 28ps, capture: 28ps)
|   slack -45ps pin:.../q -> pin:.../d (distance: 1232.345um)
  
```

next path has better slack

0ps – sink at bottom of constraint window

## Halo Violations

Clock cell halo violations can be reported with the `report_ccopt_cell_halo_violations` command. For more information about cell halos, see the [Cell Halos](#) section.

## Cell Name Information

All new cell instances created by CCOpt have an identifying code in their name to let you determine why that cell was created. For example:

cuk Cts: Unknown creator, will not appear in the netlist.

cex Cts: Existing cells in the clock tree which cannot be removed

coi Cts: Cells created as a result of cancelling out inversions

ccl Cts: Created by the clustering process to meet the slew target.....

To view all the available codes and their meanings, run the [show\\_ccopt\\_cell\\_name\\_info](#) command.

## Clock Tree Convergence

The [report\\_ccopt\\_clock\\_tree\\_convergence](#) command reports statistics on the number of paths leading to clock tree sinks, and a list of the top 10 sinks with the most paths. In the example report below, 5 design IOs (primary input/outputs) have 1400 clock paths leading to them. These sinks are likely to be problematic and need further investigation. For details, see the [Clock Tree Convergence](#) section.

Convergence above clock sinks

=====

Number of paths to sink PIOs DFFs Other sinks Total

|      | PIOs | DFFs  | Other sinks | Total |
|------|------|-------|-------------|-------|
| 1    | 1    | 10118 | 0           | 10119 |
| 8    | 54   | 2134  | 183         | 2371  |
| 16   | 13   | 311   | 18          | 342   |
| 32   | 2    | 0     | 0           | 2     |
| 700  | 3    | 164   | 5           | 172   |
| 1400 | 5    | 0     | 0           | 5     |

Sinks with most paths leading to them

=====

Sink Number of paths

|               |      |
|---------------|------|
| Owest/nout[0] | 1400 |
| Owest/nout[1] | 1400 |
| Owest/nout[2] | 1400 |
| Owest/nout[3] | 1400 |
| Owest/nout[4] | 1400 |

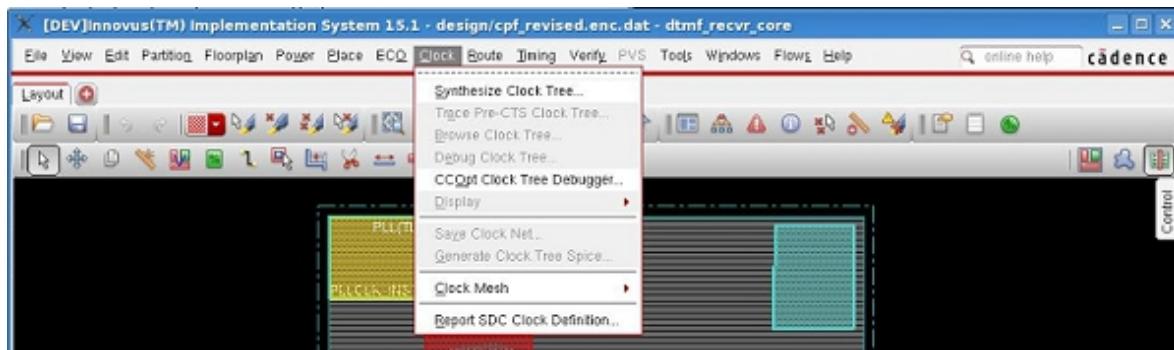
## CCOpt Clock Tree Debugger

The CCOpt Clock Tree Debugger (CTD) provides a graphical interface to explore clock trees and provides debugging capabilities to aid understanding of CCOpt-CTS and CCOpt results. The interface is based on a top-down tree view of all defined clock trees with the vertical axis representing insertion delay. Cells and nets can be colored by various attributes, sections of the clock tree graph can be hidden and unhidden to facilitate navigation of complex clock architectures, and cross probing with the layout view is supported.

Many of the features of the debugger, for example, coloring are self-explanatory from exploring the interface. This section discusses some of the more in-depth features. For details of the *CCOpt Clock Tree Debugger*, see the "CCOpt Clock Tree Debugger" section in the [Clock Menu](#) chapter in *Innovus Menu Reference*.

## Launching the CCOpt CTD

The tool can be accessed from the *Clock Menu* of Innovus. Choose *Clock < CCOpt Clock Tree Debugger*.



Note that this menu selection will only be visible if there is a CCOpt clock tree specification loaded, specifically that one or more clock trees are defined. Alternatively, the `ctd_win` command can be used from a script to launch the CTD, and optional `-id` and `-title` parameters permit the window to be customized. This permits the CTD to be open and ready for use, for example at the end of an overnight run.

The complete submenu as shown above is visible only when the `setCTSMode -engine` option is set to `ck`. For all other CCOpt engine options, `-ccopt`, `auto`, and `ccopt_from_edi_spec`, only *CCOpt Clock Tree Debugger* submenu is visible. This is shown below.



**Note:** Deleting any clock tree or skew group definitions will automatically close all open CTD windows. Multiple windows may be opened, and the following commands permit manipulation of the CTD windows:

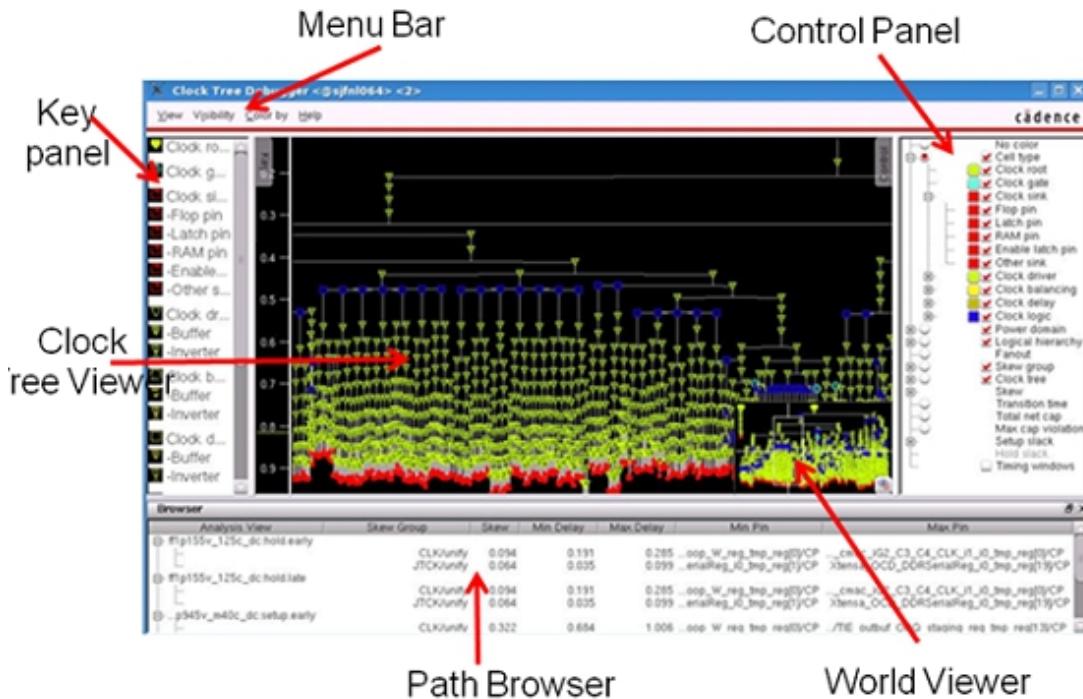
| Command Name                      | Usage                                                                                                                                                       |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">ctd_win</a>           | Open a debugger window. An ID and title can be optionally specified. The ID is used to identify this particular window when using the other commands below. |
| <a href="#">close_ctd_win</a>     | Closes either the specified window by ID, all windows, or the most recently active window.                                                                  |
| <a href="#">get_ctd_win_id</a>    | Get the ID of the most recently active window or the IDs of all open windows.                                                                               |
| <a href="#">get_ctd_win_title</a> | Get the title if the specified window by ID, or the titles of all open windows.                                                                             |
| <a href="#">set_ctd_win_title</a> | Set the title of the most recently active window or the specified window by ID.                                                                             |
| <a href="#">ctd_trace</a>         | Highlight the path to a sink in the active debugger window.                                                                                                 |

For more information about the above commands, see the *Innovus Text Command Reference*.

## CCOpt CTD Interface

The main components in the CTD window are shown below.

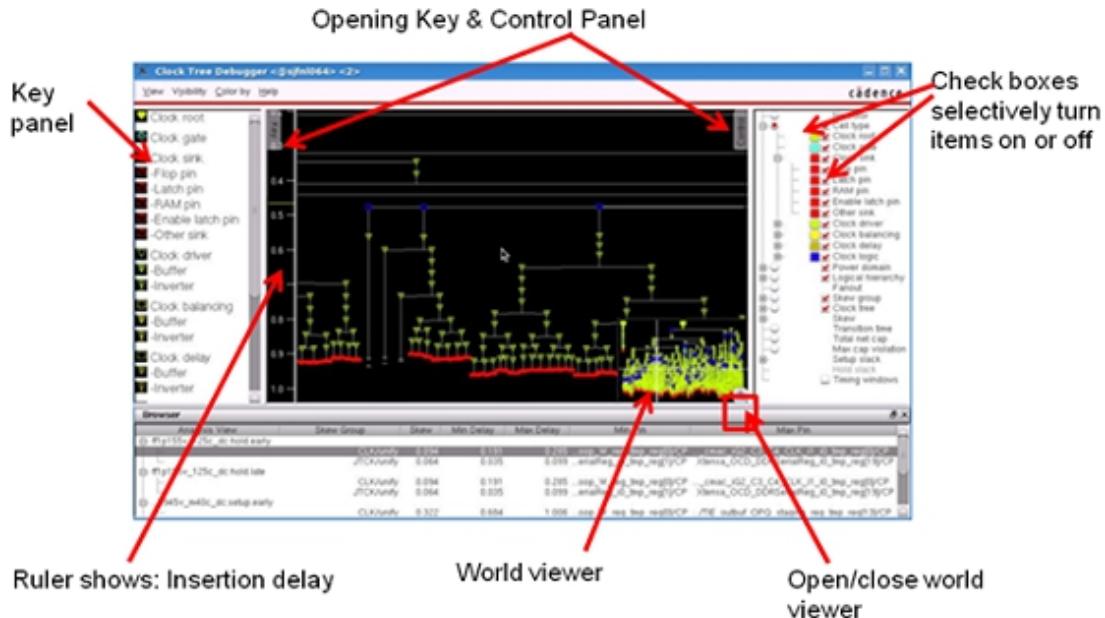
## CCOpt CTD – Main Window Components



- *Clock Tree Viewer*— Displays a top-down tree view of clock trees.
- *World Viewer*— Provides an overview even whilst the Clock Tree Viewer is zoomed in on a smaller region. Clicking in the world viewer will navigate to that area.
- *Control Panel*— Contains controls to determine visibility of different object types and coloring. Additional controls are available via the *View*, *Visibility*, and *Color by* menu items.
- *Key Panel*— A reference key indicating symbols and/or coloring used within the Clock Tree Viewer.
- *Path Browser*— Displays skew group path summary data in a table. Double-clicking on a row or using the right-click context menu permits opening the *Clock Path Analyzer* window. By default, the Path Browser opens at the bottom of the window. The *Clock Path Analyzer*, when invoked, replaces the *Clock Path Browser*.

By default, the *Control Panel* and *Key Panel* are hidden. These panels can be exposed or hidden as illustrated below.

## CCOpt CTD – Opening the Key and Control Panel



## Key Features of the CTD

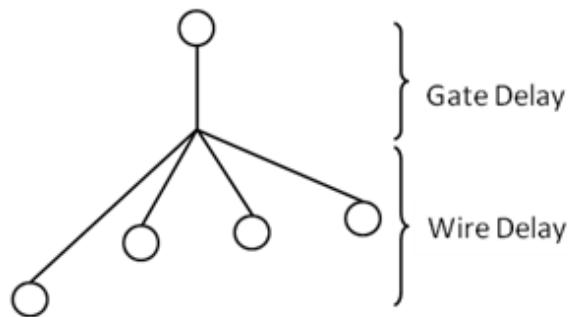
The key features of the CTD are briefly explained in the subsequent sections.

For details of the *CCOpt Clock Tree Debugger*, see the "CCOpt Clock Tree Debugger" section in the [Clock Menu](#) chapter in *Innovus Menu Reference*.

## Clock Tree Representation

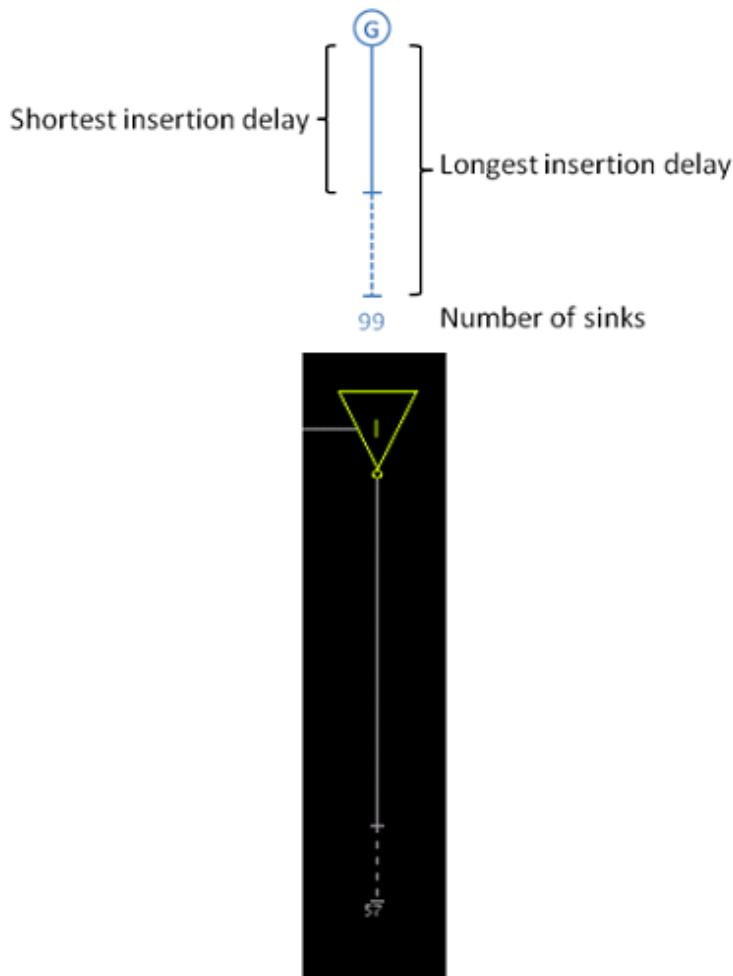
Clock trees are drawn in a top-down tree like manner with the vertical axis representing insertion delay. Different symbols are used for differing cell types, for example buffers, inverters, clock gates, logic and sinks. Gate and wire delay are separately represented, as illustrated below.

## Gate and Wire Delay



## Expanding and Collapsing Sub-trees

Any node in the tree may be either expanded or collapsed. The sub-trees of collapsed nodes are shown as a vertical summary bar indicating the maximum and minimum insertion delay below the node, and the number of sinks in the sub-tree, as illustrated below.



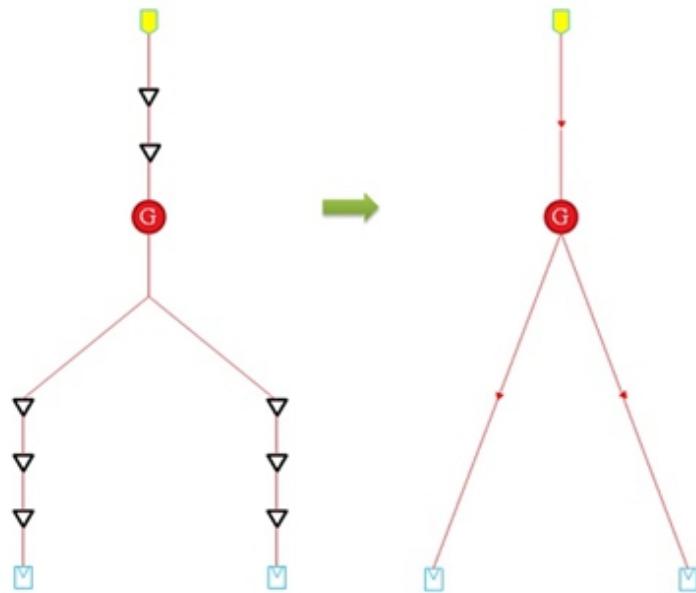
The expanded and collapsed state of a node may be toggled by double-clicking on the node, or using the Expand/Collapse item on the context menu. Additionally, a sub-tree can be marked as un-collapseable by using the context menu. An un-collapseable sub-tree will not be collapsed when its parent is collapsed.

## Simplification

The *View – Simplify* option in the menu bar can be used to further manage visibility, including the following:

- *Mark All Collapsible* – Mark all nodes as collapsible.
- *Collapse all* – Collapse all sub-trees such that each clock tree is fully collapsed.
- *Expand All by Skew Group* – Expand all sub-trees that pass through the specified skew group.
- *Abstraction* – Specified cell types or cell instances can be abstracted using the *View -> Simplify -> Abstract* menu option. Abstracted cells are simply omitted, as represented in the diagram below

where buffer cells, but not clock gates, have been abstracted.



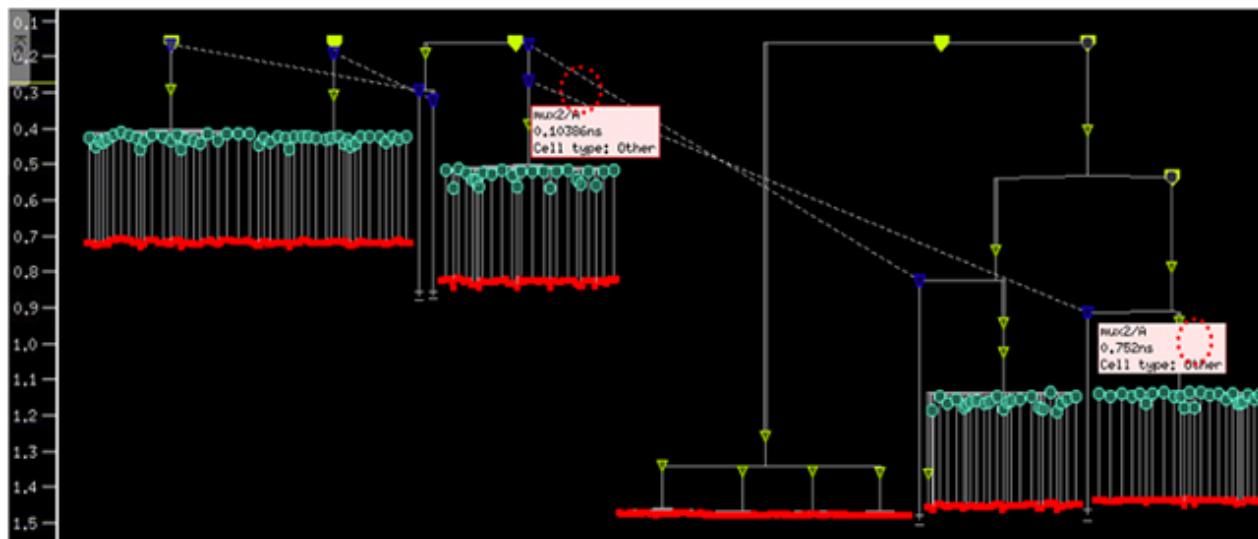
## Context Menu

Right-clicking on a cell opens a context menu. This menu permits various operations to be performed, such as copying the cell name, highlighting the cell, highlighting paths to the cell, opening a schematic viewer, or opening cell properties.



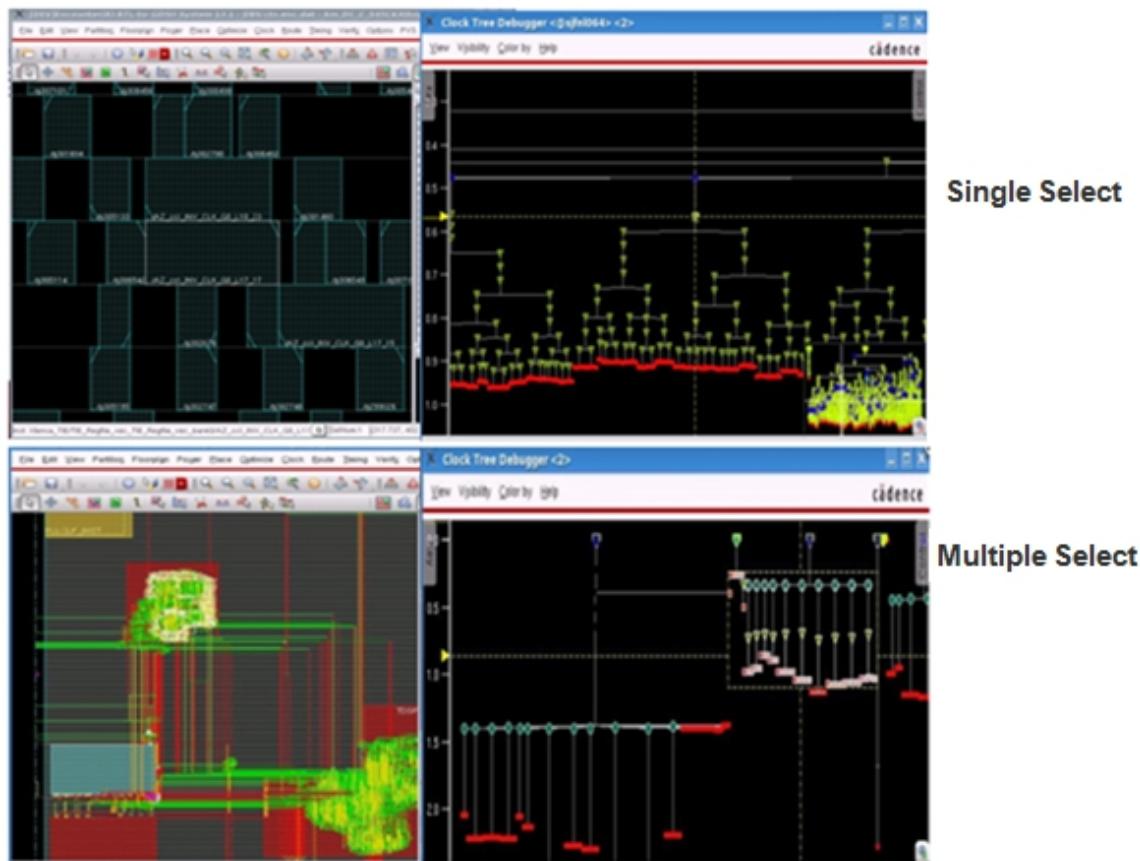
## Multiple Input Cells

A multiple input cell, for example a multiplexer, can exist in more than one clock tree. Multiple input cells are shown in multiple clock trees, but the sub-tree underneath the cell can only be expanded in one clock tree at a time. A dotted line is drawn between the instances of the same cell as illustrated below.



## Cross Probing

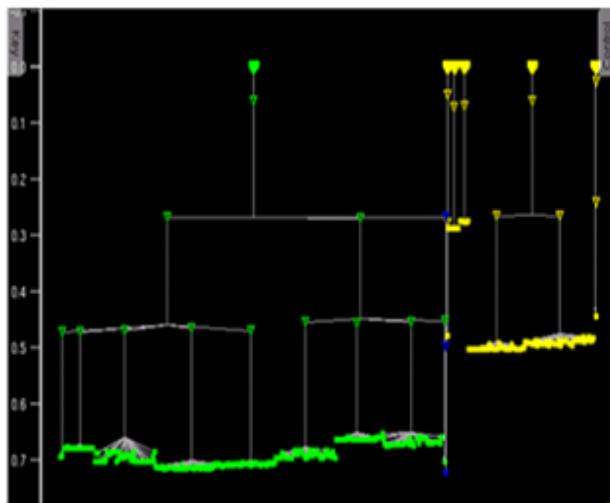
When an object is selected in the main Innovus layout window it will also be selected in the CTD window. Conversely, objects selected in the debugger window will be selected in the layout window. The right-hand mouse button can be used to draw a bounding box to perform multiple selections.



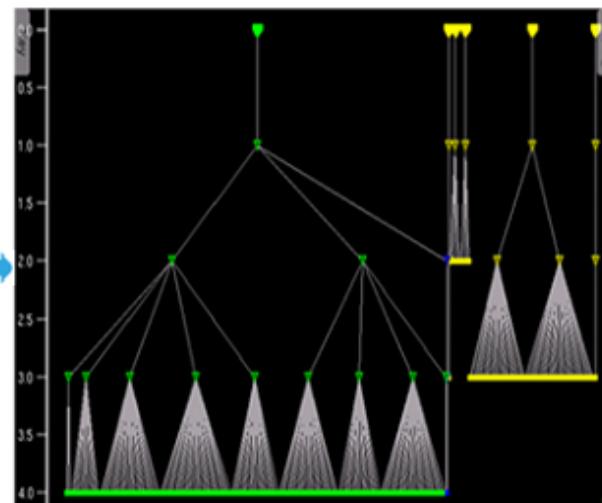
## Unit Delay

The *Visibility – Unit Delay* menu option changes to a unit delay model where each cell has a delay of 0 and each wire a delay of 1.0. This mode is useful for inspecting the clock graph structure before running CCOpt or CCOpt-CTS.

Insertion delay

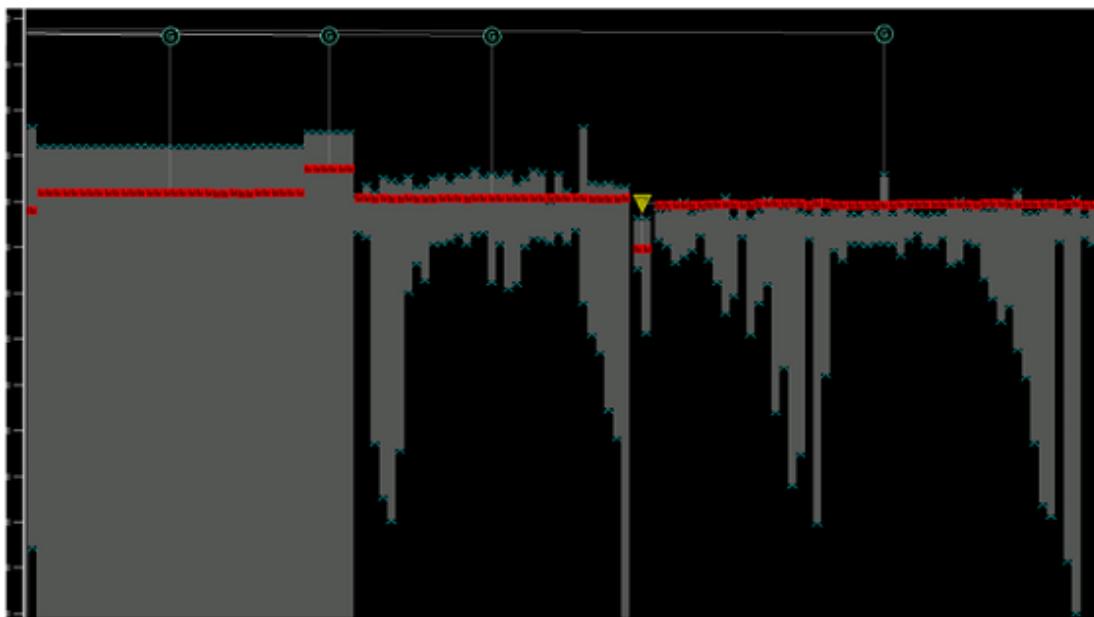


Unit delay



## Timing Windows

After running CCOpt, the timing windows can be shown by clicking on the *Timing windows* option in the *Control Panel*. For each sink, the window is drawn in a grey color as shown in the example below. Timing windows are discussed further in the *Timing Windows* section.



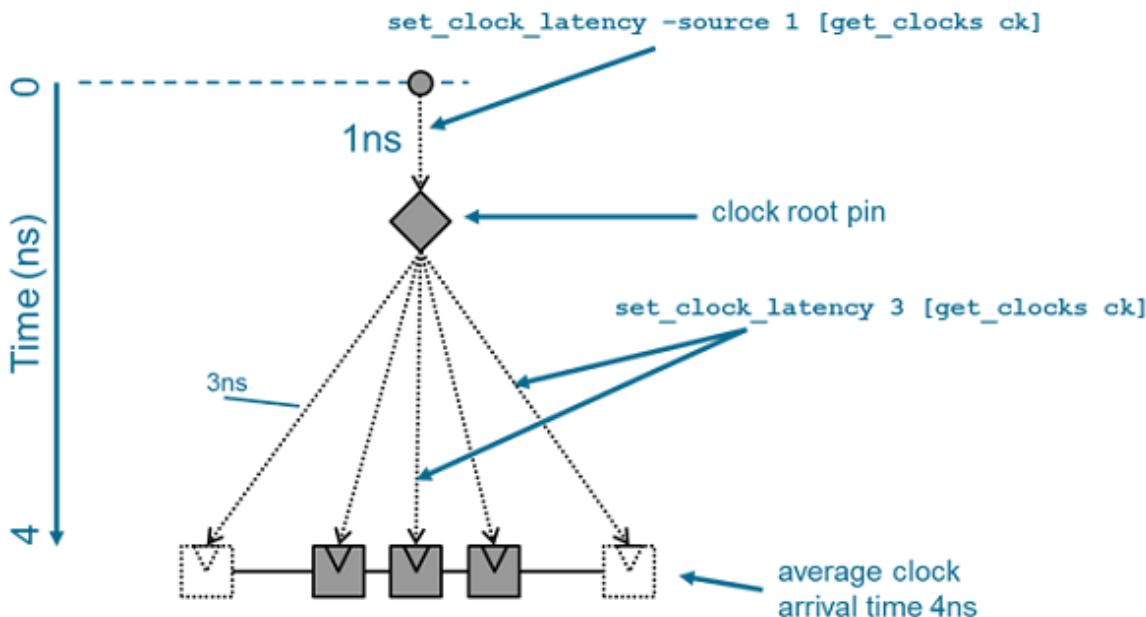
## Additional Topics

### Source Latency Update

Source latency update is performed to ensure that after CTS when clocks are switched to propagated mode that I/O timing and inter-clock timing is consistent with the ideal mode timing model. Stated succinctly:

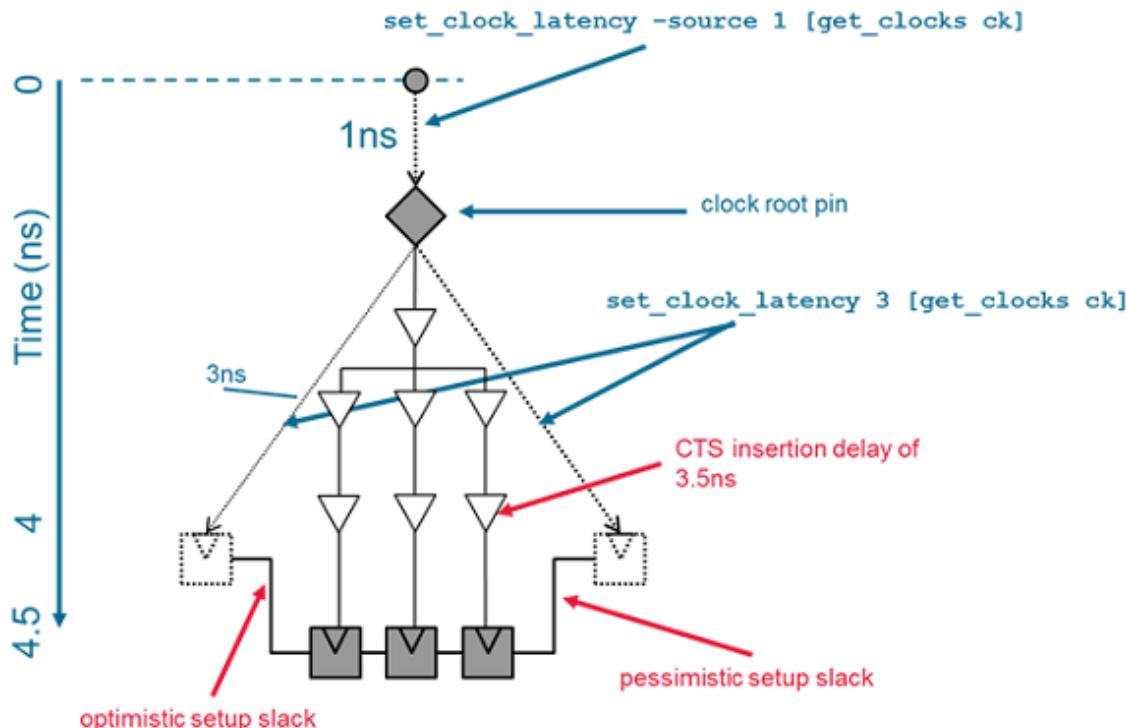
- The source latency update step updates the source latencies at clock root pins such that the per clock average clock arrival time is identical postCTS as it was preCTS.
- This mechanism is best explained using an example. The diagram below represents the before CTS ideal clock mode timing. In this example, there is a single clock with a clock source latency of 1ns and a network latency of 3ns. Therefore, the average clock arrival time at both the I/O pins (represented by dotted flops) and the real sinks is 4ns.

#### Source Latency Update – Before CTS



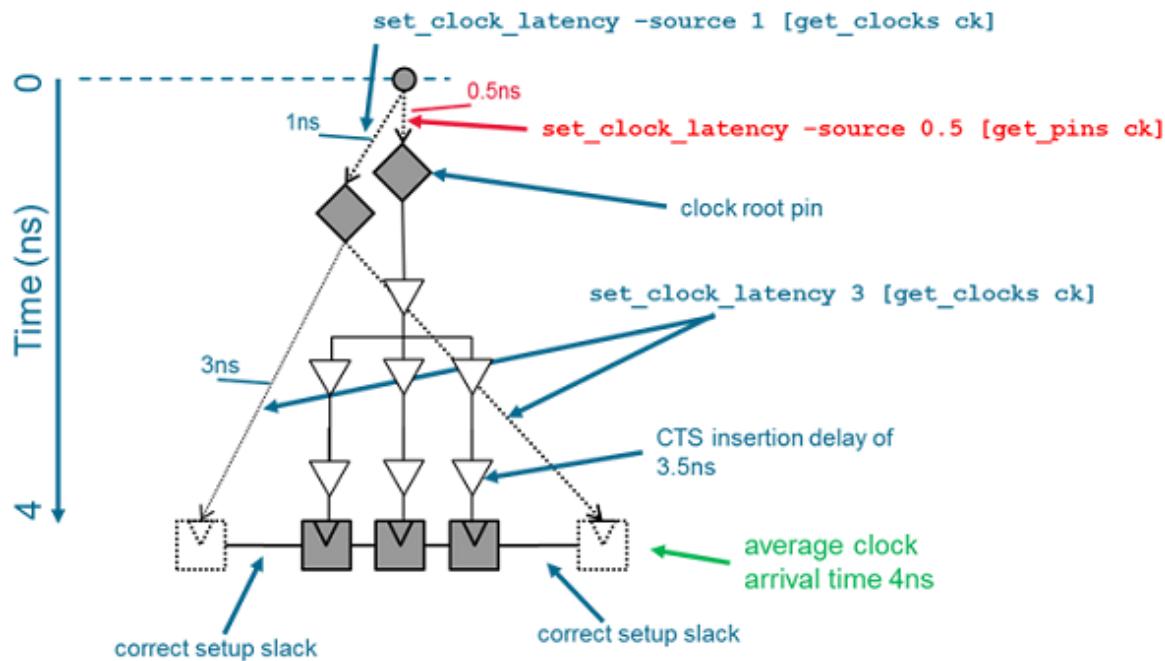
The next diagram below illustrates what would happen if CTS were performed but instead of achieving a 3ns insertion delay CTS achieves a 3.5ns insertion delay. The clock arrival time at the I/O pins is unchanged, but the clock arrival time at the real flops is now 3.5ns. This results in optimistic setup slack on input paths and pessimistic setup slack on output paths.

## Source Latency Update – CTS without Source Latency Update



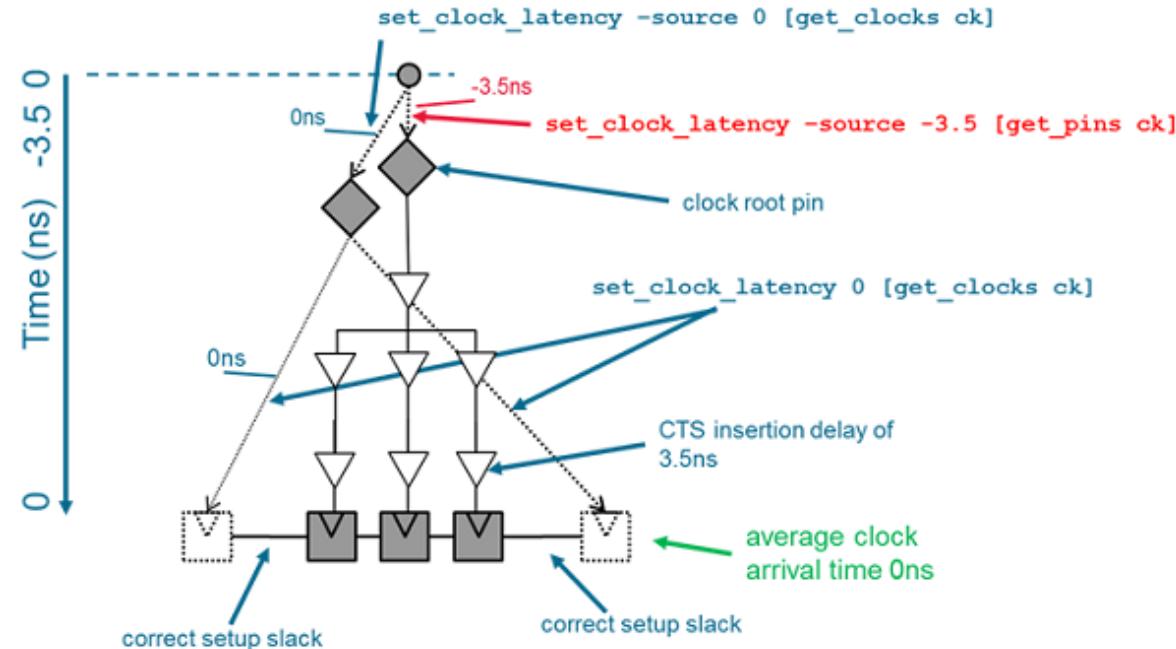
The next diagram illustrates the effect of source latency update. The clock source latency and network latency are unchanged, and the I/O pin timing is unchanged. The clock root pin has the source latency overridden to be 0.5ns instead of 1ns. This adjusts the clock arrival time at the real sinks such that it remains at 4ns. The input paths and output paths are now timed in the manner which was intended preCTS. Unlike other 'I/O latency modification' schemes, this scheme operates correctly in the presence of multiple clock domains communicating with I/O pins without any need for averages or need to match up virtual clocks with real clocks. The source latency modification is performed per clock root pin per clock, such that cross-clock timing consistency is also maintained from before CTS to after CTS. Virtual clocks, if present, do not need modification.

## Source Latency Update – CTS with Source Latency Update



The next diagram below is the same example but with the initial "before CTS" clock and network latencies both at zero. This gives rise to a negative source latency set at the clock root pin. Before CTS the average clock arrival time is zero and this is maintained after CTS via the source latency update.

## Source Latency Update – Variation with Zero Initial Latencies



In the CCOpt-CTS flow, the source latency update step is performed near the end. In the CCOpt flow, the source latency update step is performed after initial virtual delay balancing before timing optimization and useful skew scheduling commences. The source latency updates are reflected in the Innovus timing constraints and will be saved in any saved database or exported in SDC as normal.

In a block-level design with multiple clocks it is likely that each clock will obtain a different source latency modification. For example, a small clock tree might have a source latency modification of -0.5ns while a large clock tree might have a source latency modification of -2ns. This correctly maintains the validity of the timing constraints which were present before CTS for after CTS usage. However, it means that the 1.5ns difference between the small and large clock tree needs to be synthesized at the top level, but it is usually considerably more efficient to do this at the top level than it is at a block level. Typically, at the top level, an ILM model of blocks is used, in which case CTS will be able to directly see the clock paths inside the ILM so the users need take no action to configure this 1.5ns offset.

The source latency modification scheme can be disabled using "`set_ccopt_property update_io_latency false`". The latency modification scheme should be disabled for top-level chips as balancing clocks outside the chip is unlikely to be practical or if the flow requires balancing between clocks to be performed inside the block level. When latency modification is disabled, the designer needs to correctly estimate the achievable clock tree insertion delay and configure clock network latencies accordingly to avoid a timing jump over CTS and the switch to propagated clock mode timing.

## Cell Halos

Cell halos provide a means to enforce additional spacing between clock tree cell instances, specifically between non-sink cell instances and non-sink cell instances, for example between all clock tree buffers, clock gates, and clock tree logic. Halos can be configured by library cell or by clock tree.

For example, use the following commands to set halo distances by library cell:

```
set_ccopt_property cell_halo_x -cell CLKBUFX2 10um
set_ccopt_property cell_halo_y -cell CLKBUFX2 5um
```

Use the following commands to set halo distance by clock tree:

```
set_ccopt_property cell_halo_x -clock_tree ck1 30um
set_ccopt_property cell_halo_y -clock_tree ck1 30um
set_ccopt_property cell_halo_x -clock_tree ck2 10um
set_ccopt_property cell_halo_y -clock_tree ck2 10um
```

The `-cell` and `-clock_tree` parameters can be combined to specify halos per library cell per clock tree. The '`um`' suffix is optional as micrometers are the default units.

The default value for both the `cell_halo_x` and `cell_halo_y` properties is `auto`. When set to `auto`, the `cell_density` and `adjacent_rows_legal` properties will be used instead. From the software's 14.2

release onwards, the default settings for `cell_density` and `adjacent_rows_legal` are changed to `0.75` and `false`, respectively.

Reporting of cell halo compliance (but not density or adjacent rows compliance) is available via the `report_ccopt_cell_halo_violations` command. For more information, see the [Halo Violations](#) section.

## Power Management

CCOpt-CTS and CCOpt respect power management constraints specified via CPF or UPF. Typically, on most designs it is important to permit CTS access to “always-on” buffers that have additional non-switched power. This is important so that CTS can buffer across power domains where the primary power is switched. This is performed simply by including always-on buffers and inverters with the regular buffers and inverters in the cell selection settings. For example:

```
set_ccopt_property inverter_cells {INVX4 INVX8 INVX12 PMINVX4 PMINVX8}  
set_ccopt_property buffer_cells {BUFX4 BUFX8 BUFX12 PMBUFX4 PMBUFX8}
```

**Note:** The order of the cells in the lists is not important.

Just after `ccopt_design` is invoked, the log file will report which library cells are being used in each power domain. For example:

Clock tree balancer configuration for `clock_tree ck`:

CCOpt power management detected and enabled.

For `power_domain SW` and effective domain `power_domain SW`:

Buffers: BUFX8 BUFX4 BUFX1

Inverters: INVX8 INVX4 INVX1

For `power_domain SW` and effective domain `power_domain AO`:

Buffers: PMBUFX8 PMBUFX2

Inverters: PMINVX8 PMINVX2

For `power_domain AO` and effective domain `power_domain SW`:

Buffers: PMBUFX8 PMBUFX2

Inverters: PMINVX8 PMINVX2

For `power_domain AO` and effective domain `power_domain AO`:

Buffers: BUFX8 BUFX4 BUFX1

Inverters: INVX8 INVX4 INVX1

Unblocked area available for placement of any clock cells in `power_domain SW`:

178511.090um<sup>2</sup>

Unblocked area available for placement of any clock cells in `power_domain AO`: 5000.000um<sup>2</sup>

If CTS detects an illegal effective power domain crossing in the clock tree, it will attempt to manage the situation by temporarily overriding the effective domain of the fan-out of a violating clock tree net to match the driver's effective domain. If this happens, then in the log, there will be messages like these:

\*\*ERROR: (ENCCCOPT-1044): CTS has found the clock tree is inconsistent with the power management setup: cell buf1 (a lib\_cell BUFX2) at (10.000,0.000), in power domain PDA has power\_domain PDA and effective domain power\_domain PDA but drives modb/flop2/clk which has power\_domain PDB and effective domain power\_domain PDB.

\*\*WARN: (ENCCCOPT-1110): Clock tree clk has power supply illegalities. Attempting to manage these so that CTS can continue.

Type 'man ENCCCOPT-1110' for more detail.

\*\*WARN: (ENCCCOPT-1110): Considering pin modb/flop1/clk to have power context=power\_domain PDA and effective domain power\_domain PDA, actual power context=power\_domain PDB and effective domain power\_domain PDB

Type 'man ENCCCOPT-1110' for more detail.

To disable this behavior and have CTS abort with an error instead, set the `manage_power_management_illegalities` property to `false`. To disable all power management checks completely, set the `consider_power_management` property to `false`.

## Unbufferable Regions

It is possible that the power management constraints prevent a particular net from being buffered. In this situation, the CTS quality is likely to be severely hampered. CTS logs the following message:

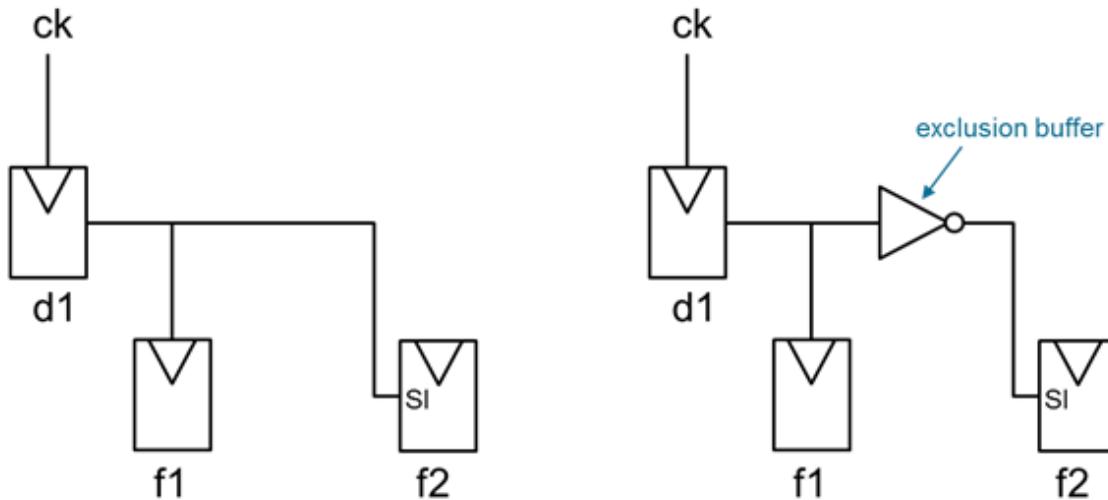
\*\*WARN: (ENCCCOPT-1042): CTS cannot select a library cell to use as a driver at one or more points of `clock_tree ck`.

In order to see the detail of such messages such as the net involved, set the `cts_detailed_cell_warnings` property. This will result in many ENCCOPT-1042 messages being logged with detailed information. Such occurrences frequently result in maximum transition violations and increased insertion delay and should be investigated.

## Shared Clock and Data Concerns

In some situations, a net may be used for both clock and datapath purposes. Consider the left-hand side example below with clock divider flop d1 clocking flop f1. However, the output of d1 is additionally connected to the SI input of f2 as part of the scan chain. The net driven by d1 is part of the clock tree graph and may not be operated on by datapath optimization including datapath hold fixing. This restriction prevents datapath transforms from disrupting clock timing. After CTS it is possible that there exists a hold violation at the input of f2, but datapath hold buffer insertion will not be able to insert a buffer to fix it.

## Addition of an Exclusion Buffer



The SI input of flop f2 will by default be a clock tree exclude pin. The command, `ccopt_add_exclusion_drivers` can be used to add exclusion buffers to isolate exclude pins from the clock tree graph. Alternatively, the `add_exclusion_drivers` property can be set to enable this to happen automatically at the start of `ccopt_design`. The inputs to exclusion buffers are explicitly set to be clock tree ignore pins.

As illustrated in the right-hand side example, once the exclusion buffer is added, datapath hold fixing, or other datapath optimization is free to operate on the net between the exclusion buffer and flop f2.

## CCOpt Property System

This section details the way the CCOpt property system operates behind the `set_ccopt_property` and `get_ccopt_property` commands. Note that some properties are read only and so many more properties are accessible to `get_ccopt_property` than `set_ccopt_property`.

## Setting Properties

Properties have a name and are assigned a value. Properties can be global, or per object type. The property name, value, and if desired or required the object type and object pattern can be specified. For example:

```
set_ccopt_property use_inverters -clock_tree ct1 true
```

```
set_ccopt_property -skew_group sg1 target_skew 0.1
set_ccopt_property -delay_corner dc1 -delay_type late -skew_group sg1 target_skew 0.15
set_ccopt_property [-< obj_type > < obj_pattern >] < prop_name > < value >
```

The most commonly used object types are: -skew\_group, -clock\_tree, -pin, -inst, -lib\_pin, -delay\_corner, -net\_type.

The property name and value parameters are positional and must appear in order. The object type and object pattern specification can appear anywhere. All of the commands below are equivalent:

```
set_ccopt_property insertion_delay -skew_group sg1 1.5
set_ccopt_property -skew_group sg1 insertion_delay 1.5
set_ccopt_property insertion_delay 1.5 -skew_group sg1
```

If an object type is not specified, then the setting will apply to all relevant objects including ones that are created in the future.

## Wildcards

The object name can involve wildcard style patterns, for example:

```
set_ccopt_property use_inverters -clock_tree test* true
```

Wildcards are resolved when the command is issued not when the property value is used. In the command shown above, if a new clock tree `test_new` was defined after the `set_ccopt_property` command was issued, it would not have the `use_inverters` property set.

## Optional Object Type Switches

If the `-object_type` parameter is omitted, this implies a "forall" on that type value. The example below specifies that CTS should use inverters for clock tree `ct1`:

```
set_ccopt_property use_inverters -clock_tree ct1 true
```

The example below does not specify the `-clock_tree` object type:

```
set_ccopt_property -use_inverters true
```

And is equivalent to:

```
foreach ct [get_ccopt_clock_trees *] {
    set_ccopt_property -use_inverters -clock_tree $ct true
}
```

## Delay Corner Special Handling

There are some exceptions to the above rule that are designed to capture common user intent. The main exceptions are properties for which the `-delay_corner` object type and `-early/-late` parameters apply. When these parameters are omitted, the default behavior is that the property is not set across all delay corners but instead it will be set just for the primary delay corner. In the following example, the leaf-level properties are maintained internally and one or more properties can be set at a time by specifying additional information. The example illustrates the internal representation of the `target_skew` property in a design with one delay corner used for setup timing and two delay corners used for hold timing. The initial default values are shown below:

|         | Early  | Late   |
|---------|--------|--------|
| SetupDC | Ignore | Auto   |
| Hold1DC | Ignore | Ignore |
| Hold2DC | Ignore | Ignore |

If the command “`set_ccopt_property target_skew 0.05`” is issued, the internal representation will change to become:

|         | Early  | Late   |
|---------|--------|--------|
| SetupDC | Ignore | 0.05   |
| Hold1DC | Ignore | Ignore |
| Hold2DC | Ignore | Ignore |

If the command “`set_ccopt_property target_skew -delay_corner Hold1DC 0.1`” is issued, the representation will change to:

|         | Early  | Late   |
|---------|--------|--------|
| SetupDC | Ignore | 0.05   |
| Hold1DC | 0.1    | 0.1    |
| Hold2DC | Ignore | Ignore |

Note that the specification of “`Hold1DC`” matches two cells in the table. To restrict further, add `-early` or `-late`.

## Getting Properties

The `get_ccopt_property` command is used to retrieve the values of properties. For example, in the last table shown above, the command “`get_ccopt_property -target_skew -delay_corner SetupDC -late`” will return a value of 0.05.

However, if some or all of the “`-key_name key_value`” switches are omitted, then multiple values may be returned in a list format. An example is shown below.

```
get_ccopt_property -target_skew -delay_corner SetupDC
```

returns the following:

```
{ \
  { -delay_corner SetupDC -early -value default } \
  { -delay_corner SetupDC -late -value 0.05 } \
}
```

If all the selected cells have the same value then a single value instead of a list will be returned. To force the return of a fully expanded list, even if all values are the same, use the `-list` parameter. For a summary of all parameters of the `get_ccopt_property` and `set_ccopt_property` commands, see *Innovus Text Command Reference*.

## Getting a List of Properties and Descriptions

To obtain help on a property, or to find properties matching a wildcard pattern:

```
get_ccopt_property -help propertyName or pattern
```

To obtain a list of all available properties:

```
get_ccopt_property -help *
```

For example:

```
get_ccopt_property -help target_max_trans
```

This specifies the target skew for clock tree balancing. This may be set to a numeric value, or one of 'auto', 'ignore' or 'default'.

If set to 'auto' this indicates that an appropriate skew target should be computed.

If set to 'ignore' this indicates that skew should not be balanced for this corner/path combination.

If unspecified then the value of this property is 'default'.

If the value of the property is 'default' the target skew for late delays in the primary delay corner is interpreted as 'auto' and as 'ignore' otherwise.

Valid values: default | auto | ignore | double

Default: default

Optional applicable arguments: "-skew\_group <name>", "-delay\_corner <name>", "-early" and "-late".

## Migrating from FE-CTS

A list of the CTS engines and how to choose the CTS engine used is given in the [The Clock Tree Synthesis Engines](#) section. It is recommended that users migrate from using FE-CTS to using CCOpt-CTS. There are several possible variants:

- **Clean start** – The CCOpt-CTS clock tree specification is created using the `ccopt_clock_tree_spec` command and FE-CTS clock specification files and FE-CTS related commands are not required in the flow. This is the recommended approach.
- **Specification translation** – An existing FE-CTS clock tree specification is translated to a CCOpt-CTS clock tree specification. This is the default behavior for the automatic engine setting if a FE-CTS clock tree specification is loaded, or if this mode is explicitly requested. A list of the CTS engines and how to choose the CTS engine used is given in the [The Clock Tree Synthesis Engines](#) section. This approach is recommended only for existing projects where a heavily manually customized FE-CTS specification is in use such that there is not time to deploy the ‘clean start’ approach. To deploy CCOpt, rather than CCOpt-CTS, this approach is not recommended.
- **Global setting translation** – This is essentially the same as ‘clean start’ except that some global settings can be configured from an FE-CTS specification file. This can be done by using the command, `set_ccopt_mode -import_edi_cts_spec true`, which will configure `set_ccopt_mode` and `setCTSMode` settings based on the FE-CTS specification contents. This approach is not recommended since `set_ccopt_mode` support is for backward compatibility.
- **Macro model translation** – This is essentially the same as ‘clean start’ except that data for partitions or macros can be read from existing FE-CTS macro model files. The command, `set_ccopt_mode -edi_spec_for_macro_models filename` will read MacroModel and DynamicMacroModel statements from the specified file. MacroModel statements are translated to `skew_group_insertion_delay` property settings, while DynamicMacroModel statements are translated to `create_ccopt_skew_group` commands with `-exclusive_sinks` to balance the specified sinks.

## Specification Translation

To run CCOpt-CTS using an existing FE-CTS specification the recommended flow is as follows:

```
specifyClockTree -filename fcts.spec
create_ccopt_clock_tree_spec -file ccopt.spec -from_fcts_spec
source ccopt.spec
ccopt_design -cts
```

This flow loads the FE-CTS specification and `create_ccopt_clock_tree_spec` is used to translate the FE-CTS specification to a CCOpt-CTS specification. The CCOpt-CTS specification can be inspected and edited prior to loading before invoking CCOpt-CTS. By default, the `clockDesign` command will do the conversion and run CCOpt-CTS but without the opportunity to inspect the converted specification.

## Concept Mapping

The following table provides an approximate mapping between FE-CTS specification commands and the CCOpt-CTS equivalents. Note that due to the more flexible data model used by CCOpt-CTS, which eliminates the need for sequential CTS steps, an exact 1:1 relationship does not exist.

| FE-CTS                 | CCOpt-CTS                                                             |
|------------------------|-----------------------------------------------------------------------|
| AutoCTSRootPin         | <code>create_ccopt_clock_tree</code>                                  |
| LeafPin/ExcludedPin    | <code>set_ccopt_property sink_type stop/ignore/exclude</code>         |
| SinkMaxTran/BufMaxTran | <code>set_ccopt_property target_max_trans -net_type leaf/trunk</code> |
| maxSkew                | <code>set_ccopt_property target_skew</code>                           |
| MacroModel             | <code>set_ccopt_property insertion_delay -pin</code>                  |
| DynamicMarcoModel      | <code>create_ccopt_skew_group -exclusive_sinks</code>                 |
| clkGroup               | <code>create_ccopt_skew_group -balance_skew_groups</code>             |
| CellHalo               | <code>set_ccopt_property cell_halo_x/cell_halo_y</code>               |
| RouteTypeName          | <code>create_route_type</code>                                        |

## Legacy FE-CTS Flow

Performing CTS with `clockDesign` includes the following tasks:

- Creating the clock tree specification file
- Building a buffer distribution network
- Routing clock nets using NanoRoute

## Creating the `clockDesign` Specification File

CTS is a series of procedures to build a buffer distribution network to meet the design's timing targets. The clock tree specification file is used to direct `clockDesign` and includes the following:

- Design constraints including latency, skew, and design rules
- Buffer and routing type definitions
- Trace and synthesis controls such as MacroModel, ClkGroup, NoGating, LeafPin, ExcludedPin, PreservePin, ThroughPin, and GatingGroupInstances
- Flow controls such as whether or not to:
  - Generate a detail report
  - Route the clock net
  - Perform postCTS optimization

You can generate the default clock tree specification file using the following command:

```
createClockTreeSpec -file filename
```

Automatically generating a clock tree specification translates information from the timing constraint file into suitable records for the clock tree specification file.

- Use the `createClockTreeSpec -bufferList bufferList` command to specify the buffers that CTS should use
- Route types control how the clock nets are routed. Route types for nets connected to leaf cells and nets connected to non-leaf cells can be specified separately in the specification file with `LeafRouteType` and `Routetype`. Also, you can use the `setCTSMode` command before running the `specifyClockTree` command to change the default routing type and global CTS controls.

The clock tree specification file is very important and directly affects the result of `clockDesign`. A good clock tree plan including suitable constraints and placement space can improve the results of CTS and avoid problems for postCTS timing closure.

The PreCTS Clock Tree Tracer (*Clock - Trace PreCTS Clock Tree*) user interface can be used to traverse the clock tree structure logically and physically based on the applied clock specification file

before committing CTS. You can use it as a basis for changing the clock tree specification file to consolidate the clock tree structure and improve the results of CTS.

Sometimes certain elements must be skewed manually. This can happen when preCTS useful skew is not enabled or preCTS cannot predict the magnitude of the problem due to skew/derating. In preCTS, you can model this using the `set_clock_latency` SDC construct. The following example shows the clock delay to A/B/RAM1/CLKA is pulled in 500ps:

```
set_clock_latency -0.5 A/B/RAM1/CLKA
```

To model this in the CTS specification file it would appear as follows. :

```
MacroModel pin A/B/RAM1/CLKA 0.5ns 0.5ns 0.5ns 0.5ns 0pF
```

The `+0.5ns` means that `500ps` of latency is "inside" the `CLKA` pin of A/B/RAM1

## Synthesizing the Clock Tree with `clockDesign`

To generate the clock tree, use the `clockDesign` command. This command performs the following operations during CTS:

- Deletes any existing buffers on the clock nets
- Builds a buffer distribution network to distribute the clock signal(s) to the registers
- Routes the clock nets using NanoRoute
- Optimizes the clock tree

The `clockDesign` command is a super-command that runs the commands in the CTS flow, such as, `createClockTreeSpec`, `specifyClockTree`, `deleteClockTree`, `ckSynthesis`, and so on. It is important to note that `clockDesign` automatically sets some CTS options that are disabled by default. So if you are comparing a `clockDesign` run to a run where each command is run separately, ensure that the settings are consistent. The table below compares the `clockDesign` settings to the default settings.

| Option       | <code>clockDesign</code> Setting | Default Setting |
|--------------|----------------------------------|-----------------|
| RouteClkNet  | Yes                              | No              |
| PostOpt      | Yes                              | Yes             |
| OptAddBuffer | Yes                              | No              |

The `clockDesign` command generates the default clock tree specification file (if not specified), deletes existing clock trees, builds the clock tree, calls NanoRoute to route the clock nets, and then optimizes the clock tree to improve the skew including resizing buffers or inverters, adding buffers, refining placement, and correcting routing.

If you have performed useful skew optimization using the `setOptMode -usefulSkew true` command, `clockDesign` automatically checks for any scheduling file in the working directory, or checks for "rda\_input ui\_scheduling\_file", and honors the scheduling file while building the clock tree.

If the `clockDesign` command calls NanoRoute to route the clock nets, then direct NanoRoute to follow the route guide by using the command, `setCTSMode -routeGuide true`. This is enabled by default. This operation can improve the correlation between preRoute and postRoute clock nets.

## Analyzing and Debugging the `clockDesign` Results

You can use the Clock Tree Browser (*Clock - Browse Clock Tree*) user interface to fine-tune the clock tree to improve the results. From the user interface you can perform the following operations:

- Add buffers
- Delete buffers
- Size cells
- Change net connections

Use Global Clock Tree Debug (*Clock - Debug Clock Tree*) to debug the timing result. Refer to the [Legacy FE-CTS Capabilities](#) chapter in the *Innovus User Guide* for more information. Also, see the [Clock Menu](#) chapter in the *Innovus Menu Reference* for descriptions of the forms and fields of the user interface. Sometimes a degradation in clock delay or skew occurs during CTS when comparing the results before and after the clocks are routed. If this occurs, try the following:

- Confirm that the RC scaling factors for the clocks are set properly. See [How to Generate Scaling Factors for RC Correlation](#).
- Constrain the routing to two upper routing layers using a RouteType in the CTS specification file. Constraining the routing to two layers reduces differences in layer assignment between CTS and NanoRoute.
- Use `displayClockMinMaxPaths` with the `-preRoute` and `-clkRouteOnly` options to compare preRoute and clock route paths.

## Optimizing a `clockDesign` Built Clock Tree

Below are suggestions to optimize the `clockDesign` results.

The following setting is used to reduce the size of the tree. CTS performs optimization after the tree construction to delete and downsize elements to recover area. Also, when too many cells are inserted, try relaxing the constraints (typically the Buf/Sink MaxTran):

```
setCTSMode -optArea true
```

When routing rules are causing the problems, consider only using the rules for the non-sink levels (or using a less restrictive rule for the sinks)

- In the CTS spec file:
  - RouteType controls the routing rules for non-sink levels
  - LeafRouteType controls the routing rules for the sink level
- MaxTran constraints typically have the largest effect on the size of the tree so relaxing these helps reduce impact

Mode settings to reduce latency

- The following affects actual tree construction. It can increase run time considerably and ignores MinDelay constructs:  

```
setCTSMode -synthLatencyEffort high
```
- The following performs optimization after the tree construction mostly by optimizing the location of the tree elements and can also add significant run time:  

```
setCTSMode -optLatency true
```
- The following reduces the size of the tree by performing optimization after the tree construction to delete and downsize elements to recover area:  

```
setCTSMode -optArea true
```

Manual skewing

- Sometimes certain elements must be manually skewed. This can arise when preCTS useful skew is not enabled, or preCTS cannot predict the magnitude of the problem due to skew/derating. In preCTS you can model this using the set\_clock\_latency SDC construct. The following example shows that the clock delay to A/B/RAM1/CLKA is pulled in 500ps:

```
set_clock_latency -0.5 A/B/RAM1/CLKA
```

- To model this in the CTS spec file, it would appear as follows. The +0.5ns means that 500ps of latency is "inside" the CLKA pin of A/B/RAM1:

```
MacroModel pin A/B/RAM1/CLKA 0.5ns 0.5ns 0.5ns 0.5ns 0pF
```

After `clockDesign`, `ckECO` can be used to improve the tree based on the parasitics and timing seen by the optimizer.

- `ckECO` by default can use all the allowed buffers/inverters. To limit it to only those in the CTS spec file, use the `-useSpecFileCellsOnly` parameter.

```
ckECO -postCTS -useSpecFileCellsOnly
```

- A similar flow can be used after detailed routing. Be aware that if useful skew was applied during postCTS optimization, `ckECO -postRoute` may undo this because its goal is to minimize skew:  
`ckECO -postRoute -useSpecFileCellsOnly`
- If you are looking for local skew reduction (skew between talking flip-flops) use the `-localSkew` parameter:  
`ckECO -postCTS -useSpecFileCellsOnly -localSkew`
- Check the CTS log file for clock gating element movement during `optDesign -postCTS`:
  - Use the `-clockGateAware true` parameter of the `setPlaceMode` command in placement
  - Or do not allow gated elements to move during CTS:  
`setCTSMODE -optLatencyMoveGate false`

# Legacy FE-CTS Capabilities

- Before You Begin
- Results
- Understanding the CTS Operation Modes
  - Manual CTS Mode
  - Automatic CTS Mode
- How CTS Calculates Skew Values
- Improving PostRoute Correlation
  - Method 1
  - Method 2
- Specifying Macro Model Delays
  - Macro Model Support for MMMC Views
  - Dynamic Macro Model
- Grouping Clocks
- Analyzing Hierarchical Clock Trees
- Module Placement Utilization
- Clock Designs with Tight Area
- Balancing Pins for Macro Models
- Timing Model Requirement for Cells
- Delay Variation and OCV
- Understanding PostCTS Clock Tree Optimization
  - Using the ckECO Command for PostCTS Clock Tree Optimization
  - Support for Local Skew Optimization
  - Command Modes for the ckECO Command
  - Using a SPEF File with the ckECO Command for RC Estimation
  - Running PostCTS Optimization with the ckECO Command
  - Guidelines for Using the ckECO Command
- Creating a Clock Tree Specification File
  - Using the Automatic Clock Tree Specification File Generator
    - Creating the CTS Specification File in the MMMC Mode
  - Example of a Clock Tree Specification File
  - Naming Attributes Section
  - NanoRoute Attribute Section

- Macro Model Data Section
- Clock Grouping Data Section
- Clock-Tree Topology Section
- Automatic Gated CTS Section
- Log File Headings
- CTS Report Descriptions
  - General Information
  - Macro Model Information
  - Power Information
  - AC Current Density Violations
- Supported SDC Constraints



## Before You Begin

**i** CCOpt-CTS is Default

As of the software's 14.2 release, the default clock tree synthesis (CTS) engine is CCOpt-CTS. To use the FE-CTS engine, run the following command:

```
setCTSMODE -engine ck
```

Before you run CTS on your design, make sure the following files are available:

- Clock tree specification file  
`DontAddNewPortModule`
- Verilog netlist
- GDSII or LEF physical library
- Proper RC model from LEF, Innovus™ technology file, or Innovus™ Implementation System (Innovus) capacitance table  
For information on RC extraction in Innovus, see [RC Extraction](#) chapter in the *Innovus User Guide*.
- Timing constraints file (optional)
- `.lib` file or TLF file with timing models for standard cells and cell footprint names
- Placement information, such as a DEF file or an Innovus placement file

## Results

After a CTS run, CTS creates reports on the results of the run in ASCII text or HTML format. CTS also creates routing guide files (to guide NanoRoute on routing the clock nets) and macro model files (for partitions or modules).

## Understanding the CTS Operation Modes

There are two modes for running CTS: manual and automatic.

- Manual CTS mode allows you to control the number of levels and the number of buffers, and

specify the types of buffers at each level.

- In automatic CTS mode, CTS automatically determines the number of levels and buffers based on the timing constraints in the clock tree specification file, such as the maximum delay and maximum skew.

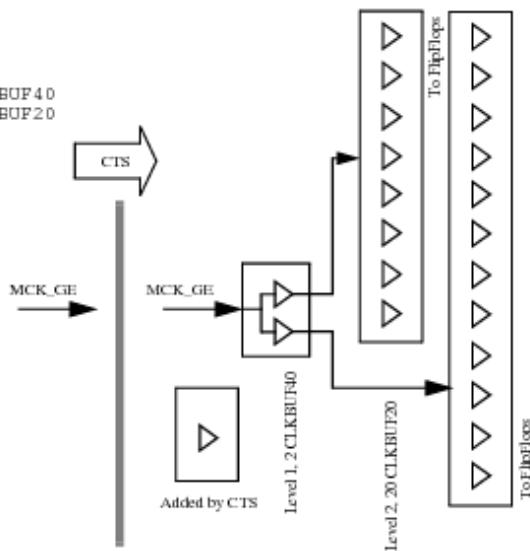
**Note:** For details on how the clock tree specification file is created and to view an example of the same, refer to the "Creating a Clock Tree Specification File" section.

## Manual CTS Mode

You can run manual CTS on a clock net and specify the levels of clock buffers. CTS builds the clock buffer tree according to the clock tree specification file, generates the clock tree topology, and balances the clock phase delay with inserted clock buffers. However, CTS does not trace the clock net.

Following is an example of clock-tree specification file syntax and a graphic representation of the syntax:

```
ClockNetName MCK_GE
LevelNumber 2
LevelSpec 1 2 CLKBUF40
LevelSpec 2 20 CLKBUF20
PostOpt YES
End
```



## Automatic CTS Mode

In this mode, CTS traces the clock tree starting from a root pin. The tracing begins at the root pin, then continues through the buffers, inverters, multi-output cells, and gated instances to establish the clock tree. The tracing stops at the following:

- A clock pin
- An asynchronous set/reset pin

- An input pin without any timing arc to an output pin
- A user-specified leaf pin or excluded pin
- Data pin of registers (or flops)
- Asynchronous set or reset pin of registers (or flops)
- Enable pins of tristate instances

After the tracing, CTS builds the clock buffer tree topology to balance the clock phase delay with inserted clock buffers.

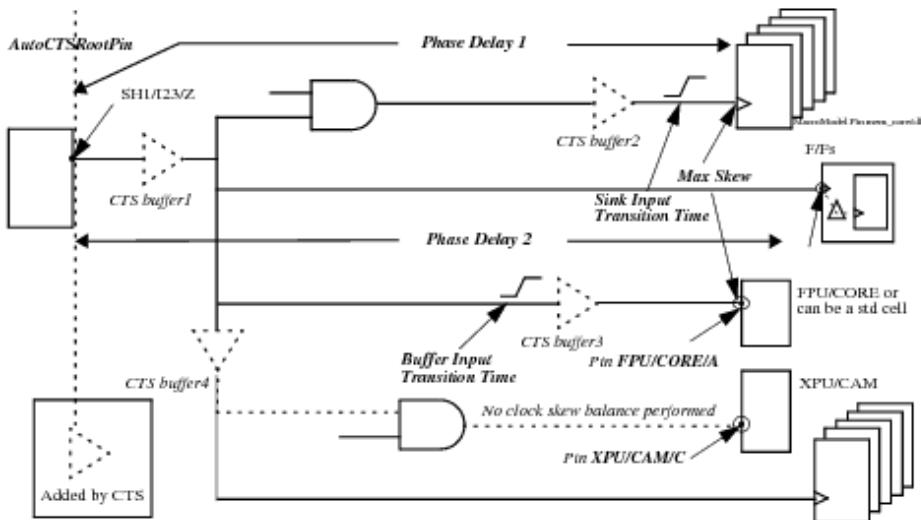
**Note:** Cadence recommends using the `ckSynthesis -check` command to check the gated clock tree of your design before running automatic gated CTS mode. After running the command, review the trace report file, `topCellName.cts_trace`. If tracing fails, the `-forceReconvergent` parameter of the `ckSynthesis` command can be used to resolve tracing failures.

An example of clock tree specification file syntax for automatic CTS on a gated clock and a graphic representation of the syntax is shown below:

```

AutoCTSRootPin SH1/I23/Z
MaxDelay 5ns
MinDelay 0ns
MaxSkew 500ps
MaxDepth 20
NoGating NO
RouteType CK1
LeafPin
+ FPU/CORE/A rising
ExcludedPin
+ XPU/CAM/C
PreservePin
+ pinA
+ pinB
MaxCap
+ buf1 20FF
+ buf2 50FF
Buffer buf1 buf2 invl inv2 dell
End

```



## How CTS Calculates Skew Values

CTS calculates skew at the edge of the clock root in the following manner:

- *Rise skew* and *fall skew* are calculated relative to the edge of the clock root. For example, rise skew is calculated based on the rising edge at the clock root.
- **Note:** The edge polarity at the leaf pins can be rising or falling, regardless of whether CTS is reporting on rise skew or fall skew.
- *Rise skew* is the maximum difference of all the arrival times of the clock signal at the leaf inputs, as measured from a rising edge at the clock root.
- *Fall skew* is the maximum difference of all the arrival times of the clock signal at the leaf inputs, as measured from a falling edge at the clock root.

- *Trigger-edge skew* is based on all the arrival times of the active clock signal at the leaf inputs. The calculation considers the trigger-edge polarity of the receiving leaf inputs, and represents the worst-case trigger-edge-to-trigger-edge skew in the design. See the accompanying figure.

**Note:** Trigger-edge skew can be greater or smaller than rise skew or fall skew.

The following example illustrates how CTS calculates various skew values:

Assume that a design has two flip-flops, FF1 and FF2:

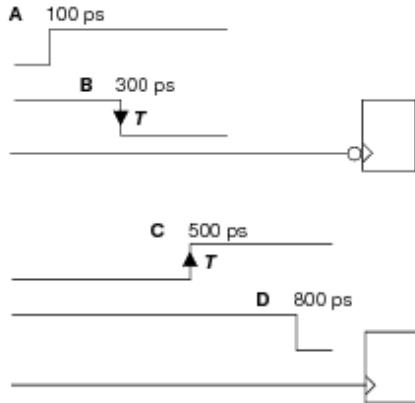
- FF1 rise: 3.0 ns; fall: 3.6 ns
- FF2 rise: 3.4 ns; fall: 4.0 ns

Rise skew is 0.4 ns; fall skew is 0.4 ns.

Assume that FF1 is a falling-edge-triggered flip-flop, and that FF2 is a rising-edge-triggered flip-flop. The trigger-edge skew is 3.6 ns - 3.4 ns = 0.2 ns.

Assume that FF1 is a rising-edge-triggered flip-flop, and that FF2 is a falling-edge-triggered flip-flop. The trigger-edge skew is 4.0 ns - 3.0 ns = 1.0 ns.

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



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

## Improving PostRoute Correlation

You can create a routing guide file that CTS automatically uses to direct the global detailed routing of clock nets. This process helps achieve tighter correlation between preRoute (Steiner tree) and postRoute topologies.

There are two methods of improving postRoute correlation with a routing guide file. In **Method 2**, CTS reports on the clock tree *before* you complete the detailed routing on the design. The flow for both methods is as follows:

## Method 1

1. In the Innovus console, type `setCTSMODE -routeGuide true`.
2. In the clock tree specification file, include `RouteClkNet YES`.
3. In the Innovus console, type `ckSynthesis`.
4. Check your run directory for the clock tree timing report (`top_level_cell.ctsrpt`).

## Method 2

1. In the clock tree specification file, include `RouteClkNet NO`.
2. In the Innovus console, type `ckSynthesis`.
3. In the Innovus console, type `routeClockNetWithGuide [-clk clock_root_pin_name]`
4. Check your run directory for the clock tree timing report (`top_level_cell.ctsrpt`).

## Specifying Macro Model Delays

You can use the `MacroModel` statement to specify pin delays. A macro model is a block with synthesized clock trees, and therefore, has delays that have been identified.

There are three ways to define the macro model:

- Cell or port delay specification: All instantiations of cells have the same pin delay.

```
MacroModel port cellName/portName maxRiseDelay minRiseDelay  
maxFallDelay minFallDelay extraCap
```

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

```
MacroModel port ram256x64/clk 10ns 80ns 110ns 7ns 0.35ff
```

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

```
MacroModel pin leafPinName maxRiseDelay minRiseDelay
```

*maxFallDelay minFallDelay extraCap*

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

```
MacroModel pin mem_pin/clk 20ps 18ps 20ps 18ps 0.29ff
```

**Note:** The delay units for MacroModel statements must be specified in nanoseconds (ns) or picoseconds (ps), for example, 200ps, 1ns.

- INSERTION\_DELAY statement in the TLF file.

```
INSERTION_DELAY(CLK FAST 01 01 DELAY(InsDelay0) SLEW(InsSlew1))  
INSERTION_DELAY(CLK SLOW 01 01 DELAY(InsDelay0) SLEW(InsSlew1))
```

For information on TLF, see the *Timing Library Format Reference* manual.

For illustrations of MacroModel behavior, see "Automatic CTS Mode" and "Automatic Gated CTS Section".

## Macro Model Support for MMMC Views

Macro models support MMMC views.

- The cell-based macro models are specified as:

```
MacroModel port cellName/portName maxRiseDelay minRiseDelay maxFallDelay minFallDelay  
extraCap viewName
```

- The instance-based macro models are specified as:

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

**Note:** The view names are the analysis views specified by the MMMC setting.

For MMMC setup, if you do not specify a view name, the macro model is applied to the default setup view while all other views are applied with some auto-scaling according to the ratio of the cell delay of each view to the default setup view. If MMMC is not enabled and view name is specified in the macro model, then the tool displays an error message on specifying the view name.

## Example

Consider the following statements:

```
MacroModel pin inst1/CK 0.4ps 0.4ps 0.4ps 0.4ps 0pf
MacroModel pin inst1/CK 0.3ps 0.3ps 0.3ps 0.3ps 0pf view2
```

In the above statements, one macro model statement for a pin is specified without any view and another statement for the same pin is specified with a view name, `view2`. Assume that there are three active views: `view1`, `view2`, and `view3` and `view 1` is the default setup view. Then `view1` will get the value from first statement because it is the default view, `view2` will get the delay value of the second statement because it is clearly specified, and `view3` will get a scaled delay value.

Therefore, the tool interprets it as :

```
MacroModel pin inst1/CK 0.4ps 0.4ps 0.4ps 0.4ps 0pf view1
MacroModel pin inst1/CK 0.3ps 0.3ps 0.3ps 0.3ps 0pf view2
MacroModel pin inst1/CK 0.4ps 0.4ps 0.4ps 0.4ps 0pf view3
```

## Dynamic Macro Model

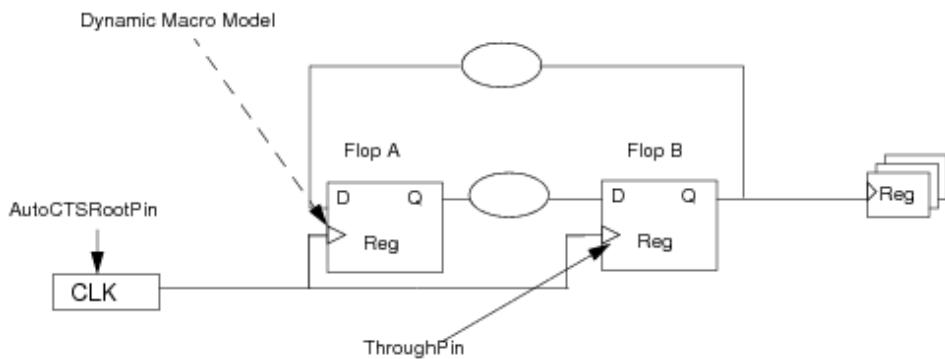
A dynamic macro model is used to minimize the skew between the reference pin and the target pin during CTS. The reference pin is a clock instance pin along a clock path. The target pin must be a leaf pin.

```
DynamicMacroModel ref refInstPinName pin targetInstPinName [offset delayNumber{ns|ps}]]
```

where `refInstPinName` is the reference instance pin name, `targetInstPinName` is the target instance pin name, and `delayNumber` specifies the offset arrival delay in nanoseconds or picoseconds.

As an example, the `DynamicMacroModel` statement can be used when your design contains clock dividers. The following figure contains two flops, A and B. A `ThroughPin` has been defined in the clock pin of Flop B. So, the clock pin of Flop A is balanced with the group of flops and not with the clock pin of Flop B because of the `ThroughPin` that has been defined in Flop B. Using a dynamic macro model in Flop A, you can balance the skew between the two flops. You can then specify clock pin of Flop B as a reference pin and clock pin of Flop A as the target pin so that the clock pin of flop A is balanced with the clock pin of flop B. The `DynamicMacroModel` statement minimizes the skew between these two flops to avoid timing violation on the data path.

The following figure illustrates how the skew is minimized between the reference pin and the target pin using dynamic macro model.



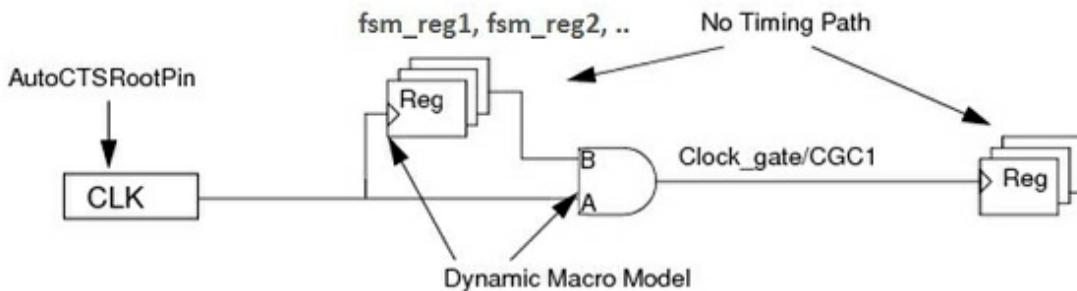
If there are multiple `DynamicMacroModel` statements where the target pin is referenced to more than one reference pin, the latter overrides the previous statement.

The offset parameter is optional. Instead of minimizing the difference between the arrival delays at the reference and the target pins, the `offset` parameter allows you to specify the offset arrival delay for the target pin. The offset arrival delay can be a positive or a negative value. A positive `offset` number indicates a shorter clock path for the target pin as compared to that of a reference pin. The default value of the `offset` parameter is 0 .

## Example

In the following example, a clock tree is built by minimizing the skew between the reference pin (`Clock_gate/CGC1/CLK`) and the target pins (`fsm_reg1/CLK`, `fsm_reg2/CLK`, and so on).

```
...
DynamicMacroModel ref Clock_gate/CGC1/A pin fsm_reg1/CLK offset 1ps
DynamicMacroModel ref Clock_gate/CGC1/A pin fsm_reg2/CLK offset 1ps
...
```



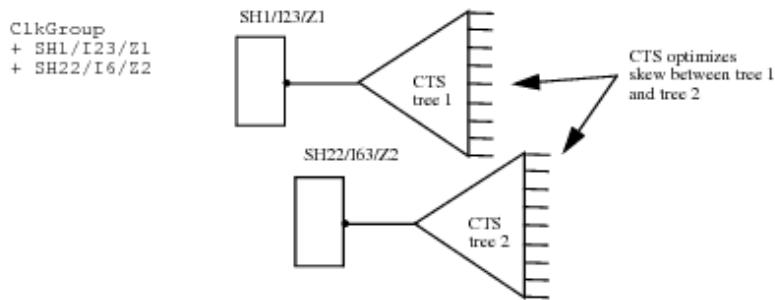
## Grouping Clocks

Clock grouping is available in automatic CTS mode. All clock root pin names entered into a clock group that will have their sinks meet the maximum skew as specified in the clock tree specification file. CTS balances the clock tree roots as if they were one tree.

The sinks of all clock root pins listed in a `ClkGroup` statement will meet the maximum skew value set in the clock tree specification file. Clock grouping inserts delays to balance the clocks, and attempts to meet clock skew for all clocks.

**Note:** You can define more than one clock group in the clock tree specification file.

The following is an example of clock group syntax and its graphical representation:



**Note:** All `ClkGroup` statements must be specified in lines following macro model line(s), and before any clock specification.

## Analyzing Hierarchical Clock Trees

Innovus designs clock trees in a two-step, bottom-up fashion.

Within Innovus, the designing of the clock tree is done bottom-up in two steps. After partitioning the design, you can run CTS on each partition individually. Once the partitions are synthesized, the top-level partition runs CTS hierarchically. So CTS runs at the top-level partition, and the partitions' clock tree results are treated as macro model instances.

To generate the partition macro models, use the *Synthesize Clock Tree* form from the *Clock* menu or use the following command when running CTS for the partition:

```
ckSynthesis -macromodel fileName
```

The rise time, fall time, and input capacitance for the clock pins are characterized, and the `fileName` output model file is used when creating the top-level partition's clock tree specification file. Running CTS for the top-level partition balances the clock phase delay between the top-level and the partitions.

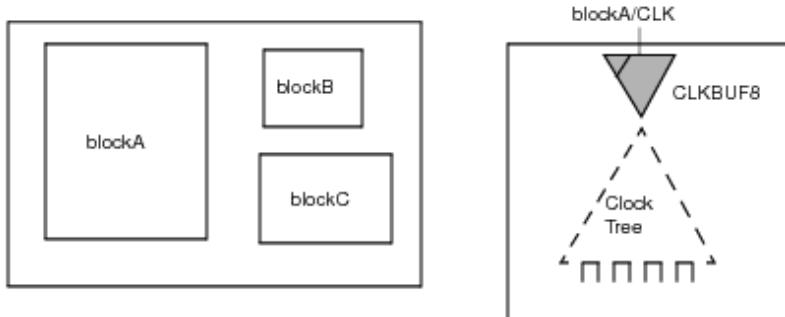
- i** The macro model specifications for each partition are at the top of the clock tree specification file.

For example, in a design with three partitions (`blockA`, `blockB` and `blockC`), you should first synthesize the partitions individually. To run CTS on the partition's blocks, you should add the `AddDriverCell` statement in the clock tree specification file. Use the `AddDriverCell driver_cell_name` statement for block-level CTS to place a driver cell name at the closest possible location to the clock port location. For example:

```
AutoCTSRootPin blockA/clk
...
...
AddDriverCell CLKBUF8
End
```

**Note:** If the clock root is an instance pin and not a primary input port, then the `AddDriverCell` command is not honored.

CTS adds buffer `CLKBUF8` after the input pin, as shown in the following figure:



After running CTS on the blocks, run CTS on the top level of the design. To run top-level CTS, you must include all the macro models from block-level CTS in the clock tree specification file.

- i** If your top-level design has a large amount of blockage and is limited in routing resources, you should add the `Obstruction Yes` statement in the clock tree specification file. That statement instructs CTS to run the detail maze router to detect the obstruction (which increases CTS runtime). Use this statement only when routing resources are extremely limited, such as in top-level CTS.

The following example shows the `Obstruction Yes` command in the clock tree specification file:

```
MacroModel port blockA/clk 900ps 800ps 900ps 800ps 17ff
MacroModel port blockB/clk 1100ps 1000ps 1100ps 1000ps 18ff
MacroModel port blockC/clk 500ps 400ps 500ps 400ps 19ff
AutoCTSRootPin clk
....
...
Obstruction Yes
End
```

## Module Placement Utilization

Ensure that the modules' placement utilization, which contains the clock nets, is set to 5-7 percent less than the desired final chip utilization (placement density). This provides placement resources for adding clock buffers during CTS.

## Clock Designs with Tight Area

For a clock design that is limited to a tight area, use the *Specify Cell Padding* form (*Place - Specify - Cell Padding*) to create placement resources near clocked flip-flop cell types.

## Balancing Pins for Macro Models

CTS can balance a pin of a macro model. These macro models are user specified. CTS balances the phase delay of all leaf pins in the clock tree, including leaf pins of macro models.

The timing models for macro models are defined in the clock tree specification file `MacroModel` statement.

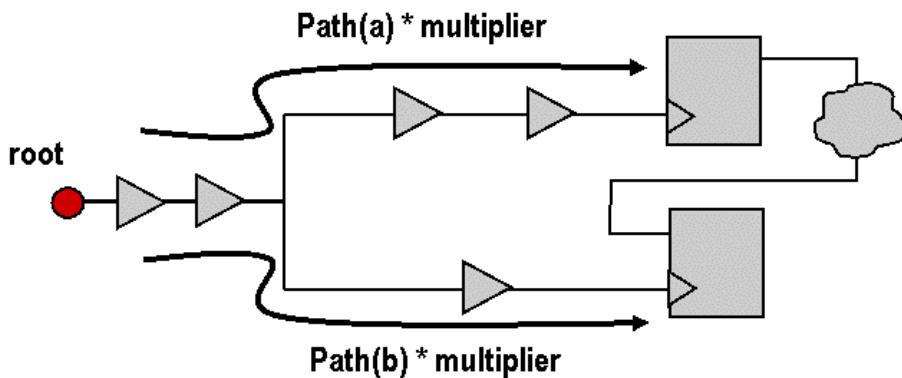
## Timing Model Requirement for Cells

Ensure that all cells have a timing model. If a cell does not have a timing model, CTS will not trace through the gate, and may set the gate's input pin as a leaf pin.

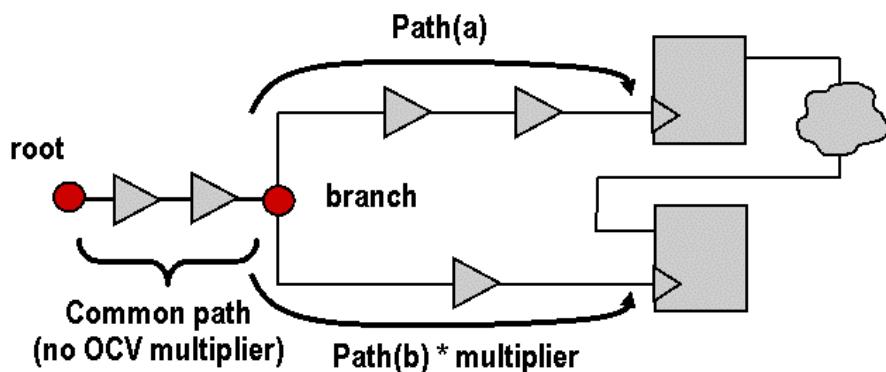
## Delay Variation and OCV

CTS can analyze your design for delay variation and on-chip variation (OCV). CTS identifies adjacent registers and then calculates the delay, using derating factors that you specify with the `setAnalysisMode` and `set_timing_derate` commands.

The following diagram illustrates how CTS applies wire and cell derating factors to the whole path in a clock tree, including the common path.



The following diagram illustrates how CTS applies cell and wire derating factors only to the paths after a branch point - and not to the common path.



# Understanding PostCTS Clock Tree Optimization

## Using the ckECO Command for PostCTS Clock Tree Optimization

Use the `ckECO` command for postCTS optimization of clock tree(s) either in the same Innovus session as that in which the `ckSynthesis` command was run, or in a new session. The sole aim of the `ckECO` command is to improve the skew of each clock and clock group, and to resolve minimum phase delay violations. The `ckECO` command does not attempt to correct any design rule violations. However, in trying to improve skew, the `ckECO` command does not significantly worsen maximum transition or maximum capacitance violations.

The `ckECO` command performs resizing and buffer insertion or dummy buffer insertion to improve skew. In addition, the `ckECO` command might move gating cells when the `ckECO` command runs `refinePlace`.

## Support for Local Skew Optimization

The `ckECO` command also supports local skew optimization (with the `-localSkew` parameter). Local skew optimization considers the skew between adjacent flip-flops that have data path connection (from a Q-pin of one flip-flop to the D-pin of another flip-flop).

Load the timing constraints into the Innovus session (design import stage) when you want to perform local skew optimization.

At a minimum, the clock roots need to be defined in the SDC file so the `ckECO` command can identify the adjacent register pairs.

## Command Modes for the ckECO Command

The `ckECO` command can be run in four modes:

- `ckECO -preRoute`
- `ckECO -clkRouteOnly`
- `ckECO -postCTS`
- `ckECO -postRoute`

**Note:** The `ckECO -postCTS` is run by default during `clockDesign`.

It is important to choose the correct mode, based on the state of the design. Otherwise CTS might use the wrong RC model, which can lead to quality-of-result (QOR) and correlation problems.

## Using a SPEF File with the ckECO Command for RC Estimation

As an alternative to using CTS estimation for RCs it is possible to load a SPEF file.

If you use an external SPEF file (`spefIn`) you just use the `ckECO -postRoute` command, and CTS does *not* create a clock tree report after the `ckECO -postRoute` process is done. You will need to re-extract the RC values for all the wires.

After you create a new SPEF file, load this new file into the Innovus session. Then run `reportClockTree -postRoute`. CTS then creates the clock report.

If clock nets are in a routed state you run the `ckECO` command, any wires that are disturbed by the `ckECO` command will automatically be rerouted using NanoRoute in an ECO mode.

## Running PostCTS Optimization with the ckECO Command

You must load the clock tree specification file to run the `ckECO` command. (The clock tree specification file defines the clocks that you want to optimize.)

Whether the `ckECO` command performs resizing or ECO routing depends on these clock tree specification file or the `setCTSMODE` settings:

- `RouteClkNet YES | NO`
- `setCTSMODE -routeClkNet {true | false}`
- `PostOpt YES | NO`
- `setCTSMODE -opt {true | false}`

The following examples show the usage of the `ckECO` command for two general design states.

### Example 1

If your design has the following characteristics:

- Clock tree is already inserted
- Clocks are routed

- Signal nets are not routed

Then use this command sequence:

```
specifyClockTree
```

```
ckECO -clkRouteOnly
```

## Example 2

If your design has the following characteristics:

- Clock tree is already inserted
- Clock nets are not routed
- Signal nets are not routed

Then use this command sequence:

```
specifyClockTree
```

```
ckECO
```

**Note:** When you specify the `ckECO` command without a parameter, you instruct CTS to use the default mode for the command, which is `-preRoute`.

## Guidelines for Using the `ckECO` Command

- If the design contains signal routes routed by Trial Route or NanoRoute, you must specify `ckECO -postRoute`; otherwise, the software generates an error message and stops. When you specify `ckECO -postRoute` on designs with trial-routed nets, the software removes any trial-routed nets before running the command. After the command has run, the software calls trial route to replace the trial-routed nets.
- Note:** The new trial-routed nets are *not* guaranteed to be identical to those before you ran the `ckECO` command.
- By default, the `ckECO` command inserts a maximum of 50 buffers. The number of buffers is controlled by the `OptAddBufferLimit` statement in the clock tree specification file.
- Often it is possible to make incremental improvements to skew by running the `ckECO` command multiple times.

For details on the `ckECO` command and its parameters, see Legacy FE-CTS Commands chapter in *Innovus Text Command Reference*.

## Creating a Clock Tree Specification File

Before you can run CTS, you must create a clock tree specification file by:

- Using the *Generate Clock Spec* form (choose *Clock - Synthesize Clock Tree - Gen Spec...*)
- Using the `createClockTreeSpec` command
- Using the `specifyClockTree` command with the `-template` parameter

The `specifyClockTree` command creates the basic clock tree specification file template file `template.ctstch` in the directory in which you are running the Innovus software. Edit the template file with any text editor.

This file contains the following information on the clock or clocks you want to analyze with CTS:

- Timing constraint file (optional)
- Naming attributes (optional)
- Macro model data (optional)
- Clock grouping data (optional)
- Attributes used by NanoRoute™ Ultra SoC routing solution (optional)
- Requirements for manual CTS or automatic, gated CTS

You can create a clock tree specification file for all the clocks in your design, for a subset of clocks in your design, or for a single clock.

 The general sections of the clock tree specification files must appear in the order given above. However, the individual statements within each section can appear in any order.

## Using the Automatic Clock Tree Specification File Generator

You can specify `setCTSMode -specMultiMode true`, to enable the automatic clock tree specification file generator, which generates clock tree specification files that are more timing aware than regular specification files.

The auto clock tree specification file generator can be used only with the `createClockTreeSpec` and the `clockDesign` command. By specifying `createClockTreeSpec`, to only generate timing-aware clock tree specification file. By specifying `clockDesign` on its own, to generate the specification file, and then running CTS.

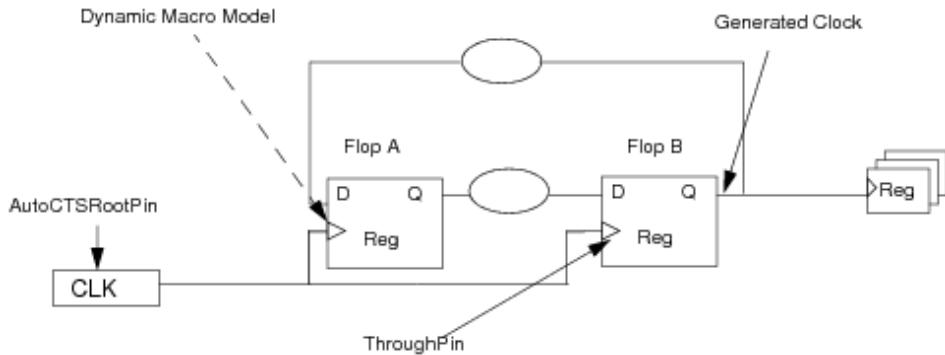
The auto clock tree specification file generator works differently depending on whether the software is in multi-mode multi-corner (MMMC) analysis mode or not.

If you specify the following commands when the software is not in MMMC analysis mode:

```
setCTSMode -specMultiMode true addCTSCellList "BUFX2 BUFX4 BUFX8"
```

The auto clock tree spec generator does the following:

- Generates a spec file with dynamic macro models to help close on timing issues between the flops (which function as clock dividers) that are responsible for providing the generated clock. It adds dynamic macro models when appropriate.



For more information about SDC-driven CTS, see `createClockTreeSpec`.

If you specify the same commands when the software is in MMMC analysis mode, the auto clock tree spec generator generates multiple specification files as above, using the naming convention `fileName . modeName`.

If you run `clockDesign` without any parameter, it generates specification files with merging all the views, and also performs the following CTS functions on all the active analysis views:

- Creates and loads a clock tree specification file for each mode.
- Runs CTS.

If you want to run `clockDesign` view by view, run the following:

```
createClockTreeSpec -view;  
clockDesign - [-specViewList {{spec1 view1 view2 ...} \  
{ spec2 view1 view2 ...}...}]
```

The `createClockTreeSpec` command creates and loads a clock tree specification file for each mode. The `clockDesign` command:

- Runs CTS
- Determines which instances of the synthesized clock tree to preserve for building the next clock tree
- Removes the clock tree specification file from the memory

For more information, see `createClockTreeSpec` and `clockDesign`.

## Creating the CTS Specification File in the MMMC Mode

To run CTS in the MMMC mode, follow below guidelines:

1. Use one merged CTS specification file for all modes
  - If you have an existing CTS specification file, use the following commands:  
`set_analysis_view -setup (or -hold)`  
`setCTSMode -specMultiMode false (which is the default value)`  
`clockDesign -specFile <merged_spec>`
  - If you do not have a CTS specification file, use the following commands:  
`set_analysis_view -setup (or -hold)`  
`setCTSMode -specMultiMode false`  
`clockDesign`
1. Use one specification file for each mode
  - If you already have all specification files, use the following commands:  
`set_analysis_view -setup (or -hold)`  
`setCTSMode -specMultiMode true`  
`clockDesign -specViewList {{spec1 view1 view2...} {spec2 view3 view4...}...}`

- If you do not have any specification file, use the following commands:

```
set_analysis_view -setup (or -hold)
setCTSMode -specMultiMode true
clockDesign
```

## Example of a Clock Tree Specification File

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

```
UseSingleDelim YES
NameDelimiter |
MacroModel pin freq/mod004048/CLK 20ps 18ps 20ps 18ps 0.29ff
UseCTSRouteGuide NO
#Start RouteTypeName section
RouteTypeName CK1
NonDefaultRule rule1
PreferredExtraSpace 1
TopPreferredLayer 5
BottomPreferredLayer 4
Shielding VSS
End
#End RouteTypeName section
#Example of manual CTS specification information
ClockNetName CK
LevelNumber 2
LevelSpec 1 1 BUFX2
LevelSpec 2 2 BUFX2
PostOpt YES
OptAddBuffer YES
End
LeafPinGroup grp1
+ a/b/c/a_reg/CK
+ c/d/e/g_reg/CK
```

```
GlobalPowerDomainBuffer
+ pd1 buf1 buf2
+ pd2 buf3 buf4
GlobalUnsyncPin
+ clk_div_reg1/CKN
GlobalUnsyncPort
+ DFFNSRX4/CKN
GlobalThroughPort
+ DFFNSRX1/Q
GlobalPreservePort
+ DFFNSRX4/CK
AutoCTSRootPin cgen/i_5/Y
MaxDelay 5.0ns
MinDelay 0ns
MaxFanout 30
MaxSkew 250ps
SinkMaxTran 550ps
BufMaxTran 550ps
SetBBoxPinAsSync true
RootInputTran XYZ
NoGating NO
DetailReport YES
Obstruction YES
RouteType CK1
LeafRouteType specialRoute
CellRouteType
+ BUFX2 routeName1
+ BUFX4 routeName2
CellLeafRouteType
```

```
+ BUFX8 routeName3
+ BUFX12 routeName4
RouteClkNet YES # Turns on NanoRoute. The default value is NO
PostOpt YES # Turns on optimization. The default value is YES
OptAddBuffer YES
OptAddBufferLimit 100
Buffer BUFX2 BUFX4 BUFX8 BUFX12 INVX1
MaxCap
+ BUFX2 1pf
+ BUFX4 1pf
+ BUFX8 1pf
+ BUFX12 1pf
ForceMaxTran NO
ThroughPin
+ df/mod000446/CK Y
ThroughPort
+ df/mod002300/ax2
LeafPin
+ PCLK66_gate_i/A rising
LeafPort
+ ssfd2s/D rising
KeepPortPolarity
+ MOD1/PORT1
PreservePin
+ cgen/mod000043/A
ExcludedPin
+ freg/mod004048/CLK
ExcludedPort
+ DFF_B/CLK
```

GatingGrpInstances

- + cg1/an cg1/i\_0
- + cg2/an cg2/i\_0
- + cg1/an cg3/i\_0
- + cg4/an cg4/i\_0

GatingGrpModule

- + grp\_module1
- + grp\_a\*

CellHalo

+BUFX411

End

MasterGateTargetFanout32

CellObstruction

- + ram128x32 Detailed

InstanceObstruction

- + cpu/itag/ram2 Ignored

AutoCTSRootPin clk

MaxDelay 3.1ns

MinDelay 0ns

MaxSkew 100ps

ForceMaxFanout YES

SkipRouteGuide YES

SinkMaxTran 200ps

BufMaxTran 200ps

Buffer CLKBUFX4

NoGating NO

MasterGateInst

- + 7 clk\_latch

End

```
AutoCTSRootPin clk
MaxDelay 3.1ns
MinDelay 0ns
MaxSkew 100ps
SinkMaxTran 200ps
BufMaxTran 200ps
Buffer CLKBUFX4
NoGating NO
EquivGateInst
+ G1
+ G2
EquivGateInst
+ clk_and1 clk_neg1
+ clk_and2 clk_neg2
```

## Naming Attributes Section

The following table describes the entries for the naming attributes section:

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NameDelimiter<br><i>delimiter</i> | <p>Allows you to customize the name delimiter that CTS uses when inserting buffers and updating clock root and net names. There are no restrictions on the type of characters that you specify for the name delimiters in the clock tree specification file but you must specify only a single character. For example:</p> <p>NameDelimiter # creates names with the format <code>clk##L3#I2</code>, rather than the default format, <code>clk__L3_I2</code>.</p> <p>Insert the <code>NameDelimiter</code> statement after <code>MacroModel</code> and <code>ClkGroup</code> statements but before an <code>AutoCTSRootPin</code> statement.</p> <p><b>Note:</b> If you have multiple <code>NameDelimiter</code> statements in the clock tree specification file, CTS uses only the last <code>NameDelimiter</code> statement in the file.</p> <p><b>Note:</b> The <code>NameDelimiter</code> and <code>UseSingleDelim</code> statements are independent of each other.</p> |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>UseSingleDelim</b><br>YES   NO | <p>Instructs CTS whether to use single name delimiters after the first element in clock root and net names. For example:</p> <p><code>UseSingleDelim YES</code> creates clock and net names with the format <code>clk_L3_I2</code>, rather than the default format, <code>clk__L3_I2</code>.</p> <p>By default, CTS always inserts double (or, in some cases, multiple) name delimiters after the first element of clock root or net names. The <code>UseSingleDelim YES</code> statement overrides this behavior by instructing CTS to use <i>only</i> a single delimiter after the first element of the name:</p> <p>Insert the <code>UseSingleDelim</code> statement after <code>MacroModel</code> and <code>ClkGroup</code> statements but before an <code>AutoCTSRootPin</code> statement.</p> <p><b>Note:</b> The <code>UseSingleDelim</code> and <code>NameDelimiter</code> statements are independent of each other.</p> |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## NanoRoute Attribute Section

The following table describes the entries for the section of the clock tree specification file that deals with attributes that CTS passes to NanoRoute Ultra.

|                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>UseCTSRouteGuide</b><br>YES<br>  NO | <p>Specifies whether NanoRoute should route clock nets with the routing guide file generated by CTS.</p> <p>NanoRoute will produce better pre- and postroute correlations, though at the expense of longer run time.</p> <p>If you specify <code>UseCTSRouteGuide YES</code> and <code>RouteClkNet YES</code>, clock nets will be routed based on the route guide file created by CTS.</p> <p>If you specify <code>UseCTSRouteGuide YES</code> and <code>RouteClkNet NO</code>, CTS creates a route guide file. But because you instructed CTS not to route the clock nets, the guide file is not used. If you subsequently use the <code>routeClockNetWithGuide</code> command, CTS generates a new routing guide file that is based on the synthesized clock structure, and uses it to automatically route the clocks.</p> |
| <b>RouteTypeName</b> <i>name</i>       | <p>Specifies the routing type for which you are defining routing attributes.</p> <p><b>Note:</b> <code>RouteTypeName</code> statements must appear in the clock tree specification file before their corresponding <code>RouteType</code> statements.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

|                                       |                                                                                                                                                                                        |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NonDefaultRule<br><i>ruleName</i>     | Specifies the LEF <code>NONDEFAULTRULE</code> statement that the router should use.<br><br><i>Default:</i> If you do not use this statement, the router uses the default routing rule. |
| PreferredExtraSpace<br>[0 - 3]        | Specifies the spacing attribute, with which to add space around clock wires.<br><br><i>Default:</i> If you do not use this statement, CTS uses a preferred extra space value of 1.     |
| TopPreferredLayer<br><i>number</i>    | Specifies the top-most preferred routing layer.<br><br><i>Default:</i> 4                                                                                                               |
| BottomPreferredLayer<br><i>number</i> | Specifies the bottom-most preferred routing layer.<br><br><i>Default:</i> 3                                                                                                            |
| Shielding <i>GNetName</i>             | Defines the ground net name.                                                                                                                                                           |
| End                                   | Marks the end of the NanoRoute Ultra attribute section.                                                                                                                                |

- i** In all clock tree specification file sections that contain delay values or capacitance values, you *must* include the appropriate unit with the values you specify. The unit designations are case-insensitive. For example, `pF`, `pF`, `Pf`, and `PF` are all valid unit designations for picofarads.

## Macro Model Data Section

The following table describes the entries for the macro model port data section:

| MacroModel port <i>cellName_or_portName delay_and_capacitance_values</i> |                                                                 |
|--------------------------------------------------------------------------|-----------------------------------------------------------------|
|                                                                          | Specifies the name of the macro model cell or port.             |
| <i>maxRiseDelay</i> {ns   ps}                                            | Specifies the maximum rise delay in nanoseconds or picoseconds. |
| <i>minRiseDelay</i> {ns   ps}                                            | Specifies the minimum rise delay in nanoseconds or picoseconds. |
| <i>maxFallDelay</i> {ns   ps}                                            | Specifies the maximum fall delay in nanoseconds or picoseconds. |
| <i>minFallDelay</i> {ns   ps}                                            | Specifies the minimum fall delay in nanoseconds or picoseconds. |

The following table describes the entries for the macro model pin data section:

| MacroModel pin <i>pinName</i> <i>delay_and_capacitance_values</i> |                                                                 |
|-------------------------------------------------------------------|-----------------------------------------------------------------|
|                                                                   | Specifies the name of the macro model pin.                      |
| <i>maxRiseDelay</i> {ns ps}                                       | Specifies the maximum rise delay in nanoseconds or picoseconds. |
| <i>minRiseDelay</i> {ns ps}                                       | Specifies the minimum rise delay in nanoseconds or picoseconds. |
| <i>maxFallDelay</i> {ns ps}                                       | Specifies the maximum fall delay in nanoseconds or picoseconds. |
| <i>minFallDelay</i> {ns ps}                                       | Specifies the minimum fall delay in nanoseconds or picoseconds. |

The following table describes the entries for the global directive section.

**(i)** Global directive statements can not be used between `AutoCTSRootPin` and `End` statements.

|                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                           |  |        |                                                                          |  |         |                                                                           |
|-------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|--|--------|--------------------------------------------------------------------------|--|---------|---------------------------------------------------------------------------|
| GlobalLeafPin<br>+ <i>pinName</i> rising   falling<br>+ ... | <p>Marks the input pin as a "leaf" pin for non-clock-type instances globally (throughout the design), stops tracing, and balances clock skew.</p> <p><b>Note:</b> Use the <code>GlobalLeafPin</code> statement <i>only</i> with input pins. CTS ignores <code>GlobalLeafPin</code> statements that are associated with output pins.</p> <p>Choose one of the following:</p> <table border="1"> <tr> <td></td><td>rising</td><td>CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.</td></tr> <tr> <td></td><td>falling</td><td>CTS treats the input pin as a falling-edge-triggered flip-flop clock pin.</td></tr> </table> |                                                                           |  | rising | CTS treats the input pin as a rising-edge-triggered flip-flop clock pin. |  | falling | CTS treats the input pin as a falling-edge-triggered flip-flop clock pin. |
|                                                             | rising                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.  |  |        |                                                                          |  |         |                                                                           |
|                                                             | falling                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | CTS treats the input pin as a falling-edge-triggered flip-flop clock pin. |  |        |                                                                          |  |         |                                                                           |
|                                                             | rising                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.  |  |        |                                                                          |  |         |                                                                           |
|                                                             | falling                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | CTS treats the input pin as a falling-edge-triggered flip-flop clock pin. |  |        |                                                                          |  |         |                                                                           |

|                                                                                 |                                                                                                                                                              |
|---------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>GlobalLeafPort + <i>cellType/inputPinName</i> rising   falling + ...</pre> | <p>Marks the pin as a "leaf" pin for cells globally (throughout the design), stops tracing, and balances clock skew.</p> <p>Choose one of the following:</p> |
|                                                                                 | <p><b>rising</b> CTS treats the pin as a rising-edge-triggered flip-flop clock pin.</p>                                                                      |
|                                                                                 | <p><b>falling</b> CTS treats the pin as a falling-edge-triggered flip-flop clock pin.</p>                                                                    |
| <pre>GlobalExcludedPin + <i>pinName</i> + ...</pre>                             | <p>Treats the pin as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.</p>                                 |
| <pre>GlobalExcludedPort + <i>cellType / inputPinName</i> + ...</pre>            | <p>Treats the pin of particular cell type as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.</p>         |
| <pre>GlobalUnsyncPin +<i>PinName</i></pre>                                      | <p>Treats the pin as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.</p>                                 |
| <pre>GlobalUnsyncPort +<i>cellType/inputPinName</i></pre>                       | <p>Treats the pin of particular cell type as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.</p>         |

|                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GlobalThroughPort<br><pre>+cellType/<i>PinName</i> [<i>outputPinNames</i>]</pre> | <p>Traces through the pin of a particular cell type globally (throughout the design), even if the pin is a clock pin.</p> <p>By default, when a <code>ThroughPin</code> is specified at the input pin of multi-output cells, it traces through all the outputs as long as there is valid timing arc from the input pin to the output pin. If you want CTS to trace through a certain output pin, you must use <code>outputPinName</code> to instruct CTS which output pin to trace through.</p> <p>For example, if you want CTS to trace through the output pin <code>Q</code> of library cell <code>DFF</code>, specify the following in the clock tree specification file:</p> <pre>GlobalThroughPort + DFF/Q</pre> |
| GlobalPreservePort<br><pre>+cellType/<i>inputPinName</i></pre>                   | <p>Preserves the netlist for the pin of a particular cell type and pins below the pin in the clock tree globally (throughout the design). However, CTS considers any synchronized pins after the pin when computing skew.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

|                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>GlobalThroughPin + <i>pinName</i> [<i>outputPinNames</i>] + ...</pre> | <p>Traces through the pin globally (throughout the design), even if the pin is a clock pin.</p> <p>By default, when a <code>ThroughPin</code> is specified at the input pin of multi-output cells, it traces through all the outputs as long as there is valid timing arc from the input pin to the output pin. If you want CTS to trace through a certain output pin, you must use <code>outputPinName</code> to instruct CTS which output pin to trace through.</p> <p>For example, if you want CTS to trace through an instance <code>top/mmcore</code> from the input pin <code>clk</code> to <code>clk01</code> and <code>clk02</code>, although this instance has another clock output <code>clk03</code>, specify the following in the clock tree specification file:</p> <pre>ThroughPin + top/mmcore/clk clk01 clk02</pre> |
| <pre>GlobalPreservePin + <i>inputPinName</i> + ...</pre>                   | <p>Preserves the netlist for the pin and pins below the pin in the clock tree globally (throughout the design). However, CTS considers any synchronized pins after the pin when computing skew.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

|                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>GlobalValidCell + cellName1 + cellName2 + ... </pre> | <p>Specifies all cells that can be used by CTS as <code>GlobalValid</code> cell in the clock tree specification file. The software issues a warning if any intermediate clock instance (for example, MUX, clock gaters, and buffers along clock path) is found to not conform to the <code>GlobalValidCell</code> list. Once a cell type is specified as <code>GlobalValidCell</code>, CTS ignores the <code>dont_touch</code> and <code>dont_use</code> attributes.</p> <p>This statement is honored by the <code>ckSynthesis</code>, <code>ckECO</code>, <code>ckCloneGate</code>, <code>ckDecloneGate</code>, and <code>deleteClockTree</code> commands.</p> <p><b>Note:</b> The <code>GlobalValidCell</code> statement must be specified outside the <code>AutoCTSRootPin</code> and <code>End</code> statement.</p> |
| <pre>DontTouchNet + netName1 + netName2 + ... </pre>      | <p>Specifies a list of nets for which the <code>ckSynthesis</code> and <code>ckEco</code> commands should not insert buffers. The <code>deleteClockTree</code> command does not delete buffers if their input or output nets have the <code>DontTouchNet</code> attribute.</p> <p><b>Note:</b> The <code>DontTouchNet</code> statement is not a physical parameter; any net specified in this statement can still be routed.</p>                                                                                                                                                                                                                                                                                                                                                                                         |

|                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>DontTouchFromToPin + pinName1 pinName2</pre>                 | <p><b>Instructs the <code>ckSynthesis</code> and <code>ckEco</code> commands to not insert buffers for nets that are between the specified start instance pin and end instance pin. Any nets between these pins are considered to have the <code>DontTouchNet</code> attribute.</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <pre>DontAddNewPortModule + moduleName1 + moduleName2 + ...</pre> | <p>Does not add a new port to the specified logical modules at their given hierarchical levels.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>● The <code>DontAddNewPortModule</code> statement applies only to the specified modules but not the nested submodules. When you specify this statement for any given module, only that module does not have a new port added to it but the nested submodule still might have a new port added.</li> <li>● The <code>DontAddNewPortModule</code> statement might increase the latency of the clock tree and more buffers could be added because due to this restriction, the instances of different modules cannot be shared by the same buffer.</li> </ul> |

|                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>LeafPinGroup <i>groupName</i></pre> | <p>Instructs CTS to balance the group of leaf pins separately with the main clock tree. By using <code>LeafPinGroup</code>, the skew for leaf pin groups is reported separately.</p> <p>Specify <code>LeafPinGroup</code> for the leaf pins that do not need to be balanced with main group (that is, default group) of leaf pins, while the leaf pins under the same leaf pin group have to be balanced.</p> <p><b>Note:</b> Specifying group name is optional. If group name is not specified, by default, CTS uses names such as <code>LeafPinGroup0</code>, <code>LeafPinGroup1</code>, and so on.</p> |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Clock Grouping Data Section

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

|                                                                              |                                                                                        |
|------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <pre>ClkGroup + <i>clockRootPinName</i> + <i>clockRootPinName</i> ... </pre> | <p>Specifies two or more clock domains for which you want CTS to balance the skew.</p> |
|------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|

## Clock-Tree Topology Section

The clock-tree topology section provides a method for you to manually define buffers at particular levels. The following table describes the entries for the clock-tree topology section:

|                                        |                                             |
|----------------------------------------|---------------------------------------------|
| <pre>ClockNetName <i>netName</i></pre> | <p>Specifies the name of the clock net.</p> |
|----------------------------------------|---------------------------------------------|

|                                                                          |                                                                                                      |                                                                                                  |
|--------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| LevelNumber<br><i>totalNumberOfClockTreeLevels</i>                       | Specifies the total number of clock tree levels.                                                     |                                                                                                  |
| LevelSpec <i>levelNumber</i><br><i>numberOfBuffers</i> <i>bufferType</i> | Provides the specifications for an individual clock level.<br>Specify all the following information: |                                                                                                  |
|                                                                          | <i>levelNumber</i>                                                                                   | Sets the level number in the clock tree.                                                         |
|                                                                          | <i>numberOfBuffers</i>                                                                               | Specifies the total number of buffers CTS should allow on the specified level of the clock tree. |
|                                                                          | <i>bufferType</i>                                                                                    | Specifies the buffer type (based upon the LEF file).                                             |
| End                                                                      | Marks the end of a clock tree topology section.                                                      |                                                                                                  |

## Automatic Gated CTS Section

The following table describes the entries for the automatic gated CTS section:

|                                                                    |                                       |                                                                                                                                                                                                                                                                                                                                                       |
|--------------------------------------------------------------------|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GlobalPowerDomainBuffer                                            | <i>+PowerDomainNamesListofBuffers</i> | Specifies the power domain names and a list of buffers associated with the power domain.                                                                                                                                                                                                                                                              |
| AutoCTSRootPin <i>clockRootPinName</i>                             |                                       | Specifies the name of the clock root pin name from which to start tracing.                                                                                                                                                                                                                                                                            |
| CellObstruction<br>+ <i>cellName</i><br>Entire Ignored Detailed HV |                                       | Overrides CTS's obstruction modeling for a cell. <ul style="list-style-type: none"> <li>● <b>Entire:</b> Specifies the cell name and instructs CTS to treat the entire cell as routing obstruction globally.</li> <li>● <b>Ignored:</b> Specifies the cell name and instructs CTS to ignore routing obstruction due to this cell globally.</li> </ul> |

- **Detailed:** Specifies the cell name and instructs CTS to honor routing obstruction due to cell globally if the routing obstruction layers obstruct all the preferred routing layers.  
For example, if preferred routing layers are set to METAL4 and METAL5 and 33 cell routing obstruction layers are between METAL1 to METAL4, then CTS will ignore the routing obstruction.
- **HV:** Specifies the cell name and instructs CTS to honor routing obstruction of the cell for layers that obstructs the preferred routing layers.  
For example, if the preferred routing layers are set to METAL4 and METAL5 and the cell routing obstruction layers are from METAL1 to METAL4, then CTS honors the cell obstruction for METAL4 only. By default, CTS ignores the cell obstruction if there is one or more preferred routing layers are not obstructed.

***Default:*** No manual overriding, the obstruction modeling is auto-computed by tool.

|                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| InstanceObstruction<br>+ InstanceName Entire Ignored Detailed HV | <p>Overrides CTS's obstruction modelling for an instance.</p> <ul style="list-style-type: none"><li>• <b>Entire:</b> Specifies the instance name and instructs CTS to treat the entire instance as routing obstruction.</li><li>• <b>Ignored:</b> Specifies the instance name and instructs CTS to ignore routing obstruction due to this instance.</li><li>• <b>Detailed:</b> Specifies the instance name and instructs CTS to honor routing obstruction due to the instance if the routing obstruction layers obstruct all the preferred routing layers.</li><li>• <b>HV:</b> Specifies the instance name and instructs CTS to honor routing obstruction of the instance for layers that obstructs the preferred routing layers.</li></ul> <p><i><b>Default:</b></i> No manual overriding, the obstruction modeling is auto-computed by tool.</p> |
|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                      |                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>ForceMaxFanout {Yes   No}</pre> | <p>Specifies the maximum fanout as a soft constraint. If the value is specified as <code>yes</code>, then the maximum fanout is specified as a hard constraint.</p> <p><i>Default:</i> <code>No</code></p>                                                                                                                                                                                                           |
| <pre>SkipRouteGuide {Yes   No}</pre> | <p>Specifies the clock domains where you need not create routing guide files.</p> <p>For example, consider the following syntax:</p> <pre>AutoCTSRootPin clkx ... RouteClkNet Yes SkipRouteGuide YES End  AutoCTSRootPin clky ... RouteClkNet Yes End</pre> <p>Here, no routing guide file is created for <code>clkx</code> but created by default for <code>clky</code>.</p> <p><i>Default:</i> <code>No</code></p> |
| <pre>MaxDelay number{ns   ps}</pre>  | <p>Specifies the maximum phase delay constraint.</p> <p><i>Default:</i> If you do not use this statement, CTS uses a maximum phase delay constraint of 10 ns.</p>                                                                                                                                                                                                                                                    |

|                                          |                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>MinDelay number{ns ps}</pre>        | <p>Specifies the minimum phase delay constraint.</p> <p><i>Default:</i> If you do not use this statement, CTS uses a minimum phase delay constraint of 0.0 ns.</p>                                                                                                                                                                                                                                   |
| <pre>SetBBoxPinAsSync {true false}</pre> | <p>Treats I/O pins as synchronous pins, instead of as excluded pins.</p> <p>For example, if you specify <code>setCTSMode</code> as <code>true</code> and <code>setBBoxPinAsSync</code> as <code>false</code>, then all other clocks are <code>true</code> except the one that is specified as <code>false</code> in the clock tree specification file.</p> <p><i>Default:</i> <code>false</code></p> |
| <pre>SinkMaxTran number{ns ps}</pre>     | <p>Specifies the maximum input transition time constraint for sinks (clock pins). CTS does not allow you to specify a value greater than 10,000 ns.</p> <p><i>Default:</i> If you do not use this statement, CTS uses a maximum sink transition time constraint of 200 ps.</p>                                                                                                                       |
| <pre>BufMaxTran number{ns ps}</pre>      | <p>Specifies the maximum input transition time constraint for buffers. CTS does not allow you to specify a value greater than 10,000 ns.</p> <p><i>Default:</i> If you do not use this statement, CTS uses a maximum buffer transition time constraint of 200 ps.</p>                                                                                                                                |

|                           |                                                                                                                                                                                                   |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MaxGateRatio <i>ratio</i> | Specifies the maximum gate ratio for the clock tree. Gate ratio is defined as:<br><br>sum of all cell delay / clock path delay (from clock root to all registers)<br><br><i>Default:</i> 1        |
| MinGateRatio <i>ratio</i> | Specifies the minimum gate ratio for the clock tree.<br><br>Gate ratio is defined as:<br><br>sum of all cell delay / clock path delay (from clock root to all registers)<br><br><i>Default:</i> 0 |

MaxDeltaGateRatio *delta\_ratio*

Specifies the maximum delta gate ratio for the clock tree. Delta gate ratio is defined as the difference between fmGateRatio and toGateRatio, where:

- fmGateRatio is the sum of all cell delay divided by the clock path delay from the branching point to the source register.
- toGateRatio is the sum of all cell delay divided by the clock path delay from the branching point to the destination register.

**Note:** fmGateRatio and toGateRatio are local measurements: the starting point is at the input pin of the last buffer of the common clock path.

*Default:* 1

|                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>MinDeltaGateRatio delta_ratio</pre> | <p>Specifies the minimum delta gate ratio for the clock tree. Delta gate ratio is defined as the difference between <code>fmGateRatio</code> and <code>toGateRatio</code>, where:</p> <ul style="list-style-type: none"> <li>• <code>fmGateRatio</code> is the sum of all cell delay divided by the clock path delay from the branching point to the source register.</li> <li>• <code>toGateRatio</code> is the sum of all cell delay divided by the clock path delay from the branching point to the destination register.</li> </ul> <p><b>Note:</b> <code>fmGateRatio</code> and <code>toGateRatio</code> are local measurements: the starting point is at the input pin of the last buffer of the common clock path.</p> <p><i>Default:</i> 1</p> |
| <pre>SrcLatency number{ns}</pre>         | <p>Specifies an external latency value for each clock. CTS adds this value to the clock tree latency when grouping and balancing clocks. Consider the following clock tree specification file extracts:</p> <pre>AutoCTSRootPin clock1 MaxDelay 5.0ns MinDelay 4.5ns SrcLatency 2.0ns</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

MaxSkew 300ps

...

End

AutoCTSRootPin clock2

MaxDelay 2.0ns

MinDelay 1.5ns

SrcLatency 5.0ns

MaxSkew 300ps

...

End

The values for MaxDelay, MinDelay, and SrcLatency must satisfy the following relationship for all clocks listed in a clock tree specification file ClkGroup statement for that ClkGroup statement to be valid:

SrcLatency<sub>clock1</sub> + MinDelay<sub>clock1</sub> = SrcLatency<sub>clock2</sub> + MinDelay<sub>clock2</sub> = SrcLatency<sub>clock n</sub> + MinDelay<sub>clock n</sub>

SrcLatency<sub>clock1</sub> + MaxDelay<sub>clock1</sub> = SrcLatency<sub>clock2</sub> + MaxDelay<sub>clock2</sub> = SrcLatency<sub>clock n</sub> + MaxDelay<sub>clock n</sub>

***Default:*** 0ns

`RootInputTran number{ns|ps}`

Specifies the input transition time for an input pin or an input port. Use this statement when you do not want CTS to use the default Innovus input transition time.

**Note:** If your SDC file contains a `set_input_transition` constraint, the `RootInputTrans` statement overrides that constraint.

 You *cannot* use this statement with output pins or output ports.

|                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>MaxSkew number{ns ps} [viewName]</pre> | <p>Specifies the maximum skew between sinks (clock pins). You can specify different skew limits to be used for specific active analysis views (<i>viewName</i>) for multi-mode CTS.</p> <p>If the <code>MaxSkew</code> value is specified without specifying any analysis views, then the value is applied to all views.</p> <p>If multiple <code>MaxSkew</code> values are specified for the same pin and the same analysis view, the value specified last will override any previously specified values.</p> <p><i>Default:</i> 4% of the clock period specified through <code>create_clock</code> statement in the SDC file.</p> |
| <pre>DefaultMaxCap integer</pre>            | <p>Specifies the default maximum capacitance value for all buffers, inverters, and gating cells that do not have a specific maximum capacitance value specified with the <code>MaxCap</code> and <code>PinMaxCap</code> statements.</p>                                                                                                                                                                                                                                                                                                                                                                                             |

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>MaxFanout integer</code>  | <p>Limits the number of clock buffer fanouts to the number you specify.</p> <p>If you specify the <code>MaxFanout</code> statement and the <code>setCTSMode -useLibMaxFanout true</code> parameter, CTS uses the worst case constraint specified.</p> <p>CTS does not support a <code>fanout_load</code> that is not equal to 1.</p> <p><b>Note:</b> CTS considers the <code>MaxFanout</code> statement to be a soft constraint. CTS will try to meet it, but might need to make adjustments in order to meet physical constraints.</p> |
| <code>MaxNumLevel number</code> | Specifies the maximum number of buffer levels that CTS builds in a clock tree that has no gating components.                                                                                                                                                                                                                                                                                                                                                                                                                            |

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ClockGatingProb <i>number</i> | <p>Specifies the probable amount of time that the clock is turned on.</p> <p>The <code>ClockGatingProb</code> value is used for statistical estimation of power. CTS calculates internal power by multiplying the activity of the clock net by the <code>ClockGatingProb</code> value.</p> <p>(The activity of the clock net is taken from the <code>clock Period</code> statement in the clock tree specification file.)</p> <p>If you have a single clock gate at the root, and all of the buffers are after the clock gating cell, CTS calculates the activity rate as:</p> <p><i>(activity of clock net at gate's input) x ClockGatingProb</i></p> <p>CTS computes the activity rate of a net driven by a buffer in the clock path as:</p> <p><i>(activity of clock net at buffer's input) x ClockGatingProb</i></p> <p><i>Default:</i> 1 (clock is always on)</p> |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LevelBalanced YES   NO | <p>Instructs CTS to build a clock tree in which all paths from the root to the leaves have the same number of levels. (Equal numbers of levels can be helpful in high-performance designs, such as designs incorporating structured ASICs.)</p> <p><b>Note:</b> Be aware of the following requirements:</p> <ul style="list-style-type: none"><li>• This statement does not optimize for skew: It only builds trees with equal numbers of levels. (If you want CTS to optimize for skew, specify LevelBalanced NO after initial CTS.)</li><li>• To avoid creating maximum capacitance violations, include the YES statement in the clock tree specification file.</li></ul> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 20px;"><p> You cannot use this statement for gated clocks.</p></div> |
| Period value           | <p>Specifies the clock period of a root pin.</p> <p><i>Default:</i> If you omit this parameter, CTS uses a value of 10 ns.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

|                                  |                                                                                                    |                                                                                                                                                                                                                  |
|----------------------------------|----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NoGating {rising   falling   NO} | <p>Sets the criteria for tracing through logic gates.</p> <p>Choose only one of the following:</p> |                                                                                                                                                                                                                  |
|                                  | rising                                                                                             | Stops tracing through a gate (including buffers and inverters) and treats the gate as a rising-edge-triggered flip-flop clock pin.                                                                               |
|                                  | falling                                                                                            | Stops tracing through a gate (including buffers and inverters) and treats the gate as a falling-edge-triggered flip-flop clock pin.                                                                              |
|                                  | NO                                                                                                 | <p>Allows CTS to trace through clock gating logic.</p> <p><i>Default:</i> This is the default behavior for gated-clock designs. (If you omit the NoGating statement, CTS traces through clock gating logic.)</p> |

|                                                                  |                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AddDriverCell <i>driver_cell_name</i>                            | Places a driver cell name at the closest possible location to the clock port location for block-level CTS.                                                                                                                                                                                                                                                                    |
| MaxDepth <i>number</i>                                           | Sets the maximum depth of clock tree tracing.<br><br><i>Default:</i> If you do not use this statement, CTS limits the number of levels of clock tree tracing to 1024.                                                                                                                                                                                                         |
| RouteType <i>routeTypeName</i>                                   | Specifies the name of the routing attributes definition.<br><br><b>Note:</b> CTS uses the RouteType statement <i>only if</i> you specify RouteClkNet YES.<br><br><b>Note:</b> If you use a RouteType statement, you must also use a corresponding <i>routeTypeName</i> .                                                                                                      |
| LeafRouteType <i>leafRouteTypeName</i>                           | Specifies the route type name for which you are defining a routing attribute. Use this statement when you want to define a particular routing type for nets that connect to leaf pins.<br><br>The LeafRouteType statement applies to that part of a net that runs from the last non-leaf cell to leaf cell(s). The LeafRouteType statement is useful for high-fanout designs. |
| cellRouteType<br>+ <i>cellName</i> <i>RouteTypeName</i><br>+ ... | Specifies the routing attribute for the output net driven by the specified cell.                                                                                                                                                                                                                                                                                              |

|                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>cellLeafRouteType + <i>cellName</i> <i>RouteTypeName</i> + ...</pre> | <p>Specifies the routing attribute for the output net driven by the specified leaf cell. This applies only to the leaf nets.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <pre>DetailReport YES   NO</pre>                                          | <p>Determines whether CTS provides a detailed report. The detailed report includes timing information for every component in the design. The non-detailed report contains only summary information for the design.</p> <p><i>Default:</i> If you do not use this statement, CTS does not provide a detailed report.</p>                                                                                                                                                                                                                                                  |
| <pre>Obstruction YES   NO</pre>                                           | <p>Specifies whether CTS should take design obstructions into account during flip-flop clustering.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <pre>RouteClkNet YES   NO</pre>                                           | <p>Specifies whether CTS routes clock nets.</p> <p>The following combinations of the <code>RouteClkNet</code> specification file statement and the <code>setCTSMode -routeGuide true</code> command are possible:</p> <ul style="list-style-type: none"> <li>● If you specify <code>RouteClkNet NO</code>, CTS does not route clock nets. Timing is based on the pre-routed design using a steiner model. The <code>RouteClkNet NO</code> statement is useful for rapid prototyping.</li> <li>● If you specify <code>RouteClkNet YES</code> together with the</li> </ul> |

`setCTSMode -routeGuide  
false` command, CTS routes clocks using NanoRoute but CTS does *not* use a routing guide file. This combination of settings is useful for straightforward designs for which preRoute-postRoute correlation is not an issue. CTS runs more quickly than when it uses a guide file. Note that NanoRoute routes the clocks but does not follow the pattern determined by CTS. Thus, this method cannot take advantage of the balanced routing that takes place during CTS.

- If you specify `RouteClkNet YES` together with the `setCTSMode -routeGuide true` command, CTS routes the clocks with NanoRoute and the CTS routing guide file. This method ensures appropriate preRoute-postRoute correlation but at the expense of longer runtime.

*Default:* If you do *not* use this statement, CTS does not route clock nets. In other words, the default behavior is as if you had specified `RouteClkNet NO`.

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PostOpt YES   NO | <p>Specifies whether CTS resizes buffers or inverters, refines placement, and corrects routing for signal and clock wires.</p> <p><i>Default:</i> YES</p> <p>When used together, the PostOpt and OptAddBuffer statements try to meet the trigger edge skew constraints as defined in the clock tree specification file. However, phase delay, buffer transition time, and sink transition times are not necessarily improved by using these two statements.</p> <p><b>Note:</b> If you specify PostOpt NO, CTS does <i>not</i> resize gating components. If you specify PostOpt YES, CTS <i>attempts</i> to resize gating components, though it may not do so.</p> |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>OptAddBuffer {YES   NO}</code>            | <p>Controls whether CTS adds buffers during optimization.</p> <p><b>ⓘ</b> The <code>OptAddBuffer</code> statement is effective <i>only if you specify PostOpt YES.</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>OptAddBufferLimit positive_integer</code> | <p>When used together, the <code>PostOpt</code> and <code>OptAddBuffer</code> statements try to meet the trigger edge skew constraints as defined in the clock tree specification file. However, phase delay, buffer transition time, and sink transition times are not necessarily improved by using these two statements.</p> <p>Specifies the maximum number of buffers that CTS can add to a clock tree during optimization. The number you specify can be no smaller than 1. However, the higher the number you specify, the longer the runtime.</p> <p><i>Default:</i> If you do not use this statement, CTS inserts no more than 50 buffers.</p> |

|                                                              |                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                              |
|--------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <pre>AddSpareFF <i>cellName</i> <i>number</i></pre>          | <p>Specifies the maximum number of flip-flops to add to the lowest level of the clock tree.</p> <p>The inputs of the flip-flops are tied off and the outputs are left floating.</p> <p>Adding extra flip-flops during CTS ensures that if a spare flip-flop needs to be connected at some later time (for example, by a metal-only ECO change), the existing clock network will not be disturbed.</p> |                                                                                              |
|                                                              | <i>cellName</i>                                                                                                                                                                                                                                                                                                                                                                                       | Represents the name of the flip-flop(s) to be inserted in the clock root.                    |
|                                                              | <i>number</i>                                                                                                                                                                                                                                                                                                                                                                                         | Represents the maximum number of flip-flops to insert. You can specify 1 or more flip-flops. |
| <pre>Buffer <i>cell1</i> <i>cell2</i> <i>cell3</i> ...</pre> | <p>Specifies the names of buffer cells to use during automatic, gated CTS.</p> <p><b>Note:</b> To turn on the buffer insertion mechanism during the optimization process, you must have these statements in your clock tree specification file:</p> <ul style="list-style-type: none"> <li>● PostOpt YES</li> <li>● OptAddBuffer YES</li> </ul>                                                       |                                                                                              |

DummyBuffer *cell1* *cell2* *cell3* ...

Specifies the names of dummy buffer cells to use during the optimization process.

Use this statement if you want CTS to use buffers other than those specified in the `Buffer` statement. (Buffers defined in the `Buffer` statement might be too large, or have an input capacitance value that is too small.)

**Note:** To turn on the buffer insertion mechanism during the optimization process, you must have these statements in your clock tree specification file:

- PostOpt YES
- OptAddBuffer YES

|                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                           |
|-----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| <pre>LeafBuffer buffer_list</pre>                   | <p>Specifies a list of one or more buffer cells or inverters for CTS to use when inserting buffers at the leaf level of the clock tree.</p> <p>CTS ignores the <code>LeafBuffer</code> statement if:</p> <ul style="list-style-type: none"> <li>• The netlist has an inverter driving a small group of flops, and the design rule constraints are not violated.</li> <li>• The gating cells (such as AND gates) are physically close to the flops, and have sufficient drive strength to drive the flops at its fanout zone.</li> </ul> |                                                                           |
| <pre>LeafPin + pinName rising   falling + ...</pre> | <p>Marks the input pin as a "leaf" pin for non-clock-type instances, stops tracing, and balances clock skew.</p> <p><b>Note:</b> Use the <code>LeafPin</code> statement with an input pin.</p> <p>Choose one of the following:</p>                                                                                                                                                                                                                                                                                                      |                                                                           |
|                                                     | rising                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.  |
|                                                     | falling                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | CTS treats the input pin as a falling-edge-triggered flip-flop clock pin. |

|                                                                                  |                                                                                                                                                                                                                                                                                                                                                 |                                                                     |
|----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|
| <pre>LeafPort + <i>cellType</i>/<i>inputPinName</i> rising   falling + ...</pre> | <p>Marks the pin of a particular cell type as a "leaf" pin for non-clock-type instances, stops tracing, and balances clock skew.</p> <p>Choose one of the following:</p>                                                                                                                                                                        |                                                                     |
|                                                                                  | rising                                                                                                                                                                                                                                                                                                                                          | CTS treats the pin as a rising-edge-triggered flip-flop clock pin.  |
|                                                                                  | falling                                                                                                                                                                                                                                                                                                                                         | CTS treats the pin as a falling-edge-triggered flip-flop clock pin. |
| <pre>ExcludedPin + <i>pinName</i> + ...</pre>                                    | <p>Treats the pin as a non-leaf pin, and prevents tracing and skew analysis of the pin.</p> <p><b>Note:</b> The final excluded point may be different than the specified excluded pin.</p>                                                                                                                                                      |                                                                     |
| <pre>ExcludedPort + <i>cellType</i> / <i>inputPinName</i> + ...</pre>            | <p>Treats the pin of a particular cell type as a non-leaf pin, and prevents tracing and skew analysis of the pin.</p>                                                                                                                                                                                                                           |                                                                     |
| <pre>UnsyncPin + <i>pinName</i> + ...</pre>                                      | <p>Treats specified pin as a non-leaf pin, and prevents CTS tracing and skew balancing of the pin. <i>UnsyncPin</i> is specified between <i>AutoCTSRootPin</i> and <i>END</i> statement and it only affects a single clock.</p> <p><b>Note:</b> CTS fixes the DRV for the <i>UnsyncPin</i>, which is different than the <i>ExcludedPin</i>.</p> |                                                                     |

|                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>KeepPortPolarity + <i>moduleName/portName</i></pre>             | <p>Preserves the polarity of specified module ports while running the <a href="#">deleteClockTree</a> command.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <pre>ThroughPin + <i>pinName</i> [<i>outputPinNames</i>] + ...</pre> | <p>Traces through the pin, even if the pin is a clock pin.</p> <p>By default, when a ThroughPin is specified at the input pin of multi-output cells, it traces through all the outputs as long as there is valid timing arc from the input pin to the output pin. If you want CTS to trace through a certain output pin, you must use <i>outputPinName</i> to instruct CTS which output pin to trace through.</p> <p>For example, if you want CTS to trace through an instance <code>top/mmcore</code> from the input pin <code>clk</code> to <code>clk01</code> and <code>clk02</code>, although this instance has another clock output <code>clk03</code>, specify the following in the clock tree specification file:</p> <pre>ThroughPin + top/mmcore/clk clk01 clk02</pre> |
| <pre>ThroughPort + <i>cellType/inputPinName</i> + ...</pre>          | <p>Traces through the pin of a particular cell type, even if the pin is a clock pin.</p> <p><b>Note:</b> You can specify the <code>cellHalo</code> value for a flip-flop if the clock pin is specified as ThroughPin or ThroughPort.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>SetDPinAsSync YES   NO</pre>  | <p>Determines whether CTS automatically treats the Data pins of flip-flops as synchronous pins, instead of as excluded pins.</p> <p>Data pins include:</p> <ul style="list-style-type: none"> <li>● Data pins</li> <li>● Enable pins</li> <li>● Scan-in pins</li> <li>● Scan-enable pins</li> <li>● Synchronous set and reset pins</li> </ul> <p>This parameter does not control tristate control pins. By default, the software treats them as synchronous pins.</p> <p><i><b>Default:</b></i> If you omit this statement, CTS treats the Data pins of flip-flops as excluded pins.</p> |
| <pre>SetIoPinAsSync YES   NO</pre> | <p>Determines whether CTS automatically treats I/O pins as synchronous pins, instead of as excluded pins.</p> <p>This parameter does not control tristate control pins. By default, the software treats them as synchronous pins.</p> <p><i><b>Default:</b></i> If you omit this statement, CTS treats I/O pins as excluded pins.</p>                                                                                                                                                                                                                                                    |

|                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>PreservePin + <i>inputPinName</i> + ...</pre>                                                                                 | <p>Preserves the netlist for the pin and pins below the pin in the clock tree. However, CTS considers any synchronized pins after the pin when computing skew.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <pre>PinMaxCap + <i>instanceName/pinName1 capValue1 {pf   ff}</i> + <i>instanceName/pinName2 capValue2 {pf   ff}</i> ... ...</pre> | <p>Specifies the maximum capacitance value for the specified pins.</p> <p><i>Default:</i> If there is no <code>PinMaxCap</code> statement specified in the clock tree spec file, but there is a <code>DefaultMaxCap</code> statement specified, the software uses the <code>DefaultMaxCap</code> value to constrain pins.</p> <p>If there are no <code>PinMaxCap</code> or <code>DefaultMaxCap</code> statements specified in the clock tree spec file, but you specify <code>setCTSMODE -useLibMaxCap true</code>, the software uses maximum capacitance values in the timing library.</p> <p>If there are no <code>PinMaxCap</code> or <code>DefaultMaxCap</code> statements specified in the clock tree spec file and you do not specify <code>setCTSMODE -useLibMaxCap true</code>, the software does not apply a maximum capacitance constraint.</p> |

|                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>MaxCap + <i>bufferName1</i> <i>capValue1</i>{pf   ff} + <i>bufferName2</i> <i>capValue2</i>{pf   ff} + <i>clockGatingCell1</i> <i>capValue1</i>{pf   ff} + <i>clockGatingCell2</i> <i>capValue2</i>{pf   ff} + ...</pre> | <p>Specifies a maximum capacitance value for the specified buffers, inverters, or gating cells.</p> <p><i>Default:</i> If there is no <code>MaxCap</code> statement specified in the clock tree spec file, but there is a <code>DefaultMaxCap</code> statement specified, the software uses the <code>DefaultMaxCap</code> value to constrain buffers, inverters, or gating cells.</p> <p>If there are no <code>MaxCap</code> or <code>DefaultMaxCap</code> statements specified in the clock tree spec file, but you specify <code>setCTSMODE -useLibMaxCap true</code>, the software uses maximum capacitance values in the timing library.</p> <p>If there are no <code>MaxCap</code> or <code>DefaultMaxCap</code> statements specified in the clock tree spec file and you do not specify <code>setCTSMODE -useLibMaxCap true</code>, the software does not apply a maximum capacitance constraint.</p> |
|                                                                                                                                                                                                                               | <p><i>bufferName</i></p> <p>Specifies the name of the buffer for which you specify a maximum capacitance value.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                           |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
|                            | <i>capValue</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Specifies the maximum capacitance for the buffer, in picofarads (pF) or femtofarads (fF). |
| ExcludedBuffer <i>cell</i> | <p>Specifies the buffer to insert to isolate the clock path that is excluded during clock tree tracing. The <code>ExcludedBuffer</code> statement can be used to separate the normal path to leaf pins and the excluded path to non-leaf pins.</p> <p><b>Note:</b> Excluded pins can be identified by clock tree tracing, as well as by being specified with the <code>ExcludedPin</code> statement in the clock tree specification file.</p> <p><i>Default:</i> If you do not specify the <code>ExcludedBuffer</code> statement, the software chooses a buffer to insert from the <code>Buffer</code> statement.</p> |                                                                                           |
| ForceMaxTran YES   NO      | <p>Increases the number of buffers to ensure that the clock net has no maximum transition violations.</p> <p><b>Note:</b> If you specify <code>ForceMaxTran YES</code> and run <code>ckECO -fixDRVOnly</code>, the software attempts to correct maximum transition violations.</p>                                                                                                                                                                                                                                                                                                                                    |                                                                                           |
| ForceReconvergent YES   NO | Controls whether CTS allows                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                           |

or prevents reconvergence conditions for individual clocks.

- YES indicates that you want CTS to allow reconvergence on a particular clock. CTS synthesizes such clocks.
- NO indicates that you do not want CTS to allow reconvergence on a particular clock. When CTS synthesizes such a clock, the software stops and issues a warning if a reconvergence condition is found for the clock.

The following example illustrates how to specify different reconvergence conditions on clocks `clk` and `clk2`:

```
AutoCTSRootPin clk
...
ForceReconvergent YES
...
END

AutoCTSRootPin clk2
...
ForceReconvergent NO
...
END
```

**Note:** You can override any

|                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                 | <p>ForceReconvergent statement by specifying <code>ckSynthesis -forceReconvergent</code>. In this case, CTS always synthesizes clocks that have reconvergence conditions.</p> <p><i>Default:</i> If you omit this statement from the clock tree specification file, CTS behaves as if you had specified ForceReconvergent NO.</p>                                                                                                                                                                                                                   |
| GatingGrpInstances<br>+ <i>instanceName instanceName ...</i><br>+ <i>instanceName instanceName ...</i><br>+ ... | <p>Instructs CTS to group instances. CTS will not insert buffers between the instances you specify.</p> <ul style="list-style-type: none"><li>• Each line that begins with a + represents a group of instances.</li><li>• Instances that you specify on the same line are in the same group.</li><li>• Each instance list must include at least one gate and one latch.</li><li>• You can specify no fewer than two, and no more than five instances, on each line.</li><li>• Each latch and gate you specify must be connected together.</li></ul> |

```
GatingGrpModule
+ moduleName
+moduleName
+ ...
```

Instructs CTS to treat all the instances within a specified module as if you had included the module's instances individually in a GatingGrpInstances statement.

- Each line that begins with a + represents a module and its instances.
- You can specify only one module on each line.
- You can use the wildcards \* and ?.

|                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| <pre>CellHalo + <i>cellName</i> <i>xHalo</i> <i>yHalo</i></pre>                                                                                                       | <p>Provides spacing rules between various clock instances. It can be specified on buffers and inverters which are used by CTS to build a clock tree. It can also be specified on the gating cells.</p> <p>For example, if you specify a <i>xHalo</i> of 3 um for BUFX4 and if there are two BUFX4 instances placed close to each other, then the x-direction spacing must be equal to or greater than 3 um.</p> <p>If you specify 3 um for BUFX4 and 4 um for BUFX6 and if the two buffers are placed close to each other, then the spacing between these buffers is 4 um or greater.</p> <p><b>Note:</b> CellHalo applies to clock instances. It does not apply between a clock tree buffer and the instance which is not a part of the clock tree.</p> |                                                         |
| <pre>MasterGateInst + <i>numClusters1</i> <i>instance1</i> + <i>numClusters2</i> <i>instance2</i> + <i>numClusters3</i> <i>instance3</i> <i>instance4</i> + ...</pre> | <p>Controls instance cloning.</p> <p><b>Note:</b> To clone gating cells in groups, specify more than one instance name.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                         |
|                                                                                                                                                                       | <i>numClusters</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Indicates how many times each instance is to be cloned. |
|                                                                                                                                                                       | <i>instance</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Represents the instance name.                           |

|                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                   |                                                       |
|----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------|
| MasterGateModule<br>+ <i>numClusters1 module1</i><br>+ <i>numClusters2 module2</i><br>+ ...        | Controls module cloning:                                                                                                                                                                                                                                                                                                                                                          |                                                       |
|                                                                                                    | <i>numClusters</i>                                                                                                                                                                                                                                                                                                                                                                | Indicates how many times each module is to be cloned. |
|                                                                                                    | <i>module</i>                                                                                                                                                                                                                                                                                                                                                                     | Represents the module name.                           |
| MasterGateTargetFanout <i>number</i>                                                               | <p>Limits the number of cloning cells.</p> <p>The number of cloning cells is equal to the number of fanouts divided by the master gate target fanout number.</p> <p><b>Note:</b> <a href="#">ckCloneGate</a> uses this constraint to limit the number of cloning cells. However, the final fanout of gating cells might differ depending on the loads the gating cells drive.</p> |                                                       |
| EquivGateInst<br>+ <i>instance1</i><br>+ <i>instance2</i><br>+ <i>instance3 instance4</i><br>+ ... | <p>Specifies the instances to be cloned. <a href="#">ckDeCloneGate</a> merges the instances you specify into one instance.</p> <p>To declone groups of gating cells, specify more than one instance on one line.</p>                                                                                                                                                              |                                                       |
| End                                                                                                | Marks the end of an automated gated CTS section.                                                                                                                                                                                                                                                                                                                                  |                                                       |

## Log File Headings

Given particular settings for the `RouteClkNet` and `PostOpt` statements, and for the `reportClockTree` command, the log file reports results in the following sections:

| Clock tree specification file statements | Example clock tree text command sequences                                                                                | Innovus log file section heading:<br><code>Clock clockName plus--</code> |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| RouteClkNet NO                           | <code>ckSynthesis</code><br><code>globalDetailRoute</code><br><code>reportClkTree -clk <i>clock</i></code>               | PreRoute Timing Analysis                                                 |
| RouteClkNet YES                          | <code>ckSynthesis</code><br><code>globalDetailRoute</code><br><code>reportClkTree -clk <i>clock</i> -clkRouteOnly</code> | Clk-Route-Only Timing Analysis                                           |
| RouteClkNet YES<br>PostOpt YES           | <code>ckSynthesis</code><br><code>globalDetailRoute</code><br><code>reportClkTree -clk <i>clock</i> -postRoute</code>    | PostRoute Timing Analysis                                                |

## CTS Report Descriptions

The standard CTS report (`top_level_cell.ctsrpt`) contains several sections by default, and several sections that are controlled by the settings of various CTS text commands.

**Note:** The report extracts that follow are based on various designs, and should not be construed as results from a single CTS run.

## General Information

The beginning of each CTS report contains the following general information about the clock tree:

- Library information

```
#####
#
```

```
# Complete Clock Tree Timing Report
#
#
```

```
# CLOCK: cgen/i_5/Y  
  
#  
  
# Mode: preRoute  
  
#  
  
# Library Name : slow  
  
# Operating Condition : slow  
  
# Process : 1  
  
# Voltage : 1.62  
  
# Temperature : 125  
  
#  
  
#####
```

- Clock tree structure information

```
Nr. of Subtrees : 1  
Nr. of Sinks : 343  
Nr. of Buffer : 9  
Nr. of Level (including gates) : 2  
Max trig. edge delay at sink(F) : TPRAM/mod166798/CK 477.7(ps)  
Min trig. edge delay at sink(R) : TPRAM/mod167332/CK 459.6(ps)
```

- Delay, skew, and transition information

(Actual) (Required)

```
Rise Phase Delay : 459.6~477.7(ps) 0~5000(ps)  
Fall Phase Delay : 432.8~446.7(ps) 0~5000(ps)  
Trig. Edge Skew : 18.1(ps) 250(ps)  
Rise Skew : 18.1(ps)  
Fall Skew : 13.9(ps)  
Max. Rise Buffer Tran : 238.5(ps) 550(ps)  
Max. Fall Buffer Tran : 141.4(ps) 550(ps)  
Max. Rise Sink Tran : 366.2(ps) 550(ps)
```

Max. Fall Sink Tran : 204.5 (ps) 550 (ps)  
Min. Rise Buffer Tran : 120 (ps) 0 (ps)  
Min. Fall Buffer Tran : 120 (ps) 0 (ps)  
Min. Rise Sink Tran : 340.6 (ps) 0 (ps)  
Min. Fall Sink Tran : 192 (ps) 0 (ps)

- Maximum transition time violations

\*\*\*\*\* Max Transition Time Violation \*\*\*\*\*

Pin Name (Actual) (Required)

-----  
reg/CK [406 353.5] (ps) 400 (ps)  
reg2/CK [406 353.4] (ps) 400 (ps)  
clk0\_\_L6\_I2/A [345.5 288.1] (ps) 300 (ps)  
clk0\_\_L7\_I4/A [346.2 296.3] (ps) 300 (ps)  
clk0\_\_L9\_I11/A [351.6 299.9] (ps) 300 (ps)  
clk0\_\_L9\_I10/A [361.5 305.9] (ps) 300 (ps)

- Skew distribution information

cgen/i\_5/Y delay[0 0] ( CK\_\_L1\_I0/A )

\*\*\*\*\* Skew Distribution \*\*\*\*\*

LEVEL 1 Buffer:

-----  
Input Delay Range Nr of Buffers  
-----  
[0.6 0.6] 1  
(max, min, avg, skew) = (0.6 (ps) 0.6 (ps) 0.6 (ps) 0 (ps))  
-----

```
Output Delay Range Nr of Buffers
-----
[195.5 195.5] 1
(max, min, avg, skew) = (195.5(ps) 195.5(ps) 195.5(ps) 0(ps)) LEVEL 2 Buffer:
-----
Input Delay Range Nr of Buffers
-----
[212.8 212.8] 1
(max, min, avg, skew) = (212.8(ps) 212.8(ps) 212.8(ps) 0(ps))

-----
Output Delay Range Nr of Buffers
-----
[414.5 414.5] 1
(max, min, avg, skew) = (414.5(ps) 414.5(ps) 414.5(ps) 0(ps))
```

## Macro Model Information

Information on macro models appears in the standard report:

Max trig. edge delay at sink(R) : AClk 4971(ps) \*Mmodel\*

Min trig. edge delay at sink(R) : AClk 4671(ps) \*Mmodel\*

\*Mmodel\* Trig. edge delay calculation uses MacroModel for the sink pin.

## Power Information

Power information appears at the end of the general section of the CTS report, immediately after the transition information. The software reports power information only if you specify `setCTSMode -powerAware true` and include the `Period` statement in the clock tree specification file.

CTS reports the total net switching power, internal clock instances power, internal leaf pin power, and leakage power for the clock.

CTS calculates internal power by multiplying the activity of the clock net by the `ClockGatingProb`

statement value. The activity of the clock net is taken from the `clock Period` statement in the clock tree specification file. The `ClockGatingProb` statement value is specified in the clock tree specification file, and is used for statistical estimation of power.

For example,

```
AutoCTSRootPin sysclk
MaxDelay 4ns
MinDelay 0ns
MaxSkew 0.2ns
...
ClockGatingProb 0.1
Period 100ns

END
```

If you do not specify a value for the `ClockGatingProb` statement, CTS uses the default value of 1.

If you have a single clock gate at the root, and all of the buffers are after the clock gating cell, CTS calculates the activity rate as:

$$(\text{activity of clock net at gate's input}) \times \text{ClockGatingProb}$$

CTS computes the activity rate of a net driven by a buffer in the clock path as:

$$(\text{activity of clock net at buffer's input}) \times \text{ClockGatingProb}$$

CTS uses the same net activity for calculating the switching power of the net.

|                      |   |                  |
|----------------------|---|------------------|
| NetSwitchPower       | : | 0.000997444 (mW) |
| InstInternalPower    | : | 0.00409096 (mW)  |
| InstLeakagePower     | : | 1.9129e-06 (mW)  |
| LeafPinInternalPower | : | 0.0013708 (mW)   |
| Total Power          | : | 0.00646111 (mW)  |

## AC Current Density Violations

Information on AC current density violations appears in the standard CTS report only if your LEF file contains an `ACCURRENTDENSITY` statement.

AC Irms Limit Violation : 0.387332 (mA) (17216.7%)

Information on AC current density violations appears in a special violations section:

\*\*\*\*\* AC Irms Limit Violation \*\*\*\*\*

Pin Name (Actual) (Required) (Violation)

---

|           |               |                 |               |
|-----------|---------------|-----------------|---------------|
| ClkMux/Y  | 0.389581 (mA) | 0.00224974 (mA) | 0.387332 (mA) |
| scanClk/Y | 0.156409 (mA) | 0.00224974 (mA) | 0.154159 (mA) |

## Supported SDC Constraints

Clock Tree Synthesis supports only the following SDC timing constraints during tracing (`setCTSMode -traceHonorConstants true`):

- `set_logic_one`
- `set_logic_zero`
- `set_case_analysis`
- `set_disable_timing`

The `createClockTreeSpec` command automatically translates certain SDC constraints into clock tree specification file statements. The following table shows the SDC constraints that the `createClockTreeSpec` command automatically translates, and also shows the default values in those statements if an SDC constraint does not exist:

| <b>SDC Constraint</b>                         | <b>Clock Tree Specification File Statement (default)</b>                                                        |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <code>create_clock</code>                     | <code>AutoCTSRootPin</code>                                                                                     |
| <code>set_clock_transition</code>             | <code>SinkLeafTran</code> and<br><code>BufMaxTran (Default: 200 ps)</code>                                      |
| <code>set_clock_latency value</code>          | <code>MaxDelay (Default: clock period)</code> <code>MinDelay (Default: 0)</code>                                |
| <code>set_clock_latency - source value</code> | <code>SrcLatency value ns</code>                                                                                |
| <code>set_clock_uncertainty</code>            | <code>MaxSkew (Default: 4% of the clock period specified through create_clock statement in the SDC file)</code> |
| <code>create_generated_clock</code>           | <code>Add ThroughPin when necessary</code>                                                                      |

By default, CTS takes 4% of the clock period specified through `create_clock` statement in SDC as `MaxSkew`. However, if the value of `set_clock_uncertainty` is lower than that, the value of `set_clock_uncertainty` is passed to CTS as `MaxSkew`.

**Note:** If there is a `set_clock_transition` statement in SDC, CTS translates the value to `BufMaxTran` and `SinkMaxTran`. If not, CTS checks if the `setDesignMode` command is set, and follows the table below to determine the `BufMaxTran` and `SinkMaxTran` values in the generated clock tree specification file.

| <b>setDesignMode -process num</b> | <b>BufMaxTran, SinkMaxTran (ps)</b> |
|-----------------------------------|-------------------------------------|
| Default                           | 200                                 |
| 111 to 250                        | 250                                 |
| 78 to 110                         | 200                                 |
| 65 to 78                          | 150                                 |
| Below 65                          | 80                                  |

# Working with Clock Mesh Structures

- Overview
- Clock Meshes Versus Clock Trees
- Creating Clock Meshes
  - Determining the Mesh Structure
  - Implementing the Clock Mesh
  - Analyzing the Clock Mesh
  - Generating Multiple Spice Run Deck For Big Clock-Mesh Networks
- MultiSpine Clock Mesh

## Overview

The Innovus™ Implementation System (Innovus) provides a semi-automatic way to synthesize clock mesh structures by automating the tasks of netlist update, physical implementation and analysis.

Clock meshes typically provide tighter skew control and limit the impact of process variation compared to clock trees. However, these advantages might be offset by increased routing resource usage and increased power dissipation due to larger switching capacitance.

## Clock Meshes Versus Clock Trees

There are advantages and drawbacks to using clock meshes instead of clock trees. Consider the following factors when determining whether a clock mesh is appropriate for a given block or clock domain.

- Skew  
Clock meshes can often deliver lower skew than a clock tree. By their nature and design, clock meshes deliver low skew to their leaf inputs. However, if any leaves require different clock arrival times (such as macro-models with different built-in insertion delays or useful skew), then clock mesh alone will not deliver a good overall solution. Clock mesh does not directly support early or late clocks.

**Note:** It might be possible to implement early and late clocks by hand for a limited number of leaves.

- Insertion delay  
Because meshes can use multiple buffers driving in parallel, they can potentially fan out to the

clock inputs with fewer stages and lower insertion delay than a tree.

If clock tree synthesis can meet the performance targets (skew and insertion delay), considering the effects of on-chip and process variation, there might not be a compelling reason to consider a clock mesh. CTS is currently more highly automated than clock mesh synthesis, and typically uses less power. However, if CTS cannot meet the skew or insertion delay constraints, it might be worth considering a clock mesh.

- Tolerance to variation

Mesh structures are generally less sensitive to on-chip variations (OCV) and process variations than corresponding trees.

- Power

Generally meshes can consume somewhat more power than trees, although the amount of extra power is highly design dependent. In some cases, clock meshes might actually consume less power than trees. Because most power typically is consumed at the final (leaf) level, using local distribution can significantly narrow the power gap between mesh and tree implementations.

- Gating

Clock mesh structures are not as flexible as clock trees, with respect to gating. Because the final mesh stage is a single net, gating must be implemented either at the root level, or the local level.

- Floorplan limitations

Clock meshes rely on arranging buffers and routing in regular symmetric patterns in order to achieve good skew. When constructing a mesh, the software tries to adjust positions of individual trunks and branches slightly in order to avoid conflicts and violations with existing placement or routing blockages, or pre-routes such as power stripes. Therefore, clock meshes work best in floorplan areas that are rectangular and relatively free of obstructions. Highly non-rectangular floorplans, such as L-shaped areas, or areas with macros or obstructions placed in the middle of the floorplan, can make it difficult or impossible to implement a mesh. Always evaluate the floorplan to determine if it looks feasible to insert a mesh.

- Degree of automation

Clock tree synthesis is usually 100 percent automatic; you specify the constraints, and the tool does all of the work. Currently, clock mesh is not fully automatic. You must choose the structure of the clock mesh in terms of style, number of levels, and so on. Also, depending on the floorplan and obstructions, you might need to experiment with different implementation parameters in order to find a feasible solution.

- Available Types of Driver Cells

Although the clock mesh feature is intended to work with arbitrary driver cells, some buffer or

inverter cells are better suited as mesh drivers than others. The ideal situation is to have specially designed mesh drivers, but if such cells are not available, usually the best approach is to select from the normal clock buffer cells used for CTS.

There are two primary aspects to consider when designing or selecting a mesh driver cell: its electrical or timing characteristics, and the geometry of its output pad.

- Electrical Characteristics

Clock mesh driver cells have the following electrical characteristics:

- Drive strength

Clock mesh nets typically have higher loading than the nets in clock trees. Even though meshes can arrange many drivers in parallel to drive large loads, it is usually preferable to have higher strength drivers available for clock meshes. In addition to simplifying the design, having fewer drivers in parallel often improves the performance and memory usage of delay calculation and timing analysis.

- Multiple row cells

Very strong mesh drivers draw large currents from the power supply rails. Arranging drivers to span multiple rows can reduce their impact on the power distribution system.

- Decoupling capacitors

Consider including decoupling capacitors inside the mesh driver to help supply transient current when the driver is switching.

- Output via stacks

Consider integrating via stacks up to mesh routing layers within the mesh driver cell. Including the stack within the cell can help ensure that it has sufficient current carrying ability and has minimal impact on routing resources.

- Balanced rise and fall

As with normal clock buffers, balanced rise and fall characteristics are important for clock mesh drivers to maintain duty cycle, and so on.

- Output Pad Geometry

The geometry of driver output pads is important because the clock mesh relies on direct driver-trunk connections. It does this by placing the driver underneath the trunk in such a way that either the wire directly connects to the output pad shape, or it crosses above it on a higher layer and connects with a stacked via.

Consider the following output pad shape guidelines when choosing (or designing) a driver cell:

- Use a single rectangle pad

Complex pad geometries composed of many shapes usually make it more difficult to locate and drop a clean via stack onto the pad.

- Use pads on the mesh layer  
Connections to pads on the trunk or branch routing layers are easier because they do not require additional via stacks.
- Be careful when input pin is on the mesh layer  
If the input pin is on the mesh routing layer, it must be possible to connect to the output without shorting to the input. For example, consider a driver with input and output both on  $M2$ . If the mesh is using a vertical trunk on  $M2$  and the input and output are aligned vertically, it might not be possible to make the trunk connection without a short or spacing violation.

## Creating Clock Meshes

Creating a clock mesh generally consists of the following tasks:

1. [Determining the Mesh Structure](#)
2. [Implementing the Clock Mesh](#)
3. [Analyzing the Clock Mesh](#)

## Determining the Mesh Structure

Similar to clock tree synthesis, the clock mesh feature uses a "specification" to define the scope of the clock domain, express constraints, and control the structure of the mesh. In addition to loading and saving specification files, you can also interactively edit the specification using the Clock Mesh Specification form. This makes it easier to experiment with different clock mesh structures.

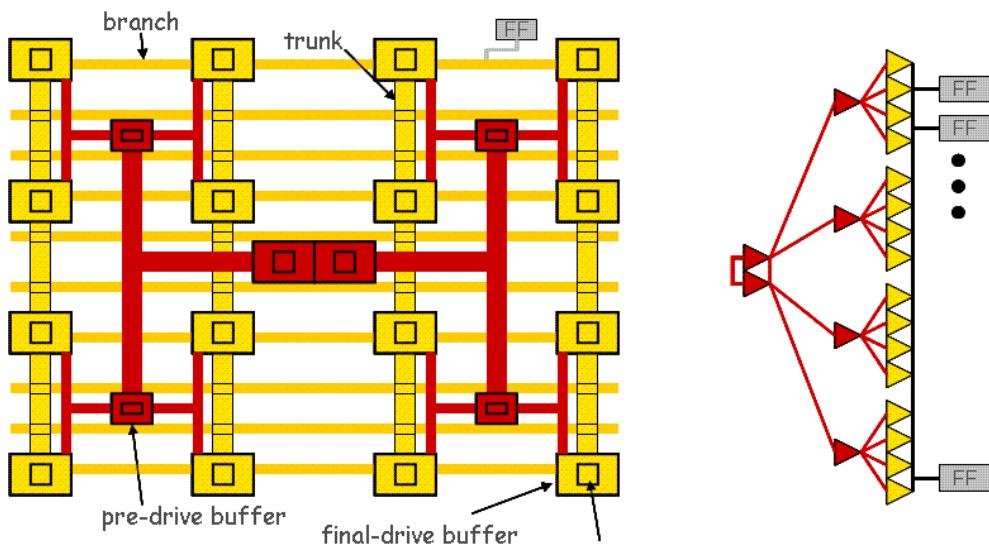
For more information on *Clock Mesh Specification* form, see the "Clock Mesh" section of the [Clock Menu](#) chapter in *Innovus Menu Reference*.

## Supported Mesh Styles

There is a wide variety of mesh styles currently in use. The Innovus software can synthesize the following mesh styles:

- H-tree + Mesh  
This classic style uses a multilevel H-tree pre-drive, followed by a general mesh final stage. Although the pre-drive is a tree structure, it can still employ multiple drivers on a single net. The final stage consists of a rectangular grid of final drivers feeding a rectangular mesh grid of trunks and branches.

Figure 21-1 H-Tree + Mesh Style

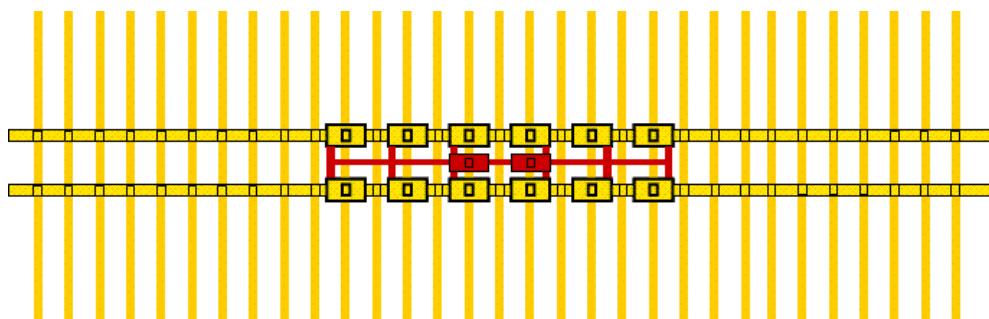


The advantage to this style is that the pre-drive is highly symmetrical and can achieve good skew control for large clock domains with many flip-flops. The drawback is that it can require higher power than other mesh styles.

- Fishbone

The final stage of a basic fishbone structure uses multiple drivers feeding a single trunk (spine) that, in turn, drives a number of orthogonal branches (bones). Pre-drive stages consist of multi-driven pre-drive trunks placed sufficiently near the next-stage driver inputs.

Figure 21-2 Single and Double Fishbone Style



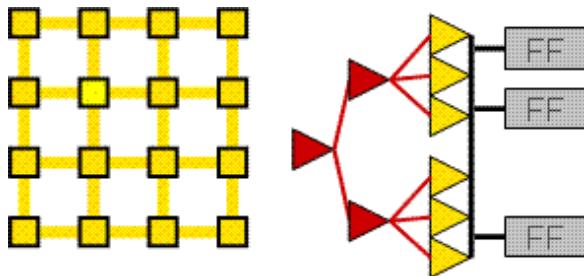
Additionally, there is a double-fishbone variant, in which the final stage uses two parallel trunks. The fishbone style can be a very good style for smaller clock domains.

## Clock Mesh Structure Characteristics

There also is a wide variety of meshed clock distribution structures currently in use. The clock mesh structures synthesized by the Innovus software have the following basic characteristics:

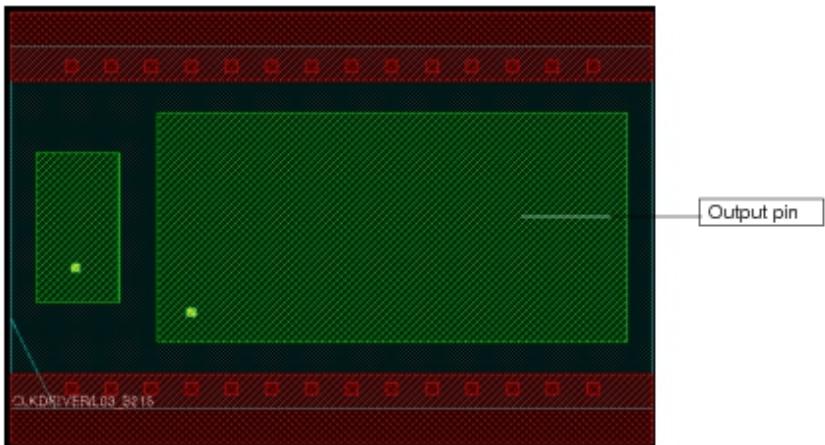
- Symmetry  
Meshes rely heavily on symmetry of netlist structure and routing pattern to achieve tight skew control and tolerance to variation.
- Routing patterns and topologies  
A major difference between clock meshes and clock trees are their routing patterns. Mesh routing commonly has the following characteristics:
  - Makes heavy use of non-tree routing topologies containing cycles or loops.
  - Uses wide wires to achieve low resistance and well controlled capacitance. For example it is not unreasonable for a mesh trunk to be 10 µm wide.
  - Routing is implemented using a combination of special and regular routes.
- Parallel drive  
Clock meshes typically rely on multiple buffers or inverters working in parallel to drive a given net in the pre-drive and final-drive stages, whereas clock trees always use exactly one driver per net. Using multiple parallel drivers makes it possible to drive heavily loaded nets from multiple points. This allows a mesh to fan out with fewer stages and better skew control than a tree.

**Figure 21-3 Parallel Drive with Cycle Routing Pattern**



- Driver output connections by abutment  
The connections between the mesh driver output pins and trunks are made by placing the trunk wire directly over the output pad, and possibly dropping a stacked via if required. This method ensures a strong connection that is capable of carrying the current supplied by the driver.

**Figure 21-4 Driver Cell with Large Output Pin for Abutment Connection**



- Input connections completed with regular routing

In contrast to driver output pins, which are connected directly to mesh trunks, input pins, such as driver inputs and flip-flop inputs in the final stage, are connected to mesh trunks and branches by regular routes created by the detail router (NanoRoute). Because individual input connections do not need to carry large currents, minimum width connections are generally sufficient. Nondefault rules can be used if wider widths or larger spacings are desired.

- Drivers placed in core area

Mesh driver cells are placed in the core area in rows like other standard cells, rather than being specially placed macro block cells. Large drivers can span multiple rows. Because mesh driver connections are made by abutment, driver cell placement is typically "fixed" to keep them from being unintentionally moved (and disconnected) during subsequent operations.

## Multilevel Structure of a Mesh

Like clock trees, clock meshes typically use a multilevel structure to fan the clock signal out from the root to all the clock input pins (flip-flops, memories, and so on). The Innovus software creates multilevel meshes that can include the following sections:

- Top-level chain

The top-level chain is a cascaded buffer chain from the mesh root to the first level of mesh pre-drive buffers. Chains can be used either to supply a suitable input transition to the mesh pre-drive, or to pad the mesh with extra insertion delay.

The routing for mesh chain nets is handled entirely by the NanoRoute router rather than by using a combination of special and regular routes. If wide routing is required for mesh chain routes, use LEF nondefault rules. Meshes are not required to include a top-level chain; the root can connect directly to the first pre-drive stage, if it is suitable.

- Global mesh

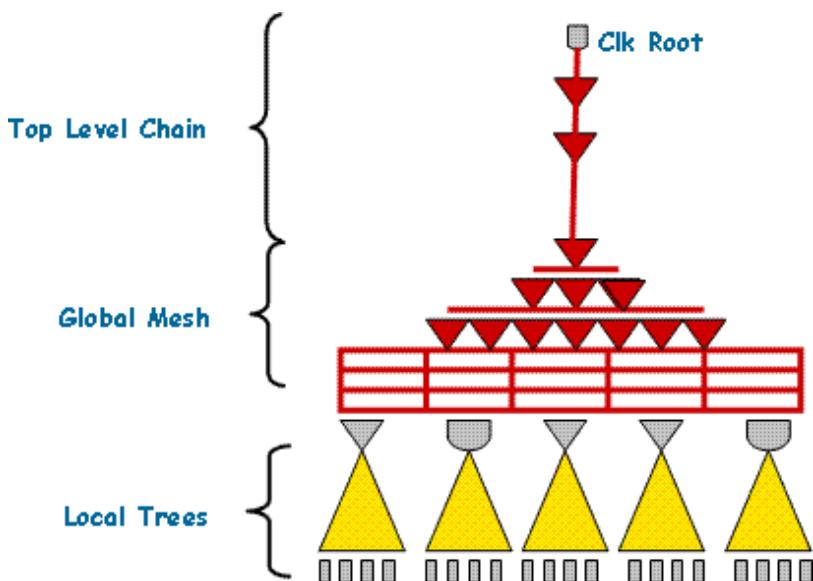
The purpose of the global mesh is to distribute a single clock signal across the entire clock domain with good insertion delay and skew control. The global mesh consists of multiple (zero or more) pre-drive stages, followed by a single final-drive stage. While the pre-drive stages can have multiple nets at a given level (for example in an H-tree), the final stage always drives a single final global mesh net. This final mesh net can connect directly to the clock input pins, or there can be an additional level of local distribution.

- Local distribution

Local distribution is an optional section in which multiple small trees distribute the single global signal of the final mesh drive net to individual flip-flops or memory inputs. The simplest form of local distribution consists of multiple small single-buffer "local trees," each driving a cluster of flip-flops. NanoRoute handles all of the local-level routing; there are no special routes.

Using local distribution can have advantages over direct mesh-to-flip-flop connections in the final stage. For example, local distribution significantly reduces the loading in the final mesh stage, and can allow for a mesh structure with lower overall power consumption. Also, for large clock domains with non-uniform flip-flop distributions, adding extra local buffers can help to balance the load seen by the mesh, and allow for better skew control.

**Figure 21-5 Multilevel Mesh Structure**



## Implementing the Clock Mesh

Implementing a clock mesh involves various tasks, such as inserting drivers into the netlist, placing and "fixing" the drivers, creating mesh routes, and so on. The Innovus clock mesh feature includes the following major implementation capabilities:

- Synthesis  
The `synthesizeClockMesh` command implements the clock mesh according to the current specification, by inserting and placing buffers, and creating mesh special routes as needed.
- Mesh routing  
The `routeClockMesh` command uses the native NanoRoute™ router to complete detailed routing connections for driver and flip-flop inputs, top-level cascade chains, and local trees.
- Wire trimming  
After mesh routing is complete, the `trimClockMesh` command removes unused portions of the mesh trunks and branches, eliminates antenna violations, and reduces overall mesh capacitance.

## Analyzing the Clock Mesh

The Innovus software can analyze clock meshes at different stages of the implementation process (preRoute or postRoute), using default extraction, detail extraction, or externally supplied RC data, and using either delay calculation technology or a circuit simulator. Various reports are available. To help visualize mesh quality, delay or transition information can be displayed graphically using a color scale.

Clock meshes have several characteristics that make them more difficult to analyze than clock trees:

- Meshes use a mixture of special routing and regular routing, which makes estimating and extracting wire RC information more challenging.
- Nets with multiple drivers need special consideration during delay calculation.
- A final stage clock mesh net might have thousands of drivers and tens of thousands of receivers, resulting in tens (or hundreds) of millions of driver-to-receiver timing arcs. This huge number of arcs can potentially lead to memory or performance problems during timing analysis.

The following sections describe how the Innovus software addresses some of the unique challenges to analyzing clock mesh timing.

## PreRoute Wire Estimation

The software's clock mesh synthesis inserts and places buffers and draws mesh trunks and branches, but it does not immediately start NanoRoute to complete detail routing connections for mesh driver inputs and flip-flops; this is handled by a separate command. Therefore, some wire estimation is needed to complete preRoute clock mesh analysis.

For nets without any special net routing, such as top-chain nets and local distribution nets, the software uses a simple two-layer Steiner routing estimation based on the routing preferences given in the clock mesh specification.

For global nets, the software builds an RC network for the existing special net routing and then estimates nearest connections for each unconnected driver input and flip-flop. For these nets, nearest point-to-point connections are more appropriate than Steiner routing estimates because NanoRoute will connect them using "pattern trunk" connections.

## RC Extraction

There are several different ways to get RC information for clock mesh analysis:

- Default extraction

Default extraction is simple 1-D extraction based on per-unit-distance resistance and capacitance values for mesh routes. Default extraction is most appropriate before full detailed routing is complete.

**Note:** Default extraction is the only available method to analyze preRoute clock mesh nets.

- Detail extraction

The Innovus software detail extraction can be used when mesh routing is complete.

**Note:** The clock mesh analysis commands will not automatically start detailed net extraction; to use this method, you must directly set the extraction mode then run the `extractRC` command prior to mesh analysis.

- External SPEF

To use RC data from an external parasitic extraction tool, load the SPEF prior to clock mesh analysis.

## Computing Mesh Delays

There are two ways of computing mesh delays and transitions times: delay calculation and circuit simulation.

- Delay Calculation

The fact that clock meshes rely on multiple buffers driving a single net presents some challenges for delay calculation. Although not all delay calculators can handle it, Innovus clock mesh uses delay calculation technology that supports multi-driven nets. For delay calculation to be accurate, all drivers on a given net must have similar input waveforms. This restriction is consistent with a well-designed mesh and probably does not present a major limitation in practice. The software generates warnings when the skew at multiple driver inputs exceeds a given fraction of their input slew.

- Circuit Simulation

The software supports simulation with Virtuoso® UltraSim™ fast SPICE simulator as an alternative to delay calculation for analyzing clock mesh delays and slews. Virtuoso® UltraSim™ simulator is a fast-SPICE simulator that is well-suited for analyzing clock meshes, even with postRoute RC data. Virtuoso UltraSim simulator is not packaged with the Innovus software, therefore to use it, you must have the Virtuoso UltraSim simulator executable in your path, and an appropriate license.

The software also supports simulation with external SPICE-like simulators, but the flow is not as convenient as with Virtuoso UltraSim simulator. The steps are as follows:

- a. Dump a SPICE netlist for the clock mesh.
- b. Manually run the simulator outside of the Innovus software.
- c. Backannotate the simulation measurement results. Innovus clock mesh provides commands to write the spice netlist and backannotate the results.

To simulate clock meshes, either with Virtuoso UltraSim simulator or with an external simulator, the Innovus software needs to have driver sub-circuit and model information available. This information must be provided via a plain-text cdb database file.

## **Generating Multiple Spice Run Deck For Big Clock-Mesh Networks**

For some large meshes, simulating the entire structure, from root to mesh receivers, with a single spice run can be prohibitive in terms of run time. As an alternative, Innovus clock mesh provides the multi-part spice mechanism, which can break the large spice simulation into two or more smaller simulations.

## **Steps to Generate Multi-Part Spice**

The following steps describe the process of multi-part spice:

1. Enable multi-part spice simulation with the `MultiPartSpice` keyword in the Analysis section of the

clock mesh specification file.

2. Choose a level in the mesh structure to partition the spice simulation into L1 (or global) level, and one or more L2 (or local) level simulation runs. Specify this partition point using the `MultiPartSpicePartitionLevel` keyword in the clock mesh specification file. The final receiver points of L1 will correspond to the initial stimulus points of the L2 simulation runs.
3. Generate spice L1 and L2 run decks.
4. Simulate the L1 spice deck. Measure delays and transitions for all points. Measure detailed waveforms for lowest level receivers.
5. Use lowest level waveforms from previous step to generate corresponding voltage sources L2 runs.
6. Simulate L2 spice decks.
7. Use L1 and L2 measurement results to back-annotate complete mesh timing to Innovus.
8. Innovus clock mesh provides three different methods, with varying degrees of automation, to manage the process of multi-part spice. The following table provides a description of these methods:

| Methods                                                                       | Steps                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Simulate and back-annotate automatically with UltraSim                        | <ol style="list-style-type: none"> <li>1. Set up multi-part spice simulation in the clock mesh specification file.</li> <li>2. Set <code>-preRouteAnalysis</code> or <code>-postRouteAnalysis</code> mode to UltraSim.</li> <li>3. Invoke an analysis command, such as <code>reportClockMesh</code>.</li> </ol>                                                                                                                                             |
| Simulate and back-annotate automatic with a user-defined simulation procedure | <ol style="list-style-type: none"> <li>1. Set up multi-part spice simulation in the clock mesh specification file.</li> <li>2. Set <code>-preRouteAnalysis</code> or <code>-postRouteAnalysis</code> mode to Spice.</li> <li>3. Define a procedure that can invoke the user-defined Spice simulator.</li> <li>4. Set <code>simSpiceProc</code> to name of procedure.</li> <li>5. Invoke analysis command, such as, <code>reportClockMesh</code>.</li> </ol> |
|                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

|                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Manual simulation and back annotation | <ol style="list-style-type: none"><li>1. Set up multi-part spice simulation in the clock mesh specification file.</li><li>2. Use the <code>spiceClockMesh</code> command to dump the L1 and L2 spice decks.</li><li>3. Manually simulate L1.</li><li>4. Use <code>spiceClockMesh -genL2Sources {}</code> to create the voltage sources to stimulate the L2 runs.</li><li>5. Manually simulate L2.</li><li>6. Back-annotate results using <code>spiceClockMesh -readMapMeas {}</code>.</li><li>7. Invoke analysis command, such as <code>reportClockMesh</code>.</li></ol> |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Sample Scripts to Run Spice Simulation

Use the following scripts to run multi-part Spice simulations:

### ***Simulate Automatically With UltraSim***

```
specifyClockMesh -file CLK.spec  
setClockMeshMode -postRouteAnalysis UltraSim  
reportClockMesh -delay -out CLK_delay.rpt
```

### ***Simulate with User-Defined Procedure***

```
specifyClockMesh -file CLK.spec  
setClockMeshMode -postRouteAnalysis Spice  
setClockMeshMode -simSpiceProc runMySpectre  
reportClockMesh -delay -out CLK_delay.rpt
```

The following Tcl procedure handles running the Spectre simulator on a Spice netlist, such as `file.sp` and leaves the simulation results in `file.print0` and `file.meas0`:

```
proc runMySpectre {file} {  
    regexp {^(\S+).sp\$} $file match base
```

```
file delete $base.meas0 $base.print0 $base.measure $base.print
exec spectre $file
if [file exists $base.measure] {
    file rename -force $base.measure $base.meas0
}
if [file exists $base.print] {
    file rename -force $base.print $base.print0
}
}
```

## Simulate Manually

```
specifyClockMesh -file CLK.spec
spiceClockMesh
#(simulate CLK_L1.sp)
spiceClockMesh -genL2Sources {CLK_L1.meas CLK_L1.print {CLK_L2_1.sp CLK_L2_2.sp...}}
#(simulate CLK_L2_*.sp)
spiceClockMesh -readMapMeas {CLK.mp CLK_L1.meas0 CLK_L2_1.meas0 CLK_L2_2.meas0...}
reportClockMesh -delay -out CLK_delay.rpt
```

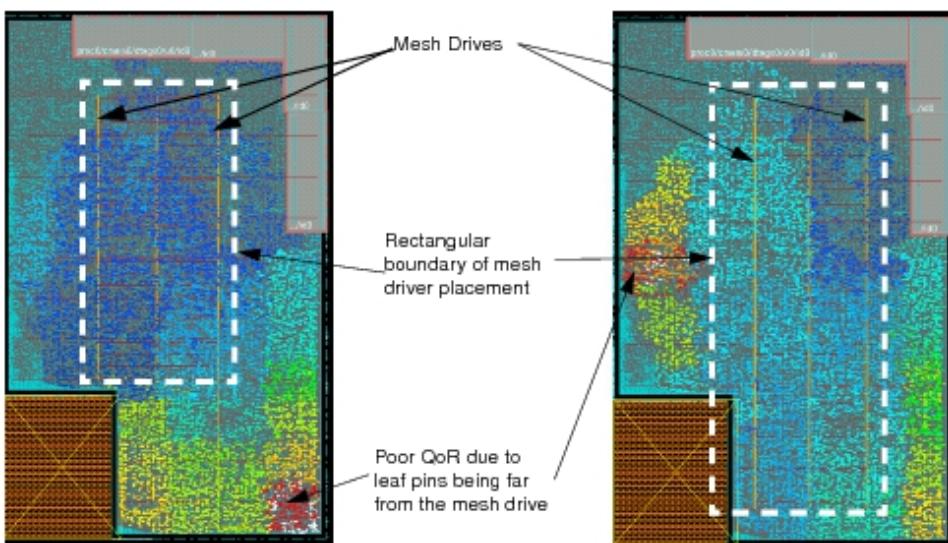
**Note:** The `setClockMeshMode -simMultiPartSpiceBoundaryReceiverAsInst` parameter controls whether the final receivers in L1 are simple pin capacitance values, or actual instantiated subcircuits.

**Note:** The `setClockMeshMode -simMultiPartSpiceNrInstThreshold` parameter combines the local portions of a clock network in one or separate SPICE files.

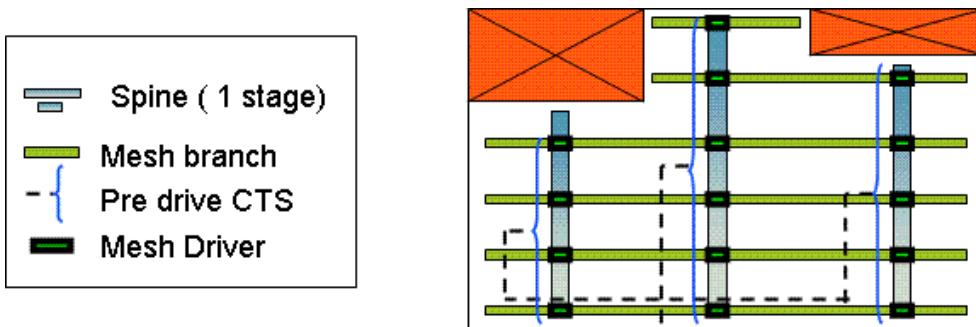
## MultiSpine Clock Mesh

The H-Tree+Mesh or fishbone meshes work best in floorplan areas that are rectangular. For highly non-rectangular floorplans, such as L-shaped, it can be very difficult to obtain a good clock performance. The limitation comes from the physical implementation of the last level of global mesh. The final mesh drivers that feed the mesh grid are placed in a rectangular shape. It is difficult to ensure all the receivers of the mesh grid receive sufficient driving strength if the rectangular-bounded mesh drivers are placed in non-rectangular floorplan.

As shown in the following diagrams, there are always some flops that are far away from the mesh driver as long as the boundary of the mesh driver forms a rectangle.



The multi-spine mesh is a mesh that offers advantage over the above two types of meshes. It can be implemented in a non-rectangular floorplan as shown below.



The pre-drive of the multispine mesh is a regular CTS (clock tree synthesis) clock tree.

The last stage of global mesh consists of multiple spines which in turn drive the orthogonal branches. Together, the spines and the branches form a mesh grid. Each spine is driven by the mesh drivers. Since each spine can have different length and so is the placement of the mesh drivers, the multi-spine mesh can be implemented in non-rectangular floorplan with good clock performance.

## Optimizing Timing

- Overview
- Before You Begin
- Results
- Interrupting Timing Optimization

- Performing Optimization Before Clock Tree Synthesis
  - Correcting Violations in PreCTS Mode for the First Time
  - Performing Rapid Timing Optimization for Design Prototyping
  - Using Additional PreCTS Timing Optimization Parameters
  - Performing Incremental PreCTS Optimization
  - Changing Default Settings in PreCTS Mode
- Performing PostCTS Optimization
  - Correcting Violations in PostCTS Mode
    - Skipping Path Groups During Hold Fixing
    - Reporting Violations Remaining After Hold Fixing
  - Using Additional PostCTS Timing Optimization Parameters
  - Performing Incremental PostCTS Optimization
  - Changing Default Settings in PostCTS Mode
- Performing PostRoute Optimization
  - About PostRoute Optimization
  - Correcting Violations and Signal Integrity Issues using GigaOpt Technology in postRoute Mode
  - GigaOpt in PostRoute Setup Timing Flow
  - GigaOpt in PostRoute Hold Timing Flow
    - GigaOpt in PostRoute Use Model
  - Changing Default Settings in postRoute Mode
- Optimizing Power During optDesign
  - Leakage Power Optimization
  - Dynamic Power Optimization
  - Leakage and Dynamic Power Optimization Combined
  - Migrating from –leakagePowerEffort and -dynamicPowerEffort to –powerEffort
  - Specifying the Correct Power Views for Optimization
- Using Useful Skew
  - Using Useful Skew in PreCTS Mode
  - Using Useful Skew in PostCTS Mode
  - Controlling Useful Skew Optimization
- Distributed Timing Analysis for Hold Fixing
- Using Active Logic View for Chip-Level Interface Circuit Timing Closure
- Optimizing Timing in On-Chip Variation Analysis Mode

- Specifying the MMMC Environment
- Optimizing Timing in OCV Mode Using the Default Delay Calculator
- Using Conformal Constraint Designer During Timing Optimization
  - Post-Processing Approach
  - Integrated Approach
- Optimizing Timing Using a Rule File
- Optimizing Timing When the Constraint File Includes the set\_case\_analysis Constraint
- Using the Footprintless Flow
- Using Cell Footprints
- Viewing Added Buffers, Instances, and Nets
  - Default Naming Conventions
- Using Signoff Timing Analysis to Optimize Timing
- Running MMMC SignOff ECO within Innovus
  - Signoff Timing Analysis in Innovus using Timing Debug
  - Enabling PBA Timing Analysis and Optimization
  - Path Group Support
  - Sample Template Scripts

## Overview

Optimize timing after running trial route and extracting RCs, after clock tree synthesis (CTS), and after routing. The goals of timing optimization are to correct design rule violations (DRVs) and signal integrity (SI) violations and meet timing. Timing optimization includes the following operations, depending on the design stage:

- Adding buffers
- Resizing gates
- Restructuring the netlist
- Remapping logic
- Swapping pins
- Deleting buffers
- Moving instances
- Applying useful skew

Use the `setOptMode` command (or the Tools - Set Mode- Mode Setup form) to specify global timing optimization parameters. Use the `optDesign` super command (or the ECO - Optimize Design form) to optimize timing.

Use the `place_opt_design` command to do placement and preCTS optimization together.

## Before You Begin

Before you optimize timing for the first time, complete the following steps:

1. Reserve placement space of more than five percent of the targeted final design utilization so that there is room to add buffers and remap the network to meet timing requirements.
2. Specify default and detailed extraction scale factors by using the following commands:
  - `generateRCFactor`
  - `create_rc_corner`
4. Use the following method to set input transitions for the high fanout nets for delay calculation:
  - Run the `delaycal_input_transition_delay` command.
6. Create and load footprints. (Optional)

You are not required to specify footprints. For more information, see the [Using the Footprintless Flow section](#).

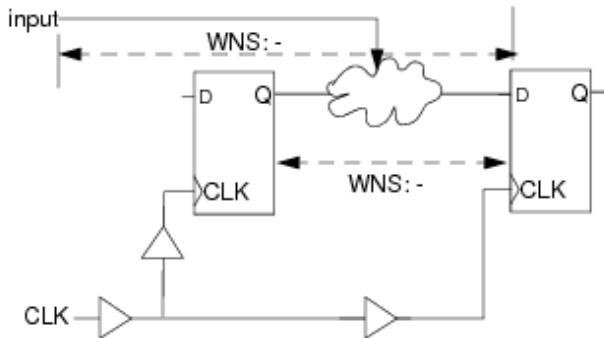
## Results

After optimizing timing, the Innovus™ Implementation System (Innovus) appends the log file with the following information:

- Worst negative slack, total negative slack (TNS) and the number of failing (violating) paths. The software also reports hold violations if you specify the `-hold` parameter in postCTS or postRoute mode. It writes the values to the log file and writes reports to the working directory.

**Note:** The overall TNS and number of failing paths of a design might not be equal to the total of the TNS and failing paths of the individual path groups. This is because the TNS and number of failing paths are based on the end-point of the path and are not path based.

For example, the following figure has a register with two paths, one from a primary input with a slack of -0.6ns and other from another register with a slack of -0.3ns.



In this case the overall TNS will be -0.6ns with 1 violating path (end point-based). But the individual reg2reg TNS will be -0.3ns with 1 violating path and input to register with a TNS of -0.6ns with 1 violating path. Therefore, the sum total of individual path group TNS is not the same as overall TNS.

- Number of `max_tran`, `max_cap`, and `max_fanout` violations
- Utilization (density)

If you specify path groups, the software produces a slack file and `tarpt` report for them. If you do not specify path groups, the software produces the following violation reports:

- Register-to-register
- Register-to-clock-gate
- Default: Includes all other paths, including paths to and from inputs/outputs.

The reports contain information about the following violations for the top 50 critical paths:

- Setup violations
- Hold violations
- DRVs (maximum capacitance, maximum transition, and maximum fanout violations)

The software generates the reports and saves them in the file specified by `optDesign -outDir` (or in the `timingReports` directory if `-outDir` is not specified).

The filenames are as follows:

- `designName_preCTS_pathGroup.tarpt`
- `designName_postCTS_pathGroup.tarpt`

- `designName_postRoute_pathGroup.tarpt`

The summary report has the following format:

```
-+-----+-----+-----+-----+  
| Setup mode      | all     | reg2reg | reg2cgate |  
+-----+-----+-----+-----+  
WNS (ns):	-0.491	-0.491	N/A
TNS (ns):	-866.856	-866.856	N/A
Violating Paths:	5273	5273	N/A
All Paths:	69372	69343	N/A
+-----+-----+-----+-----+			
+-----+-----+-----+			
	Real	Total	
DRVs +-----+-----+-----+			
	Nr Nets(terms)	Worst Vio	Nr Nets (terms)
+-----+-----+-----+			
max_cap	70(70)	-0.004	70(70)
max_tran	0(0)	.000	0(0)
max_fanout	0(0)	0	0(0)
+-----+-----+-----+  
  
Density: 53.455%  
Routing Overflow: 0.00% H and 0.00% V
```

For more information on timing reports, see the [timeDesign](#) command.

## Interrupting Timing Optimization

To stop timing optimization, use the Ctrl+C key combination. On pressing Ctrl+C, the command runs until the database is in a state where the command can stop safely.

When the command stops, the software presents the Interrupt menu. For information on the Interrupt menu, see "Interrupting the Software" in the [Getting Started](#) chapter.

**Warning:** When you interrupt the software with `ctrl+C`, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.

## Performing Optimization Before Clock Tree Synthesis

The following topics are covered in this section:

- Correcting Violations in PreCTS Mode for the First Time
- Performing Rapid Timing Optimization for Design Prototyping
- Using Additional PreCTS Timing Optimization Parameters
- Performing Incremental PreCTS Optimization
- Changing Default Settings in PreCTS Mode

### Correcting Violations in PreCTS Mode for the First Time

- Before optimizing timing in preCTS mode, you must break all timing loops by disabling arcs in the constraint file. If you do not disable the arcs, the software cannot make a valid comparison of WNS between two different runs since it might not break the loops at the same point each time. Use the following command:

`set_disable_timing`

- Use the following command to optimize timing:

`optDesign -preCTS`

- To repair only DRVs, use the following command:

`optDesign -preCTS -drv`

**Note:** By default, the `optDesign` does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

`setOptMode -fixFanoutLoad true`

## Performing Rapid Timing Optimization for Design Prototyping

To optimize timing using low-effort mode for design prototyping, use the following commands:

```
setOptMode -effort low  
optDesign -preCTS
```

In low-effort mode, `optDesign` resizes gates and performs global buffer insertion and repair DRVs, but does not restructure the netlist.

## Using Additional PreCTS Timing Optimization Parameters

You can use the following `optDesign` features either separately or in combination.

- To run optimization with useful skew, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -preCTS
```

- To run optimization on specific path groups, use the following commands:

```
setAnalysisMode -honorClockDomains false  
group_path -name groupName -from sourcePoint -to destinationPoint  
setPathGroupOptions groupName -effortLevel high  
optDesign -preCTS
```

- To run optimization on register-to-register paths, use the following commands:

```
setAnalysisMode -honorClockDomains false  
createBasicPathGroups  
setPathGroupOptions reg2reg -effortLevel high
```

- To disable area reclaiming, use the following commands (`optDesign` reclaims area by default):

```
setOptMode -reclaimArea false  
optDesign -preCTS
```

## Performing Incremental PreCTS Optimization

Optimize timing incrementally to optimize setup times and area on critical paths. You can use the following features either separately or together.

- To run incremental setup-only optimization, use the following command:

```
optDesign -preCTS -incr
```

- To run incremental optimization with useful skew, use the following commands:

```
setOptMode -usefulSkew true
```

```
optDesign -preCTS -incr
```

- To run incremental optimization on specific path groups, use the following commands:

```
setAnalysisMode -honorClockDomains false
```

```
group_path -name groupName -from sourcePoint -to destinationPoint
```

```
setPathGroupOptions groupName -effortLevel high
```

```
optDesign -preCTS -incr
```

For a list of supported source and destination points, see the [Using Additional PostCTS Timing Optimization Parameters](#) section.

- To run incremental optimization on register-to-register paths, use the following commands:

```
setAnalysisMode -honorClockDomains false
```

```
createBasicPathGroups
```

```
setPathGroupOptions reg2reg -effortLevel high
```

```
optDesign -preCTS -incr
```

## Changing Default Settings in PreCTS Mode

You can change or add parameters for the following commands that `optDesign` runs automatically:

|                              |                                                                                                                                                                          |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setAnalysisMode</code> | optDesign sets <code>-clkSrcPath false</code> and <code>-clockPropagation forcedIdeal</code> by default. You cannot override these values. You can add other parameters. |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

optDesign sets `-clkSrcPath false` and `-clockPropagation forcedIdeal` by default. You cannot override these values. You can add other parameters.

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setExtractRCMode</code>  | <p><code>optDesign</code> sets the extraction mode to <code>default</code>. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>setOptMode</code>        | <p><code>optDesign</code> sets the following parameters:</p> <ul style="list-style-type: none"> <li>● <code>-drcMargin</code><br/>                     If you use <code>setOptMode -drcMargin</code>, the value you specify is added to a dynamically calculated, internal margin. For example, if you set a margin of <code>0.2</code> (20 percent), this multiplies the <code>max_cap</code> and <code>max_tran</code> SDC constraints by 0.8. The margin can be positive or negative. If you set a margin of <code>-0.2</code>, this multiplies the <code>max_cap</code> and <code>max_tran</code> SDC constraints by 1.20. <code>optDesign</code> writes the margin value to the log file.</li> <li>● <code>-holdTargetSlack</code><br/>                     If you use <code>setOptMode -holdTargetSlack</code>, the value you specify is added to a dynamically calculated, internal margin. The <code>optDesign</code> command writes the hold target slack value to the log file.</li> <li>● <code>-setupTargetSlack</code><br/>                     If you use <code>setOptMode -setupTargetSlack</code>, the value you specify is added to a dynamically calculated, internal margin. The default <code>-setupTargetSlack</code> value is <code>0</code>. The <code>optDesign</code> command writes the setup target slack value to the log file.</li> </ul> |
| <code>setTrialRouteMode</code> | <p>You can add parameters, but you cannot override the default settings. The <code>optDesign</code> command sets the <code>-handlePreroute</code> parameter to <code>true</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

## Performing PostCTS Optimization

The following topics are covered in this section:

- Correcting Violations in PostCTS Mode
- Performing Incremental PostCTS Optimization

- Changing Default Settings in PostCTS Mode

## Correcting Violations in PostCTS Mode

- To optimize timing after the clock tree is built, use the following commands:

```
optDesign -postCTS
```

`optDesign` in postCTS fixes DRVs, reclaims area, and fixes setup violations.

**Note:** If using CCOpt, instead of CCOpt-CTS, then additional post-CTS setup optimization is not normally required as CCOpt combines CTS and post-cts style datapath optimization. Refer to [Clock Tree Synthesis](#) for further details of CCOpt.

By default, the `optDesign` does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

```
setOptMode -fixFanoutLoad true
```

- To repair setup and hold violations, use the following commands:

```
optDesign -postCTS
```

```
optDesign -postCTS -hold
```

- To repair only DRVs, use the following command:

```
optDesign -postCTS -drv
```

- To repair only hold violations, use the following command:

```
optDesign -postCTS -hold
```

## Skipping Path Groups During Hold Fixing

You can instruct the Innovus software to exclude path groups from hold fixing by using the `-ignorePathGroupsForHold` parameter of the `setOptMode` command.

## Reporting Violations Remaining After Hold Fixing

You can generate report files to view the hold violation paths that remain after hold fixing by using the following commands:

```
optDesign -postCTS -hold -holdVioData fileName
```

```
optDesign -postRoute -hold -holdVioData fileName
```

## Using Additional PostCTS Timing Optimization Parameters

You can focus timing optimization on specific paths using path groups. Running the `createBasicPathGroups` command creates reg2reg and reg2cgate (if present in the design) path groups and sets them to high effort. It also creates the in2reg reg2out and in2out path groups and sets these to low effort. To optimize these path groups, run the following commands:

```
createBasicPathGroups  
optDesign -postCts
```

If path groups are not defined, the `optDesign` command will temporarily generate two high effort `path_groups` (reg2reg and reg2clkgate).

- A `path_group` can be set as “low” or “high” effort.
  - A high-effort `path_group` receives a higher focus from the optimization engine than a low effort `path_group`
- All high-effort path groups that are defined are optimized at the same time.
- You can add slack adjustment and priority to any path group using the `setPathGroupOptions` command.
  - By default, `optDesign` will use the slack adjustment value that leads to the worst slack
  - The priority is used when an endpoint is a part of several path groups so the software can choose the adjustment value to be used

The flow to create and optimize path groups is as follows:

```
group_path -name path_group_name -from from_list -to to_list -through through_list  
setPathGroupOptions ...  
optDesign -preCTS -incr
```

Creating path groups is not mandatory to achieve the best results because of the following reasons:

- Too many custom path groups may impact the runtime significantly.
- Too many overlapping or nested path groups may impact TNS timing closure.

## Performing Incremental PostCTS Optimization

- To optimize setup time incrementally and reduce area, use the following commands:  

```
setOptMode -reclaimArea true  
optDesign -postCTS -incr
```
- To take advantage of useful skew when optimizing timing in incremental postCTS mode, use the following commands:  

```
setOptMode -usefulSkew true  
optDesign -postCTS -incr
```

If you have already performed detail routing on the clock tree, the software performs global and detailed ECO routing automatically using the NanoRoute router in postCTS useful skew mode. If you do not want the software to do this, specify the `-noECORoute` parameter, as follows:

```
setOptMode -usefulSkew true  
optDesign -postCTS -noECORoute -incr
```

If you specify `-noECORoute`, `optDesign` performs trial routing to estimate clock delays.

- To run incremental postCTS optimization if your design has a clock mesh, use the following commands:

```
setOptMode -usefulSkew false  
optDesign -postCTS -incr
```

## Changing Default Settings in PostCTS Mode

You can change or add parameters for the following commands and parameters that `-optDesign` runs automatically:

|                              |                                                                                                                                                                                                           |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setAnalysisMode</code> | <code>optDesign</code> sets <code>-clockPropagation</code> <code>autoDetectClockTree</code> and <code>-clkSrcPath true</code> by default. You cannot override these values. You can add other parameters. |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

`setExtractRCMode`

`optDesign` sets the extraction mode to `-engine preRoute`. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.

## Performing PostRoute Optimization

The following topics are covered in this section:

- About PostRoute Optimization
- Correcting Violations and Signal Integrity Issues using GigaOpt Technology in PostRoute Mode
- Correcting Signal Integrity Violations

## About PostRoute Optimization

In `postRoute` mode, the Innovus software corrects setup violations and design rule violations unless you specify otherwise. It first operates on register-to-register paths (and register-to-clocks path groups, if present), and then on the default path group in `setDesignMode -flowEffort high` mode. The software performs incremental RC extraction and delay calculation, and runs the NanoRoute router to perform ECO routing. In case the final timing after ECO routing is degraded as compared to what was expected before ECO routing, the software automatically calls an incremental optimization to recover the setup timing.

If filler cell definitions were provided during design import, the `optDesign` command removes or adds them as needed, following the information given by the `setFillerMode` command.

If only non-SI Timing is being looked at, there should be very few timing violations that need correction. The primary sources of these violations include the following:

- Inaccurate prediction of the routing topology during `preRoute` optimization due to congestion-based detour routing
- Minor correlation issues between default and detailed RC extraction.

- i** Because the violations at this stage are due to inaccurate modeling of the final route topology and the attendant parasitics, it is critical not to introduce additional topology changes beyond those needed to correct the existing violations.

Making unnecessary changes to the routing at this point can lead to a scenario where fixing one violation leads to the creation of other violations. This cascading effect creates a situation where it becomes impossible to close on a final timing solution with no DRVs. One of the strengths of postRoute optimization is its ability to simultaneously cut a wire and insert buffers, create the new RC graph at the corresponding point, and modify the graph to estimate the new parasitics for the cut wire without re-running extraction.

To take even more advantage of this feature, you can provide an external SPEF file generated by a sign-off extraction tool for improved convergence. However, you must provide a full SPEF (reduced SPEF does not work) and one of the following conditions must be met:

- The SPEF file must be generated with node locations using the starN format.  
or
- The resistance values in the LEF file must match those in the technology file used by signoff extraction to generate the SPEF file, which enables the Innovus extraction engine to match the routes with the SPEF RC graph.

## Correcting Violations and Signal Integrity Issues using GigaOpt Technology in postRoute Mode

GigaOpt technology is default for the postRoute flow, including setup/hold/power optimization. GigaOpt simplifies the postRoute closure flow. It is recommended that onChipVariation should be turned on and CPPR is used as shown below:

```
setAnalysisMode -analysisType onChipVariation -cppr both
```

## GigaOpt in PostRoute Setup Timing Flow

GigaOpt technology has three phases:

- Design-rule violations fixing
- SI slew and glitch fixing
- Setup timing fixing on base and SI delay

GigaOpt does multi-threading combined base and SI delay timing optimization. It supports the following:

- Leakage and dynamic power optimization
- External SPEF-flow (with node location)
- External SDF-flow
- Filler cells deletion/insertion
- Smart ECO routing
- ILM/GigaFlex flow and MSV flow

## GigaOpt in PostRoute Hold Timing Flow

In setup aware hold fixing, hold violations are fixed while having the full setup timing graph in memory.

GigaOpt supports the following:

- External SPEF-flow
- External SDF-flow
- Filler cells deletion/insertion
- Smart ECO routing

GigaOpt hold fixing generates detailed diagnostic report for all remaining hold-violated nets. GigaOpt hold fixing performs the following:

- Buffer insertion along the route
- Wire and RC cutting
- Legal location searching
- Cell resizing
- Distributed hold analysis

## GigaOpt in PostRoute Use Model

To optimize timing setup and hold with base and SI delay, use the following commands:

```
optDesign -postRoute  
optDesign -postRoute -hold
```

To optimize timing setup and hold with base delay only, use the following commands:

```
setDelayCalMode -SIAware false  
optDesign -postRoute  
optDesign -postRoute -hold
```

**Note:** For run-time reduction, you can also perform combined setup and hold optimization. Use the

following command instead of running both `optDesign -postRoute` and `optDesign -postRoute -hold`:

```
optDesign -postRoute -hold -setup
```

## Examples

- GigaOpt hold optimization will always try to fix all hold TNS. To run GigaOpt for performing setup timing TNS optimization on base and SI delay, use the following commands:

```
setOptMode -flowEffort high
optDesign -postRoute
```

- To run GigaOpt for performing setup and leakage timing optimization on base and SI delay, use the following commands:

```
setOptMode -leakagePowerEffort high
optDesign -postRoute
```

- To run GigaOpt for performing setup with SI slews optimization on base and SI delay, use the following commands:

```
setOptMode -fixSISlew true
optDesign -postRoute
```

**Note:** By default, the `optDesign` command does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

```
setOptMode -fixFanoutLoad true
```

**Note:** Hold repair does not degrade the setup worst slack to less than the original value or the `setupTargetSlack` value. You can override the `setupTargetSlack` value by specifying `setOptMode -setupTargetSlack` before you run `optDesign`. By default, hold repair is allowed to degrade the setup total negative slack. Therefore, to disable this feature, set the following:

```
setOptMode -fixHoldAllowSetupTnsDegrade false
```

You can instruct Innovus to exclude path groups from hold fixing. For more information, see "Skipping Path Groups During Hold Fixing".

- To take clock reconvergence pessimism removal (CRPR) into consideration when running timing optimization, use the `setAnalysisMode` command before you run `optDesign`. For example:

```
setTimingDerate -max -clock -early 0.8 -late 1.2
setTimingDerate -min -clock -early 0.8 -late 1.2
setAnalysisMode -cppr both
```

```
optDesign -postRoute
```

- To run postRoute timing optimization on designs containing Interface Logic Models (ILMs), use the following command:

```
optDesign -postRoute
```

This will automatically flatten ILMs, optimize timing, and then unflatten the ILMs.

- To run postRoute setup or hold optimization based on external SPEF files (with node locations using the `starN` format) for a design with four active RC corners (two for setup and two for hold), run the following:

```
spefIn -rc_corner cornerMax1 rcMax1.spef  
spefIn -rc_corner cornerMax2 rcMax2.spef  
spefIn -rc_corner cornerMin1 rcMin1.spef  
spefIn -rc_corner cornerMin2 rcMin2.spef  
optDesign -postRoute [-hold]
```

**Note:** You must provide SPEF information for each active `rc_corner` (setup and hold). If one corner does not have a SPEF, the software will rerun RC extraction for every corner.

- To run postRoute optimization based on external SDF files (that contain only non-SI timing) for a design with two active setup views and two active hold views, use the following commands:

```
read_sdf -view viewMax1 sdfMax1.sdf  
read_sdf -view viewMax2 sdfMax2.sdf  
read_sdf -view viewMin1 sdfMin1.sdf  
read_sdf -view viewMin2 sdfMin2.sdf  
optDesign -postRoute [-hold] -useSDF
```

## Changing Default Settings in postRoute Mode

You can change or add parameters for the following commands that `optDesign` runs automatically:

`setAnalysisMode`

`optDesign` sets `-clockPropagation` `autoDetectClockTree` and `-clkSrcPath true`. You can override other parameters.

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setExtractRCMode</code> | optDesign sets the extraction mode to <code>-engine postRoute -effortLevel medium</code> ( <code>tQuantus</code> ). If no Quantus QRC techfile is available then <code>-effortLevel low</code> will be used. Note, you can always change this to <code>-effortLevel high</code> ( <code>IQRC</code> ) or <code>-effortLevel signoff</code> (Standalone QRC) but both require a Quantus QRC license. Ensure that you set the appropriate extraction scale factors if using the low/medium/high effort levels. |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Optimizing Power During optDesign

During timing optimization, when the correct power effort is specified, the software is fully aware of the impact of each optimization technique in terms of both leakage and, if specified, dynamic optimization, and it will choose the best one considering all metrics. This functionality is called power-driven optimization. It specifically calls both leakage and dynamic power optimizations.

### Leakage Power Optimization

To activate leakage power optimization during timing optimization, run the following command:

```
setOptMode -powerEffort none | low | high –leakageToDynamicRatio 1.0
```

- When `-powerEffort` is set to `none`, `optDesign` performs minimal leakage-power optimization without any real impact on the timing and DRV QOR. This is the default option.
- When `-powerEffort` is set to `low`, `optDesign` will optimize leakage power after each timing optimization step. It is fully leakage power-aware at every step of optimization and makes decisions based on that. There are no high leakage cells inserted during hold fixing.
- When `-powerEffort` is set to `high`, `optDesign` will optimize leakage power at all stages and it will use a more complex preRoute flow to gain more leakage savings than just a simple call to leakage optimization. There are no high leakage cells inserted during hold fixing. During timing optimization, `optDesign` will use high-effort techniques to ensure the best possible leakage impact of each timing optimization change. For more details, see the "Low Power App Note" available on the Cadence Online Support (COS) website.

**Note:** To achieve the best leakage power results, load all the different Vth libraries. It is important to ensure that the correct power view is specified using the `-analysis_view` parameter of the `set_power_analysis_mode` command. The optimal view for leakage is the one with higher temperature corners (85/125 degrees) and typical libraries.

You can also call leakage power optimization using the standalone command, `optLeakagePower` . This command lets you perform a standalone leakage power optimization at any point after placement without the need to call it within the `optDesign` command. Although this provides more flexibility but, in general, it is recommended that power-driven optimization should be used within `optDesign`.

## Dynamic Power Optimization

To activate dynamic power optimization during timing optimization, run the following command:

```
setOptMode -powerEffort none | low | high –leakageToDynamicRatio 0.0
```

- When `-powerEffort` is set to `none`, `optDesign` will not optimize dynamic power. This is the default option.
- When `-powerEffort` is set to `low`, `optDesign` will only optimize dynamic power in the preCTS setup optimization phase.
- When `-powerEffort` is set to `high`, `optDesign` will optimize dynamic power in the entire setup optimization phases

**Note:** Ensure that the correct power view is specified using the `-analysis_view` parameter of the `set_power_analysis_mode` command. It is also recommended that you provide an activity file. This can be done by using the following command:

```
read_activity_file -format {VCD | TCF | SAF | FSDB} file_name
```

In the absence of a switching file, it is recommended you use the following command:

```
set_default_switching_activity -input_activity 0.2 -seq_activity 0.2
```

This will ensure both predictability and consistency throughout the flow.

You can also call dynamic power optimization using the standalone command, `optDynamicPower` . This command lets you do a standalone switching and internal power optimization at any point after placement without the need to call it within the `optDesign` command. This provides more flexibility but, in general, it is recommended that power-driven optimization should be used within `optDesign`.

## Leakage and Dynamic Power Optimization Combined

To activate leakage and dynamic power-driven optimization together during timing optimization, you can specify a `-leakageToDynamicRatio` value between 0.0 and 1.0. In general, there is little value in specifying any value that is not an increment of 0.1.

**Note:** To decide on the value, it is recommended that you run the `report_power` command once the

design has been setup correctly with respect to the following:

- Correct power views are specified
- MVT library is setup
- Switching activity is supplied, and so on

If overall total power reduction is the goal, then looking at the overall impact of internal and switching power versus leakage power, you can decide on the ratio to be specified to achieve the greatest impact. Some examples are provided below:

- `setOptMode -leakageToDynamicRatio 0.5` : This implies that timing optimization will be both dynamic and leakage power-driven and trade-offs need to be done to achieve a balance between leakage and dynamic power optimization.
- `setOptMode -leakageToDynamicRatio 0.1` : This implies that timing optimization will be both dynamic and leakage power-driven but dynamic optimization will be favored.
- `setOptMode -leakageToDynamicRatio 0.9` : This implies that timing optimization will be both dynamic and leakage power-driven but leakage optimization will be favored.

## Migrating from **-leakagePowerEffort** and **-dynamicPowerEffort** to **-powerEffort**

This section provides a rough guide on how to move from the previous leakage and dynamic power optimization flows to using power-driven optimization. Since power-driven optimization functionality is introduced in the Innovus 15.1 release, there is no direct one to one mapping from previous to new. However, some close approximations are provided below:

- `setOptMode -leakagePowerEffort low -dynamicPowerEffort none` is roughly equivalent to  
`setOptMode -powerEffort low leakageToDynamicRatio 1.0`
- `setOptMode -leakagePowerEffort low -dynamicPowerEffort low` is roughly equivalent to  
`setOptMode -powerEffort low leakageToDynamicRatio 0.5`
- `setOptMode -leakagePowerEffort low -dynamicPowerEffort high` is roughly equivalent to  
`setOptMode -powerEffort high leakageToDynamicRatio 0.1`
- `setOptMode -leakagePowerEffort high -dynamicPowerEffort none` is roughly equivalent to  
`setOptMode -powerEffort high leakageToDynamicRatio 1.0`
- `setOptMode -leakagePowerEffort high -dynamicPowerEffort low` is roughly equivalent to  
`setOptMode -powerEffort high leakageToDynamicRatio 0.9`
- `setOptMode -leakagePowerEffort high -dynamicPowerEffort high` is roughly equivalent to

```
setOptMode -powerEffort high leakageToDynamicRatio 0.5
```

- setOptMode -leakagePowerEffort none -dynamicPowerEffort none is roughly equivalent to  
setOptMode -powerEffort none leakageToDynamicRatio <any value>
- setOptMode -leakagePowerEffort none -dynamicPowerEffort low is roughly equivalent to  
setOptMode -powerEffort low leakageToDynamicRatio 0.0
- setOptMode -leakagePowerEffort none -dynamicPowerEffort high is roughly equivalent to  
setOptMode -powerEffort high leakageToDynamicRatio 0.0

## Specifying the Correct Power Views for Optimization

As mentioned above, it is important to specify the correct leakage and dynamic view for optimization. The optimal view for leakage is the one with higher temperature corners (85/125 degrees) and typical libraries. The optimal view for dynamic power is dependent both on the design and on your inputs. Ensure that the activity is provided by one or the other methods mentioned above. For specifying the power view, consider the following:

- If the leakage and dynamic view is to be the same, then run the following command:

```
set_power_analysis_mode -leakage_power_view power_view_name -dynamic_power_view
power_view_name
```

You can still use the `-analysis_view power_view_name` parameter but this parameter will be made obsolete in a future release, so it is not recommended.

- If the leakage and dynamic view is to be different, then run the following command:

```
set_power_analysis_mode -leakage_power_view leakage_view_name -dynamic_power_view
dynamic_view_name
```

If the view is not an active view, it will be automatically handled by the optimization code. However, the `report_power` command does not support non-active views. So, for this command, you will need to add the view to the active views using the `set_analysis_view` command and then call the `report_power -view power_view_name` command. Also, in terms of leakage, if the view is not active then the optimization will be forced to set the `-state_dependent_leakage` parameter of the `set_power_analysis_mode` command to `false`.

**Note:** If you want to have state-dependent leakage (`-state_dependent_leakage true`) optimization, then the view needs to be made part of the active view list. Also, it is important to ensure that the specified views used are always well defined from both a power and timing point of view to get the optimal QOR.

## Using Useful Skew

The useful skew feature in the Innovus software modifies the clock arrival time on sequential elements to improve the datapath timing between sequential elements. The software provides two approaches to using useful skew, depending on whether you have run CTS or not:

- PreCTS mode  
Advances the clock signal for critical path start points. The start point must be a sequential element: No input paths are allowed.
- PostCTS mode  
Delays the clock signal for critical path end points. The end point must be a sequential element: No output paths are allowed.

## Using Useful Skew in PreCTS Mode

To take advantage of useful skew during preCTS optimization, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -preCTS
```

The software determines the sequential instances whose clock signals can be advanced, and then generates the following two files:

- latency\_file.sdc  
This latency file models the proposed clock advancement for timing analysis.
- schedule\_file.cts.pid  
This file contains scheduling information for clock tree synthesis. You must specify this file when you specify the CTS constraints. For example:

```
specifyClockTree -clkfile schedule_file.cts.pid  
specifyClockTree -clkfile original_constraints.cts
```

You can change the names of the latency and scheduling files by using the following commands:

- `setLatencyFile fileName`
- `setSchedulingFile fileName`

Use the following commands to retrieve the names of the latency and schedule files:

- `getLatencyFile`
- `getSchedulingFile`

## Using Useful Skew in PostCTS Mode

To take advantage of useful skew during postCTS optimization, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -postCTS
```

In this case, the clock tree is already in place. The software determines the sequential instances whose clock signals can be delayed, and adds buffers or inverters to their clock nets accordingly. If the clock is already detail routed, these commands perform ECO routing on the clock tree after useful skew optimization.

## Controlling Useful Skew Optimization

To control how the Innovus software employs useful skew, use the following command:

```
setUsefulSkewMode
```

If you choose specific cells for clock tree synthesis, use `setUsefulSkewMode -useCells` to specify the cells to use for padding the clock nets. If you have no constraint on the type of cells allowed in the clock tree, you can omit this parameter, and the software selects the best combination of cells to achieve the required delay. For example, if you want clock buffers or inverters only, specify the following command:

```
setUsefulSkewMode -useCells {...}
```

To advance or delay sequential elements more aggressively than it does by default, without degrading the worst negative slack, use the following command:

```
setUsefulSkewMode -maxSkew true
```

When you specify this parameter, the tool skews other registers as much as possible regardless of the worst slack on a particular register. This approach can help with difficult timing closure situations. In postCTS mode, critical paths probably have been fully optimized, so further traditional optimization cannot dramatically improve timing.

To close timing, you might need to delay the endpoint clock pins more than the useful skew feature would do by default, by only padding the clock nets until the data path meets the target slack. To take advantage of this feature, use the following command:

```
setUsefulSkewMode -maxSkew true
```

To exclude boundary sequential cells in useful skew calculations, use the following command:

```
setUsefulSkewMode -noBoundary true
```

If you do not specify this parameter, the software takes boundary cells and ordinary sequential elements into account when calculating useful skew.

To use NanoRoute detailed routing to route nets that are added or changed during useful skew optimization, use the following command:

```
setUsefulSkewMode -ecoRoute true
```

To limit the amount of slack, the innovus software can borrow from neighboring flip-flops when performing useful skew operations, use the following command:

```
setUsefulSkewMode -maxAllowedDelay true
```

The Innovus delay calculation and RC extraction methods might differ from those of other sign-off tools, so other setup violations might occur if Innovus borrows too much slack. By having control over slack borrowing, you can prevent these setup violations. Limiting borrowed skew also limits the clock tree skew to avoid large hold violations. If you do not specify this parameter, the Innovus software automatically borrows the amount of slack needed (there is no maximum) to reduce setup violations.

To report the current `setUsefulSkewMode` settings, use the following command:

```
getUsefulSkewMode
```

## Distributed Timing Analysis for Hold Fixing

In hold fixing, more than half of the CPU runtime is spent in computing the setup and hold timing. This is done one time before the fixing process and once after this process. In Innovus, the setup and hold timing analysis are distributed.

The distribution is either local or on remote hosts, depending on the EDP settings applied using the `setMultiCpuUsage` command. It is enabled by default and the use model (on a local machine) is :

```
setMultiCpuUsage -localCpu number
```

```
optDesign -hold -postCts/-postRoute
```

**Note:** The timing computed in the distributed mode can be different than in default mode when `set_global_timing_cppr_threshold_ps` is applied with a value higher than 1ps. But this does not impact timing convergence since in distributed mode the timing would always be on the pessimistic side.

# Using Active Logic View for Chip-Level Interface Circuit Timing Closure

The Innovus software provides a top-level interface timing operation flow to perform partitioning and budgeting on a trimmed-down version of the timing graph: an active logic view. This flow helps you close the timing issues of the interface top-level paths as your design has gone through the hierarchical flow until the postRoute stage. This flow also saves the memory usage and provides faster runtime on large designs.

To perform optimization using an active logic view at the postRoute stage, complete the following steps:

1. Load the hierarchical design in the database that is created by `-assembleDesign` using the entire post-routed block partition and the top-level partition. Specify the partition information in the database.

```
restoreDesign assembled.enc.dat toplevel_design_name
```

2. Perform timing analysis on the design to identify the timing of the full-chip design.

```
timeDesign -postRoute -prefix preOpt
```

3. Set the optimization mode to use active logic view. If you specify this parameter, `optDesign` observes the floorplan fence constraint when moving or adding cells.

```
setOptMode -virtualPartition true
```

4. Run `optDesign`. The `optDesign` command honors active logic view.

```
optDesign -postRoute
```

5. Perform timing analysis again to ensure that there are no timing issues.

```
timeDesign -postRoute -prefix postOpt
```

## Optimizing Timing in On-Chip Variation Analysis Mode

PostRoute timing optimization must be done using on-chip variation (OCV) to account for variations in process, voltage, and temperature (PVT) across the die. When it takes OCV into account, the software calculates early and late delays, and uses them to evaluate setup and hold timing checks. You introduce the delays into the analysis by specifying different min/max corner timing libraries and operating conditions. Early/late variation might also be present due to slew merging effects of multiple input gates in the clock path.

To enable the software to consider multiple libraries and operating conditions, you must specify a multi-mode/multi-corner (MMMC) environment. The MMMC environment must be set up and OCV must be turned on, otherwise the `optDesign` command exits with an error message.

For more information on OCV and MMMC, see the following sections:

- On-Chip Variation (OCV) Timing Analysis Mode
- Configuring the Setup for Multi-Mode Multi-Corner Analysis

## Specifying the MMMC Environment

There are three MMMC scenarios for timing optimization in OCV mode:

- One library and one operating condition per corner
- One library and two operating conditions per corner
- Two worst-case libraries and two best-case libraries per corner

The operating condition specifications you provide to the `create_delay_corner` command determine the MMMC scenario for OCV mode. These specifications give the software the values to use for early and late timing.

The following sections show the specifications necessary for each scenario. The differences are highlighted in bold-face type.

- One library and one operating condition per corner

```
create_library_set -name libs_min -timing [list $bestcase_lib]
create_library_set -name libs_max -timing [list $worstcase_lib]
create_rc_corner -name rc_worst -cap_table CMAX.capTbl
create_rc_corner -name rc_best -cap_table CMIN.capTbl
create_constraint_mode -name postCTS [list xxx.sdc]
create_delay_corner -name delay_corner_max \
    -library_set libs_max \
    -opcond_library stdcmos90T125 \
    -opcond cmos90T125 \
    -rc_corner rc_worst
create_delay_corner -name delay_corner_min \
    -library_set libs_min \
    -opcond_library stdcmos90Tm40 \
    -opcond cmos90Tm40 \
    -rc_corner rc_best
```

```
create_analysis_view -name postCts_max \
    -delay_corner delay_corner_max \
    -constraint_mode postCTS

create_analysis_view -name postCts_min \
    -delay_corner delay_corner_min \
    -constraint_mode postCTS

set_analysis_view -setup postCts_max -hold postCts_min
```

- One library and two operating conditions per corner

```
create_library_set -name libs_min -timing [list $bestcase_lib]
create_library_set -name libs_max -timing [list $worstcase_lib]

create_rc_corner -name rc_worst -cap_table CMAX.capTbl
create_rc_corner -name rc_best -cap_table CMIN.capTbl

create_constraint_mode -name postCTS [list xxx.sdc]

create_delay_corner -name delay_corner_max \
    -library_set libs_max \
    -late_opcond_library stdcmos90T125 \
        -late_opcond cmos90T125_slow \
        -early_opcond_library stdcmos90T125 \
        -early_opcond cmos90T125 \
    -rc_corner rc_worst

create_delay_corner -name delay_corner_min \
    -library_set libs_min \
    -late_opcond_library stdcmos90Tm40 \
        -late_opcond cmos90Tm40 \
        -early_opcond_library stdcmos90Tm40 \
        -early_opcond cmos90Tm40_fast \
    -rc_corner rc_best

create_analysis_view -name postCts_max \
    -delay_corner delay_corner_max \
    -constraint_mode postCTS

create_analysis_view -name postCts_min \
```

```
-delay_corner delay_corner_min \
-constraint_mode postCTS

set_analysis_view -setup postCts_max -hold postCts_min
```

- Two worst-case libraries and two best-case libraries per corner

```
create_library_set -name libs_min_std -timing [list $bestcase_lib_std]
create_library_set -name libs_max_std -timing [list $worstcase_lib_std]
create_library_set -name libs_min_fast -timing [list $bestcase_lib_fast]
create_library_set -name libs_max_fast -timing [list $worstcase_lib_fast]
```

```
create_rc_corner -name rc_worst -cap_table CMAX.capTbl
create_rc_corner -name rc_best -cap_table CMIN.capTbl
```

```
create_constraint_mode -name postCTS [list xxx.sdc]
```

```
create_delay_corner -name delay_corner_max
-late library_set libs_max_std \
-late_opcond_library stdcmos90T125 \
-late_opcond cmos90T125 \
-early library_set libs_max_fast \
-early_opcond_library fastcmos90T125 \
-early_opcond cmos90T125 \
-rc_corner rc_worst
```

```
create_delay_corner -name delay_corner_min
-late_library_set libs_min_std \
-late_opcond_library stdccmos90Tm40 \
-late_opcond cmos90Tm40 \
-early_library_set libs_min_fast \
-early_opcond_library fastcmos90Tm40 \
-early_opcond cmos90Tm40 \
-rc_corner rc_best
```

```
create_analysis_view -name postCts_max \
-delay_corner delay_corner_max \
-constraint_mode postCTS
create_analysis_view -name postCts_min \
-delay_corner delay_corner_min \
-constraint_mode postCTS

set_analysis_view -setup postCts_max -hold postCts_min
```

## Optimizing Timing in OCV Mode Using the Default Delay Calculator

After you specify the MMMC environment, use the following commands to set OCV on and enable Clock Path Pessimism Removal (CPPR), which is highly recommended:

```
setAnalysisMode -analysisType onChipVariation -cppr both
optDesign -postRoute [-hold]
```

## Using Conformal Constraint Designer During Timing Optimization

The Innovus software is tightly linked to the Conformal Constraint Designer (CCD) software. One of the features CCD provides is the ability to analyze critical false paths based on Innovus-CTE timing information and constraints. CCD outputs a file that lists a set of false paths, and the file can be loaded back into the Innovus software. Identifying the false paths in this way eliminates unnecessary netlist optimizations and improves design area, power, and timing.

You can use CCD to improve timing optimization in one of the following ways:

- Post-Processing Approach
- Integrated Approach

## Post-Processing Approach

Even after optimizing the design, you might not achieve the frequency target. Usually, this problem is due to tight timing constraints that are difficult to meet. By using CCD to verify whether the worst slack paths are valid, you may find that your critical paths are actually invalid and, therefore, need not be taken in account.

In this approach, the CCD tool is used only for post-processing and the Innovus timing closure flow is unchanged.

The following example shows the commands in the post-processing approach. Use these commands after running `timeDesign` or `optDesign` in setup mode.

```
deriveFalsePathCCD -outputDir . -outputFile trv.sdc  
source trv.sdc
```

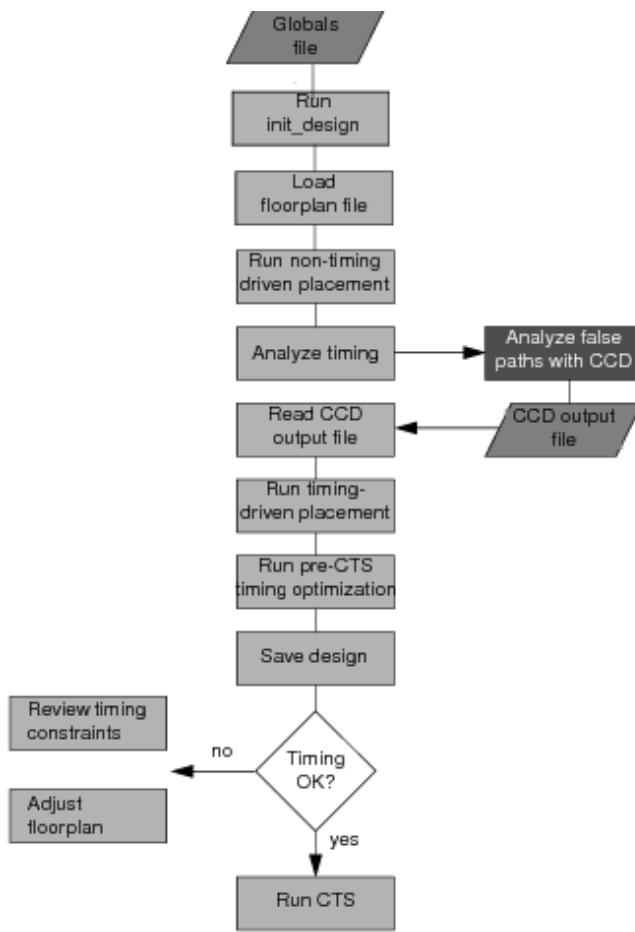
The `deriveFalsePathCCD` command runs CCD in batch mode and generates a list of critical false paths. Sourcing these new constraints applies any new critical false paths in the Innovus software.

After running these commands, re-analyze timing by running `timeDesign`. The slack should be better because false paths have been removed from consideration.

## Integrated Approach

A second approach is to integrate CCD with the Innovus timing optimization flow. Cadence recommends this method because it gives you the advantage of identifying false paths that are timing critical earlier in the implementation process. In addition, this method enables faster timing closure and leads to a better optimized netlist in area/leakage/power.

The following figure shows the recommended flow for this approach:



1. Load the floorplanned design and place standard cells in non-timing driven mode. Placing the cells in non-timing driven mode speeds up placement. You run placement again later, in timing-driven mode, in this flow. For example,

```

source myGlobalsFile.globals
init_design
loadFPlan myFloorPlan.fp
setPlaceMode -timingDriven false
placeDesign
  
```

2. Run `timeDesign` to build a clean timing graph. For example,

```
timeDesign -preCTS -outDir timing_place_noTD
```

3. Analyze and report setup timing violated paths using CCD. For example,

```
deriveFalsePathCCD -outputDir . -outputFile trv.postnotdp.sdc
```

This command runs CCD in batch mode. It outputs an SDC file that lists the false paths. CCD exits when the command completes. For more information, see [deriveFalsePathCCD](#) in the *Innovus Text Command Reference*.

4. Read the CCD output file into the Innovus software. For example,

```
source trv.postnotdp.sdc
```

5. Place the design in timing-driven mode and run preCTS timing optimization. Save the design. For example,

```
setPlaceMode -timingDriven true  
placeDesign  
timeDesign -preCTS -outDir timing_place_TD  
setOptMode -effort high  
optDesign -preCTS -outDir timing_optimized  
saveDesign post_opt_prects.enc
```

If the design meets timing, you can go on to clock tree synthesis; if not, look at the timing constraints and the floorplan to make sure they are reasonable. If CCD reports too many false paths, the CPU run time for timing analysis and tinge optimization might be affected.

-  If WNS and TNS do not change when reverting to initial timing constraints, it may be that the false paths identified by CCD could have met timing easily. This can happen because the false paths generated by CCD are based on timing post-placement timing. At this stage, no timing optimization has been performed and removing the false paths does not necessarily uncover violated paths.

## Optimizing Timing Using a Rule File

In a partitioned design, top-level and leaf partitions are generated. Before implementation, the leaf partitions' timing models are not completely accurate. Because accurate timing cannot be derived without accurate timing models for leaf partitions, rule-based optimization is a more suitable option than timing analysis-based optimization at this design stage. You can use a rule file for the top-level design by using the following command:

```
insertRepeater
```

## Optimizing Timing When the Constraint File Includes the set\_case\_analysis Constraint

If you include the `set_case_analysis` constraint in the timing constraint file, the Innovus software sets a constant value on specified signals before performing timing analysis. This constant value is then propagated through the path.

If you use the same timing constraint file for timing optimization, the software does not perform timing optimization on the constant nets because the delays are 0.

To run timing optimization on these nets, you must first specify the following command:

```
setAnalysisMode -caseAnalysis false
```

## Using the Footprintless Flow

By default, the Innovus software creates an internal footprint structure based on cell functionality. It is possible that cells with same functionality may be split across 2 or 3 different footprints, based on certain other characteristics, such as, drive strength. This methodology is referred to as the footprintless flow, and has the following advantages over a flow that relies on footprint information from the libraries:

- The libraries do not need to contain footprints, and you do not need to specify a footprint file.
- The following commands are not necessary because the software detects the functionality for inverters and buffers and decides whether a buffer is a delay cell, based on the cell's timing characteristic. The commands have no effect if specified in this flow.
  - `setBufFootPrint`
  - `setInvFootprint`

- `setDelayFootPrint`
- The software considers cells with the same functionality but different function syntax as equivalent and allows sizing between such cells.
- If the cell functionality is not defined, the software considers cells with the same `user_function_class` as equivalent and allows sizing between such cells. For example:

```
cell (AND2X1) {  
    user_function_class : my_and2_class;  
}  
  
cell (AND2X2) {  
    user_function_class : my_and2_class;  
}
```

Since both cells have the same `user_function_class` value, "my\_and2\_class", they will be treated as functionally equal.

**Note:** Ensure that the following statement is included in the library file, otherwise the `user_class_function` is not read:

```
define(user_function_class,cell,string);
```

- The software prints the list of usable and unusable ("don't use") buffers, inverters, and delay cells to the log file after reading in the libraries, for example:

```
Total number of combinational cells: 620  
Total number of sequential cells: 247  
Total number of tristate cells: 42  
Total number of level shifter cells: 0  
Total number of power gating cells: 0 Total number of isolation cells: 0  
List of usable buffers: BFX1 BFX2 BFX3 BFX4  
Total number of usable buffers: 4  
List of unusable buffers: BFX20 BFX32  
Total number of unusable buffers: 2  
List of usable inverters: IVX1 IVX2 IVX3 IVX4  
Total number of usable inverters: 4  
List of unusable inverters:
```

```
Total number of unusable inverters: 0
List of identified usable delay cells: DLY2 DLY4 DLY8
Total number of identified usable delay cells: 3
List of identified unusable delay cells:
Total number of identified unusable delay cells: 0
```

To revert to the behavior in previous releases (that is, to rely on footprint information in the libraries), use the `loadFootPrint` command. As in those releases, you must specify buffers, inverters, and delay cell footprints according to what was loaded in the footprint file. For more information, see "Using Cell Footprints".

To exclude cells from timing optimization, for example, if the libraries have clock buffers or clock inverters that should be used during CTS but not during timing optimization, set the "don't use" attribute in the timing constraints file, library, or command shell. Timing optimization can resize a "don't use" cell, but does not insert it.

**Note:** This is the default and recommended methodology since all of it is automated.

For more information see the `setDontUse` command.

## Using Cell Footprints

Timing optimization can use information in a footprint file. For example, the buffering mechanisms in `optDesign` add cells only if they are defined in the buffer footprint file.

To disable the footprintless flow (the default timing optimization flow) and load a footprint file, specify the following command:

```
loadFootPrint -infile footprint_file_name
```

Define footprints in your library or a footprint file by using the following commands, which are enabled when you specify `loadFootPrint`:

- `setBufFootPrint`
- `setInvFootPrint`
- `setDelayFootPrint`

**Note:** This is not the recommended methodology and should only be used as a workaround.

## Viewing Added Buffers, Instances, and Nets

After running timing optimization, use the Design Browser to view the added buffers, instances, and nets. The names of the buffers, instances, and nets added as a result of timing optimization are annotated with the prefix `FE_`.

For information on using the Design Browser, see the Design Browser section in the "Tools Menu" chapter of the *Innovus Menu Reference*.

## Default Naming Conventions

| Prefix     | Description                                                                        | Command                               |
|------------|------------------------------------------------------------------------------------|---------------------------------------|
| FE_MDBC    | Instance added by multi-driver net buffering                                       | <code>optDesign</code>                |
| FE_MDBN    | Net added by multi-driver net buffering                                            | <code>optDesign</code>                |
| FE_OCP_RBC | Instance added by rebuffering                                                      | <code>optDesign</code>                |
| FE_OCP_RBN | Net added by rebuffering                                                           | <code>optDesign</code>                |
| FE_OCPC    | Instance added by critical path optimization during preRoute optimization          | <code>optDesign</code>                |
| FE_OCPN    | Net added by critical path optimization during preRoute optimization               | <code>optDesign</code>                |
| FE_OFC     | Buffer instance added by insertRepeater or DRV fixing during preRoute optimization | <code>insertRepeater/optDesign</code> |
| FE_OFN     | Buffer net added by insertRepeater or DRV fixing during preRoute optimization      | <code>insertRepeater/optDesign</code> |
| FE_PHC     | Instance added by hold time repair                                                 | <code>optDesign</code>                |
| FE_PHN     | Net added by holdswd time repair                                                   | <code>optDesign</code>                |
| FE_PSBC    | Instance added by buffer insertion during postRoute optimization                   | <code>optDesign</code>                |

|         |                                                             |           |
|---------|-------------------------------------------------------------|-----------|
| FE_PSBN | Net added by buffer insertion during postRoute optimization | optDesign |
| FE_PSRC | Instance added by postRoute restructuring                   | optDesign |
| FE_PSRN | Net added by postRoute restructuring                        | optDesign |
| FE_PSC  | Instance added by postRoute setup repair                    | optDesign |
| FE_PSN  | Net added by postRoute setup repair                         | optDesign |
| FE_PDC  | Instance added by postRoute DRV fixing                      | optDesign |
| FE_PDN  | Net added by postRoute DRV fixing                           | optDesign |
| FE_RC   | Instance created by netlist restructuring                   | optDesign |
| FE_RN   | Net created by netlist restructuring                        | optDesign |
| FE_USC  | Instance added during useful skew optimization              | optDesign |

## Using Signoff Timing Analysis to Optimize Timing

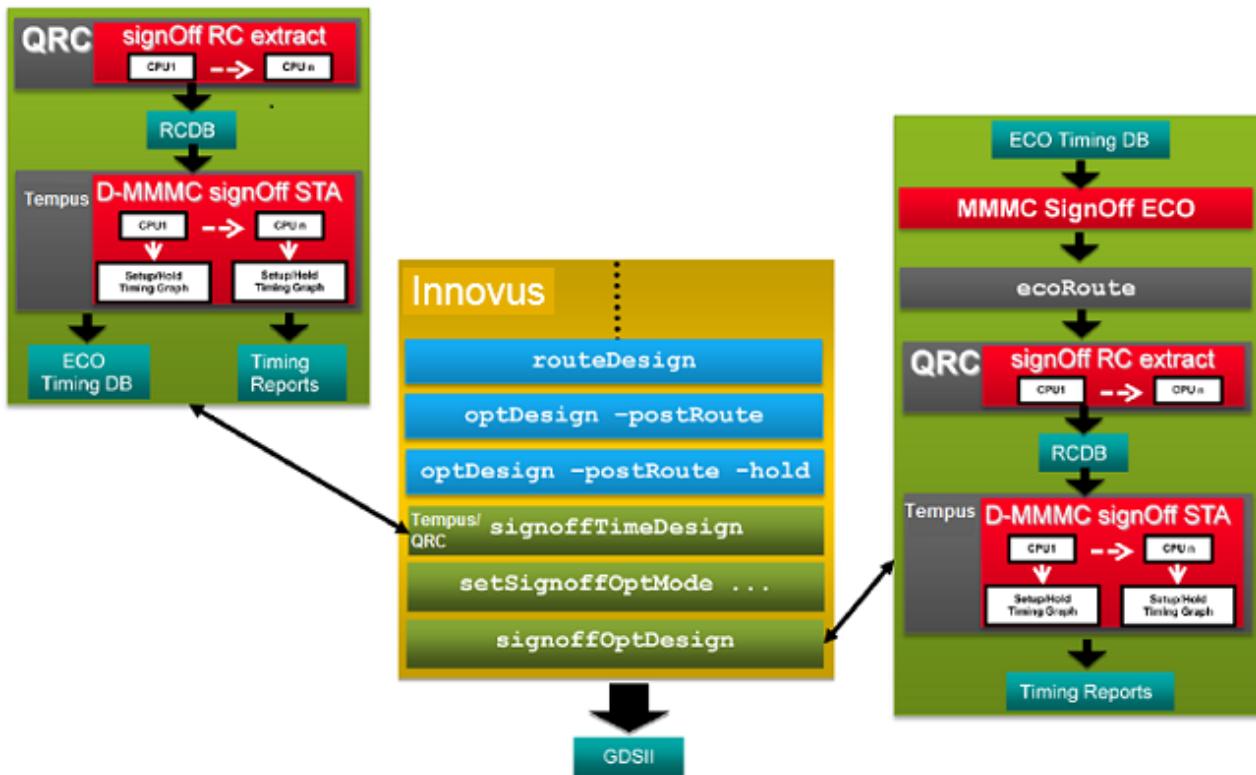
Signoff Timing Optimization feature allows you to run timing and leakage optimization within Innovus on Signoff parasitic from Quantus QRC and Signoff timing from Tempus. This feature gives a complete automated solution for using the entire signoff tools through one high level super command.

In a good timing closure methodology, at the implementation stage the timing targets that are set by the signoff Static Timing Analysis (STA) tool should be met. In order to ensure that the design state is close to sign-off quality, the timing reported by the implementation tool must correlate as much as possible to the signoff STA tool. To do this, signoff timing optimization provides the following features:

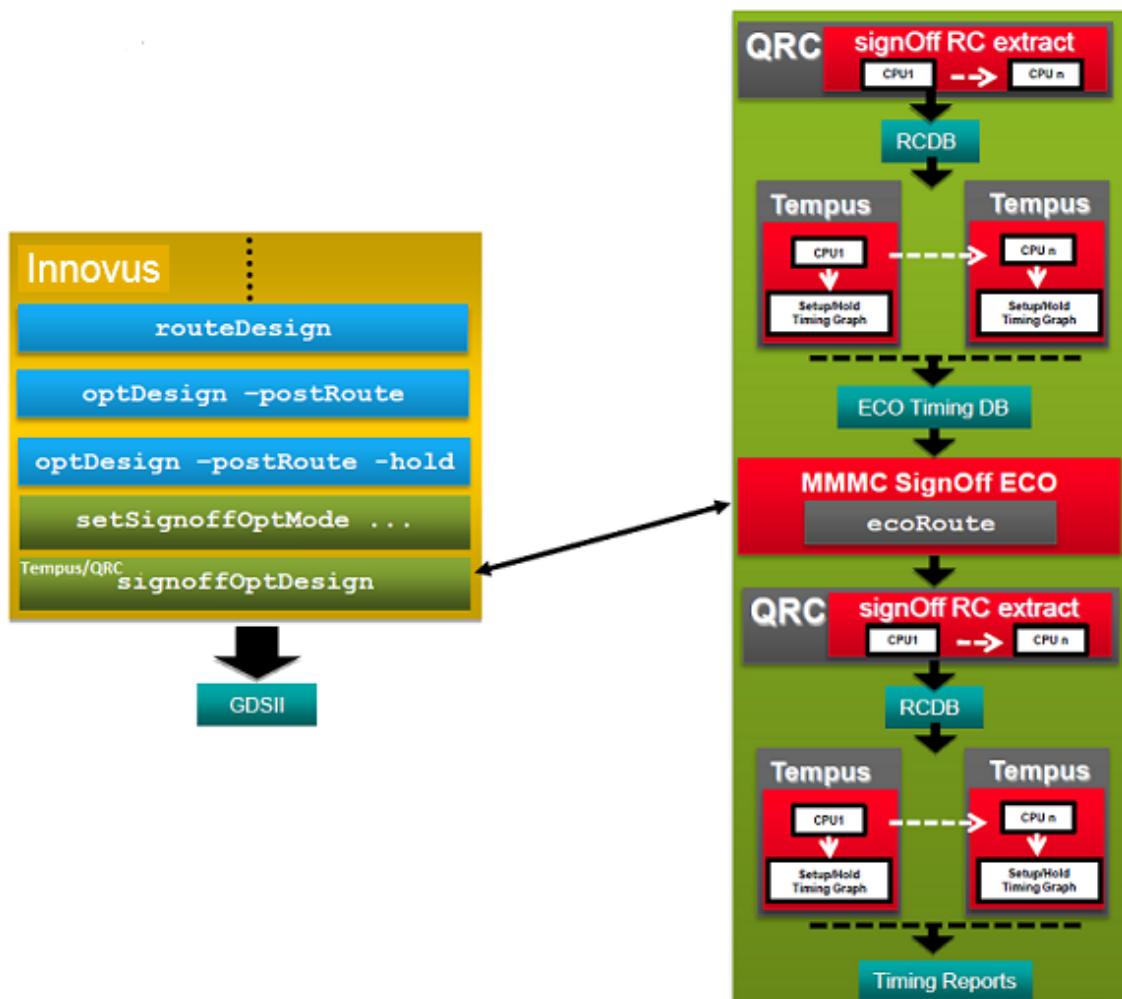
- Allows you to run timing and leakage optimization on signOff timing within Innovus.
- Maximizes the usage of Cadence tools - Innovus will enable you to run extraction (Quintus QRC) and Tempus without extra effort.

## Running MMMC SignOff ECO within Innovus

To provide signoff timing report in Innovus using Tempus, you can use the `signoffTimeDesign` command. This command uses Quantus QRC and Tempus in standalone mode to perform signoff STA using the DMMMC infrastructure and save an ECO Timing DB per view. This signoff timing can be optimized using `signoffOptDesign`. Similarly, `signoffOptDesign` can be used to perform leakage optimization on this signoff timing. The following diagram illustrates the flow and architecture of each signoff command:



In case the ECO Timing DB are not available when `signoffOptDesign` command is run, then the super command will automatically run `signoffTimeDesign` under the hood, as shown in the figure below:



Signoff Timing Optimization provides a flexible flow to accommodate any specific methodology:

- If parasitics should be extracted using tQuantus or IQRC, this can be done manually by the user. Then `signoffTimeDesign` can be used with the `-reportOnly` option to reuse those parasitics and skip the Quantus QRC call.
- In case specific steps/options should be performed before/during ECO routing, this step can be performed manually after running `signoffOptDesign` with the `-noEcoRoute` option.
- When specific signoff STA commands/options/globals should be applied, you can set these in a file and pass it to Tempus through `setSignoffOptMode -preStaTcl file` option.

## Signoff Timing Analysis in Innovus using Timing Debug

Signoff timing analysis is performed by Tempus that saves signoff timing report (in .mtarpt) files per view. These files can be loaded in Innovus through the global timing debug (GTD) interface in order to perform fine grain timing analysis.

## Enabling PBA Timing Analysis and Optimization

You can use the `setSignoffOptMode` command to enable path-based (PBA) timing analysis and optimization.

## Path Group Support

MS-ECO supports the following ways for path group-based fixing :

1. Endpoint-based inclusion/exclusion per view
2. Endpoint-specific slack adjustment per view (both positive and negative adjustments)
3. Option to fix only register to register paths

Path group support for MS-ECO can be used for both Hold and Setup fixing.

**Note:** This feature is currently not supported for DRV or leakage optimization.

### 1. Endpoint-based inclusion/exclusion per view

You can specify endpoints that need to be included or excluded during optimization for a view with the following `setSignoffOptMode` parameters:

- -selectHoldEndpoints
- -selectSetupEndpoints

#### File format:

```
<View> include/exclude <Endpoints>
```

`<View>` can be specified as either `viewName` or `v*` (if the endpoints are to be included or excluded for all views)

`<Endpoints>` can be specified as a list of endpoint names separated by a space. For example, `E1 E2 E3` slack range in nanosecond: `<minSlack> <maxSlack>` where all endpoints with a slack greater than or equal to `minSlack` and slack less than or equal to `maxSlack` will be included or excluded. For example, `-2.0 -1.0 E*`, if all endpoints for the view need to be included or excluded.

## Sample configuration files

The lines starting with # are comments and will be skipped.

**File:** hold\_exclude\_example

```
# Exclude endpoints EXECUTE_INST/pc_acc_reg/SE and EXECUTE_INST/read_prog_reg/SE from view
core+typ-rcMin for hold fixing
core+typ-rcMin exclude EXECUTE_INST/pc_acc_reg/SE EXECUTE_INST/read_prog_reg/SE
```

**File:** hold\_include\_example\_with\_range

```
# Include only the endpoints that have slacks >= -0.5 ns and <= 0.04 ns for view core+best-
rcTyp for hold fixing
core+best-rcTyp include -0.5 0.04
```

## 2. Endpoint-specific slack adjustment per view

You can specify slack adjustment (margin) for selected endpoints during optimization for a view with the following `setSignoffOptMode` parameters:

- -specifyHoldEndpointsMargin
- -specifySetupEndpointsMargin

### File format:

```
<View> <Margin> <Endpoints>
```

`<View>` can be specified as either `viewName` or `v*` (if the endpoints are to be included or excluded for all views)

`<Margin>` is specified as float value in nanosecond. Margin value is subtracted from the endpoint slack so a positive margin means that the endpoint slack would be degraded by the margin amount.

`<Endpoints>` can be specified as a list of endpoint names separated by a space. For example, `E1 E2 E3` slack range in nanosecond: `<minSlack> <maxSlack>` where all endpoints with a slack greater than or equal to `minSlack` and slack less than or equal to `maxSlack` would be included or excluded. For example, `-2.0 -1.0 E*` if all endpoints for the view need to be included or excluded.

### Sample configuration files

The lines starting with # are comments and would be skipped.

**File :** hold\_margin\_allViews\_slackRange\_example

```
# Apply margin of 0.1 nanosecond to all endpoints that have slacks between -0.07 to -0.03
for all hold views
v* 0.1 -0.07 -0.03
```

**File :** hold\_margin\_allEndPoints\_example

```
# Apply margin of 0.2 nanosecond to all endpoints for hold view core+typ-rcMin
core+typ-rcMin 0.2 E*
```

**File : hold\_margin\_selectedEndPoints\_example**

```
# Apply margin of 0.6 ns to endpoints TDSP_CORE_MACH_INST/phi_6_reg/SE and
TDSP_CORE_MACH_INST/phi_1_reg/SE for hold view core+best-rcTyp
core+typ-rcMin 0.6 TDSP_CORE_MACH_INST/phi_6_reg/SE TDSP_CORE_MACH_INST/phi_1_reg/SE
```

### 3. Option to fix only register-to-register paths

You can choose to fix only register-to-register paths during Hold/Setup fixing with the parameter specified below. Other path group configurations will be ignored in this mode. The timing for non-register-to-register paths for fixing mode (Hold or Setup) will be adjusted to 0 but the timing for other mode (Setup during Hold fixing/Hold during Setup fixing) is not affected.

```
setSignoffOptMode -optimizeCoreOnly {true | false}
```

## Sample Template Scripts

- The following example shows graph-based analysis (GBA) signoff STA followed by Setup and Hold optimization:

```
source postRoute.enc
setMultiCpuUsage -localCpu 4 -remoteHost 3 -cpuPerRemoteHost 4
setSignoffOptMode -preStaTcl preStaTcl.tcl
signoffTimeDesign
signoffOptDesign -setup -noEcoRoute
signoffOptDesign -hold
```

- The following example shows GBA signoff STA followed by Setup optimization that is based on tQuantus:

```
source postRoute.enc
setMultiCpuUsage -localCpu 4 -remoteHost 3 -cpuPerRemoteHost 4
setExtractRCMode -coupled true -engine postRoute -effortLevel medium
extractRC
setSignoffOptMode -preStaTcl preStaTcl.tcl
signoffTimeDesign -reportOnly
signoffOptDesign -setup -noEcoRoute
ecoRoute
extractRC
signoffTimeDesign -reportOnly -noEcoDb
```

- The following example shows signoff STA followed by Hold optimization in PBA Setup and Hold mode:

```
source postRoute.enc
setMultiCpuUsage -localCpu 4 -remoteHost 3 -cpuPerRemoteHost 4
setSignoffOptMode -preStaTcl preStaTcl.tcl
setSignoffOptMode -retime aocv_path_slew_propagation -checkType both
signoffTimeDesign
signoffOptDesign -hold
```

**Note:** The preStaTcl.tcl script allows you to apply any signoff STA related settings or reset any of the set\*mode commands.

# Using the NanoRoute Router

- About NanoRoute Routing Technology
- Routing Phases
  - Global Routing
  - Detailed Routing
- NanoRoute Router in the Innovus Flow
- Before You Begin
  - Checking Your LEF Files
  - Checking for Problems with Cells, Pins, and Vias
  - Generating Tracks
  - Specifying Routing Layers
- Interrupting Routing
- Using the routeDesign Supercommand
- Results
- Use Models
  - Running the NanoRoute Router with Innovus Menu Commands and Forms
  - Running the NanoRoute Router with Innovus Text Commands
  - Running the NanoRoute Router in Standalone Mode
- Using NanoRoute Parameters
  - Using Attributes and Options Together
- Accelerating Routing with Multi-Threading and Superthreading
  - When to Accelerate Routing
  - Superthreading Log File Excerpts
- Following a Basic Routing Strategy
  - Using the InnovusText Commands
  - Using the Innovus GUI
- Checking Congestion
  - Using the Congestion Analysis Table
  - Using the Congestion Map
- Resolving Open Nets
  - Log File Examples
  - Diagnosing Problems Using verifyTracks
  - Resolving Additional Open Net Problems

- [Running Timing-Driven Routing](#)
  - [Input Files](#)
  - [Using the CTE and the NanoRoute Router in Native Mode](#)
  - [Using the CTE and Standalone NanoRoute](#)
- [Routing Clocks](#)
  - [Setting Attributes for Clock Nets](#)
  - [Routing Clock Nets Using the GUI Forms](#)
  - [Running Postroute Optimization](#)
- [Preventing and Repairing Crosstalk Problems](#)
  - [Crosstalk Prevention Options](#)
- [Running ECO Routing](#)
  - [ECO Limitations](#)
  - [ECO Flow](#)
- [Evaluating Violations](#)
  - [DRC Marker Name Comparison Table](#)
  - [Violations on Upper Metal Layers](#)
  - [Violations in Timing-Driven Routing](#)
  - [Deleting Violated Nets](#)
  - [Using Additional Strategies to Repair Violations](#)
- [Concurrent Routing and Multi-Cut Via Insertion](#)
- [Postroute Via Optimization](#)
- [Optimizing Vias in Selected Nets](#)
- [Via Optimization Options](#)
- [Performing Shielded Routing](#)
  - [Shielding Option](#)
  - [Performing Shielded Routing Using the GUI](#)
  - [Performing Shielded Routing Using Text Commands](#)
  - [Interpreting the Shielding Report](#)
- [Routing Wide Wires](#)
  - [Using Non-Default Rules](#)
- [Repairing Process Antenna Violations](#)
  - [Repairing Violations on Multiple-Pin Nets](#)
  - [Changing Layers](#)
  - [Using Diodes](#)

- Deleting and Rerouting Nets with Violations
- Repairing Violations on Cut Layers
- Process Antenna Options
- Examples
- Creating RC Model Data in tQuantus Model File
  - Use model for tQuantus Model File
  - Example1: Use Model for Track Assignment-Based Timing and SI Optimization
  - Example2: Use Model for optDesign -postRoute and timeDesign –postRoute Optimization
- Using a Design Flow that Includes Astro or Apollo
- Troubleshooting

## About NanoRoute Routing Technology

The NanoRoute<sup>®</sup> router performs concurrent signal integrity, timing-driven, and manufacturing aware routing (SMART routing) of cell, block, or mixed cell and block level designs. The router is optimized for routing designs with the following features:

- More than 300K instances or nets and at least five routing layers
- 180 nanometer or smaller process technology
- Signal integrity critical
- Timing critical
- Detailed-model (full-model) abstracts

**Note:** The WRoute router is also included in the Innovus Implementation System (Innovus) software. Your routing results might be better with the WRoute router when the technology is 180 nm or larger, and you have fewer than five routing layers and 300K instances.

## Routing Phases

Full routing consists of global and detailed routing. You can repeat detailed routing incrementally on a routed database. Incremental detailed routing is not the same as ECO routing. For information, see [Global Routing and Detailed Routing](#).

ECO routing consists of incremental global and detailed routing passes on a routed design. During ECO routing, the router completes partial routes and makes minimal changes to existing wire segments. For information, see [Running ECO Routing](#).

## Global Routing

During this phase, the router breaks the routing portion of the design into rectangles called global routing cells (gcells) and assigns the signal nets to the gcells. The global router attempts to find the shortest path through the gcells, but does not make actual connections or assign nets to specific tracks within the gcells. It tries to avoid assigning more nets to a gcell than the tracks can accommodate. The detailed router uses the global routing paths as a routing plan.

The router can generate a map of the gcells, called a congestion map, that you can examine to see the approximate number of nets assigned to the gcells. The congestion map uses colors to indicate whether there are too few, too many, or the correct number of nets assigned to the gcells. If the router assigns too many nets to a gcell, it marks the gcell as over-congested. You can also read the Congestion Analysis Table in the Innovus log file to learn the distribution and severity of the congestion after global routing.

## Related Topics

- For more information on gcells, see "[GCell Grid](#)" in the "DEF Syntax" chapter of the *LEF/DEF Language Reference*.
- For more information on the congestion map and table, see [Checking Congestion](#).

## Detailed Routing

During this phase, the NanoRoute router follows the global routing plan and lays down actual wires that connect the pins to their corresponding nets. The detailed router creates shorts or spacing violations rather than leave unconnected nets. You can run detailed routing on the entire design, a specified area of the design, or on selected nets. In addition, you can run incremental detailed routing on a database that has already been detail routed. The router runs search-and-repair routing during detailed routing. During search and repair, it locates shorts and spacing violations and reroutes the affected areas to eliminate as many of the violations as possible. The primary goal of detailed routing is to complete all of the required interconnect without leaving shorts or spacing violations.

During detailed routing, the router divides the chip into areas called switch boxes (SBoxes), which align with the gcell boundaries. The SBoxes can be expressed in terms of gcells; for example, a 5x5 SBox is an SBox that encompasses 25 gcells. The SBoxes overlap with each other and their size and amount of overlap might vary during search-and-repair iterations. The router also runs postroute optimization as part of detailed routing. During postroute optimization, it runs more rigorous search and repair steps. Detailed routing stops automatically if it cannot make further progress on routing the design. The routed data is saved as part of the Innovus database.

## NanoRoute Router in the Innovus Flow

The NanoRoute router is part of the block implementation and the top-level implementation stages of the Innovus flow. Run the router early in the design flow to identify and fix routability problems or avoid them altogether. You can run the router in non-timing-driven mode after the default parasitic extraction step to establish a baseline for future steps. If the design is congested or unrouteable, stop and resolve problems before continuing.

## Before You Begin

The NanoRoute router reads designs directly from Innovus. Before running the router, ensure your design meeting the following conditions:

- It is fully placed and the placement is legal, without any overlaps. Use the `checkPlace` command to check for overlaps.
- (Optional) Run the `verifyGeometry` command and fix any problems. In general, it is easier to fix geometry problems before routing than after routing.
- Power is routed. Use the `sroute` command to route power structures.

## Checking Your LEF Files

You can avoid violations and save time if you ensure your LEF files are optimized for routing. Check the following statements and edit the files with a text editor if necessary:

- **MINSIZE**

The router does not support specifying `MINSIZE` without specifying `AREA`. `MINSIZE` allows a geometry that is smaller than `AREA`.

- **UNITS**

The router does not support a value of 100 for `DATABASE MICRONS` in the `UNITS` statement. If the LEF technology file specifies `DATABASE MICRONS 100`, run the following command before importing the design:

```
setImportMode -minDBUPerMicron 1000
```

- **MANUFACTURINGGRID**

The router requires that you define the manufacturing grid.

- MACRO

To improve pin access, ensure that all standard cell macros are defined as `CLASS CORE`.

You must use real shapes, not block-style abstracts, for the shapes on the layers where you expect the router to connect to pins of standard cell macros.

- VIA

The `TOPOFSTACKONLY` keyword is unnecessary if there are `LEF LAYER AREA` statements, because the router automatically derives `TOPOFSTACKONLY` vias based on the `AREA` statements. If a default via satisfies the `AREA` statement, the router tags it internally as a `TOPOFSTACKONLY` via.

If there is no `AREA` statement for a routing layer, the router looks for `TOPOFSTACKONLY` vias that go up to the next metal layer. If `TOPOFSTACKONLY` vias exist, it derives the `AREA` rule from those vias--the smallest area of the bottom layer metal of all such vias becomes the `AREA` rule. This feature provides backward compatibility with LEF files that do not have `AREA` rule support.

## Related Topics

- [Unsupported LEF and DEF Syntax](#)
- [LEF Syntax chapter of the \*LEF/DEF Language Reference\*.](#)

## Checking for Problems with Cells, Pins, and Vias

- Make sure that all power and ground pins in the `SPECIALNETS` section of the DEF file are marked +  
`USE POWER` or + `USE GROUND`.
- Overlapping cells  
Overlapping cells make pins short each other and create violations on `meta1`. Check for overlaps by using one of the following commands:
  - `verifyGeometry`
  - `checkPlace`
- Pins underneath power routes  
Pins that are underneath power routes are inaccessible and cause violations on `meta1` and `meta2`. Check for pins underneath power routes by using the *Auto Query* feature.

For more information, see Auto Query in the [The Main Window](#) chapter of the *Innovus Menu Reference*.

- Lack of rotated vias

Rotated vias help reduce design rule violations by making pins accessible. The router does not rotate vias automatically and creates violations on *metal1* when it cannot access the pins.

If there is no rotated via defined in the LEF, you can use [generateVias](#) to create them.

Define rotated vias in the LEF file.

## Generating Tracks

In the Innovus environment, the router generates tracks automatically, based on the routing pitch, layer width and spacing, and minimum via widths.

If you import a DEF file, run the [generateTracks](#) command prior to global routing to correct faulty track definitions and tune the tracks to routing.

## Related Topics

- For more information, see [generateTracks](#) in the *Innovus Text Command Reference*.
- For information on importing DEF files, see [Import and Export Commands](#) in the *Innovus Text Command Reference*.

## Specifying Routing Layers

By default, the router uses all possible routing layers for routing wires. In some situations, you might want to limit routing to a layer range that does not include all routing layers. For example, you might want to reserve the top layers for power and ground stripes or perform ECO routing on a few layers only. You can specify hard limits for routing all nets within a layer range or you can specify soft limits to route specified nets within a layer range.

## Specifying Hard Layer Limits

When you specify hard layer limits, the router routes all nets within those limits. If there is a pin outside the limits you specify, the router uses vias, including stacked vias, to access the pin.

Use the following [setNanoRouteMode](#) parameters to specify hard layer limits:

- `-routeBottomRoutingLayer`

- -routeTopRoutingLayer

At times it might not be possible to route the nets within the limits without creating violations. For example, assume two pins, `pin_a` is on `meta/8` and `pin_b` is on `meta/7`. The pins overlap in the X and Y direction. If you specify that the top routing layer is `meta/6`, the router connects to `pin_a` by using stacked vias, creating a short with `pin_b`.

## Specifying Soft Layer Limits

When you specify soft layer limits, the router attempts to route specific nets within a layer range, but might route some nets outside the layer range if necessary to complete routing without creating violations. In addition, you can specify the effort level for staying within the range. You can also route specific nets within the layer range and others outside the layer range. For example, you can route critical nets within a narrower layer range than you route the rest of the nets in order to improve timing.

Use the following `setAttribute` parameters to specify soft layer limits and set the effort level toward honoring the limits:

- -bottom\_preferred\_routing\_layer
- -top\_preferred\_routing\_layer
- -preferred\_routing\_layer\_effort

## Interrupting Routing

To interrupt routing, press `Ctrl+C`. The `routeDesign` or `globalDetailRoute` command continues to run until the database is in a state where the command can stop safely. When the software stops, it presents the Interrupt menu. For information on the Interrupt menu, see "Interrupting the Software" in the [Getting Started](#) chapter.

**Warning:** When you interrupt routing with `Ctrl+C`, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.

## Using the `routeDesign` Supercommand

The recommended Cadence design flows use the `routeDesign` command to run global and detailed routing and to optimize vias and wirelength after routing.

`routeDesign` honors the `setNanoRouteMode` and `setAttribute` settings and has the following advantages over using the `globalRoute` and `detailRoute` or `globalDetailRoute` commands:

- It runs SMART routing by default; that is, it runs in both timing- and signal integrity-driven mode by default.  
The other routing commands are not timing- or signal-integrity driven by default, but you can use the following `setNanoRouteMode` parameters to turn on timing- and signal-integrity-driven routing for those commands:
  - `-routeWithTimingDriven true`
  - `-routeWithSiDriven true`
- It changes the status of clock nets from `FIXED` to `ROUTED` so it can modify them during routing and routes them before routing other nets. Once the status of the clock nets is set to `ROUTED`, it does not change it back to `FIXED`.
  - To keep clock nets' status `FIXED`, run the following command before running `routeDesign`:  
`setNanoRouteMode -routeDesignFixClockNets true`
  - To stop the router from routing clock nets first, run the following command before running `routeDesign`:  
`setNanoRouteMode -routeDesignRouteClockNetsFirst false`
- It runs a placement check prior to routing to ensure that the placement is clean.  
To turn off the placement check, specify the following `routeDesign` parameter:  
`-noPlacementCheck`
- It checks for conflicts in `setNanoRouteMode` settings and issues warning messages when it detects problems. In some cases, it resets a mode in order to continue processing. For example, trying to fix postroute lithography problems and optimize vias concurrently can cause conflicts. If `routeDesign` detects requests for both types of operation, it issues a warning, turns off via optimization, and proceeds with fixing lithography problems.
- It has parameters that simplify via and wire optimization after routing. In addition, some `setNanoRouteMode` parameters work with `routeDesign`, but not with other routing commands.
  - The `routeDesign` parameters for via and wire optimization are `-viaOpt` and `-wireOpt`.
  - The `setNanoRouteMode` parameters that work only with `routeDesign` are - `routeDesignFixClockNets` and `-routeDesignRouteClockNetsFirst`.

## Related Topics

For more information, see the following commands in the [Route Commands and Global Variables](#) chapter of the *Innovus Text Command Reference* :

- [detailRoute](#)
- [globalDetailRoute](#)
- [globalRoute](#)
- [routeDesign](#)
- [setAttribute](#)
- [setNanoRouteMode](#)

## Results

The NanoRoute router outputs can include the following (depending on the run-time options you set):

- Section in the Innovus log file
- Routed DEF file
- GDSII file
- SDF or SPEF file
  - For information on outputting GDSII and DEF files, see [Importing and Exporting Designs](#).
  - For information on outputting an SDF or SPEF file, see [Timing Analysis](#) .
- The following reports:
  - Routing statistics
    - For information, see the [reportRoute](#) command.
  - Routing connectivity
    - For information, see the [checkRoute](#) command.
  - Wire statistics, including wirelength
    - For information, see the [reportWire](#) command.
  - Shielding statistics
  - Timing analysis
    - For information, see [Timing Analysis](#) .
  - Capacitance
    - For information, see [RC Extraction](#).
  - Design rule checking (DRC) and layout versus schematic (LVS)

- Process antenna violations  
For information on DRC, LVS, and process antenna reports, see [Identifying and Viewing Violations](#).
- Signal integrity  
For information on signal integrity reports, see [Analyzing and Repairing Crosstalk](#).

**Note:** The number of nets that were not routed due to the existence of mixed signal constraints are reported in the log file by the NanoRoute Router.

## Use Models

### Running the NanoRoute Router with Innovus Menu Commands and Forms

Use the following forms to route the design.

- [Mode Setup - NanoRoute](#)  
Use this form to specify the run-time options and the global parameters for the NanoRoute router.
- [NanoRoute/Attributes](#)  
Use this form to specify attributes for nets.
- [NanoRoute](#)  
Use this form to set routing options.
- [Set Congestion Map Style - NanoRoute](#)  
Use this form to customize the congestion map.

### Running the NanoRoute Router with Innovus Text Commands

Use the following commands to set NanoRoute attributes and options, generate tracks, LEF files, and vias that are optimized for the router, route the design, and optimize vias and wirelength after routing. The text commands include some NanoRoute options that are not included on the forms.

- [generateTracks](#)  
Generates optimized tracks for the router (only necessary if you import a non-native DEF) file.
- [generateLef](#)  
Generates an optimized LEF file (only necessary if you import a non-native or non-current LEF file).
- [generateVias](#)

Generates vias that are optimized for the router (useful if you import an incomplete or non-current LEF file).

- `getAttribute/setAttribute`  
Display and set net attributes.
- `getNanoRouteMode /setNanoRouteMode`  
Display and set run-time options for the router.
- `globalRoute , detailRoute , ecoRoute , globalDetailRoute , routeDesign`  
Route the design.

The recommended design flows use `routeDesign`. For more information, see Using the `routeDesign` Supercommand.

## Running the NanoRoute Router in Standalone Mode

The commands and options for running the NanoRoute router in standalone mode are not described in this document. The standalone NanoRoute router has its own GUI and command syntax. The commands, options, and attributes used by the standalone router are described in the *NanoRoute Technology Reference*.

## Using NanoRoute Parameters

The NanoRoute router has two kinds of parameters: attributes and options.

- Attributes assign characteristics to nets.  
For example, use attributes to specify nets that have the following attributes: they are routed first (or last), they are routed on certain layers, they are protected by extra spacing, they are shielded (or act as shields), and signal integrity violations that affect them are repaired after routing.
- Options determine run-time operations.  
For example, use options to perform the following run-time operations: run global or detailed routing, route selected nets only, repair antenna or design-rule violations, run timing driven or signal integrity driven routing, and specify the number of processors to use.

The following table lists attribute and option characteristics:

| Characteristic | Attributes                    | Options                                |
|----------------|-------------------------------|----------------------------------------|
| Application    | Apply locally to a net object | Apply globally to a process or command |

|                                   |                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Specification                     | <ul style="list-style-type: none"> <li>• NanoRoute/Attributes form</li> <li>• <code>setAttribute</code> command</li> <li>• Some attributes can only be specified by <code>setAttribute</code>.</li> </ul>                                                                                     | <ul style="list-style-type: none"> <li>• NanoRoute form</li> <li>• <code>setNanoRouteMode</code> command</li> <li>• Some options can only be specified by <code>setNanoRouteMode</code>.</li> </ul>                                                                                                                                                                                                                                       |
| Persistence                       | <p>Saved with the database.</p> <p>When you set an attribute and save the database and exit, the attribute setting is saved. If you reload the database, the object retains the attribute setting.</p>                                                                                        | <p>Saved with the database.</p> <p>When you set an option, save the database and exit, the option setting is saved. If you reload the database, the router retains the option value.</p>                                                                                                                                                                                                                                                  |
| Format                            | <ul style="list-style-type: none"> <li>- <code>attribute_name</code></li> <li>● Example:<br/>-<code>avoid_detour</code></li> <li>● Case sensitive (all lower case)</li> <li>● Mandatory underscores separate words</li> <li>● Native and standalone NanoRoute formats are the same</li> </ul> | <ul style="list-style-type: none"> <li>-<code>optionName</code></li> <li>● Example:<br/>-<code>drouteAutoStop</code></li> <li>● Case sensitive (mixed case).<br/>Each word starts with an uppercase letter</li> <li>● No underscores.</li> <li>● Native and standalone NanoRoute formats are different (standalone options are all lowercase; words are separated by underscores; options do not start with a leading hyphen).</li> </ul> |
| See settings for this run ...     | Use the <code>getAttribute</code> command                                                                                                                                                                                                                                                     | Use the <code>getNanoRouteMode</code> command                                                                                                                                                                                                                                                                                                                                                                                             |
| More information available at ... | <code>setAttribute</code> in the <i>Innovus Text Command Reference</i>                                                                                                                                                                                                                        | <code>setNanoRouteMode</code> in the <i>Innovus Text Command Reference</i>                                                                                                                                                                                                                                                                                                                                                                |

## Using Attributes and Options Together

You can use attributes and options together. For example, to run global and detailed routing and repair signal integrity violations on a specified net during postroute optimization, set the `-si_post_route_fix` attribute for the net and route the design with the `-routeWithSiPostRouteFix` option set to `true`.

Using text commands, issue the following commands:

```
setAttribute -net net1 -si_post_route_fix true
setNanoRouteMode -routeWithSiPostRouteFix true
globalDetailRoute
```

Using the GUI, make the following selections:

- On NanoRoute/Attributes form,
  - a. Type the name of the net in the *NetName(s)* text box.
  - b. Select *SI Post Route Fix True* .
- On the NanoRoute form,
  - a. Select both *Global Route* and *Detail Route* .
  - b. Select *Post Route SI* .

## Accelerating Routing with Multi-Threading and Superthreading

Innovus products accelerate routing by using more than one processor in the same machine and by distributing routing to multiple machines.

Both signal routers, the NanoRoute router and the WRoute router, can use more than one processor in the same machine. This is called multi-threading. For more information, see "Running Multi-Threading" in Accelerating the Design Process By Using Multiple-CPU Processing.

The NanoRoute detail router accelerates routing even more by distributing detailed routing across the network to multiple machines. This capability combines multi-threading with distributed processing, and is called Superthreading. When used with a gigabit Ethernet connection, Superthreading makes data communication time negligible.

Superthreading has the following features:

- Uses available Innovus licenses. No special licenses are necessary.
- Platform independent.
  - Different operating systems--including Solaris, Linux, and HP-UX--can be used in the same job.
  - Different hardware--including Sun, IBM, and HP--can be used in the same job.

- 64-bit and 32-bit versions of the NanoRoute router can be used in the same job. For example, you can start a large job on a 64-bit server and run the job on smaller 32-bit clients.
- Can run using the `rsh` command, and with LSF, Sun Grid, or SSH configurations.
  - The RSH and SSH method tie multi-threaded jobs together.
  - The LSF and Sun Grid methods tie single jobs together.

## Related Topics

- [Accelerating the Design Process By Using Multiple-CPU Processing](#) chapter in the *Innovus User Guide*
- [Multiple-CPU Processing Commands](#) chapter in the *Innovus Text Command Reference*
- Multiple CPU Processing form in the [Options Menu](#) chapter of the *Innovus Menu Reference*

## When to Accelerate Routing

Not all designs or network topologies can take advantage of accelerated routing. Consider the following issues, and use single threading, multi-threading, or Superthreading when appropriate:

- Small (10k), simple designs or designs that do not have a lot of violations  
Small jobs or designs that are easily routed probably do not need multiple CPUs or machines.
- Slow networks  
The speed (10 Mb, 100 Mb, or 1,000 Mb) and type (LAN or WAN) of the network affect Superthreading speed.
- Loaded networks  
Sharing CPU cycles with other processes increases Superthreading run time.
- Full or pending LSF queues or queues configured for one job  
A queue that is set up to run only one job decreases efficiency.

## Usage Notes

- If you use the `rsh` command for Superthreading, you must be able to run the remote shell from the server machine to the client machines without a password prompt.
- The NanoRoute software must be accessible to the server and client machines.
- Client machines must be able to access the same version of the NanoRoute software.
- If you run the router in native mode, it will be the server program and the standalone router will be the client program.
- Start your routing job on the fastest multi-threaded machine available.
- Include the host machine as a client, otherwise it will be a server only and will not perform any routing jobs.
- If any CPUs crash, your job will not complete.

If there is a crash, most likely it will happen during the client routing stage, and the server will continue to run. The database on the server will be maintained in the state it had prior to the crash.

Check the messages in the log file to determine the problem and zoom into the area of the crash to see a graphical representation of the cause of the failure. After you fix the problem, you can continue routing from the crash point.

- If your job includes both Sun and Linux clients, include a different path to each executable in the command script.
- You can run a job that uses both a Sun queue and a Linux queue.

## Superthreading Log File Excerpts

The following excerpts from a log file show progress during Superthreading. The software uses the following definitions to calculate the time:

- `client CPU time` is the CPU time on clients only.
- `cpu time` is the server CPU time plus the `client CPU time`
- `elapsed time` is the complete run time (the total elapsed time).

The first file fragment shows that the job is running with RSH, with two threads on the same host. The NanoRoute router pauses as the data on the server is synchronized.

#server my\_machine is up on port 123456 waiting for connection .....

```
# client 2thread 1 from host machine_1
# client 2thread 2 from host host_machine_1
# Sync client 2 data ...
```

```
# cpu time = 00:00:03, elapsed time = 00:04:18, memory = 561.87 (Mb)
```

The second fragment shows that only 86 percent of the client CPU time is being used. Another process (in addition to the route job) is using CPU resources.

```
# client 3thread 1 from host machine_2
# client 3thread 2 from host machine_2
# Sync client 3 data ...
# cpu time = 00:00:03, elapsed time = 00:04:31, memory = 561.87 (Mb)
#
# Start Detail Routing.
# Start initial detail routing ...
# completing 10% with 0 violations
...
# completing 90% with 14 violations
# elapsed time = 00:12:29, memory = 606.02 (Mb)
# completing 100% with 10 violations
# elapsed time = 00:12:53, memory = 567.24 (Mb)
# number of violations = 0
# client cpu time = 00:03:12, memory 562.70 (Mb), util = 86%
#cpu time = 00:01:21, elapsed time = 00:123:54, memory = 566.24 (Mb)
. . .
```

The third fragment shows that the job took less elapsed time than cpu time. The elapsed time is less than the cpu time because two clients are being used to route one job.

```
#Total number of violations on LAYER M8 = 4
#Total number of violations on LAYER M9 = 1
#Total number of violations on LAYER M10 = 0
#Client cpu time = 17:38:54
#Client peak memory = 795.22 (Mb)
#Cpu time = 19:18:40
#Elapsed time = 10:15:51
```

The final fragment shows the time the job completed.

```
#Increased memory = 92.98 (Mb)
#Total memory = 628.17 (Mb)
#Peak memory = 1019.30 (Mb)
#Complete global_detail_route on Fri Apr 16 10:14:33 2004
```

## Following a Basic Routing Strategy

In general, the first time you route a design, you should be able to accept the default values on the NanoRoute form. You can look at the Innovus log file to see the processes that the NanoRoute router runs and the problems it encounters. Then you can adjust the net attributes or run-time options to improve your results.

The strategy presented in this section shows how you can break the routing processes into steps, so you can analyze and solve problems easily. After each step, check for data problems and congestion and make repairs. Repeat the step and repair remaining violations. Continue this process until the design is free of violations before going to the next step.

## Using the InnovusText Commands

The following commands show the basic routing strategy using the Innovus text commands.

1. The router globally routes the design:

```
globalRoute
```

2. The router does the initial detailed routing (iteration 0 does not include a search-and-repair step), and saves the design as droute0:

```
setNanoRouteMode -drouteStartIteration 0
setNanoRouteMode -drouteEndIteration 0
detailRoute
saveDesign droute0
```

3. The router does the first search-and-repair iteration and saves the design for analysis:

```
setNanoRouteMode -drouteStartIteration 1
setNanoRouteMode -drouteEndIteration 1
detailRoute
saveDesign droute1
```

4. The router does the second to nineteenth search-and-repair iterations and saves the design for analysis. The switch box grows larger as the iteration number increases.

```
setNanoRouteMode -drouteStartIteration 2
setNanoRouteMode -drouteEndIteration 19
```

```
detailRoute  
saveDesign droute19
```

5. The router runs postroute optimization (`drouteEndIteration default`) and additional search-and-repair operations and saves the design as `droute`:

```
setNanoRouteMode -drouteStartIteration 20  
setNanoRouteMode -drouteEndIteration default  
detailRoute  
saveDesign droute
```

## Using the Innovus GUI

The following section describes the basic routing strategy using the GUI.

### Run Global Routing

1. Choose *Route - NanoRoute - Route*.
2. Select *Global Route* on the NanoRoute form.
3. Click *OK*.
4. Save as `groute`.
5. Check the congestion map.

If you see congested areas after global routing, your design is unroutable.

## Run Initial Detailed Routing

1. Choose *Route - NanoRoute - Route*.
2. Set the following options on the NanoRoute form:
  - Detail Route*
  - Start Iteration 0*
  - End Iteration 0*
3. Click *OK*.
4. Save the design as `droute0`.
5. Check the violations in the log file.

If you have many violations on *metal1* and *metal2*, you probably have pin-access problems, incorrect track settings, or overlapped cells. Check your LEF file and correct any problems. See [Evaluating Violations](#) for an excerpt of a log file from a design with many violations on *metal1* and *metal2*.

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

## Run Search and Repair

Break search and repair into two phases. Check congestion after each phase and repair violations.

To run the first phase of search and repair, complete the following steps:

1. Choose *Route - NanoRoute - Route*.
2. Set the following options on the NanoRoute form:
  - Detail Route*
  - Start Iteration 1*
  - End Iteration 1*
3. Click *OK*.
4. Save the design as `droute1`.
5. Check the violations in the log file and graphically.

To run the second search-and-repair phase, complete the following steps:

1. Choose *Route - NanoRoute - Route*.
2. Set the following options on the NanoRoute form:

- Detail Route*
- Start Iteration 2*
- End Iteration 19*

In this phase, the router makes additional search-and-repair passes. It reroutes nets with violations within a local area (a switch box). In each successive pass, the size of the switch box size increases, so the router can make the repairs over larger areas.

3. Click *OK*.
4. Save the design as `droute19`.
5. Check congestion.

If you still have many violations (more than 1,000) or an unbalanced distribution of violations, you might still have a problem with the data or a congested design.

For help resolving the violations, see [Evaluating Violations](#).

## Run Postroute Optimization

Ensure your data and library are violation-free before you run postroute optimization, or the router might spend a lot of time trying to repair violations that it cannot repair. Postroute optimization takes longer than any of the other steps because the router does more rigorous search and repair during postroute optimization than previous steps.

To run postroute optimization, complete the following steps:

1. Choose *Route - NanoRoute - Route*.
2. Set the following options on the NanoRoute form:
  - Detail Route*
  - Start Iteration 20*
  - End Iteration default*

**Note:** In general, do not set *Start Iteration* or *End Iteration* higher than 20 because it does not increase the quality of results.

1. Click *OK*.
2. Save the design as `droute`.

During postroute optimization, the router runs both global and detailed routing and makes global changes to repair violations.

## Checking Congestion

Check congestion in your design after global routing by using the Congestion Analysis Table in the Innovus log file and the congestion map in the Innovus main window.

## Using the Congestion Analysis Table

The congestion analysis table shows the distribution and severity of congestion in global routing cells (gcells) on each routing layer.

**Note:** For information on global routing and on gcells, see [Global Routing](#).

Following is an example of a Congestion Analysis table:

| Layer   | Congestion Analysis:       |                            |                            |                             |                   |
|---------|----------------------------|----------------------------|----------------------------|-----------------------------|-------------------|
|         | OverCon<br>#Gcell<br>(1-2) | OverCon<br>#Gcell<br>(3-4) | OverCon<br>#Gcell<br>(5-6) | OverCon<br>#Gcell<br>(7-12) | %Gcell<br>OverCon |
| Metal 1 | 22 (0.01%)                 | 10 (0.00%)                 | 0 (0.00%)                  | 0 (0.00%)                   | (0.01%)           |
| Metal 2 | 5531 (2.39%)               | 1680 (0.73%)               | 370 (0.16%)                | 123 (0.05%)                 | (3.33%)           |
| Metal 3 | 4114 (1.78%)               | 19 (0.01%)                 | 0 (0.00%)                  | 0 (0.00%)                   | (1.79%)           |
| Metal 4 | 1333 (0.58%)               | 137 (0.06%)                | 0 (0.00%)                  | 0 (0.00%)                   | (0.64%)           |
| Metal 5 | 5852 (2.53%)               | 4 (0.00%)                  | 0 (0.00%)                  | 0 (0.00%)                   | (2.53%)           |
| Metal 6 | 27 (0.01%)                 | 0 (0.00%)                  | 0 (0.00%)                  | 0 (0.00%)                   | (0.01%)           |
| Total   | 16879 (1.22%)              | 1850 (0.13%)               | 370 (0.03%)                | 123 (0.01%)                 | (1.39%)           |

#Max overcon = 12 tracks.

#Total overcon = 1.39%

#Worst layer Gcell overcon rate = 2.53%

- The first column, Layer, lists the metal layers that have over-congested gcells. The NanoRoute router marks a gcells as over-congested if the global router has assigned more nets to the gcell than the gcell has available tracks.
- The second through fifth columns, labeled OverCon #Gcell, list the number and percentage of

gcells on each layer that are over-congested.

- The numbers in parentheses after OverCon #Gcell indicate how many additional tracks within the gcell are needed to accommodate the global routing assignments. For example, OverCon #Gcell (1 - 2) means that one or two additional tracks are needed to accommodate all the nets that the global router has assigned the gcells listed in the column. As you move from left to right in the table, congestion increases because the difference between the number of nets assigned to the gcell by the global router and number of available tracks within the gcell increases.
- The number of columns in the table is determined by the number of additional tracks needed by the gcells with the worst congestion. For example, if the most over-congested gcells need only four additional tracks, the table would include columns for 1-2 and 3-4 tracks, but not for 5-6 or more tracks.
- The NanoRoute router creates only one column for gcells that need seven or more additional tracks. In the example, all gcells that need seven to 12 additional tracks are listed in the column labelled OverCon #Gcell (7 - 12).
- The NanoRoute router displays the maximum number of tracks needed in the last OverCon #Gcell column. In the example, the maximum number of tracks needed is 12. If some gcells needed 14 more tracks, the column would be labelled OverCon #Gcell (7-14). If the maximum number of tracks needed were only eight, the column would be labelled OverCon #Gcell (7-8).

Within each column, the table does not indicate exactly how many additional tracks are needed. For example, in the column labelled OverCon #Gcell (7-12), The NanoRoute router does not distinguish between gcells that need seven, eight, nine, ten, 11, or 12 additional tracks.

- The last column, %Gcell OverCon, lists the percentage of all gcells on the layer that are over-congested. In the example, on layer Metal 1, only 0.01% of the gcells are over-congested.
- The last row of the table, Total, lists the total number and percentage of over-congested gcells in each column. In the example, 1,850 gcells in the design, or 0.13% of all gcells, need three or four more tracks.
- The last row of the last column displays the overall percentage of over-congested gcells in the design. In the example, 1.39% of all cells are over-congested.
- Following the table NanoRoute summarizes a few key values. The maximum number of tracks any Gcell needed, the total over congestion number for all layers, and the worst layers Gcell congestion rate.
- The worst layer Gcell overcon rate is intended to report the routing congestion so the pin access layer or the layer below the pin access layer is not reported even if it is higher.

## Interpreting the Table

- Read the table horizontally to see the distribution and percentage of gcells on each layer that have a greater demand for tracks than they have supply of tracks.
- Read the table vertically to see which layers have the most over-congested gcells and how severe the congestion is.
- The table does not show how closely the over-congested gcells are clustered. Look at the congestion map in the GUI to see clusters of congestion and their exact location.
- There is no specific number that determines whether the design is routable. In general, the more columns, and the more the percentages increase toward the right side of the table, the worse the congestion.

## Using the Congestion Map

Check obstructions and congestion in your design graphically by analyzing a congestion map. The information in the map is directly extracted from the router after you run global routing. You choose the layers to display on the map. The Innovus software displays the congestion map in the main window when you complete the following steps:

1. Globally route the design.
2. Select *Physical view* in the *Views* area of the Innovus main window.
3. Click the *All Colors* button. This displays *Color Preferences* form.
4. Select the *View Only* tab.
5. Make *Congestion* viewable.
6. Select both *Horizontal Congest* and *Vertical Congest*.

For more information on selecting the objects and colors, see [The Main Window](#) in the *Innovus Menu Reference*.

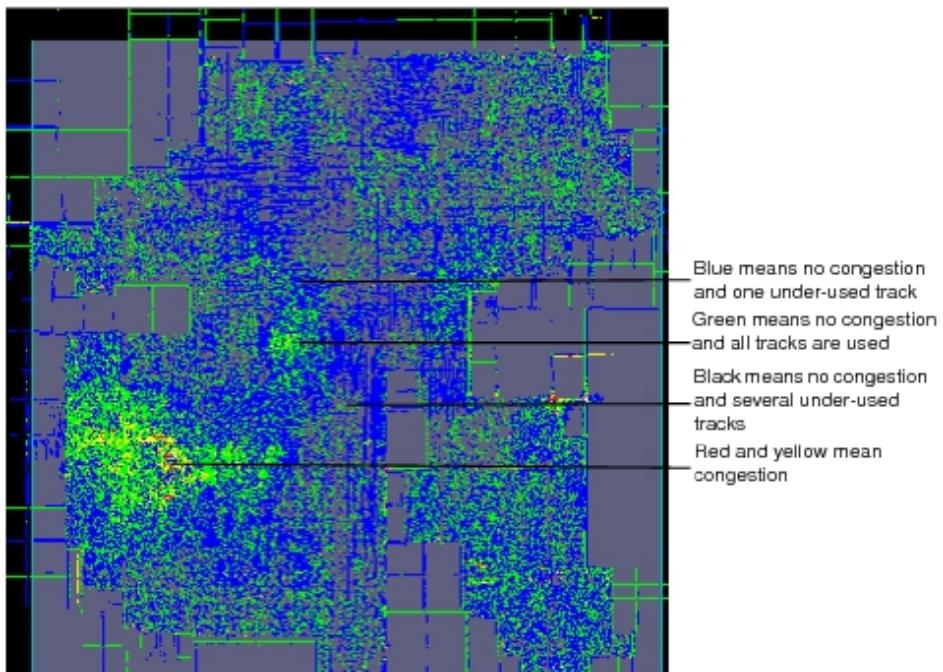
## Interpreting the Congestion Map

In the map, blue or black indicate an acceptable level of congestion; white indicates an unacceptable level. However, this depends on your design. For example, a design that is mostly uncongested might have small areas (often called hot spots) that are highly congested. You must look at the overall congestion graphically to assess routability.

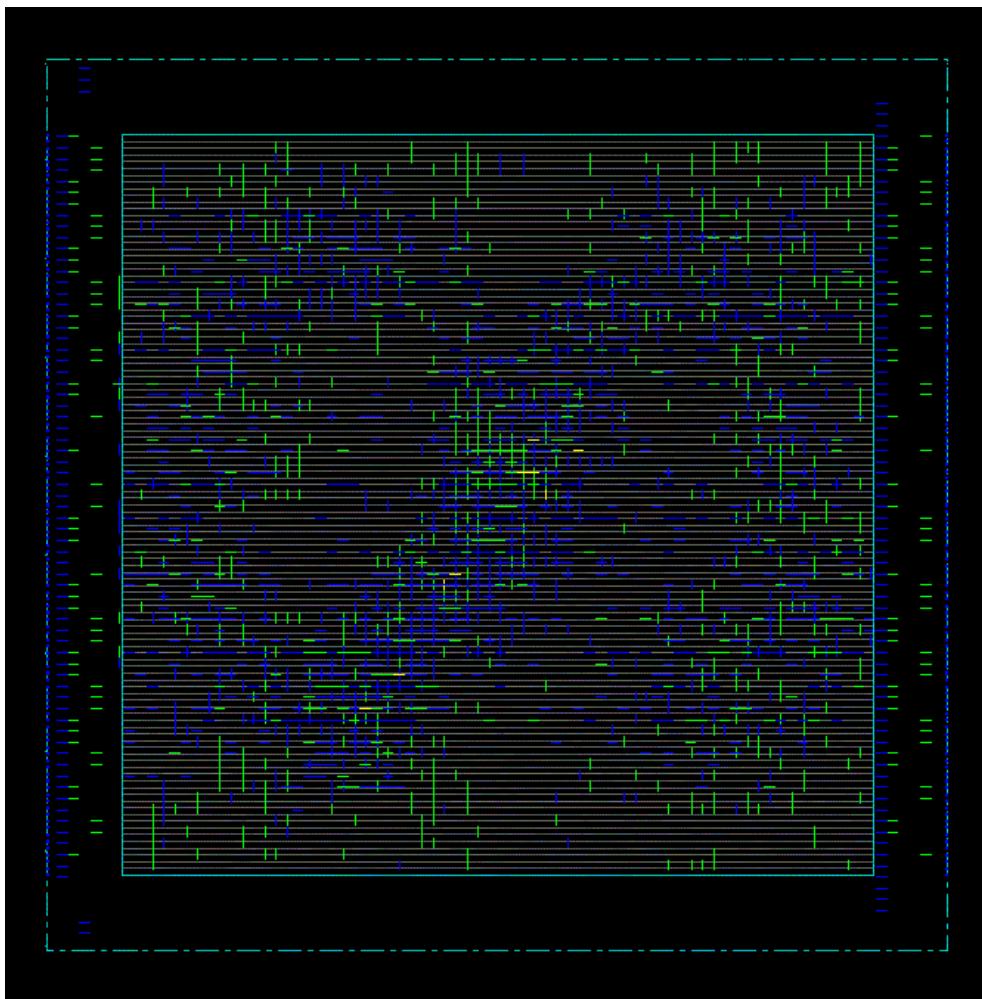
The following table explains the meaning of the default colors in the congestion map:

| Color   | Explanation                                                                 |
|---------|-----------------------------------------------------------------------------|
| Black   | No congestion: You have at least two tracks that are under-used.            |
| Blue    | No congestion: You probably have one track that is under-used.              |
| Green   | No congestion: All the tracks are used.                                     |
| Yellow  | Low congestion: You probably have one track that is over-used.              |
| Red     | Some congestion: You probably have two tracks that are over-used.           |
| Magenta | Moderate congestion: You probably have three tracks that are over-used.     |
| White   | High congestion: You probably have at least four tracks that are over-used. |

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



In the congestion map shown below, the design is not congested.



## Resolving Open Nets

If the router cannot complete the connection of a net during routing, it generates an open net warning message in the Innovus log file and sets the net status to open. Additionally, the log file provides a list of open nets in summary format.

- During detailed routing, problems with pin modelling, routing track definitions, floorplanning, or conflicts between `setNanoRouteMode` option settings can cause open nets.
- During global routing, missing power or ground routing can cause open nets.

To resolve open net problems, complete the following steps:

1. Run `verifyTracks` to diagnose a subset of open net problems in standard cells. This command generates a report in the Innovus log file. Use the report to determine the specific cause of the

- open net. For more information, see [Diagnosing Problems Using verifyTracks](#).
2. Determine the cause of the remaining problems--mostly those caused by option conflicts or libraries--by manual analysis. For more information, see [Resolving Additional Open Net Problems](#)
  3. Resolve the problems.
  4. Re-run global and detailed routing.

## Log File Examples

The following examples show sections of an Innovus log file that includes five open net warning messages generated during detailed routing:

```
#Start Detail Routing.  
#start initial detail routing ...  
#WARNING (NR) Fail to route NET example56/cp_aclk_2 in region ( 302.295 272.894 331.695  
306.495) Set net status to open.  
#WARNING (NR) Fail to route NET example56/cp_aclk_3 in region ( 302.295 272.894 331.695  
306.495) Set net status to open.  
...  
  
#start 1st optimization iteration ...  
#WARNING (NR) Fail to route NET example12/cp_bclk_5 in region ( 402.295 372.894 431.695  
406.495) Set net status to open.  
#WARNING (NR) Fail to route NET example12/cp_bclk_6 in region ( 402.295 372.894 431.695  
406.495) Set net status to open.  
...  
#start 2nd optimization iteration ...  
#WARNING (NR) Fail to route NET example99/cp_cclk_8 in region ( 502.295 472.894 531.695  
506.495) Set net status to open.  
...
```

The following section of the same log file includes the open net summary:

```
# number of violations = 0  
#cpu time = 00:00:01, elapsed time = 00:00:01, memory = 51.15 (Mb)  
#Complete Detail Routing.  
#WARNING (NR) There are 5 open nets.  
#Please refer to Innovus User Guide for details of open net messages and possible root  
causes.  
#After resolving it, please re-run globalDetailRoute command.  
#List of open nets :  
# example56/cp_aclk_2
```

```
# example56/cp_aclk_3
# example12/cp_bclk_5
# example12/cp_bclk_6
# example99/cp_cclk_8
#
#Total wire length = 340827 um.
#Total half perimeter of net bounding box = 298122 um.
```

## Diagnosing Problems Using verifyTracks

The `verifyTracks` command reports the following types of problems in the Innovus log file:

- Pins that are too far inside a blockage  
For more information, see Macro Obstruction Statement syntax and the accompanying figures in the "LEF Syntax" chapter of the LEF/DEF Language Reference.
- Pins that are not aligned with routing tracks  
Align pins with routing tracks to assure the maximum number of pickup points.
- Pins that are above or underneath power stripes on the adjacent metal layer  
The router might not able to access a pin if it is blocked by a power stripe.

For more information, see [verifyTracks](#).

## Resolving Additional Open Net Problems

If the router generates an open net message after you correct the problems reported by `verifyTracks`, or if `verifyTracks` does not report any problems, check for the following additional problems:

- Pin modelling or library problems
  - Pins without physical geometry
  - Pins that are less than the minimum width
  - Minimum-width pins that are placed off the manufacturing grid
  - Pins that are blocked for planar access, and are not accessible through a via without violating the adjacent-cut rule
  - Pins that trigger multiple-cut vias, but no multiple-cut vias are specified in the LEF file
- Floorplanning problems

- Cell overlaps introduced during placement  
Use the `checkPlace` command to check for cell overlaps. For information, see [checkPlace](#) in the *Innovus Text Command Reference*.
- Problems caused by `setNanoRouteMode` option settings or conflicts between option settings and library specifications
  - No via access in pin but `-routeWithViaOnlyForStandardCellPin true` is specified
  - No via access in pin but `-routeBottomRoutingLayer` is too high or `-routeTopRoutingLayer` is too low for the router to connect without using a via
  - Via stacking is not allowed but `-routeBottomRoutingLayer` is higher than the pin layer (or `-routeTopRoutingLayer` is lower than the pin layer) so via stacking is required to reach the pin
- Problems caused by missing power or ground routing
  - Missing special routes for stripes or followpins to connect tie-high or tie-low nets causes open power or ground nets during global routing.  
The global router issues open net warning messages such as the following:  
`#WARNING (NR) There is no prerouted stripe wire within routing layer range 1:9 for special net VSS.`  
`#WARNING (NR) Please reroute special net wires before running NR.`

## Running Timing-Driven Routing

In the Innovus environment, during timing-driven routing, the router uses the Common Timing Engine (CTE) by default. All the related tasks (route estimation for the timing graph, capacitance extraction, timing analysis, timing graph generation) are executed within the Innovus environment.

Timing-driven routing might cause longer run time and more violations than nontiming-driven routing. For information, see [Violations in Timing-Driven Routing](#).

## Input Files

To run timing-driven routing you need the following files:

- Physical libraries in LEF
- Timing library in .lib format
- Timing constraints in .sdc format or a timing graph

**Note:** For information on the timing constraints that are compatible with the Innovus CTE, see [Data Preparation](#).

- Extended capacitance table generated by the Innovus software
- Netlist in DEF or Verilog format
- Placed design in DEF

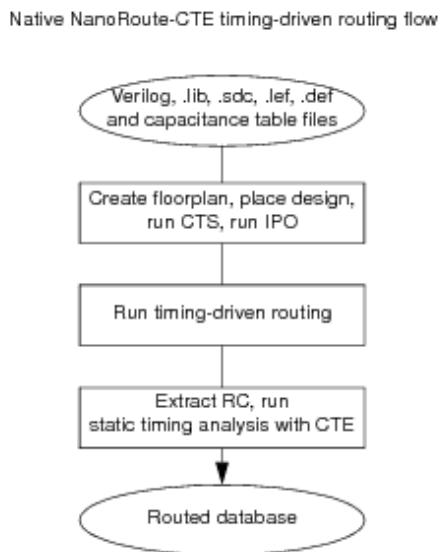
## Using the CTE and the NanoRoute Router in Native Mode

The Figure 1 shows the design flow for routing with NanoRoute in native mode using the CTE. In native mode, the router uses the CTE as its default timing engine.

In native mode, the following commands use the CTE:

```
setNanoRouteMode -routeWithTimingDriven true  
globalDetailRoute
```

**Figure 1: Flow for Routing - NanoRoute in Native Mode**



## Using the CTE and Standalone NanoRoute

Figure 2 shows the design flow for standalone NanoRoute using the CTE. The standalone NanoRoute router is loosely integrated with the CTE.

In standalone mode, the following commands use the CTE:

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

The flow is shown in three main steps:

1. Generating a timing graph

To generate a timing graph, load your design into the Innovus software and use the **Innovus writeDesignTiming** command.

```
writeDesignTiming design.tif
```

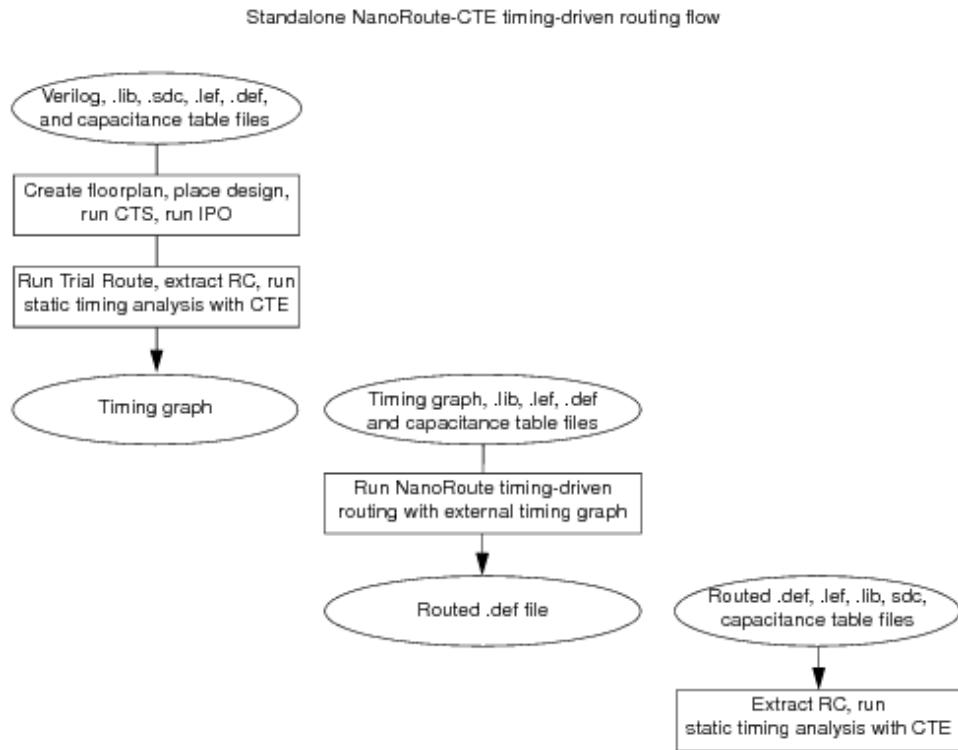
2. Routing

When you route the design, type the following commands:

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

3. Extracting capacitance and analyzing timing

**Figure 2 : Design Flow for Standalone NanoRoute**



## Routing Clocks

Route clock nets before routing the rest of the signal nets. If you are using the `routeDesign` supercommand, the NanoRoute router changes the status of clock nets from `FIXED` to `ROUTED` so they can be moved and routes them before routing other nets.

This section gives additional information on you can use to route clocks manually.

Layer assignments for clock nets might not correlate in global and detailed routing. For tight control over clock timing, run global and detailed routing on clock nets before routing other nets. Fix the locations of the nets during detailed routing and unfix them during postroute optimization. Use net weights to ensure priority during search and repair.

## Setting Attributes for Clock Nets

If clock nets are marked `USE CLOCK` in the DEF file or you have defined a clock net in the Innovus database, the router automatically sets the following values. You can change the values by setting attributes on the NanoRoute Attributes form. If the clock nets are not defined, type the name of a clock net in the *Net Name(s)* text box to set attributes for the net.

- *Weight*
- The default net weight for clock nets is 10 to give clock nets priority during global routing (the default net weight for other nets is 2). During global routing, the router goes from global routing cell to global routing cell within each switch box, and routes the nets with the highest weight first.
- *Bottom Layer*  
The default bottom layer for routing clock nets is 3, to ensure that the router has access to *meta1* pins during routing. This attribute sets a soft limit, and the router might route some nets on lower layers, if necessary to complete the routing.
- *Top Layer*
- The default top layer for routing clock nets is 4. This attribute sets a soft limit, and the router might route some nets on upper layers, if necessary to complete the routing.
- *Avoid Detour*  
*Avoid Detour* is *True* for clock nets, so they are routed as straight as possible.

Set the following attribute in the Innovus console, using the `setAttribute` command:

`-preferred_extra_space 1`

The `-preferred_extra_space` parameter adds spacing around the clock nets, which improves coupling capacitance. It is not included on the NanoRoute/Attributes form.

**Tip:** Select *SI Prevention True* to set *Weight*, *Avoid Detour* and `-preferred_extra_space` all at once. *SI Prevention True* sets *Weight* to 10, *Avoid Detour* to *True*, and `-preferred_extra_space` to 1 for clock nets.

## Routing Clock Nets Using the GUI Forms

Specify the following options on the NanoRoute form:

- *Selected Nets*  
Specify *Selected Nets* to route the clock nets first. Unlike the *Weight* attribute, which gives priority to routing nets within a switch box, *Selected Nets* is a global option that routes whole nets.
- *Global Route*

- *Detail Route*

Specify *End Iteration 19* to stop routing before the postRoute optimization step.

## Running Postroute Optimization

To prevent rip-up and rerouting of clock nets during postroute optimization, specify the following:

- On the NanoRoute/Attributes form, keep the attributes you have already set, and specify *Skip Routing True*.
- On the NanoRoute form, specify *Start Iteration 20* and *End Iteration default*.

## Related Topics

- [Using the routeDesign Supercommand](#)
- [routeDesign](#)
- [Legacy FE-CTS Capabilities](#)

## Preventing and Repairing Crosstalk Problems

During SMART routing, the NanoRoute router automatically prevents crosstalk problems by wire spacing, net ordering, minimizing the use of long parallel wires, and selecting routing layers for noise-sensitive nets. The router performs these operations concurrently with other operations.

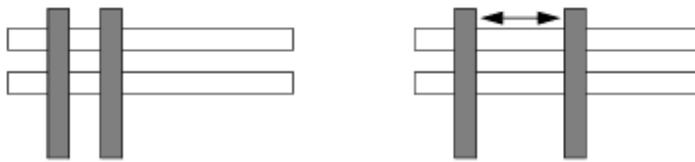
In addition to the operations it performs automatically, the router also performs shielded routing to protect critical wires from crosstalk.

During postroute signal integrity repair, the router performs these same operations.

The following sections describe the crosstalk prevention and repair operations the router performs, and whether you can set net attributes to control them.

- Wire Spacing

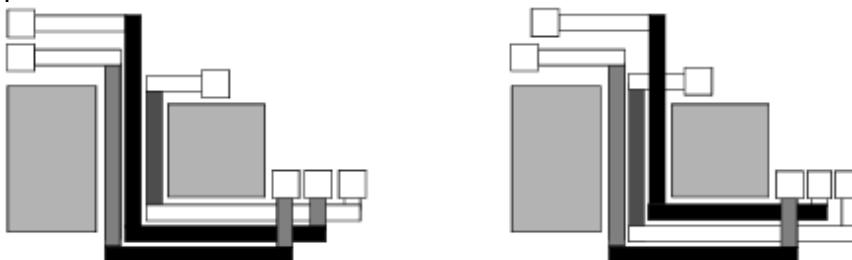
The router automatically adds extra space between critical nets. You can also use the `-preferred_extra_space` attribute to add space. For information on this attribute, see [setAttribute](#).



- Net Ordering

The router automatically routes critical nets first and avoids detours on those nets so they are as short as possible.

- You can also use the `-weight` attribute to give priority to critical nets within a switch box, so they are routed first.
- You can use the `-avoid_detour` attribute to ensure that critical nets are routed as short as possible.



- Minimizing the use of long parallel wires

The router automatically minimizes the use of long parallel wires, based on an internal algorithm. You cannot set an attribute to control this feature.



- Selecting routing layers

The router automatically restricts routing layers for critical nets to reduce both coupling and resistance. It routes clocks on layers 3 and 4.

- You can set the `-bottom_preferred_routing_layer` and `-top_preferred_routing_layer` attributes to specify preferred layers for critical nets.
- You can specify how strictly to enforce these attributes by specifying the `-preferred_routing_layer_effort` attribute.



- Shielding

The router can shield critical nets with power or ground wires to protect them from coupling. Shielding is not an automatic operation--you control it with the `-shield_net` attribute.

## Related Topic

- Performing Shielded Routing

## Crosstalk Prevention Options

To minimize problems caused by crosstalk, set the following NanoRoute options:

```
setNanoRouteMode -routeWithTimingDriven true  
setNanoRouteMode -routeWithSiDriven true
```

These options specify timing-driven and signal integrity-driven routing and fine-tune the priorities the router assigns to timing, signal integrity, and congestion. Use these options together to minimize crosstalk. After meeting the timing requirements of your design, adjust the values and rerun routing, following these guidelines:

- If your design is congested, use a low timing-driven effort.
  - If your design is not congested, use a high timing-driven effort
- Tip:** Because designs with severe signal-integrity problems are usually not congested, use a high timing-driven effort for those designs.
- If increasing the timing-driven effort creates a jump in the number of timing violations, decrease the timing-driven effort.

For more information on these options, see [setNanoRouteMode](#).

For more information, see [Analyzing and Repairing Crosstalk](#).

## Running ECO Routing

The NanoRoute router performs ECO routing by completing partial routes with added logic while maintaining the existing wire segments as much as possible. ECO routing is useful in cases such as the following:

- After the chip is initially routed, the customer or chip owner gives you a new netlist with minor changes.
- After the chip is initially routed, buffers were added to repair setup or hold violations or DRVs during physical optimization.
- Buffers were added or gates were resized during hand editing of a routed design.
- Antenna diodes were added interactively after routing to repair process antenna violations.
- After metal fill is added to the design.

During ECO routing, the router does the following:

- Reroutes partial routes and nets without routing.  
You can use wire editing commands to partially preroute wires to guide global ECO routing. The router does not globally reroute nets that are automatically prerouted, such as clock nets, but it might make minor routing changes to preroutes to increase routability of the design. Examples of minor routing changes include the following:
  - Completely moving a preroute
  - Changing the routing topology within the current routing switch box.
- Retains fully prerouted nets and pin-to-pin paths.
- Might use dangling paths in order to complete routes, but removes dangling wires left after global routing.
- Keeps connectivity within the bounding box, but does not constrain layers or positions.

## ECO Limitations

- Do not use the `globalRoute` command in ECO mode. To route in ECO mode, use `globalDetailRoute`.
- If more than 10 percent of the nets are new or partially routed, run full global and detailed routing instead of ECO routing.

## ECO Flow

To perform ECO routing, specify the following commands and options:

```
setNanoRouteMode -routeWithEco true
```

```
globalDetailRoute
```

**Info:** The `- routeWithEco` option constrains changes but might lead to violations or long run times if it causes the router to move more signals to resolve the routing.

## Specifying Nets for ECO Routing

The router automatically identifies the nets that need changes during ECO routing.

To route only a few nets, and skip routing on all the other nets, specify the following commands:

```
setAttribute -net @PREROUTED -skip_routing true
```

```
setAttribute -net eco_net_name1 -skip_routing false
```

```
setAttribute -net eco_net_name2 -skip_routing false
```

## ECO Routing After Multiple-Cut Via Insertion

If your design is already fully routed and multiple-cut vias have been inserted for manufacturing, specify the following commands for ECO route:

```
setNanoRouteMode -routeWithEco true
```

```
setNanoRouteMode -drouteUseMultiCutViaEffort low
```

```
globalDetailRoute
```

For more information on using Innovus ECO commands and flows, see [Interactive ECO](#).

## Evaluating Violations

## DRC Marker Name Comparison Table

During the detail routing stage, the NanoRoute router reports violations after each iteration. The NanoRoute violation summary is printed to log file by default.

The following is an example of the violation table:

```
# number of DRC violations = 13975
#
# By Layer and Type :
#          MetSpc Notch Short NdrSpc Mar EolOpp Others Totals
#    M1        0     0     0     0     0     0     1     1
#    M2        1     0     0     0     0     0     1     2
#    M3        1     0     0     0     0     1     0     2
#    M4        2     1    15     0     1     2     4    25
#    M5        0     0     7     0    39     1     1    48
#    M6        2     5    19     0     5     1     5    37
#    M7        1     0     0   7527     0     0     2   7530
#    M8        0     0     4   6322     0     0     1   6327
#    M9        1     0     0     0     1     0     1     3
#    Totals    8     6    45   13849    46     5    16   13975
#
# By Non-Default Rule:
#      Rule           M1  M2  M3  M4  M5  M6  M7  M8  Totals
# DBLCUT_DBSPACE_RULE  +S  0  0  0  0  0  0  8396  6617  15013
#    Totals          0  0  0  0  0  0  8396  6617  15013
```

The violation table is *Per-Layer* and *Per-Rule* violation count. Hence,  $M7 \text{ NdrSpc } 7527$  means there are 7527 NDR spacing violations on M7 layer, and such NDR rules are defined as hard rules(with HARDSPACING). The Non-Default Rule table is *Per-NDR-Rule* and *Per-Layer* VIOLATED-NDR-NET count.

Hence,  $DBLCUT_DBSPACE_RULE \text{ 8296 } M7$  means there are 8296 NDR net violation locations.

More precisely,

- If one NdrSpc violation is between a NDR net and a regular rule net, this counts for 1 in NDR table.
- If one NdrSpc violation is between a NDR net and another NDR net, this counts for 2 in NDR table.

Note: +SW means the spacing/width defined with non-default value in NDR rule for at least one layer.

+S means the spacing defined with non-default value in NDR rule for at least one layer.

+W means the width defined with non-default value in NDR rule for at least one layer.

The following table describes the DRC violations with abbreviations:

| Short Name | Marker Name |
|------------|-------------|
| Enc        | Enclosure   |

|          |                                |
|----------|--------------------------------|
| Enc2Jt   | EnclosureToJoint               |
| EncEdg   | EnclosureEdge                  |
| EncEO    | EnclosureEdgeOpposite          |
| EncPrl   | EnclosureParallel              |
| MinEnc   | MinimumEnclosedArea            |
| CutDen   | CutDensity                     |
| AdjCut   | AdjacentCutSpacing             |
| C2MCon   | CutToMetalConcaveCornerSpacing |
| C2MCvx   | CutToMetalConvexCornerSpacing  |
| C2MO     | CutToMetalOrthogonalSpacing    |
| C2MSpc   | CutToMetalSpacing              |
| C2MWW    | CuttoWrongWayMetalSpacing      |
| CShort   | CutShort                       |
| CutInr   | DifferentLayerCutSpacing       |
| CutOrt   | CutOrthogonalSpacing           |
| CutSpc   | SameLayerCutSpacing            |
| IsoCut   | IsolatedCut                    |
| SharedE  | Same-Metal-Share-EdgeSpacing   |
| EOLspc   | EndOfLineSpacing               |
| EOLColor | EndOfLineColorSpacing          |
| EolExt   | EolExtensionSpacing            |
| EolOpp   | OppositeEolSpacing             |
| Mar      | MinimumArea                    |
| ColChg   | MetalColorChange               |

|        |                              |
|--------|------------------------------|
| Color  | SameMaskSpacing              |
| CorSpc | CornerSpacing                |
| DSLSpc | DirectionalSpanLengthSpacing |
| EncSpc | EnclosureSpacing             |
| InfSpc | InfluenceSpacing             |
| JCSpc  | JointCornerSpacing           |
| MetSpc | ParallelRunLengthSpacing     |
| NdrSpc | NonDefaultRuleSpacing        |
| NSMet  | Non-sufficientMetalOverlap   |
| PinAcc | PinAccessConstraint          |
| Short  | MetalShort                   |
| SpnSpc | ParallelSpanLengthSpacing    |
| VolSpc | VoltageSpacing               |
| MinCut | MinimumCut                   |
| MinStp | MinStep                      |
| CutCtr | CutOnCenterLine              |
| CutEol | CutEolSpacing                |
| MaxStk | MaxViaStack                  |
| Ant    | Antenna                      |
| FbdSp  | ForbiddenSpacing             |
| OffGrd | OffGridorWrongWay            |
| Protru | Protrusion                   |
| WMJ    | JogToJogSpacing              |
| WreExt | WireExtension                |

|        |                   |
|--------|-------------------|
| ArSpac | MetalAreaSpacing  |
| CorFil | CornerFillSpacing |
| Loop   | MetalLoop         |
| MaxWid | MaximumWidth      |
| MinEnc | MinHole           |
| MinWid | MinimumWidth      |
| Notch  | NotchSpacing      |
| SLTbl  | SpanLengthTable   |
| WidTbl | WidthTable        |

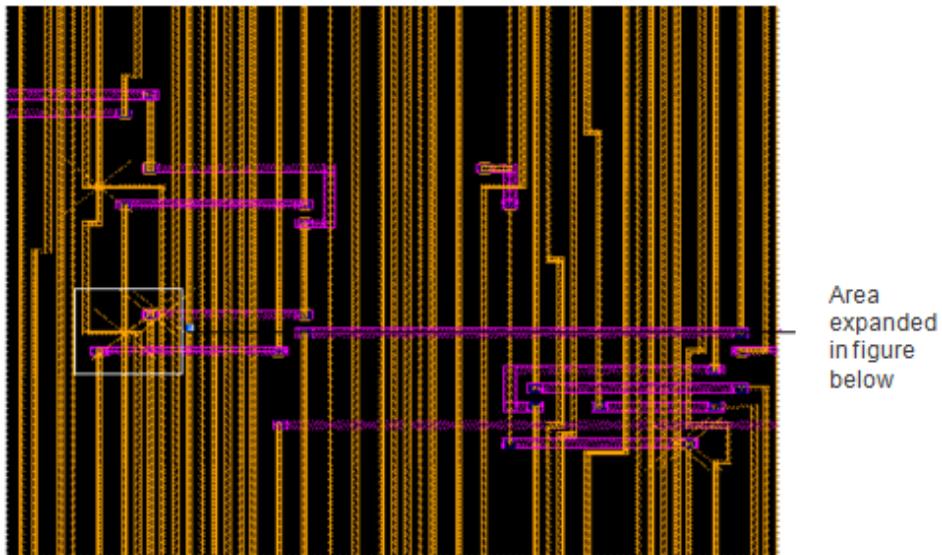
## **Violations on Upper Metal Layers**

Upper layers are typically used to route on top of macros where only a few routing layers are allowed. These upper layers typically have larger vias than lower layers. When the routing pitch is not set at line-to-via distance, two types of violations are likely to occur:

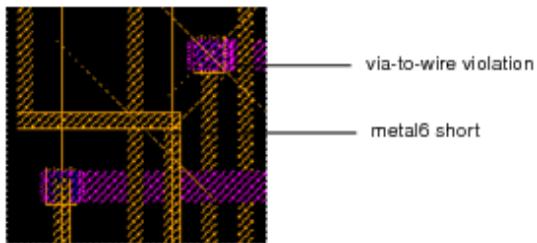
- Via-to-wire violations
- Shorts

Figure 3, Figure 4, and the LEF and DEF file excerpts that follow show a design with many violations on *metal6*.

**Figure 3: Design with Violations**



**Figure 4: Violations on Metal 6**



The relevant LEF file excerpt is:

```
LAYER Metal6
  TYPE ROUTING ;
  PITCH 0.46 ;
  WIDTH 0.2 ;
  SPACING 0.21
  DIRECTION VERTICAL ;
END Metal6
LAYER Metal7
  TYPE ROUTING ;
  PITCH 0.82 ;
  WIDTH 0.4
  SPACING 42 ;
  DIRECTION HORIZONTAL ;
END Metal7
VIA via6 DEFAULT
  LAYER Metal6 ;
  RECT -0.19 -0.23 0.19 0.23 ;
  LAYER Via6 ;
```

```
RECT -0.18 -0.18 0.18 0.18 ;
LAYER Metal7 ;
RECT 0.29 -0.2 0.29 0.2 ;
RESISTANCE 0.68
END via6
```

The relevant DEF file excerpt is:

```
TRACKS X -4749270 D0 6324 STEP 460 LAYER Metal6
```

To repair the shorts and via-to-wire violations, align the tracks as much as possible without sacrificing them. Change the TRACKS statement in the DEF file to have at least line-to-via STEP (pitch).

The line-to-via calculation for *meta6* is:

$$\begin{aligned}\text{Line to via meta6} &= 1/2 \text{ Width} + \text{Spacing} + 1/2 \text{ Via} \\ &= 0.1 + 0.21 + 0.19 \\ &= 0.5\end{aligned}$$

## Violations in Timing-Driven Routing

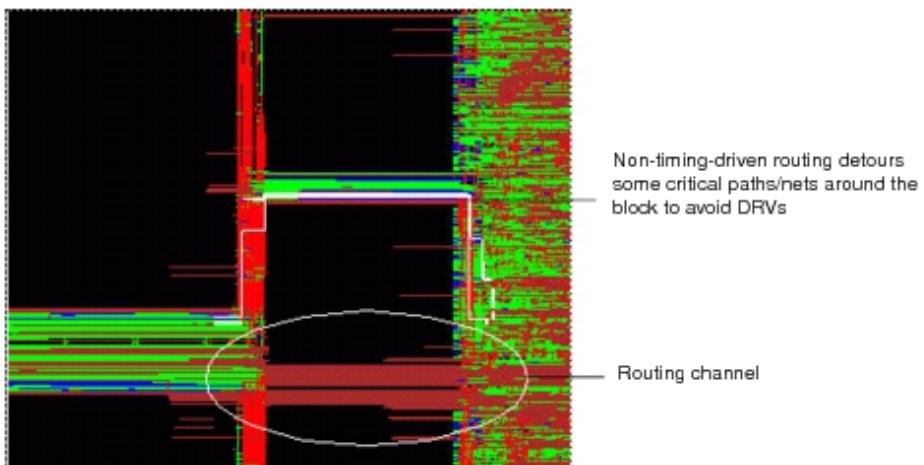
Run time and the number of violations often increase during timing-driven routing because the router restricts the routing of timing-critical nets.

During non-timing-driven routing, the router might detour some nets in order to avoid creating violations. In timing-driven mode, however, the router does not detour timing-critical nets. Instead, it forces them to be routed as short as possible, which can create congestion in the channels. Later, when design-rule checking takes precedence, the router detours timing-critical nets in overly congested channels.

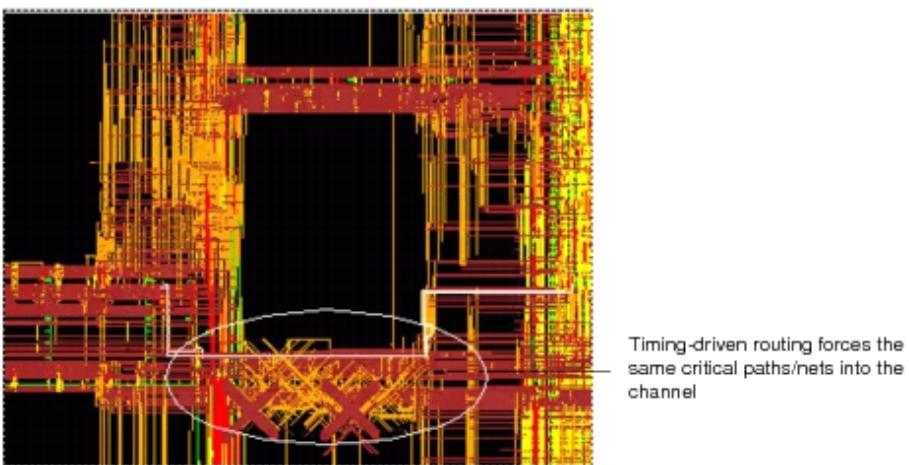
For information on the timing-driven routing flow, see [Running Timing-Driven Routing](#).

Figure 5 and Figure 6 illustrate non-timing-driven and timing-driven routing results for the same design.

### Figure 5: Non-Timing Driven Routing Results



**Figure 6: Timing Driven Routing Results**



## Deleting Violated Nets

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

## Related Topics

- [detailRoute](#)
- [ecoRoute](#)
- [globalDetailRoute](#)
- [setNanoRouteMode](#) - `routeWithEco`

## Using Additional Strategies to Repair Violations

### Process Antenna Violations

Repair process antenna violations with antenna repair options or the wire editing commands.

- For information on verifying process antenna violations, see "Verifying Process Antennas" in [Identifying and Viewing Violations](#) chapter of the *Innovus User Guide*.
- For information on process antenna options, see [Repairing Process Antenna Violations](#) .
- For information on wire editing, see [Editing Wires](#).

### Core Congestion

Ensure that blocks are placed in corners and near boundaries to help ease core congestion.

## Concurrent Routing and Multi-Cut Via Insertion

The NanoRoute router can insert multiple-cut vias during detailed routing in order to achieve a high ratio of multiple-cut to single-cut vias, minimize the number of vias in the design, and increase yield.

To specify the effort level for inserting multiple-cut vias and route the design concurrently, run the following commands:

```
setNanoRouteMode -drouteUseMultiCutViaEffort {medium | high}  
detailRoute
```

For more information on this parameter, see [setNanoRouteMode](#).

## Postroute Via Optimization

The NanoRoute router can optimize vias on a fully routed design by replacing single-cut vias with multiple-cut vias or with fat vias (single or multi-cut vias with an extended metal overhang). The router does not replace multiple-cut vias during this step.

The router replaces vias by substituting vias in the following order:

1. Fat double-cut vias
2. Normal double-cut vias
3. Fat single-cut vias

Ensure the following before replacing the vias:

- Double-cut vias and fat vias are automatically generated or defined in the LEF file.  
Use the `generateVias` command to generate vias.
- The design is completely global and detailed routed.  
If you delete any wires after routing, reroute the design before replacing the vias.
- The design is free of all DRC violations.

Complete the following steps:

1. To run postroute via reduction, type the following commands:

- `setNanoRouteMode -drouteMinSlackForWireOptimization slack`
- `setNanoRouteMode -droutePostRouteMinimizeViaCount true`
- `routeDesign -viaOpt`

**Note:** When you run these commands, the software optimize and reduce the via count. Use these commands only if no concurrent via optimization was done.

2. To run postroute single-cut to multiple-cut via swapping, complete one of the following steps:
  - a. To run postroute single-cut via to multiple-cut via swapping, type the following commands:

- `setNanoRouteMode -drouteMinSlackForWireOptimization slack`
- `setNanoRouteMode -droutePostRouteSwapVia multiCut`
- `routeDesign -viaOpt`

- b. To run non-timing-driven postroute single-cut via to multiple-cut via swapping, type the following commands:

- `setNanoRouteMode -routeWithTimingDriven false`
- `setNanoRouteMode -droutePostRouteSwapVia multiCut`

■ routeDesign -viaOpt

3. To raise some via priority in multiple-cut via swapping, type the following commands:

- setNanoRouteMode -routeWithTimingDriven false
- setNanoRouteMode -dbViaWeight "viaName viaWeight"
- setNanoRouteMode -droutePostRouteSwapVia multiCut
- routeDesign -viaOpt

## Related Topics

- Using the routeDesign Supercommand
- [routeDesign](#)

## Optimizing Vias in Selected Nets

To optimize vias in selected nets, set the `-skip_routing` attribute to true for all nets, then set the attribute to `false` for nets with vias you want to optimize.

```
setAttribute -net * -skip_routing true
setAttribute -net ... -skip_routing false
globalDetailRoute
```

## Via Optimization Options

Following are the via optimization options:

- `-droutePostRouteSwapVia`
- `-dbViaWeight`
- `-drouteUseMultiCutViaEffort`
- `-routeConcurrentMinimizeViaCountEffort`

To control the router to choose vias with the largest overhang first, specify the following option with

higher viaWeight than the other vias:

```
setNanoRouteMode -dbViaWeight {viaName viaWeight}
```

**Note:** You can specify the priority for any via by using the -dbViaWeight parameter, not just the largest overhang vias. For more information on -dbViaWeight parameter, see the [setNanoRouteMode](#) command.

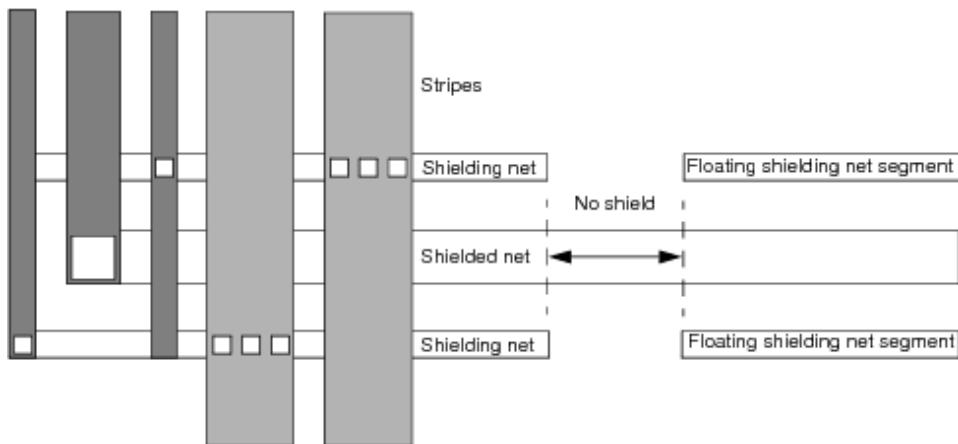
## Performing Shielded Routing

The NanoRoute router can protect noise-sensitive nets, such as clock nets, from crosstalk by shielding them with power or ground wires. NanoRoute automatically generates a shielding and statistics report after routing. For information on the report, see [Interpreting the Shielding Report](#).

The Figure below shows a section of a design with a shielded net. In the figure, you can see the following:

- The signal net is shielded by a power net on one side and a ground net on the other side.
- Multiple vias can be dropped where a stripe crosses the shielding net at a right angle, if the stripe is wide enough to accommodate them.
- A segment of the signal net is not shielded.
- There are some floating shielding net segments.

**Figure: Shielded Routing**



## Shielding Option

You can add more vias on the shielding wire for power stripe connection using the `editPowerVia` command after using the `createShield` command.

```
editPowerVia -add_vias 1 -skip_via_on_wire_status {[routed] [fixed] [cover] [shield]} -  
skip_via_on_wire_shape {[Blockring] [Stripe] [Followpin] [Corewire] [Blockwire] [Iowire]  
[Padring] [Ring] [Fillwire] [Noshape]}
```

**Note:** Some vias might be removed by NR during ECO. Use the `editPowerVia` command each time the shielding is recreated.

## Performing Shielded Routing Using the GUI

1. From the main menu, choose *Route - NanoRoute - Specify Attribute*. This opens the *NanoRoute/Attributes* form.
2. On the NanoRoute/Attributes form, enter the name of the net to shield (this is the shielded net in the figure) in the *Net Name(s)* field.
3. Enter the name of the power ground net (or both) in the *Shield Net(s)* field. These are the shielding nets in the figure.
  - To shield both sides with ground wires, enter the name of the ground net.
  - To shield one side with a ground wire and one side with a power wire, enter both the ground and the power net names.
4. Click *OK* or *Apply*.
5. Use the Innovus `selectNet` command to specify the net to shield. It must be the same as the net you specified on the NanoRoute/Attributes form.
6. From the main menu, choose *Route - NanoRoute - Route*. This opens the NanoRoute form.
7. On the NanoRoute form, select the following:
  - In the *Job Control* area, select *Selected Nets*.
  - In the *Mode* area select both *Global Route* and *Detail Route*.
8. Click *OK* or *Apply*.

To route the remaining nets, complete the following steps:

1. On the NanoRoute/Attributes form, set the *Skip Routing True* for the shielded nets.

**Tip:** You can also skip routing on prerouted nets by issuing the following command:

```
setAttribute -net @PREROUTED -skip_routing true
```

@PREROUTED applies to a net that has any wiring, including partial wiring.

2. On the NanoRoute form, deselect *Selected Nets Only*.
3. Click *OK* or *Apply* to reroute the design.

## Performing Shielded Routing Using Text Commands

- The following commands shield `net1` with both power and ground wires, and shield `net2` with a ground wire:

```
setAttribute -net net1 -shield_net vdd -shield_net vss  
setAttribute -net net2 -shield_net vss  
globalDetailRoute
```

- The following commands show how to shield two nets (do not shield more than one net with the same command):

```
setAttribute -net net1 -shield_net abc_gnd  
setAttribute -net net2 -shield_net abc_gnd
```

## Interpreting the Shielding Report

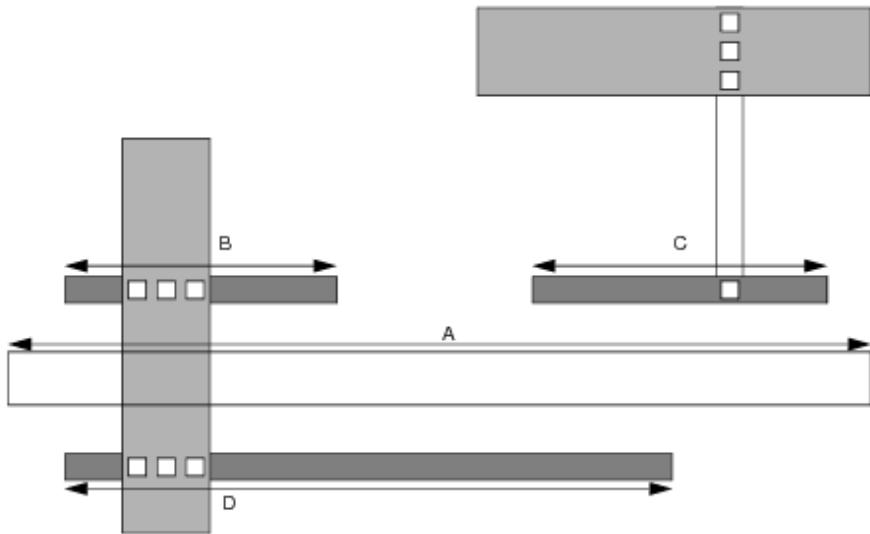
The software generates a shielding report for the NanoRoute router when you run the `reportShield` command. You can customize the report to output information on the whole design or on selected nets, and you can report per-layer statistics.

Following is a section of a report:

```
-----  
Name      : Shielded net name  
Length    : Shielded net length  
Shield    : Total length of shielding wire  
ratio     : Average shielding ratio  
-----  
Name      Length   Shield   Ratio    Layer:      Length   Shield   Ratio  
-----  
netA:      211.5   378.3   0.894          METAL2:      5.0       2.2      0.222
```

|              |          |       |       |       |
|--------------|----------|-------|-------|-------|
| METAL3 :     | 107.4    | 180.1 | 0.839 |       |
| METAL4 :     | 99.1     | 196.0 | 0.989 |       |
| <br>average: | 211.5    | 378.3 | 0.894 |       |
|              | METAL2 : | 5.0   | 2.2   | 0.222 |
|              | METAL3 : | 107.4 | 180.1 | 0.839 |
|              | METAL4 : | 99.1  | 196.0 | 0.989 |

To help understand the report, see the following figure, which shows a section of `netA`:



In the figure,

|             |                              |
|-------------|------------------------------|
| A           | Represents the shielded net. |
| B, C, and D | Represent shielding wires.   |

The software calculates the shielding ratio by using the following formula:

$$\text{Shielding Ratio} = \frac{B + C + D}{A \times 2}$$

## Reporting the Tapping Information for Shielded Nets

The software generates a shielding report for nets that fail the tapping check. The tapping information is printed in a report file with the name, `<design_name>_shield_tapping.rpt`. This file is generated by default when you run the `reportShield` command. The report lists the required tapping distance, the names of the shielded nets for which the check is being performed, the names of the violating layers, and the total number of missing tapping points. The report syntax and example are provided below.

Required tapping distance: <micron>

---

Shielded Nets : <Netname>

Violating Wire : <layer name> <shape box coordinates> : <micron>

Violating Wire : <layer name> <shape box coordinates> : <micron>

Shielded Nets : <Netname>

Violating Wire : <layer name> <shape box coordinates> : <micron>

Violating Wire : <layer name> <shape box coordinates> : floating

---

Total number of missing tapping points: <number>

---

A sample report is provided below:

---

Required tapping distance: 100.000

---

Shielded Nets: Net1

Violating Wire : M8 81.462 618.222 82.462 619.222 : 194.480

Violating Wire : M7 728.080 1.200 782.462 9.222 : 150.200

Shielded Nets: Net2

Violating Wire : M8 81.462 618.222 82.462 619.222 : 194.480

Violating Wire : M7 728.080 1.200 782.462 9.222 : floating

---

Total number of missing tapping points = 728

---

## Routing Wide Wires

The NanoRoute router automatically tapers wide wires when connecting to pins, including input/output pins of standard cells, macro cells, and block output pins. The tapered portion of a wire uses the minimum-width wire (the default width). If you do not want tapering at the output pins, specify the following parameter:

```
setNanoRouteMode -drouteNoTaperOnOutputPin true
```

**Note:** By default, the NanoRoute router prohibits tapering on top level pins.

## Using Non-Default Rules

By default, the NanoRoute router treats non-default rule spacing as a soft option; that is, when routing resources are available, it honors the non-default rule. If the area is too congested, and resources are not available, the router might not honor the rule. If you enable signal-integrity driven routing, the router attempts to minimize overall coupling capacitance in the design. If you enable timing-driven routing, the router also favors critical nets when choosing spacing. You can use up to 254 nondefault rules. Nondefault rules do not necessarily decrease the routing speed. Routing speed does decrease, however, due to the following factors:

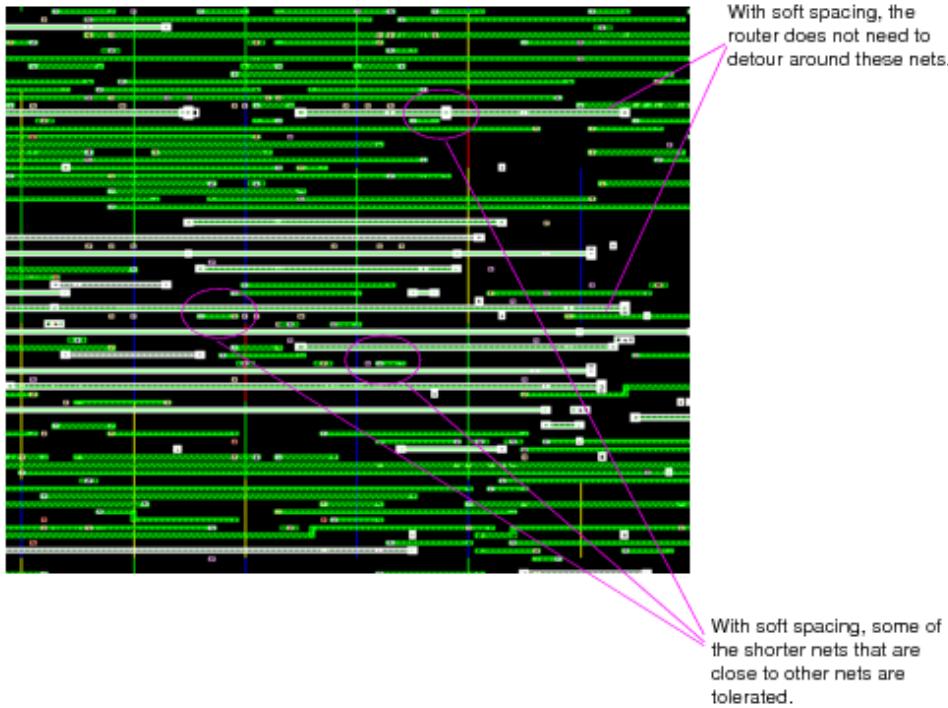
- The ratio of non-default rule wires to default rule wires increases.
- The amount of space between wires increases.
- The number of additional nondefault vias increases, due to the nondefault rules.

In congested areas, the router might violate nondefault spacing rules in order to avoid design-rule violations and complete the routing. Its flexibility with regard to nondefault spacing decreases the overall wirelength and benefits timing and signal integrity because it allows some shorter nets to be more easily tolerated near adjacent nets without causing violations.

**Note:** You can force the router to honor the nondefault rules by specifying the following option: `setNanoRouteMode -routeStrictlyHonorNonDefaultRule true`

Figure 8 illustrates nondefault spacing ("soft spacing") routing.

**Figure 8: Non-Default Spacing Routing**



## Repairing Process Antenna Violations

The NanoRoute router can repair process antenna violations concurrently with DRC violations during the search-and-repair step. During the postroute optimization step, when there are no more DRC violations, the router repairs additional process antenna violations. This two-step methodology allows the router to use more aggressive methods to repair the process antenna violations early on and saves CPU time. During postroute optimization, the router repairs antenna violations by changing layers (also called antenna stapling or layer hopping). It also repairs process antenna violations by inserting diode cells as close as possible to input gates to discharge current, and deleting and rerouting nets with violations.

**Note:** After routing, run the `globalNetConnect` command to ensure connectivity to power and ground pins in antenna cells added during process antenna repair. For information, see the `globalNetConnect` command.

The router supports hierarchical process antenna calculations and repair. For information on PAE calculations, and the LEF and DEF antenna parameters, see "Calculating and Fixing Process Antenna Violations" in the *LEF/DEF Language Reference*.

## Repairing Violations on Multiple-Pin Nets

On multiple-pin nets, the router does the following:

- On a two-pin net that has one input pin with antenna information and one output pin without antenna information, the router tries to repair the antenna violation based on input antenna information only.
- On a two-pin net that has one input pin without antenna information and one output pin with antenna information, there is usually no antenna violation on the output pin, so the router skips antenna repair.
- On a two pin-net where the router does not have any antenna information on either pin, the router skips antenna repair.
- On a three-pin net that has two input pins--one with antenna information and one without antenna information--and one output pin without antenna information, the router skips antenna repair.

## Changing Layers

The router automatically shortens wires whose area exceeds the gate/wire area ratio set in the LEF file. This process might not guarantee that it can resolve all antenna violations--if the routing area is congested, process antenna violations can still occur, just as shorts and spacing violations can occur.

## Using Diodes

The router inserts antenna diode cells or uses preplaced diode cells to repair violations. It can swap filler cells with antenna diode cells and fill the gap automatically if an antenna diode cell is not the same size as the filler cell it replaced. A later routing pass does not remove previously placed diodes. The antenna diode cells must have the same LEF SITE definition as the standard cells. Specify the diode cell name using the *Diode Cell Name* option on the NanoRoute form or the `-routeAntennaCellName` option on the text command line.

## Deleting and Rerouting Nets with Violations

If the design has more than 100 DRC violations, and you are using LEF 5.4 or later, the router deletes and reroutes nets with process antenna violations.

## Repairing Violations on Cut Layers

The NanoRoute router detects antenna violations on cut layers and repairs them by inserting diodes. To repair these violations, you must specify a value in your LEF file for the `ANTENNADIFFAREARATIO` (or `ANTENNACUMDIFFAREARATIO`) for the cut layers. For each cut layer, the value for `ANTENNADIFFAREARATIO` (or `ANTENNACUMDIFFAREARATIO`) must be larger than the value for `ANTENNAAREARATIO` (or `ANTENNACUMAREARATIO`).

**Info:** If you do not use diodes to repair process antenna violations, the router cannot repair the violations on cut layers.

**Tip:** To highlight the diodes that the router inserts, use the choose *Edit - Select by Name*. To highlight the diodes, type `*_antenna_*`.

For information on the *Select by Name* form, see "Edit Menu" in the *Innovus Menu Reference*.

To specify the diode cells, use `-routeAntennaCellName`.

**Tip:** To force the router to do more layer changing and skip diode insertion, specify the following option:  
`setNanoRouteMode -routeInsertAntennaDiode false`

After the router repairs as many violations as possible by layer changing, reset this option to `true` and repeat process antenna repair.

## Process Antenna Options

Use the following options to repair violations caused by process antennas:

- `setNanoRouteMode` options:
  - `-drouteFixAntenna`
  - `-routeAntennaCellName`
  - `-routeAntennaPinLimit`
  - `-routeFixTopLayerAntenna`
  - `-routeIgnoreAntennaTopCellPin`
  - `-routeInsertAntennaDiode`
  - `-routeInsertDiodeForClockNets`
- `setAttribute -nets netName -skip_antenna_fix`

## Examples

- The following commands show the basic strategy for repairing process antenna violations:

```
setNanoRouteMode -drouteFixAntenna true  
setNanoRouteMode -routeAntennaCellName "ANTENNA"  
setNanoRouteMode -routeInsertAntennaDiode true  
globalDetailRoute  
globalNetConnect
```

The NanoRoute router runs global and detailed routing. After repairing DRC violations, it repairs as many process antenna violations as it can by layer hopping during postroute optimization. If any process antenna violations remain, the router repairs them by inserting antenna diode cells named ANTENNA.

- The following commands repair process antenna violations by inserting diodes and filler cells. The filler cells are specified by the `setFillerMode -core` command. They fill any gaps that are left when a diode replaces a large filler cell.

```
setNanoRouteMode -routeInsertAntennaDiode true  
globalDetailRoute  
globalNetConnect
```

For information on adding filler cells, see the `setFillerMode` and `addFiller` commands.

## Creating RC Model Data in tQuantus Model File

You can create and store RC model data in a tQuantus model file, which is generated by the tQuantus extraction engine. The tQuantus extraction engine is an advanced extraction engine that is an improvement on the TQRC extraction engine for `postRoute -effortLevel medium` extraction. As compared to TQRC, tQuantus is much faster, consumes less memory, is deterministic, and provides results that are equivalent to signoff QoR. It is tightly integrated with NanoRoute and drives track assignment-based timing and SI optimization/postRoute optimization, and timing-driven routing.

The tQuantus model file generated using this engine has a benefit over the Quantus techfile in that it provides a simplified RC model data for implementation as compared to the Quantus techfile that has a complicated RC table. However, there is a small tradeoff for accuracy and runtime. You can create the tQuantus model file by using the `createTQuantusModelFile` command inside the Innovus environment.

Before creating the tQuantus model file, ensure that the following are available in the Innovus database:

- QRC techfiles
- Pitch information from the technology LEF file
- RC corners

## Use model for tQuantus Model File

The use model for the creation and usage of tQuantus model file is detailed below:

1. Generate the tQuantus model file using the `createTQuantusModelFile -file` command.
2. Once the tQuantus model file is available, run the following command to bring it into the Innovus environment:  
`setExtractRCMode -engine postRoute -effortLevel medium -tQuantusModelFile tQuantus_model_file`
3. Use the `checkTQuantusModelFile` command to check the consistency of the tQuantus model file against the various input files listed above inside the Innovus database. The following commands automatically invoke the `checkTQuantusModelFile` command to check the tQuantus file, if tQuantus is chosen as the extraction engine.
  - a. `routeDesign -trackOpt`
  - b. `timeDesign`
  - c. `optDesign -postRoute`
  - d. `extractRC`

**Note:** The software issues warnings if inconsistencies are detected in the RC corner information, QRC techfile, or pitch information in the LEF technology file. If such warnings are issued, the tQuantus model file will be generated automatically .

1. Run the following commands to use the tQuantus model file during track assignment-based timing and SI optimization (TA-Opt) and `optDesign -postRoute` optimization:
    - a. Specify the `-trackOpt` option of the `routeDesign` command. Specifying this parameter ensures that tQuantus model file is used during `timeDesign` and `optDesign -postRoute` optimization.  
`routeDesign -trackOpt`
- Note:** Specify the `-idealClock` parameter along with the `-trackOpt` parameter to enable the

ideal clock mode for optimization.

```
routeDesign -trackOpt -idealClock
```

- b. The `-tQuantusForPostRoute` parameter of the `setExtractRCMode` command is set to `true` by default. This setting instructs the software to use the tQuantus model file for `extractRC/optDesign -postRoute/timeDesign -postRoute` optimization. If this parameter is set to `false`, the software will revert to the TQRC use model. The complete behavior of `timeDesign` and `optDesign -postRoute` with respect to tQuantus model file usage is depicted in the diagram below:

| Setting1                                                  | Setting2                                          | Outcome                                                                                                                                                                                                      |
|-----------------------------------------------------------|---------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setExtractRCMode -tQuantusForPostRoute true</code>  | <code>setExtractRCMode -effortLevel medium</code> | Use tQuantus for <code>timeDesign / optDesign -postRoute</code> and <code>extractRC</code> . If the <code>setExtractRCMode -tQuantusModelFile</code> is not specified, it will be generated by the software. |
| <code>setExtractRCMode -tQuantusForPostRoute true</code>  | <code>setExtractRCMode -effortLevel other</code>  | Use "other" extraction for <code>timeDesign/optDesign -postRoute</code> and <code>extractRC</code> .                                                                                                         |
| <code>setExtractRCMode -tQuantusForPostRoute false</code> | <code>setExtractRCMode -effortLevel medium</code> | Issue a warning stating that TQRC is no longer the default engine and will be obsoleted in a future release. However, the software will proceed with TQRC for <code>timeDesign/optDesign -postRoute</code> . |
| <code>setExtractRCMode -tQuantusForPostRoute false</code> | <code>setExtractRCMode -effortLevel other</code>  | Use "other" extraction for <code>timeDesign/optDesign -postRoute</code> and <code>extractRC</code> .                                                                                                         |

## Example1: Use Model for Track Assignment-Based Timing and SI Optimization

The example below details the use model for TA-Opt:

```
createTQuantusModelFile -file tQuantus_model.bin
setSIMode -reset
setSIMode -analysisType aae
setExtractRCMode -engine postRoute -effortLevel medium -tQuantusModelFile
tQuantus_model.bin
setDelayCalMode -SIAware true
setAnalysisMode -analysisType onChipVariation -cppr both
routeDesign -trackOpt
```

**Note:** When `routeDesign -trackOpt` is specified, the `setDelayCalMode -SIAware` parameter must be set to `true`. If it is set to `false`, the software will error out. Also, you can enable the ideal clock mode for optimization by specifying the `-idealClock` parameter along with the `-trackOpt` parameter.

## Example2: Use Model for optDesign -postRoute and timeDesign – postRoute Optimization

The example below details the use model for `optDesign -postRoute` and `timeDesign -postRoute` optimization:

```
createTQuantusModelFile -file tQuantus_model.bin
setSIMode -reset
setSIMode -analysisType aae
setExtractRCMode -engine postRoute -effortLevel medium -tQuantusForPostRoute true -
tQuantusModelFile tQuantus_model.bin
setDelayCalMode -SIAware true
setAnalysisMode -analysisType onChipVariation -cppr both
optDesign -postRoute/timeDesign -postRoute
```

## Using a Design Flow that Includes Astro or Apollo

The NanoRoute router uses the information in the Milkyway technology file to automatically map vias and layers in the design to Astro scheme format. The router maps only the routing data, so it does not map DEF vias or special routing. To use Astro or Apollo in your design flow, you must have a LEF technology file that contains layer and via descriptions that match one-to-one with the descriptions in the Milkyway technology file.

**Warning: Check with the foundry before making any changes to the original files.**

You can create a rule file to map layers or vias that are not mapped automatically.

In native mode, the following commands are used in this flow:

- [generateLef](#)  
Converts Milkyway technology file descriptions, Milkyway CLF files, customized LEF technology files from customers, and other older syntax and non-optimal LEF files to a LEF file with target rules and automatically generated vias. Issue this command before routing.
- [defOut](#)  
Outputs a routed database based on the mapping results. Issue this command after routing.

If you are running the router standalone mode, run the following command:

```
pdi export_design -lef -rule
```

It finds the technology file and automatically maps the layers and vias from the LEF file to a format that Astro can read. Optionally, includes user-created rules for additional mapping. Run this command after routing.

The following command outputs a file named *MyOutputFile*, using a rule file named tfo.map:

```
pdi export_design -def -rule tfo.map MyOutputFile
```

If you do not need any additional mapping, the only line in tfo.map is

```
techfile apollo.tf
```

## Troubleshooting

If you have problems with your design, try the following troubleshooting tips:

1. Check the log file for errors and warnings. Correct the problems and continue routing or reroute, as appropriate. For example, the router might stop routing automatically if it finds too many violations. If the router stops unexpectedly, check the log file for a message that the router has reached the maximum number of violations and then set `setNanoRouteMode -drouteAutoStop` parameter to false to continue routing.
2. Verify connectivity and geometry before and after routing and compare results. You can also use the `checkRoute` command to verify the connectivity. It is faster than `verifyConnectivity`, but does not output a report.
3. Save the database after routing and restore it in a new session in the Innovus software. Saving and restoring the database clears temporary data structures in memory.
4. Issue the `defOut` command, then `defIn`, and reroute. The `defOut` command saves all routing information in DEF and restores a clean database for routing.

- [Violations in Timing-Driven Routing](#)

## Optimizing Metal Density

- [Overview](#)
- [Before You Begin](#)
- [After You Complete Adding Via and Metal Fills](#)
- [Metal Fill Features](#)
- [Specifying Metal Fill Parameters](#)
- [Recommendations for Adding Timing-Aware Metal Fill](#)
- [Adding Metal Fill Over Macros](#)
- [Recommendations for Power Strapping Mode](#)
- [Adding Via Fill](#)
- [Recommendations for Metal/Via Fill Flow](#)
- [Recommendations for In-design Sign-off Metal Fill Flow](#)
- [Achieving Gradient Density with Preferred Density Setting](#)
- [Specifying Metal Fill Spacing Table](#)
- [Trimming Metal Fill](#)
- [Trimming Metal Fill for Timing Closure](#)
- [Verifying Metal Density](#)
- [Adding Metal Fill Using the GUI](#)
- [Adding Metal Fill with Iteration](#)

## Overview

The dielectric layers in chip designs often vary in thickness due to the different patterns of metal on successive metal layers. These variations reduce yield and impact chip performance. To minimize these, you can add inactive metal segments, called metal fills, to the open areas of the design. The metal fill makes the topology of the metal layers more uniform, which reduces the variations in metal density.

The additional metal increases cross-coupling capacitance, however, so it is important to balance the decrease in thickness variations with the increase in capacitance.

- To simplify the estimation of cross-coupling capacitance added by the metal fill, the software adds the metal fill in a staggered pattern. For more information, see [Metal Fill Features](#).
- To minimize cross-coupling capacitance within layers, the software adds the metal fill in the

timing-aware mode. For more information, see [Recommendations for Adding Timing-Aware Metal Fill](#).

In addition to adding the metal fill to reduce thickness variations in metal layers, the software can also add cuts to meet minimum cut density requirements. The added cuts are modeled as a via fill. For more information, see [Adding Via Fill](#).

The chip manufacturer usually specifies a target metal density percentage for the metal layers and a range of acceptable minimum and maximum metal densities. The metal fill commands help you achieve a metal density within the acceptable range and the via fill commands help you meet the cut density requirements.

In addition, the `verifyMetalDensity` and `verifyCutDensity` commands enable you to verify that the metal density of the metal and cut layers is within the minimum and maximum density values specified by the LEF file or the `setMetalFill` and `setViaFill` commands, respectively.

The software uses parameters specified in the LEF file or the fill commands to analyze the density and determine the size and position of the fill. It divides the design into windows and adds metal or cuts to the open areas in each window until the metal and cut densities meet the density requirements.

You can add the fill to one or more layers at both the chip and block levels.

If you perform additional routing after inserting the fill, you can trim away fill that causes DRC violations.

## Before You Begin

- Complete detailed routing.

To make sure the metal fill is viewable, select the following options in the main window:

- Floorplan or Physical view
- *Special Net* visibility toggle
- *Metal Fill* visibility toggle - To view the *Metal Fill* visibility toggle, click the *All Colors* button in the *Layer Control* window.

For more information on setting object visibility, see [The Main Window](#) chapter in the *Innovus Menu Reference*.

## Adding Metal Fill in the Multiple-CPU Processing Mode

You can add the metal fill to the design in the multi-threading mode by running the following command before adding it:

- [setMultiCpuUsage](#)

For more information on this and other multiple-CPU commands, see the [Multiple-CPU Processing Commands](#) chapter in the *Innovus Text Command Reference*.

Alternatively, fill in the appropriate parameters on the Options - Set Multiple CPU Usage - Multiple CPU Processing form. (You can also access this form by clicking the *Set Multiple CPU* button on the Route -- Metal Fill -- Add -- Add Metal Fill form.)

For more information, see [Accelerating the Design Process By Using Multiple-CPU Processing](#).

## After You Complete Adding Via and Metal Fills

After adding via and metal fills, extract parasitics and run timing and signal-integrity analysis, as needed. The metal fill and verify usage is not normally used as sign-off (although it is possible with a strict methodology). In practice, you will still run a final sign-off script on the full-chip to add any fill inside hard-blocks (LP: hard blocks?). Alternatively, you might run a sign-off script at the hierarchical boundaries or at the die-boundary. In some cases, you may chose to do another extraction with QRC including the extra fills from GDS of the sign-off script.

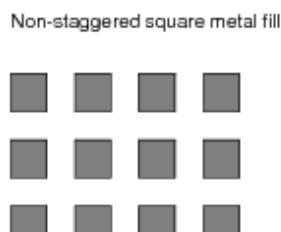
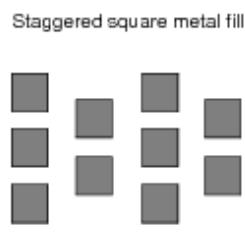
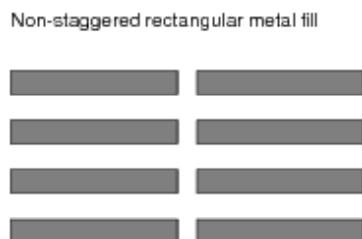
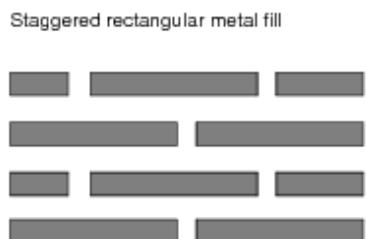
## Metal Fill Features

The metal fill has the following features:

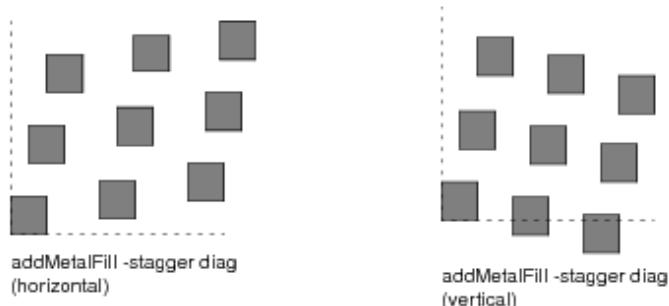
- It can be square or rectangular.
- It can be added in a staggered or non-staggered pattern.
- It can be connected to power or the ground (tied-off) or left unconnected (floating).
- It can be added in timing aware or non-timing aware mode.
- It can be part of the power and ground structure.

## Staggered Metal Fill Pattern

The staggered metal fill spreads out the effects of cross-coupling capacitance because the staggered pattern ensures that the metal fill does not line up on adjacent layers. This pattern is most effective on lightly congested layers. By default, the software adds a metal fill that is staggered in the preferred routing direction and not staggered in the non-preferred direction. The following figures show staggered and non-staggered patterns for both rectangular and square metal fills.



A metal fill that is staggered in both directions can also be added. This type of metal fill has a diagonal pattern. It is most apparent when it is added to the upper layers where there is not a lot of routing. The following figures show a metal fill that is staggered diagonally:



## Connected and Floating Metal Fill

- Metal fill segments can be connected (tied-off) to power or ground shapes on adjacent routing layers or left unconnected (floating). The software creates vias, when it ties off the metal fill, that fit within the area where the metal fill segment overlaps with a power or ground shape on an adjacent routing layer. It does not create vias that are larger than the overlapped area, or "cross-vias," in which the via layer is contained within the same layer as the metal fill segment.

By default, the software creates both connected and floating metal fills. It is difficult to tie off all metal fills, therefore, some shapes are usually left floating. You can minimize the number of floating shapes by including the following parameters when you run the `setMetalFill` command:

-removeFloatingFill  
-*netNameList*

If you remove the floating metal fill, however, it is more difficult to reach the preferred density requirements. In addition, a floating metal fill has the following advantages over a tied-off metal fill:

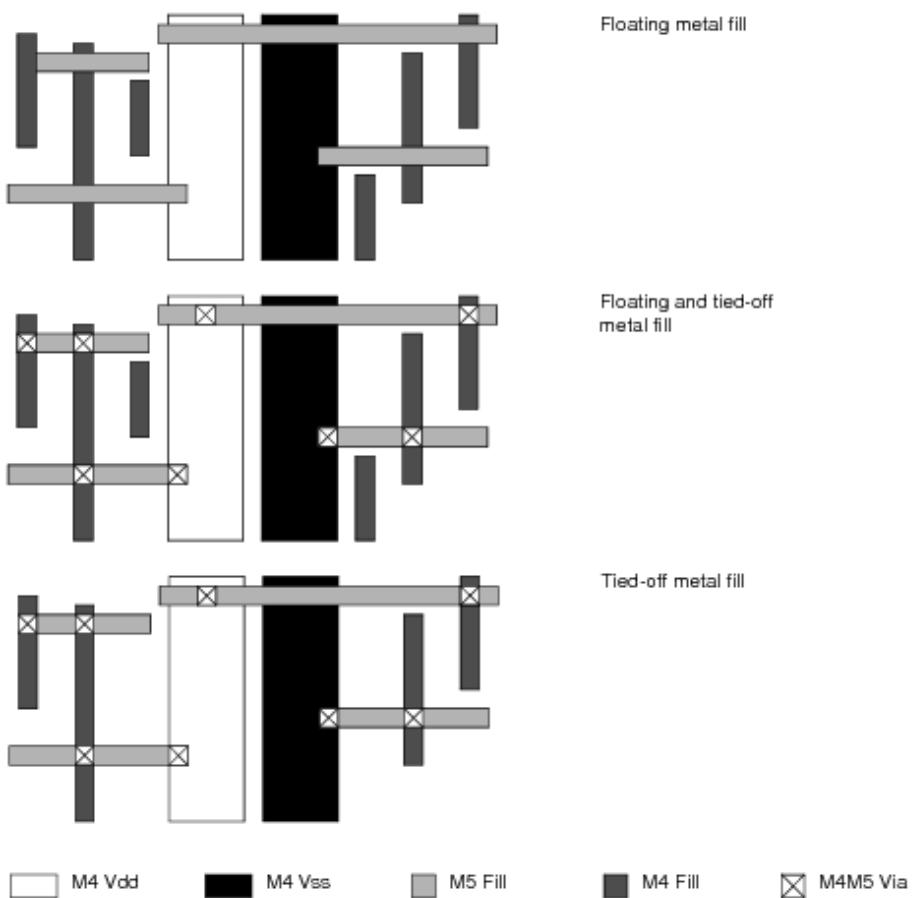
1. Lower cross coupling capacitance, especially if you specify short metal fill segments (long metal fill segments behave like they are really tied off).
2. Easier to trim when there are violations. You can trim a floating metal fill that causes DRC violations with the `trimMetalFill` command. If you add a tied-off metal fill, however, you must either delete it manually to avoid problems with vias or use `fixOpenFill` to address isolated fills.

When a tied-off metal fill is trimmed, the vias cause the following problems:

- If not deleted, they cause shorts to new wires.
- If deleted:
  - An isolated piece of previously tied-off metal fill might be left after trimming.
  - If the new routing was added during an ECO in which some layers were frozen, the change might affect a layer that should have been left frozen.

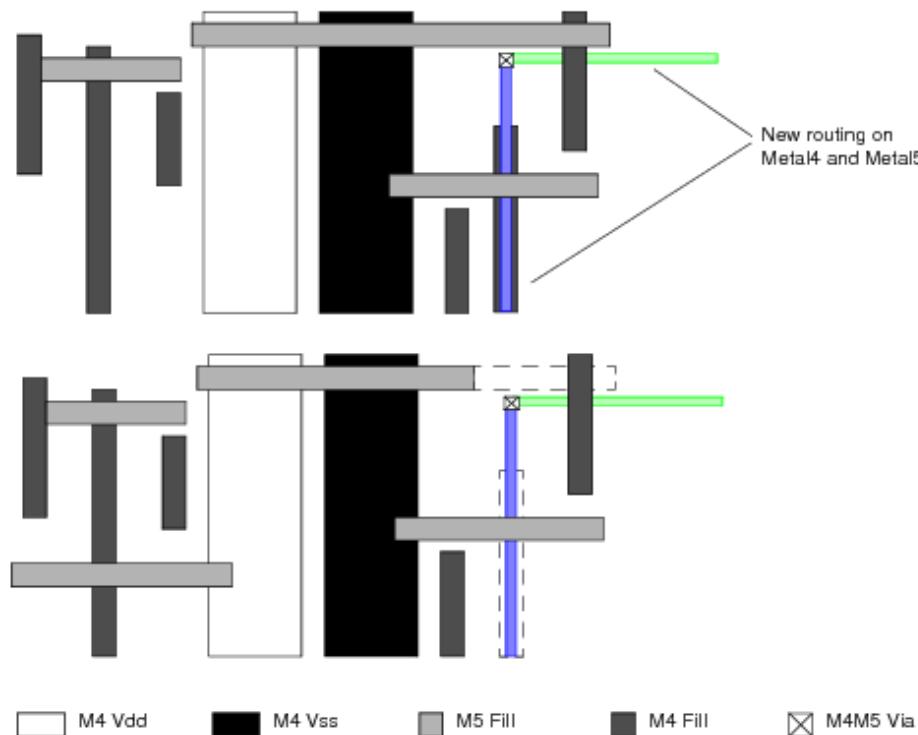
For more information, see the figures below and "[Trimming Metal Fill](#)".

The figures below show a section of a design with a metal fill. In the first figure, the whole metal fill is floating. In the second figure, some of the metal fill is floating and some is tied off. In the third figure, all of the metal fill is tied off.

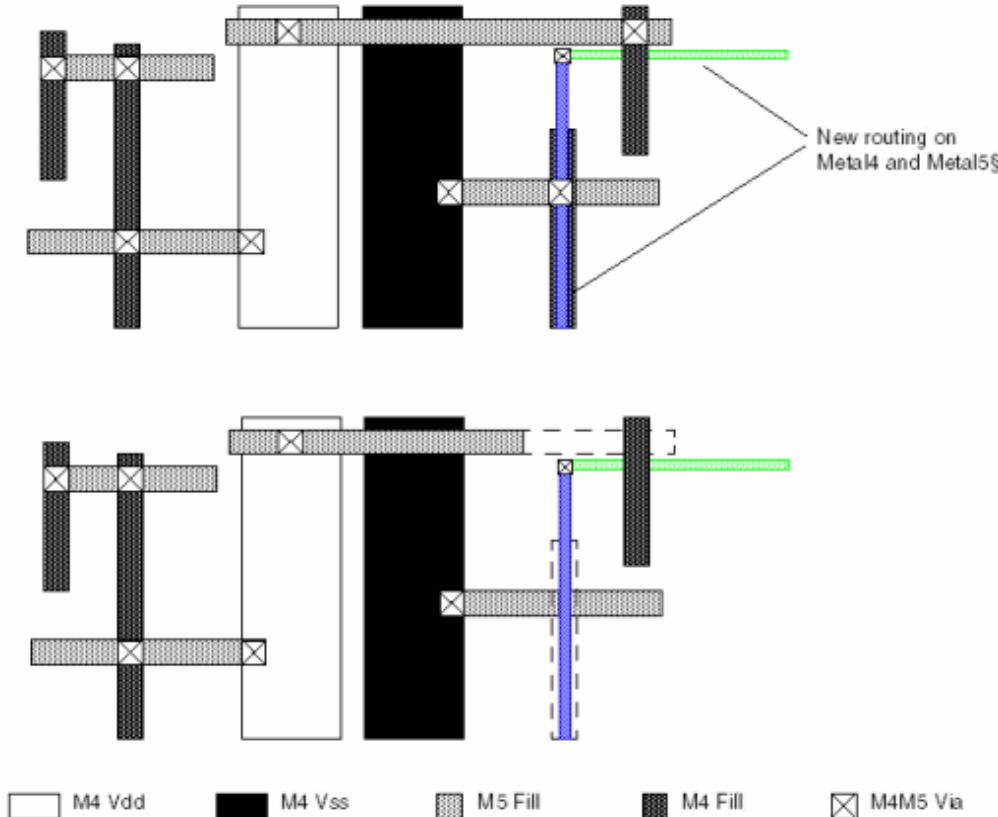


The figures below show the same design after an ECO, in which routing was added on *Metal4* and *Metal5*.

These figures show what happens when you use a floating metal fill. The first figure shows the design with the added routing. The second figure shows the design after the metal fill is trimmed. The dotted lines show where the metal fill was trimmed.



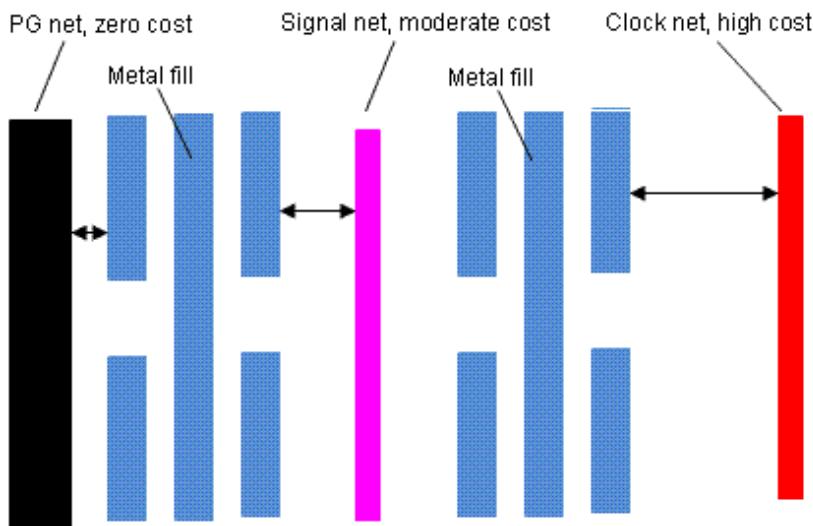
These figures show what happens when you use a tied-off metal fill. The first figure shows the design with the added routing. The second figure shows the design after the metal fill is trimmed. The dotted lines show where the metal fill was trimmed.



## Timing-Aware Metal Fill

When it adds a timing-aware metal fill, the Innovus software avoids adding the fill near clock and signal nets and adds more near the power and ground nets.

The software assigns a high cost to adding a metal fill near clock nets, a moderate cost to adding it near signal nets, and zero cost to adding it near power and ground nets. It adds the fill, based on the cost, to achieve the preferred metal density with the least effect on timing.



The software adds timing-aware metal fill by default.

- To add a non-timing aware metal fill, type the following command:  
`addMetalFill -timingAware off`
- To use the Innovus common timing engine (CTE) for static timing analysis (STA), type the following command:  
`addMetalfill -timingAware sta -slackThreshold value`

If the `buildTimingGraph` command was run already, the software adjusts the costs as a function of the slack (nets with the worst slack have the highest cost). For more information, see [Timing Analysis](#).

When you run the software in the STA mode, it assigns costs to four categories of nets:

- Clock nets are assigned the highest cost.
- Signal nets that have a slack value less than the threshold are assigned a moderate cost. You can use `-slackThreshold` and `-extraCriticalNet` to choose critical signal nets.
- Non-critical signal nets are assigned a small cost.
- Power and ground nets (nets marked + USE POWER or + USE GROUND in the DEF file) are assigned zero cost.

## Specifying Metal Fill Parameters

Some of the metal fill parameters can be specified in the Layer (Routing) section of the LEF file. All the parameters can be specified by the Innovus metal fill commands or forms. The parameters that can be specified in the LEF file are listed in the table below.

If a parameter is specified in the LEF file, use the specified value. If a parameter is not specified, check the chip manufacturer's DRC manual for the correct metal fill (or dummy fill) values and specify them manually with the command or form.

**!** *If not specified properly, a metal fill can cause DRC violations and increase capacitance unnecessarily. Parameters specified by the metal fill commands override parameters specified in the LEF file only if they are more restrictive than the LEF parameters.*

The following table lists the metal fill parameters that can be specified in the LEF file and the corresponding Innovus metal fill parameters:

| Description                                                                                                      | LEF Statements     | setMetalFill Parameter | Setup Metal Fill Parameter |
|------------------------------------------------------------------------------------------------------------------|--------------------|------------------------|----------------------------|
| Minimum distance between a segment of metal fill and another type of object in the design, such as a signal wire | FILLACTIVESPACING  | -activeSpacing         | Active Spacing             |
| Minimum density allowed in the design                                                                            | MINIMUMDENSITY     | -minDensity            | Metal Density % Min        |
| Maximum density allowed in the design                                                                            | MAXIMUMDENSITY     | -maxDensity            | Metal Density % Max        |
| Area the Innovus software uses to examine metal density                                                          | DENSITYCHECKWINDOW | -windowSize            | Step Size X<br>Step Size Y |

|                                                         |                  |             |                                              |
|---------------------------------------------------------|------------------|-------------|----------------------------------------------|
| Distance the window moves for each metal fill iteration | DENSITYCHECKSTEP | -windowStep | <i>Window Size X</i><br><i>Window Size Y</i> |
|---------------------------------------------------------|------------------|-------------|----------------------------------------------|

The Innovus software maintains the values specified for these parameters until you reset them or you restart the software.

For more information on LEF, see the [LEF/DEF Language Reference](#).

## Recommendations for Adding Timing-Aware Metal Fill

Follow the following recommendations for metal fill parameters that specify the preferred density, metal fill shape, the space between the metal fill segments, and whether to use a metal fill that is connected to special wiring. These parameters are not specified in the LEF file.

- Specify a preferred metal density between 25 percent and 40 percent.

Metal density within this range minimizes the density variation in design windows as well as the impact on added capacitance. The reduced variation improves correlation with early RC estimates, that is, it gives you faster timing convergence, and increases yield.

Determining the appropriate metal density is a process of balancing the decrease in density variation with the increase in capacitance: A density of 35 percent minimizes variation and increases the capacitance by a moderate amount; a density of 25 percent adds less capacitance but does not decrease the variation quite as much.

- Insert rectangular metal fill segments rather than square metal fill segments.

You can achieve the preferred metal density with fewer pieces of a rectangular metal fill than with a square metal fill. Adding rectangular segments reduces the number of flashes on the reticle, minimizes the density variation across the design windows, and approaches the preferred metal density in more windows.

The following dimensions for rectangular metal fill segments work with most 28 nm and 45 nm process rules:

- Length: 0.1 um to 10 um
  - Width: Use the width specified in the chip manufacturer's DRC manual for the minimum value. Use two to three times that value for the maximum width.
- For example, you can specify the following dimensions:

- 0.1 um to 1.0 um for thin layers

- 0.2 um to 2.0 um for thick layers

Alternatively, for lower capacitance at the expense of more density variation, reduce the maximum width to the same value as the minimum width.

- Follow the chip manufacturer's DRC manual for the spacing between metal fill shapes. This is called the gap spacing. The gap spacing is generally one to three times the minimum metal fill width. The following dimensions for gap spacing work with most 28 nm and 45 nm process rules:

- 0.1 um for thin layers
- 0.2 um for thick layers

Alternatively, for lower capacitance at the expense of more density variation, use values like 0.8 um for thin layers and 1.6 um for thick layers.

- Add metal fill to all metal layers or run the [verifyMetalDensity](#) command to determine where metal fill is needed.
- Use metal fill that is not connected to special wiring.  
Unconnected (floating) metal fill adds less capacitance to the design and is easier for postroute and postmask changes to handle than connected (tied-off) metal fill.

Alternatively, you can use tied-off metal fill whenever possible and floating metal fill when tied-off metal fill cannot be created. Either method is more likely to meet the preferred-metal density requirements than using tied-off metal fill throughout the design.

## Timing-Aware Examples

The following examples specify conservative values for a 28 nm or 45 nm eight-layer design where metal layers 1 through 6 are thin metal and metal layers 7 and 8 are thick metal.

The following command sets values for the active spacing, window size, window step, minimum density, and maximum density for all eight layers:

```
setMetalFill -layer "1 2 3 4 5 6 7 8" -activeSpacing 0.4 \
    -windowSize 100 -windowStep 50 \
    -minDensity 15 -maxDensity 85
```

The following command sets values for the gap spacing, preferred density, minimum and maximum width, and minimum and maximum length for the thin-metal layers:

```
setMetalFill -layer "1 2 3 4 5 6"
    -preferredDensity 35 -gapSpacing 0.2 \
    -minWidth 0.1 -maxWidth 1.0 \
    -minLength 0.1 -maxLength 10.0
```

The following command sets values for the gap spacing, preferred density, minimum and maximum width, and minimum and maximum length for the thick-metal layers:

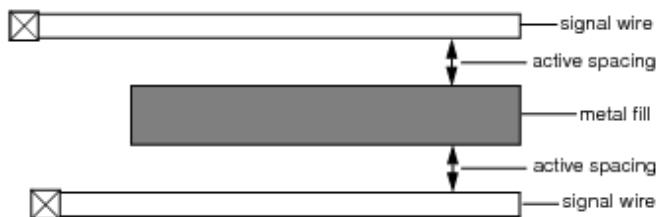
```
setMetalFill -layer "7 8"  
-preferredDensity 35 -gapSpacing 0.4 \  
-minWidth 0.2 -maxWidth 2.0  
-minLength .2 -maxLength 20.0
```

The following command adds metal fill to all eight layers:

```
addMetalFill -layer "1 2 3 4 5 6 7 8" -timingAware sta"
```

## Specifying the Active Spacing Value

The space between metal fill and nonmetal-fill geometries is called the *active spacing*, as shown in the following figure.



The Innovus software uses the `FILLACTIVEspacing` value in the LEF file for the active spacing. If `FILLACTIVEspacing` is not specified, you can set it manually by using one of the following methods:

- Specifying a value for `setMetalFill -activeSpacing` on the text command line

**Note:** The `setMetalFill -activeSpacing` settings are used for creating regular `FILLWIRE` shapes. For `FILLWIREOPC` shapes, design rules are used.

- Specifying a value for *Active Spacing* on the Setup Metal Fill form

If no value is specified in the LEF file, and you do not specify one manually, the software uses 0.6 microns for thin layers (less than 0.24 microns) and 0.8 microns for thick layers as the default active spacing value.

The default active spacing value is usually large enough that you can avoid using Optimal Proximity Correction (OPC) for the metal fill shapes. In addition, the default active spacing minimizes the increase in cross-coupling capacitance caused by the metal fill, which in turn reduces the additional timing delay.

As you increase the active spacing value, you reduce the space available for metal fill. A large increase—for example, 1 um to 2 um for a 28 nm or 45 nm process—might prevent you from meeting the minimum density rule for some windows.

## Adding Metal Fill Over Macros

For adding metal fill to macros, the added metal fill is based on the recalculated metal density for that metal layer. If the added fill is less than the preferred metal density (the default is 35 percent), the software tries to add metal fill shapes on that metal layer to meet the preferred density goal. Otherwise, it does not add any metal fill shapes.

For better metal density accuracy, use the LEF `DENSITY` table--a list of rectangles with metal density numbers. These rectangles cover the entire macro bounding box for that metal layer. The rectangles are defined in the `MACRO` section of the LEF file and are honored by the `addMetalFill` and `verifyMetalDensity` commands. To force `addMetalFill` to place correct metal fill shapes for a layer, place obstructions on the layer to block areas where metal fill should not be placed.

The `DENSITY` rectangles on a layer should not overlap and should cover the entire area of the macro. Choose the size of the rectangles based on the uniformity of the density of the block. If the density is uniform, a single rectangle can be used. If the density is not uniform, the size of the rectangles can be specified to be 10 to 20 percent of the density window size, so that any error due to non-uniform density inside each rectangle area is small.

For example, if the metal density rule is for a 100um x 100um window, the density rectangles can be 10um x10um squares. Any non-uniformity will have little impact on the density calculation accuracy.

If two adjacent rectangles have the same or similar density, they can be merged into one larger rectangle, with one average density value. The choice between accuracy and abstraction is left to the abstract generator.

The `DENSITY` table syntax is:

```
[DENSITY
  {LAYER layerName ;
   {RECT x1 y1 x2 y2 densityValue;} ...
  } ...
END] ...
```

For more information on LEF MACRO DENSITY, see the [Macro](#) section of the "LEF Syntax" chapter in the *LEF/DEF Language Reference*.

**Note:** If you want to ignore the MACRO `DENSITY` table in the LEF file, you can specify the `-ignoreLEFDensity` parameter with the `addMetalFill` and `verifyMetalDensity` commands. When `-ignoreLEFDensity` is specified, the `addMetalFill` and `verifyMetalDensity` commands use the default macro density calculation method for considering the density.

## Estimating Density of Blockage

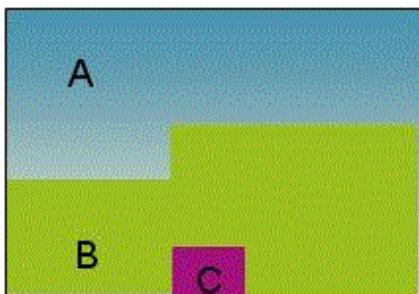
The attribute of a routing blockage declares its purpose; and the purpose, in turn, determines how the density of blockage is interpreted.

A routing blockage can have the following attributes:

- FILL - A blockage on the specified layer to block metal fill insertion. A FILL blockage is created to hold white place where metal fill cannot be inserted. The density of a FILL blockage is calculated as 0, but the other objects under a FILL blockage are accounted with real density.
- PUSHDOWN - A routing blockage with no SPACING or DESIGNRULEWIDTH (DRW) that is pushed down with +PUSHDOWN. This blockage requires wide-wire spacing as PUSHDOWN shapes are treated as real shapes. The density of the blockage is calculated as 100% because it is considered as a real shape.
- PUSHDOWN with SPACING/DRW - By default, the possibility of wide wires added to a top-level blockage area is avoided. If you want less space for a routing blockage inside the sub-block, you can manually attach a DRW value to the top-level routing blockage. The density of the blockage is calculated as preferred density, which is defined in LEF or specified with `setMetalFill` command. It is assumed that there is no wide wire under a PUSHDOWN with SPACING/DRW blockage.
- Blockage without attribute - A blockage without attribute is used to block insertion of any shapes in areas that are used by top level routing or are booked for later steps. The density of the blockage is calculated as preferred density, which is defined in LEF or specified with `setMetalFill` command because it is supposed that the area is used by routing wires.

## Estimating Density of BLOCK Cell

If no macro density table is defined in LEF, the macro density is estimated as follows. Consider an instance represented by the outlined rectangle below:



In the instance, A is empty area which is defined by overlap OBS, B is the OBS area, and C is instance

pin area. The density of these area is calculated as A=0, B=Preferred density, C=100%.

Based on the above assumption, the following formula is used to estimate the macro density.

- If `verifyMetalDesity Area` ratio > 0.5, A and B are considered as preferred density, C is considered as 100%.
- If `verifyMetalDesity Area` ratio <= 0.5, A and B are considered as 0, C is considered as 100%.
- If OBS is outside of overlap area or macro boundary, the density of OBS is considered as 0.
- If overlap is outside of macro boundary, the density of overlap is considered as 0.
- If no OBS/PIN/OVERLAP in the macro area, the density is considered as 0.

If OBS or PIN is defined on some layers, it means the layers are used by macro. By default, the macro area should not be inserted with metal fill without `-onCell` option. With `-onCell` option, the metal fill can be inserted in macro area but the metal fill should not touch the OBS/PIN shapes.

If no OBS or PIN is defined on a layer, the layer is not used by macro. By default, the macro area should be inserted with metal fill without `-onCell` option.

## Recommendations for Power Strapping Mode

In power strapping mode, the software makes mesh connections to power and ground bus wiring, instead of the tree-type connections used in regular connected mode. This configuration allows the metal fill shapes to carry current as part of the power and ground structure. Power strapping uses the maximum possible number of cuts in vias, based on the intersection area between layers, instead of using the minimum-cut based connections used in regular connected mode.

To get the best results in power strapping mode, follow these recommendations:

- Use longer maximum lengths (at least 100 um).  
Longer lengths increase the number of times a single metal fill shape intersects with the existing power/ground mesh.
- Use higher values for preferred density (40 percent to 50 percent).  
Higher preferred density increases the number of metal fill segments retained as candidates for power strapping.
- Use wider metal fill

## Adding Via Fill

When it adds via fill, the Innovus software does the following:

- Attempts to add vias that meet cut density requirements
  - Uses metal width and spacing values specified by the `setMetalFill` command (or Set Metal Fill form) to determine size and allowed placement locations
  - Adds either tied-off or floating vias until the preferred cut density is met
    - In tied-off vias, either the top or bottom layer is connected to power or ground.
    - Floating vias are not connected to power or ground.
- Floating vias could be inserted between floating metal fills or in white space.

**i** To get the best results from via fill, add it before adding metal fill. You can minimize the need for via fill by inserting multiple-cut vias with the NanoRoute router prior to adding via fill. For information, see `setNanoRouteMode`.

Use the fill commands in the following recommended order:

1. `setViaFill`
2. `setMetalFill`
3. `addViaFill`
4. `addMetalFill`

**Note:** You can use the `verifyMetalDensity` and `verifyCutDensity` commands to verify that the metal density of the metal layers and cut layers is within the minimum and maximum density values specified by the `setMetalFill` and `setViaFill` commands, respectively.

## Recommendations for Metal/Via Fill Flow

In the recommended flow, the software adds via fill to free space prior to adding other metal fill shapes. It does not connect via fill to metal fill.

Use the fill commands in the following order:

1. Set via and metal layer parameters.

```
setViaFill -layer "Via23" -windowSize 50 50 -windowStep 25 25 -minDensity 0.005 -
```

```
maxDensity 30 ...
```

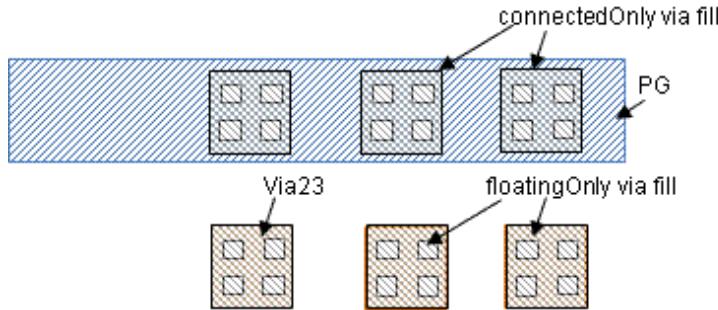
```
setMetalFill -layer "Metal2 Metal3" -activeSpacing 0.4 -gapSpacing 0.1 -maxWidth 0.1 -  
maxLength 10 -windowSize 50 50 -windowStep 25 25 -minDensity 15 -maxDensity 85 ...
```

You also can specify the parameters in the GUI using the *Setup Metal Fill Options* and *Setup Via Fill Options* forms. The `addViaFill` and `addMetalFill` commands will honor the setting to add via and metal fill.

## 2. Add via fill with specified options.

```
addViaFill -layer "Via23" -mode floatingOnly -area "0 0 100 100"
```

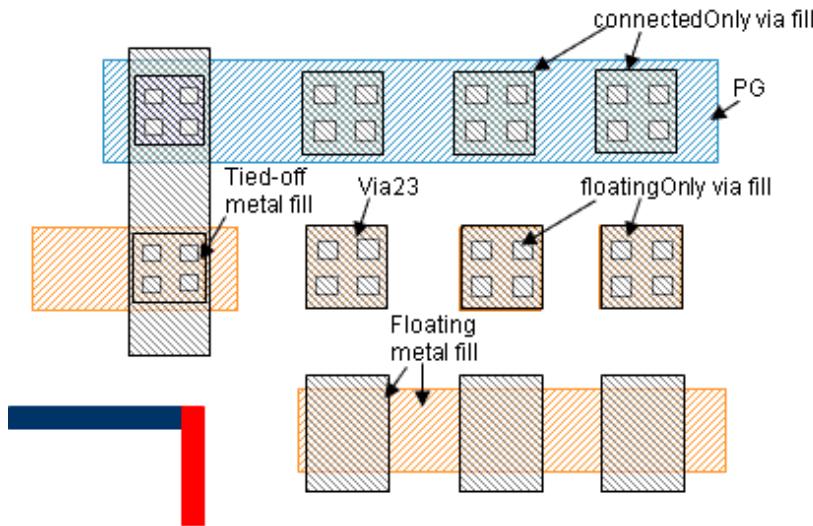
This will add via fill in white space to meet the via density requirements according to the specified rules. Via fill can be connected to power or ground nets (tied off) or unconnected (floating). Via fill cannot be connected to signal nets.



## 3. Add metal fill with specified options.

```
addMetalFill -layer {Metal2 Metal3} -area 0 0 100 100 -stagger on -timingAware sta -  
onCells -net {VDD VSS} -mesh
```

This will insert inactive metal into white space to achieve the metal density that is required by a specific manufacturing process. However, the inactive metals do not touch any other metal fill.



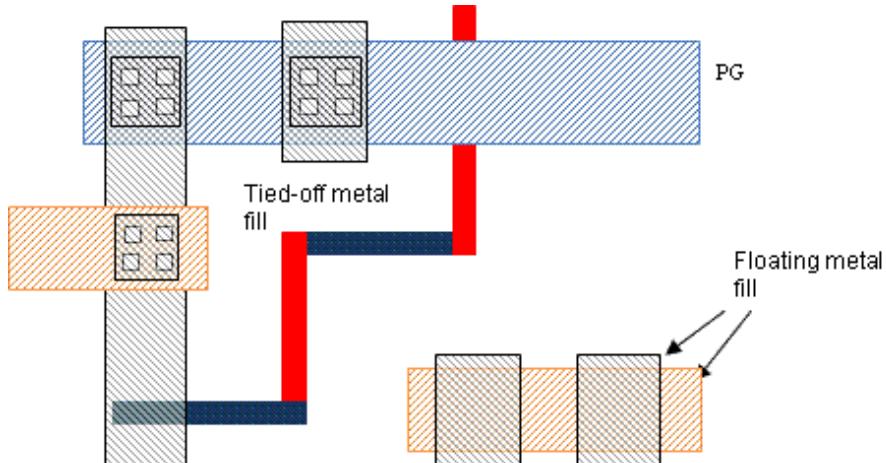
Innovus now provides an alternative flow in which the software adds metal fill to free space prior to adding via fill shapes. It can connect metal fill with special via fill. In this alternative flow, you:

1. Set metal layer parameters.

```
setMetalFill -layer "Metal2 Metal3" -activeSpacing 0.4 -gapSpacing 0.1 -maxWidth 2.0 -
maxLength 10 -decrement 2 -diagOffset 0.4 0.4 -windowSize 50 50 -windowStep 25 25 -
minDensity 15 -maxDensity 85 ...
```

2. Add metal fill with specified options.

```
addMetalFill -layer {Metal2 Metal3} -excludeVia Dvia -net {VDD VSS} -area 0 0 100 100
-stagger on -timingAware sta -onCell -mesh
```



This step inserts inactive metal into a placed and routed design to achieve the required metal

density according to the specified parameters. The software attempts to connect metal fill to the first net in the list, then the next net, and so on. However, the Dvia should not be used to connect to special nets. If the metal fill cannot connect to special nets, keep them floating.

3. Set via layer parameters.

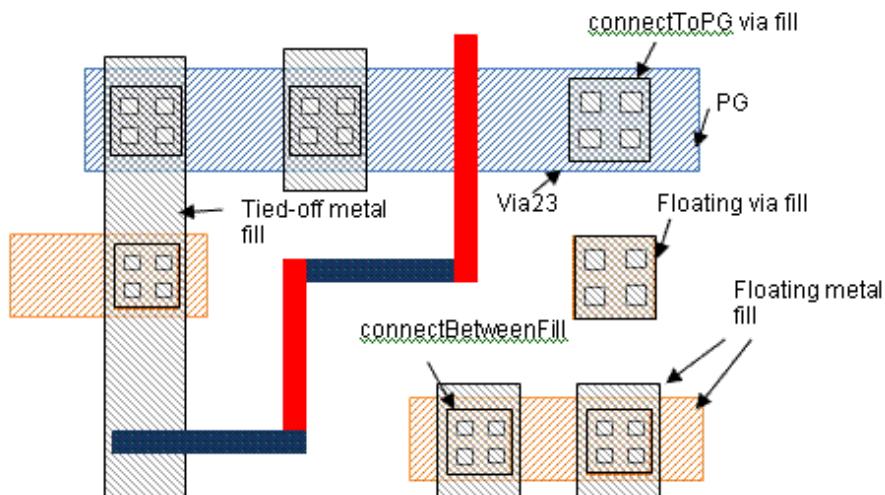
```
setViaFill -layer "Via23" -windowSize 50 50 -windowStep 25 25 -minDensity 0.005 -  
maxDensity 30 ...
```

The parameters honor settings in the following order:

- a. setViaFill
- b. setMetalFill
- c. LEF
- d. Manufacture process default

4. Add via fill with the specified options.

```
addViaFill -layer "Via23" -area "2 4 6 8" -mode {floating connectToPG  
connectBetweenFill} -includeVia Dvia
```



This will connect the floating metal fill with the special Dvia to meet the via density requirements.

## Recommendations for In-design Sign-off Metal Fill Flow

In the recommended flow, the software calls the Physical Verification System (PVS) engine to insert sign-off metal fill in design. The metal fill near critical nets can be trimmed for timing closure.

Use the fill commands in the following order:

1. Insert sign-off metal fill in design.

Before inserting sign-off metal fill, stream out a GDSII stream file of the current database. Specify the mapping file and units that match with the rule deck you specify while inserting metal fill. If necessary, include the detailed-cell Graphic Database System (GDS).

**Note:** Prepare the mapping file to align with the rule deck layer definition. Use the same unit as the rule deck.

```
streamOut -mapFile $gds_map -outputMacros -units $gds_unit pvs.fill.gds
```

`run_pvs_metal_fill` calls the PVS engine to insert metal fill and then dump the metal fill in Innovus. The DEF mapping file is required to ensure that the metal fill is put in the correct layers. The top cell name is also required.

```
run_pvs_metal_fill -ruleFile $MF_RULE_DECK -defMapFile $def_out -gdsFile pvs.fill.gds  
-cell $top_cell
```

2. Analyze the timing impact by metal fill.

After the metal fill is inserted, run `timeDesign` to check the timing.

```
timeDesign -postRoute -outDir postfill
```

If the timing result is acceptable, the metal fill step is complete.

3. Trim metal fill for timing closure.

If the inserted metal fill impacts timing, use `trimMetalFillNearNet` to trim the metal fill near nets for timing closure.

You can trim the same and upper/bottom layer metal in timing aware mode. Use `-layer` to specify the layers on which metal can be trimmed. The spacing between metal fill and critical nets can be specified with the three spacing options. You can specify different spacing for different slack net with multi-pass trimming.

You can specify the critical net with the `-net` option. You can also specify the slack threshold. If you do so, the tool decides the critical net list with the slack threshold.

If the minimum metal density needs to be controlled, specify the window size and step. The metal density can then be calculated with window setting.

a. Set metal fill parameters

```
setMetalFill -minDensity 10 -maxDensity 85 -preferredDensity 35 -windowStep 62.5  
62.5 -windowSize 125 125
```

If the minimum density target is specified, trimming stops as soon as the minimum density is achieved. If you do not specify a minimum density target, metal fill is trimmed with the specified spacing.

b. Trim the metal fill near more critical nets with bigger spacing.

```
trimMetalFillNearNet -createFillBlockage -slackThreshold $slack1 -spacing 1.0 -  
spacingAbove 1.0 -spacingBelow 1.0 -minTrimDensity $min_density
```

c. Trim the metal fill near less critical nets with comparatively smaller spacing.

```
trimMetalFillNearNet -createFillBlockage -slackThreshold 0.0 -spacing 0.4 -  
spacingAbove 0.4 -spacingBelow 0.4 -minTrimDensity $min_density
```

4. Analyze the timing impact after trimming metal fill.

```
timeDesign -postRoute -outDir posttrimfill
```

If the timing result is still not acceptable, repeat Step 3 until timing closure.

**Note:** Innovus metal fill does not support 20nm and below node design rules. We strongly recommend the PVS metal fill solution for 20nm and below. If you have sign-off metal fill rule deck for 28nm and above available, we recommend you to move to PVS solution too.

**!** *trimMetalFillNearNet does not check DRC rules. It only removes the metal fill with specified spacing. Do not perform ECO operations after dump in sign-off metal fill, especially, at 20nm and below nodes. The sign-off metal fill typically does not cause DRC issues with regular wires. If you perform an ECO action, the tool cannot get DRC clean because trimMetalFill and verifyGeometry do not support 20nm and below node design rules.*

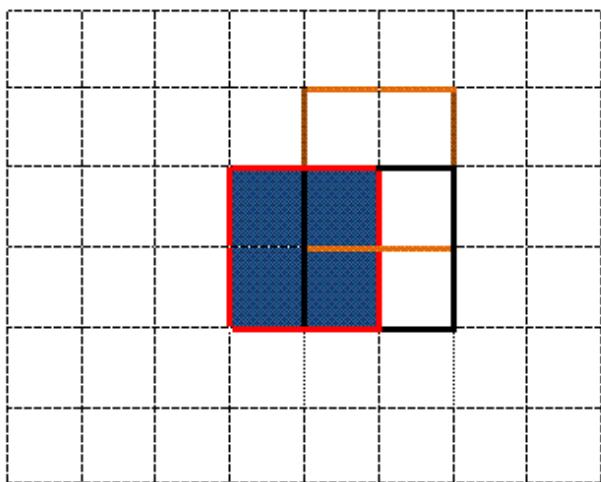
## Achieving Gradient Density with Preferred Density Setting

To prevent density in neighboring regions from varying too much, the `addMetalFill` targets a preferred density. This minimizes the variation in density from window to window. You can set the parameters as follows:

```
-minDensity 15 -maxDensity 85 -preferredDensity 35  
addMetalFill -layer {Metal1 Metal2 Metal3}
```

The metal fills are inserted into white space to meet the preferred density. When the metal density in a window is less than the minimum metal fill density value, `addMetalFill` adds metal fill to achieve a density slightly above the preferred density, if possible. If the density is larger than maximum density after it pre-calculates the window density, no metal fills are inserted into the design. The metal fills are inserted based on the preferred density in all windows. This way, the density variation from window to window is minimized.

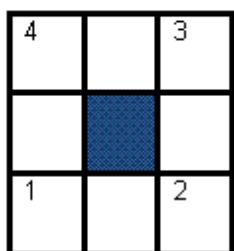
The `windowStep` parameter can be used to get further global uniformity. With this parameter, the metal densities in the window are calculated and changed by step as shown in the diagram.



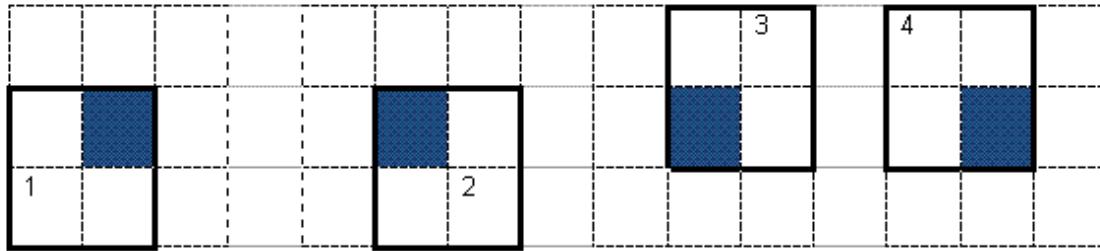
Window\_1   
 Window\_2   
 Window\_3

When add metal commands are applied, the engine calculates the Window\_1 density and tries to insert metal fill until the window density reaches the preferred density target. When Window\_1 is finished, the engine moves to the next window. The window step is specified in `setMetalFill` command. Note that half of Window\_2 overlaps with Window\_1. This means when Window\_2 density is calculated, half of Window\_1 is considered in Window\_2. In other words, Window\_1 and Window\_2 have mutual influence on each other. After Window\_2 is finished, the engine moves to Window\_3. Window\_3 has half part overlapping with Window\_2 and one-fourth part overlapping with Window\_1. Metal fill is inserted in the remaining windows using a similar method.

For each  $25 \times 25$  window step, the window density is cross-locked with the adjacent window steps (labeled 1, 2, 3, and 4 in the diagram).



The engine calculates the window densities (1, 2, 3, 4 - Size 100\*100) and tries to insert metal fill in it. The window step (size 50\*50) is considered in it respectively. This way, sudden changes in density between adjacent windows are smoothed out.



## Specifying Metal Fill Spacing Table

During an Engineering Change Order (ECO) on a verified database with metal fill, NanoRoute routing sometimes creates shorts and spacing violations. To resolve these violations, you would need to further add and trim metal fill. However, as the original database was already verified for metal fill density and timing, you would want the changes to the database to be small and local with as less impact to the original database and existing metal fill as possible.

In such cases, you can use the `setMetalFillSpacingTable` command to specify the spacing table based on existing metal fill width. The spacing table includes the following spacing values:

- Fill to active spacing
- Fill to fill spacing
- Fill to OPC spacing
- OPC to OPC spacing
- OPC to active spacing

As `trimMetalFill` honors the spacing table, you can then run `trimMetalFill` to trim metal fill and get expected spacing.

**Note:** If no spacing table is specified, `trimMetalFill` honors the `setMetalFill` settings for `-activeSpacing` and `-gapSpacing`. If `setMetalFill` values are also not specified, `trimMetalFill` uses the default settings.

Here are some examples that demonstrate how to specify spacing table for trimming metal fill:

- Fill to active spacing table

The following set of commands specify the spacing table for fill to active spacing (as given below) and trim metal fill accordingly:

| Layer | Minwidth (=>) | Maxwidth (<=) | activeSpacing (>=) |
|-------|---------------|---------------|--------------------|
| 2     | 0.32          | $\infty$      | 0.63               |
| 2     | 0.14          | 0.32 - 1MFG   | 0.36               |
| 2     | 0.08          | 0.14 - 1MFG   | 0.27               |

```
setMetalFillSpacingTable -layer {2} -fill_to_active {{0.08 0.27}{0.14 0.36}{0.32 0.63}}
trimMetalFill -layer 2
```

The software checks the existing metal fill with the specified spacing table. If no violations are detected, existing metal fill is retained. If the existing metal fills have activeSpacing violations, trimMetalFill uses the fill\_to\_active spacing table to trim metal fill.

Sample output is as follows:

```
*** START TRIM METALFILL *** (CPU Time: 0:00:00.0 MEM: 558.359M)
Number of metal fills with spacing or/and short violations: 4920
Total number of deleted metal fills: 4920
Total number of added metal fills: 4779
(CPU Time: 0:00:01.2 MEM: 558.359M)

*** END OF TRIM METALFILL ***
```

- Fill to fill spacing table

The following set of commands specify the spacing table for fill to fill spacing (as given below) and trim metal fill accordingly:

| Layer | Minwidth (=>) | Maxwidth (<=) | gapSpacing (>=) |
|-------|---------------|---------------|-----------------|
| 3     | 0.32          | $\infty$      | 0.6             |

|   |      |           |     |
|---|------|-----------|-----|
| 3 | 0.14 | 0.32-1MFG | 0.3 |
| 3 | 0.08 | 0.14-1MFG | 0.2 |

```
setMetalFillSpacingTable -layer {3} -fill_to_fill {{0.08 0.2}{0.14 0.3}{0.32 0.6}}
```

```
trimMetalFill -layer 3
```

The software checks the existing metal fill with the specified spacing table. If no violations are detected, existing metal fill is retained. If the existing metal fills have gapSpacing violations, trimMetalFill uses the fill\_to\_fill spacing table to trim metal fill.

Sample output is as follows:

```
*** START TRIM METALFILL *** (CPU Time: 0:00:00.0 MEM: 559.441M)
```

```
Number of metal fills with spacing or/and short violations:6119
```

```
Total number of deleted metal fills: 6119
```

```
Total number of added metal fills: 6022
```

```
(CPU Time: 0:00:01.0 MEM: 560.602M)
```

```
*** END OF TRIM METALFILL ***
```

- OPC to active spacing table

The following set of commands specify the spacing table for OPC to active spacing (as given below) and trim metal fill accordingly:

| Layer | Minwidth (=>) | Maxwidth (<=) | opcActiveSpacing (>=) |
|-------|---------------|---------------|-----------------------|
| 3     | 0.1           | $\infty$      | 0.12                  |
| 3     | 0.08          | 0.1-1MFG      | 0.1                   |
|       | 0.05          | 0.08-1MFG     | 0.08                  |

```
setMetalFillSpacingTable -layer {3} -opc_to_active {{0.05 0.08}{0.08 0.1}{0.1 0.12}}
```

```
trimMetalFill -layer 3
```

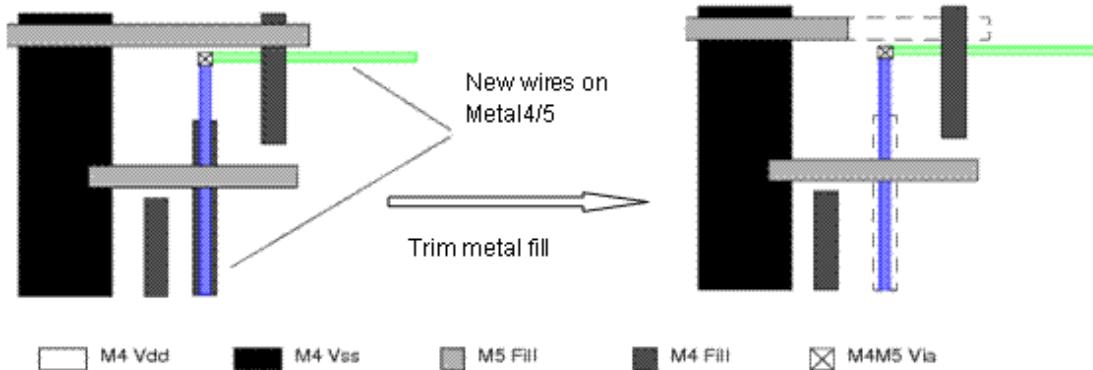
When required, you can use the `getMetalFillSpacingTable` command to print specified or all spacing tables used by `trimMetalFill`.

## Trimming Metal Fill

The automatic routers, including the NanoRoute® router, ignore metal fill (`FILLWIRE` and `FILLWIREOPC`) shapes and might create routes that cause shorts or DRC violations.

The following case illustrates the DRC violation after NanoRoute ECO. You can use `trimMetalFill` to clean the violations according to user setting, LEF setting, and default parameters.

```
trimMetalFill -deleteViol
```



This command deletes metal fill shapes that cause DRC violations or shorts. After running the `trimMetalFill` command, the remaining shapes are still rectangles.

This means you need not delete the metal fill before ECO and then add it again after ECO. Instead, you can trim metal fill in the window that has been impacted by ECO. `trimMetalFill` can minimize the impact caused by the ECO on the timing of other paths (due to cross-coupling changes) that were not involved in the ECO.

To remove the shorts and violations, complete the following steps:

- To remove floating metal fill that causes shorts or violations, run the following command:  
`trimMetalFill [-deleteViols] [-ignoreSpecialNets]`

This command repairs violations caused by the metal fill shapes. If the metal density drops below the target after trimming the metal fill, re-run the `addMetalFill` command.

The `trimMetalFill` command trims metal and via fill shapes based on the following spacing

rules:

- Between `FILLWIRE` and `FILLWIREOPC` shapes, the active spacing value or minimum spacing based on DRC rules, whichever is larger, is required.
- Between `FILLWIRE` shapes, the gap spacing value or minimum spacing, whichever is larger, is required.
- Between `FILLWIREOPC` and active shapes, the OPC active spacing value or minimum spacing, whichever is larger, is required.
- Between `FILLWIREOPC` shapes, minimum spacing is required.

For more information, see [trimMetalFill](#).

- To specify the layers that you want to trim to fix DRC violations, use the `-layer` parameter of the `trimMetalFill` command. For example, to trim metal fill in METAL2 and METAL3 layers, use the following command:

```
trimMetalFill -layer {METAL2 METAL3}
```

**Note:** This option is recommended for use with only floating metal fill. If you use `trimMetalFill -layer` on tied-off fill shapes, some of the shapes may become isolated from the Power/Ground network.

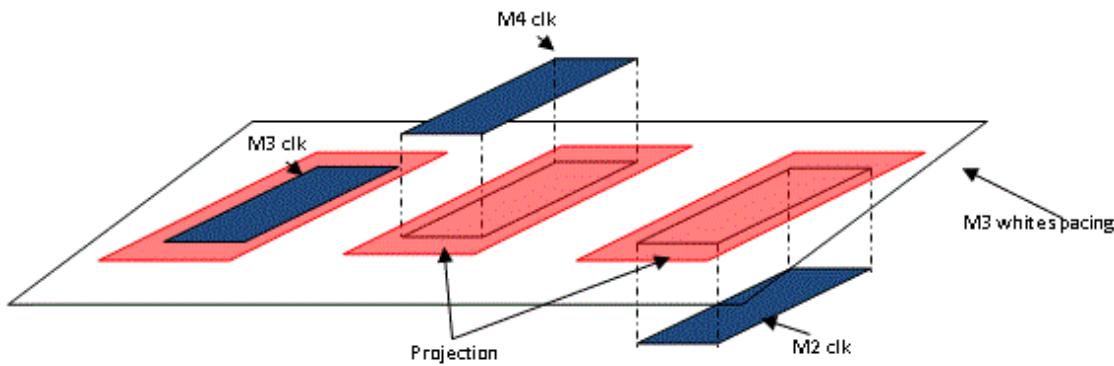
- To limit the area that in which metal fill is trimmed, use the `-area` parameter of the `trimMetalFill` command. This option is recommended for use with only floating metal fill. If you use `trimMetalFill -area` on tied-off fill shapes, some of the shapes may become isolated from the Power/Ground network.
- To remove connected metal fill, complete the following steps:
  - a. Trim metal fill.
  - b. Fix isolated fill issues with [fixOpenFill](#). You can choose to either change isolated PG fills to floating fill or remove isolate fills (`fixOpenFill -remove`). If you choose to remove the isolated fills, you can then add metal fill incrementally to see if any of those islands can be tied either to the same or another PG rail.

For information on `FILLWIRE` and `FILLWIREOPC`, see [Shape](#) in the "DEF Syntax" chapter of the *LEF/DEF Language Reference*. (`FILLWIREOPC` is not supported by LEF 5.6.)

## Trimming Metal Fill for Timing Closure

The metal fill (`FILLWIRE` and `FILLWIREOPC`) shapes potentially impact the timing if the metal fills are close to the critical nets. Although Innovus provides timing-aware metal fill solution, you could also use post-fill trimming to improve the timing result. If you load third-party metal fill in Innovus with `defIn`, you could rely on the post-fill trimming for timing closure.

In the following diagram, if you insert metal fill in the red area to meet the metal density requirements, it may impact timing. Use `trimMetalFillNearNet` to trim the metal fill if the timing impact is high.



To specify critical nets:

- Specify the net list with `-net`.
- Use `-clock` to specify that metal fill should be trimmed around all clock nets.
- Use `-slackThreshold` to specify the slack threshold. All the nets whose slack value is less than the specified threshold are identified as critical nets.

To specify the spacing for trimming:

- Use `-spacing` to specify the distance to be kept around critical nets in the same layer when trimming fill.
- Use `-spacingAbove` to specify the distance to be kept around critical nets for the top layer.
- Use `-spacingBelow` to specify the distance to be kept around critical nets for the bottom layer.

To prevent incremental metal fill close to the critical nets:

- Use `-createFillBlockage` to create fill blockage after trimming metal fill. The fill blockage prevent metal fill in incremental steps.

To prevent minimum density issues when trimming:

- Set density parameters before running the `trimMetalFillNearNet` command.  
Use the `-minTrimDensity` parameter to specify the minimum density value. Innovus calculates the metal density while trimming metal fill. If the metal density is less than the minimum density, trimming stops.

The following example illustrates how to use the `trimMetalFillNearNet` command.

1. Set metal fill parameters.

```
setMetalFill -minDensity 10 -maxDensity 85 -preferredDensity 35 -windowStep 62.5 62.5 -  
windowSize 125 125
```

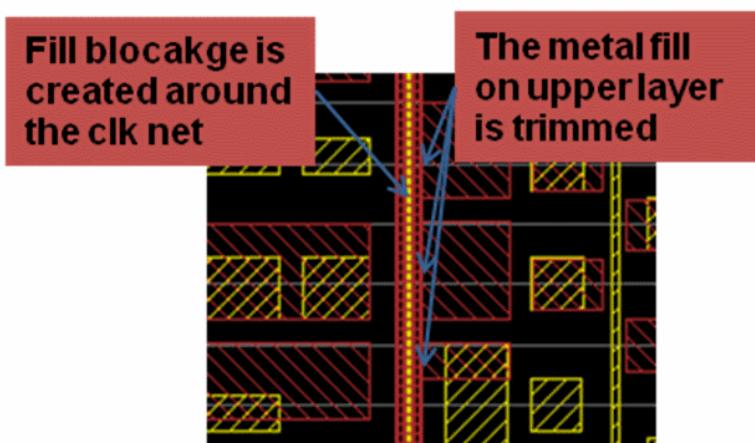
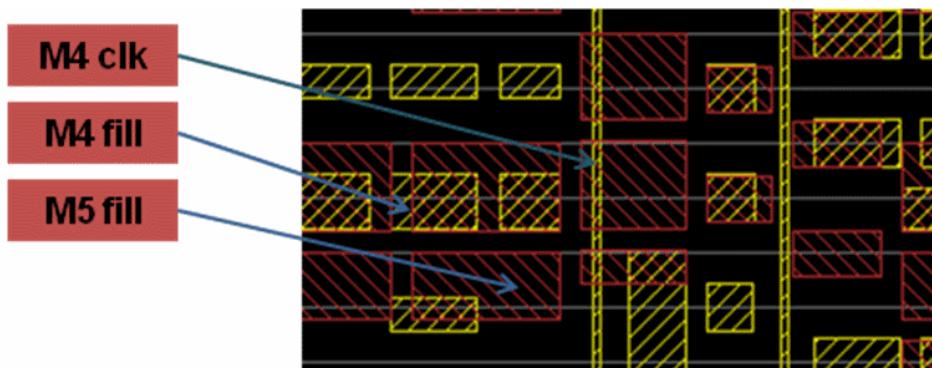
2. Trim metal fill with larger spacing near more critical nets.

```
trimMetalFillNearNet -createFillBlockage -slackThreshold $slack1 -spacing 1.0 -  
spacingAbove 1.0 -spacingBelow 1.0 -minTrimDensity $min_density
```

3. Trim metal fill with comparatively smaller spacing near less critical nets

```
trimMetalFillNearNet -createFillBlockage -slackThreshold 0.0 -spacing 0.4 -  
spacingAbove 0.4 -spacingBelow 0.4 -minTrimDensity $min_density
```

The diagram below shows that the upper layer metal fill is trimmed.



## Verifying Metal Density

After adding or trimming metal fill, use the Verify Metal Density and Verify Geometry features to verify that the metal fill has been added correctly.

Ensure that the minimum, preferred, and maximum density values and window size and step are defined in the `default` iteration name. `verifyMetalDensity` uses the `setMetalFill` settings from the `default` iteration name. The `default` iteration name settings are the settings used when `setMetalFill` is run either without the `-iterationName` parameter or with `-iterationName default`. If these settings are not available, `verifyMetalDensity` uses the LEF settings. If the LEF settings are not available, `verifyMetalDensity` uses the internal default values for verifying density.

For more information, see the "Verify Commands" chapter of the *Innovus Text Command Reference*.

## Adding Metal Fill Using the GUI

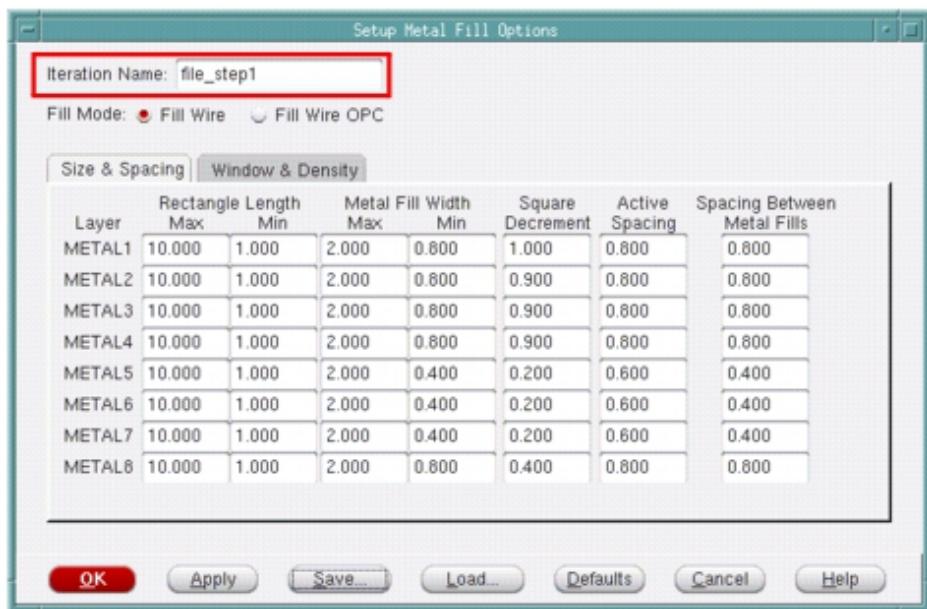
1. Determine the minimum and maximum size for metal fill shapes for each layer, then set these values on the *Size & Spacing* page of the Setup Metal Fill form.
  - If you are using rectangular metal fill, use the *Rectangle Length* and *Metal Fill Width* values.
  - If you are using square metal fill, use the *Metal Fill Width* and *Square Decrement* values.
3. Determine the spacing around metal fill shapes for each layer, then set the value on the *Size & Spacing* page of the Setup Metal Fill form. You must set two types of spacing values:
  - Spacing between a metal fill shape and an active metal shape. An active metal shape can be a signal wire, a power wire, a cell, a pin, or any other structure that is not classified as a fillwire.
  - Spacing between a metal fill shape and another metal fill shape.
5. Determine the minimum, maximum, preferred, and external metal density for each layer, then set these values on the *Window & Density* page of the Setup Metal Fill form.
6. Use the Verify Metal Density form to create a Verify Density report.
7. Locate an area in the design for which metal density is too low, then select that area on the Add Metal Fill form.
8. Determine whether you want metal fill to be square or rectangular, then choose the appropriate value on the Add Metal Fill form.
9. Click *OK* or *Apply* on the Add Metal Fill form to add metal fill shapes to the area that you specified.

## Adding Metal Fill with Iteration

Metal fill can be added iteratively with different parameter settings. You can specify a name for a set of values for `setMetalFill` parameters.

```
setMetalFill -iterationName file_step1 -layer Metall -minDensity 15 -windowSize 100 100 -  
windowStep 50 50
```

You can also save the iteration file using GUI. To do so, open the *Setup Metal Fill Options* form, specify the parameters in the form, key in a file name, such as `file_step1`, in the *Iteration Name* text box, and click *OK*.



The window size and step must be the same for all iterations of a specific layer. For example, the following specifications are NOT allowed because the values are not consistent:

```
setMetalFill -iterationName file_step1 -layer Metal1 -minDensity 15 -windowSize 100 100
>windowStep 50 50

setMetalFill -iterationName file_step2 -layer Metal1 -minDensity 15 -windowSize 50 50 -
>windowStep 25 25

setMetalFill -iterationName file_step1 file_step2 -layer Metal1
```

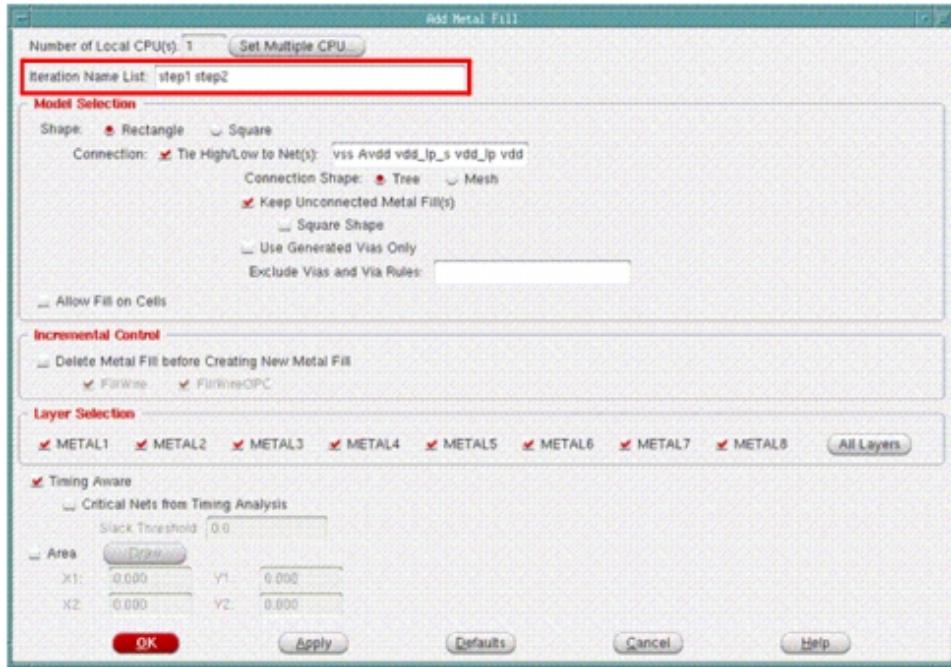
If you want to specify different window size and step when adding metal fill, you need to run `addMetalFill` in separate steps. In the following example, the specified values for `-windowSize` and `-windowStep` in `step1`, `step2`, and `step3` are different:

```
setMetalFill -iterationName step1 -layer -windowSize 100 100 -windowStep 50 50
setMetalFill -iterationName step2 -layer -windowSize 100 100 -windowStep 50 50
setMetalFill -iterationName step3 -layer -windowSize 50 50 -windowStep 25 25
```

Here, you can run `addMetalFill` for the first two steps in a single iteration. However, you must run `step3` in a separate iteration because its window size and step values are different from those of `step1` and `step2`. Use `addMetalFill -iterationNameList` to add the metal fill using the stored set of parameters:

```
addMetalFill -iterationNameList {step1 step2} ...
addMetalFill -iterationNameList step3 ...
addMetalFill -layer {Metal1 Metal2 Metal3} -area 0 0 100 100 -nets {VDD VSS} -
iterationName step1 step2
```

You can also do the same through the GUI by using the *Route - Metal Fill - Add* command.



Key in the existing file list in *Iteration Name List* text box in the *Add Metal Fill* form and then click *OK*.

The engine processes the iterations in the order listed and stops when the preferred density is reached in any iteration.

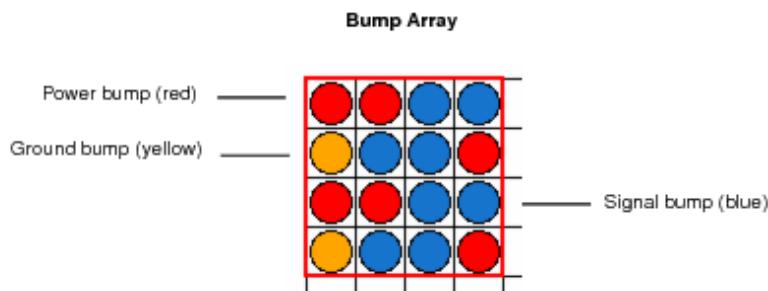
## Flip Chip Methodologies

- Overview
  - Related Packaging Tools
  - Before You Begin
  - Using This Chapter
  - Related Flip Chip Information
- Flip Chip Flow in Innovus
- Introduction to Flip Chip Methodology
  - SiP Bump Flow
  - Area I/O (AIO) Flow
  - Peripheral I/O (PIO) Flow
  - Flow Methodologies

- [Data Preparation](#)
  - [LEF](#)
  - [NETLIST](#)
- [Flip Chip Floorplanning](#)
  - [Bump Creation and Assignment](#)
  - [Bump Assignment Optimization](#)
- [Viewing Flip Chip Flightlines](#)
  - [Automatic Redraw Feature](#)
  - [Selection-Based Highlighting](#)
  - [Colored Flightlines](#)
  - [Object-Specific Flightlines](#)
  - [DIFFPAIR-Based Highlighting](#)
  - [viewBumpConnection Display Rules](#)
  - [Long Pin Connection Display](#)
- [Power Planning in Flip Chip Design](#)
- [RDL Routing](#)
  - [Introduction](#)
  - [Useful Constraints for Flip Chip Routing](#)
  - [Useful Extra Configurations for Flip Chip Routing](#)
  - [Power Routing](#)
  - [ECO Routing](#)
  - [P2P Router](#)
  - [Handling Flip Chip Designs with Complex Floorplans](#)
  - [Flip Chip Router Report](#)
- [Advanced Flip Chip Features](#)
  - [Two-Layer RDL Routing](#)
  - [Routing Bumps in the eWLB Process](#)
  - [Pillar Bump Support](#)
  - [fcroute Bus Routing for DDR3](#)
- [RDL Extraction](#)
- [SI and Timing Analysis](#)

## Overview

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



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

## Related Packaging Tools

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

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

## Before You Begin

Before using flip chip, the following information is required:

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

## Using This Chapter

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

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

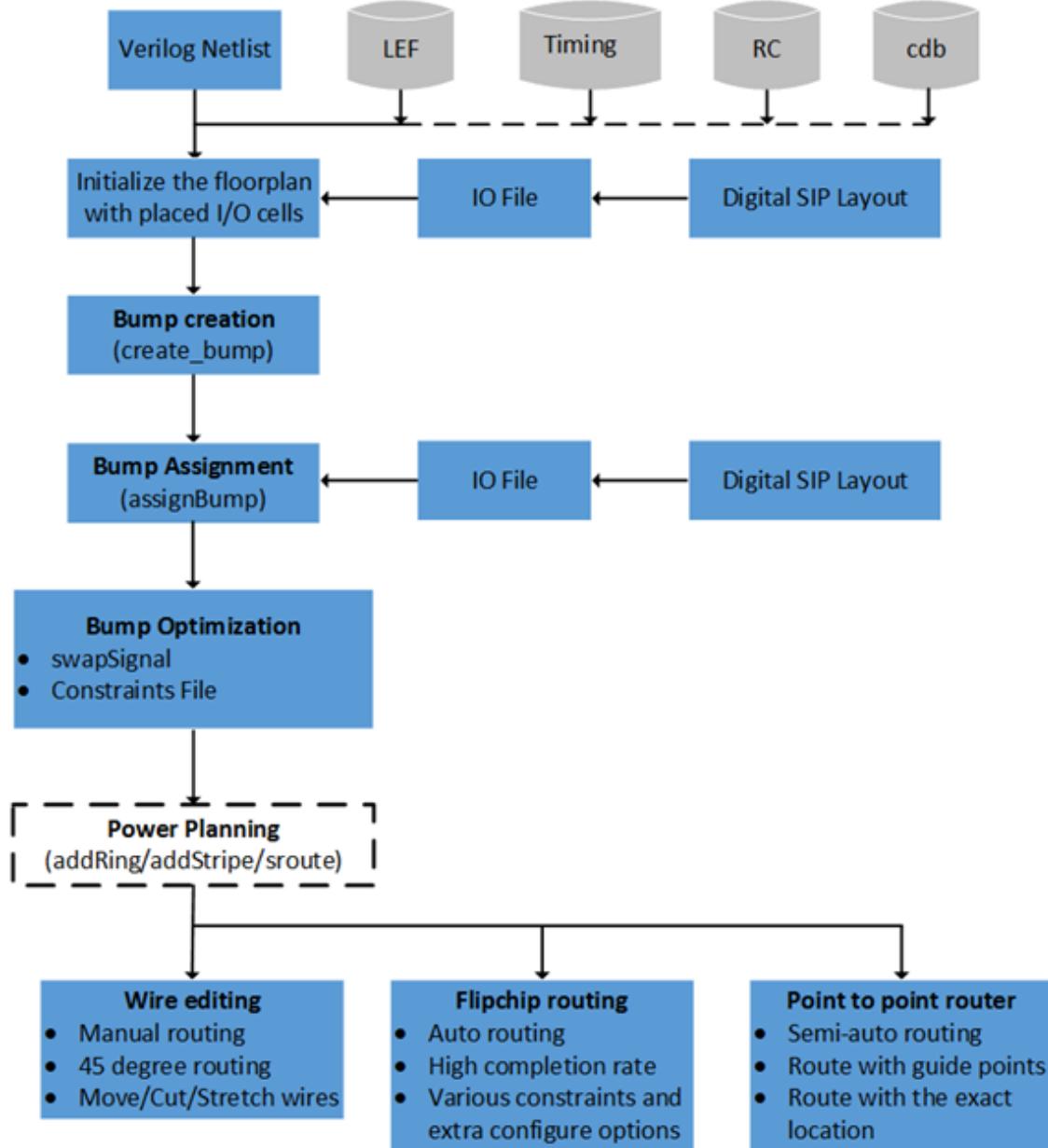
In addition to the above, Innovus also supports a mix of Peripheral IO and Area IO.

## Related Flip Chip Information

- Text commands  
For information on the flip chip commands, see the [Flip Chip Commands](#) chapter of the *Innovus Text Command Reference*.
- Flip Chip Toolbox Menu  
For information on the flip chip forms, see the [Flip Chip](#) section of the *Tools Menu* chapter in the *Innovus Menu Reference*.

# Flip Chip Flow in Innovus

The following figure shows the general Innovus flip chip flow.



# Introduction to Flip Chip Methodology

This section describes the various implementation and flow methodologies for flip chip.

Flip Chip supports the following implementation methodologies:

- SiP Bump Flow
- Area I/O Flow
- Peripheral I/O Flow

## SiP Bump Flow

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

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

You can use the `-noCoreCells` option of the `defOut` command to reduce data size for import into SiP.

The syntax is as follows:

```
defOut -noCoreCells
```

This flow bypasses the bump flow (see [Flip Chip Flow in Innovus](#)).

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

## Testing the Package Routing Feasibility

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

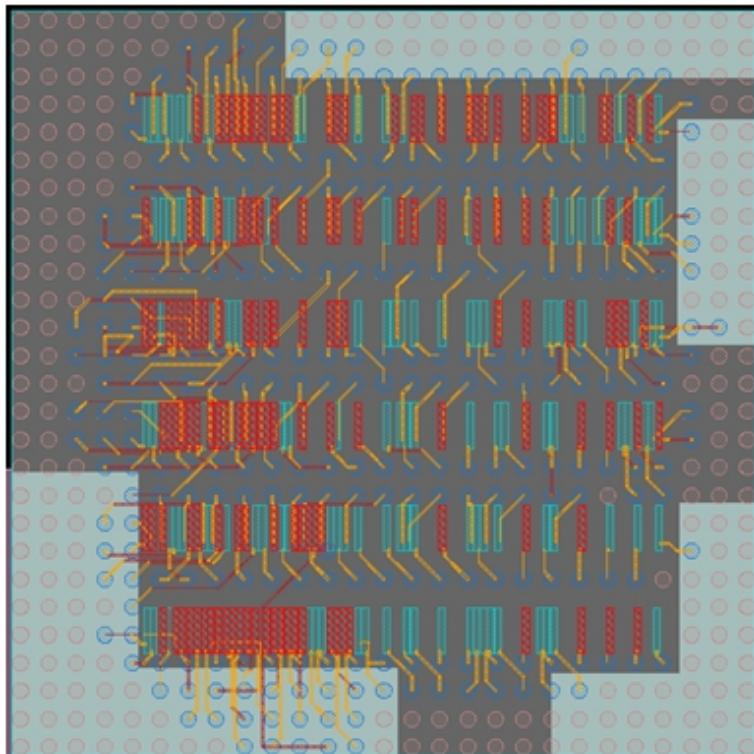
For more information, see the *Cadence Chip I/O Planner User Guide* or the *SiP Digital Architect/Layout User Guide* on SourceLink.

## Area I/O (AIO) Flow

For Area I/O (AIO) designs, the IO PADs are placed in the core area. You can define IO pad row clusters in the core, where these IO pads will be placed, and from where they will be routed. With this implementation, routing is much less constrained. As a result, routing congestion issues arise rarely. The bumps can be defined and placed close to the IO pads shortening the net length.

The disadvantage of this methodology is that the IO pad placement affects the standard cell placement and therefore the full timing closure flow. Power routing is also more demanding in this implementation as dedicated power stripes must be routed to feed the power requirement for the IOs.

In general, this implementation style is more complex but offers much less net delay from IO pad to the bump. In addition, the SI effect is greatly reduced as the net length is much shorter.



To create a flip chip design, complete the following steps:

1. Load the floorplan with I/O pad placement.
2. Define the bumps.

Use the `create_bump` command or the Create Bump Array form to set up the bump array.

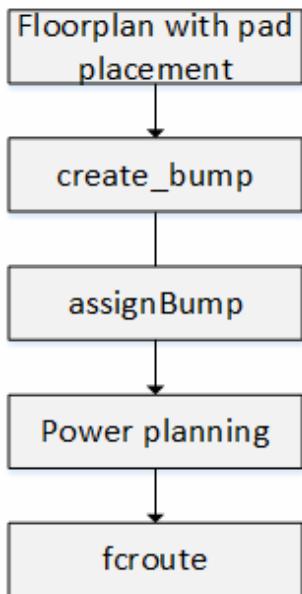
3. Assign signals, power, and ground to the bumps.

- Use the *Bump Assignment* tab of the Flip Chip form to assign the signals, power and ground to bumps. Signal bumps are blue-filled squares. Power bumps are red-filled squares. Ground bumps are yellow-filled squares.
4. Create power rings and stripes.
    - Use the [Add Rings](#) form to create rings around the core area and around the power and ground bumps.
    - Use the [Add Stripes](#) form to create stripes that connect to the power and ground bumps.
  5. Connect power, from bumps to I/O cells or from bumps to rings/stripes.
    - Use the *RDL Routing* tab of the Flip Chip form to establish the power connections.

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

## AIO Command Flow

The AIO command flow can be depicted as follows:



## Routing Bumps to I/O Driver Cells (Hierarchical AIO Flow)

The hierarchical AIO flow allows you to route the bumps, using the `fcroute` command, to I/O driver cells and then push down (partition) this data into a lower level.

The text command is:

```
handlePtnAreaIo buffer_name
```

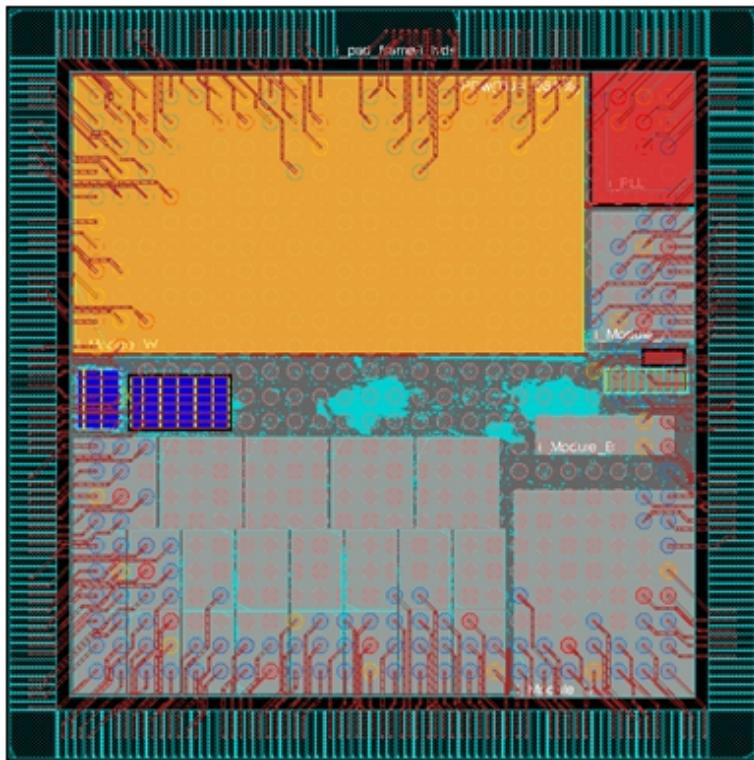
This command pushes down data in the partition as follows:

- Bumps become routing blockages
- I/O cells become placement and routing blockages
- An internal pin is created over the I/O cell pin
- A boundary pin is created
- A buffer is created between the internal pin and the boundary pin

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

## Peripheral I/O (PIO) Flow

IO PADs are placed on the periphery of the die. If the design is severely pad limited, it could have multiple IO rings around the core area. The pin of the pad is routed to the bumps using the redistribution layer (RDL). These bumps are located in the core area of the chip. The diagram below depicts a typical peripheral design.



The recommendation is to floorplan the design, including the bump location, assignment, and RDL routing. The benefit is that you can take the advantage of having more freedom when moving IO pads and bumps. Moving IO pads that are not related to analog blocks is feasible at this stage. Bump movement needs to be verified for routing purposes in SiP. SiP verifies that there is no routing congestion between bumps and package balls using the current bump assignment.

It is not recommended to implement the flip chip features as a post process after design closure. During full chip implementation, you could choose not to use the RDL layer for signal or power routing and reserve this layer for the flip chip router. In this case, the bump assignment and routing can be performed in parallel to the implementation flow or as a post step after final timing signoff. This methodology restricts the movement of the IO pads as they are fixed after implementing the design closure flow. In addition, the package tool (SiP) can limit the bump location. In this case, it is common to face routing congestion and hotspots.

A PIO design has no impact on the default timing/area/power/DFM-DFY closure implementation flow.

## PIO Flow Steps

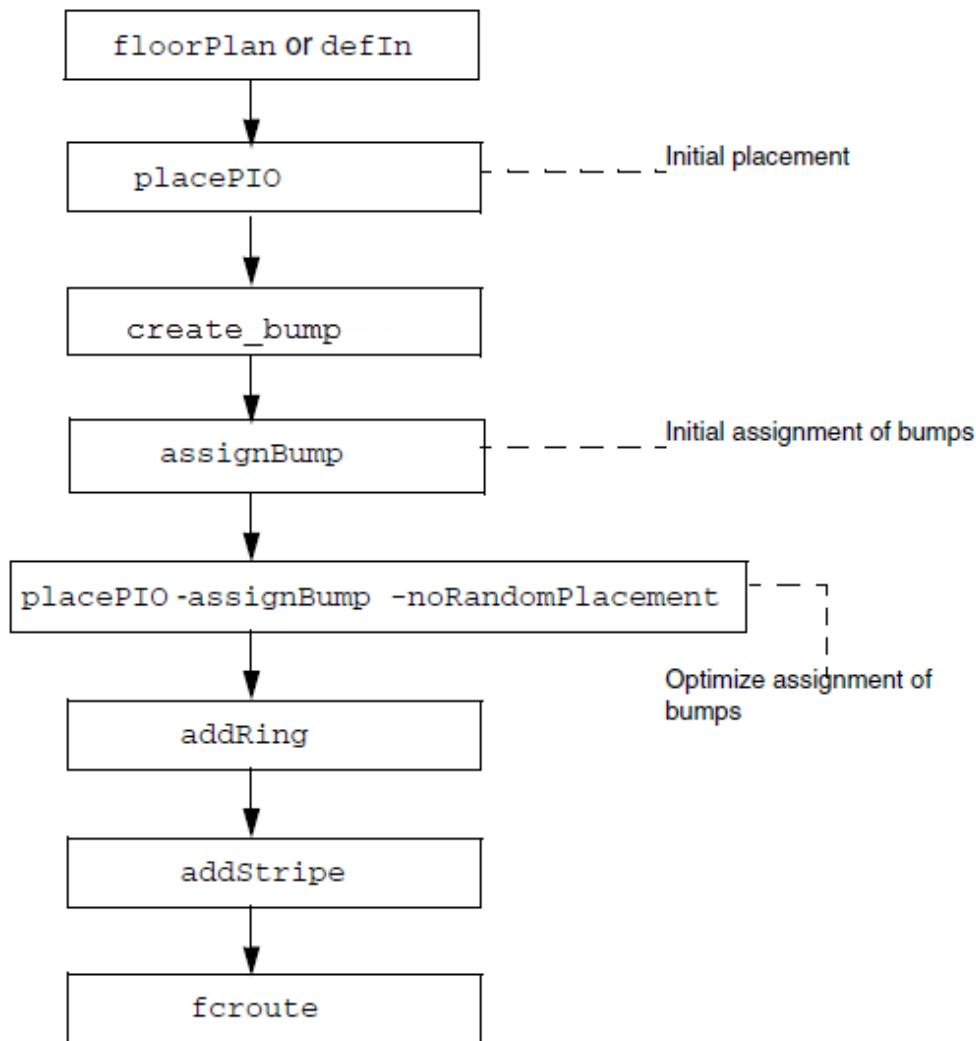
The PIO implementation flow is similar to the traditional physical implementation flow, except for the handling of bump cells and RDL routing.

There are four major elements of the flow:

- After the initial floorplanning stage (set die and area and place I/O driver cells), the RDL implementation flow includes bump placement and assignment, optimization of I/O driver cell placement, and RDL routing.
- The bump placement and assignment is passed to APD (Allegro® Package Designer) for package design. You can determine the route feasibility by using APD to check the bump routability to the package. This can be invoked from the Innovus user interface.
- The RDL-routed design is then ready for power planning/Quantus/other placement and routing operations.
- Initial parasitics can be extracted in Innovus using the `extractRC` command. If more accurate parasitics are required, the signal-routed design can be streamed out in GDSII format and sent to Assura™ RCX for extracting RC parasitics, which can be used for timing and SI analysis with the RDL effects.

## PIO Command Flow

The PIO command flow can be depicted as follows:



## Flow Methodologies

The design of an IC and its package/carrier has traditionally been two separate development processes done in succession (Serial Design Flow), driven from a common specification.

### Innovus-driven Floorplanning

The digital implementation engineer has an initial and rough floorplan to start with. For example:

- The size of the chip might be given as a constraint from marketing or from the customer.
- The location and ordering of some or all the I/Os may already be known. This information may come from the PCB or package designer, or there may be some inherent placement requirements

imposed by an analog block in the core.

All this information can be fed into Innovus during the implementation. In this case, Innovus can drive the flip chip implementation by placing these PADs and then creating and assigning the bumps. After this implementation, the design needs to go to SiP Layout (Cadence's package implementation tool) to verify that the bump placement can be routed to the package balls. This is becoming very critical as users are trying to reduce the size of the package which is limiting the routing resources.

## **Package-driven Floorplanning**

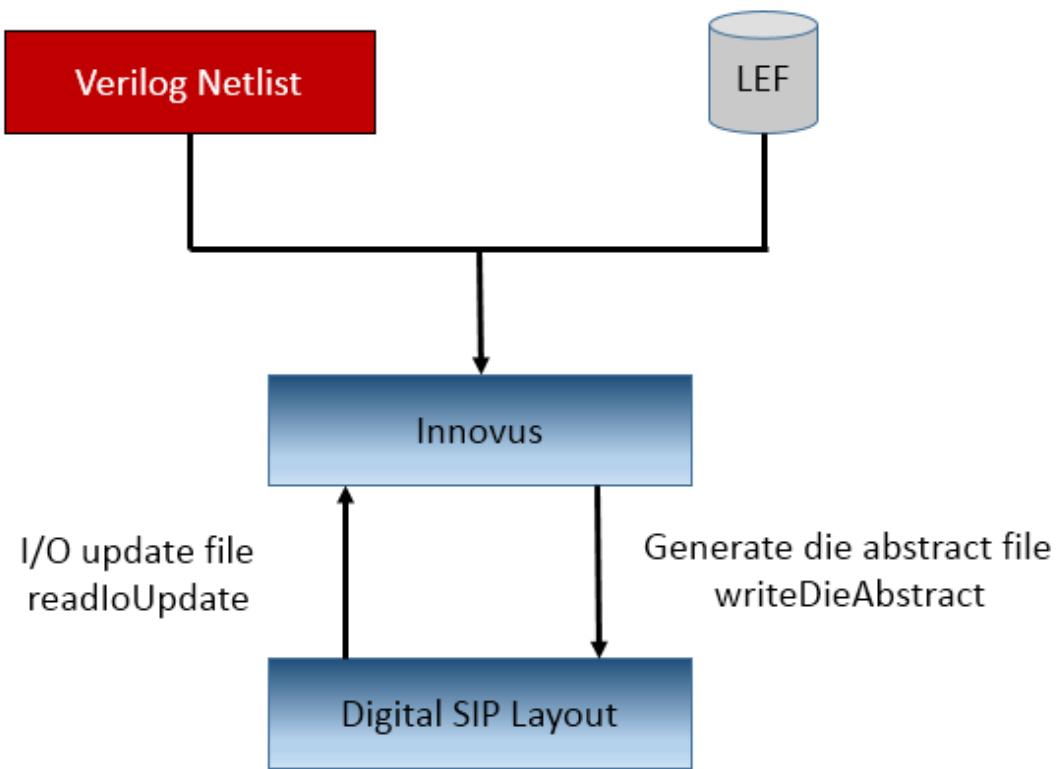
In this case the bump creation and assignment are driven by the package tool and by the constraints from the PCB. This information is fed into Innovus to drive the I/O pad placement. In this flow, the I/O pad and bump planning needs to be done before much of the digital implementation steps, as the bump assignment is driven by package requirements. Once the placement of these I/Os is fixed, the digital implementation in Innovus can start. There will be no need to come back to SiP to check the bump assignment as these should not have been modified during the design closure steps. The downside of this approach is that the package engineer does not have knowledge of the limitations and constraints coming from the logic in the design. It is probable that analog blocks have specific placement constraints, which drive the pad placement and therefore the bump assignment.

## **Co-design-driven Floorplanning**

This is the best method to achieve a quick compromise between digital implementation and the package/PCB board design.

The advantage of this method is that you can move between digital implementation and the package implementation by using SiP Layout and Innovus. This methodology allows you to see IO pads and bumps by die abstract file in Sip Layout System and by I/O file in Innovus, which can help in achieving closure on floorplanning faster.

## High-level flow diagram for co-design



In the above figure, the starting point in the flow is Innovus. However, the methodology is flexible enough for the engineer to choose either of the tools, Innovus or SiP Layout, as the starting point. Once the compromise between the two design domains is achieved in terms of routing feasibility, designers in the two domains can work on their own design issues separately as well as in parallel.

The following command in Innovus enables reading the I/O and bump -- placement and assignment information – from SiP layout into Innovus:

- `readIoUpdate`

The package balls in the package file dumped out by the SiP layout in XML format can be correctly displayed in Innovus even when the design is a flip chip design.

After saving the package XML file in the SiP Layout, you can load the package data in the Innovus floorplan view using the `readPackage` command.

For more information, see the [Flip Chip Commands](#) chapter in the *Innovus Text Command Reference*.

## Data Preparation

This section describes the data preparation required for a flip chip design.

## LEF

Innovus relies on the LEF files to identify the bumps cells and flip chip pads. The cells involved in a flipchip design must have the corresponding keywords.

## RDL Layer

The definition of the RDL layer for the flip chip flow is similar to that of other layers in LEF. Here's an example:

```
LAYER metalRDL
  TYPE ROUTING ;
  DIRECTION VERTICAL ;
  PITCH 0.800 ;
  OFFSET 0.100 ;
  HEIGHT 3.7350 ;
  THICKNESS 0.9000 ;
  MINSTEP 0.400 ;
  FILLACTIVESPACING 0.600 ;
  WIDTH 0.400 ;
  MAXWIDTH 12.0 ;
  SPACINGTABLE
    PARALLELRUNLENGTH      0.00   1.50   4.50
    WIDTH      0.00          0.40   0.40   0.40
    WIDTH      1.50          0.40   0.50   0.50
    WIDTH      4.50          0.40   0.50   1.50 ;
    AREA      0.565 ;
  MINENCLOSEDAREA 0.565 ;
...
END metalRDL
```

## BUMP

The macro type of BUMP cells need to be CLASS COVER BUMP. Here is an example of a bump cell:

```
MACRO BUMPCELL

    CLASS COVER BUMP ;

    ORIGIN 0 0 ;

    SIZE 60.0 BY 60.0 ;

    SYMMETRY X Y ;

    PIN PAD

        DIRECTION INPUT ;

        USE SIGNAL ;

        PORT

            LAYER M9 ;

            POLYGON 17.25 0.0  42.75 0.0  60.0 17.25  60.0 42.75  42.75 60.0  17.25 60.0
0.0 42.75  0.0 17.25 ;

        END

    END PAD

OBS

    LAYER via89 ;

    POLYGON 17.25 0.0  42.75 0.0  60.0 17.25  60.0 42.75  42.75 60.0  17.25 60.0
0.0 42.75  0.0 17.25 ;

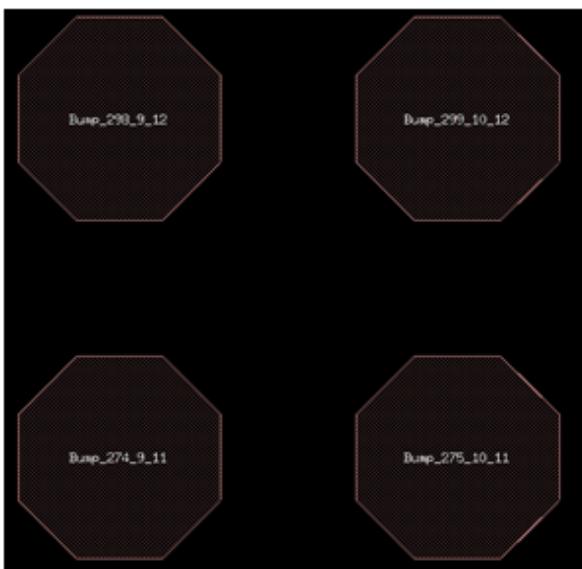
    END

END BUMPCELL
```

In the example above:

- The bump has an octagonal shape of RDL layer. Innovus also supports rectangular bumps.
- OBS can also be defined on the macro of BUMP cells and is honored by the flip chip router (fcroute).
- Innovus will display the BUMP shape in GUI.

The following figure shows how bumps are displayed in Innovus.



**Note:** Polygon shapes are supported from LEF 5.6.

## I/O Pad

The macro type of I/O pads need to be CLASS PAD AREAIO. Here is an example,

```
MACRO iopad
```

```
    CLASS PAD AREAIO ;
```

```
    ORIGIN 0.000 0.000 ;
```

```
    SIZE 35.000 BY 246.000 ;
```

```
    SYMMETRY x y r90 ;
```

```
    SITE pad ;
```

```
    PIN PAD
```

```
    PORT
```

```
    CLASS BUMP ;
```

```
.....
```

In the example above:

- The port of PIN PAD has CLASS BUMP attribute, which is an optional attribute for I/O pads with CLASS PAD AREAIO.

**Note:** More information about CLASS BUMP is in the section of CLASS BUMP.

**Note:** From the 15.1 release, the flip chip flow supports CLASS PAD I/O cell by default.

## Hard Macro

If the design has hard macros that have pins to be connected to bumps directly at the top level, these hard macros will keep their original CLASS BLOCK and the PORTS definition will be enhanced to have a CLASS BUMP associated to them. This is an example of how the LEF will look:

```
MACRO Dummy_HM

    CLASS BLOCK ;
    SIZE 2661.1200 BY 696.6000 ;
    ORIGIN 0 0 ;
    SYMMETRY X Y R90 ;
    PIN A1
        DIRECTION OUTPUT ;
        USE SIGNAL ;
    PORT
        CLASS BUMP ;
        LAYER top_layer ;
        RECT 2469.1800 0.0000 2490.1800 83.0000 ;
    END
    PORT
    ...
END Dummy_HM
```

In the example above:

- If the hard macro does not have any port with CLASS BUMP, it will not be considered for flip chip flow even if the pin A1 is on an I/O net.

**Note:** More information about CLASS BUMP is in the section of CLASS BUMP.

## CLASS BUMP Attribute

CLASS BUMP is one type of the port. It explicitly indicates that the port is a bump connection point and it can help you to distinguish which ports in a pin are for flip chip flow.

Only CLASS PAD AREAIO and CLASS BLOCK can have the CLASS BUMP attribute.

**Note:** From the 15.1 release, the flip chip flow supports CLASS PAD I/O cell by default. So the CLASS BUMP attribute is allowed to add to CLASS PAD cells and the behavior of a CLASS PAD cell with CLASS BUMP is the same as that of CLASS PAD AREAIO with CLASS BUMP.

- For CLASS PAD AREAIO, this attribute is optional. Following definitions are correct for flip chip flow.

```
MACRO iopad
  CLASS PAD AREAIO ;
  ...
  PIN PAD
  PORT
    CLASS BUMP ;
  ...
  END
  PORT
    CLASS BUMP ;
  ...
  END
  ...
  END PAD
  ...

MACRO iopad
  CLASS PAD AREAIO ;
  ...
  PIN PAD
  PORT
    #CLASS BUMP ;
  ...
  END
  PORT
    #CLASS BUMP ;
  ...
  END
  ...
  END PAD
  ...

MACRO iopad
  CLASS PAD AREAIO :
  ...
  PIN PAD
  PORT
    #CLASS BUMP ;
  ...
  END
  PORT
    CLASS BUMP ;
  ...
  END
  ...
  END PAD
  ...
```

- For CLASS BLOCK, if the CLASS BUMP attribute is not specified, the macro will not be considered for flip chip flow even if there are I/O pins connected to them.

CLASS BUMP will affect assignment and routing results.

## From PORT Level

- If none of the ports in a pin has the CLASS BUMP attribute:
  - For CLASS BLOCK macro, this pin will be excluded from flip chip flow.

```
MACRO iopad
  CLASS BLOCK ;
  ...
  PIN PAD
    PORT
      #CLASS BUMP ;
  ...
  END
  PORT
    #CLASS BUMP ;
  ...
  END
  PORT
    #CLASS BUMP ;
  ...
  END
END PAD
...
```

This pin is not for flipchip flow and will be excluded for assignment and flipchip routing.

- For CLASS\_PAD AREAIO macro, all ports will be considered as one object for assignment and flip chip routing.
  - Only one bump is assigned to the pin.
  - Flipchip router will pick one port for routing based on its intelligence.

```
MACRO iopad
  CLASS PAD AREAIO ;
  ...
  PIN PAD
    PORT
      #CLASS BUMP ;
  ...
  END
  PORT
    #CLASS BUMP ;
  ...
  END
  PORT
    #CLASS BUMP ;
  ...
  END
END PAD
...
```

All 3 ports are for flipchip flow.  
Only assign one bump to pin PAD.

- If all ports with the CLASS\_BUMP attribute are equal in one pin:

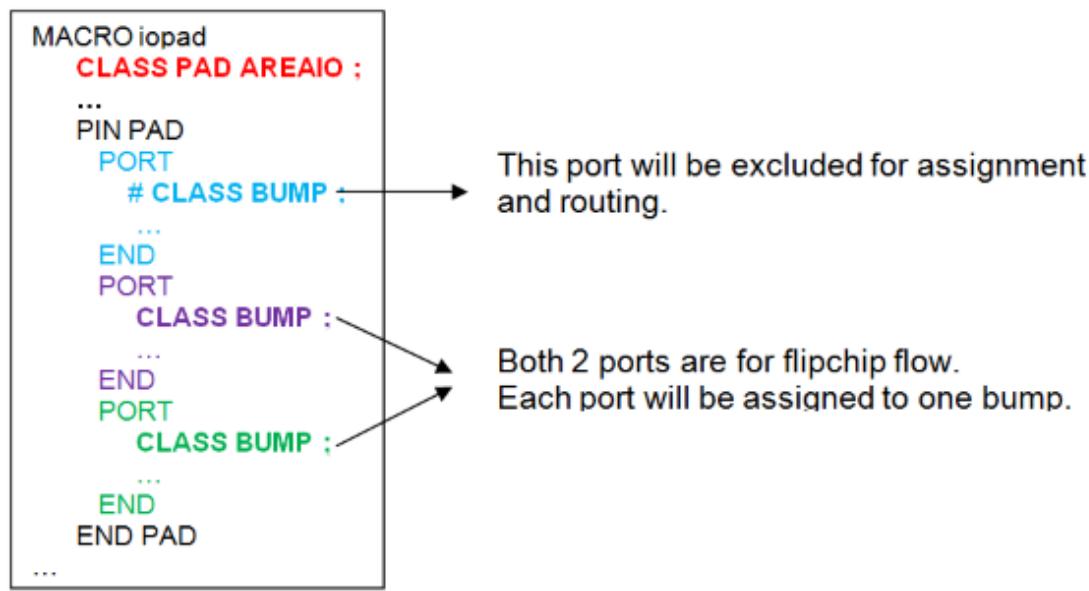
- All ports with CLASS BUMP are applied to both CLASS PAD AREAIO and CLASS BLOCK macros in terms of assignment and routing.
- Each port will be assigned to one bump.
- Every port can be routed to one or multiple bump, which depends on the setup of `fcout`. You can control the pairing of ports and bumps by adding bump connect target property.

```
MACRO iopad
  CLASS PAD AREAIO :
    ...
    PIN PAD
      PORT
        CLASS BUMP :
        ...
        END
        PORT
        CLASS BUMP :
        ...
        END
        PORT
        CLASS BUMP :
        ...
        END
      END PAD
    ...
  
```

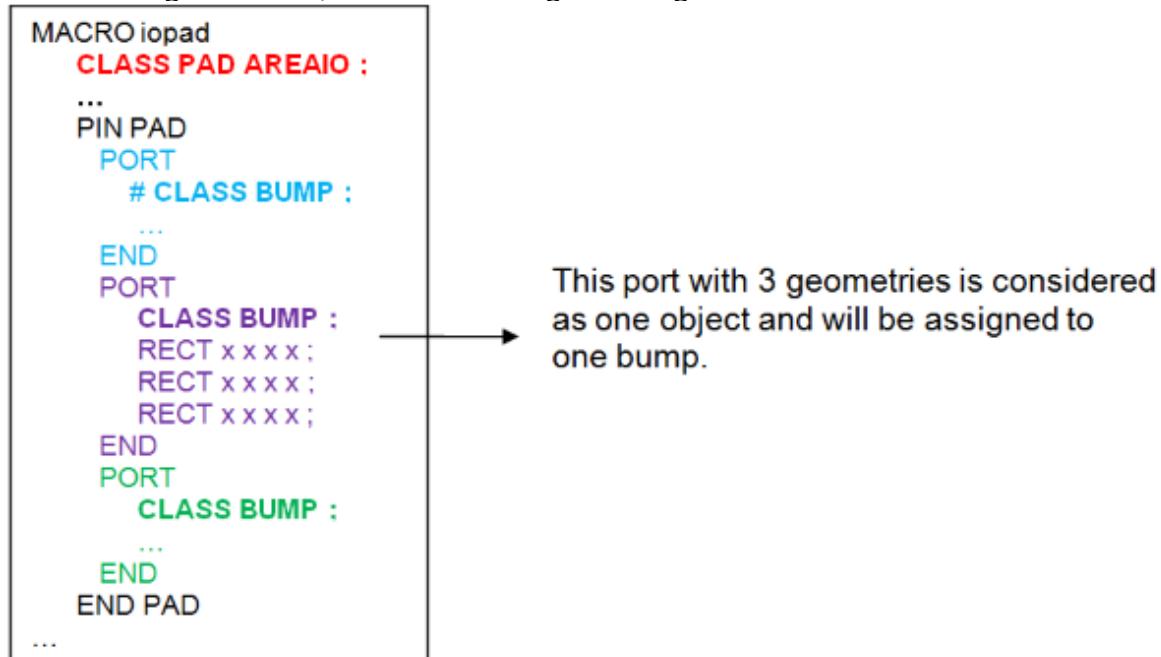
All 3 ports are for flipchip flow. Each port will be assigned to one bump.

- If some ports in the same pin have the CLASS BUMP attribute while some do not:
  - The ports without the CLASS BUMP attribute will be excluded for assignment and flip chip routing. This applies to both CLASS PAD AREAIO and CLASS BLOCK macros.

The example below depicts what happens when some ports have the CLASS BUMP attribute and some do not.



- If a port with the **CLASS BUMP** attribute has multiple geometries or port shapes:
  - It is considered as one object for assignment and flip chip routing. This applies to both **CLASS PAD AREAIO** and **CLASS BLOCK** macros in terms of assignment and routing.
  - The flip chip router will pick one geometry for routing based on its intelligence because there is no mechanism to link a specific geometry in one port to a specific bump. In case of very close geometries, **fcroute** will merge some geometries then choose better one for routing.

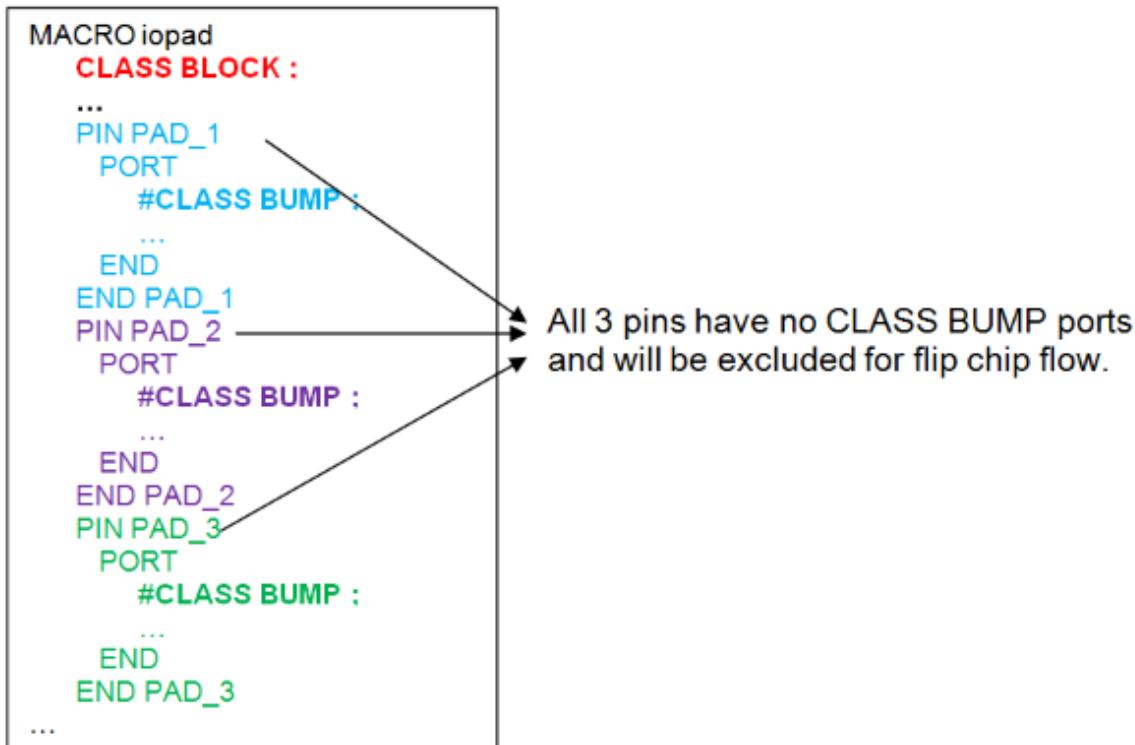


## From PIN Level

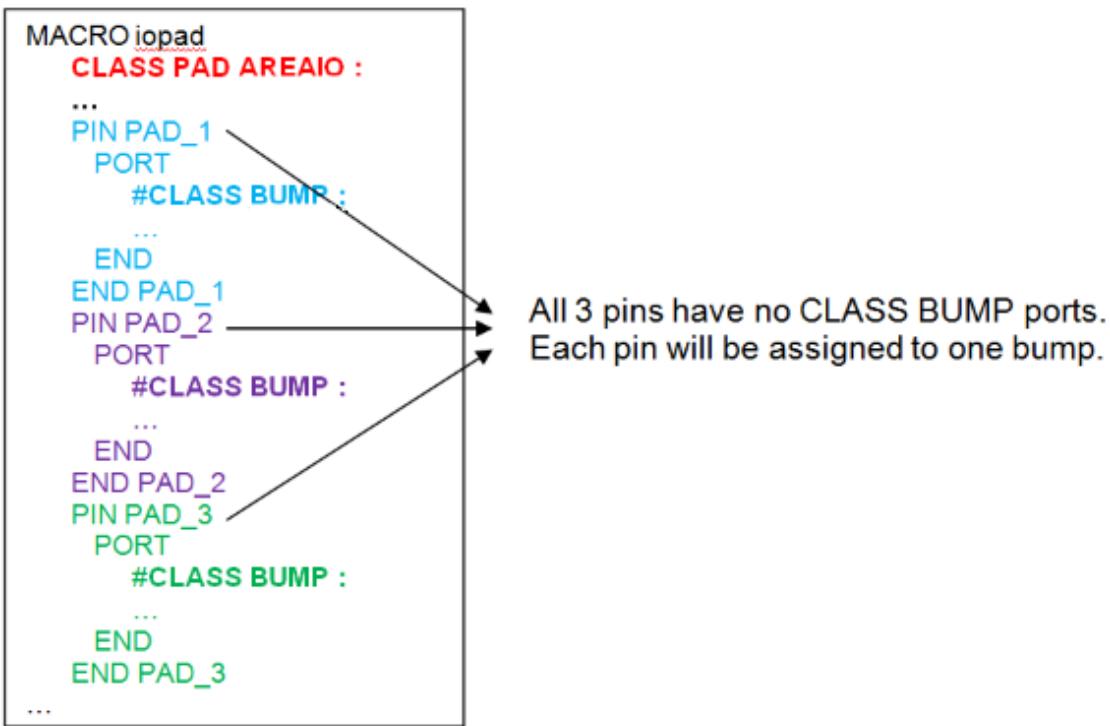
From PIN level, assume all pins are correctly defined as I/O port in the netlist and need to be connected to bumps.

- If none of pins have CLASS BUMP ports:

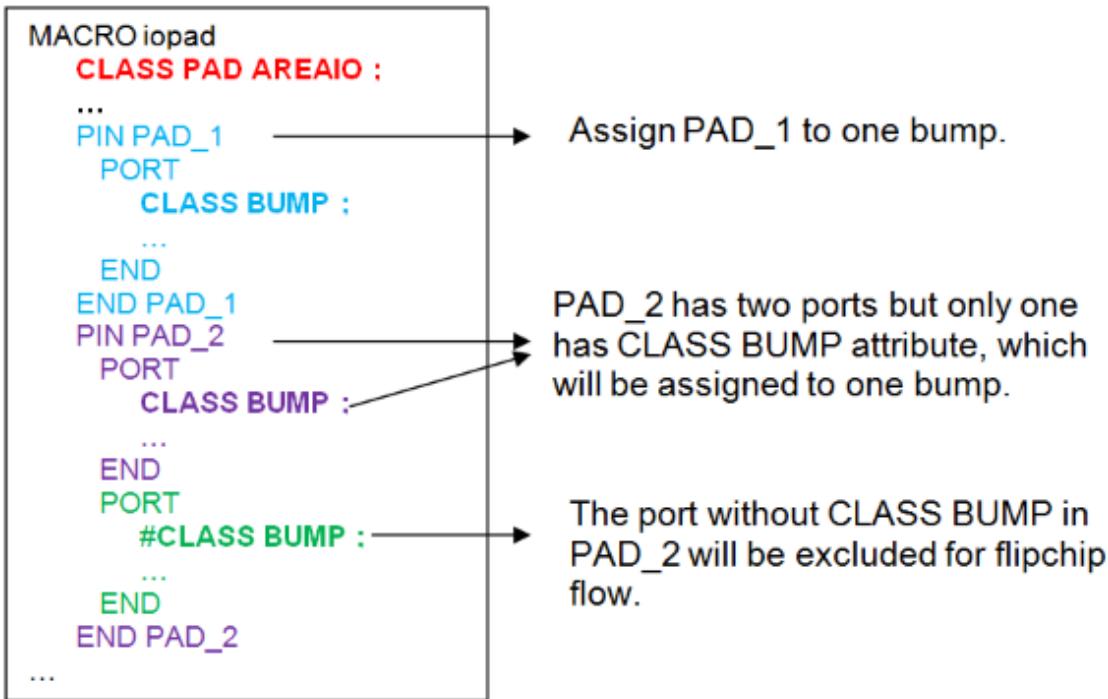
- For the CLASS BLOCK macro, these pins will be excluded from flip chip flow.



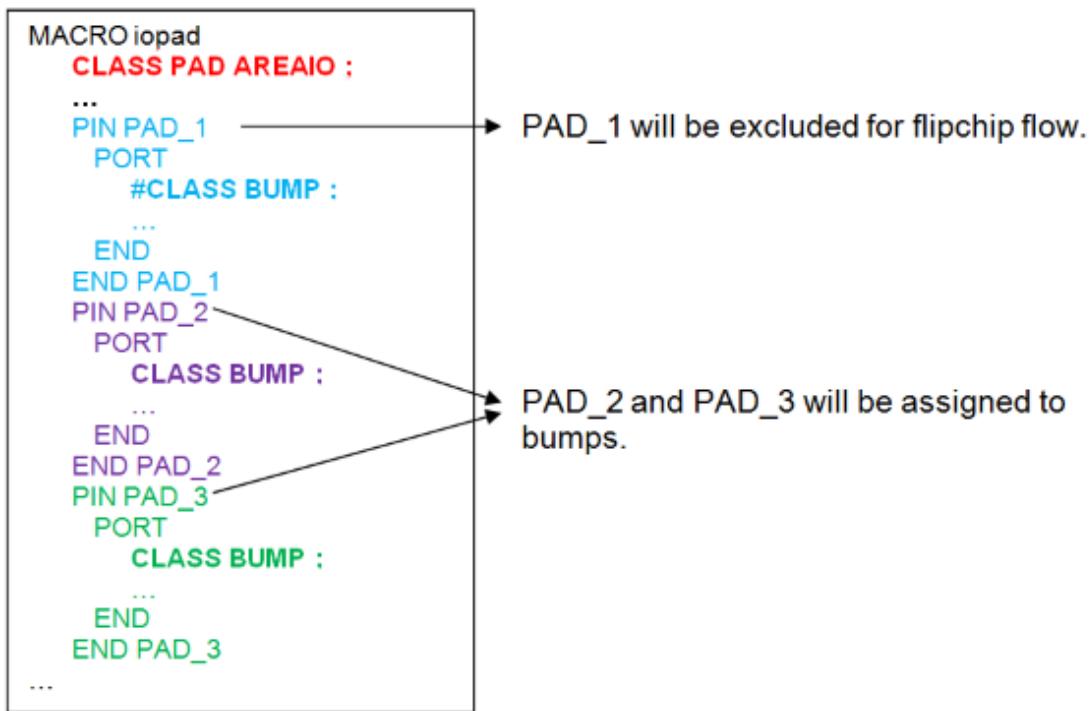
- For the CLASS PAD AREAIO macro, each pin will be assigned to one bump.



- If all pins with `CLASS BUMP` ports are equal in one macro:
  - All ports with `CLASS BUMP` are applied to both `CLASS PAD AREAIO` and `CLASS BLOCK` macros in terms of assignment and routing.
  - Each port with `CLASS BUMP` in one pin will be assigned to one bump.
  - Every port with `CLASS BUMP` in one pin can be routed to one or multiple bump.



- If some pins in a macro have `CLASS BUMP` ports but others do not:
  - The pins in the macro without `CLASS BUMP` ports will be excluded for assignment and flipchip routing. This applies to both `CLASS PAD AREAIO` and `CLASS BLOCK` macros in terms of assignment and routing.



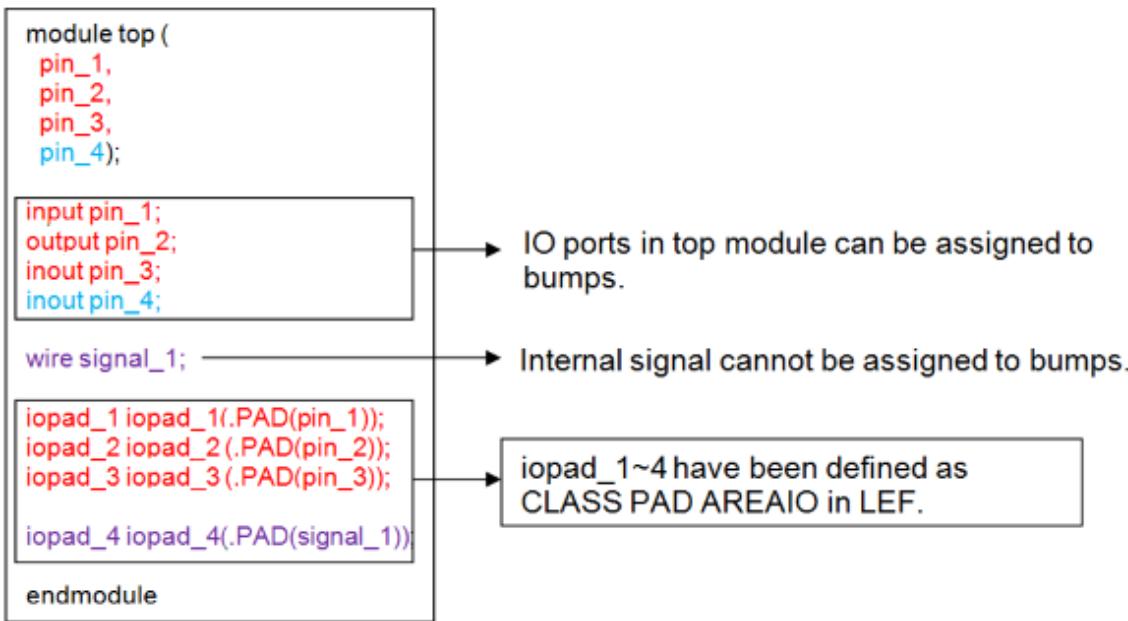
## NETLIST

A bump is physical cell, which must not be defined in the netlist. The relationship between bump and pad is constructed during bump assignment which is based on the shortest distance. Then, the flip chip router is used to connect the IO pad to its assigned bump. Defining bumps in the netlist and assigning nets to these bumps is not compatible with Innovus and cannot be handled by Innovus.

IO pads must be defined in netlist.

## Signal Pads

To be able to assign signal pads to bumps and connect them to their assigned bumps, the netlist needs to have their connection. This is an example:

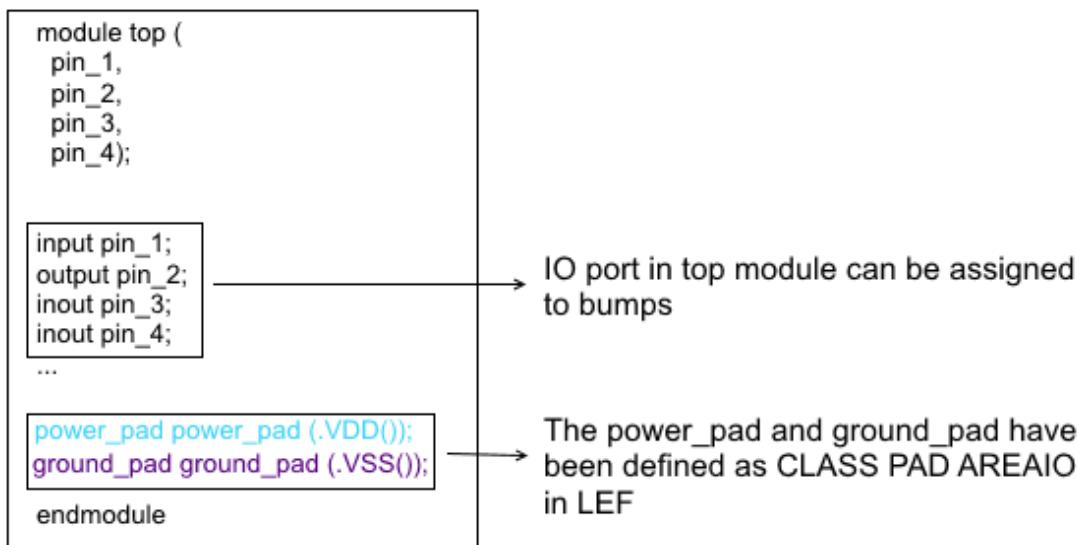


In the example above:

- IO ports pin\_1~3 have related IO pads iopad\_1~3 with which they can be connected. Therefore, the signal pads iopad\_1~3 can be connected to their assigned bumps by the flip chip router.
- IO port pin\_4 does not have an IO pad with which to be connected. You may get the following WARNING message after assignment and the flip chip router will not route net pin\_4:  
`**WARN: (ENCSP-6014) : I/O pin 'pin_4' does not connect to placed Area I/O instance or hard macro.`
- Internal wire signal\_1 cannot be assigned to bumps, so iopad\_4 with signal\_1 does not have assigned bumps and the flip chip router will not route the net signal\_1.

## Power and Ground Pads

To be able to assign PG pads to bumps and connect them to their assigned bumps, the netlist could have their connection. This is an example:



In the example above, you can find the pin VDD of power\_pad or the pin VSS of ground\_pad does not have any related IO port.

If the initial netlist does not have physical IO pads (for example, power and ground IO pads), you have the option of adding them during floorplanning with [addIoInstance](#).

To create the PG connections:

1. The power and ground net names must first be defined in the power and ground fields in the Innovus config file or .globals file:
  - Setting in the config file
 

```
set rda_Input(ui_pwrnet) {VDD}
set rda_Input(ui_gndnet) {VSS}
```
  - Setting in the .globals file
 

```
setUserDataValue init_pwr_net {VDD}
setUserDataValue init_gnd_net {VSS}
```
2. Then, you need to run the [globalNetConnect](#) command or read the CPF file ([loadCPF](#) / [commitCPF](#)).

After the PG connection creation, you can assign PG pads to bumps and connect them to their assigned bumps by using the flip chip router.

If the PG connection is not created, the flip chip router will not route these PG nets.

**Note:** PG nets definition is variable. Some users define these nets as logical nets in the netlist without declaring them in the \*.globals file. In such a case, Innovus cannot correctly recognize these nets as special nets and therefore some feature may not work properly.

# Flip Chip Floorplanning

## Bump Creation and Assignment

The flip chip die requires solder bumps to be attached to the package substrate. Bump generation is typically a two-step process -- placement and signal assignment.

### Bump Creation

There are multiple ways to create bumps in Cadence tools. You can create a single bump or a bump pattern by using the `create_bump` command.

**Note:** The `ciopCreateBump` has been made obsolete from the 14.2 release.

Usually, bumps are created in a regular pattern with fixed pitch. The `create_bump` command can support the following bump patterns in the chip:

- `-pattern_full_chip`
- `-pattern_side {side width}`
- `-pattern_array {row column}`
- `-pattern_ring width`
- `-pattern_center {row column}`

**Note:** All these different bump patterns can coexist in the same floorplan.

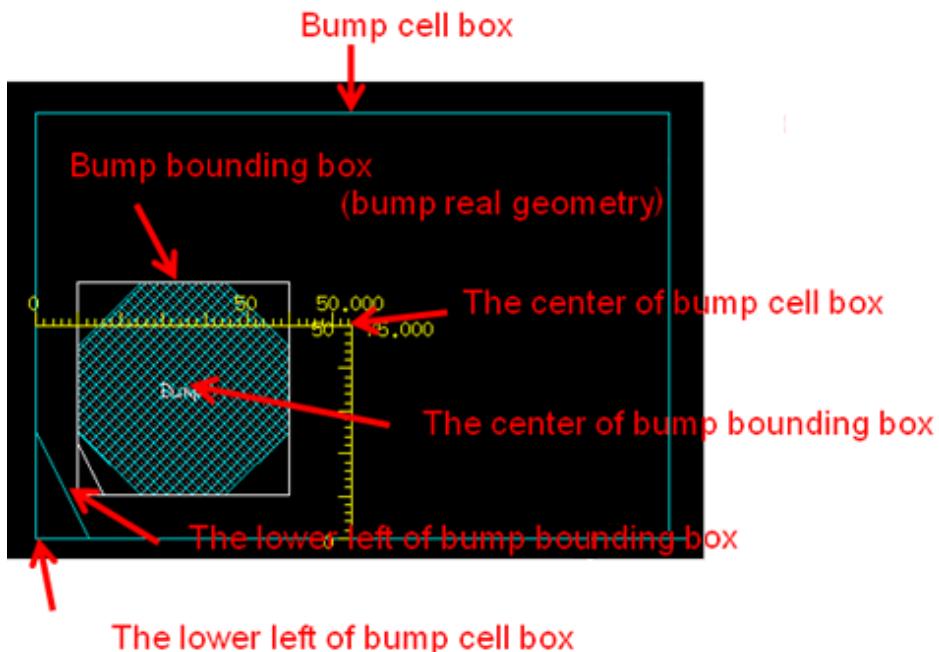
During bump creation, the tool will issue a warning and will not create bumps where there are overlaps with other bumps based on bump geometry. But you could specify the `-allow_overlap` option to create bumps with overlapping.

The bump creation process does not look into any type of routing blockages or obstruction in hard macro LEFs. The `deleteBumps` command has the options `-overlap_blockages` and `-overlap_macros`, which can be used to clean up the placement of the bumps before signal assignment.

Currently, `verifyGeometry` will not highlight overlaps between bumps and routing blockages. The bumps need to be assigned (committed) in order for `verifyGeometry` to flag the short violation. It is a normal procedure to delete the unassigned bumps after a flip chip design implementation.

A bump has four location types as specified below. `create_bump` provides you the capability to specify which location type is used for bump creation:

- Bump cell center
- Bump cell lower left location  
This location can be obtained from bump attribute editor or by using the following command:  
`dbget [dbget top.bumps.name $Bump_name -p].pt`
- Bump geometry (bounding box) center  
This location can be obtained with the following command:  
`dbGet top.bumps.bump_shape_center`
- Bump geometry (Bounding box) lower left location  
This location can be obtained with the following command:  
`dbGet top.bumps.bump_shape_bbox`



## Bump Assignment

After creating bumps, the user can now assign signal and PG bumps.

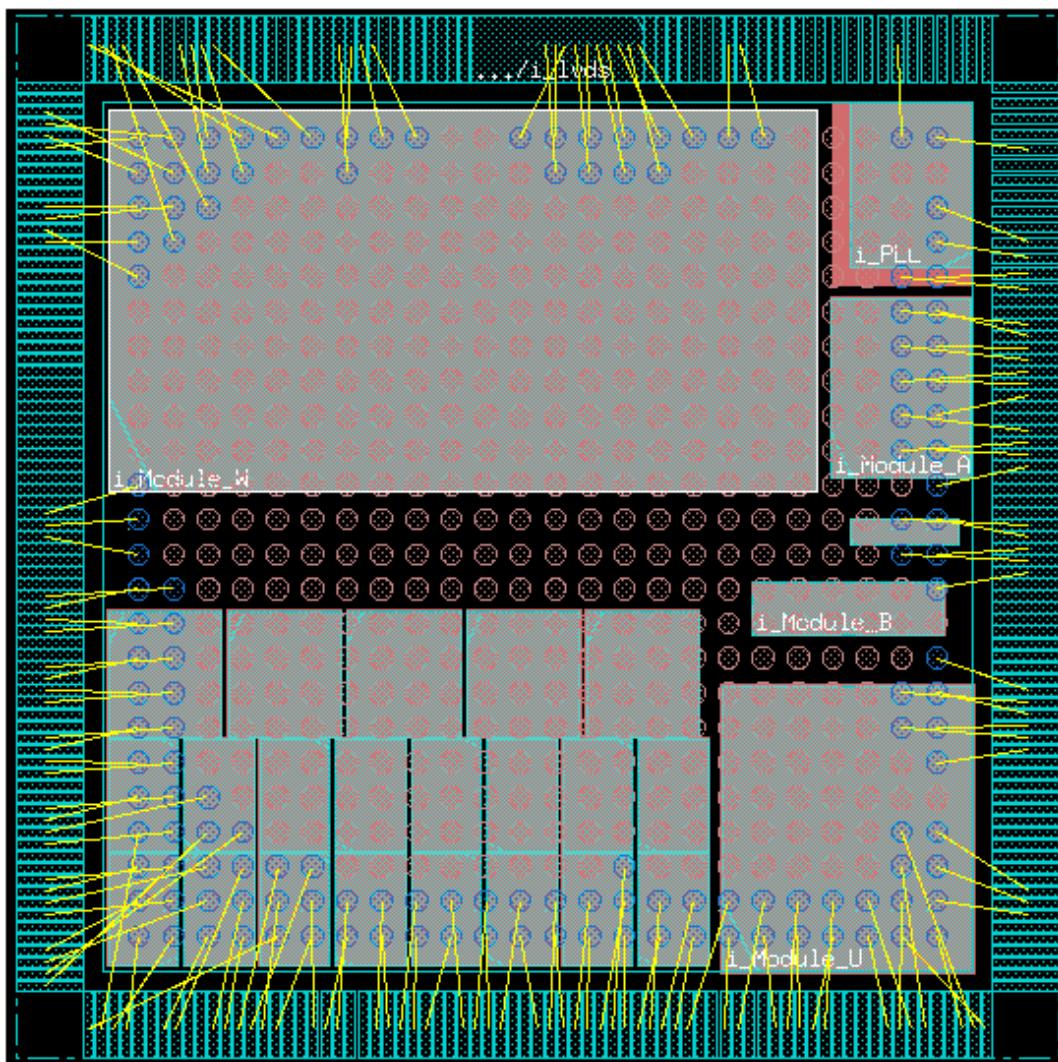
## Signal Assignment

For signal assignment, you can automatically assign the signal bumps to the closest pad IO. This normally gives a suboptimal assignment for routing.

- Automatic signal assignment

You can use the `assignBump` command for signal assignment. This is fully automatic assignment and it will assign all available signals for flip chip to bumps based on the shortest distance.

- Manual signal assignment with the `assignSigToBump` command.



## P/G Assignment

For PG assignment, the tool accepts the possibility of associating PG pads to specific bumps along with signal assignment. You can choose to assign PG and signal pads to bumps together or just do the assignment separately .

- Assign PG and signal pads to bump together

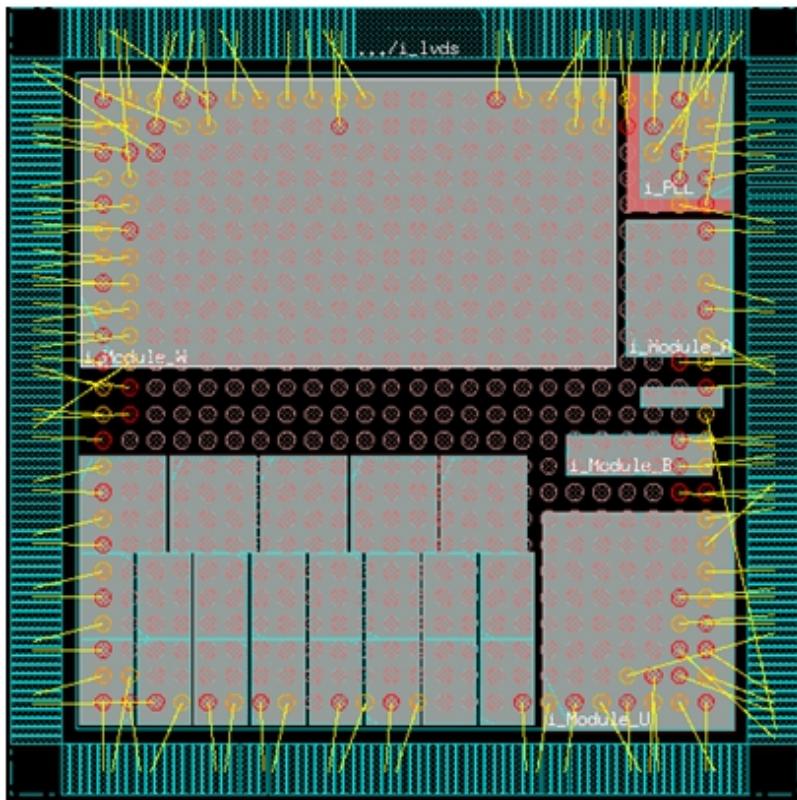
The assignment can be done automatically by using the command `assignBump -pgnet`

{net\_list}. With this usage, the tool will consider PG nets and signal nets together and distribute appropriate bump resource from the global view based on the shortest distance.

- Assign PG nets only

The tool also allows you to assign only PG nets to bumps automatically by using the command `assignBump -pgonly -pgnet {net_list}`.

If you want to have more control over the assignment of the power and ground nets, the tool offers a manual assignment method by using the `assignPGBumps` and `assignSigToBump` command.



#### Notes:

- Notice how the bumps change color once they are assigned. By default, blue for an assigned signal bump, red for a power bump, and yellow for a ground bump.
- You can change the color of the assignment by supplying the net name and a valid color, which can be obtained from the link <http://www.w3.org/TR/SVG/types.html#ColorKeywords>. This is useful when you want to track the assignment of very specific critical nets. The command to read this text file containing color mapping for bumps is `clopLoadBumpColorMapFile`. This is an example of the file.

*Clk green*

*Address\* magenta*

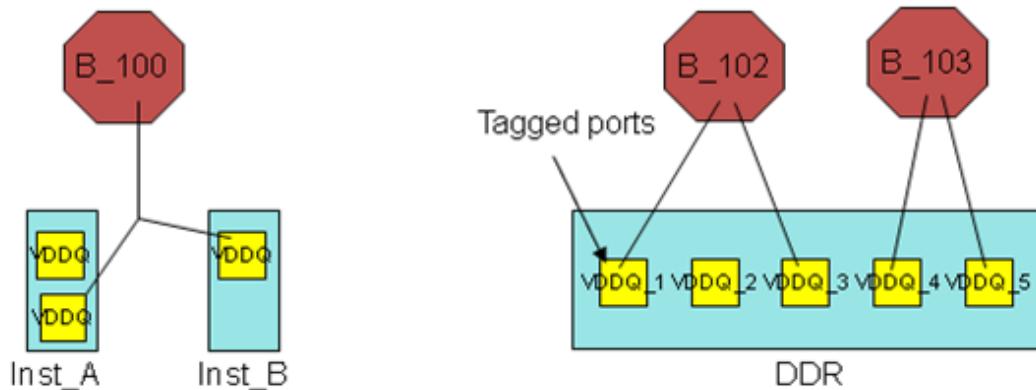
It can support wildcards for ease of use.

- After assignment, you can use the `viewBumpConnection` command, which can display the connection as a flight line between IO pads and bumps.  
From the 11.1 release, the flight lines of the `viewBumpConnection` command can be automatically redrawn after bump manipulations.  
You can use the `viewBumpConnection -bumpType power` to display the flight lines of only PG connections.  
You can use the `viewBumpConnection -multiBumpsToPad` or `viewBumpConnection -multiPadsToBumps` command to turn on multiple connections. Without these two options, this command will only display one-to-one connections.

## Port Numbering Approach in Assignment

For signal assignment, one net is usually related to one bump and the flip chip router can find the connection based on nets; however, for PG assignment, many PG pads are connected to the same net. Therefore, you need to explicitly specify the pairing for PG connection. This means that for a customized pattern of multiple pads to multiple bumps, `fcroute` needs to know exactly which pin or ports need to be routed to which bump.

The port numbering approach, available since Release 11.1, allows you to specify the connection between pads to bumps explicitly during the assignment stage.



The port numbering feature allows you to specify explicitly:

- Which bump is to be connected to which port

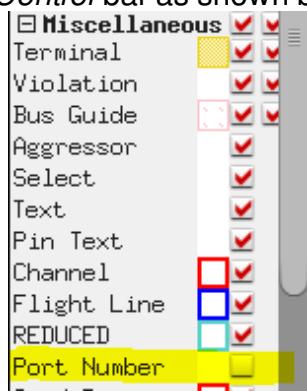
For example: Bump B\_102, B\_103, and DDR

- Which pad is to be connected to which bump in case of multiple pads to multiple bumps  
For example: Inst\_A, Inst\_B and bump B\_100

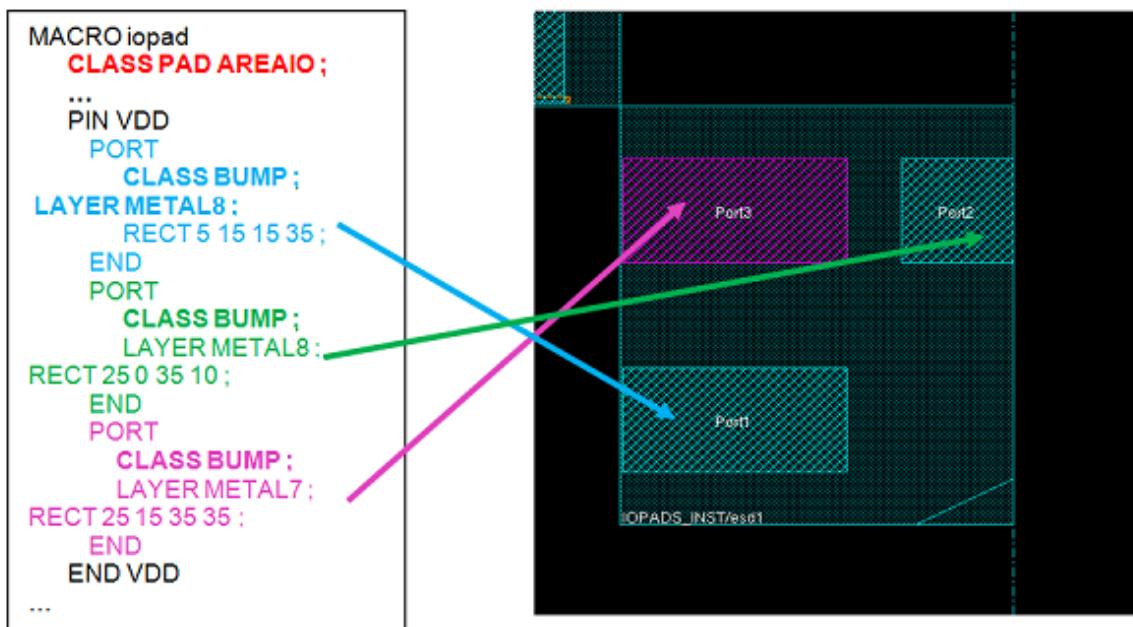
You can add the CLASS BUMP attribute in LEF onto pins/ports to specify explicitly which pin/port is for flip chip assignment and routing. This attribute has been introduced in the section **CLASS BUMP**.

PORts are numbered uniquely per cell in the Innovus database so that they can be referred during assignment and routing based on instance.

- Numbers are references for ports.
- Regardless of whether or not the CLASS BUMP attribute is specified in LEF, ports will be numbered.
- You can instantly view port numbers by selecting *Miscellaneous -> Port Number* on the *Layer Control* bar as shown below:

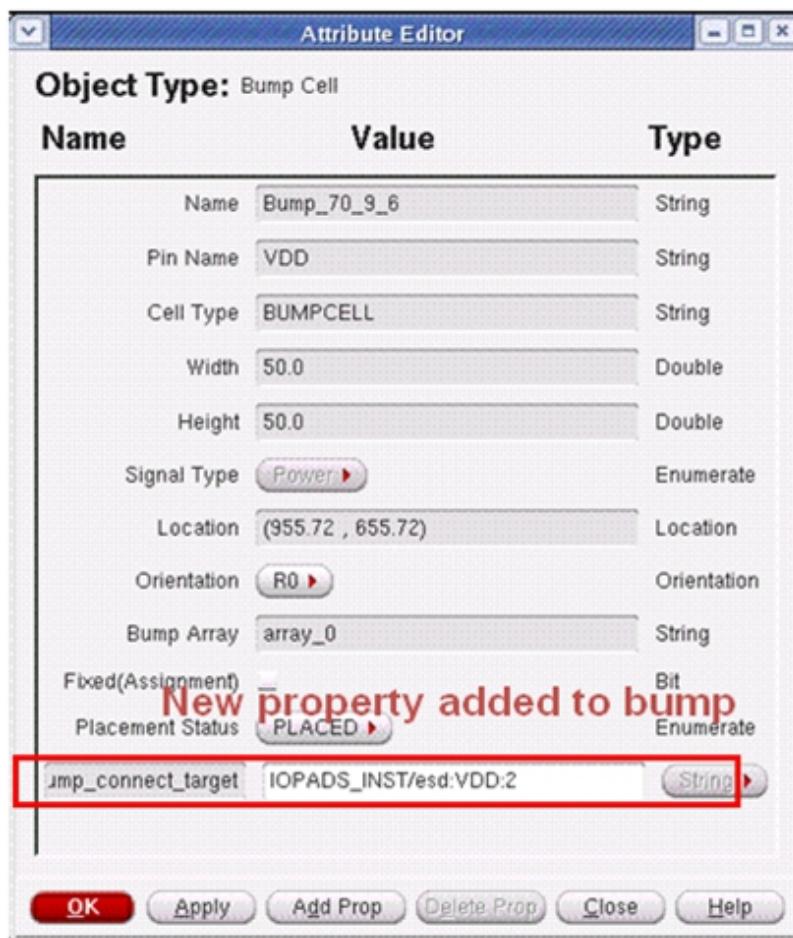


The figure below shows how the tool displays port number based on the LEF definition.



You can use the `assignBump -useTargetProp ...` command to turn on the port numbering feature for automatic assignment, which can be used for both signal and PG assignment. This will add the port number property onto bumps and you can open the Attribute Editor for a bump to see the added property values after assignment. The value takes the following format:

|                        |                                                                       |
|------------------------|-----------------------------------------------------------------------|
| Property name          | bump_connect_target                                                   |
| Type                   | string                                                                |
| Property string format | <i>inst_name or inst_name:pin_name or inst_name:pin_name:port_num</i> |



Use the commands listed in the table below for manipulating and saving or restoring properties.

| Assignment         | Property Manipulation             | Property Save/restore | Routing |
|--------------------|-----------------------------------|-----------------------|---------|
| assignBump         | addBumpConnectTargetConstraint    | writeFlipChipProperty | fcroute |
| deleteBumps        | editBumpConnectTargetConstraint   | readFlipChipProperty  |         |
| swapSignal         | deleteBumpConnectTargetConstraint |                       |         |
| viewBumpConnection | findPinPortNumber                 |                       |         |

### Notes

- Add `srouteFcroutePadPinTagging TRUE` in the extra configuration file to turn on the port numbering feature for flip chip router (`fcroute`).
- Since 14.2, the number of pin port can be obtained using the `dbGet` command. If the design uses

the old config flow, you have to call *init\_flipchip* to get port.

```
> dbget selected.instTerms.cellTerm.pins.?

pin: allShapes class layerShapeShapes objType shapeViaShapes portNumber
> dbget ${instCellpgTerm_gnd}. pins.portNumber

1 2 3 4 ... 10 11

> dbget ${instCellpgTerm_gnd}. pins.??

allShapes: 0x1ece25b8           allShapes: 0x1ece2608           allShapes: 0x1ece25e0
class: undefined                 class: bump                   class: bump
layerShapeShapes:               layerShapeShapes:             layerShapeShapes:
0x1ece25b8                     0x1ece2608                 0x1ece25e0
objType: pin                   objType: pin                 objType: pin
shapeViaShapes: 0x0             shapeViaShapes: 0x0             shapeViaShapes: 0x0
portNumber:1                   portNumber:10                portNumber:11
...                                ...
```

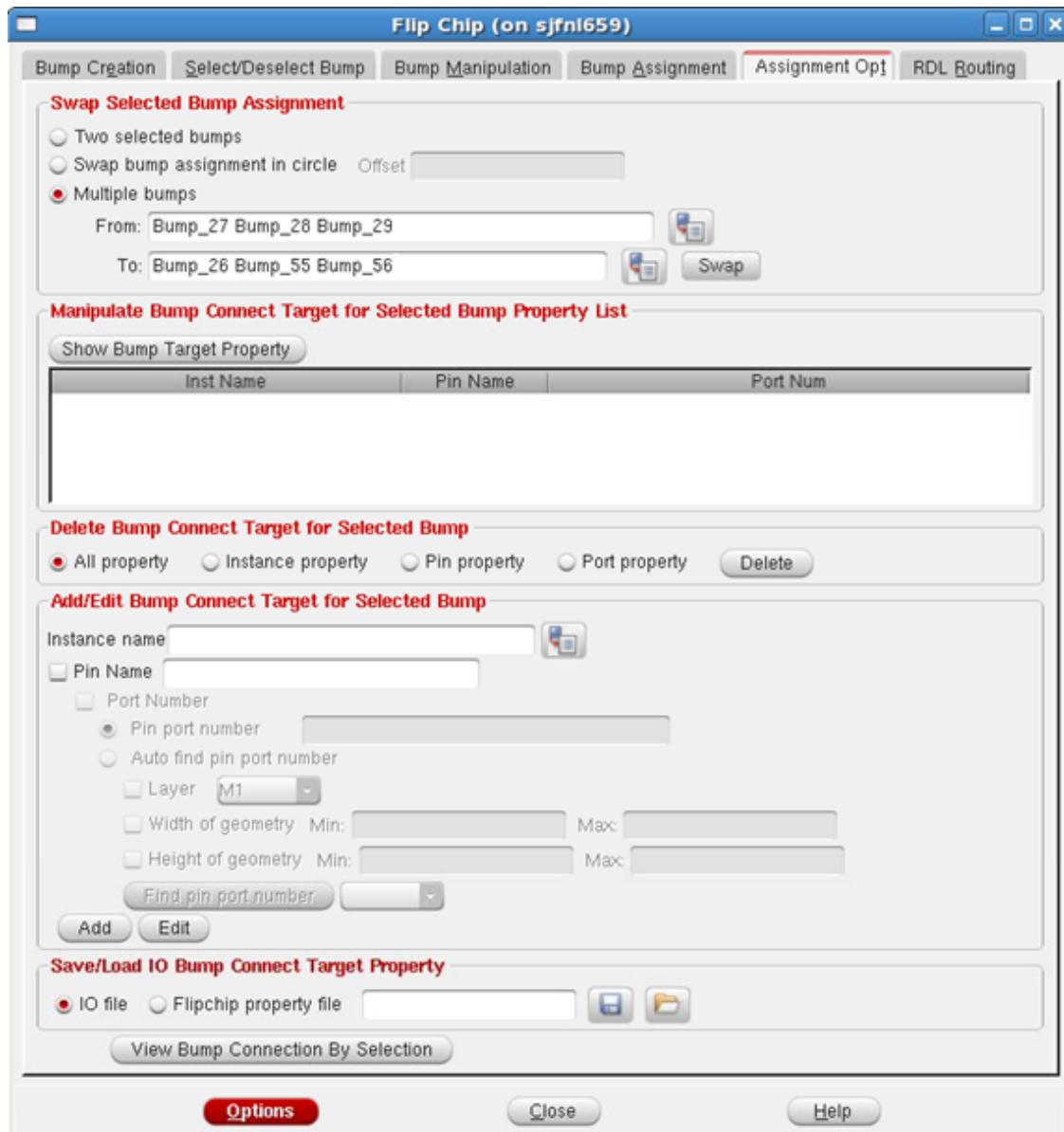
## Bump Assignment Optimization

The tool offers two methods for bump assignment optimization.

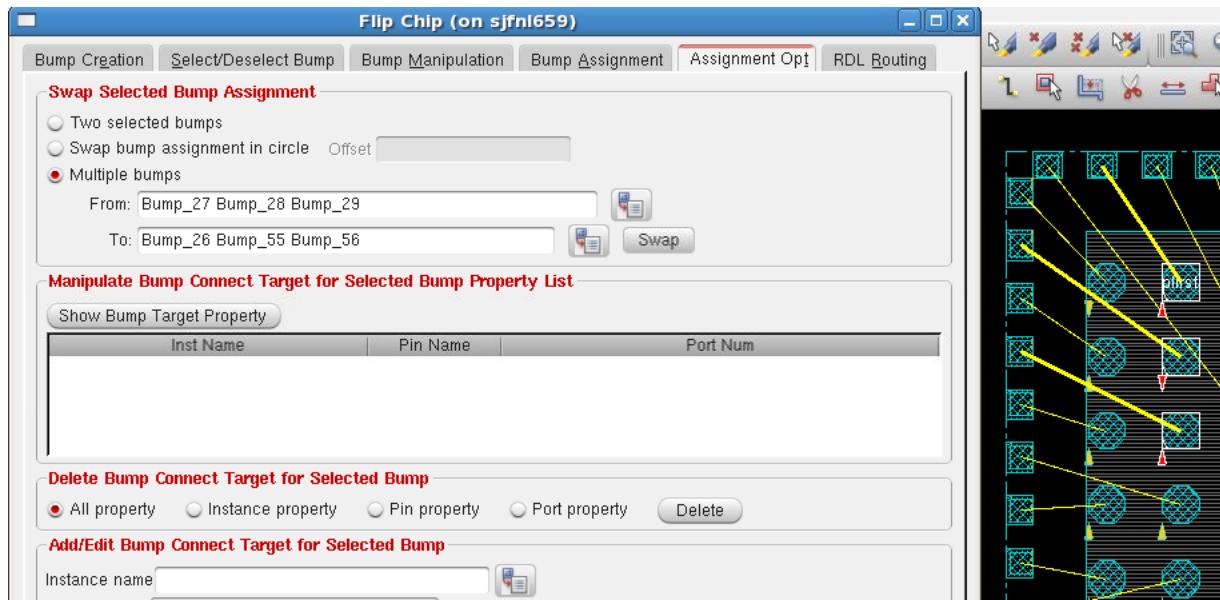
- Manual optimization
- Use bump assignment constraints

### Manual Bump Assignment Optimization

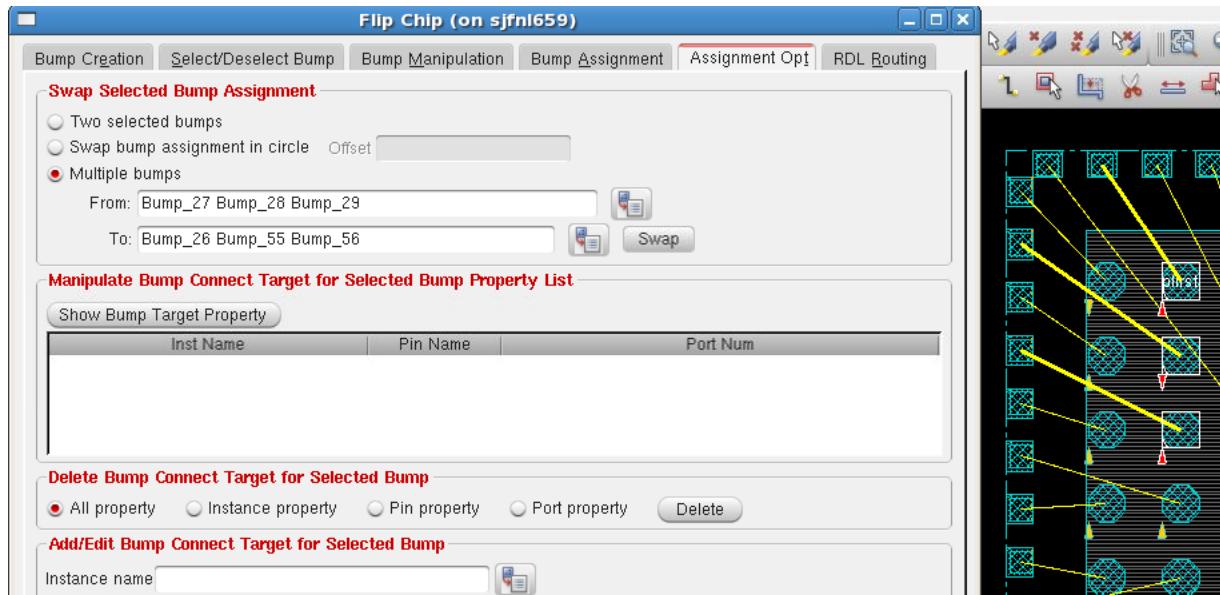
If you need to perform minor ECOs on the assigned bumps, use the [swapSignal](#) command or choose *Tools ->Flip Chip* and then click the *Assignment Opt* tab in the Flip Chip form.



The following figure shows the signals before swapping.



The following figure shows the signals after swapping.



## Using Bump Assignment Constraints

If you have any constraints for bump assignment, such as constraints to filter ports of pad pin, you can specify these in a constraint file. The bump assignment constraint file is loaded using the `assignBump_constraint_file` option.

At present, the following types of bump assignment constraints are supported:

- `SHARE_FIND_PORT` constraint to filter unnecessary ports
- `ASSIGN_ANALOG_PG_NETS` constraint to specify which signal nets are analog PG nets
- `SHARE_IGNORE_*` and `ASSIGN_IGNORE_*` constraints to exclude instances, macros, pins, or nets for assignment.
- `ASSIGN_PAD2BUMP_RATIO` constraint to specify the pad to bump ratio per net, macro, or instance.

## SHARE\_FIND\_PORT Constraint

The syntax for the `SHARE_FIND_PORT` constraint is as follows:

```
SHARE_FIND_PORT
PIN pin_name_list
MACRO macro_name_list
      LAYERS top_layer[:bottom_layer]
      GEOMETRY_SHORT_EDGE min_value [:max_value]
      GEOMETRY_LONG_EDGE min_value [:max_value]
net_name_list
END SHARE_FIND _PORT
```

The `SHARE_FIND_PORT` constraint can help you find a specific port. However, the property of `CLASS BUMP` port in the LEF file is higher priority than this constraint, which means the `SHARE_FIND_PORT` constraint cannot filter the port with `CLASS BUMP` in the LEF file.

## Parameters

- Net name and at least one of the parameters - `LAYERS`, `GEOMETRY_SHORT_EDGE`, and `GEOMETRY_LONG_EDGE` - must be specified to filter unnecessary pin ports.
- PIN *pin\_name\_list*
  - Specifies the list of the constraint pins connected with the specified net.
  - Is optional. If not specified, the constraints for the specified net work on all available pin ports for the flip chip design.
  - Supports wildcard matching.

- MACRO *macro\_name\_list*
  - Specifies the list of the constraint MACROs whose pins are connected with the specified net.
  - Is optional. If not specified, the constraints for the specified net work on all available MACROs for the flip chip design.
  - Supports wildcard matching.
- LAYERS *top\_layer[: bottom\_layer]*
  - Specifies the layer or the layer region on which the pin connected with the specified net is located.
  - The name of the layer must support layer ID and layer name in LEF.
- GEOMETRY\_SHORT\_EDGE *min\_value [:max\_value]*
  - Specifies the value or the region of the short edge length of the pin geometry connected to the specified net. If only one value is specified, it means the expected short edge length of pin geometry equals the value. Otherwise, the expected short edge length is between *min\_value* and *max\_value*.
  - The unit is micron.
  - GEOMETRY\_SHORT\_EDGE means the length of the short edge.
  - assignBump ignores the geometries with a short edge length that does not meet the rule.

For example, pin VDD has following different ports:

- port1: 5x25
- port2: 15x10
- port3: 5x5
- port4: 10x20
- port5: 15x25

If constraint file specifies GEOMETRY\_SHORT\_EDGE 10:20, then port2, port4 and port5 can become candidates for assignment.

On the other hand, if constraint file specifies GEOMETRY\_SHORT\_EDGE 5, port1 and port3 can become candidates for assignment.

- GEOMETRY\_LONG\_EDGE *min\_value [:max\_value]*

- Specifies the value or the region of the long edge length of the pin geometry connected to the specified net. If only one value is specified, it means the expected long edge length of pin geometry equals the value. Otherwise, the expected long edge length is between `min_value` and `max_value`.
  - The unit is micron.
  - `GEOMETRY_LONG_EDGE` means the length of the long edge.
  - `assignBump` ignores those geometries with a long edge length that does not meet the rule.
- `net_name_list`
    - Specifies the list of the nets. This is a mandatory parameter.
    - Supports wildcard matching and also `@signal`, `@power` and `@ground`.

Here's an example of the `SHARE_FIND_PORT` constraint:

```
SHARE_FIND_PORT
    PIN VDD
    MACRO VDD_PAD DDR_VDD_PAD
        LAYERS metalALP
        GEOMETRY_SHORT_EDGE 20:25
        GEOMETRY_LONG_EDGE 15
    @power
END SHARE_FIND_PORT
```

If this constraint is specified in the `assignBump` constraint file, only those pins that meet the following requirements are available for bump assignment. `assignBump` ignores all bumps that do not meet these requirements:

- To be connected to the `power` net in the design
- The name in LEF is `VDD`
- Belongs to the macros `VDD_PAD` or `DDR_VDD_PAD`
- On the layer `metalALP`
- Has short edge between 20 and 25 microns and long edge equals 15 microns

## ASSIGN\_ANALOG\_PG\_NETS Constraint

The syntax for the ASSIGN\_ANALOG\_PG\_NETS constraint is as follows:

```
ASSIGN_ANALOG_PG_NETS  
  net_name_list  
END ASSIGN_ANALOG_PG_NETS
```

## Parameters

- *net\_name\_list*
  - Specifies the list of nets that are defined as signal nets in LEF or netlist but are actually analog PG nets.
  - Supports wildcard matching.

### Notes:

- The nets specified with the ASSIGN\_ANALOG\_PG\_NETS constraint are still signal nets but are used for auto power/ground assignment. assignBump treats these nets in the same way as other normal power/ground nets.
  - a. These nets can be specified as arguments of the assignBump -pgnet {net\_list} or -exclude\_pgnet {net\_list} options.
  - b. These nets are considered when the assignBump -pginst {instance\_list} is used.
  - c. These nets are excluded from signal assignment of assignBump.
- assignBump does not modify the property of specified nets in ASSIGN\_ANALOG\_PG\_NETS. Therefore, after assignment, the bumps that are assigned with the specified nets in ASSIGN\_ANALOG\_PG\_NETS must be signal bumps instead of power/ground ones.

## SHARE\_IGNORE\_\* and ASSIGN\_IGNORE\_\* Constraints

The SHARE\_IGNORE\_\* constraint can be used to exclude instances, macros, pins, or nets for assignment. The syntax of the the constraint varies depending on what is being excluded.

## Excluding Instances

Use the following syntax to specify list of instances to be excluded from assignment:

```
SHARE_IGNORE_INSTANCE
```

```
instance_name_list
```

```
END SHARE_IGNORE_INSTANCE
```

Here, *instance\_name\_list* specifies the list of instances that are to be excluded during assignment. It supports wildcards.

## Excluding Macros

Use the following syntax to specify list of macros to be excluded from assignment:

```
SHARE_IGNORE_MACRO
```

```
macro_name_list
```

```
END SHARE_IGNORE_MACRO
```

Here, *macro\_name\_list* specifies the list of macros that are to be excluded during assignment. It supports wildcards.

## Excluding Instance Pins

```
ASSIGN_IGNORE_INSTANCE_PIN
```

```
instance_name:pin_name
```

```
...
```

```
END ASSIGN_IGNORE_INSTANCE_PIN
```

Here *instance\_name:pin\_name*:

- Specifies which pin of the specified instance is to be excluded during assignment.
- Supports multiple parameters.
- Supports wildcards.
- Does not allow any blank spaces before or after ":"

## Excluding Macro Pins

```
ASSIGN_IGNORE_MACRO_PIN  
macro_name:pin_name  
...  
END ASSIGN_IGNORE_MACRO_PIN
```

Here *macro\_name:pin\_name*:

- Specifies which pin of the specified macro is to be excluded during assignment.
- Supports multiple parameters.
- Supports wildcards.
- Does not allow any blank spaces before or after “:”

## Excluding Nets

```
ASSIGN_IGNORE_NET  
net_name_list  
END ASSIGN_IGNORE_NET
```

Here, *net\_name\_list* specifies the list of nets that are to be excluded during assignment. It supports wildcards.

## ASSIGN\_PAD2BUMP\_RATIO Constraint

The syntax for the ASSIGN\_PAD2BUMP\_RATIO constraint is as follows:

```
ASSIGN_PAD2BUMP_RATIO  
TOLERANCE net_name integer  
PGNET net_name ratio  
PGMACRO macro:pin ratio  
PGINST inst:pin ratio  
END ASSIGN_PAD2BUMP_RATIO
```

The ASSIGN\_PAD2BUMP\_RATIO constraint specifies the pad to bump ratio per net, macro, or instance.

## Parameters

### Parameters for Ratio Greater than 1

The following parameters work with a pad to bump ratio greater than 1.

- TOLERANCE *net\_name integer*
- PGNET *net\_name ratio*

When the pad to bump ratio is greater than 1, `assignBump` calculates the number of groups of multiple PG ports to multiple bumps and the maximum number of ports in each group based on the specified ratio per net.

Assume the total number of ports is *total\_port*, and the ratio is *port\_num:bump\_num*.

- If *total\_port* is an integer multiple of the ratio:

Number of groups =  $\text{total\_port} \div (\text{port\_num}/\text{bump\_num}) + \text{TOLERANCE}$

Else:

Number of groups =  $[\text{total\_port} \div (\text{port\_num}/\text{bump\_num})] + 1 + \text{TOLERANCE}$

- If *port\_num* is an integer multiple of *bump\_num*:

Maximum number of ports in each group =  $\text{port\_num} \div \text{bump\_num}$

Else:

Maximum number of ports in each group =  $[\text{port\_num} \div \text{bump\_num}] + 1$

For example, if there are 5 pads with VDD and the VDD pin of each pad has only one port and the pad to bump ratio is 3:2. Then:

Number of groups = Round off  $\{5 \div (3/2) + 0\} = 4$

Maximum number of ports in each group = Round off  $\{3/2\} = 2$

This feature tries to provide the best assignment result in following conditions:

- The number of groups remains unchanged.
- In each group, the number of ports must be no more than the maximum number of ports.

Following is the description of the parameters for ratio greater than 1:

- TOLERANCE *net\_name integer*
  - Specifies the tolerance value for specific nets. It only supports positive integers. With this option, the number of pad/port groups will be counted by the tolerance value.
  - The default value is zero.

- It works only for PGNET *net\_name ratio*
- *net\_name* supports wildcard matching.
- PGNET *net\_name ratio*
  - Specifies the ratio for specific nets.
  - *net\_name* supports wildcard matching.
  - The ratio must be more than 1. If you specify a ratio less than 1, assignBump issues an ERROR message and ignores the ratio.
  - If the ratio is not in the simplest form, the tool will simplify it and give a WARNING.
  - For different PG nets, define the ratio per net separately. If you specify more than 1 ratio for the same net, assignBump issues a WARNING message and chooses the last specified ratio for the net.

### Example1

If the ratio for both VDD and VDD0 is 2:1 and the ratio for VSS is 3:2, the syntax is as below:

```
PGNET VDD 2:1      #The ratio for VDD is 2:1
PGNET VDD0 2:1     #The ratio for VDD0 is 2:1
PGNET VSS 3:2      #The ratio for VSS is 3:2
PGNET VSS 3:2      #As this is the second and last ratio for VSS, assignBump
                   #issues a WARNING message and use it as the ratio of
                   #net VSS.
PGNET VSS0 1:2      #As the ratio for VSS0 is less than 1, assignBump issues
                   #an ERROR message and ignores it.
```

### Example 2

```
PGNET * 2:1        #The ratio for all pg nets specified in the assignBump command
is 2:1
```

### Example 3

In the ratio.const constraint file, following is specified:

```
PGNET VDD 4:2
PGNET VSS 4:2
```

Run the following command:

```
assignBump -constraint_file ratio.const -pgnet {VDD VSS}
```

- The tool first simplifies the ratio to 2:1.
- The tool calculates the number of pad/port groups and maximum number of ports in each group as follows:

- For VDD, as the total number of ports with VDD (5) is not an integer multiple of *port\_num* (2), therefore:

$$\begin{aligned} \text{Number of groups} &= [\text{total\_port} \div (\text{port\_num}/\text{bump\_num})] + 1 + 0 = [5 \div (2/1)] + 1 \\ &= [2.5] + 1 = 2 + 1 = 3 \end{aligned}$$

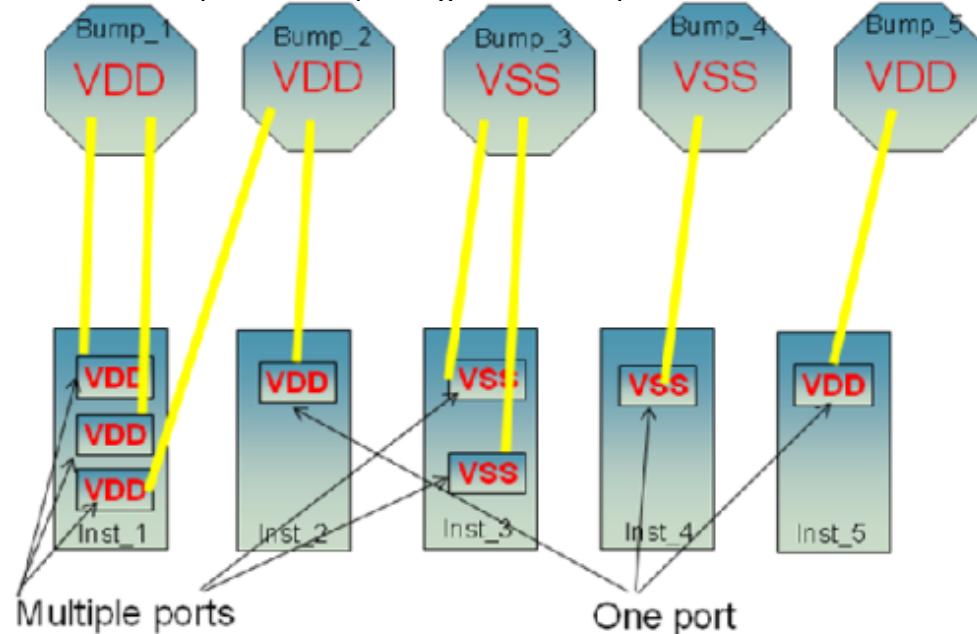
- For VSS, as the total number of ports with VSS (3) is not an integer multiple of *port\_num* (2), therefore:

$$\begin{aligned} \text{Number of groups} &= [\text{total\_port} \div (\text{port\_num}/\text{bump\_num})] + 1 + 0 = [3 \div (2/1)] + 1 \\ &= [1.5] + 1 = 1 + 1 = 2 \end{aligned}$$

- Both VDD and VSS use the same ratio 2:1. As in this ratio, *port\_num* (2) is an integer multiple of *bump\_num* (1), therefore:

$$\begin{aligned} \text{Maximum number of ports in each group for VDD and VSS} \\ = \text{port\_num} \div \text{bump\_num} &= 2/1 = 2 \end{aligned}$$

The multi-PG pad to bump assignment is depicted below.



## Parameters for Ratio Less than 1

The following parameters work with a ratio less than 1.

- PGMACRO *macro:pin ratio*
  - Specifies the ratio for the specific MACRO cell.
  - *macro:pin* supports wildcard.
  - The ratio must be less than 1. Otherwise, assignBump issues an ERROR message and ignores it.
  - If you specify more than 1 ratio for the same pin, assignBump issues a WARNING message and chooses the last specified ratio for it.
  - Choose the out-most geometry as the target and record the port number into *bump\_connect\_target*.

### Example 1

```
PGMACRO DDR1:VDD    1:2    #The ratio for VDD pin in DDR1 is 1:2
PGMACRO DDR2:VDD    1:2    #The ratio for VDD pin in DDR2 is 1:2
PGMACRO DDR2:VDD    1:3    #As this is the second ratio and last ratio for
                           #VDD pin in DDR2, assignBump issues a WARNING
                           #message and uses it as the ratio of VDD pin in DDR2.
PGMACRO DDR3:VSS    2:1    #As the ratio is more than 1,
                           #assignBump issues an ERROR message and ignores it.
```

### Example 2

```
PGMACRO * 1:2          #The ratio for all pg pins in all macros of the
                        #instances specified in the assignBump command
                        #is 1:2.
PGMACRO DDR1:* 1:2      #The ratio for all pg pins in DDR1 is 1:2.
PGMACRO *:VDD 1:2       #The ratio for VDD pin in all macros of the
                        #instances specified in the assignBump command
                        #is 1:2.
```

- PGINST *inst:pin ratio*
  - Specifies the ratio for the specific instance.
  - *inst:pin* supports wildcard.

- The ratio must be less than 1. Otherwise, `assignBump` issues an ERROR message and ignores it.
- If you specify more than 1 ratio for the same pin, `assignBump` issues a WARNING message and chooses the last specified ratio for it.
- Choose the out-most geometry as the target and record the port number into `bump_connect_target`.

**Example 1**

```

PGINST inst1:VDD 1:2 #The ratio for VDD pin in inst1 is 1:2
PGINST inst2:VDD 1:2 #The ratio for VDD pin in inst2 is 1:2
PGINST inst2:VDD 1:3 #As this is the second and last ratio for VDD pin
                     #in inst2, assignBump issues a WARNING message
                     #and uses it as the ratio of VDD pin in inst2.

PGINST inst3:VSS 2:1 #As the ratio is more than 1, assignBump issues
                     #an ERROR message and ignores it.
    
```

**Example 2**

```

PGINST * 1:2          #The ratio for all pg pins of all instances
                      #specified in the assignBump command is 1:2.

PGINST inst1:* 1:2    #The ratio for all pg pins in inst1 is 1:2.

PGINST *:VDD 1:2      #The ratio for VDD pin of all instances
                      #specified in the assignBump command is 1:2.
    
```

If there is some conflict over ratio, the priority order is `PGINST` > `PGMACRO` > `PGNET` as shown in the following example.

```

ASSIGN_PAD2BUMP_RATIO

PGNET VDD 2:1          #The ratio for VDD is 2:1.

PGMACRO DDR:VDD 1:2    #If the VDD pin in DDR is connected to VDD,
                      #the ratio for VDD pin in DDR is 1:2 as the priority
                      #of PGMACRO is higher than PGNET.

PGINST inst:VDD 1:3     #The MACRO cell of inst is DDR. As the priority of PGINST
                      #is higher than PGMACRO, the ratio for VDD pin in inst
                      #is 1:3

PGNET VSS 3:1
    
```

```
PGINST inst1:VSS 1:3 #If the VSS pin in inst1 is connected to VSS, the ratio
#for VSS pin in inst1 is 1:3 as the priority of
#PGINST is higher than PGNET.

END ASSIGN_PAD2BUMP_RATIO
```

The usage of `ASSIGN_PAD2BUMP_RATIO` is as below:

- It turns on the feature of pads to bumps assignment by ratio.
- If used with `-pgnet`, `ASSIGN_PAD2BUMP_RATIO` works on only the specified PG nets.
- If used with `-exclude_pgnet`, `ASSIGN_PAD2BUMP_RATIO` ignores the constraints for the specified excluded PG nets.
- If used with `-pginst`, `ASSIGN_PAD2BUMP_RATIO` works on only the instances specified by `PGMACRO` and `PGINST`.
- If used with `-pgnet` and `-pginst`, `ASSIGN_PAD2BUMP_RATIO` works on only the specified PG nets of the instances specified by `PGMACRO` and `PGINST`.
- If used with `-exclude_pgnet` and `-pginst`, `ASSIGN_PAD2BUMP_RATIO` works on the PG nets of the instances specified by `PGMACRO` and `PGINST`, with the exception of the specified excluded PG nets.
- If the objects specified by `-pgnet`, `-exclude_pgnet` and `-pginst` are not included in `ASSIGN_PAD2BUMP_RATIO`, `assingBump` issues a warning message and uses 1 as the ratio value.
- If not used with `-pgnet`, `-exclude_pgnet` or `-pginst`, `ASSIGN_PAD2BUMP_RATIO` ignores all the defined constraints.

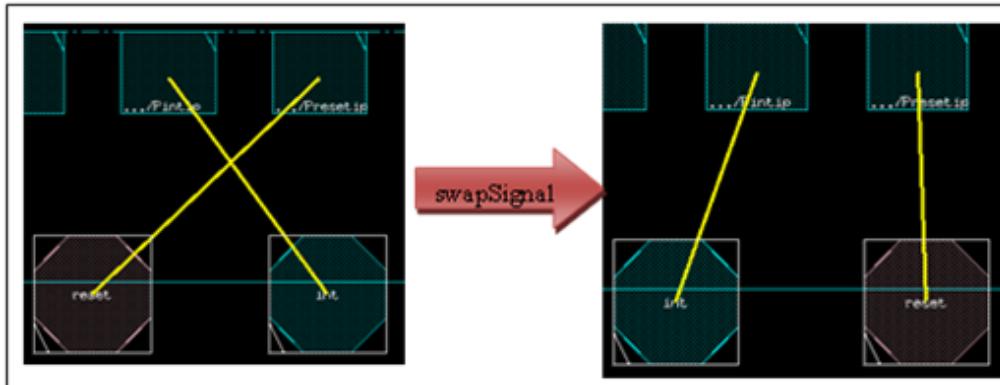
# Viewing Flip Chip Flightlines

In flip chip designs, flightlines are used extensively to interact with the design. Flip chip flightlines are different from the normal blue flightlines in Innovus and can be displayed using the `viewBumpConnection` command. `viewBumpConnection` provides the following features to make it easy for you to use flightlines:

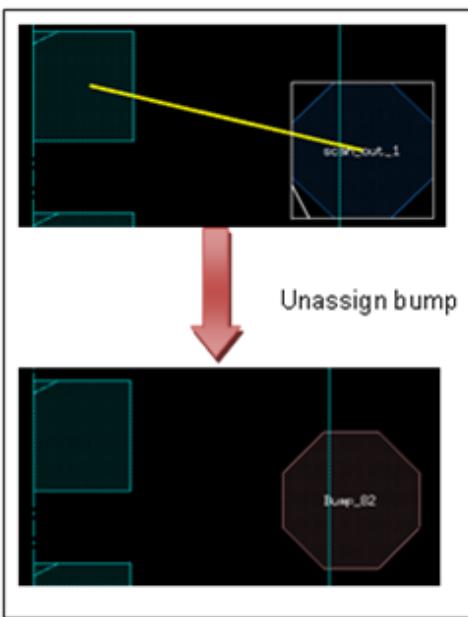
## Automatic Redraw Feature

`viewBumpConnection` redraws flightlines automatically after the following bump manipulation actions:

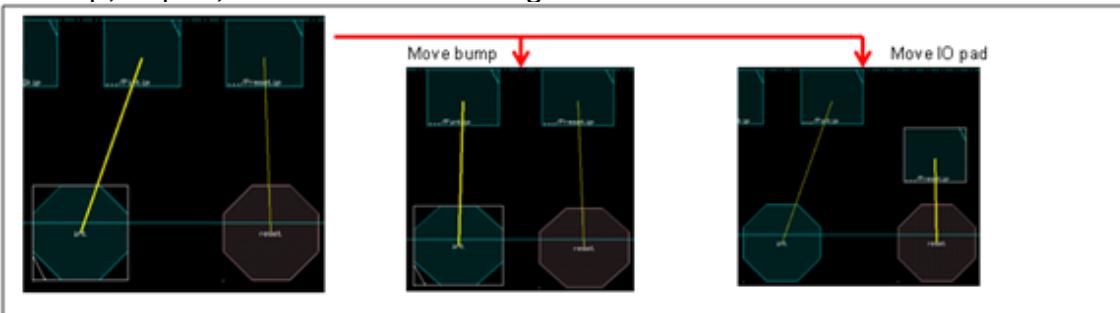
- Bump assignments are swapped using `swapSignal`: Flightlines of the selected bumps are swapped to reflect the manipulation.



- Bumps are unassigned using `unassignBump`: Flightlines of specified bumps are removed.



- A bump, IO pad, or block is moved: Flightlines are redrawn to reflect the new location.

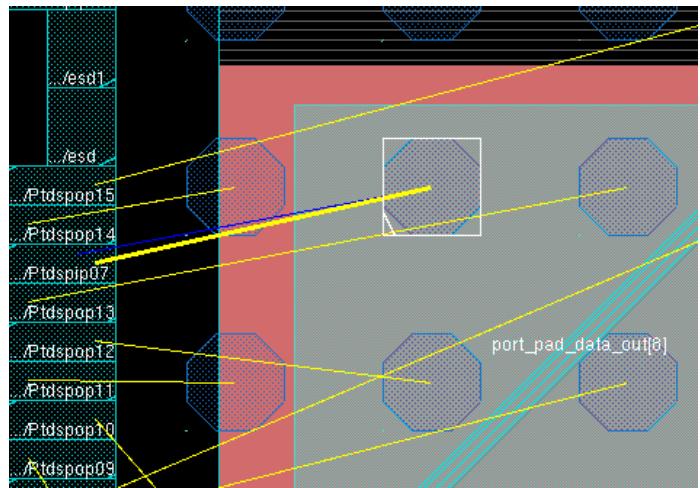


## Selection-Based Highlighting

When you select an object, the corresponding flightlines are highlighted in bold.

- When a bump or IO pad is selected, its corresponding flightline is highlighted in bold.
- When multiple bumps/IO pads are selected, all their flightlines are highlighted in bold.
- If a block with multiple IO pins is selected, all its flightlines are highlighted in bold.
- When the objects are deselected, the corresponding flightlines return to non-bold status.

1. Run `viewBumpConnection` to display all flip chip flightlines.
2. Click on an object to highlight its flightline in bold.



## Colored Flightlines

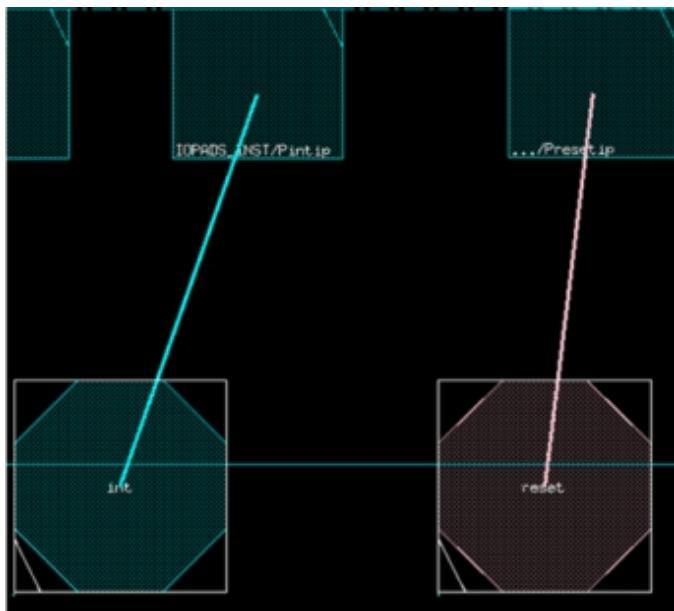
By default, all flip chip flightlines are displayed in yellow. You can use the `viewBumpConnection -honor_color` option to color these flightlines based on either bump type or the nets to which the bumps are assigned:

- To color flightlines by bump type, simply run `viewBumpConnection -honor_color`. The tool displays flightlines using the default colors of the bumps:
  - Blue for signal bumps
  - Red for power bumps
  - Yellow for ground bumps
- To color flightlines based on the nets to which they are assigned, you must:
  1. Define bump color settings in a bump color map file using the following format:

```
net_name color_name
```

Example:

```
int cyan
reset pink
```
  2. Load the bump color file using the `cioLoadBumpColorMapFile` command.
  3. Run `viewBumpConnection -honor_color`.



For bumps whose nets are not defined in the bump color file, the default colors are used as follows – blue for signal bumps, red for power bumps, and yellow for ground bumps. A flightline has the same color as its bump.

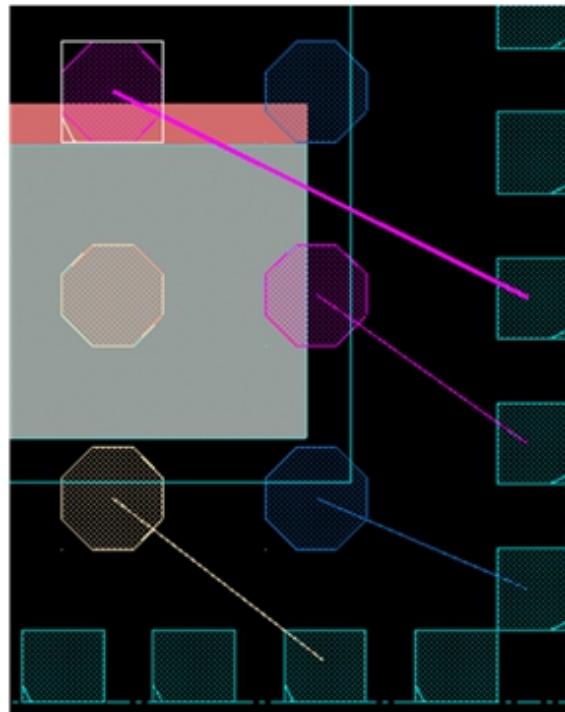
## Object-Specific Flightlines

You can easily view connections for specific objects, such as bumps, nets, and IO instances, using the following `viewBumpConnection` parameters:

- `-bumps {bump_list}`: Use this parameter to view connections of specified bumps.
- `-io_inst {io_inst_list}`: Use this parameter to view connections of specified IO instances or blocks.
- `-nets {net_list}`: Use this parameter to view connections of specified nets.
- `-selected`: Use this parameter to view connections of selected bumps or IO pads in bold. If a block with multiple IO pins is selected, all its flightlines are displayed in bold.

For example, the following command displays the flightlines for the `port_pad_data_out[10]` net, the `Bump_29` bump, and the `IOPADS_INST/Ptdspop07` instance. It also displays in bold the flightline for the selected bump:

```
viewBumpConnection \
-net {port_pad_data_out[10]} \
-bump Bump_29 \
-io_inst IOPADS_INST/Ptdspop07 \
-selected \
-honor_color
```



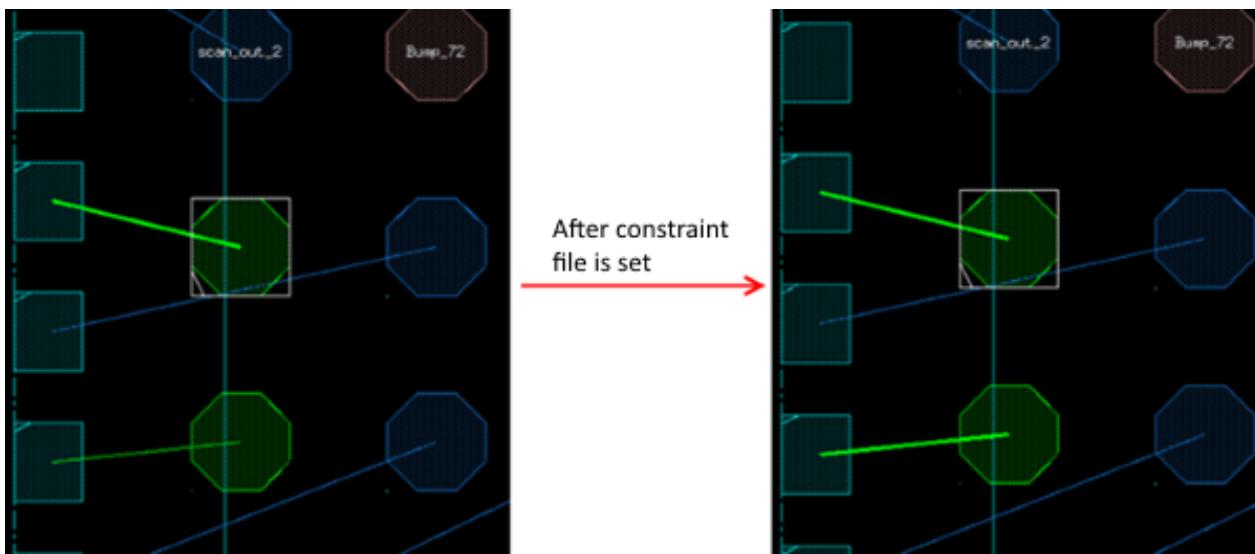
## DIFFPAIR-Based Highlighting

Flip chip flightlines now honor the DIFFPAIR constraints specified in the flip chip router constraint file. This means that when you select any one bump or IO pad that is part of a DIFFPAIR constraint, the tool highlights all flightlines of that DIFFPAIR in bold.

For example, suppose the flip chip router constraint file, `diffpair.const`, has the following setting:

```
SHARE_DIFFPAIR
    port_pad_data_in[15]
    port_pad_data_in[13]
END SHARE_DIFFPAIR
```

Now after `setFlipChipMode -constraintFile diffpair.const` is set, the flightlines for the DIFFPAIR are highlighted in bold when any one bump or IO pad of the DIFFPAIR is selected:



Currently, you cannot turn off normal flightlines to focus on DIFFPAIR flightlines. However, you can use `viewBumpConnection -nets net_list` as a workaround. Here, `net_list` specifies nets of the DIFFPAIR. This way, you can display only the flightlines for the DIFFPAIR and turn off all other flightlines.

## viewBumpConnection Display Rules

A PAIR constraint is frequently used in the constraint file to define the connection between bump and IO pad for power/ground net explicitly. The flip chip flightlines, viewed using `viewBumpConnection`, honor the `SHARE_PAIR` constraint and display the connection between bump and IO pad for power/ground net.

If you want flip chip flightlines to honor the `SHARE_PAIR` constraint:

1. Specify the `SHARE_PAIR` constraint in the flip chip constraint file using the following syntax:

```
SHARE_PAIR
net_name pad_name_list bump_name_list
END SHARE_PAIR
```

2. Specify the constraint file using the following command:

```
setFlipChipMode -constraintFile file_name
```

`viewBumpConnection` displays the connection between bump and IO pad based on the net. The following examples use net VDD to explain the `viewBumpConnection` display rules:

## Example 1: Design has only one bump, `bump_vdd`, for `VDD`

- If `bump_vdd` does not have the port number property or the PAIR constraint, `viewBumpConnection` auto-pairs it and displays the connections for `VDD`.
- If `bump_vdd` has only the `SHARE_PAIR` constraint, `viewBumpConnection` displays the connection specified by the `SHARE_PAIR` constraint.
- If `bump_vdd` has only the port number property, `viewBumpConnection` displays the connection specified by the port number.
- If `bump_vdd` has both the port number property and the `SHARE_PAIR` constraint, `viewBumpConnection` displays only the connection specified by port number.

## Example 2: Design has multiple bumps for `VDD`

- If none of the bumps has either the port number property or `SHARE_PAIR` constraint, `viewBumpConnection` auto-pairs them and displays the connections for `VDD`.
- If all of the bumps have only the `SHARE_PAIR` constraint, `viewBumpConnection` displays the connections specified by the `SHARE_PAIR` constraint.
- If all of the bumps have only the port number property, `viewBumpConnection` displays the connections specified by the port number.
- If one bump has both the port number property and the `SHARE_PAIR` constraint as in the following example:

1. `Bump_1`: Does not have either the port number property or the `SHARE_PAIR` constraint.

2. `Bump_2`: Only has a `SHARE_PAIR` constraint:

```
SHARE_PAIR
VDD pad_2 Bump_2
END SHARE_PAIR
```

3. `Bump_3`: Only has the port number property.

4. `Bump_4`: Has both the `SHARE_PAIR` constraint and the port number property:

```
SHARE_PAIR
```

```
VDD pad_4 Bump_4  
END SHARE_PAIR
```

5. Bump\_5 and Bump\_6: Only have a `SHARE_PAIR` constraint:

```
SHARE_PAIR  
VDD pad_1 pad_5 pad_6 Bump_5 Bump_6  
END SHARE_PAIR
```

6. Bump\_7 and Bump\_8: Have a `SHARE_PAIR` constraint as below and Bump\_8 also has the port number property:

```
SHARE_PAIR  
VDD pad_7 pad_8 Bump_7 Bump_8  
END SHARE_PAIR
```

In this case, `viewBumpConnection` displays flightlines as follows:

- Does not display the connection of Bump\_1.
- Displays the connection between pad\_2 and Bump\_2 based on the `SHARE_PAIR` constraint.
- Displays the connection of Bump\_3 based on the port number property.
- Displays the connection of Bump\_4 based on only the port number property.
- Displays the connections based on `viewBumpConnection` auto-pairing results for pad\_1, pad\_5 and pad\_6 with Bump\_5 and Bump\_6.
- Displays the connection of Bump\_8 based on only the port number property. For Bump\_7, ignores the `SHARE_PAIR` constraint and does not display the connection of Bump\_7. This means if one or more bumps in a `SHARE_PAIR` constraint have the port number property, the other bumps without port number property in this `SHARE_PAIR` constraint are ignored and their connections are not displayed.

Note that the `SHARE_PAIR` constraint does not support the following scenarios and will treat the last pairing as the available one.

### **Scenario #1**

```
SHARE_PAIR  
VDD pad_1 Bump_1  
END SHARE_PAIR
```

```
SHARE_PAIR  
VDD pad_1 Bump_2 # This constraint works  
END SHARE_PAIR
```

If you want to pair pad\_1 to Bump\_1 and Bump\_2, use the following syntax:

```
SHARE_PAIR  
VDD pad_1 Bump_1 Bump_2  
END SHARE_PAIR
```

## **Scenario #2**

```
SHARE_PAIR  
VDD pad_1 Bump_1  
END SHARE_PAIR  
  
SHARE_PAIR  
VDD pad_2 Bump_1 # This constraint works  
END SHARE_PAIR
```

If you want to pair pad\_1 and pad\_2 to Bump\_1, use the following syntax:

```
SHARE_PAIR  
VDD pad_1 pad_2 Bump_1  
END SHARE_PAIR
```

## Long Pin Connection Display

Normally, `viewBumpConnection` uses the center of the pin of an IO instance for the connection display between the IO instance and a bump. However, in case of a long pin with multiple connections, using the center of the long pin geometry for the connection display may be confusing when you check the connection. From the 15.1 release, `viewBumpConnection` has been enhanced to display a long pin connection with the correct location, instead of the center of the pin, by default for both signal and power pins.

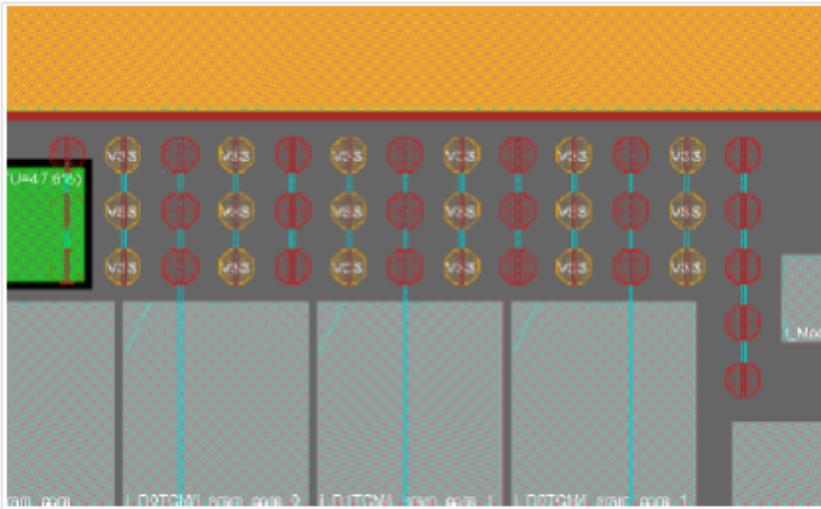
## Power Planning in Flip Chip Design

The general power planning step does not differ from a non-flip-chip design. However, there are flow recommendations worth mentioning.

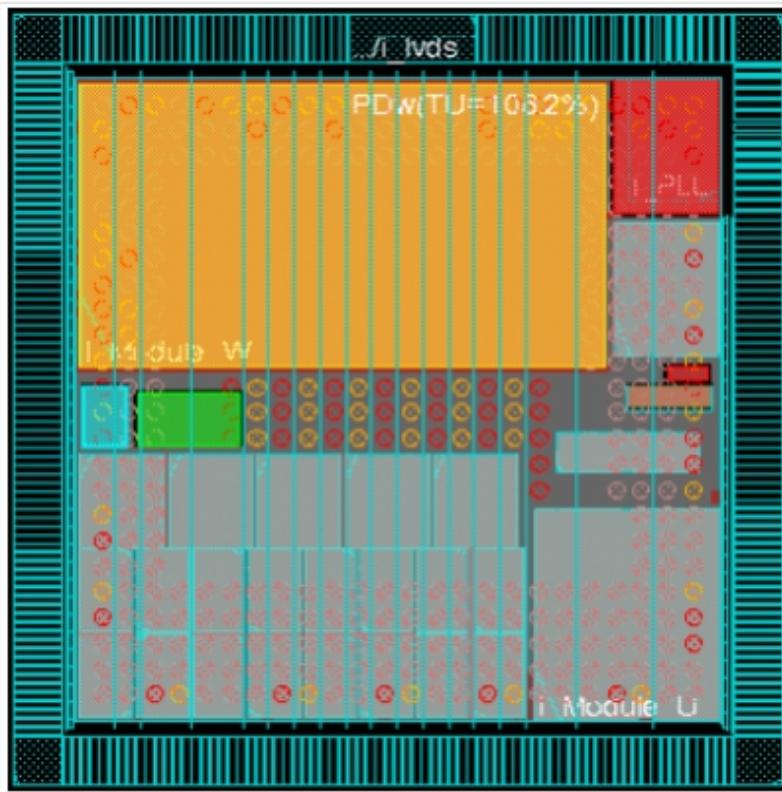
It is advisable NOT to use the RDL layer before pad to bump routing, as the flip chip router is very sensitive to routing obstructions. The power routing, i.e. stripe generation, with the RDL layer can be done after routing of I/O pads to bumps.

The `addStripe` command provides two options for easier flip chip design support. These options are:

- -over\_bumps
  - This option will create a power stripe over PG bumps.
  - The stripe generation will STOP at the end of a valid PG bump.
  - The image below shows power stripes in metal 8 generated over bumps in metal 9. The command will automatically drop the generated vias if bumps do not have OBS to prevent via under bump as defined in LEF.



- -between\_bumps
  - This option will create a power stripe between PG bumps.
  - The stripes will NOT be created in areas where there are no PG bumps.
  - The image below shows the result of running the `addStripe` command with the `-between_bumps` option. Notice the missing stripes where there are no PG bump.



These options allow for the possibility of providing an area (rectangular or rectilinear) for stripe routing. This is very flexible and improves the usability of the command when there are multiple power domains in your design.

At this stage, it is not recommended to route/create the connection between power bumps and power routing/power pads using the RDL layer. This step will be performed during or as a post step of the signal routing.

#### Notes:

- The `addStripe` command creates stripes over unassigned bumps. It is recommended that after pad and bump optimization, you delete all the spare bumps. `verifyGeometry` will not highlight these shorts as violations as the bump is not assigned.
- If you are planning to create power/ground stripes in the same layer as the RDL routing, it is recommended to do this after the signal routing by `fcroute`.

## RDL Routing

## Introduction

The flip chip router (`fcroute`) is mainly used for routing the nets between bumps and I/O PADs.

The `fcroute` command supports two types for routing:

- `fcroute -type power`
  - The `power` type can connect PG bumps to rings or stripes.
  - It only supports Manhattan routing style.
  - It does not honor the setting specified with the command `setFlipChipMode`.
- `fcroute -type signal`
  - The `signal` type can connect bumps to I/O pads.
  - The router under this type supports two routing styles, Manhattan and 45-degree routing. 45-degree routing is the default routing style.
  - It honors all the settings specified with the command `setFlipChipMode`.
  - Used with the command `setFlipChipMode -connectPowerCellToBump true`, it can route PG bumps to I/O pads.
  - Used with the command `setFlipChipMode -prevent_via_under_bump true`, it can prevent via generated on the bump.

The command `fcroute -type signal` accepts two design styles, peripheral IO (PIO) and area IO (AIO). However, it is not limited by design styles, which means that you can use the AIO mode for some specific purpose in a PIO design.

**Note:** In this document, the AIO or PIO mode is used to describe which design style is specified for `fcroute`.

Here are some examples of `fcroute` usage:

- `fcroute -type signal -designStyle aio`

This command will use only the detail router to finish the connections between bumps to IO pads and it routes nets one by one, so it is incremental.

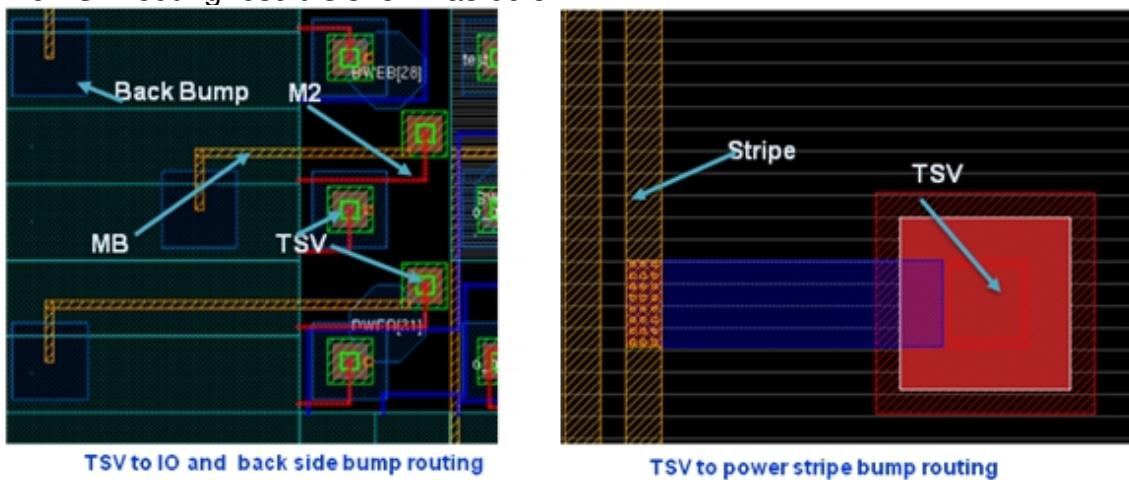
- `fcroute -type signal -designStyle pio`

This command will first use the global router to distribute all routing resource and then use the detail router to finish the connections between bumps to IO pads followed by a post processing step to finalize routing.

- `fcroute -type signal -designStyle pio -area {x1 y1 x2 y2} -incremental`  
`fcroute` partially supports area-based routing in the specified coordinates for the area in which the net is routed. The `-incremental` option is required for area-based PIO mode routing.
- `fcoute` can support TSV routing, which can route TSV to bumps/stripes/instance pins. The related commands are listed as below:
  - Connect TSV to bumps: `fcroute -type signal -connectTsvToBump`  
 As TSV has front-side and back-side bumps, `fcroute` should route them separately. Add `srouteExcludeBumpType bump_cell_name` in the extra configuration file to specify which bumps should be excluded for routing by `fcroute`.
    - When connecting TSV to the back-side bump, use the extra configuration file to exclude the front bump.
    - When connecting TSV to the front-side bump, exclude the back-side bump.

- Connect TSV to I/O pads: `fcroute -type signal -connectTsvToPad`
- Connect power TSV to stripes: `fcroute -type power -connectTsvToRingStripe`

The TSV routing result is shown as below:



The flip chip router (`fcroute`) is an intelligent and predictable RDL router and it can help you evaluate floorplan change based on the quick flip chip routing result. In the initial floorplan stage, you can use the following general setting to get a quick routing result:

## To set routing layers

- Use the `setFlipChipMode` command to set routing layers:

```
setFlipChipMode -layerChangeBotLayer bot_layer_name -layerChangeTopLayer  
top_layer_name
```

- Use the `fcroute` command to set.

```
fcroute -layerChangeBotLayer bot_layer_name -layerChangeTopLayer top_layer_name
```

## To set routing width

- Use the `setFlipChipMode` command to set.

```
setFlipChipMode -routeWidth value
```

- Use the `fcroute` command to set.

```
fcroute -routeWidth value
```

You can use the following commands with the above setting:

- Use `setFlipChipMode` for settings and then run `fcroute` for routing:

- Specify the routing setting for `fcroute` as follows:

```
setFlipChipMode -layerChangeBotLayer bot_layer_name -layerChangeTopLayer  
top_layer_name -routeWidth value
```

- Get the routing setting for `fcroute` to make sure all the settings are as expected:

```
getFlipChipMode
```

- Run `fcroute` with PIO mode as an example:

```
fcroute -type signal -designStyle pio
```

- Use `fcroute` for both setting and routing:

- Get routing setting before routing to make sure all the settings are as expected:

```
getFlipChipMode
```

- Run `fcroute` with the PIO mode with general settings as an example:

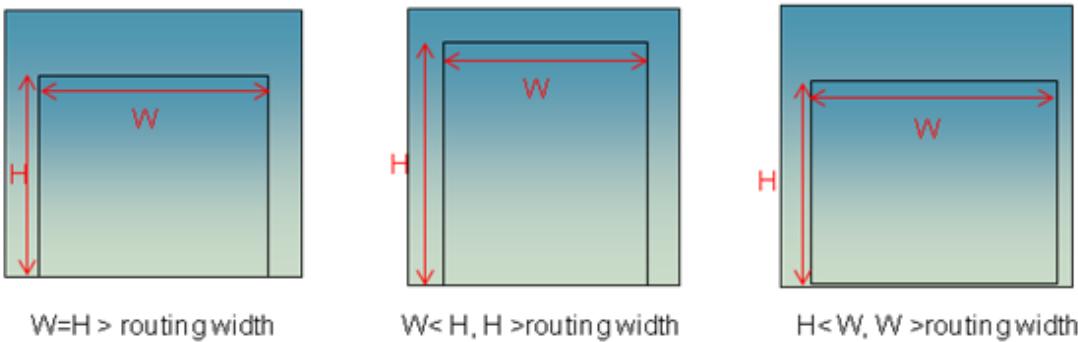
```
fcroute -type signal -designStyle pio -routeWidth value -layerChangeBotLayer  
bot_layer_name -layerChangeTopLayer top_layer_name
```

**Note:** The differences between the settings for `setFlipChipMode` and `fcroute` are:

- All the settings set by `setFlipChipMode` can be saved and restored by `saveDesign/restoreDesign`. These settings can work on `fcroute` if they are not reset or overwritten by the options of `fcroute`.
- All the settings set by `fcroute` cannot be saved after running `fcroute`. In addition, these settings work only when they are used in `fcroute`.

Generally, `fcroute` can well handle typical pin shape shown as below:

- Only one geometry for `fcroute`
- Width/height of pin geometry will not be very different from each other.
- At least one of width/height is larger than the routing width.



The above is the basic function of `fcroute`, and it is greatly extended by constraints and extra configuration options for customized requirements.

- All constraints are specified in a text file, which can be specified as input to the command `fcroute -constraintFile` or `setFlipChipMode -constraintFile`.
- All extra configuration options are specified in a text file, which can be specified as input to the command `fcroute -extraConfig` or `setFlipChipMode -extraConfig`.

Some useful constraints and extra configuration options for `fcroute` will be described in detail in the next two sections.

## Useful Constraints for Flip Chip Routing

The flip chip router (`fcroute`) provides you the capability to specify routing constraints in a text file, which can be specified as input to the command `fcroute -constraintFile` or `setFlipChipMode -constraintFile`.

**Note:**

- All length constraints take micron as the unit.
- All resistance constraints take ohm as the unit.
- All values can of floating type.

The routing constraints supported by `fcroute` are as follows:

## Global Constraints

You can specify general constraints affecting all the nets in the design. These general constraints need to be declared at the top of the file. The accepted constraints are as follows:

- `WIDTH value`
  - It specifies the global width for all the nets.
  - The unit is micron.
  - Both AIO and PIO modes support this constraint.
  - Both Manhattan and 45-degree routing support this constraint.
  - If you also specify routing width in the command `setFlipChipMode` or `fcroute`, `fcroute` will use the width specified in the command.
  - It can also be applied into the `NETS` constraint, which is a local constraint. In this case, the specified width value only works on the associated nets in the `NETS` constraint.
- `WIDTRANGE min_value:max_value`
  - It only works for PIO modes.
  - It specifies a width range and allows `fcroute` to optimize wire width by considering `MAXRES` constraints.
  - The unit is micron.
  - It can also be applied into the `NETS` constraint, which is a local constraint. In this case, the specified width range only works on the associated nets in the `NETS` constraint.
- `MAXRES value`
  - It specifies the maximum resistance allowed to all the specified nets.
  - The unit is ohm.
  - Only the PIO mode supports this constraint.
  - Both Manhattan and 45-degree routing support this constraint.

- The command `fcroute` can report the resistance with the setting `srouteFCReport res_file_name` in the extra configuration file.
- It can also be applied into the `NETS` constraint, which is a local constraint and the specified resistance only works on the associated nets in the `NETS` constraint.
- `SPACING value`
  - It specifies the distance in microns between the net routed by the flip chip router and all other routes.
  - The unit is micron.
  - Both the AIO and PIO modes support this constraint.
  - Both Manhattan and 45-degree routing support this constraint.
  - The router uses the spacing value to limit the effect of coupling capacitance on the total capacitance of the net.
  - It can also be applied into the `NETS` constraint, which is a local constraint and the specified spacing only works on the associated nets in the `NETS` constraint.
- `PIOLAYERCHANGE PAD`
  - It turns on the layer change feature.
  - Only the PIO mode supports this constraint.
  - Both Manhattan and 45-degree routing support this constraint.
  - Different setting for routing layers will have the different behaviors as shown in the following examples. Assume the top RDL is `TOP_RDL` and the second top RDL is `2nd_RDL`.
    - Use `TOP_RDL` as much as possible and `2nd_RDL` is used only when a single layer cannot finish routing in case of cross-over.
      - `layerChangeBotLayer TOP_RDL -layerChangeTopLayer TOP_RDL`
    - Freely change layers so that the tool will use the layer resources by its intelligence.
      - `layerChangeBotLayer 2nd_RDL -layerChangeTopLayer TOP_RDL`
- `FINDPINLAYERS layer_name`
  - It specifies the layer of the pin geometry for connection.
  - Both the AIO and PIO modes support this constraint.
  - Both Manhattan and 45-degree routing support this constraint.

- **MINPINSIZE** *width height*
  - It specifies the minimum geometry of the pin for connection.
  - The unit is micron.
  - Both AIO and PIO modes support this constraint.
  - Both Manhattan and 45 degree routing support this constraint.

## SPLIT Constraint

The **SPLIT** constraint is used to split wires when the routing width is larger than **MAXWIDTH** defined in LEF or the value specified by the user. The **SPLIT** constraint works for the AIO and PIO modes. In addition, it supports both Manhattan and 45-degree routing. The syntax of the **SPLIT** constraint is as follows:

```
SPLITSTYLE RIVER|MESH  
SPLITWIDTH value  
SPLITGAP value  
SPLITKEEPTOTALWIDTH TRUE|FALSE
```

## Parameters

SPLITSTYLE RIVER|MESH

- It specifies the interleaving style used for the split wires.
- The default value is RIVER.
- RIVER: The split wires do not have an interleaving pattern.



- MESH: The split wires interleave with one another, as shown in the following illustration:



SPLITWIDTH *value*

- It specifies the width of the split wire segment. If routing width is larger than this value, `fcroute` will auto split them.
- The unit is micron.
- Default value: `MAXWIDTH` defined in LEF.

`SPLITGAP value`

- It specifies the gap between split wire segments.
- The unit is micron.
- If you specify this constraint, ensure that the distance between the split wire segments must be greater than the specified gap value.
- If you do not specify this constraint, the distance between split wire segments is the default minimum spacing value that does not cause DRC violations.

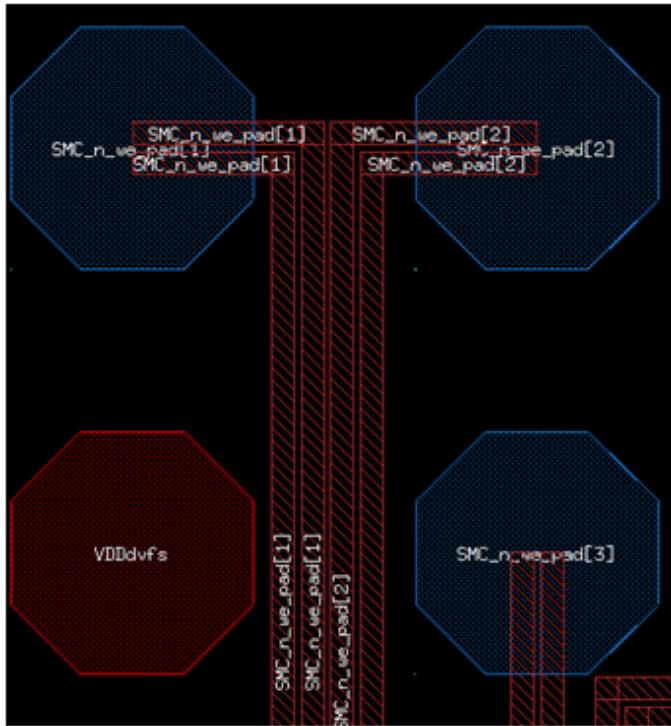
`SPLITKEEPTOTALWIDTH TRUE | FALSE`

- It is used to control different formula to compute the width of splitting wires.
- If the value is `FALSE`:
  - This is the default value.
  - The splitting wire width is calculated using the following formula:  
$$\text{Total\_wire\_width} = \text{split\_wire\_width} \times n + \text{split\_gap} \times (n - 1)$$
  - For example, routing width=13, `MAXWIDTH` in LEF=10, `split_gap`=1.5.  
Therefore, `split_wire_width=(13-1.5)/2=5.75`
- If the value is `TRUE`:
  - The splitting wire width is calculated using the following formula:  
$$\text{Total\_wire\_width} = \text{split\_wire\_width} \times n$$
  - For example, routing width=13, `MAXWIDTH` in LEF=10, `split_gap`=1.5.  
Therefore, `split_wire_width=13/2=6.5`
  - The difference from `FALSE` is to take the split gap out of the wire width calculation so that splitting wire width can be controlled by the user.

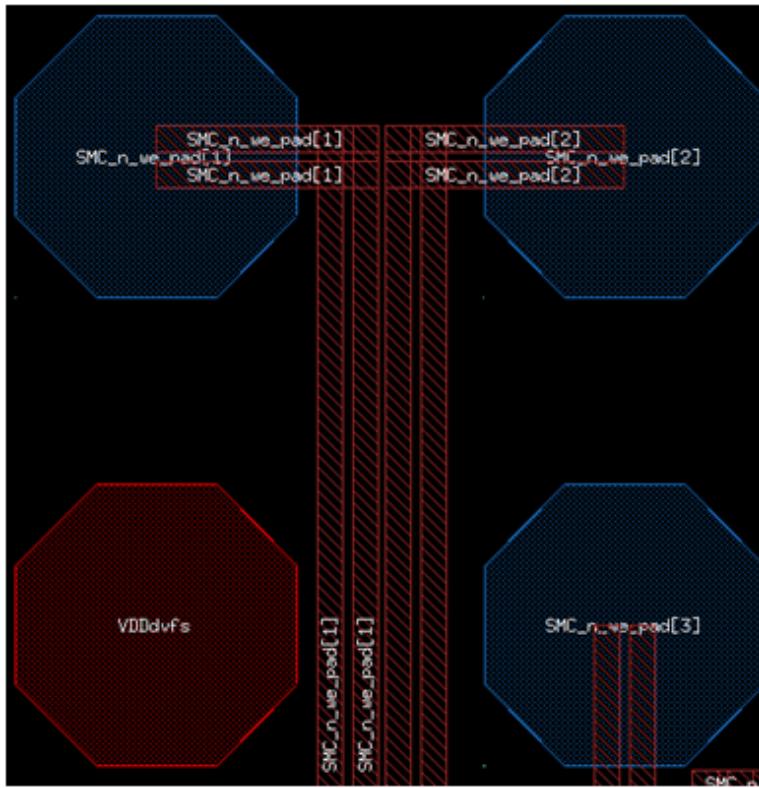
It can also be applied into the `NETS` constraint, which is a local constraint. In this case, the `SPLIT` feature works only on the associated nets in the `NETS` constraint.

Examples of different split styles are given below:

- `SPLITSTYLE RIVER`



- SPLITSTYLE MESH



## NETS Constraint

You can use the `NETS` constraint to set some specific constraint for nets. It can work for the AIO and PIO modes. It supports both Manhattan and 45-degree routing.

The syntax of the `NETS` constraint is as follows:

```
NETS
    WIDTH value
    ROUTELAYERS bottom_layer:top_layer
    SPACING value
    net_name_list
END NETS
```

## Parameters

- `WIDTH value` and `SPACING value` are the same as the one for Global Constraints.
- `ROUTELAYERS bottom_layer:top_layer`
  - It specifies a layer range for routing.
  - The name of layer could be specified the layer number or the layer name in LEF.  
For example, `bottom_layer` is `metal7` (`METG2` in `LEF`) and `top_layer` is `metal8` (`METTOP` in `LEF`).  
Following usage is acceptable for `fcroute`.

```
ROUTELAYERS 7:8;
ROUTELAYERS metal7:metal8;
ROUTELAYERS METG2:METTOP// This is recommended usage.
```
- `<net_name_list>`
  - It specifies the names of nets.
  - It supports general matching methods as below:
  - It supports “~”, which negates the specified net.
  - It supports wildcard matching(\*)
  - It supports `@SIGNAL`, `@POWER`, `@GROUND`:
    - `@SIGNAL` means all signal nets;

- @POWER means all power nets;
- @GROUND means all ground nets.

## Change Pin Access Direction

You can use the `PADACCESSDIR` soft constraint under `NETS` in the routing constraint file to change pin access direction. This is a soft constraint:

```
PADACCESSDIR { FROMCORE | FROMDIEBOUNDARY | EAST | WEST | SOUTH | NORTH }
```

You can set `FROMCORE` or `FROMDIEBOUNDARY` exclusively to set the preferred pin access direction from the core and from the die boundary, respectively. For example, `FROMCORE` means from the West side for I/O pad pins on the right side. Similarly, `FROMDIEBOUNDARY` means from North for the top side.

You can also set one of `EAST | WEST | SOUTH | NORTH` directions. `EAST`, `WEST`, `SOUTH` and `NORTH` directions represent the 'from' direction to the pin. For example, if you want `fcroute` to access pins of I/O pads on the West side (left side) from the center, you can set `PADACCESSDIR` to either `FROMCORE` or `EAST`.

Here is an example of using `PADACCESSDIR`:

```
NETS
PADACCESSDIR EAST
    tdigit[5]
END NETS
```

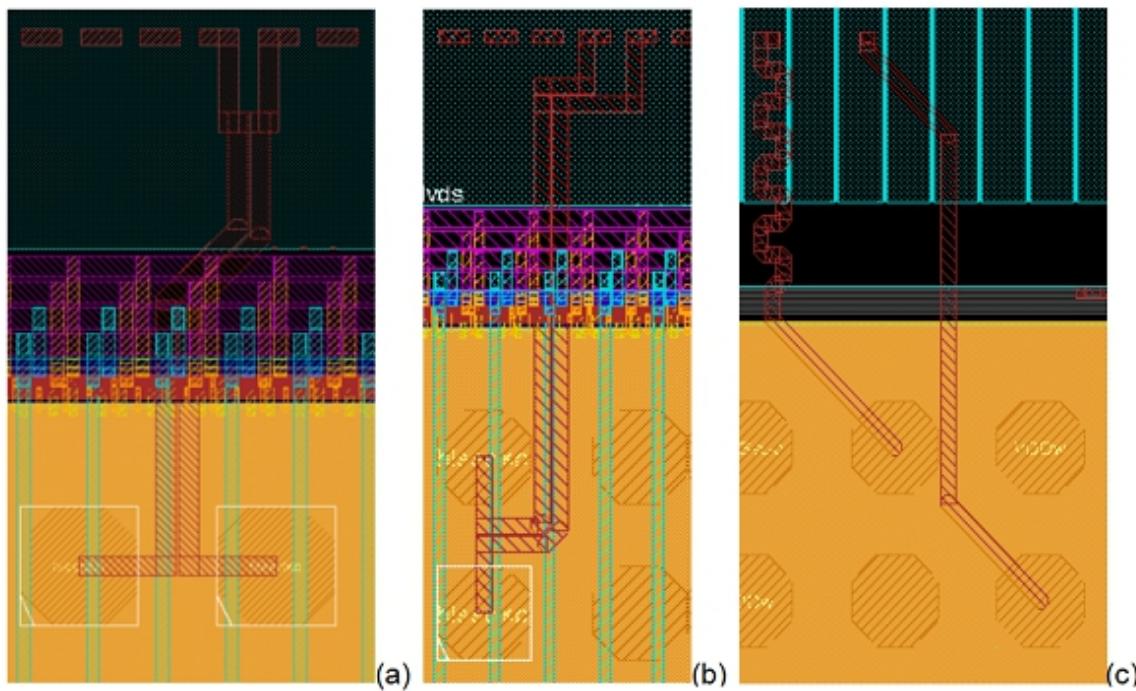
**Note:** This is a soft constraint.

## Differential Routing Constraint

Differential routing is aimed at providing similar net delays and it applies to pairs of nets. It only works for the AIO mode and also it supports both Manhattan and 45-degree routing.

To route these nets, `fcroute` will try,

1. Balance pair routing (matching routing length), if it fails then
2. Topological pair routing, if it fails then
3. Match L/W routing



The same constraints have been used to create the above three routing results, the differences were on the nets selected for routing. In these routing examples:

- Picture (a) has balanced routing matching the length and topology of both nets.
- Picture (b) cannot match the length but keeps the topological matching.
- Picture (c) cannot match the topological so `fcroute` matches the length and the width of both nets.

All these methodologies guarantee a similar net delay.

## Syntax

```
SHARE_DIFFPAIR
  REL_DIFFERENCE value
  ABS_DIFFERENCE value
  DPAIRGAP value
  net_1 net_2
```

```
END SHARE_DIFFPAIR
```

Here:

- `SHARE_DIFFPAIR` in constraint file is supported in both the `AIO` and `PIO` modes by default.

- `REL_DIFFERENCE value` is used to check whether the relative difference in length of the two nets in the differential after routing meets the specified constraint value.
  - The default value is `0.2`.
  - The relative difference is calculated by using the following formula:

$$\text{REL\_DIFFERENCE} = \frac{\text{Length of the longer net} - \text{Length of the shorter net}}{\text{Length of the shorter net}}$$

For example, if the length of the constrained nets are 24 microns and 20 microns after routing, the relative difference is calculated as  $(24 - 20) / 20 = 0.2$ .

- `ABS_DIFFERENCE value` is used to check whether the absolute difference in length of the two nets in the differential pair after routing meets the constraint.
  - The default value is `40`
  - The unit is microns.
  - The absolute difference is calculated by following formula:

$$\text{ABS\_DIFFERENCE} = \text{Length of the longer net} - \text{Length of the shorter net}$$

For example, if the lengths of the constrained nets are 24 microns and 20 microns after routing, the absolute difference would be calculated as  $(24 - 20) = 4$ .

- `DPAIRGAP value` is used to control the spacing between the two nets in the differential pair.
  - The default value is the minimum spacing value in the LEF file.
  - The unit is microns.

Any violation in `REL_DIFFERENCE` or `ABS_DIFFERENCE` is reported in `srouteFcReport`.

The `SHARE_DIFFPAIR` constraint can be used with resistance constraints if both nets in the `SHARE_DIFFPAIR` are restrained by the same `MAXRES` and `WIDTHRANGE` constraints.

NETS

`MAXRES value`

`WIDTHRANGE min_width: max_width`

`net_list`

END NETS

SHARE\_DIFFPAIR

```
REL_DIFFERENCE value
ABS_DIFFERENCE value
DPAIRGAP value
net_1 net_2
END SHARE_DIFFPAIR
```

If the two nets in a `SHARE_DIFFPAIR` are restrained by different `MAXRES` and `WIDTHRANGE` constraints, an error occurs and only the `SHARE_DIFFPAIR` constraint is applied on those two nets.

**Note:** The two nets in a `SHARE_DIFFPAIR` have the same width after the resistance-driven routing feature is applied.

## Example

```
SHARE_DIFFPAIR
REL_DIFFERENCE 0.2
ABS_DIFFERENCE 4
port_pad_data_out[7]
port_pad_data_out[8]
END SHARE_DIFFPAIR
```

## Match Routing Constraint

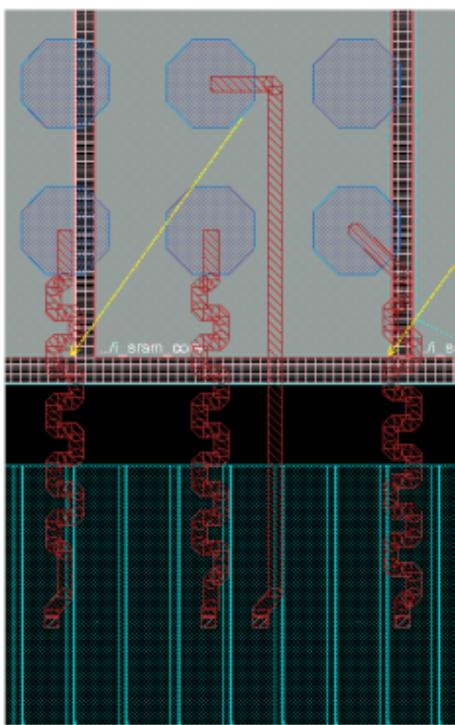
For a group of nets (more than two), the router tries to match routing length. It only works for the AIO mode. It also supports both Manhattan and 45-degree routing.

The syntax of `MATCH` constraint is as follows:

```
MATCH
TOLERANCE value
<2 or more nets>
END MATCH
```

Here, `TOLERANCE` specifies the tolerance value for differential routing. `<2 or more nets>` specifies the nets for which differential routing is done.

In the picture below, the user has selected four nets to `MATCH` the length of the routing.



## Shielding Routing Constraint

The flip chip router supports the capability of shielding nets during routing. To enable the feature, the recommendation is to use the constraint file. Both the AIO and PIO modes support this feature. In addition, both Manhattan and 45 degree routing support it.

The syntax of SHIELDING constraint is as below:

```
SHIELDING
  SHIELDBUMP TRUE | FALSE
  SHIELDWIDTH value
  SHIELDGAP value
  SHIELDSTYLE a | b | c
  SHIELDNET net_name
  net_name_list
END SHIELDING
```

## Parameters

`SHIELDBUMP TRUE | FALSE`

- If the value is `TRUE`
  - Shields bump with the specified shield net.
  - It only works in the `AIO` mode and Manhattan routing style.
- If the value is `FALSE`
  - Does not shield bump.
  - It is the default value for `SHIELDBUMP`.

`SHIELDWIDTH value`

- It specifies the width of the *Shield Net*, measured in microns.

`SHIELDGAP value`

- It specifies the distance in microns between the shield (the special net) and the shielded net (the signal net).

`SHIELDSTYLE a | b | c`

- It specifies where you want the shield to be placed, *Above* or *Below* or on the *Common* layer.
  - a = above the layer containing bumps
  - b = *below* the layer containing bumps
  - c = on the layer containing bumps ("common layer")

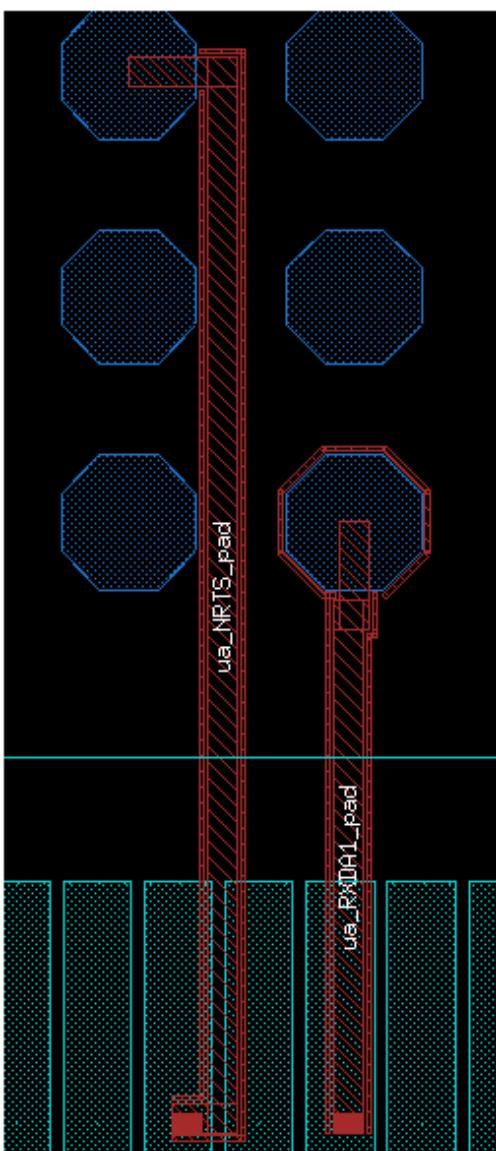
`SHIELDNET net_name`

- It specifies a special net (typically VSS) used to shield the net.

`net_name_list`

- It specifies the shielded nets.

In the picture below, the left net does not have `SHIELDBUMP` and the right net can be achieved by using `SHIELDBUMP TRUE`.



## Notes

- The shielding created will be floating. You will need to connect the shielding to the correct power/ground supply by using the `editPowerVia` command.
- `defOut` can mark the shielded nets as `SHIELD`, while displaying the `SHAPE` and `ROUTED` status of the metal shield wire. See the following example.

## Example

Consider the following example in which the `fcroute` command connects signal bumps to I/O cells using 90-degree signal routing for AIO. The command adds a side shield (VSS) on both sides of the signal route.

```
setFlipChipMode -route_style manhattan
fcroute -type signal -designStyle aio -layerChangeTopLayer 8 -layerChangeBotLayer 7 -
routeWidth 8 -constraintFile CFG/aio.constr
```

### Constraint File CFG/aio.constr: Shield Net Description

SHIELDING

```
SHIELDBUMP true
SHIELDWIDTH 0.4
SHIELDLAYERS abc
SHIELDNET VSS
scan_out_2
port_pad_data_out[15]
```

END SHIELDING

## DEF Syntax

`defOut` contains the `SHIELD` syntax as follows:

```
- scan_out_2 ( Bump_27_6_2 PAD ) (IOPADS_INST/Pscanout2op PAD)
+ ROUTED METAL8 16000 + SHAPE IOWIRE (1255310 541920 ) ( 1369310 *)
NEW METAL8 16000 + SHAPE IOWIRE (1263310 533920 ) ( * 695760 )
+ PROPERTY BUMP_ASSIGNMENT "ASSIGNED"
;
-VSS ( * VSS )

+ SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275310 554320 ) ( 1315310 * )
NET METAL7 16000 + SHAPE IOWIRE ( 1255310 541920 ) ( 1315310 * )
NET METAL8 800 + SHAPE IOWIRE ( 1250510 529520 ) ( 1315310 * )
+ ROUTED METAL6 16000 + SHAPE STRIPE ( 1553200 109600 ) ( * 186800 )
NET METAL6 16000 + SHAPE STRIPE ( 1753200 109600 ) ( * 186800 )
+ SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275710 553920 ) ( * 675760 )
```

```
METAL7 16000 + SHAPE IOWIRE ( 1263310 533920 ) ( * 695760 )  
METAL8 800 + SHAPE IOWIRE ( 1250910 529120 ) ( * 675760 )
```

## PAIR Constraint

The `SHARE_PAIR` constraint is used to control the pairing between bumps and pads, especially for PG connection because it is a common case for PG bumps and pads to have multiple bumps to multiple pads connection requirements and the user needs to define the pairing. Both AIO and PIO modes support this constraint. In addition, both Manhattan and 45-degree routing support it.

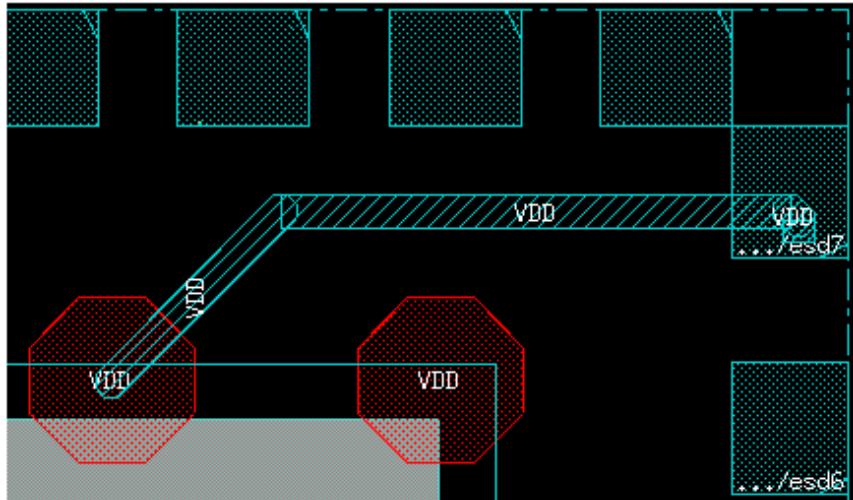
The syntax of the `SHARE_PAIR` constraint is as follows:

```
SHARE_PAIR  
net_name pad_name_list bump_name_list  
END SHARE_PAIR
```

## Parameters

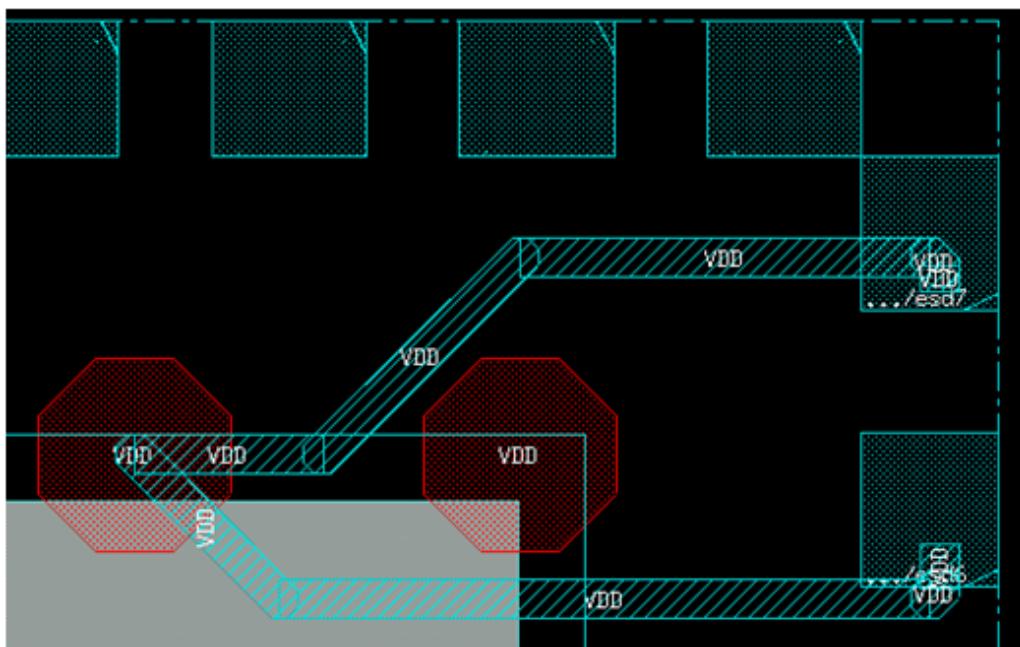
This constraint supports the following pairing pattern.

- One pad to one bump pairing



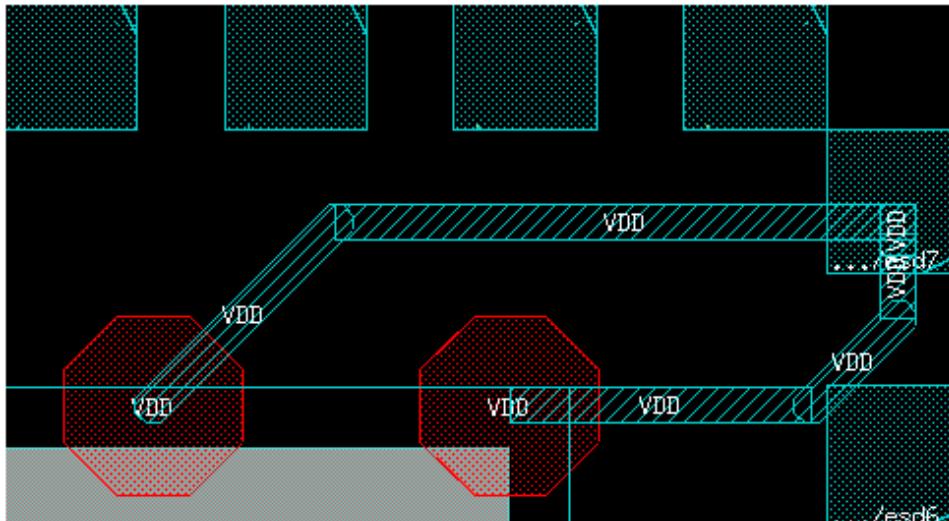
- Multiple pads to one bump pairing

Turn on this feature with the option `-multipleConnection multiPadsToBump` in `setFlipChipMode`.



- Multiple bumps to one pad pairing

Turn on this feature with the option `-multipleConnection multiBumpsToPad` in `setFlipChipMode`.



If the specified bump is not assigned or assigned to another net which is different from the specified net, `fcroute` will ignore it.

For those I/O pads or bumps not in the list but needed to be connected to the specified net, `fcroute` will ignore them and only route the specified ones by PAIR constraint for the same net.

## Resistance Driven Constraint

You may constrain the net resistance during routing. Both the AIO and PIO modes support this feature. In addition, both Manhattan and 45-degree routing support it.

The syntax of Resistance Driven Constraint is as follows:

```
NETS
```

```
    MAXRES value  
    WIDTHRANGE min_value: max_value  
    net_name_list
```

```
END NETS
```

- `MAXRES` is the maximum resistance value allowed for the nets defined in the constraint.
- `WIDTHRANGE` specifies a variable width. The router is allowed to use any width value between these two limits to avoid violating the max resistance.

It is recommended to add `srouteFCReport res_file_name` in the extra configuration file.

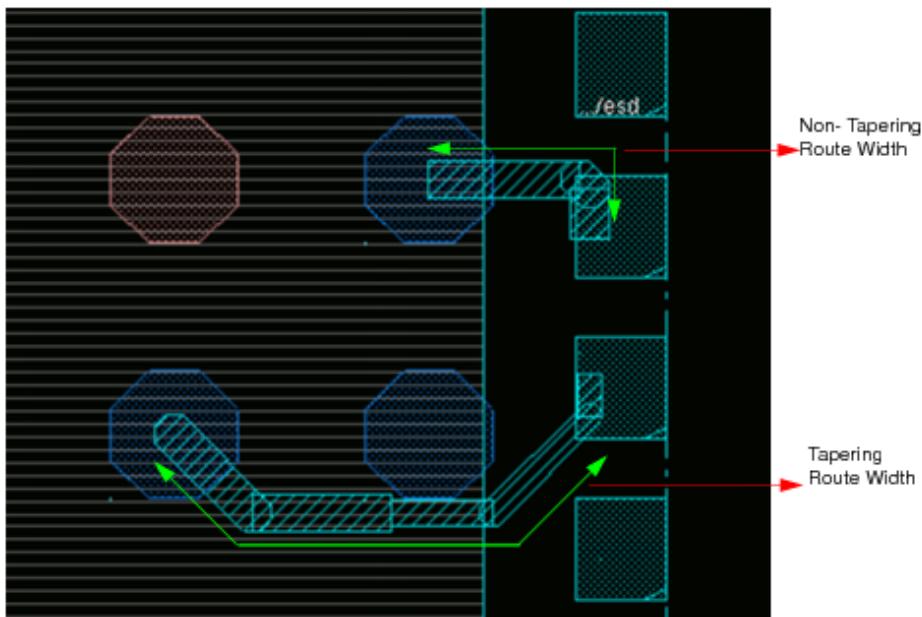
The resistance values achieved after routing will be written to this file. This file will report the resistances in a table as follows:

```
#=====  
#.....Resistance Table.....  
# resSQ: 0.100 for METAL8  
#=-----  
'int' 0.875  
...  
'port_pad_data_out[15]' Actual:3.159 Constraint:3.000
```

This information can be useful for debugging purposes. For accurate values of resistance it is recommended to run Quantus (Cadence Signoff Extraction Tool). Quantus can be run from Innovus.

## Tapering feature

Tapering feature is enabled in the area I/O mode, wherein fcroute uses a thin routing width on I/O pins and wide routing width on the bumps.



You can specify the Tapering constraint in the `fcroute` constraint file. The constraint syntax is as follows:

## Syntax

NETS

```
TAPERSTEP step_value
TAPERWIDTH width_value
<nets>
```

END NETS

Here:

- **TAPERSTEP** is the constraint to turn on tapering feature. It takes two values: 0 and 1.
  - 0: There is no tapering needed.
  - 1: Allow router taper once from normal routing width to the taper width at somewhere on the routing wires. The tapering point is determined by the tool and user cannot control it.
- **TAPERWIDTH** defines the wire width after tapering. It takes floating value and should be in the range of [MINWIDTH, normal\_width].
  - **MINWIDTH** is the minimum width allowed for the metal layer and has been defined in technology LEF file.
  - **normal\_width** is defined by the `-routeWidth` option in `fcroute` or routing width constraint

specified in constraint file.

## Example

```
NETS
TAPERSTEP 1      # 1 enables tapering, 0 disables tapering
TAPERWIDTH 10    # After tapering, the width would be 10
vssx_0          # Specifies the net name
vddcx_1          # Specifies the net name
END NETS
```

## Useful Extra Configurations for Flip Chip Routing

The flip chip router (`fcroute`) provides you the capability to specify extra options into a text file, which can be given as an input to the command `fcroute -extraConfig` or `setFlipChipMode -extraConfig`. Because of the rich patterns in flip chip routing and extensive customization of RDL routing, this file defines extra options in addition to general settings that you may need for RDL routing. You can choose the variables that can meet your requirements.

**!** *You should only use the extra configuration file if you are familiar with its use.*

The following variables can be used in the extra configuration file:

- `srouteBumpToBumpRoute` [TRUE | FALSE]  
Specifies bump to bump routing. The default is TRUE.
- `srouteBumpToWireShape` BLOCKWIRE  
Connects the net to block wires
- `srouteConnectToAnyOfBump` [TRUE | FALSE]
  - If it is set to TRUE, it means that it is unnecessary for `fcroute` to route to the center of bump but it is acceptable to touch the bump geometry anywhere.
  - The default value is FALSE.
- `srouteConnectToCenterOfPin` [TRUE | FALSE]

- If it is set to TRUE, `fcroute` can connect to the center of any pin except bump.
  - The default value is FALSE.
  - It is usually used when the width of the wire is less than the width of the pin.
- `srouteConnectToEdgeOfBump [TRUE | FALSE]`
    - The default is FALSE.
    - Connects a wire to the edge of the bump if it is set to TRUE.
  - `srouteDifferentialRouteTolerance value`

The default value is 2.

If the actual difference between the length/width ratios is less than the specified value, the software omits differential routing. If the actual difference is greater than the specified value, the software attempts differential routing, but if the software cannot meet the specified value, the software displays a warning message.
  - `srouteEcoMode [TRUE | FALSE]`

The default is FALSE. When you set `fcroute -eco` to enable the ECO mode for Flip Chip route, `srouteEcoMode` is set to TRUE in the extra configuration file.
  - `srouteExcludeBumpType bumpCellName`

Specifies which bump cell will be excluded during routing.
- Note:** Currently, `srouteExcludeBumpType` supports both power and signal bumps. Use `fcroute -type power ... -extraConfig config_file_name` to route certain types of bumps. The excluded types of bumps are defined with `srouteExcludeBumpType` in the config file.
- `srouteExcludeRegion "l1x1 l1y1 urx1 ury1 l1x2 l1y2 urx2 ury2 ..."`
    - It specifies the region to exclude bumps and pads from `fcroute`.
    - It takes a string quoted by “” and in the string, multiple rectangular shapes are defined even they are disjointed.
    - The format of the string is like below,  
`"rect_l1_x rect_l1_y rect_ur_x rect_ur_y [more_rects]"`
    - The unit is DB unit.
  - `srouteFcCompaction [TRUE | FALSE]`

- The default is FALSE.
- Turns on compaction routing, when set to TRUE . In the flip chip post-route stage, the compaction function pushes the routing as close as possible to bumps in order to leave more routing resource for subsequent routing steps, such as P/G bump connections or general power routing.

**Note:** This feature cannot be used with resistance driven and diffpair routings because the compaction action may change the wire length and routing pattern. This feature does not support two-layer RDL routing.
- `srouteFcDieAreaOffset "left bottom right top"`
  - Specifies the distance (in microns) to which the expanded area should extend from the edges of the chip. If specified, `fcroute` can finish the routing in this expanded die area instead of the actual die area. The default value is “ 0 0 0 0 ”. This option does not support negative values.

**Note:** Routing is not allowed outside the expanded die area. If `fcroute` cannot finish the routing to specific bumps in the expanded die area, it leaves these bumps open.
- `srouteFCReport file_name`
  - Writes the resistance report into the specified file using the resistance constraint MAXRES and the inputs. It also reports the width and length of special nets routed by `fcroute`.
- `srouteFcrMazeRoute45 [TRUE | FALSE]`

Enables maze routing for 45 degree style routing. Maze routing increases the final routing completion rate for `fcroute` and improves results for nets with complex patterns and long routing path.
- `srouteFcRouteAllowOverCongestion [TRUE | FALSE]`

Specifies that all nets are routed even if there is not enough spacing in some area. The default is FALSE.
- `srouteFCrouteLayerIsPreferred. n [TRUE | FALSE]`

Allows you to specify layer priorities for routing in a multi-layer design. For example, if you want M9 to be used only in very congested area, set `srouteFCrouteLayerIsPreferred.9` to FALSE to indicate that M9 is not a preferred layer for routing. Without any setting, every layer is considered a preferred layer, and the routing is distributed evenly.

**Note:** Currently, layer priority can be specified only in PIO flow.

- `srouteFcroutePadPinTagging [TRUE | FALSE]`
  - It enables global router to honor port numbering if it is set to TRUE.
  - The default value is FALSE.
- `srouteGrouteClusterRegions x1 y1 x2 y2 x3 y3 x4 y4 ...`

Allows you to specify the area I/O cluster regions in microns. Here, `x1 y1 x2 y2` coordinates are equivalent to the `llx, lly, urx, and ury` coordinates of the first area I/O cluster, `x3 y3 x4 y4` represent the second, and so on.

Default: " "
- `srouteGrouteIncremental {TRUE | FALSE}`

The default is FALSE.
- `srouteGrouteLengthDriven [TRUE | FALSE]`

The default is FALSE. When optimizing placement and bump assignment, bias to vertical / horizontal connection instead of 45-degree connection.
- `srouteGrouteMaxPathPerBump num_of_routes`

Specifies the maximum number of routes coming from a single bump. Use along with `srouteGrouteMaxPathPerPad` to control the maximum number of connections to a pin port for multiple bump routing. Here, `num_of_routes` has a range of 1 to 15.
- `srouteGrouteMaxPathPerPad num_of_routes`

Specifies the maximum number of routes to a single pad pin geometry. Use along with `srouteGrouteMaxPathPerBump` to control the maximum number of connections to a pin port for multiple bump routing. Here, `num_of_routes` has a range of 1 to 15.
- `srouteGrouteOptimizeWidth [TRUE | FALSE]`

The default is FALSE.

**Note:** You must use this option in conjunction with the `srouteGrouteOptimizeSpacing` option and they cannot be set to true at the same time.

Automatically adjusts the width of the nets specified in the constraint file within the max and min

constraint. When set to `true`, this configuration file option calls the global router to get the optimized width for each net and writes the results to a constraint file called `width.cons`. If it is an `fcroute -designStyle PIO`, the router routes the given nets as wide as possible (still within the width range) while maintaining routability.

You can create your own constraint file to specify how the width is calculated. For example, if you have 10 nets in the design (`n1` through `n10`), you can create the following constraint file:

```
NETS
MINWIDTH 2.0
MAXWIDTH 5.0
WIDTHSTEP 0.1
n1 n2 n3
END NETS
```

```
NETS
MINWIDTH 5.0
MAXWIDTH 10.0
WIDTHSTEP 0.5
n4 n5 n6
END NETS
```

```
NETS
WIDTH 3.0
n7 n8
END NETS
```

```
NETS
WIDTH 12
n9 n10
END NETS
```

where:

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <code>n1, n2, and n3</code> | Have a width between 2 and 5, and an incremental size of 0.1.  |
| <code>n4, n5, and n6</code> | Have a width between 5 and 10, and an incremental size of 0.5. |
| <code>n7 and n8</code>      | Have a fixed width of 3.0.                                     |

|            |                             |
|------------|-----------------------------|
| n9 and n10 | Have a fixed width of 12.0. |
|------------|-----------------------------|

**Note:** All the nets within the same NET group will use the same final width. If you allow the nets to have different widths, while satisfying the min and max value, you can use the following configuration file option:

`srouteGrouteUniformWidth FALSE`

- `srouteGrouteOptimizeSpacing [TRUE | FALSE]`

The default is `FALSE`.

**Note:** You must use this option in conjunction with the `srouteGrouteOptimizeWidth` option and they cannot be set to `true` at the same time.

Automatically adjusts the spacing within the max and min constraint. When set to `true`, FCroute calculates the maximum allowable spacing (for the given net width) and writes it to the log file. All spacing values are the same.

- `srouteGrouteSerialBumpRouting [TRUE | FALSE]`

Allows bumps of the same net to be connected together. The default is `FALSE`.

- `srouteGrouteUniformWidth [TRUE | FALSE]`

All nets in the same width group have uniform width. The default is `TRUE`.

- `srouteJogControl [TRUE | FALSE]`

Allows jogs during routing to avoid DRC violations. The default is `FALSE`.

- `srouteLayerChangeExcludeRegion "1lx1 1ly1 urx1 ury1 1lx2 1ly2 urx2 ury2 ... "`

Specifies the region to be excluded from layer change initiated by `fcroute`. In the argument string for this option, you can define one or more rectangular shapes even if they are disjointed. You can also define a rectilinear shape. The format of the argument string is as follows:

○ `"rect_ll_x rect_ll_y rect_ur_x rect_ur_y [more_rects]"`

or

○ `"rectilinear_x1 rectilinear_y1 rectilinear_x2 rectilinear_y2 ..."`

`srouteLayerChangeExcludeRegion "0 0 0 0"` means layer change is allowed on the whole chip. `srouteLayerChangeExcludeRegion` must be used with `PIOLAYERCHANGE PAD` in the constraint file. Only the PIO mode supports this constraint.

- `srouteLengthLimit [integer value]`  
Specifies the maximum routing wire length for each flip chip net. The default is 0.
- `srouteMinLength [integer value]`  
Specifies the minimum segment length. The default is 0.
- `srouteOutputFailedResistanceGroute [TRUE | FALSE]`  
Displays nets that failed maximum resistance with different colors. The default is FALSE.
- `sroutePinSpacing dbUnitValue`  
Defines the I/O pad pin spacing.
- `sroutePioBusRoute [TRUE | FALSE]`  
Routes only the nets and bumps defined in the NETGROUP constraint, when set to TRUE. The default is FALSE.
- `sroutePioDiffPair [TRUE | FALSE]`  
Supports differential pair (DIFF PAIR) routing in the PIO mode. The default is FALSE.
- `sroutePowerBumpAllDir [TRUE | FALSE]`  
Connects power bump to power stripes on all directions. The default is FALSE.
- `sroutePrevent45ForLowerLayer [TRUE | FALSE]`
  - It can prevent 45-degree routing for lower layer if it is set to TRUE.
  - The default value is FALSE.
- `sroutePreventViaUnderBump [TRUE | FALSE]`  
Prevents via under a bump. The default is FALSE.
- `sroutePushAndShove [TRUE | FALSE]`  
If TRUE, detail routing gets ripped and rerouted, to route open nets. The default is TRUE.
- `sroutePushAndShoveVerbose [TRUE | FALSE]`  
If TRUE, displays debugging messages during push\_and\_shove. The default is FALSE.
- `srouteReduceLayerChanges integer`  
The default is 0.

- `srouteRouteSpacing [integer value]`  
Specifies the default routing space for each flip chip net. The default is 0.
- `srouteRouteWidthForLowerLayer dbUnit`
  - Indicates that `fcroute` should use `dbUnit` as width for the lower layer.
  - For example, if DB unit is 2000 and the width for lower layer is 8 micro, then this value should be set as  $2000*8=16000$ .
- `srouteSpreadWiresFactor float`  
Specifies automatic spreading of wires during bump routing in order to prevent any SI violations. Here, `float` is applied as a multiple of the minimum spacing that is required.
- `srouteStraightRouteOnly {TRUE | FALSE}`  
Specifies straight routing connections between targets. The default is `TRUE`.
- `srouteUseSpecifiedWidthForTopLayer {TRUE | FALSE}`  
If `TRUE`, forces `fcroute` to use user-defined width for the top layer connecting to the bump. The default is `FALSE`.

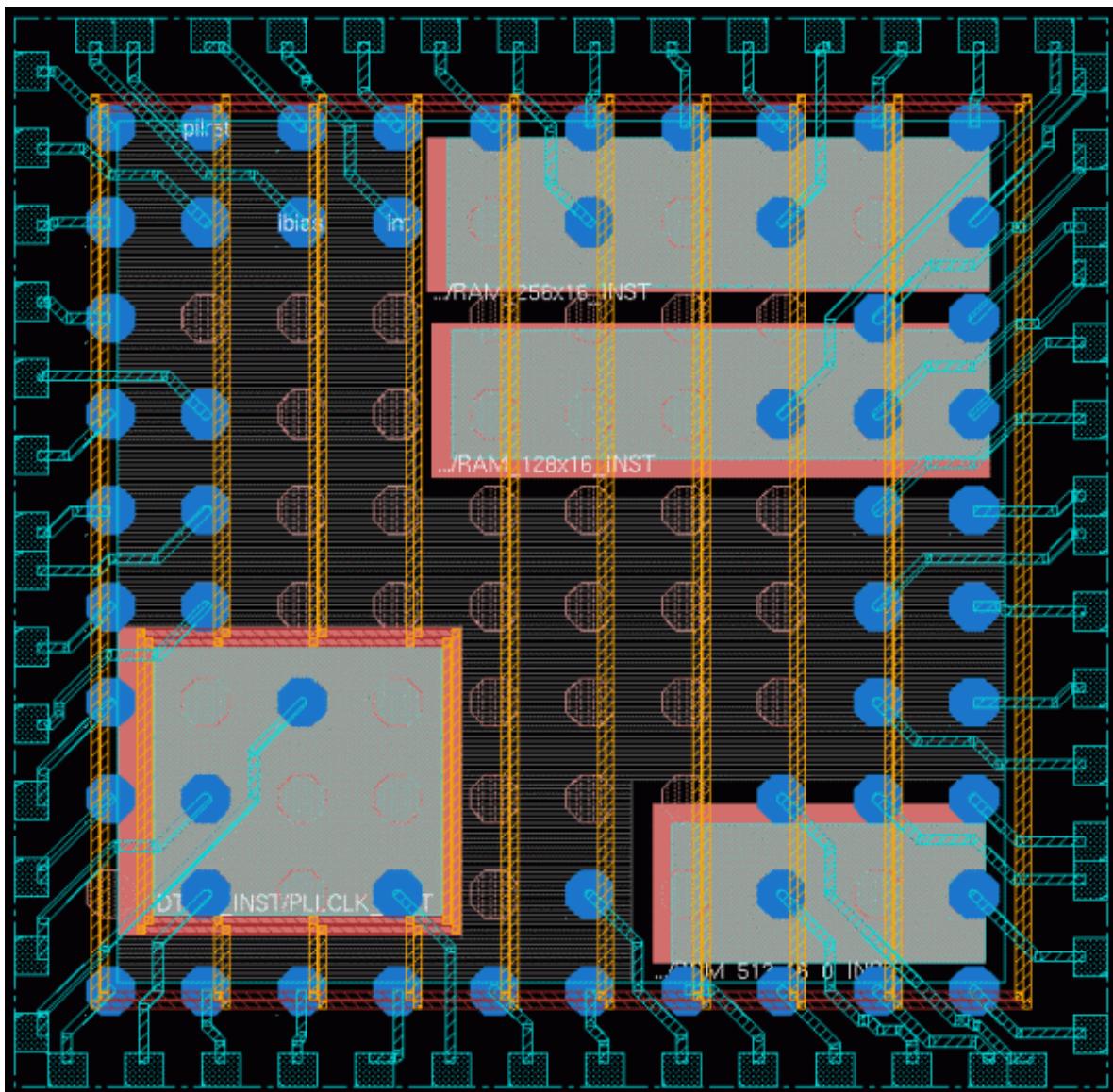
## Power Routing

There are two methodologies for connecting the power and ground bumps that can coexist in the same design:

- Connect PG bumps to the I/O pads.
- Connect PG bumps to rings or stripes.

### Connect PG Bumps to I/O Pads

It is recommended that these connections are done at the same time as the signal routing. First use the command `setFlipChipMode -connectPowerCellToBump true` to enable connecting power cells to bumps, and then use the command `fcroute -type signal` to route power cells to bumps.



**Note:** In a design, some Power/Ground (PG) bumps may be already connected to PG stripes before flip chip routing is done. From the 14.1 release, `fcroute` has been enhanced to route bumps with port numbers to I/O cells even if the bumps are already connected to Power/Ground stripes. The flip chip router now first checks whether port number is defined in bumps. If yes, `fcroute` routes the bumps to IO even if they are already routed by PG stripes.

The command `fcroute` will automatically pair PG bumps with PG pads based on routability with some intelligence if there are no specific pairing constraints by the user. Also `fcroute` will choose the suitable pin for routing based on its intelligence if there is no constraint by the user.

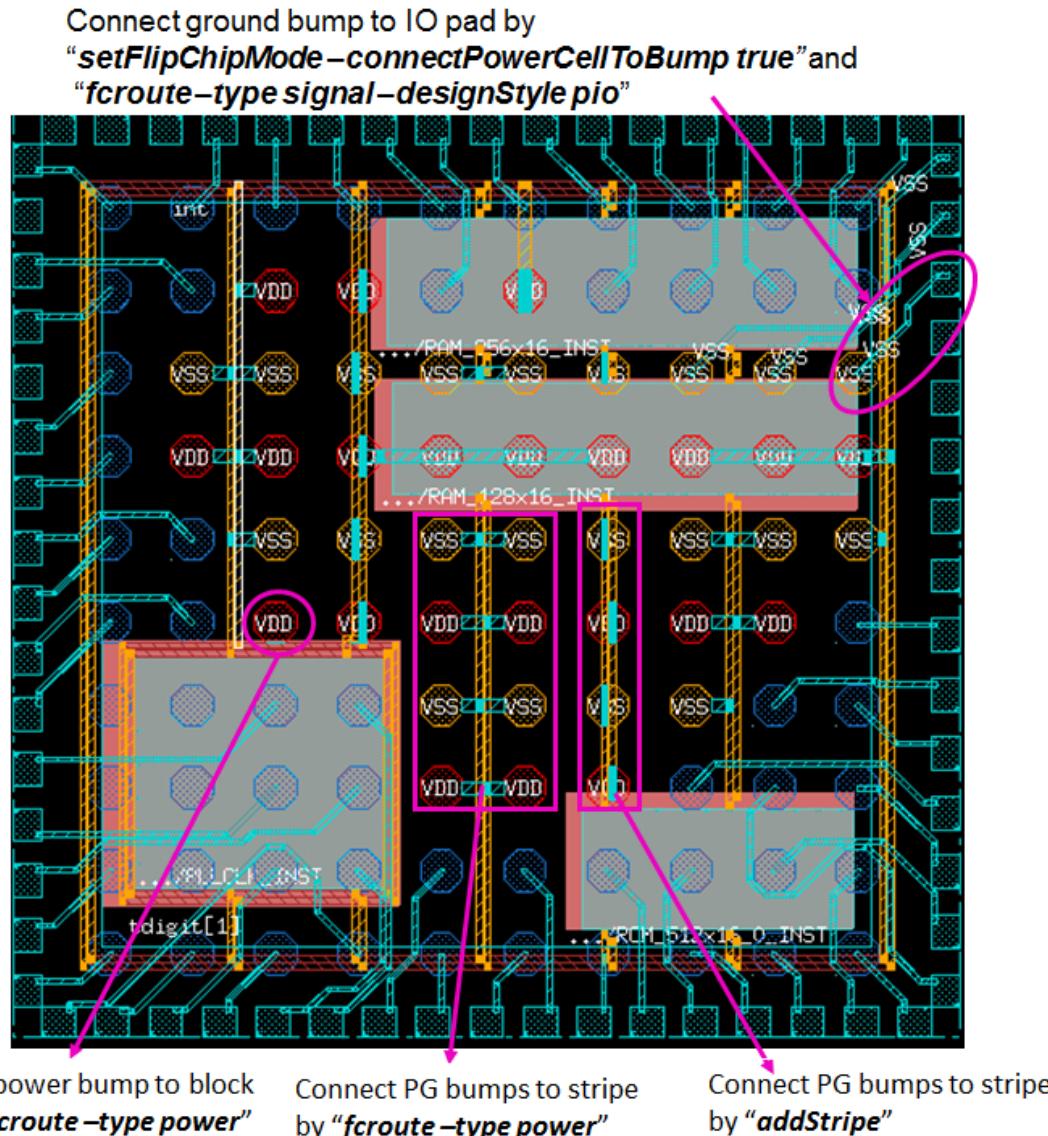
For more control of the connection between PG bumps and PG pads, the designer can use the `SHARE_PAIR` constraint in the constraint file or use port numbering feature.

## PG Bumps Connect to Rings or Stripes

The command `fcroute -type power` can connect PG bumps directly to rings or stripes.

You can create some extra power and ground bumps to reduce the IR drop. This is one of the advantages of the flip chip design.

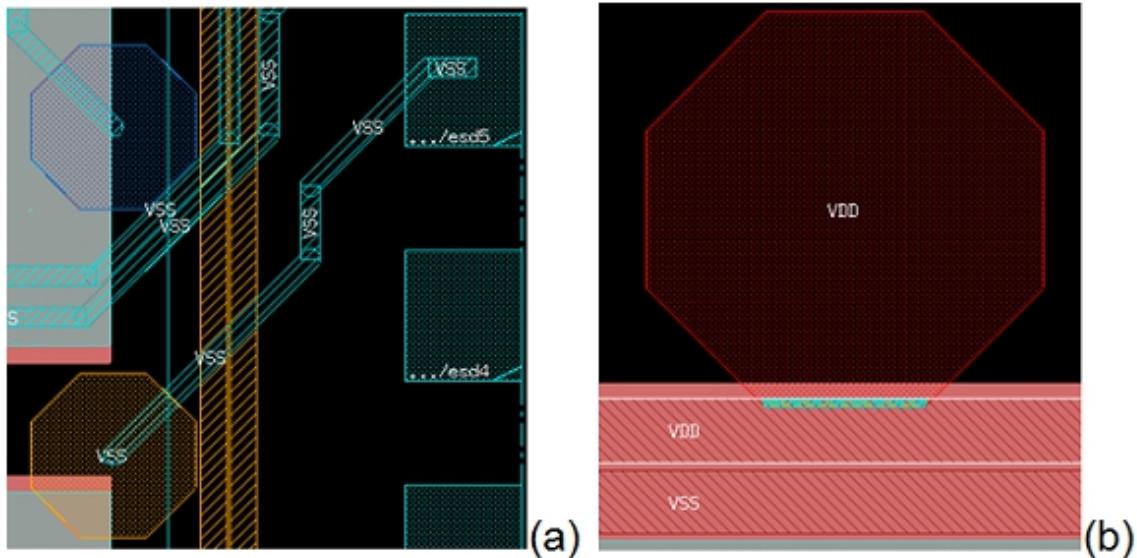
The image below shows the difference between `fcroute -type signal -designStyle pio`, `fcroute -type power`, and `addStripe`.



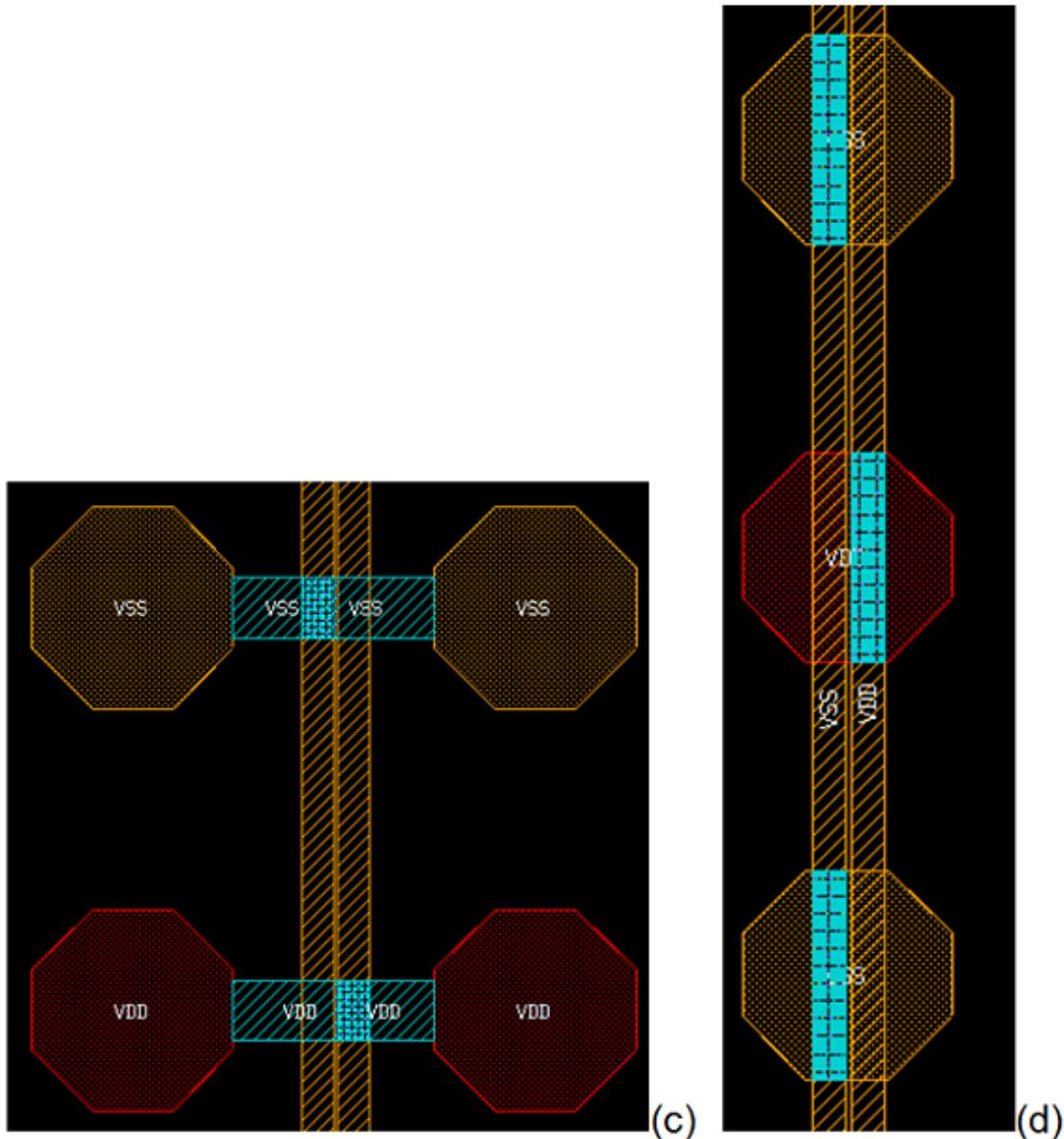
- Connect ground bump to I/O pad with `setFlipChipMode -connectPowerCellToBump true` and

`fcroute -type signal -designStyle pio`, as shown in (a).

- Connect power bump to block ring with `fcroute -type power`, as shown in (b).



- Connect PG bumps to stripe with `fcroute -type power`, as shown in (c).
- Connect PG bumps to stripe with `addstripe`, as shown in (d).



## ECO Routing

The `fcroute` command detects ECO changes and performs ECO routing automatically. As a result, you do not need to modify routing manually to complete ECO changes. To enable ECO mode flip chip routing, use the `-eco` parameter of the `fcroute` command. When you specify the `-eco` parameter, `fcroute` automatically performs typical ECO routing steps, such as deleting existing routing results or re-routing affected nets, whenever there is an ECO change.

Typical ECO routing working modes are:

- Working Mode I:
  - Delete existing routing results
  - Re-route affected nets
- Working Mode II
  - Re-route affected nets
- Working Mode III
  - Delete existing routing results

Let's see how typical ECO changes impact routing:

- IO-related ECO changes

| <b>ECO Change</b>                     | <b>Impact on Routing</b>                             |
|---------------------------------------|------------------------------------------------------|
| Change in IO pad location             | Invokes <code>fcroute -eco</code> (Working Mode I)   |
| Change in IO pad orientation          | Invokes <code>fcroute -eco</code> (Working Mode I)   |
| Change in IO pad placement status     | No ECO routing needed                                |
| Add a new IO ring                     | No ECO routing needed                                |
| Delete an IO ring                     | No ECO routing needed                                |
| Change IO ring to die boundary margin | No ECO routing needed                                |
| Add a new IO pad                      | Invokes <code>fcroute -eco</code> (Working Mode II)  |
| Delete an IO pad                      | Invokes <code>fcroute -eco</code> (Working Mode III) |

- Bump-related ECO changes

| <b>ECO Change</b>          | <b>Impact on Routing</b>                           |
|----------------------------|----------------------------------------------------|
| Change in bump location    | Invokes <code>fcroute -eco</code> (Working Mode I) |
| Change in bump orientation | No ECO routing needed                              |

|                                                    |                                                      |
|----------------------------------------------------|------------------------------------------------------|
| Change in bump placement status                    | No ECO routing needed                                |
| Change in bump assignment status                   | No ECO routing needed                                |
| Assign bump to another signal (including unassign) | Invokes <code>fcroute -eco</code> (Working Mode I)   |
| Add a new bump                                     | Invokes <code>fcroute -eco</code> (Working Mode II)  |
| Delete a bump                                      | Invokes <code>fcroute -eco</code> (Working Mode III) |
| Change the characters of an existing bump array    | No ECO routing needed                                |
| Add a new bump array                               | No ECO routing needed                                |
| Delete a bump array                                | No ECO routing needed                                |

- Routing ECO change

| ECO Change                                                                      | Impact on Routing                                                                                                                                                |
|---------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Routing is partially deleted                                                    | Invokes <code>fcroute -eco</code> (Working Mode II)<br><br><b>Note:</b> The rerouting should preserve undeleted routings and do reroute on deleted routing only. |
| Extra routing wires added but they cannot form a complete path from pin to bump | No ECO routing needed                                                                                                                                            |
| Extra routing wires added and they can form a complete path from pin to bump    | Invokes <code>fcroute -eco</code> (Working Mode III) and keeps new wires                                                                                         |

- Netlist ECO changes
  - Bumps are not in netlist.
  - For netlist ECO changes related with IOs, the tool performs routing as for IO-related ECO changes.

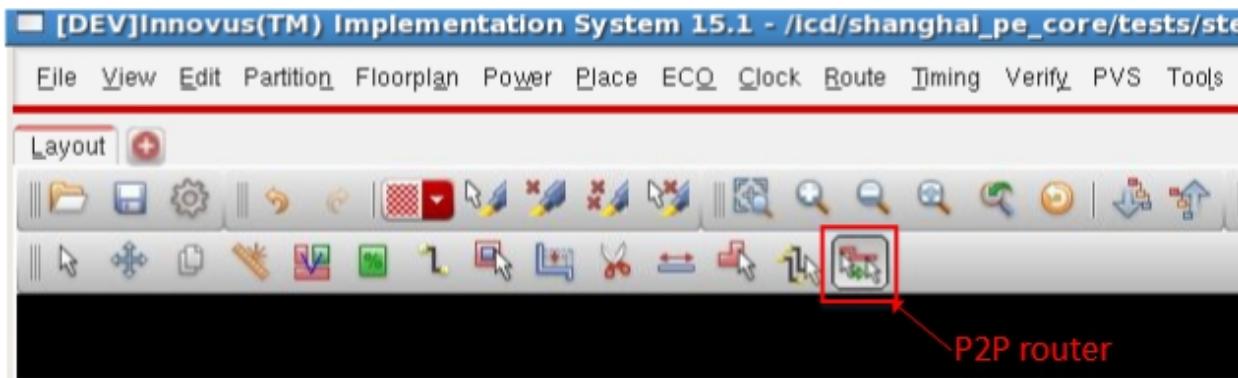
**Note:** The optimization of IO pad location and/or bump assignment after IO ECO is not supported. You can use the exclude region constraint to accomplish the task. For instance, in the example below, use `srouteExcludeRegion` in the `etr.cfg` file to exclude unaffected areas so that the optimization is done only on affected components:

placePIO -assignBump -extraConfig etr.cfg

assignBump issues an ERROR message and ignores it.

## P2P Router

Innovus supports a semi-automatic point-to-point (P2P) router. The P2P router can be accessed as follows:

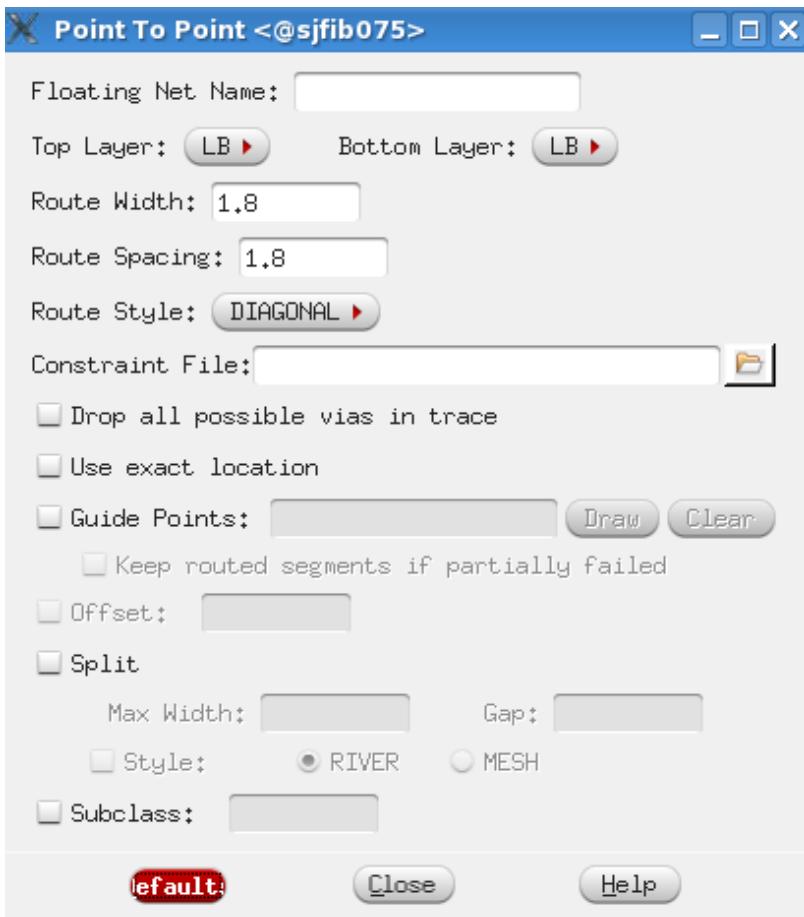


The P2P router can be used for customized and special routing pattern generation. To use the P2P router:

1. Select the P2P router button from the toolbar on the main window.
2. Press F3 to set up the P2P router.
3. Click the object that will be the source.
4. Click another object to set it as the target.

## Setting Up the P2P Router

When you press F3 for P2P setting, the following GUI form opens:

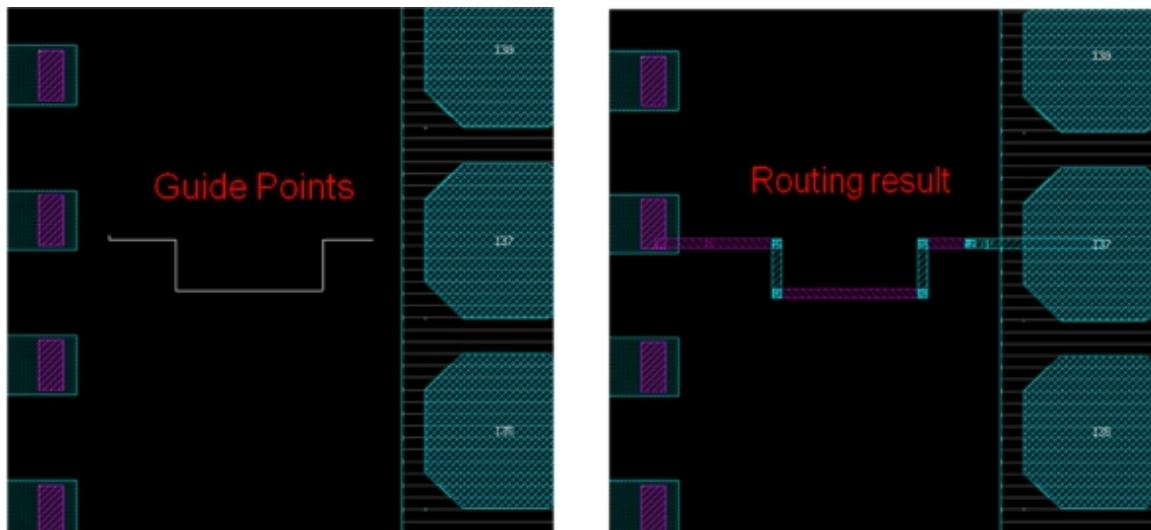


- If net name is not set, the P2P router automatically picks up the net name based on the objects selected in the GUI.
- If you do not select the *Use exact location* check box in the Point To Point form, the P2P router performs an auto search to find an access point. Note that:
  - *Use exact location* specifies that the P2P router should use the exact location that you have clicked.
  - You can additionally select the *Offset* check box and specify offset values to define the maximum search offset from the original click location.

The picture below depicts the results of auto selection versus using *Use exact location*:



- You can specify *Guide Points* in the Point to Point form to define a customized routing pattern. You can use this feature in following ways:
  - Click the *Draw* button in the form and then click the required guide points in the main window. Press the `Esc` key to return the location of selected guide points to the *Guide Points* text box. Click two objects for the source and target.  
Or
  - Press `Shift` to turn on *Guide Points*. Then, click the required guide points in the main window. Click two objects for the source and target.  
Or
  - Directly enter the location of the guide points in the *Guide Points* text box in the Point to Point form. Click two objects for the source and target.



## Handling Flip Chip Designs with Complex Floorplans

If you have a flip chip design with a complex floorplan, you might get the following error:

\*\*ERROR: Exceeding maximum number of 32 rows per cluster in cluster 0. Quit

The reason for this error is that if a design has a floorplan with IO pads of many different heights, the flip chip router may not be able to generate that many IO rows of different heights, internally. So, the flip chip router cannot proceed.

To solve this issue, you should do the routing in parts. You can try to unplace some IO pads (with CLASS PAD AREAIO or CLASS BLOCK) or filler cells (with CLASS PAD SPACER or CLASS PAD AREAIO) that may not need to be routed. With a simplified floorplan, do flip chip routing. After routing, you can load the IO pads and fillers back using the saved floorplan. Then, unplace another part and do routing.

## Flip Chip Router Report

You can use the `srouteFCReport file_name` option in the extra configuration file to report width, length, and resistance of the special nets routed by `fcroute`. The report file generated by this option has the following format:

```
#####
#####          fcroute report          #####
#####
NET net_name bump_name:pad_name:pin_name:port_num
```

```
STATUS open/resistance violation/routed  
  
Layer#: to be split/tapering/widthOpt  
  
    Path: Width xx Length xx Resistance xx  
  
    Path: Width xx Length xx Resistance xx  
  
    ...  
  
Layer#: to be split/tapering/widthOpt  
  
    Path: Width xx Length xx Resistance xx  
  
    Path: Width xx Length xx Resistance xx  
  
    ...  
  
Total length:xx  
  
Total resistance:xx (Constraint:xx)  
  
END net_name
```

## ***Format Definitions***

- *net\_name bump\_name:pad\_name:pin\_name:port\_num*

Specifies the connection between bump and pad based on net. Incomplete connection specifications are also supported:

- If the bump does not have the port number property, it outputs *bump\_name:pad\_name*
- If the port number property does not include *port\_num*, it outputs *bump\_name:pad\_name:pin\_name*

- STATUS open/resistance violation/routed

Reports the status of the net as one of the following:

| Status | Meaning                                                                                               |
|--------|-------------------------------------------------------------------------------------------------------|
| open   | If the net is not routed, its status is reported as open. No more information is output for this net. |

|                      |                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| resistance violation | If <code>fcroute</code> detects a resistance violation when it checks the resistance of the net against the resistance constraint set by MAXRES in the constraint file, the status of the net is reported as <code>resistance violation</code> . In this case, the <code>Total resistance</code> section reports the current resistance versus the expected resistance in <code>Constraint:xx</code> . |
| routed               | If the net is successfully routed without any resistance violation, the status is reported as <code>routed</code> .                                                                                                                                                                                                                                                                                    |

- Width/Length/Resistance/Total length/Total resistance

Specifies the width, length, and total length of special nets in microns.

Specifies the resistance and total resistance of special nets in ohms.

- The formula to calculate the resistance is as follows:

$$R = \sum_{i=1}^n \rho L_i / W_i$$

where  $L_i$  is the length of center line of  $i$ th wire segment,  $W_i$  is the width of the  $i$ th wire segment and  $\rho$  is read from the DB resistance table according to the layer and width information.

- Width/Length/Resistance is reported first by layer number from top to down and then by width.

- Layer#: to be split/tapering/widthOpt

Specifies the feature to be applied to the net.

|                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Layer#</b>             | <p>Indicates that no special feature is applied on this net.</p> <p>For example, suppose net vss is assigned to bump_vss with incomplete port number property.</p> <pre>NET VSS <i>bump_vss:ground_pad:vss</i>       STATUS resistance violation       Layer TOP_RDL       Path: Width 25 Length 200 Resistance 2.5       Layer 2<sup>nd</sup>_RDL       Path: Width 25 Length 50 Resistance 0.8       Total length: 250       Total resistance: 3.3 (Constraint:3) END VSS</pre>                                                       |
| <b>Layer# to be split</b> | <p>Indicates that the splitting feature will be applied on this net. Note that splitting happens after the report is output.</p> <p>For example, suppose net vss is assigned to bump_vss with incomplete port number property.</p> <pre>NET VSS <i>bump_vss:ground_pad:vss</i>       STATUS routed       Layer TOP_RDL       Path: Width 25 Length 200 Resistance 2.5       Layer 2<sup>nd</sup>_RDL <b>to be split</b>       Path: Width 25 Length 50 Resistance 0.8       Total length: 250       Total resistance: 3.3 END VSS</pre> |

|                        |                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Layer# tapering</b> | <p>Indicates that the tapering feature is applied on this net.<br/>         For example, suppose net vss is assigned to bump_vss with incomplete port number property.</p> <pre>NET VSS <i>bump_vss:ground_pad:vss</i> STATUS routed Layer TOP_RDL <b>tapering</b> Path: Width 25 Length 200 Resistance 2.5 Path: Width 15 Length 50 Resistance 1 Total length: 250 Total resistance: 3.5 END VSS</pre> |
| <b>Layer# widthOpt</b> | <p>Indicates that width optimization feature is applied on this net.<br/>         For example, suppose net vss is assigned to bump_vss with incomplete port number property.</p> <pre>NET VSS <i>bump_vss:ground_pad:vss</i> STATUS routed Layer TOP_RDL <b>widthOpt</b> Path: Width 25 Length 250 Resistance 3 Total length: 250 Total resistance: 3 END VSS</pre>                                     |

## Advanced Flip Chip Features

## Two-Layer RDL Routing

As flip chip design become more and more complex, one layer may not be sufficient for completing RDL routing. Innovus supports two-layer RDL routing for complex designs. However, in most flip chip designs, complete two-layer routing may not be required. Instead, you may need to use two layers only in the IO area and one layer in the core area to optimize routing resources. In these cases, you can add routing blockages to control where two layers are used and where only one layer is used for RDL routing. `fcroute` strictly honors any routing blockages you add to control two-layer RDL routing.

`fcroute` provides two kinds of constraints for two-layer RDL routing:

1. Use `PIOLAYERCHANGE PAD` in the constraint file and `srouteLayerChangeExcludeRegion "1lx1 1ly1 urx1 ury1 1lx2 1ly2 urx2 ury2 ..."` setting in the extra configuration file.
  - `PIOLAYERCHANGE PAD` in the constraint file
    - Turns on layer change feature in the PIO mode. This constraint is not supported by the AIO mode.
    - Both Manhattan and 45 degree routing support this constraint.
    - By default, this constraint uses the region defined by `srouteLayerChangeExcludeRegion "1lx1 1ly1 urx1 ury1 1lx2 1ly2 urx2 ury2 ..."` in the extra configuration file to prevent layer change from `fcroute`.
    - This constraint is applicable only when both `-layerChangeBotLayer` and `-layerChangeTopLayer` options specify the same routing layer. The direction of layer change is down, which means `fcroute` must change the routing layer to the layer lower than the specified routing layer.
  - `srouteLayerChangeExcludeRegion "1lx1 1ly1 urx1 ury1 1lx2 1ly2 urx2 ury2 ..."` in the extra configuration file.
    - Specifies the region to be excluded from layer change initiated by `fcroute`.
    - `srouteLayerChangeExcludeRegion "0 0 0 0"` means layer change is allowed on the whole chip.
    - This option must be used with `PIOLAYERCHANGE PAD` in the constraint file.
    - Only the PIO mode supports this constraint.
    - Both Manhattan and 45 degree routing support this constraint.
    - In the argument string for this option, you can define one or more rectangular shapes even if they are disjointed. You can also define a rectilinear shape.
    - The format of the argument string is as follows:

- ```
"rect_ll_x rect_ll_y rect_ur_x rect_ur_y [more_rects]" or  
"rectilinear_x1 rectilinear_y1 rectilinear_x2 rectilinear_y2 ..."
```
- The default region is the core region of the chip. If the core region is the same as the whole chip area, `fcroute` ignores the `PIOLAYERCHANGE PAD` setting in the constraint file.
  - This option prevents layer change in the specified region. However, multiple layers can be allowed in the region. This option places a virtual routing blockage on the cut layer. To prevent wires on a certain layer, a routing blockage is still needed.
  - This option is a soft constraint for `fcroute`.

If the setting for routing layers is `-layerChangeBotLayer TOP_RDL -layerChangeTopLayer TOP_RDL`, `fcroute` implements the above settings as follows:

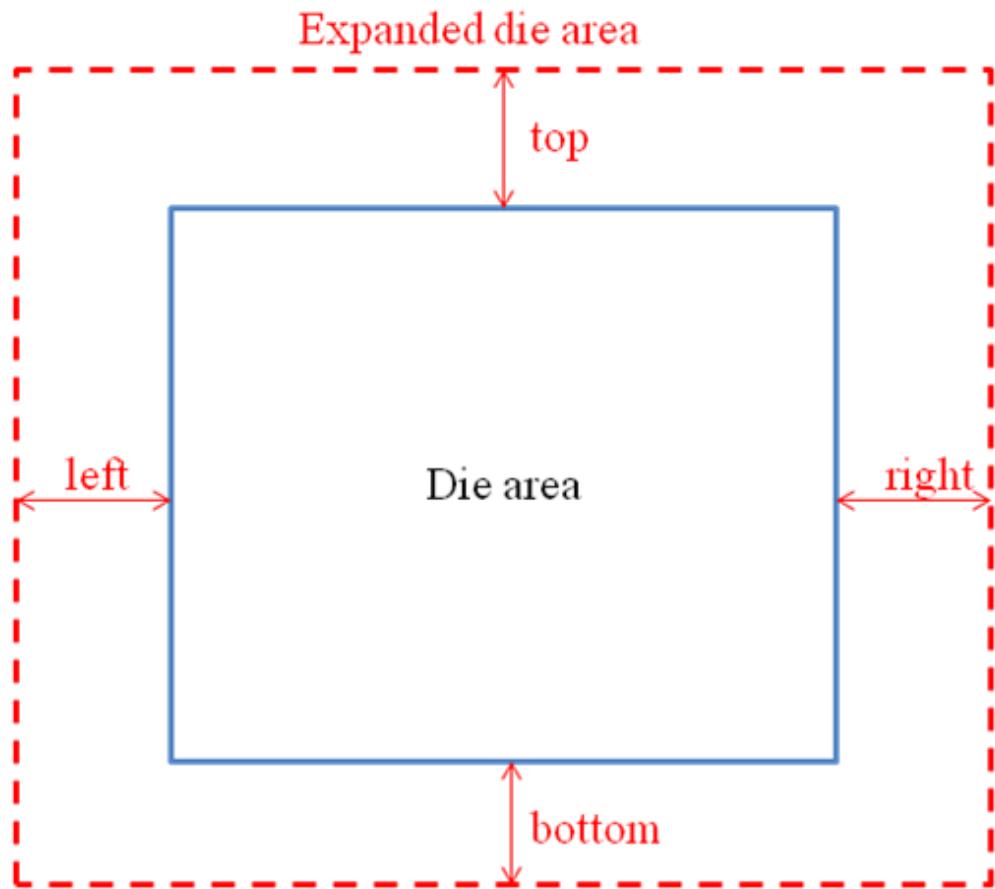
- Uses `TOP_RDL` as much as possible. It uses the second redistribution layer (RDL) only when a single layer cannot finish routing in case of cross-over.
  - Honors the settings defined by `srouteLayerChangeExcludeRegion` as a soft constraint.
2. Add routing blockage to control where to make layer change.  
`fcroute` strictly honors routing blockages. If you use a routing blockage with other routing constraints, `fcroute` honors the mixed usage as well.

## Routing Bumps in the eWLB Process

In embedded wafer level ball (eWLB) grid array process, some bumps are placed out of the chip and are required to connect to the IO pin in the die. However, out-of-die routing is not allowed by default. You can use the `srouteFcDieAreaOffset` option in the `fcroute` extra configuration file to specify the expanded area in which routing should be allowed:

```
srouteFcDieAreaOffset "left bottom right top"
```

Here, `left` specifies the distance (in microns) to which the expanded area should extend from the left edge of the chip. Similarly, `bottom`, `right`, and `top` specify the distance from the bottom, right, and top edges of the chip, respectively. `fcroute` can finish the routing in this expanded die area instead of the actual die area.

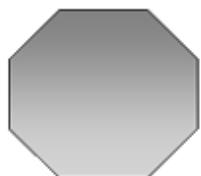


**Note:** Routing is not allowed outside the expanded die area. If `fcroute` cannot finish the routing to specific bumps in the expanded die area, it leaves these bumps open.

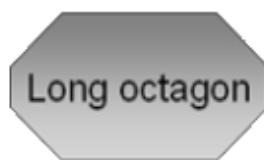
## Pillar Bump Support

Flip chip development is driven by device performance and package miniaturization trends. Higher device performance leads to more input/output connections per IC. At the same time, miniaturization requires smaller/thinner packaging, leading to smaller, closer-spaced connections. Fine-pitch flip chip pillar bumps reduce size while meeting the challenges of thinner ICs and maintaining robust IC and package reliability. As a result, pillar bumps are being used more and more in flip chip designs. With pillar bump, the shape of a bump changes from octagon to a long octagon as shown below:

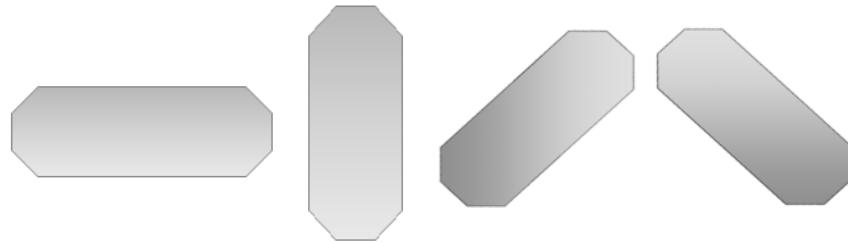
Standard Bump



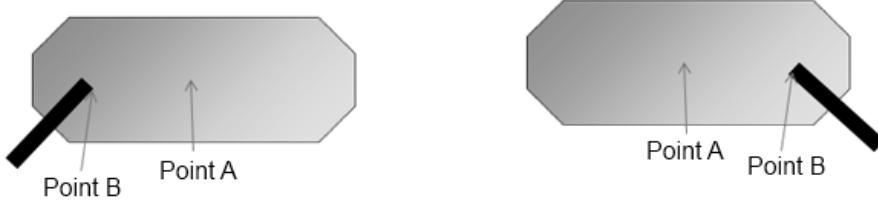
Pillar Bump



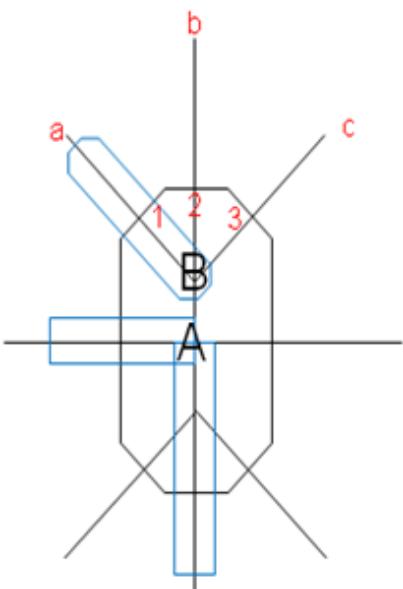
`fcroute` supports horizontal and vertical pillar bumps. In addition, it also supports 45/135-degree pillar bumps, which are already defined in LEF.



`fcroute` connects to bump center, point A for long-side or short-side entry, or point B for diagonal-side entry



Here, Point B is the intersection of lines a, b and c, which are the perpendicular bisectors of sides 1, 2 and 3 as shown below.



Sometimes, if `fcroute` cannot complete the routing when connecting to the center of bump, it may adjust the connection location with its intelligence and without causing any DRC violations.

You can use the `create_bump -orientation` option to specify the orientation of a pillar bump. When required, you can use this option to rotate a pillar bump by 90, 180, or 270 degrees to alleviate package and chip routing problems.

**Note:** Innovus does not rotate a bump by 45 degrees, but it can place and route 45-degree pillar bumps defined in LEF. Instead of rotating the pillar bump, you can change the pillar bump master for correct orientation.

## fcroute Bus Routing for DDR3

A long routing path in Double Data Rate 3 (DDR3) may cause crosstalk between signals. To prevent crosstalk, you must add P/G nets between signal nets and route these nets together with the same pattern like bus routing and specify different width and spacing per net.

In the bus routing pattern for DDR3:

- Route signal and P/G nets together with the same pattern in a long path.
- The shielding P/G net is floating, not connected to bump or pad pin in the 14.2 release.
- You can define the routing width and spacing for each net.

To support this bus routing pattern, you need to add the `NETGROUP` constraint in the constraint file. The `NETGROUP` constraint has the following format:

`NETGROUP`

```

BUSGUIDE net_group_name

SHIELDNET/BUMP name WIDTH value SPACING value

SHIELDNET/BUMP name WIDTH value SPACING value

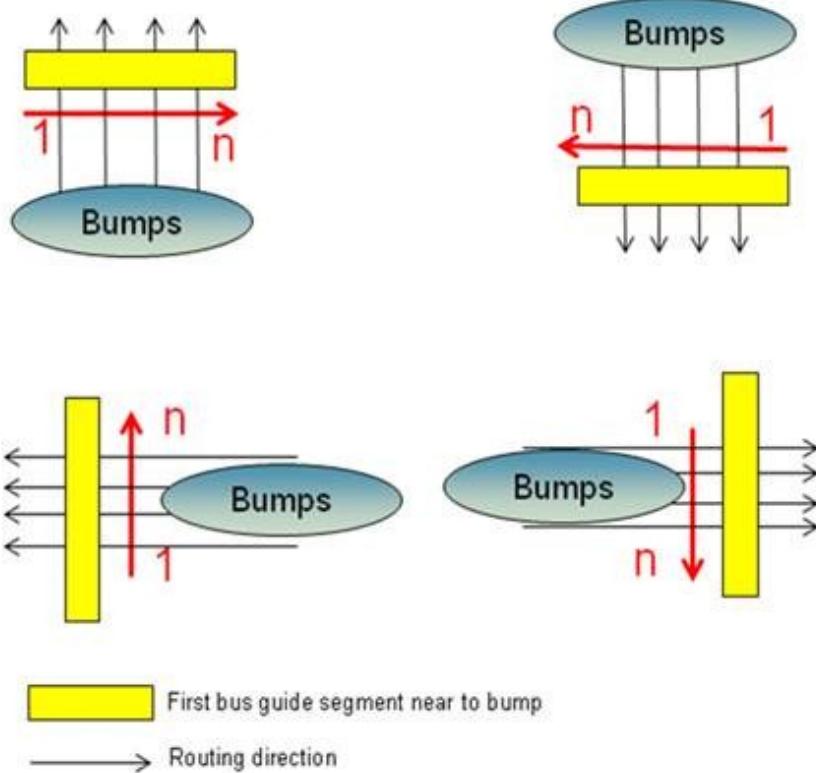
...
SHIELDNET/BUMP name WIDTH value SPACING value

```

END NETGROUP

Here:

- BUSGUIDE *net\_group\_name* guides the global router as a hard constraint. You must create the net group with `createNetGroup`. `fcroute` will honor the net order specified in NETGROUP in the constraint file.
- SHIELDNET/BUMP *name* specifies the shield net or bump name and specifies the order for routing. The tool finishes routing in the specified order.
  - In the 14.2 release, shield net will be floating after routing. The start/end point of shield net is controlled by the bus guide segments.
  - Define the net from left to right of routing accessing the first bus guide segment near to the bump as shown below. `fcroute` honors this order to complete the routing.



- **WIDTH** specifies the width for routing and overwrites the width by other options. It is optional.
- **SPACING** specifies the spacing for routing. It is optional. If it is not specified, min spacing in LEF is used.

After adding the `NETGROUP` constraint in the constraint file, use the `sroutePioBusRoute` configuration variable in the `fcroute` extra config file to control it. The use model is as follows:

1. Define the `NETGROUP` constraint in the constraint file.
2. Add `sroutePioBusRoute true` in the `fcroute` extra config file. With this setting, `fcroute` will route only the nets and bumps defined in `NETGROUP`.
3. Create a net group with `createNetGroup`.
4. Add the required nets to the net group with `addNetToNetGroup`.
5. Create a bus guide for the net group. You must use orthogonal bus guide segments for the start and end points. Others segments can be either orthogonal or 45-degree.
6. Run `fcroute` with `-selected_bump`, which specifies the bumps to be routed for `NETGROUP`.

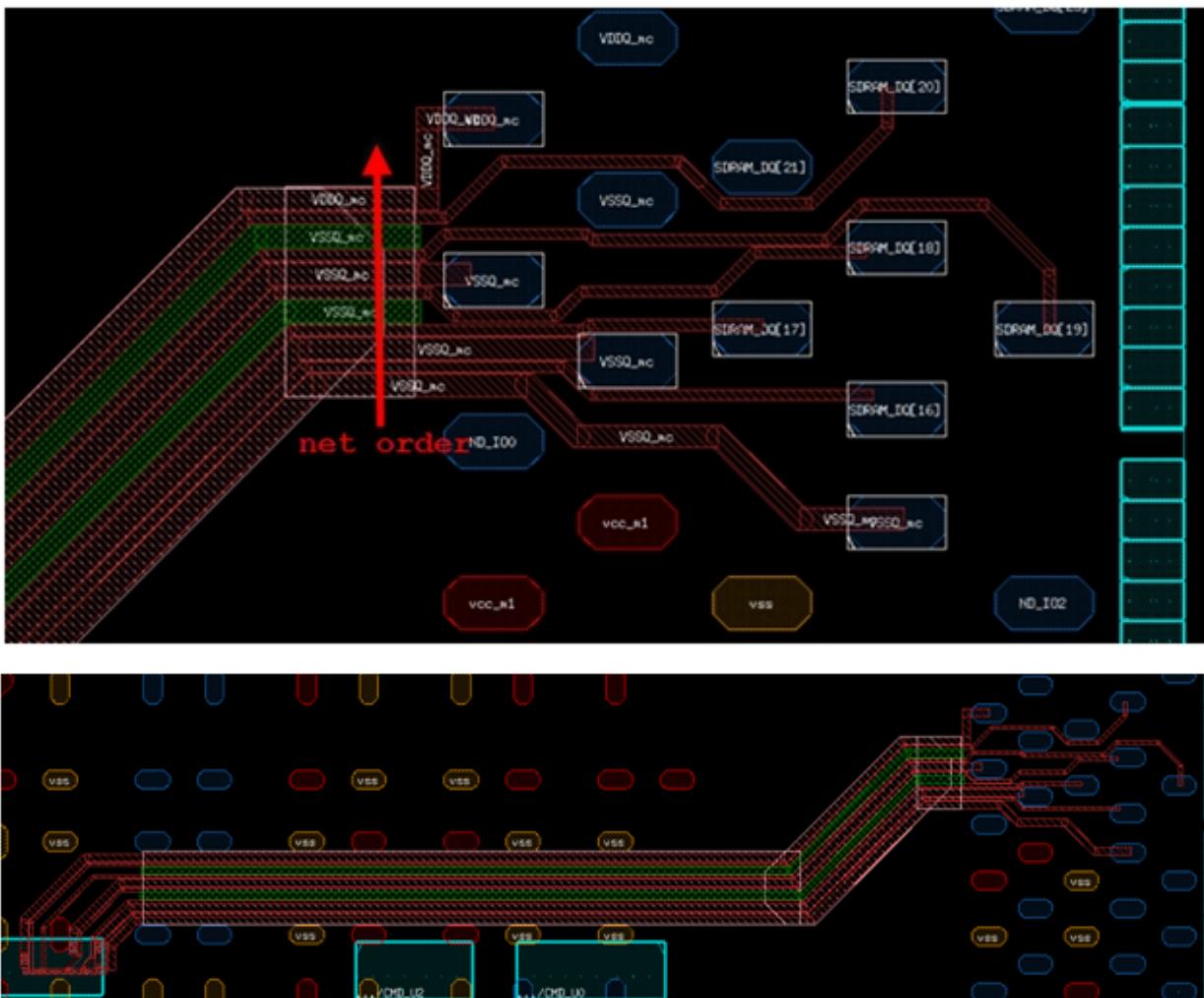
Here is an example of how `NETGROUP` constraint can be used:

NETGROUP

```
BUSGUIDE      ddr_1

    BUMP          Bump_598           WIDTH 28
    BUMP          Bump_601           WIDTH 13.2
    BUMP          Bump_602           WIDTH 28
    BUMP          Bump_606           WIDTH 13.2
    SHIELDNET    VSSQ_mc          WIDTH 28
    BUMP          Bump_607           WIDTH 13.2
    BUMP          Bump_612           WIDTH 28
    BUMP          Bump_605           WIDTH 13.2
    SHIELDNET    VSSQ_mc          WIDTH 28
    BUMP          Bump_610           WIDTH 13.2
    BUMP          Bump_613           WIDTH 28

END NETGROUP
```



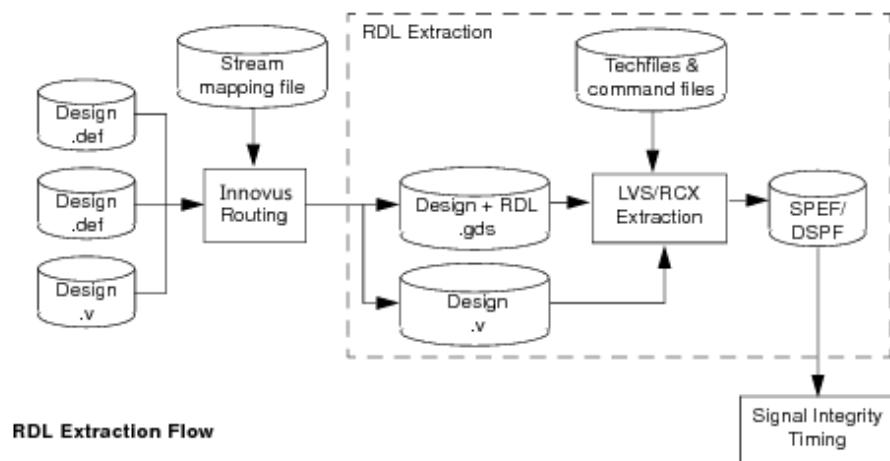
## RDL Extraction

In the RDL extraction flow for designs using peripheral I/O methodology, Innovus outputs the design with the RDL routing into a GDS file that is fed into RCX for parasitic extraction at the cell-level. RCX generates a cell-level SPEF/DSPF file that is used for timing and signal integrity analysis.

There are two steps involved in parasitic extraction with RCX.

- LVS is run to perform connectivity extraction.
- RCX is run to perform parasitic extraction.

The following diagram illustrates this flow.



### Inputs to Extraction

- Verilog netlist for annotation, generated by SoCE
- GDS of design with RDL, generated by SoCE
- RCX technology data

### Outputs from Extraction

- Cell-level SPEF/DSPF for SI/Timing analysis
- Includes coupling RDL nets to signal nets

## SI and Timing Analysis

The following procedure describes the signal integrity and timing analysis flow for an RDL design using the coupled SPEF file generated by the RCX extraction tool.

1. Restore the design.

```
restoreDesign routedSession.dat designname
```

This command restores the routed view of the design including the regular routing and RDL routing.

2. Import the coupled SPEF file from RCX.

```
spefIn rcx_coupled.spef
```

Make sure all the parasitics of the SPEF are back annotated in Innovus. If all the nets are back annotated, Innovus displays the following message:

0 nets are missing in SPEF file.

**3. Perform timing analysis in Innovus.**

- Run timing analysis using the `timeDesign` command.

```
timeDesign -postRoute -reportOnly
```

This command reports worst and total negative slack as well as register-to-register, input-to-register, and input-to-output port slacks.

**4. Analyze signal integrity by performing SI analysis in Innovus. The SI engine analyzes the design for glitch and SI violations. It generates incremental sdf for the delay induced due to SI. The incremental sdf is used to analyze timing with SI effects.**

```
timeDesign -postRoute -si
```

This command analyzes the design for SI and creates the analysis report. Later, the command uses an incremental sdf file for timing analysis and reports the worst negative slack path with SI-induced delay.

The following listing is a sample script for signal integrity and timing analysis in Innovus.

```
timeDesign -postRoute -reportOnly -si
```



# Hierarchical Flow Capabilities

---

- Partitioning the Design
- Timing Budgeting
- Using ART in Hierarchical Designs
- Top-level Timing Closure Methodologies
- Extracting Timing Models

# Partitioning the Design

- Overview
- Flow Methodologies
- Specifying Partitions and Blackboxes
- Working with Nested Partitions
- Assigning Pins
- Inserting Feedthroughs
- Generating the Wire Crossing Report
- Estimating the Routing Channel Width
- Running the Partition Program
- Restoring the Top-Level Floorplan with Partition Data
- Concatenating Netlist Files of a Partitioned Design
- Saving Partitions
- Loading Partitions
- Working with OpenAccess Database
- Parallel Job Processing
- Focused Methodologies

## Overview

Most of the system-on-a-chip devices are designed in a traditional flat flow that avoids the effort to set up a design hierarchy. However, in multi-million gate designs, this could result in memory limitations and long run time. Design teams can develop and adopt a hierarchical flow to shorten the turnaround time on large designs. Designs can be divided into manageable partitions; each partition can be independently assigned to different design groups to be developed in parallel.

## Flow Methodologies

Hierarchical design can be divided into three general stages: chip planning, implementation, and chip assembly.

- Chip Planning  
Breaks down a design into block-level designs to be implemented separately.
- Implementation  
This stage consists of two sub-stages: block implementation for a block-level design, and top-

level implementation for a design based on block-level design abstracts and timing models.

- **Chip Assembly**  
Connects all block-level designs into the final chip.

This chapter covers the following methodologies in the partitioning area:

- [Top-down Methodology](#)
- [Bottom-up Methodology](#)

## Top-down Methodology

The top-down methodology usually consists of top-down planning, implementation, and chip assembly stages. Use this methodology to create a top-level or hierarchical floorplan from a flat floorplan based on fenced modules. In this approach, the die size, shape, and I/O pads locations will drive block and partition placement. Block-level design size and pins will be generated based on the top-level floorplan.

## Chip Planning

The following steps describe the most common flow for chip planning, which includes specifying partitions and blackboxes:

1. Import the entire design to be partitioned. Import the design into the Innovus Implementation System (Innovus) environment. You can also include blackboxes.
2. (Optional) Define the blackboxes. If your design has blackboxes that are not specified in step 1, you can define them after reading in the netlist. You can also adjust the size of the blackboxes.  
For more information, see [Saving Blackboxes](#).
3. Lay out the floorplan. Manually pre-place all modules that will become partitions or blackboxes. You can also generate an initial floorplan by running plandesign to place Macros, then place standard cells and/or bring all modules inside the core.
4. Run power planning.
5. Specify the modules and blackboxes that will become partitions. You can further adjust blackbox size, if necessary.

For more information, see [Specifying Partitions and Blackboxes](#).

6. Run placement.
7. (Optional) Insert feedthrough buffers. Insert feedthrough buffers into partitions to avoid routing nets over partition areas. This step is necessary for channelless or mixed designs.  
For more information, see [Inserting Feedthroughs](#).

Run Trial Route before this step if you want to run route-based feedthrough insertion. You must also run Trial Route if you want to display and generate a list of all nets that cross over the top of each partition (using the Partition - Show Wire Crossing menu command or the `showPtnWireX` command).

8. Run Trial Route. Depending on what stage of the design is in, such as prototyping, intermediate, tapeout, use the appropriate option of the `trialRoute` command.

For example, the `-floorplanMode` parameter should be used for prototyping and the `-highEffort` option parameter should be used for tapeout mode. Use the `-handlePartition` or the `-handlePartitionComplex` parameter for channel-based designs. Use the `-handlePartitionComplex` parameter for channelless designs only after the feedthrough insertion step.

For channel based designs with thick channels, instead of running `trialRoute` with the `-handlePartitionComplex` parameter, use `trialRoute -fastRouteForPinAssign`. This route option generates routing topology similar to `trialRoute -handlePartitionComplex` but with lesser run time because it routes only the inter partitions and top-level nets.

If your design has blackboxes, you can run the `trialRoute` command with the `-routeBasedBBPin` parameter. With this parameter, the `trialRoute` command determines near-optimal location for blackbox pins with respect to top channel congestion and places blackbox pins at these locations. The `trialRoute` command then creates routes to the blackbox pins without crossing over blackboxes.

The results give the first-order location of aligning the partition pins.

9. Assign partition pins and blackbox pins using the `assignPtnPin` command.
10. Regenerate the routes that follow assign pins using the `trialRoute -honorPin` command.
11. Validate pin assignment result.
12. If needed, refine the pin assignment results or perform incremental pin assignment. If pin placement results need to be improved, you can further refine pin placement manually or automatically. After re-adjusting pins, verify pin placement again.
13. Budget the timing for blocks using the `deriveTimingBudget` command.

14. Partition the design using the [partition](#) command. If your design has multiple instantiated partitions, run the [alignPtnClone](#) command before the pin assignment step to make sure that all partition clones are well aligned with the master partition on a power mesh so you will not have any problems when flattening the partitions.  
For more information, see [Specifying Multiple Instantiated Partitions and Blackboxes](#).
15. Save the partition. This creates a directory for each block, and saves its netlist, floorplan, and budgeted constraints to this directory. For top-level designs, this also creates a directory containing the top-level netlist, floorplan, simple timing model, and physical abstract for each partition block or blackbox. Subsequent work should be done in these block-level and top-level directories for implementing the block-level and top-level designs, respectively.

**Tip:** You should do all design work in each saved partition directory, including the top-level directory.

## Implementation

After the chip planning is complete, the next stage is to implement the individual blocks. The detail of each block is implemented using the constraints for timing, size, and pin assignment determined during the planning stage. Block implementation should be done at a block directory that was generated by the [savePartition](#) step. At the completion of this step, block abstracts, timing models, a DEF file, and a GDSII file should be generated to be used in top-level implementation and chip-assembly.

The next step is to implement the top-level designs with block model data, such as LEF, timing model, power model, and noise model.

## Chip Assembly

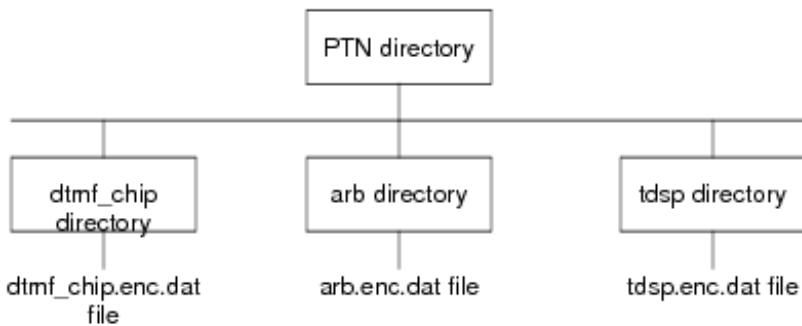
Chip assembly is the last stage in the top-down process and consists of bringing together the detailed information for the top-level and all of the blocks for full chip extraction, power, timing, and crosstalk analysis. Chip assembly is done using the [assembleDesign](#) command.

**Note:** Before using the [assembleDesign](#) command, for each design, save the top-level and the block-level designs using the [saveDesign -def](#) command.

As an example, consider a design called dtmf that has two partitions: arb and tdsp. After running the [partition](#) command, the partition directories are saved under the PTN directory. You would, therefore, implement the following:

- top-level design dtmf\_chip
- arb block
- tdsp block

The design files are arb.enc.dat and tdsp.enc.dat for the arb and tdsp blocks respectively. The following figure shows the directory structure:



You can now perform chip assembly using the `assembleDesign` command. This command does the following:

- Concatenates the Verilog netlist files from the partitions back to the top level  
**Note:** The partition netlists and top level netlist are changed from the time the save partition step was performed.
- Merges the design data with the original top design level. By default, data from DEF files is used. However, you can use the -fe parameter to specify that Innovus data should be used. You can also use data in the OpenAccess database format.
- Brings back the row information if the `-row` parameter is specified.
- Preserves scan chain information at partition block level design, thus minimizing the floorplan data loss during partition and assemble design cycle. The start and stop scan chain points at partition block I/O pins are adjusted back to instances that connect to scan chain points. Top-level scan chains are not connected to block-level scan chains.

Run this command from the directory that contains the full chip-level floorplan for the top-down hierarchical flow.

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

```
assembleDesign -topDir PTN/dtmf_chip/dtmf_chip.enc.dat -blockDir PTN/arb/arb.enc.dat -  
blockDir PTN/tdsp/tdsp.enc.dat -topFP fullChip.fp
```

This assembles the entire design. You can also use the `assembleDesign` command to bring back specified block data from OpenAccess database.

Here is an example:

```
assembleDesign -topDesign testOALib DTMF layout -block testOALib ptn1 layout -block testOALib ptn2 layout
```

In this example, the OpenAccess database top-level library is testOALib, the top-level cell name is DTMF, and the top-level view is layout. Two blocks, ptn1 and ptn2, have been specified.

**Note:** The `assembleDesign` command supports rectilinear partitions. It also supports nested blackboxes for the place-and-route data (-fe parameter) and the OpenAccess database. However, because blackbox information cannot be specified in a block-level DEF file, nested blackboxes are not supported for the DEF flow.

## Bottom-up Methodology

The bottom-up methodology consists of implementation and assembly stages. In the bottom-up methodology, the size, shape, and pin position of block-level designs will drive the top-level floorplanning.

## Implementation

Each block in the design must be fully implemented. This includes place and route as well as clock, power, and I/O. This section covers the following topics:

- [Block Implementation](#)
- [Top-level Implementation](#)

## Block Implementation

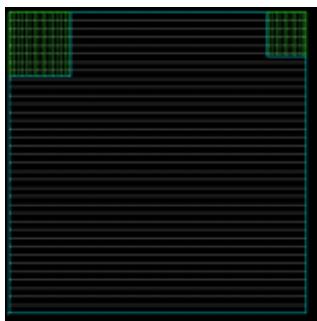
The size of a block-level design can be derived or adjusted using the *Floorplan - Specify Floorplan* menu command or the `floorplan` command. The Innovus software can support a rectilinear block level design. You can use the same procedure to create a rectilinear partition to create a rectilinear block-level design using the following steps:

1. Click on the *Cut Rectilinear* widget from the *Tools* area.
2. Move the mouse to an edge or corner of the design.
3. Left-click and drag over the area.
4. Left-click again to complete the cut.

At a block level design the rectilinear information will be stored in a floorplan file as

a CellPtnCutList syntax, for example:

```
CellPtnCutList: execute_i 2
 0.0000 142.5100 37.1200 181.4400
 156.3800 154.9350 180.1800 181.4400
```



You can run the `assignIoPins` command to assign I/O pins based on placement information. You can specify initial I/O pin placement in an I/O constraint file.

For more information, see the Generating the I/O assignment File section in the "Data Preparation" chapter of the Innovus User Guide.

You can read in the I/O constraint file into the Innovus environment during the design import step, or use the `loadIoFile` command after reading in the netlist. If an I/O constraint file does not exist, an initial I/O pin placement can be derived from cell placement. After placing macros and standard cells, the placer can internally call the `assignIoPins` command to place I/O pins based on current cell placement. By default, pins are placed under power areas on different layers.

**Note:** Use the `setPlaceMode -placeIoPins` command to disable I/O pin assignment during placement.

After I/O pins have been assigned, you can further refine the current I/O pin assignment by doing either of the following:

- Adjust pins (using the *Pin Editor* or the `editPin` command). You can also use direct pin manipulation to manually move selected pins to different locations.
- Run incremental pin assignment by running the `assignIoPins` command. This command honors fixed pins and re-assigns only the ones that have a placed or unplaced status.

**Note:** The `loadIoFile` command automatically sets the I/O pin placement status to fixed. For the pins that need to be re-assigned, you must change their pin placement status.

You can use the `legalizePin` command to resolve pin overlaps or pins off-grid.

## Top-level Implementation

After block implementation, an abstract should be developed for each block-level design that will be used in the top-level implementation. For the bottom-up approach, create a top-level floorplan where block-level abstracts would be referenced in the top-level design.

## Chip Assembly

For the bottom-up approach, see [Chip Assembly](#), to bring together all the top-level and block-level netlists and routing information.

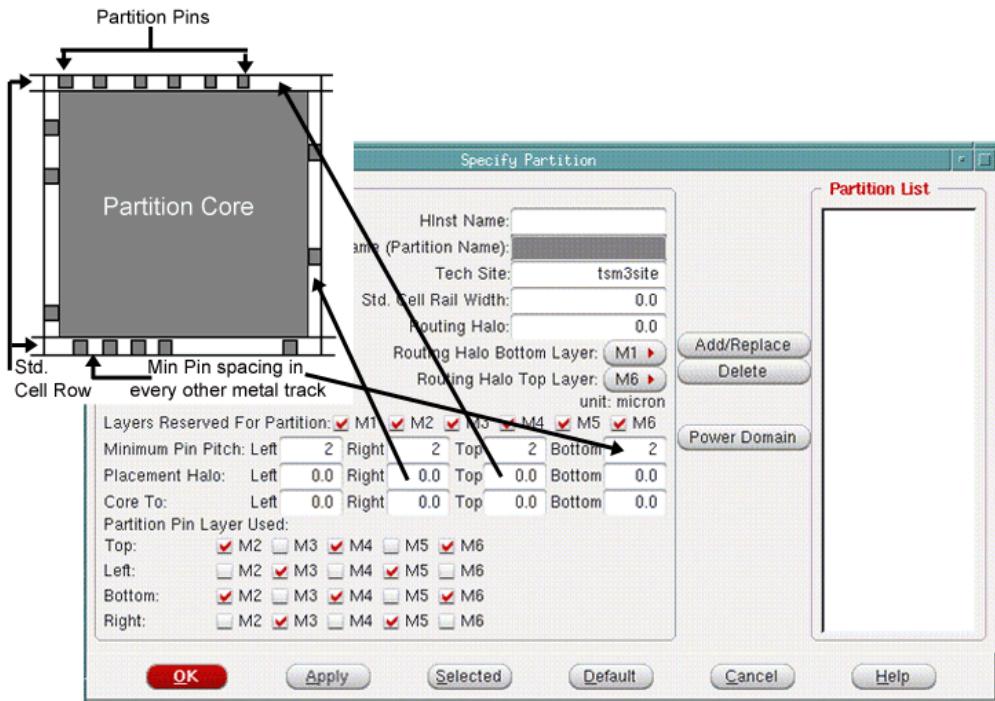
**Note:** For the bottom-up approach, do not use the `-topFP` parameter of the `assembleDesign` command.

## Specifying Partitions and Blackboxes

- [Defining Partitions](#)
- [Defining Partitions as Power Domains](#)
- [Defining Blackboxes](#)
- [Saving Blackboxes](#)
- [Handling of Blackboxes with Non-R0 Orientation](#)
- [Specifying Multiple Instantiated Partitions and Blackboxes](#)
- [Changing Partition Clone Orientation](#)
- [Specifying Rectilinear Partitions and Blackboxes](#)
- [Specifying Core-to-I/O Distance for Partition Cuts](#)
- [Working with Nested Partitions](#)
- [Assigning Pins](#)

## Defining Partitions

To designate partitions, use the `definePartition` command and the *Specify Partition* form. The following figure shows an example of how some of the fields in the *Specify Partition* form relate to the partition. For a description of all the fields, see [Specify Partition](#) in the *Innovus Menu Reference*.



To specify a module as a partition, complete the following steps:

1. Move the module inside the core area. You can manually move a module, or use the `setObjFPlanBox` command, to define a new module boundary with its coordinates in the core area.  
**Note:** A blackbox is a special partition where this restriction does not apply.  
**Note:** You cannot create donut shaped objects during the partition flow.
2. Specify the name of the partition.
3. Specify the instance name of a module that is to become a partition.  
**Note:** A partition cannot have another partition as its ancestor or descendant. For the case where more than one module is instantiated with the same cell type, see [Specifying Multiple Instantiated Partitions and Blackboxes](#).
4. Specify the space, in micrometers, between the module boundary and core design area of the

partition module.

5. (Optional) If the partition row height is different than specified in the *Core Spec* page of the Design Import form, specify the row height, in micrometers.
6. (Optional) To account for wide wires at the top-level design, specify the extra spacing, in micrometers, around the partition. At the top-level design, this information is saved as part of the partition section in a floorplan file. This information is also saved in a partition floorplan file when saving partitions. By default, this value is 0; the top-level router uses minimum wire spacing.
7. Specify the selected metal layers that are used for routing in the partition and generating partition pins. A normal six-metal layer selection process is M1,M2, M3, M4, and M5 selected, and M6 unselected. When saving the partition, the LEF generated for this partition will have routing blockages on their layers so that the top-level router is aware of which metal layers are being used in the partition.

The selected metal layers generate a file which is saved in the top-level design directory. This file notifies the top-level which metal layers are being used in the partition. In addition, the floorplan file generated by saving partition will include the routing blockage for the partition.

To customize routing interconnects over a partition, use the *Add Partition Feedthrough* widget.

8. (Optional) Specify the pin pitch dimension for the partition sides.
9. (Optional) Select or deselect the metal layers from the defaults. Deselecting all metal layers for a side of a partition prevents pins from being created for the entire side of that partition. The selection of partition pin metal layers works in conjunction with the Partition Pin Guide floorplan object. The partition pin guide object specifies exactly where the pins are to be created. When partition pin guide objects are not used, the partition pins are created where the top-level routing connects with the partition.
10. Add the partition information to the *Partition List* field and save the partition specification file.

## Defining Partitions as Power Domains

If a block-level design has different row structures than a top-level design, you will need to define a partition as a power domain. The power domain must be a hierarchical instance. The power domain will have the same size as the partition fence. To specify a partition as a power domain, complete the following steps:

1. Import the design.
2. Create the power domain.

3. Floorplan the design. In this step you would normally place the I/Os, place the power domain, and so on.
4. Assign a partition to a power domain by specifying the same power domain hierarchical instance as the partition.
5. Continue with the normal partition flow (see [Defining Partitions](#)).

## Defining Blackboxes

Normally a blackbox is a module with content that is not well defined. However, a well-defined module can also be defined as a blackbox. A blackbox is similar to a hard block, but like a fence, a blackbox can be resized, reshaped, and have pins assigned. After a blackbox has its pins assigned and is partitioned, it behaves like a hard block. The blackbox feature can be used only with a partitioned design.

After the netlist has been loaded, you can further specify which modules or cells will be regarded as blackboxes, or modify the existing blackbox sizes. A blackbox size can be specified in terms of an estimated area (an actual value or an area value in terms of gate count), or a fixed block width and height.

You can define a blackbox in the following ways:

- Use the `setImportMode -treatUndefinedCellsAsBbox false -keepEmptyModule true` command before importing a design. Once the design is imported, specify a module or hard macro as blackbox using the [specifyBlackBox](#) command or the [Specify Black Box](#) form.  
**Note:** Converting a hard macro into a blackbox will not update the blockage definitions when you change the blackbox size.
- Define LEF abstracts for blackboxes. You can specify a blackbox library in the LEF Files field of the *Design Import* form. If a blackbox LEF abstract is specified in the LEF Files field, the LEF abstract should have CLASS type as BLOCK BLACKBOX to indicate it is a blackbox.  
The following is an example of a blackbox LEF abstract:

```
MACRO amba_dsp
  CLASS BLOCK BLACKBOX ;
  ORIGIN 0 0 ;
  SIZE 4411.8600 BY 5697.3600 ;
END amba_dsp
```

After defining a blackbox with any of the above methods, you can further modify an existing blackbox size with the [specifyBlackBox](#) command.

**Note:** You can use the `getBlackBoxArea` command to retrieve the standard cell area, macro area, and cell utilization value for the specified blackbox.

**Warning:** If you convert a hard macro into a blackbox or define a blackbox with a LEF abstract that has obstructions, the obstructions size will not be updated with a new blackbox size. Due to this limitation, obstructions may be intruded outside of the new blackbox boundary.

## Blackbox Flow

**Note:** Even though there are more than one ways to define a black box, it is recommended that you define a black box by using the `specifyBlackBox` command.

The following flow specifies blackboxes with an original netlist that has modules with content that is not well-defined:

1. Import the design. By default, the Innovus software keeps empty modules (`setImportMode -treatUndefinedCellAsBbox false -keepEmptyModule true`)
2. Specify the blackboxes or load a floorplan file with blackbox information.
3. Floorplan the design.
4. (Optional) Save the design, which saves the blackbox information.
5. Run placement.
6. (Optional) Run Trial Route with or without the `-routeBasedBBPin` parameter. When you run Trial Route with this parameter, Trial Route determines near-optimal location for blackbox pins with respect to top channel congestion and places blackbox pins at these locations. Trial Route then creates routes to the blackbox pins without crossing over blackboxes.
7. Proceed with the normal hierarchical flow for the design.

There is no separate step required for assigning blackbox pins or committing the blackbox.

After the blackbox pins are placed at near-optimal location by running Trial Route with the `-routeBasedBBPin` parameter, use the `assignPtnPin` command to finally place blackbox pins to honor user-specified constraints.

When you partition the design, blackboxes as well as regular partitions are committed. Blackboxes get converted to hard macros at top-level design that display as a Block object in the [Attribute Editor](#).

The following flow is an ECO flow where the contents of the black box are now well defined.

1. Restore the design (or import the design and load a floorplan with the black box information)
2. Run the `loadBlackBoxNetlist` command to incrementally load the netlist for the blackbox.

You can run this command without exiting the current session of the Innovus software.

3. Run the `convertBlackBoxToFence` command to convert the blackbox to a fence.

**Note:** To convert the fence back to a blackbox, run the `convertFenceToBlackBox` command.

Continue with the following steps to finalize pin assignment for the black box:

4. Proceed with the normal hierarchical flow for the design.

## Saving Blackboxes

To save blackbox information, use the `saveDesign` command or the *File - Save Design* menu command.

## Reshaping Blackboxes

During `planDesign`, a blackbox can be reshaped (within specified aspect ratio range) to minimize overlaps. This reshape is based on the minimum and maximum values for the aspect ratio range while maintaining the current area. The master and clone blackboxes are reshaped such that the clone blackbox take the same size and shape as its master while meeting orientation constraints.

## Deleting Blackboxes

If a blackbox is an empty module in the netlist, or a cell without a physical macro definition, you must modify the netlist before you can delete it.

**Tip:** You should not delete a blackbox that was originally defined as a macro in the technology file; otherwise, you might have problems with loosely integrated applications because these application interfaces automatically generate only macro definitions for blackboxes. You should only use the delete capability to try out different floorplan.

## Handling of Blackboxes with Non-R0 Orientation

The partitioning- and blackbox-related commands in Innovus support only those blackboxes whose master instances have an R0 orientation. Clones with a non-R0 orientation clones are, however, supported. Partitioning-related commands such as `assignPtnPin`, `partition`, `assembleDesign`, `flattenPartition`, `convertBlackBoxToFence`, and `editPin` work only with those blackboxes whose master instances have an R0 orientation. Several commands in the Innovus software automatically convert the orientation of master blackboxes to R0. In addition, you can also run the `changeBBoxMasterToR0` command to convert the orientation of the master blackboxes to R0. This would be useful for example, you restore a design and want to convert the orientation of all the master blackboxes to R0.

**Note:** You can use the `changeBBoxMasterFromR0` command to change the orientation of the master blackboxes from R0 to a different orientation.

The following sections provide addition information about automatic conversion of orientation and about the `changeBBoxMasterToR0` command.

- Automatic Conversion of Orientation
- Performing R0 Transformation

## Automatic Conversion of Orientation

When the following commands change the orientation of a master instance blackbox to non-R0, the commands automatically convert the new orientation to R0:

- `specifyBlackBox`
- `multiPlanDesign`
- `placeInstance`
- `planDesign`

In addition:

- Opening the *Attribute Editor* for such a master blackbox automatically converts the orientation to R0.
- Using the *Flip* or the *Rotate* options from the context menu (the menu that appears when you click the middle mouse button on an object) automatically converts the orientation to R0.
- Using the *Flip* or the *Rotate* options on the *Floorplan* toolbox automatically converts the orientation to R0. For more information, see *Floorplan Toolbox* in the [Floorplan Menu](#) chapter

of *Innovus Menu Reference*.

The conversion includes the following:

- Cell blackbox geometries (PORT, OBS, and so on) are transformed.
- Master instances are converted to R0 orientation. The clone instances are oriented accordingly.  
**Note:** The placement location remains unchanged.
- Any pin guides, pin blockages, and pin constraints associated with transformed blackboxes are deleted.  
**Note :** There is no change in the design physically as a result of these transformations. Only the cell orientation and the instance representation are modified.

As an example, if the blackbox master instance is MX, then after the transformation:

- Cell geometries are transformed to MX
- The orientation of the master instance is changed to R0.

## Performing R0 Transformation

For designs that contain blackboxes whose master instances have a non-R0 orientation, you can use the `changeBBoxMasterToR0` command to convert the orientation of the master blackboxes to R0.

The syntax of the command is as follows:

```
changeBBoxMasterToR0 [-checkOnly] [{cellName | cellNameList}]
```

If `cellName`, or `cellNameList`, is not specified, the command converts the orientation of all the non-R0 master blackboxes to R0. If the `-checkOnly` parameter is specified, the command does not actually convert the orientation of any master blackbox; it only displays the number of master blackboxes whose orientation would have been changed had the command been run without the `-checkOnly` parameter.

When you are ready to run a loosely integrated application, complete the following steps:

1. Run the `saveDesign` command to make sure that you have updated the size and pin information.
2. Exit the Innovus software.
3. Rerun the Innovus software with the updated macro information.

To delete all the blackboxes in the design, use the `unspecifyBlackBox -all` command.

## Specifying Multiple Instantiated Partitions and Blackboxes

When a module with multiple instantiations (also known as repeated partitions) of the same cell type is assigned to become a partition, you can specify either one of the multiple instantiated hierarchical instances to be partitions. The name of a hierarchical instance used for partition specification becomes the master partition, and the other instantiations are clones of this master partition.

**Note:** All the master and clone hierarchical instances should be placed inside the core before you specify the partition. This restriction does not apply to blackboxes.

When working with repeated partitions, you should be aware of the following:

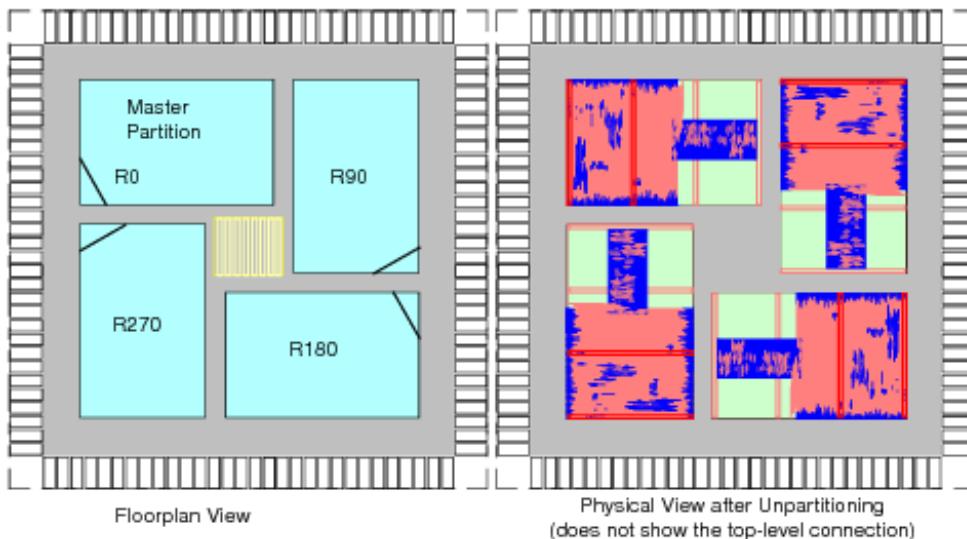
- You can only specify one instance as a master partition. The Innovus software will treat the other instances as partition clones.
- For the top-down hierarchical flow, where the top-level design is implemented first, the instance must have a R0 orientation; otherwise, you will run into problems with the pin assignment, feedthrough buffer insertion, and commit partition steps.
- For the bottom-up hierarchical flow, where the block is implemented first, the partition master can have a non-R0 orientation. Make sure all the non-uniquified instances are placed inside the core before you specify the partition.
- For non-uniquified blackboxes, the Innovus software automatically converts all hierarchical instances of a same module as repeated blackboxes. The hierarchical instance that is first instantiated in the netlist is treated as the master blackbox.
- Partition and blackbox clones can be rotated and flipped even if the vias used in the design are not square. The `assembleDesign` command will create the required symmetry of the via if its definition is missing in the LEF.
- Partition clones share the same pin assignment and pushed-down data as their partition master, you must run the `alignPtnClone` command before the commit partition step to make sure all the partition clones are well aligned with the master on power mesh so you do not run into problems when flattening the partitions.
- For master and clones partitions, the Innovus software automatically snaps the clone partitions such that clones will have the same row structure and pattern as their master. To disable this snapping capability, use the `-noEqualizePtnHInst` option of the `loadFPlan` command.

## Changing Partition Clone Orientation

After specifying the partition, you can change the partition clones' orientation by using the `setClonePtnOrient` command or through *Attribute Editor* during floorplanning. Use the `getClonePtnOrient` command to retrieve orientation information of a specific partition clone. For routing purposes, the Innovus software automatically stitches regular wires and rotates vias correctly for non-R0 orientations, such as MX, MY, R90, R180, and R270.

For example, there is a case where some of the clones follow the orientation of the master instance (R0), and some are placed with R180 orientation. After chip assembly, the Innovus software flips and places the clone instances' standard cells to match the R180 clone orientation, and repositions the routing according to the R180 orientation. Because R90 and R270 orientation clones have vertical rows, all the cell placement, routing, and IPO should be done at the top-level before flattening step. After flattening the design, you should only run full chip flat timing analysis.

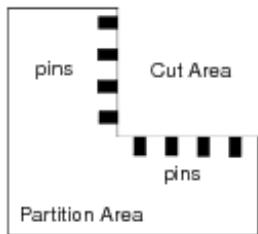
The following example shows a design that has R90, R180, and R270 orientation clones:



**Note:** The illustration above only shows the wire information inside the partition, and does not include the top-level connection.

## Specifying Rectilinear Partitions and Blackboxes

You can specify a rectilinear (non-rectangular) partition shape by adding a cut area. The partition's cut area will have no cell placement and no routing. Pins are assigned to the rectilinear partition edges, as shown in the following figure:



The rectilinear pin assignment recognizes the rectilinear edges when assigning pins, and supports any rectilinear shape. See [Assigning Pins on Rectilinear Edges](#) for more information.

To add a cut area to the partition or blackbox, complete the following steps:

1. Click on the *Cut Rectilinear* widget from the *Tools* area.
2. Move the mouse to an edge or corner of the partition or blackbox.
3. Left click and drag over the area.
4. Left click again to complete the cut.

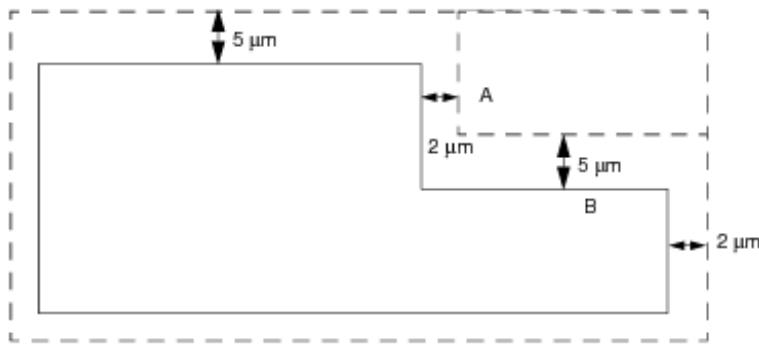
A macro definition file (LEF) will be created with blockage on the overlap layer covering the cut area. For the top-level partition, the cut area allows block or cell placements. The equivalent text command is `setObjFPlanBoxList` with the Module object type. For backward compatibility, you can also use the `createPtnCut` command. You should specify a module as a partition before using `createPtnCut`. For repeated partitions or blackboxes, when you create a cut on one instance – either master or clone – the cut is applied to the other instances as well.

**Note:** If a cut is made on a blackbox that has pins assigned to it, the affected blackbox pins are automatically moved to the new edge boundary created by the cut.

## Specifying Core-to-I/O Distance for Partition Cuts

Core-to-I/O distance is specified in the Specify Partition form. If the partition has a partition cut, core-to-I/O distance is honored where the cut is specified. The specified top, bottom, left, and right core-to-I/O distances are automatically assigned for the cutting edges that face the north, south, west, and east side, respectively.

For example, if you specify a core-to-I/O distance of 5  $\mu\text{m}$  for the top and bottom, and 2  $\mu\text{m}$  for left and right sides:



The core to I/O distance for the edge A (facing east) should be 2  $\mu\text{m}$ . The core to I/O distance for the edge B (faced to north) should be 5  $\mu\text{m}$ , same as the top side.

## Working with Nested Partitions

The Innovus software does not normally support a partition that is nested inside another partition. You can work around this limitation by using one of the following methods:

- Specifying Second-level Partitions
- Using the Multi-level Hierarchical Flow

## Specifying Second-level Partitions

For nested partitions, you can specify the second-level partition at the partition-level design. For example, consider a case where the module `mult_32` is a nested module inside the module `tdsp_core` and you want to define both `mult_32` and `tdsp_core` as partitions. For this, first define `tdsp_core` as a partition and then follow the normal partition flow to define `mult_32` as a partition.

Here are the steps:

1. Import the design.
2. Specify `tdsp_core` as partition.

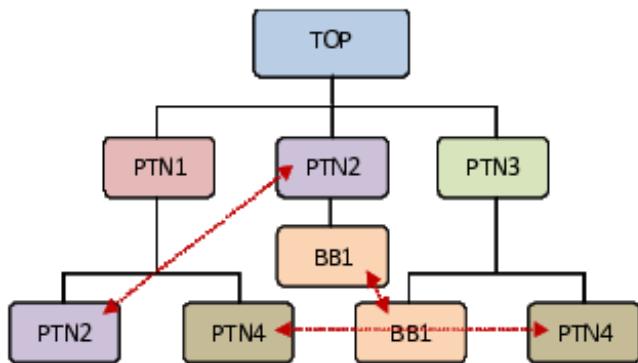
3. Perform placement and routing.
4. Commit the partition `tdsp_core`.
5. Save the partition.
6. Change to the `tdsp_core` partition directory.
7. Define `mult_32` as a partition.

## Using the Multi-level Hierarchical Flow

The multi-level hierarchical flow, enables partitions to be defined inside partitions. This helps to avoid the need to partition a big design one stage at a time. Instead of a single level partitioning of the design you can make nested partitions at different levels. This helps in better control of partition mapping and reduces the turnaround time. The multi-level hierarchical flow, supports master and clones at different levels of hierarchy.

A nested design can have:

- partition inside a partition
- clone inside a partition
- clone inside a clone



Even though a fence can be defined inside a fence in a single level of partition also, only one of these fence is allowed to be defined as a partition. In multi-level designs, all the fences can be defined as partitions. To see child fences, you can use *Show Children* from the context menu of the parent fence. All operations of object creation and manipulation are supported for nested partitions. Objects like pins, pin guides, pin blockages, bus guides etc are clearly shown. Pins are automatically (or manually) assigned on all levels of partitions on boundaries of nested partition.

**Note:** With this release the system supports partition inside partition in Low Power hierarchical flow. This means that `savePartition` can partition CPF for partiton with nested partition.

## Defining Nested Partitions

Use the `definePartition` command to specify partitions inside a partition. While specifying partitions, you can specify the definition in any order.

For example, if `PTN1` is a parent partition and `PTN2` is a child then the following commands give the same results:

```
definePartition -Hinst PTN1  
definePartition -Hinst PTN2  
  
definePartition -Hinst PTN2  
definePartition -Hinst PTN1
```

## Pin Assignment Across Nested Partitions

The `assignPtnPin` command performs automatic pin assignment of partitions inside other partitions. It places pins of nested partitions similar to single level partitions. However, while assigning pins for a single level design the aim is to achieve maximum alignment possible between pins. For nested partitions top pins are aligned first to reduce the top channel and model congestion inside partitions. Thereafter, pin assignment is done to achieve maximum possible alignment for nested partitions.

During master/clone pin assignment for nested partitions, pins are assigned such that they have comparable maximum misalignment in all scenarios. The `editPin` command is used for manual pin assignment in nested partitions. It places pins of nested partitions similar to single level partitions. Trial Route routes switches effecting hierarchical routes and honors pins at different levels of partition. It considers pin guide constraints present on the Nth level of partition and performs checks to avoid violations. The `trialRoute -handlePartition` and `trialRoute -handlePartitionComplex` commands identify nested partitions and honor boundaries of both child and parent partitions.

## Pin Checking and Legalization Across Nested Partitions

The pin checker and legalizer capabilities check and legalize partition and black box pins for all levels of partitions. The `checkPinAssignment` command reports the pin status for pins of nested partitions and is aware of shapes both inside and outside of the partition boundary of nested partitions.

The `legalizePin` command has also been enhanced to be aware of nested partitions and correct the pin position of illegal pins across hierarchies. It gives a warning if the first level pin is placed in second level partition or vice versa. However, it legalizes pins only for track placement. It also supports pin overlapping across N levels for overlapping partition boundaries.

## Handling Pin Objects Across Nested Partitions

To guide automatic pin placements for nested partitions and control pin positions, the pin objects (pin groups, net groups, pin guides, and pin blockages) are handled in the following manner:

- `createPinGroup`

Individual constraints should be defined for each hierarchical module since a pin group is associated with cells. In order to create a pin group for nested partitions, the `-cell` parameter must be used.

For example, the following command defines different spacing constraints for N level and N-1/+1 partitions.

```
createPinGroup -spacing 4 -cell PTN1  
createPinGroup -spacing 2 -cell PTN2
```

- `createNetGroup`

Since a net is a hierarchical object and is not associated with a cell, the same object can work for multiple levels of hierarchy.

- `createPinBlkg`

Since it has no specific element attached to it, it can be propagated to N+1/N-1 levels as well. In order to create pin blockages for nested partitions, the `-cell` parameter is not required. It applies to all partitions whose boundary it is touching. The `-cell` is only required when the `-edge` parameter is used.

- `createPinGuide`

In case of nested partitions, a common pin guide can be created for a net group. However, for

creating a pin guide for a individual pin groups spread across nested partitions, the `-cell` parameter is required.

## Committing Nested Partitions

The `partition` command inherits the physical cells inside correct partitions irrespective of the sequence of the partition definition. The `partition` command commits all partitions (parent and child) in a bottom up fashion. The parent is represented as a HARD MACRO at top level and child as a HARD MACRO at parent level. To make any changes in a child, both the parent and child partitions need to be flattened (parent and then child) in a sequential manner.

## Assembling Nested Partitions

The incremental assemble design capability brings back partition data for nested partitions. It first restores the top design, assembles the parent partitions, and then brings back all child nodes partitions. It ensures that all references of master and clones (which may be at different levels of hierarchy in different partitions) are assembled properly.

## Assigning Pins

You can optimize partition and blackbox pins in the Innovus environment based on routing or placement information. You can assign the pins or ports to a location on a partition, and set various constraints as per your requirements on pin assignment, for example, you can create pin blockages on specified areas. Run the Check Pin Assignment menu command of the [Partition Menu](#) or the `checkPinAssignment` command after pin assignment to make sure that all pins are assigned, are placed on routing grids, and are not overlapping.

Blackbox pins are assigned in the same way as partition pins.

Pin assignment supports the following:

- Rectilinear partitions and black boxes
- Repeated partitions and black boxes. Both master and clones are considered when assigning their pins.
- Designs with an arbitrary origin.
- Non-uniform tracks.

**Note:** Pin assignment assigns only signal pins but it does honor power/ground stripes and follow

pins. Power and ground pins are created when the design is partitioned.

**Note:** The pin assignment commands have been updated to honor the preferred routing layer attributes as soft constraint during pin assignment. Pin assignment commands try to honor the `setAttribute -top_preferred_routing_layer` and `-bottom_preferred_routing_layer` attributes. However in case they cannot be honored, the pin is assigned to any allowed layer.

The following sections describe pin assignment in Innovus:

- [Assigning Partition and Blackbox Pins](#)
- [Assigning I/O Pins](#)
- [Performing Congestion-aware Pin Assignment for Channel-based Designs](#)
- [Assigning Pins on Rectilinear Edges](#)
- [Swapping Partition Pins](#)
- [Snapping Pins to the Grid](#)
- [Assigning Pins for Bus Guides](#)
- [Pin Assignment Limitations](#)

## Assigning Partition and Blackbox Pins

Assigning pins for partitions and blackboxes includes the following steps:

- [Setting Pin Constraints](#)
- [Assigning Pins](#)
- [Validating Pin Placement Results](#)
- [Refining Pin Assignment and Fixing Pin Violations](#)
- [ECO Pin Assignment](#)

## Setting Pin Constraints

The Innovus software provides a number of constraints to control or guide partition, blackbox, or I/O pin assignment:

- [Pin Group](#)
- [Net Group](#)
- [Pin Guides](#)
- [Pin Size \(Width and Height\)](#)
- [Pin Spacing](#)
- [Pin Layers](#)
- [Pin-to-corner distance](#)
- [Pin Blockage](#)

## Pin Group

While assigning bus pins or signal pins that you want to be placed together, you can specify a constraint for these pins by creating a cell pin group. You can create a cell pin group with the `createPinGroup` command or by using the [\*Edit Pin Group\*](#) form (*Edit - Edit Pin Group*). You can add pins to a cell pin group with the `createPinGroup` command or with the `addPinToPinGroup` command.

Cell pin groups do not have to be associated with a partition pin guide because a pin group is not a constraint on any partition edge. In this case, the pin assignment program can freely place this group of pins on any edge of the partition. However, pins that belong to this pin group are still placed together in adjacent locations.

With a pin group you can:

- Optimize the order of pins within a cell pin group to improve wire length using the `-optimizeOrder` option of the `createPinGroup` command. If this option is not specified, the pin order is exactly as specified in the pin group.
- Specify pin spacing. The default minimum pin spacing between pins of a cell pin group is two tracks.

The following commands create a pin group `pGroup1` that consists of 3 `INT` bus bit pins of the module `ALU`. These pins can be optimized within the pin group:

- `createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -optimizeOrder`  
or
- `createPinGroup pGroup1 -cell ALU -optimizeOrder`  
`addPinToPinGroup -cell ALU -pinGroup pGroup1 -pin {INT[0] INT[2] INT[3]}`

Use the `deletePinGroup` command to delete a pin group or all pin groups and use the `deletePinFromPinGroup` command to delete a pin from a pin group.

## Net Group

You can create a net group using the `createNetGroup` command or by using the *Edit Net Group* form (*Edit - Edit Net Group*). You can specify net members when creating a net group or add them later using the `addNetToNetGroup` command. To be honored by pin assignment, net groups must be used in conjunction with a pin guide. As for a pin group, you can optimize the net pin order, alternate the pin layers, and specify pin spacing for a net group.

The following commands create a net group `nGroup1` that has two nets `NET1` and `NET2` with minimum pin spacing of 2 tracks.

- `createNetGroup nGroup1 -net {NET1 NET2} -spacing 2`  
Or
- `createNetGroup nGroup1 -spacing 2`  
`addNetToNetGroup nGroup1 -net {NET1 NET2}`

Use the `deleteNetGroup` command to delete a net group or all net groups and use the `deleteNetFromNetGroup` command to remove a net from a net group.

**Note:** When you delete a net group, any bus guide associated with the net group also gets deleted.

## Pin Guides

You can create a pin guide to constrain a bus, net, pin, net group, or pin group to be placed in specific areas. A pin guide is used for specifying a physical guided area where pins belonging to the pin guide will be placed. A pin guide can support multiple constraint pin layers. In addition, any bus, net, pin, net group, or pin group can be assigned to multiple pin guides. You can create a pin guide using the *Create Pin Guide* widget from the GUI or through the `createPinGuide` command.

Note: While creating a pin guide, you cannot optimize the order of pin members or specify minimum spacing. If you want to control the pin order and the pin spacing of the members of a pin guide, first create a net group or a pin group and associate this net group or pin group with a pin guide.

A physical location constraint can be specified either as a rectangular area or as an edge constraint. If you specify a physical location constraint as an edge constraint, you will also need to specify the partition/black box cell name.

## Examples

Here are a few examples of using the `createPinGuide` command to create a pin guide.

- The following command creates a pin guide for a net group nGroup1. The pin order within this net group will be optimized. The pin members of this pin guide can be placed on the top edge of the cell ALU. Pins will be placed on Metal2 or Metal4 layers:

```
createNetGroup nGroup1 -net {NET1 NET2} -optimizeOrder  
createPinGuide -netGroup nGroup1 -edge 1 -cell ALU -layer {Metal2 Metal4}
```

- The following command creates a pin guide for a pin group pGroup1 of cell/module ALU. Pins of this pin guide will have a minimum spacing of 2 tracks:

```
createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -spacing 2  
createPinGuide -area 678.52 371.25 778.53 787.33 -pinGroup pGroup1 -cell ALU
```

The pins will be assigned on the preferred layers.

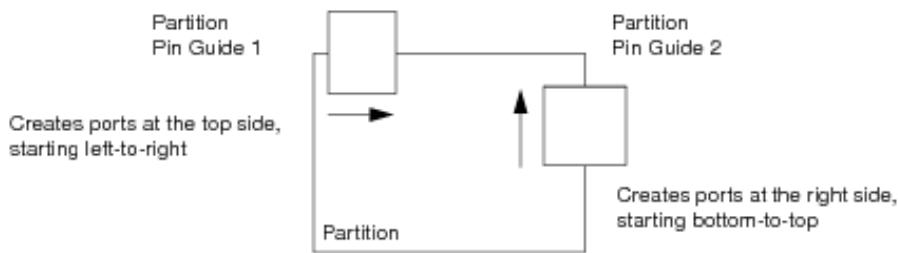
- The following command creates a pin group pGroup2. This pin group can be placed on the top edge or the right edge of the cell TDSP. For top edge, pins can be assigned on the Metal4 or Metal6 layers. For right edge, pins can be assigned on the Metal5 layer.

```
createPinGroup pGroup2 -cell TDSP -pin p_addr* -optimizeOrder  
createPinGuide -edge 1 -pinGroup pGroup2 -cell TDSP -layer {Metal4 Metal6}  
createPinGuide -edge 2 -pinGroup pGroup2 -cell TDSP -layer Metal5
```

You can also use the GUI to create a partition pin guide. After you have determined a pin guide location in the design display area, you can create a partition port for a net or bus name and add a partition pin guide.

To add a partition pin guide through the GUI, complete the following steps:

1. Select Edit - Create Pin Guide to display the *Edit Pin Guide* GUI form. Use this form to specify the pin guide name, cell name, mode (by area or by edge), and the applicable layers.  
Alternatively, click the *Create Pin Guide* widget and press the F3 key.
2. Click and drag over a partition fence overlap to specify the area or edge. For vertical edges, the first pin generated starts from the bottom intersect point. For horizontal edges, the first pin generated starts from the left intersect point, as shown in the following figure:



The default pin spacing is 2, which places one pin for every two metal tracks. You can change the pin spacing with the *Minimum Pin Pitch* field in the [Specify Partition](#) form, or by changing spacing of the associated pin group or net group.

You can use the [Move/Resize/Reshape](#) tool to modify the pin guide location.

**Note:** For a partition that has a rectangular cut, the partition pin guide must be placed on the edge of the cut. You can also use a pin guide to assign pins, net group, or a pin group to a specific side without specifying a pin guide area by using the [createPinGuide](#) command.

3. Change the partition pin guide object name to the net, bus, or net group name.

Use the partition pin guide attribute editor to change pin guide name to a net name, or the name of a predefined net group or pin group. If the partition pin guide was assigned the net group name, all nets and buses added to this net group name will have partition pins generated for the partition. Pins are generated in the order the net or bus was entered by the [addNetToNetGroup](#) command. Pins for unconnected nets and buses are randomly assigned. You can also use the partition pin guide to assign floating pins.

Use the [deletePinGuide](#) command to delete a pin guide or all pin guides.

## Pin Size (Width and Height)

By default, pin size will be created based on the minimum area rule. Use `-width` and `-depth` parameters of the [setPinConstraint](#) command to set the new pin width and depth of a routing layer for a specific partition/black box cell. When this constraint is defined, pin assignment will use these values for creating pin size.

Use the `-width` and `-depth` parameters of the [getPinConstraint](#) command to retrieve pin width and depth for particular layer(s) of specific partition/black box cell.

### Examples

- The following commands set the pin width and depth of layer Metal2 for partition cell ALU to

0.4 and 0.6 respectively.

```
setPinConstraint -cell ALU -layer Metal2 -width 0.4  
setPinConstraint -cell ALU -layer Metal2 -depth 0.6
```

- The following commands set the pin width of pin pGroup1 to 0.3 and pin depth of pin pGroup1 to default.

```
setPinConstraint -cell ALU -pin pGroup1 -width 0.3  
setPinConstraint -cell ALU -pin pGroup1 -depth default
```

With this example, all the pins of pGroup1 will have the width 0.3 and the default depth.

## Pin Spacing

You can set minimum pin spacing in terms of track number using the [Specify Partition form](#) (*Partition - Specify Partition*). The default pin spacing is 2, which places a pin for every two metal tracks. You can modify the pin spacing in the following ways:

- Global pin spacing at design level

Use the `-global` and `-spacing` parameters with the `setPinConstraint` and `getPinConstraint` commands to set and retrieve global pin spacing, respectively. This spacing value will be applied to all partition/black box pins of the design.

- Partition/black box level

Use the `definePartition` command with `-minPitchTop`, `-minPitchBottom`, `-minPitchLeft`, and `-minPitchRight` parameters to specify minimum pin spacing for a partition. Similarly, to specify the minimum pin spacing for a blackbox, use the `specifyBlackBox` command with `-minPitchTop`, `-minPitchBottom`, `-minPitchLeft`, and `-minPitchRight` parameters.

- Specific partition/black box area or edge

Use the `-edge` and `-spacing` parameters with the `setPinConstraint` and `getPinConstraint` commands to set and get the minimum pin spacing for a particular edge or all edges of a partition/black box cell.

The `-edge` parameter of the `setPinConstraint` and `getPinConstraint` commands can take the following values:

- N, S, W, E (supports both upper and lower case)
- T, B, L, R (supports both upper and lower case)
- dbcN, dbcS, dbcE, dbcW
- The following commands set the minimum pin spacing for top and bottom edge of partition cell ALU to 1 track.

```
setPinConstraint -cell ALU -edge T -spacing 1
setPinConstraint -cell ALU -edge B -spacing 1
```
- The following command sets minimum pin spacing for all edges of partition cell TDSP to 3 tracks

```
setPinConstraint -cell TDSP -all -spacing 3
```

**Note:** Use the [unsetPinConstraint](#) commands to unset the minimum pin spacing, for a specified module, area, or edge(s), that was defined with the [setPinConstraint](#) command.

- Pin group or net group

Use the [createPinGroup](#) or the [createNetGroup](#) commands to specify minimum pin spacing at the pin group or net group level. This specified minimum pin spacing will apply to all the pin members of the specified pin group or net group.

- Pin level

Use the [setPinConstraint](#) command to specify minimum pin spacing of a particular pin.

As spacing constraint can be specified at more than one level, pin assignment will honor spacing constraint in the following order:

- Pin spacing
- Net group or pin group spacing
- Partition/black box spacing on a particular edge
- Partition/black box spacing
- Global spacing

## Pin Layers

Specify pin layers that will be used for placing pins on a specific partition side using the [Specify Partition](#) form (*Partition - Specify Partition*). Alternatively, you can use the `setPinConstraint -layer -edge` command.

**Note:** Using the `setAttribute -top_preferred_routing_layer` and `-bottom_preferred_routing_layer` parameters, the preferred routing layer attributes are attached to nets. These attributes are honored throughout the routing process. The pin assignment commands honor these preferred routing layer attributes as soft constraint during pin assignment. This provides a better co-relation with the NanoRoute Router as it considers these as soft constraints. If a pin cannot be placed on these layers, the pin will not be unassigned. Pin assignment commands try to honor these attributes however in case they cannot be honored, the pin is assigned to any allowed layer.

If only one of the attributes is given, the other is assumed from the lowest or the highest value of the allowed layer list. If `-top_preferred_routing_layer` is applied, then -

`bottom_preferred_routing_layer` is assumed to be the lowest layer of allowed layer list.

Alternatively, if `-bottom_preferred_routing_layer` is applied, then -

`top_preferred_routing_layer` is assumed to be the highest layer of allowed layer list.

You can specify layer constraints at partition level, pin guide level, or pin level.

- Partition level

Layer constraint per edge can be specified at partition level using either:

- The [Specify Partition](#) form (*Partition - Specify Partition*), or
- The `definePartition` command with `-pinLayerTop`, `-pinLayerBottom`, `-pinLayerLeft`, and `-pinLayerRight` parameters. These layer constraints will be applied to all pins on a particular edge of the specified partition.
- The `setPinConstraint` command with the `-layer` and `-edge` options. This command sets the pin layers for a specified edge or for all edges.

**Note:** Use the `getPinConstraint` command to retrieve the pin layers for a specified edge or for all edges.

- Pin guide level

Use the `-layer` parameter of the `createPinGuide` command to specify layer constraints for all pin members of a pin guide. Layer constraint at pin guide will override the layer constraint at partition level.

- Pin level

Use the `-layer` parameter of the `setPinConstraint` command to specify layer constraint for a specific partition/black box pin.

**Note:** Layers can be specified using the LEF layer names or layer ID numbers.

Layer constraint at pin level will have higher priority than layer constraint at partition level.

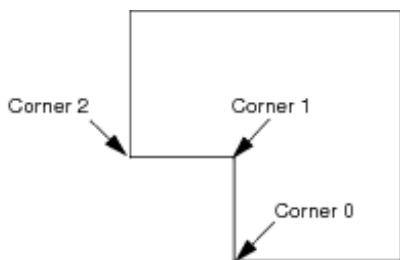
- If a layer constraint is applied to a pin that also belongs to a pin guide, the pin guide layer constraint will have higher precedence.
- If a layer constraint is being applied to a pin that already belongs to a pin group or net group, the layer constraint will not be applied. To apply layer constraint for this pin, first remove this pin from the pin group or net group, and then apply the pin layer constraint.

## Pin-to-corner Distance

To keep pins away from partition/black box corners, you can set the pin-to-corner distance constraint.

- Use the `setPinConstraint -corner_to_pin_distance` command to set pin to corner distance for a particular corner or all corners of a specific cell.
- Use the `getPinConstraint -corner_to_pin_distance` command to retrieve the pin-to-corner value of a cell-specific corner or all corners.
- Use `setPinConstraint -global -corner_to_pin_distance` to set global pin-to-corner distance that will be applied to all partition and blackboxes in the current design. The default value is 5 routing tracks.

The `-corner cornerNumber` parameter of the commands specifies the corner of the partition block. This is an integer value, where corner numbering starts at 0 from the lower-left corner of a partition clock-wise. Corner 0 is the corner that has the smallest y value.



## Example

The following command sets pin-to-corner distance for corner 0 and corner 2 of the cell ALU to 8 routing tracks.

```
setPinConstraint -cell ALU -corner 0 -corner_to_pin_distance 8  
setPinConstraint -cell ALU -corner 2 -corner_to_pin_distance 8
```

## Pin Blockage

After determining the partition pin blockage point, you can block that area from assigning pins on specific metal layers. Pin assignment engines also honor regular routing blockages if they intersect with partition edges. You can create pin blockages with the *Create Pin Blockage* widget or by using the [createPinBlkg](#) command.

**Note:** Trial Route does not honor the partition pin blockage.

To create the partition pin blockage with the *Create Pin Blockage* widget, complete the following steps:

1. Click the *Create Pin Blockage* widget from the *Toolbox*. Alternatively, select Floorplan - Edit Floorplan - Create Pin Blockage.
2. Left click and drag over a partition fence overlap.
3. Use the [Attribute Editor](#) to specify the metal layers to block.

The following command creates a pin blockage for the entire left edge of cell TDSP on layer M5.

```
createPinBlkg -edge 0 -cell TDSP -layer 5
```

If the `-layer` option is not specified, the pin blockage will be created on all partition/black box reserved routing layers.

Use the [deletePinBlkg](#) command to delete a pin blockage or all pin blockages (`deletePinBlkg -all`).

**Note:** You can create pin blockage on master or clone instances. Pin blockages are transformed on both master and clones enabling you to visually see the pin blockage on both master and clone partitions. If you delete or modify any of such pin blockages, all its variants with same corresponding box on master/clones will be deleted.

## Performing Pin Pre-Assignment

You can pre-assign a pin before pin assignment using the [Pin Editor](#) or the `editPin` command. These pre-assigned pins will have fixed placement status so pin optimizers will honor them. For more details, see the [Pin Editor](#) section in the "Edit Menu" chapter of the *Innovus Menu Reference*.

## Setting Constraints on a Specific Pin

Use the `setPinConstraint` command to specify the following constraints for a particular pin:

- Physical location  
A pin can be constrained by specifying its coordinate (x, y) location and its preferred routing layer. If specified location is not on valid track, the pin will be snapped to the closest location. To keep the pin on non-preferred routing layer or to not snap the pin, use the `editPin` command instead.  
Besides an actual physical location, a pin can also be constrained to a particular edge.
- Layer
- Spacing

The following command specifies that the pin reset of partition cell mult\_32 should be placed on top edge with either Metal5 or Metal7 routing layer.

```
setPinConstraint -cell mult_32 -pin reset -edge 1 -layer {Metal5 Metal7}
```

For setting pin size constraint for a specific pin use the `setPinConstraint` command with the `-pin -width -depth` parameters.

The following salient points apply to setting the pin constraints for a specific pin:

- If constraints are applied to a pin that also belongs to a pin guide, the pin guide constraint will have higher precedence.
- If a location and/or layer constraint is being applied to a pin that already belongs to a pin group or a net group, the constraint will not be applied. To apply location and/or layer constraint for this pin, first remove this pin from the pin group or net group, and then apply the pin constraint(s).
- If a pin with layer constraints defined is added to a net group or pin group, the pin cannot be added to a pin group or a net group with the `createPinGroup`, `createNetGroup`,

`addPinToPinGroup`, or `addNetToNetGroup` commands because the pin has already been constrained. To add this pin to a pin group or net group remove the constraints first (using the `unsetPinConstraint` command).

- If the following constraints cannot be met during pin assignment, the Innovus software will issue a warning message and the constrained pins will be placed at the lower-left corner of the partition/black box with unplaced placement status:
  - Pin constraint
  - Pin group constraint
  - Net group constraint

Use the `unsetPinConstraint` command to remove constraint settings for a specific pin.

## Assigning Pins

There is no separate step required for assigning black box pins. To assign pins, use the *Partition - Assign Pins* GUI menu or the `assignPtnPin` text command.

Pin assignment supports the following:

- Rectilinear partitions and black boxes
- Repeated partitions and black boxes.
- Non-uniform tracks

Pin assignment assigns signal pins but it does honor power/ground stripes and followpins. Power and ground pins will be created during the partition step.

## Placement-based Pin Assignment

Pin assignment is based on connectivity flightlines. Cell placement should be performed before running pin assignment.

## Route-based Pin Assignment

For route-based pin assignment, routing should be performed prior to the `assignPtnPin` command. Routing cross points with partition/black box boundary will be used as guidance for pin assignment. For a design that has blackboxes, if you want to have near-optimal locations for black box pins, the

-routeBasedBBPin option should be used. The differences between Trial Route with and without -routeBasedBBPin options are as follows:

Default Trial Route performs the following:

- Assigns initial black box pins based on connectivity
- Creates temporary routing blockages over black boxes based on black box reserved routing layers
- Runs trialRoute to route to black box pins
- Removes temporary blockages

Trial Route with the -routeBasedBBPin parameter performs the following:

- Shrinks black box boundary and runs connectivity based pin assignment to get initial pin location
- Runs partition-aware routing
- Re-adjusts black box pins to actual black box boundary based on routing cross point
- Creates temporary routing blockages
- Runs trialRoute to route to black box pins
- Removes temporary blockages

For channel-based designs that have thick channels, instead of using trialRoute -handlePartitionComplex, it is recommended to run trialRoute -fastRouteForPinAssign.

However, if the design has black boxes then you can run Trial Route with -routeBasedBBPin and -handlePartitionComplex options.

## Tips for Assigning Partition Pins

For most of the designs, running the `assignPtnPin` command without any option should give a reasonable result. However specific options can provide better results in some cases. These options are described here:

- -maxPinMovementForAlign parameter

If you have ran partition aware routing (`trialRoute -fastRouteForPinAssign` or `trialRoute -handlePartitionComplex`) for pin assignment, you should use these parameters to minimize pin movement from existing routing cross points because these routing cross points should give near-optimal pin locations.

Example:

```
trialRoute -fastRouteForPinAssign  
assignPtnPin -maxPinMovementForAlign 20
```

- `-ptn ptnName -pin pinList` parameter

Use this parameter for running incremental pin assignment or assigning specific pins of specific partitions.

This parameter can be used in the following pin assignment scenarios:

- When you want to assign critical pins first and then assign the rest of partition and/or black box pins.
- First, run pin assignment to assign these critical or specific pins. Use the above option in conjunction with the `-markFixed` parameter so these pins will not be moved by second pin assignment run.
- Run pin assignment again to assign the rest of the pins.

Example:

```
assignPtnPin -ptn tdsp_core_glue -pin {p_address[0] p_address[3]} -ptn alu_32  
-pin rom_data* -markFixed  
assignPtnPin
```

In the previous example, if you are running routed based pin assignment, you should run `trialRoute` between the first and the second pin assignment run so that the routing that will be used for the second pin assignment is based on pin locations of the first pin assignment step.

- Run incremental pin assignment

This scenario is in contrast to the first scenario where you would run pin assignment for all partition and/or black box pins, and then further re-optimize some specific pins.

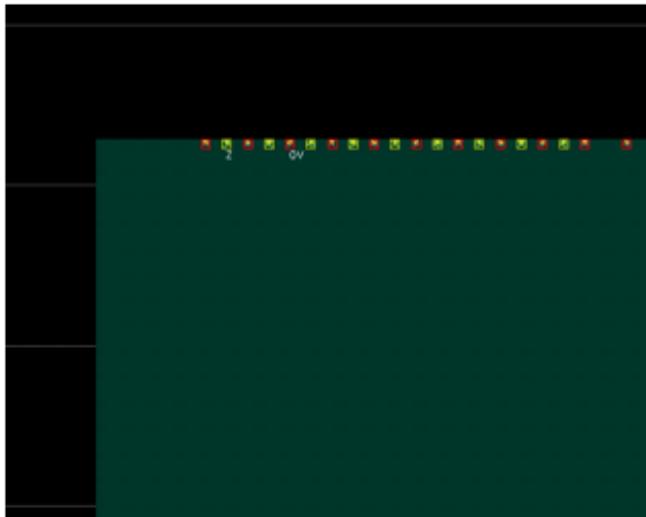
Example:

```
assignPtnPin  
assignPtnPin -ptn mult_32 -pin {reset addr*}
```

If reset and all addr pins of the partition `mult_32` have fixed placement status, you should also use `-moveFixedPin` option; otherwise pin optimizer will not move fixed pins.

- `-noPinLayerOverlap` parameter

Use this parameter if you do not want the pin optimizer to place signal pins overlapping each other on different layers. This option can be used to avoid DRC violations between adjacent pins and the routing connecting to these pins.



The previous figure shows that adjacent pins are placed on alternate layers M2 and M4.

- `-enforceRoute` parameter

With this parameter, pin assignment completely follows the routing information without honoring any user-specified pin constraints and pin locations may not be legal. This option should only be used for a rough pin assignment or for comparing pin locations based purely on routing result with pin locations that are legalized. If you want to use this pin placement result for your implementation stage, you need to run the `legalizePin` after the `assignPtnPin` command to legalize them.

## Validating Pin Placement Results

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

- [Checking the Pin Legality](#)
- [Reporting QoR of Pin Assignment](#)
- [Adjusting Pins](#)
- [Aligning Partition Pins](#)
- [Running Incremental Pin Assignment](#)
- [Adjusting Floorplan or Floorplanning the Design Again](#)
- [Performing Pin Assignment Again](#)

## Checking the Pin Legality

Use the [checkPinAssignment](#) to make sure that pins are legalized (for example, the pins snap to routing grid, are on reserved routing layers, honor user-specified constraints, do not cause any DRC violations, and so on).

You can check

- All partition/black box pins

Example: The following command checks all partition/black box pins in the current design and saves the result into the output file pinLegality.rpt.

```
checkPinAssignment -outFile pinLegality.rpt
```

- All pins of a specific partition

Example: The following command checks all pins of the partition TDSP\_CORE

```
checkPinAssignment -ptn TDSP_CORE -pin *
```

- Specific partition pins

Example: The following command checks all bus pins p\_addrs and rom\_data of the partition

TDSP\_CORE

```
checkPinAssignment -ptn TDSP_CORE -pin {p_addrs* rom_data*}
```

**Note:** You can use the -verbose parameter of the [checkPinAssignment](#) command to print detailed pin-specific information for each reported violation. You can also exclude certain checks, for example, checks related to pin spacing violation or pin layer violation. For more information, see the description of the [checkPinAssignment](#) command in the *Innovus Text Command Reference*.

From 14.2 release, the [checkPinAssignment](#) command highlights violations on individual pins by marker that covers the full pin shape. It checks violations for pin\_abutment, pin\_depth, pin\_layer, pin\_min\_area, pin\_on\_fence, pin\_on\_track, pin\_spacing, pin\_width, and logical\_pin.

If there is a marker on the master pin, marker on clone pins is not highlighted. The [checkPinAssignment](#) command will check for individual violations, which are different than master, on clone pins as well. For instance, spacing to an object near clone. The markers that are specific to clones will be reported by [checkPinAssignment](#).

For violations of guide, such as bus\_guide and pin\_guide, if there is violation in individual pins, markers will be on individual pins only.

For violations of group, pin\_group and net\_group, the first pin offending the group order will be highlighted as violation and all the pins from first pin to the last pin of the group will be highlighted

as aggressor. If the pin group is traversing multiple edges then multiple aggressor type violations will be generated. The graphic below shows an example of violations marked for pins.



In this example, there is a pin group a1, a2, a3, a4, a5, a6, a7, a8. After the placement, the order of the pins is changed to a1, a3, a2, a4, a8, a5, a7, a6. In this case, the first pin violating the order is a3, and therefore will be marked by a violation marker. Also, to reduce the number of victims, entire group from the first pin to the last pin is highlighted.

In this example, pin group is on three edges.

- On edge 1, first pin of the group is a1 and the last pin is a2.
  - On edge 2, there is only one pin a4.
  - On edge 3, the first pin is a8 and the last pin is a6.

Therefore, three victim markers will be drawn from first to last pin on each edge. These victim markers may cover other pins which fall between the first and the last pin, such as pin B and pin E on edge1 and edge 3, respectively.

## Reporting QoR of Pin Assignment

You can use the `pinAnalysis` command to report certain Quality of Results (QoR) metrics for pin assignment. The `pinAnalysis` command deletes the existing routes, reroutes the design ensuring that the routes pass through partition pins, and reports pin assignment QoR metrics.

**Info:** The `pinAnalysis` command creates new routes. The original routes are not retained.

**Note:** The `pinAnalysis` command reroutes the design using `trialRoute -honorPin`. This command thus takes at least as much time as running Trial Route. Also, because Trial Route is run, you can use the generated routing information for other applications such as time budgeting.

When you run the `pinAnalysis` command after assigning pins but before committing the partition, the following are reported:

- Pin-deviation from routing cross-points
- Estimated net-length and via-count
- Comparison between net lengths, via counts, and congestion indexes of the original and the new routes

**Note:** *Original route* refers to the routing that was performed before the pin assignment step. *New route* refers to the routing performed by the `pinAnalysis` command.

- Number of two-pin nets that have aligned pins and number of two-pin nets that have unaligned pins
- CPU time and memory usage

When you run the `pinAnalysis` command after committing the partition, the following are reported:

- Estimated total top-channel net-length and via-count
- Congestion report with overall congestion index values for top-level channel nets
- Number of two-pin nets that have aligned pins and number of two-pin nets that have unaligned pins
- Run time and memory usage

You can check the legality of pin assignment (similar to the functionality of `checkPinAssignment` command) by specifying the `-checkLegality` parameter with the `pinAnalysis` command. You can also save the output of the command in a text file by specifying the `-outFile` parameter. In addition to displaying the report on screen, this command also generates the report in an HTML file named `pinAnalysis.html`. The legality details for every partition are available through hyperlinks in the HTML report file.

As an example, the following command checks the legality of pin assignment, displays the QoR

report for pin assignment on the screen, and also prints the QoR report to a file named pinAnalysisMetricReport.

```
pinAnalysis -checkLegality -outFile pinAnalysisMetricReport
```

The output of the previous command is similar to the following:

```
Analysing Pin Assignment.....
```

```
=====
Pin Legality Report:
-----
```

Partition	Total	Internal	unplaced	Legal	Illegal
Name	Pins	Pins	Pins	Pins	Pins
results_conv	39	0	39	0	0
tdsp_core	114	0	114	0	0
dtsmf_chip	0	0	0	0	0

```
=====
-----
```

```
Pin QoR Report:
-----
```

```
Unaligned 2-pin Net: DTMF_INST/t_addrs[0].  
Unaligned 2-pin Net: DTMF_INST/m_clk.  
There are 2 unaligned 2-pin nets.
```

```
=====
| QoR Metric | Before Pin-Assignment | After Pin-Assignment | Percent Increase |
-----  
| Horizontal | 0.000% | 0.032% | NA |  
| Congestion | | | |  
-----  
| Vertical | 0.000% | 0.697% | NA |  
| Congestion | | | |  
-----  
| Total | 6.089e+05um | 6.150e+05um | 1.011% |  
| Net-Length | | | |  
-----  
| Total | 48300 | 54036 | 11.876% |  
| Via-Count | | | |  
-----
```

Completed pinAnalysis (CPU=0:00:07.6 MEM=5.9)

**Note:** The internal pins (shown in the Legality Report) are not checked for legality. Internal pins are the pins that are not on the partition boundary.

In the previous table, the `Before Pin-Assignment` column shows the results of the routing before pin assignment and the `After Pin-Assignment` column shows the results of the routing after pin assignment.

## Refining Pin Assignment and Fixing Pin Violations

After assigning partition or blackbox pins, you can further refine the current pin assignment and fix any pin violations using one or more of the following methods:

- [Adjusting Pins](#)
- [Aligning Partition Pins](#)
- [Running Incremental Pin Assignment](#)
- [Adjusting Floorplan or Floorplanning the Design Again](#)
- [Performing Pin Assignment Again](#)

These steps are explained in the following sections.

### ***Adjusting Pins***

You can Adjust pins using the [Pin Editor](#) or the `editPin` text command. You can also use direct pin manipulation to manually move selected pins to different locations.

### ***Aligning Partition Pins***

You can align partition pins with other block pins (using the Pin Editor or the `pinAlignment` text command).

The `pinAlignment` command can be used to align partition/black box pins with or without specified reference object(s). Reference objects can be hard macros, blackboxes, I/O pads, standard cells, and ptn pin.

You can use the `pinAlignment` command in different ways to align pins:

- Align specific pins with the specified referenced object

```
pinAlignment -refObj refInstName -ptnInst instName -pinNames pinList
```

- Align all pins of specified partition/blackbox instance that connect with the specified reference object

```
pinAlignment -refObj refInstName -ptnInst instName
```

- Align all pins of every partition/blackbox that connects with the specified reference object

```
pinAlignment -refObj refInstName
```

- Align specific pins of specified partition/blackbox instance

```
pinAlignment -ptnInst instName -pinNames pinList
```

- Align all pins of specified partition/blackbox

```
pinAlignment -ptnInst instName
```

- Align all possible partition/blackbox pins

```
pinAlignment
```

If the referenced object is not specified, the `pinAlignment` command will automatically derive referenced object based on connectivity information. If the aligned pin has multiple connections, the referenced object will be derived based on the following priority:

- Hard macro pin
- I/O pad pin or I/O pin
- Partition pin
- Standard cell pin

By default an aligned pin will:

- Be snapped to routing grid. Use `-noSnap` option if you want that pins should not be snapped.
- Have the same layer routing with the referenced pin. Use the `-keepLayer` option to keep existing aligned pin layer. Use the `-newLayer` option to assign new layer for aligned pin.
- Not be legalized. Use the `-legalizePin` option to legalize aligned pin(s).
- Have a fixed pin status. Use the `-markPlaced` option to assign placed status to aligned pin(s).

## ***Running Incremental Pin Assignment***

Based on the current pin assignment result, re-run pin assignment. You can specify pin constraints to further guide new pin placement. If you want to reoptimize only a few specific pins, use the `-ptn` and the `-pin` options of the `assignPtnPin` command to specify the list of pins that will be reassigned.

Example: The following command reoptimizes address bus bit pins, rom\_data bus bit pins of partition ALU, and reset pin of partition ARB:

```
assignPtnPin -ptn ALU -pin {address* rom_data[*]} -ptn ARB -pin reset
```

By default, `-ptn` and `-pin` options will not reassign specified pins if they have fixed status. Use the `-moveFixedPin` option with the `-ptn` and `-pin` options to force specified fixed pins to be reassigned. However if you want to keep only a few existing pins and re-optimize the rest of the pins, instead of specifying many pins, you can mark these existing pins to fixed placement status using the `setPtnPinStatus` command and then re-run pin assignment :

```
setPtnPinStatus -cell * -pin * -status fixed  
assignPtnPin
```

## ***Adjusting Floorplan or Floorplanning the Design Again***

Sometimes a sub-optimal floorplan can also lead to a bad pin placement result. If this is the case, re-adjust the floorplan and run pin assignment again.

## ***Performing Pin Assignment Again***

Perform pin assignment again. If the partitions or blackboxes have been committed, use the `flattenPartition` command to unassign the pins. If the partitions or blackboxes are not yet committed, use the `setPtnPinStatus` command to unplace the pins.

## ECO Pin Assignment

The Innovus software provides incremental or ECO pin assignment capability. This capability can be used in the ECO flow where partition/black box ports in the original netlist get modified (added/deleted). In this flow, you can preserve most of the existing partition/black box pin locations and let the software assign the newly added pins.

### General Flow

1. Import design.
2. Floorplan design (specify partition/black box information in this step).
3. Run placement.
4. Route design.
5. Save full chip floorplan and/or save design for later use.
6. Assign pins (`assignPtnPin`).
7. Save partition/black box pin information into a partition file (`savePtnPin`).
8. Route design to connect to new partition/black box pins (`trialRoute -honorPin`).
9. Derive timing budget (`deriveTimingBudget`).
10. Commit partitions/black boxes (`partition`).
11. Save top and partition information into each directory (`savePartition`).

After having new modified netlist, ECO pin assignment can be run as follows:

12. Import design with new modified netlist.
13. Load full-chip floorplan that saved in the previous step 5.
14. Place and route the design. Placement and routing information that are saved in the step 5 can be restored if still applicable.
15. Use the `loadPtnPin` command to load the partition file that was saved in the previous step 7 or the partition file (or DEF file) of each partition block to preserve existing partition/blackbox pin locations. Make sure that partition/blackbox pins in partition file have fixed placement status so they will not be moved in the next step, pin assignment.
16. Run pin assignment to assign the newly added pins.

### Saving the Partition Pins

Use the `savePtnPin` command to save pin placement information for later use. The command provides options to save pin information of

- Specific partition/blackbox

Example: Save pin locations of partition execute\_i into file execute\_i.ptn

```
savePtnPin -ptn execute_i execute_i.ptn
```

- All partitions and/or blackboxes

Example: Save pin information of all partitions and/or black boxes in the current design

```
savePtnPin -all allPtnPin.ptn
```

- Current block-level design

Example: Save I/O pin locations of the current design

```
savePtnPin -design ioPin.ptn
```

## ***Restore Partition Pin Information***

Use the [loadPtnPin](#) command to restore/load pin placement information of a particular partition/blackbox. The command restores the following:

- A partition file that is generated by the `savePtnPin` or the `saveFPlan` (*floorplan.fp.ptn*) commands

Example: Load pin locations of the partition ALU from partition file ALU.ptn

```
loadPtnPin -ptnName ALU -inFile ALU.ptn
```

- Block-level DEF file

Example: Load pin locations of partition ALU from ALU DEF file

```
loadPtnPin -ptnName ALU -def ALU.def
```

**Note:** You can use the [unloadPtnPin](#) command to remove the preassignment of pins that were previously reassigned by the loading.

## **Assigning I/O Pins**

For a top-down hierarchical flow, I/O pins of a block-level design will normally be assigned during the full-chip pin assignment. This pin placement information is saved in a block-level floorplan partition file (*floorplan.fp.ptn*) or a DEF file that is generated by the `savePartition` command.

For a bottom-up hierarchical flow, I/O pin placement can be generated from an I/O constraint file or

during the cell placement step.

You can also explicitly run I/O pin assignment with the `assignIoPins` command.

This section covers the following topics:

- [Setting Pin Constraints](#)
- [Performing Initial Pin Assignment](#)
- [Refining Pin Placement](#)
- [Validating Pin Placement](#)

## Setting Pin Constraints

The Innovus software provides a number of pin constraint commands to control or guide I/O pin assignment. The same set of pin constraint commands that are used for setting constraints for partition/blackbox pins can also be used for I/O pins. The only difference is that you do not need to specify the `-cell` option for I/O pins. For more information, see [Setting Pin Constraints](#) in the [Assigning Partition and Blackbox Pins](#) section of this document.

## Performing Initial Pin Assignment

For a bottom-up flow, initial pin placement can be generated by any of the following methods:

- Using an I/O constraint file
  - An I/O constraint file can be read into the Innovus environment during the design import step. Or, you can use the `loadIoFile` command to load a constraint file after netlist had been read in. An I/O constraint file can be created by manually editing a text file. For more information about I/O constraint file, see the "Generating the I/O Assignment File" section in the "Data Preparation" chapter of the *Innovus User Guide*.
- Randomly assigning I/O pins
  - You can create an I/O template file with random I/O pin assignment using the following steps. I/O placement is evenly distributed on design boundary:
    - Import design
    - Run the `saveIoFile` command with the `-template` option
    - Use the `loadIoFile` command to load I/O file generated from the step 2
- Placing the design
  - After importing a design and floorplanning it, you should place the design. By default, the

Innovus placer ([placeDesign](#)) internally invokes the I/O pin assignment to place I/O pins based on the current floorplan.

**Note:** Set the `-placeIoPins` option of the `setPlaceMode` command to `False` if you want to disable I/O pin assignment during the placement step.

## Refining Pin Placement

After I/O pins are assigned, you can further refine the current I/O pin assignment by one of the following methods:

- Manually adjust pins by direct pin manipulations or using pin editor.
- Use the [assignIoPins](#) command to further optimize I/O placement.

## ***Using the assignIoPins Command to Optimize I/O Placement***

The `assignIoPins` command assigns I/O pins based on placement information. The design must be placed before this command is run. The command supports:

- Rectilinear designs
- Non-uniform tracks
- User-specified constraints

By default, the `assignIoPins` command will honor fixed pins and only assign pins that have placed/unplaced placement status. If the initial I/O placement is generated by loading a constraint file (that is, the `loadIoFile` command automatically set I/O placement status to fixed) you should change I/O pins placement status to placed using `setPtnPinStatus` command before running I/O pin assignment.

To incrementally assign I/O pins, you can do one of the following:

- Specify pins that should be re-optimized using the `-pin` option.

Example: Re-assign all p\_address bus pins, int, and bio I/O pins of the design tdsp\_core. Optimize these specified pins even though they have fixed placement status.

```
assignIoPins -pin {p_address[*] int bio} -moveFixedPin
```

- Mark I/O pins that you want to keep with fixed status and run the `assignIoPins` command. This scenario can be used when you want to re-optimize most of I/O pins.

Example: Preserve port\_pad\_data\_in and port\_pad\_data\_out buses and clock pins, and re-

optimize the rest.

```
setPtnPinStatus -cell tdsp_core -pin port_pad_data* -status fixed  
setPtnPinStatus -cell tdsp_core -pin clk -status fixed  
assignIoPins
```

## Validating Pin Placement

After assigning I/O pins, it is recommended that you check for I/O legalization. Use the [checkPinAssignment](#) command to make sure that pins are legalized (such as they snap to routing grid, are on reserved routing layers, honor user-specified constraints, not cause any DRC violations, and so on).

You can check:

- All I/O pins

Example: Verify all I/O pins of the current design and output the result into the output file pinLegality.rpt.

```
checkPinAssignment -outFile pinLegality.rpt
```

- Specific I/O pins

Example: Verify all bus pins `BG_scan_in`, `BG_scan_out`, and the `write` pin of the design

```
checkPinAssignment -pin {BG_scan* write}
```

If any pin violation is detected, you can:

- Manually adjust pins by direct pin manipulation or using pin editor.
- Run the [legalizePin](#) command to automatically legalize pins. You can legalize all I/O pins or specific I/O pins of the design. Fixed pins will not be adjusted unless the `-moveFixedPin` option is specified.

Example1: `legalizePin`

With this example, the Innovus software will legalize all pins in the design. If the design is a block-level design that also has partition/blackbox -pins, it will also adjust the partition/blackbox pins. If you want to legalize only the I/O pins but *not* the partition/black box pins, you should use `legalizePin -pin *` instead.

Example2: `legalizePin -pin * -moveFixedPin`

With this example, the Innovus software will legalize all I/O pins. Fixed pins will also be adjusted because the option `-moveFixedPin` has been specified.

Example3: `legalizePin -pin {clock reset rom_data*}`

The Innovus software will legalize clock, reset, and all rom\_data bus bit pins of the design. Pins with fixed status will not be moved.

## Performing Congestion-aware Pin Assignment for Channel-based Designs

To perform route-based pin placement for channel-based designs, it is recommended that you run partition-aware routing instead of a routing that does not take partitions into consideration. Pin assignment decisions based on such partition-aware routing are more optimal with respect to top-channel congestion. However, Trial Route when run in partition-aware mode (`trialRoute -handlePartitionComplex`) is much slower compared to flat (partition-unaware) Trial Route.

To generate a partition-aware routing topology similar to `trialRoute - handlePartitionComplex`, but in much lesser time, you can use the Use `trialRoute -fastRouteForPinAssign` command.

This command generates a routing topology similar to the `handlePartitionComplex` topology for approximately 95% of the inter-partition nets, in about 3X-6X lesser time. For the remaining inter-partition nets, the topology is similar to that generated by flat Trial Route.

The `trialRoute -fastRouteForPinAssign` command should be called before pin assignment. The use flow is:

1. Import the design.
2. Floorplan the design.
3. Run the `trialRoute -fastRouteForPinAssign` command.
4. Assign partition pins.
5. Run `trialRoute -honorPin`
6. Derive time budgeting.

The `trialRoute -fastRouteForPinAssign` command generates a tabular output listing the nets that were routed in partition-aware manner and those that were not. An example of the output is as follows:

Inter Partition Net groups summary:

NetGrp	Normal/PtnAware	NetCount	ptnNames
1	PtnAware	87 (223)	tdsp_core (DTMF_INST/TDSP_CORE_INST)
2	PtnAware	42 (223)	ram_256x16_test (DTMF_INST/RAM_256x16_TEST_INST)
3	PtnAware	32 (223)	G1 (DTMF_INST/G1_PH)
4	PtnAware	20 (223)	results_conv (DTMF_INST/RESULTS_CONV_INST) tdsp_core (DTMF_INST/TDSP_CORE_INST) ram_128x16_test (DTMF_INST/RAM_128x16_TEST_INST)
5	Normal	18 (223)	ram_128x16_test (DTMF_INST/RAM_128x16_TEST_INST)
6	Normal	15 (223)	results_conv (DTMF_INST/RESULTS_CONV_INST)
7	Normal	3 (223)	tdsp_core (DTMF_INST/TDSP_CORE_INST) ram_128x16_test (DTMF_INST/RAM_128x16_TEST_INST)
8	Normal	2 (223)	G1 (DTMF_INST/G1_PH) results_conv (DTMF_INST/RESULTS_CONV_INST) tdsp_core (DTMF_INST/TDSP_CORE_INST)
9	Normal	1 (223)	G1 (DTMF_INST/G1_PH) tdsp_core (DTMF_INST/TDSP_CORE_INST)
10	Normal	1 (223)	G1 (DTMF_INST/G1_PH) results_conv (DTMF_INST/RESULTS_CONV_INST)
11	Normal	1 (223)	G1 (DTMF_INST/G1_PH) tdsp_core (DTMF_INST/TDSP_CORE_INST)
12	Normal	1 (223)	G1 (DTMF_INST/G1_PH) results_conv (DTMF_INST/RESULTS_CONV_INST) ram_256x16_test (DTMF_INST/RAM_256x16_TEST_INST) ram_128x16_test (DTMF_INST/RAM_128x16_TEST_INST)

- NetGrp: Indicates a group of nets. For example, NetGrp 7 indicates the set of nets that logically

connect instances only in partition `tdsp_core` and in partition `ram_128X16_test`.

- `Normal/PtnAware`: Indicates whether the nets of this group are routed in partition-aware routing topology or flat routing topology.
- `NetCount`: Indicates the number of nets in the corresponding net group. For example, `NetGrp 7` contains 3 nets out of a total of 223 inter-partition nets in this design.
- `ptnNames`: indicates the partitions to which the nets in this group of nets connect.

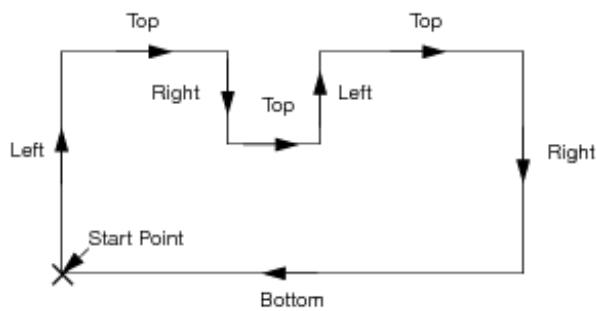
## Salient Points About Congestion-aware Pin Assignment

The following points apply to the behavior and usage of the congestion-aware pin assignment feature:

- The routing topology generated by the `trialRoute -fastRouteForPinAssign` command should be used only for the pin assignment flow.
- The net groups are sorted in descending number of nets in them.
- The net groups that have a significant number of inter-partition nets are routed in a partition-aware manner. The remaining netgroups with fewer inter-partition nets are routed in a manner similar to flat `trialRoute`.
- There is a possibility of more DRC violations in the routing topology generated by the `trialRoute -fastRouteForPinAssign` command as compared to `trialRoute -handlePartitionComplex`. However, for pin assignment purposes, it has little or no impact in deciding the location of partition pins.
- This command is suited *only for channel-based designs*. Also, the improvement in quality of results of pin assignment, with respect to top channel congestion, is more visible in case the design has thick channels.

## Assigning Pins on Rectilinear Edges

Rectilinear pin assignment can recognize rectilinear edges when assigning pins. It can support any rectilinear shape (such as L, T, and U shapes). For rectilinear boundaries created with partition cuts, the edges are identified by starting at the lower-left-most corner, moving clockwise to mark each edge with a direction flow, as shown in the following figure:



All the edges with the same direction flow are considered to be on the same side and have the same user-specified pin constraints.

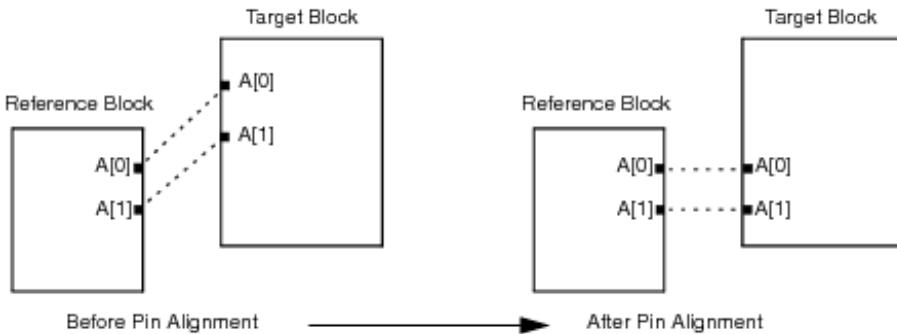
## Swapping Partition Pins

1. Select two pins of the same partition.
2. With the cursor over one of the selected pins, click and hold the middle mouse button to bring up the context pop-up menu.
3. Select Swap Pins (or use the `swapPins` command).

## Pin Alignment

Using `pinAlignment`, the following command aligns `A0` and `A1` pins of `blockB` to the reference pins of `blockA`:

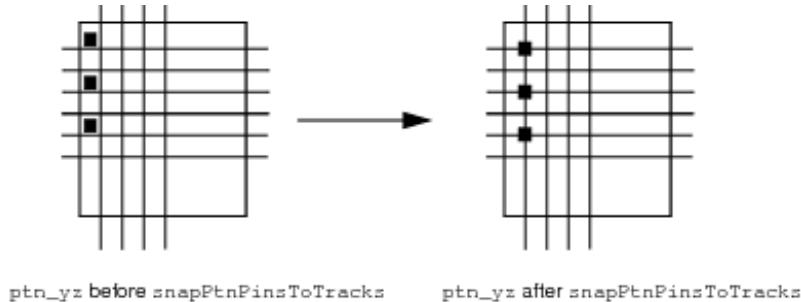
```
pinAlignment -block blockB -refBlock blockA {A0 A1}
```



## Snapping Pins to the Grid

To snap track edge of pins to the nearest intersecting routing grid, where the horizontal and vertical routing tracks cross, use the [snapPtnPinsToTracks](#) text command. For example, the following command snaps center of partition `ptn_xy` pins to the nearest intersecting routing grid:

```
snapPtnPinsToTracks ptn_yz
```



## Assigning Pins for Bus Guides

A bus guide helps ensure that buses are routed together over blocks and is typically used in early floorplanning stages. For more information on the Bus Guide feature, see Chapter 10, Bus Planning. The use model of pin assignment for a bus guide is similar to that of a pin guide. The [assignPtnPin](#) command supports bus guides by treating a bus guide as a pin guide that is associated with a net group. When you assign pins for a design containing a bus guide, all pins of the corresponding net group are placed in the specified bus guide area.

If the specified bus guide area is not large enough to cover all the net group pins, the [assignPtnPin](#) command issues a warning message and places the maximum possible net group pins in bus guide area. The rest of pins are placed outside the pin guide area such that the pins stay together. Bus guide pin assignment also supports all features of net group such as `-optimizeOrder`, `-alternateLayer`, and non-default rules.

The check pin assignment, pin legalization and pin refinement features also support bus guides. **Info:** The bus guide feature is intended to guide partition pins and blackbox pins and *not* I/O pins. The I/O pin assignment feature ([assignIoPins](#) command) does not, therefore, take bus guides into account.

## Pin Assignment Limitations

- Does not support non-R0 orientation black box (non-R0 master black box) pin assignment.  
For more information, see [Handling of Blackboxes with Non-R0 Orientation](#).
- Does not assign or legalize pins on non-preferred routing layers
- Does not assign power/ground pins. For top-down hierarchical flow, power and ground pins will be created during the partition step. For bottom-up flow, power/ground pins should be created at design boundary during power planning stage.
- Partition/blackbox pin assignment may cause routing crossing. In such cases, run the [pinAlignment](#) command to improve pin QoR (Quality of Results).

## Inserting Feedthroughs

There are two types of feedthroughs you can use for partitions: feedthrough buffers and routing feedthroughs. Both types offer different approaches for inserting feedthroughs. Inserting feedthrough buffers allows a netlist change, whereas inserting routing feedthroughs does not.

**Info:** Before inserting feedthroughs, you should determine what stage the design is in, such as prototyping, intermediate, tapeout, and set the appropriate global options by running the `setMode` commands, such as `setPlaceMode` and `setTrialRouteMode`. For example, when inserting feedthroughs during prototyping, you could set modes with the following commands:

```
setPlaceMode -fp true  
setTrialRouteMode -floorplanMode true  
setExtractRCMode -engine preRoute
```

You can use the `insertPtnFeedthrough` command (or the [Inserting Feedthroughs](#) form) to insert feedthrough buffers into the partitions, and the `createPtnFeedthrough` command (or the [Create Physical Feedthrough](#) form) to create a partition routing feedthrough object. The differences between how these two commands affect the design are as follows:

- [insertPtnFeedthrough](#)

The `insertPtnFeedthrough` text command inserts feedthrough buffers into the partitions to change the partition netlists, and avoids routing nets over partition areas. This command affects the design in the following areas:

- Changes both the top-level and partition-level netlists.
- After inserting buffers, it automatically calls `ecoPlace` to place these buffers close to the

partition boundary. However, `insertPtnFeedthrough` does not place the feedthrough pins, which should be assigned during partitioning.

- Inserted buffers will be part of the partition netlists and pushed down to the partition level during Partitioning.
- Wherever a net enters and exits a partition, two ports and a buffer (or two buffers with the `-doubleBuffer` option) are added to the partition netlist.
- For nets that enter or exit a partition several times, such as a "T" shaped connection, three ports will be created. For a cross shaped connection, four ports will be created.
- Use the Design Browser to view the newly added buffer instance and net names for each partition. The new port names have a `FE_FEEDX_..... net_name` prefix.
- For pure channelless designs, use the `-chanLess` option to insert feedthrough buffers for all nets that connect to partitions, except nets that can be connected directly between two adjacent partitions.
- For mixed designs, not all nets should become feedthrough nets. To exclude certain nets, such as clock nets or high fanout nets, use the `-excludeNet` option. This option is based on the topology of the partition neighborhood relationship, so trial routing is not required before inserting feedthrough buffers, although it could help improve the quality of results.
- To specify a file that contains net names for which to insert feedthrough buffers, use the `-selectNet` option. You can create this file manually, create a list of nets via a script, or use `showPtnWireX`.
- Whether you use the `-chanLess` or `-selectNet` options, the Innovus software does not necessarily insert a feedthrough.
- Feedthrough insertion is driven by connectivity when Trial Route is not run before `insertPtnFeedthrough`.
- You can save feedthrough insertion buffer topology tree information in a file by using the `-saveTopoFile` parameter. You can later use this topology tree file with another ECO netlist and replicate the feedthrough insertions. For more information, see "[Replicating Feedthrough Insertions Across ECO Netlists](#)" .
- The `insertPtnFeedthrough` command can detect if the design has power domains. This way, appropriate buffers can be derived automatically from power domain library binding to support both *Always On* and switchable power domains. However, an error message is reported if no regular buffer is found for an *Always On* power domain in the feedthrough path.

- The `insertPtnFeedthrough` command removes nets that are inserted with feedthrough buffers from any net groups to which they belong. After running this command you should, therefore, update the net groups that contain feedthrough nets.
- [createPtnFeedthrough](#)

The `createPtnFeedthrough` command inserts routing feedthroughs into the partitions without changing the design netlist. This command affects the design in the following areas:

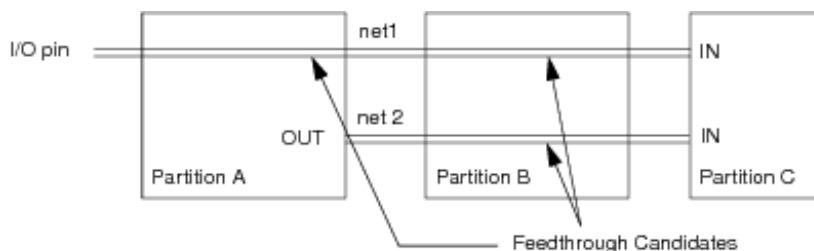
- Manages only the physical aspect of a partition, not the logical aspect.
- No new ports are added to a partition and no buffers are added to the partition netlist.
- For channel feedthroughs, this creates channels for over-the-block routing on specified layers at the top-level design. These channels are pushed down as routing blockages on the correct routing layers at the partition level during Partitioning.
- For placement island feedthroughs, the Innovus software reserves these areas for inserting buffers at the top-level design after running the `insertRepeater` command. These island feedthroughs will be pushed down as placement blockages and routing blockages on all routing layers at the partition level during partitioning.

## Inserting Feedthrough Buffers

Partition feedthrough insertion manages partitioned designs that have nets that need to be pushed down to become a component of each partition design. That is, each feedthrough buffer must be added to the partitioned design, which changes the partition's netlist. This approach is typically used in channelless designs and in designs with limited channel resources.

A pure channelless design has no channel routing resource--connections among partitions are always done by means of module abutment and pin alignment. A mixed or partially channelless design has limited routing resource in the channels; therefore, abutment and pin alignment is only performed on selected nets.

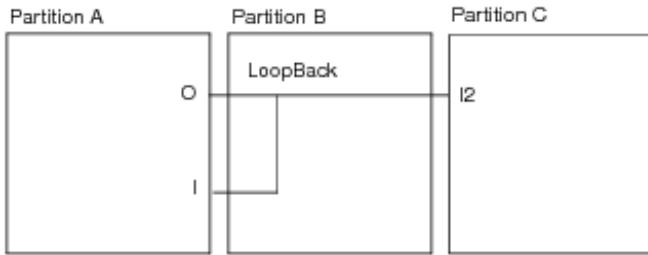
The following example shows how nets are selected for feedthrough buffers:



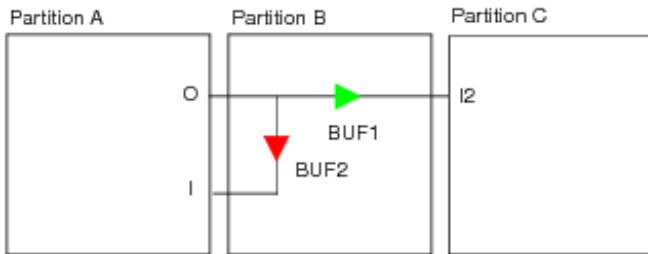
## Inserting Feedback Buffers

You can insert a feedthrough buffer to a net that loops back to an original partition to avoid the net routing over a partition area using the `insertPtnFeedthrough` command.

The following example shows a situation where net `LoopBack` connects to output pin `O` and input pin `I` of Partition A, and input `I2` of Partition C.



By inserting a feedthrough buffer (BUF1) and a feedback buffer (BUF2) with the `insertPtnFeedthrough` command, LoopBack now connects to the input pins of BUF1 and BUF2, as shown in the following figure:



## Limitations

- Each partition must be intact. A non-child instance cannot be preplaced in another partition. This would present a top-level net connection problem.
- Partition pin guides cannot be used during feedthrough insertion.
- The Unpartition program cannot remove the inserted buffers for the feedthrough nets.
- Does not handle blackboxes.
- It might not handle clock nets efficiently because the `insertPtnFeedthrough` text command does not take timing into account.
- It cannot handle nets that are connected to two or more glue logic standard cells. This type of net should be excluded from feedthrough insertion.
- It might not provide good quality of results for high fanout nets. You should exclude high fanout nets and clock nets from feedthrough insertion to avoid timing and routing problems.
- The `insertPtnFeedthrough` command does not support master-clone designs with power domains.

## Procedure

1. Design the top-level floorplan for the partition design.
2. Run Placement.
3. (Optional) Run Trial Route.

**Info:** Up to step 3, the flow is similar to a partition design flow. To control which nets get buffers inserted, complete step 4. If all nets require buffering, skip step 4 and use the `insertPtnFeedthrough -chanLess` command.

4. Create a file to identify which nets get buffers.  
You can manually edit the file, create a script, or generate a wire crossing file (see [Generating the Wire Crossing Report](#)).
5. Generate the feedthrough buffers and nets.  
Use the `insetPtnFeedthrough -chanLess` command, or `insetPtnFeedthrough -selectNet` with the created net file.  
**Note:** Step 6 returns to the normal partition design flow.
6. Run Trial Route to completely connect the design, including the inserted feedthrough buffers.
7. Run Partition to generate the partition pins and change the partition module status to hard block.
8. Run Save Partition.

This saves the design and generates a top-level directory and partition directories.

## Using a Topology File to Insert Feedthrough Buffers

You can guide the insertion of feedthrough buffers for specific nets by providing the feedthrough topology information for those nets in a topology file. You can manually create this file and subsequently edit it.

**Note:** If you are using topology files from releases prior to the 8.1 release, they will still work with this release.

**Note:** The syntax is case sensitive.

There are two versions of the topology file - Version 1 (old) and Version 2 (new).

In Version 2 of the topology file:

- You can specify exactly how the reuse of ports takes place in master clone scenarios.
- You have more control over the feedthrough topology.

The main difference from version 1.0 is that the topology needs to be specified in terms of the port names of the hierarchical modules rather than the names of the hierarchical modules.

The version 2.0 of the topology file will \*not\* obsolete the version 1.0.

## Syntax and Examples for Version 1.0 of the Topology File

The syntax of the topology information in the file is as follows.

```
# Comment line

version version_string;
nametype netname
    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);
    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);
    ...
end nametype
nametype netname
    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);
    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);
    ...
end nametype
...
```

The description of the syntax is as follows

nametype	<p>Can be <code>net</code>, <code>bus</code>, or <code>netgroup</code>. The value <code>netgroup</code> represents all nets in the net group. You should update the net group after feedthrough insertion step.</p> <p>Here are some examples of nametype:</p> <ul style="list-style-type: none"><li>• <code>bus myBus [0:1]</code> specifies bus bits</li><li>• <code>net myBus [0:1]</code> specifies a scalar net.</li><li>• <code>bus myBus [1]</code> specifies a bus bit</li><li>• <code>net mybus [1]</code> specifies a scalar net or a bus bit. In case both exist in the design, use the Verilog escape name and use the <code>dbgIsBackSlashInNamesHiddenFlag</code> variable to resolve correctly.</li></ul>
----------	--

<p><b>netname</b></p> <p>Can be a net name, bus name, or a net group name. Wildcards (*) or (?) can be used for net name, bus name, or net group name.</p> <p>If more than one net group is matched with wildcard, the <a href="#">insertPtnFeedthrough</a> command will issue a warning message and:</p> <ul style="list-style-type: none"> <li>● use only the first matched net group</li> <li>● ignore the other ones.</li> </ul> <p>Wild cards can only be used for a bus name BUT not bus range. Example you cannot specify bus busname[1:*].</p> <p><i>Specifying bus entries :</i> If a bus named databus has 32 bits (from 0 to 31), its r bus entries are specified as follows:</p> <ul style="list-style-type: none"> <li>● bus databus specifies all 32 bits from 0 to 31</li> <li>● bus databus [13:23] specifies databus[13] to databus[23]</li> <li>● bus databus [13] specifies only the bit 13 of databus</li> </ul> <p>You cannot provide any net-specific entries for multiple bus bits, net groups, or wildcard nets. Hence, bus topologies cannot be specified for bus nets connected to top-level instance pins or to I/O pins.</p> <p><i>Using escape mechanism for special characters:</i> The following escape mechanisms remove all restrictions on characters:</p> <ul style="list-style-type: none"> <li>● \\ for the backslash character (\) itself</li> <li>● \b for blank</li> <li>● \t for tab</li> <li>● \n for new line</li> <li>● \0 for null</li> <li>● \s for semicolon (semicolon (;) is the path statement terminator).</li> </ul> <p>Any other character which follows a backslash (\) is taken literally. For example, \a is considered as a. If one wants to use *,? literally then must use escaping as these are used for wildcards.</p>	<p><b>Note:</b> If a net appears twice in any form, the first entry corresponding to the net is used. The subsequent entries generate an error.</p>
---	---

fromtype	<p>Can have one of the following values:</p> <ul style="list-style-type: none"> <li>● <code>io</code> for I/O pins</li> <li>● <code>hinst</code> for hierarchical instance name of a partition or partition clone</li> <li>● <code>instterm</code> for top-level instance pins</li> </ul>
totype	<p>Can have one of the following values:</p> <ul style="list-style-type: none"> <li>● <code>io</code> for I/O pins</li> <li>● <code>hinst</code> for hierarchical instance name of a partition or partition clone</li> <li>● <code>instterm</code> for top-level instance pins</li> <li>● <code>hinstfb</code> for hierarchical instance name of a partition or partition clone. This can only be used as part of the combination <code>hinst-hinstfb</code>, which specifies a feedback buffer path.</li> </ul>
version	<p>Version is the format version. The format version for this release is 1.0.</p> <p>If the topology file does not have the version statement then the <code>insertPtnFeedthrough</code> command will parse the file as per the format of the version prior to the 8.1 release.</p>
route_data	<p>Optional field that specifies routing information.</p> <p>This is not a user-specified field. This field is created when the <code>insertPtnFeedthrough</code> command is run with the <code>-saveTopoFile</code> parameter. This field is used only for ECO purposes.</p> <p>The <code>route_data</code> parameter is not available if the <code>totype</code> is <code>hinstfb</code>.</p>

All version- and topology-statements in the topology file end with a semicolon (;). Any extra spaces are ignored.

Here is an example of a topology file:

```
#####
version 1.0;
net net1
io-hinst net1 i_b;
hinst-instterm i_b inst_c/net1;
```

```

end net

net clk*
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end net

netgroup group_a
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end netgroup

bus databus[0:31]
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end bus
#####

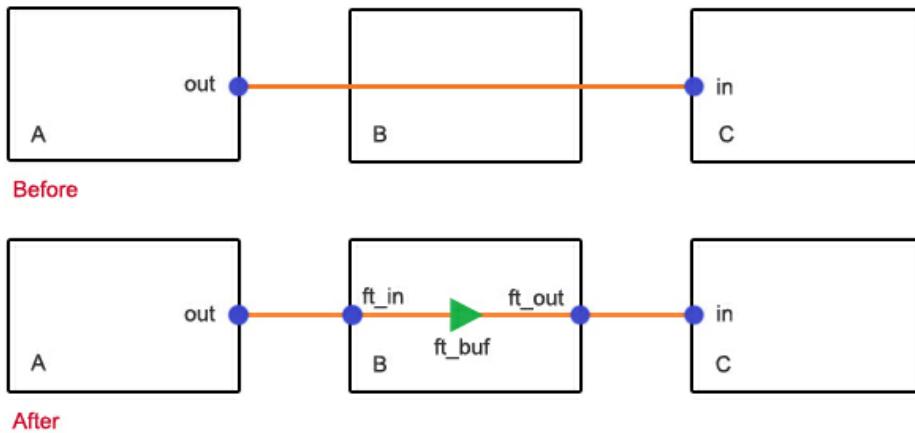
```

## Examples of usage of topology file version 1.0

The following examples illustrate topology file usage:

- **Example 1: Two pin net**

The following image shows a two pin net before and after feedthrough insertion. It has an output port on partition A and an input port on partition C.



In order to insert a buffer in partition B, the following topology syntax is used:

```

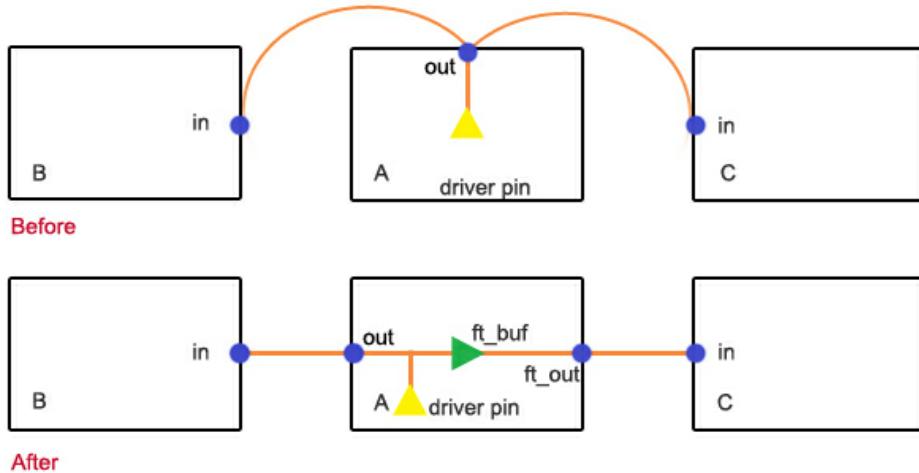
version 1.0;
net netname
hinst-hinst A B;
hinst-hinst B C;

```

end net

- **Example 2: Multifanout net with two sink partitions**

The following is a before and after feedthrough insertion image of multifanout net with two sink partitions. It has output port on partition A and input ports on partitions B and C.

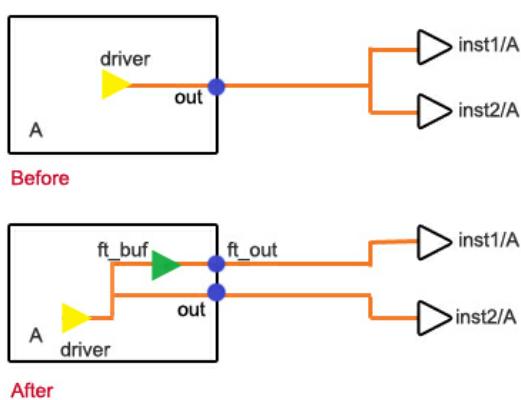


The following topology file is used

```
version 1.0;
net netname
hinst-hinst A B;
hinst-hinst A C;
end net
```

- **Example 3: Multifanout net with 2 sinks which are standard cells**

The following is a before and after feedthrough insertion image of a multifanout net with 2 sinks which are standard cells.



The following topology file is used:

```
version 1.0;

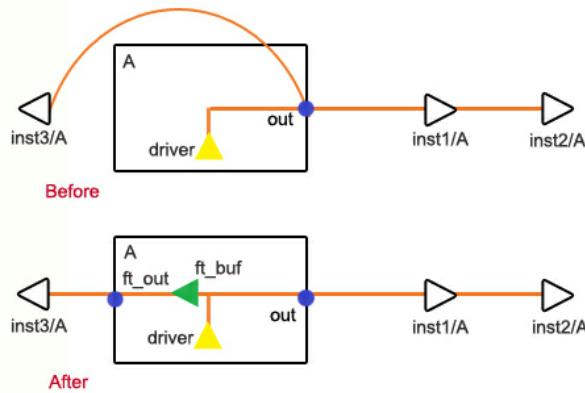
net netname

hinst-instterm A inst1/A;
hinst-instterm A inst2/A;

end net
```

- **Example 4 : Multifanout net with 3 sinks which are standard cells**

The following is a before and after feedthrough insertion image of a multifanout net with 3 sinks which are standard cells. It is not desired to insert separate buffers for inst1 and inst2.



The following topology file is used:

```
version 1.0;

net netname

hinst-instterm A inst1/A;
hinst-instterm A inst3/A;
instterm-instterm inst1/A inst2/A;

end net
```

## Syntax and Examples for Version 2.0 of the Topology File

The syntax of the topology information in the file is as follows.

```
# Comment line

version version_string;
nametype netname
fromtype-totype from_name to_name;
```

```

fromtype-totype from_name to_name;
...
end nametype
nametype netname
    fromtype-totype from_name to_name;
    fromtype-totype from_name to_name;
    ...
end nametype
...

```

The description of the syntax is as follows

nametype	Can only be <code>net</code> . It cannot be <code>bus</code> or <code>netgroup</code> .
netname	The name of the net. Wildcards (*) or (?) cannot be used for net names.
fromtype	<p>Can have one of the following values:</p> <ul style="list-style-type: none"> <li>● <code>hterm</code>: The <code>from_name</code> is to be specified as <code>hinstname/portname</code>. If the port names specified don't exist already then they will be created.</li> <li>● <code>instterm</code>: <ul style="list-style-type: none"> <li>○ For top level pins the <code>from_name</code> is specified as <code>instname/termname</code>.</li> <li>○ For multi-fanout nets originating from a hierarchical module which is a partition or a clone the topology must originate from the driver term. Instead of specifying the complete driver name such as <code>instname/termname</code> one can use the shorthand <code>driver_term</code>. Refer to examples 2,4, and 6 given below.</li> </ul> </li> <li>● <code>io</code>: For chip level I/O pins.</li> </ul> <p><b>Note:</b> The <code>hinst</code> keyword may not be used as a <code>fromtype</code> in version 2.0.</p>

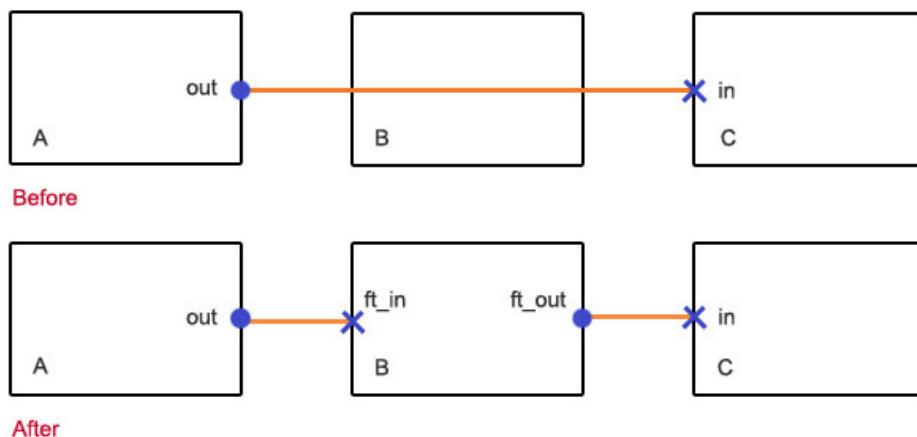
totype	<p>Can have one of the following values:</p> <ul style="list-style-type: none"> <li>● <code>hterm</code>: The <code>to_name</code> is to be specified as <code>hinstname/portname</code>. If the port names specified don't exist already then they will be created.</li> <li>● <code>instterm</code>: For top level instance pins. The <code>to_name</code> is specified as <code>instname/termname</code>.</li> <li>● <code>io</code>: For chip level I/O pins.</li> </ul> <p><b>Note :</b> The <code>hinst</code> keyword may not be used as a <code>totype</code> in version 2.0.</p>
version	<p>The version is 2.0. The topology file is saved in this format when the <code>-repeatedSymmetricAbuttedFPlan</code> option is used with the <code>-saveTopoFile filename</code> option of the <code>insertPtnFeedthrough</code> command.</p>

## Examples of usage of topology file version 2.0

The following examples illustrate topology file usage:

- **Example 1: Simple Case**

The following image shows a two pin net before and after feedthrough insertion. It has an output port on partition A and an input port on partition C.



The following topology syntax is used:

```
version 2.0;
net net
hterm-hterm A/out B/ft_in;
```

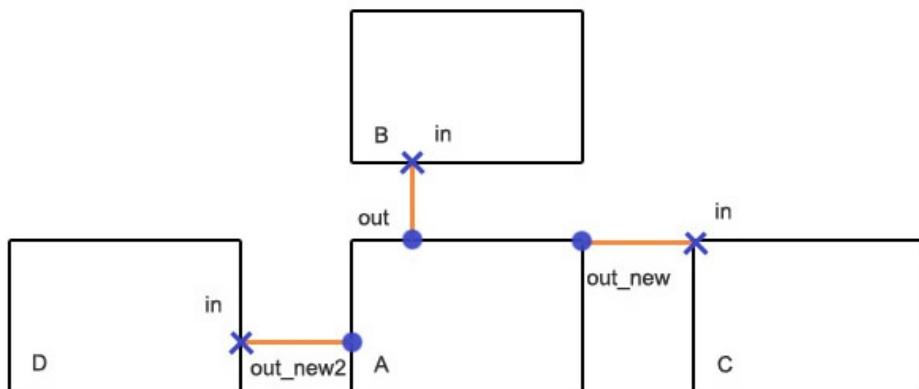
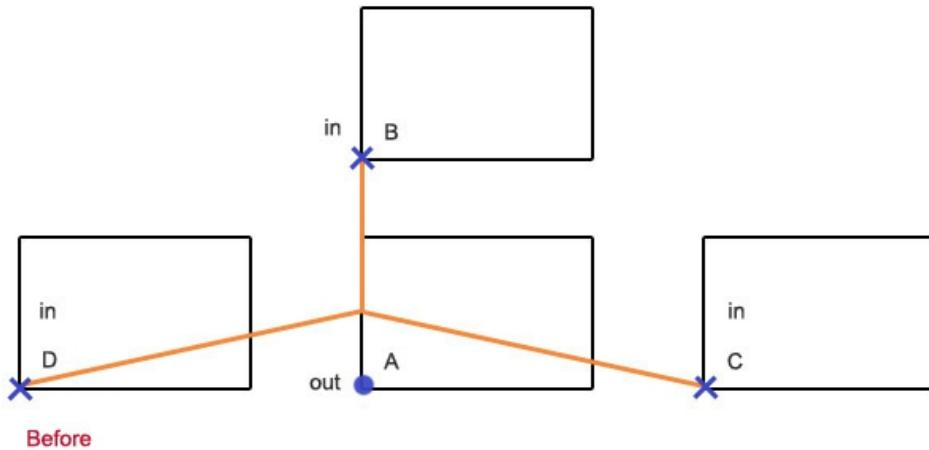
```

hterm-hterm B/ft_in B/ft_out;
hterm-hterm B/ft_out C/in;
end net

```

- **Example 2: Multifanout Case**

The following is a before and after feedthrough insertion image of multifanout net with three sink partitions. It has output port on partition A and input ports on partitions B, C, and D.



The following topology file is used

```

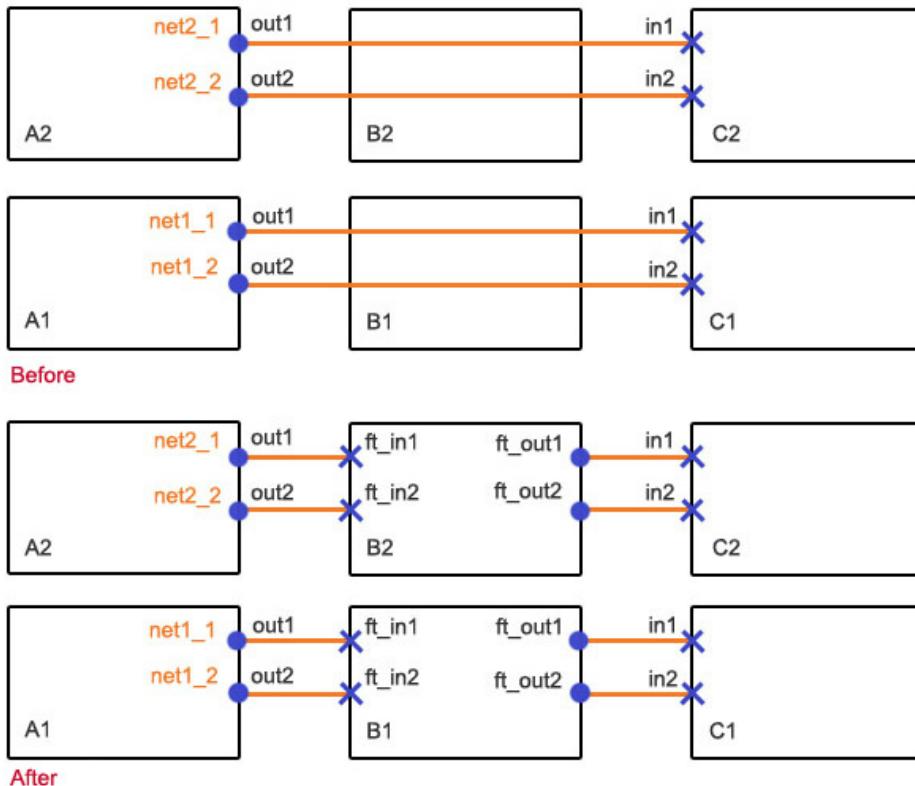
version 2.0;
net net
instterm-hterm driver_term A/out;
instterm-hterm driver_term A/out_new;
instterm-hterm driver_term A/out_new2;
hterm-hterm A/out B/in;
hterm-hterm A/out_new C/in;
hterm-hterm A/out_new2 D/in;

```

```
end net
```

- **Example 3: Reuse for simple case of Example 1**

The following is a before and after feedthrough insertion image of four nets with two pins.



The following topology file is used:

```
version 2.0;
net net1_1
hterm-hterm A1/out1 B1/ft_in1;
hterm-hterm B1/ft_in1 B1/ft_out1;
hterm-hterm B1/ft_out1 C1/in1;
end net

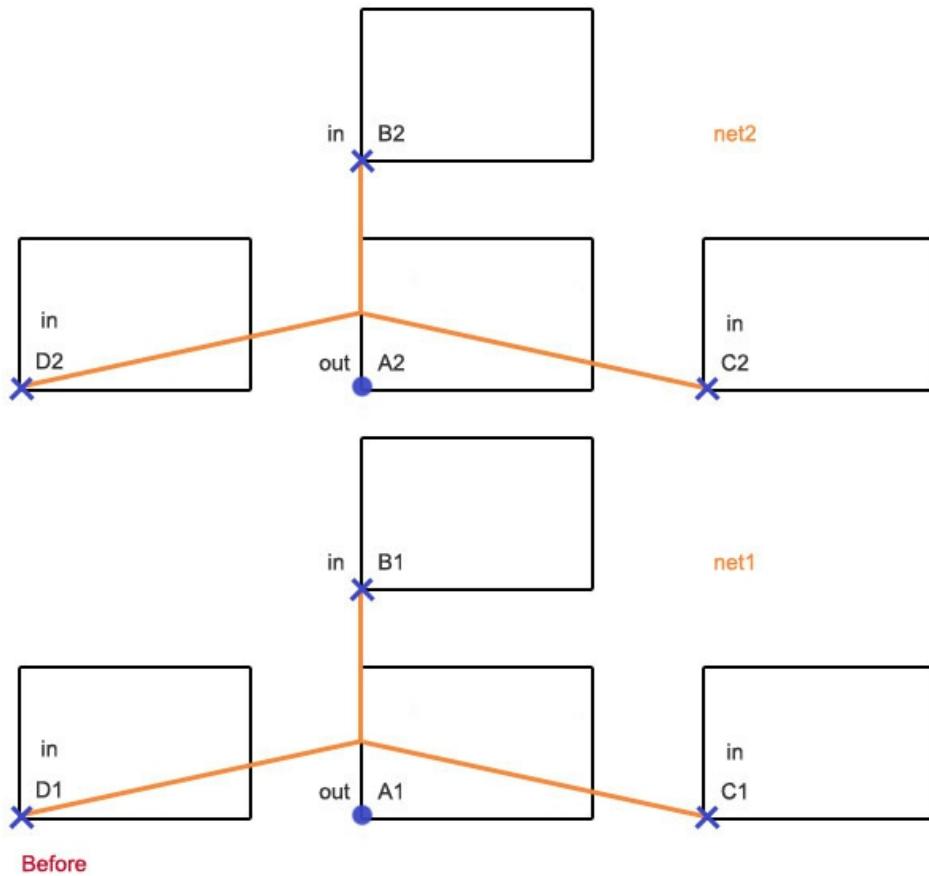
net net1_2
hterm-hterm A1/out2 B1/ft_in2;
hterm-hterm B1/ft_in2 B1/ft_out2;
hterm-hterm B1/ft_out2 C1/in2;
end net

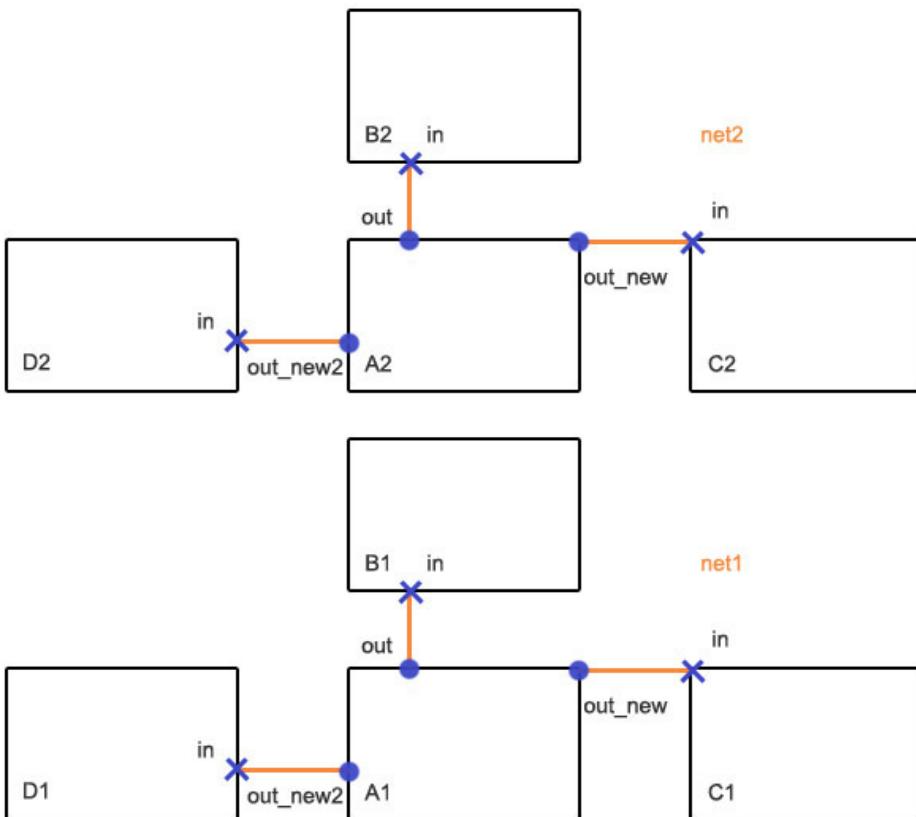
net net2_1
hterm-hterm A2/out1 B2/ft_in1;
hterm-hterm B2/ft_in1 B2/ft_out1;
```

```
hterm-hterm B2/ft_out1 C2/in1;  
end net  
  
net net2_2  
hterm-hterm A2/out2 B2/ft_in2;  
hterm-hterm B2/ft_in2 B2/ft_out2;  
hterm-hterm B2/ft_out2 C2/in2;  
end net
```

- **Example 4 : Reuse case for multi-fanout case of Example 2**

The following is a before and after feedthrough insertion image of two multifanout nets with three sink partitions each.





**After**

The following topology file is used:

```

version 2.0;
net net1
instterm-hterm driver_term A1/out;
instterm-hterm driver_term A1/out_new;
instterm-hterm driver_term A1/out_new2;
hterm-hterm A1/out B1/in;
hterm-hterm A1/out_new C1/in;
hterm-hterm A1/out_new2 D1/in;
end net

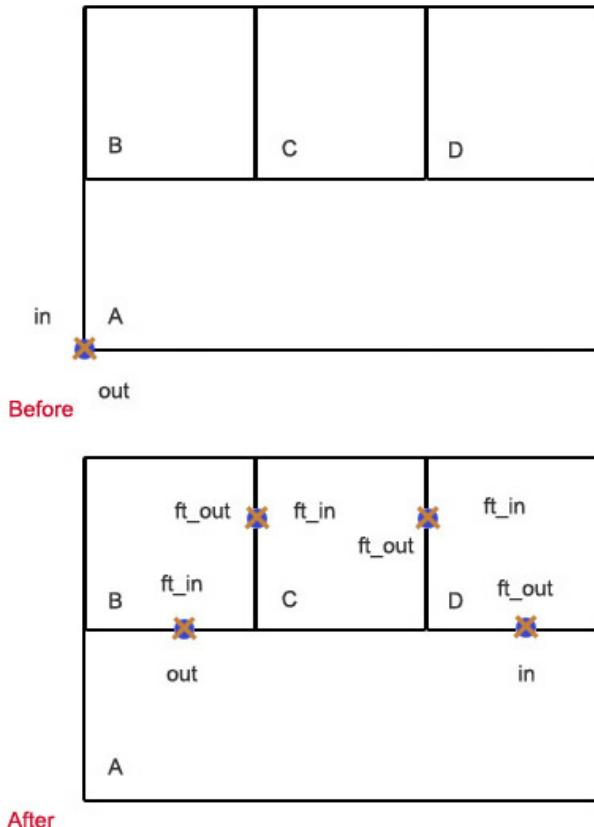
net net2
instterm-hterm driver_term A2/out;
instterm-hterm driver_term A2/out_new;
instterm-hterm driver_term A2/out_new2;
hterm-hterm A2/out B2/in;
hterm-hterm A2/out_new C2/in;
hterm-hterm A2/out_new2 D2/in;

```

```
end net
```

- **Example 5: Complex case for loopback net**

The following is a before and after feedthrough insertion image of a loopback net.

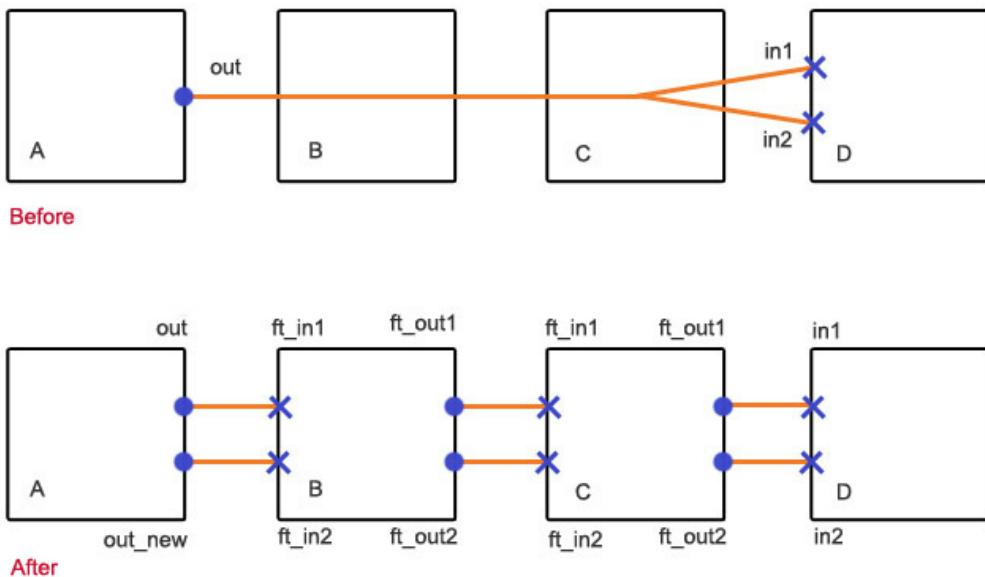


The following topology file is used:

```
version 2.0;
net loopback
hterm-hterm A/out B/ft_in;
hterm-hterm B/ft_in B/ft_out;
hterm-hterm B/ft_out C/ft_in;
hterm-hterm C/ft_in C/ft_out;
hterm-hterm C/ft_out D/ft_in;
hterm-hterm D/ft_in D/ft_out;
hterm-hterm D/ft_out A/in;
end net
```

- **Example 6: Complex case for multi-fanout net**

The following is a before and after feedthrough insertion image of a complex multifanout net.



The following topology file is used:

```
version 2.0;
net mfanout
instterm-hterm driver_term A/out;
instterm-hterm driver_term A/out_new;
hterm-hterm A/out B/ft_in1;
hterm-hterm B/ft_in1 B/ft_out1;
hterm-hterm B/ft_out1 C/ft_in1;
hterm-hterm C/ft_in1 C/ft_out1;
hterm-hterm C/ft_out1 D/in1;
hterm-hterm A/out_new B/ft_in2;
hterm-hterm B/ft_in2 B/ft_out2;
hterm-hterm B/ft_out2 C/ft_in2;
hterm-hterm C/ft_in2 C/ft_out2;
hterm-hterm C/ft_out2 D/in2;
end net
```

## Replicating Feedthrough Insertions Across ECO Netlists

While performing a feedthrough insertion through the `insertPtnFeedthrough` command, you can save the feedthrough buffer topology tree information in a file by specifying the `-saveTopoFile` parameter as follows:

```
insertPtnFeedthrough -saveTopoFile TopoFileName
```

where,

*TopoFileName* is the name of the file in which topology information is saved.

When you run the `insertPtnFeedthrough` command on another ECO netlist, you can use the saved file to replicate feedthrough buffer insertions by specifying the `-topoFile` parameter as follows:

```
insertPtnFeedthrough -topoFile SavedTopoFileName
```

where,

*SavedTopoFileName* is the name of the file that was saved earlier using the `-saveTopoFile` parameter.

This way, you can save a file with feedthrough buffer topology tree information and use it to create the same feedthrough buffer insertions across multiple netlists.

The flow can be summarized as follows:

1. Import a design.
2. Perform floorplanning on the design.
3. Perform feedthrough buffer insertion and save the feedthrough buffer topology tree information in a file (use the `-saveTopoFile` parameter of the `insertPtnFeedthrough` command).
4. Import design with a new ECO netlist.  
**Note:** The ECO netlist should not contain the original inserted feedthrough buffers.
5. Perform feedthrough buffer insertion with the topology file saved from step 3 (use the `-saveTopoFile` parameter of the `insertPtnFeedthrough` command).

**Note:** If you use the `-topoFile` parameter, only those nets that are specified in the topology file are considered for feedthrough buffer insertion.

**Note:** If a net does not exist in the design, it should not be in the topology file. For example, if ECO changes remove a net, that net should be removed from the topology file.

6. Repeat steps 4 and 5 for more ECO netlists, if required.

## Reducing the Number of Buffers and Ports Added for Route-based Feedthrough Insertions

You can use the `-reduceAddedPort` parameter of the `insertPtnFeedthrough` command to specify that feedthrough insertion should follow the routing topology more closely. This can help reduce the number of added ports and buffers.

The ports are created at the route crossing points. The status of the added ports is set to *Fixed*.

Subsequent use of Trial Route will make the routes pass through these pins. Therefore, there is no need to create partition pin guides for these pins.

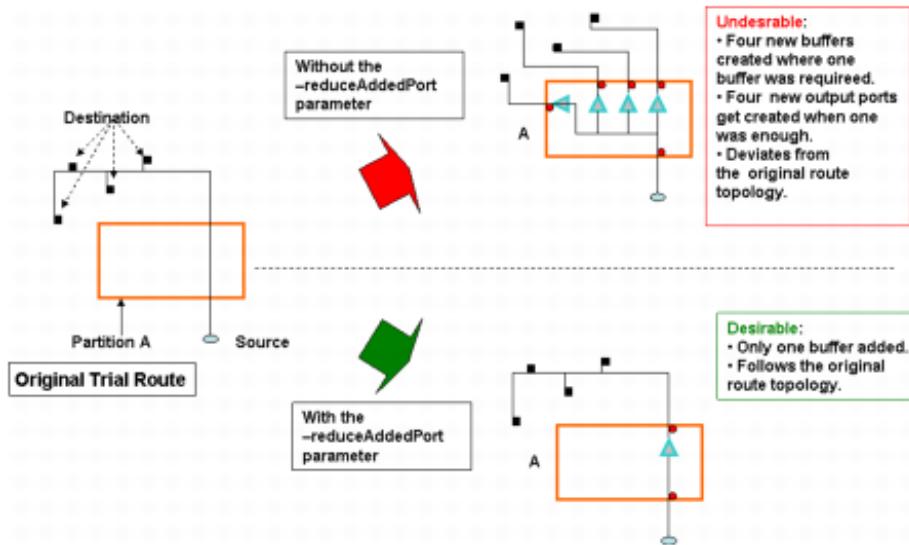
**Note:** The `-reduceAddedPort` parameter is applicable only for route-based feedthrough insertions.

This behavior is illustrated through the following scenarios:

- Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels:
- Net Connecting Through Adjoining Partition

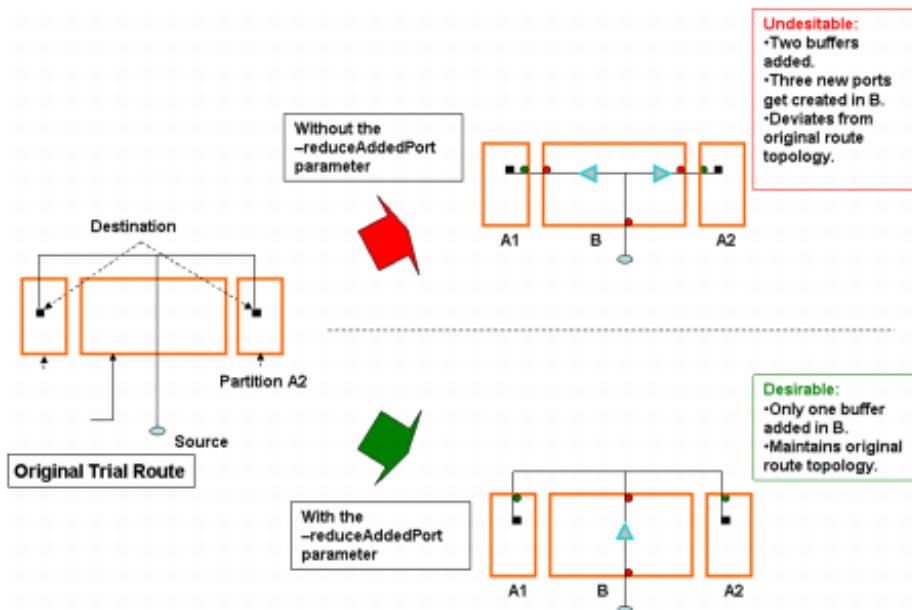
## ***Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels***

The following diagram illustrates the improvement in feedthrough insertion where a net connects to a non-partition instance terminals in the top-level routing channels.



## ***Net Connecting Through Adjoining Partition***

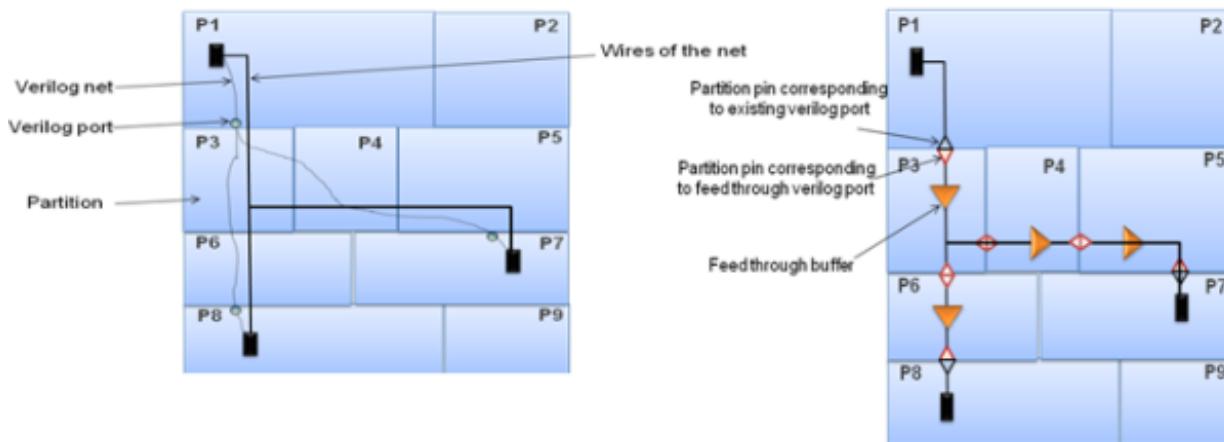
The following diagram illustrates the improvement in feedthrough insertion between partitions where there is another partition in between.



## Mentioning Some Verilog Modules as dont-add-ports

From the 14.2 release, during `insertPtnFeedthrough`, if a new verilog port gets created in a module, then the tool does not create new ports in the module if it is listed in the `dont_add_ports_to_module` option.

For example, without the `dont_add_ports_to_module` option, the feedthrough buffers and new verilog ports are added as shown in the figures below:

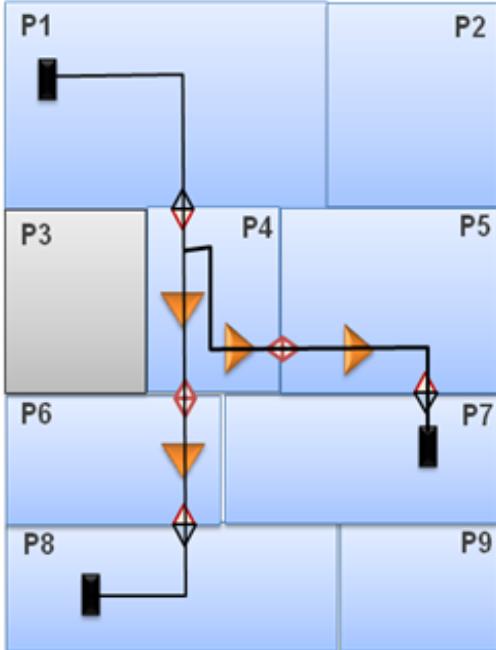


However, if some modules are specified in `dont_add_ports_to_module`, then new ports are not created in the specified modules.

For the design scenario shown above, here are some examples of the usage of

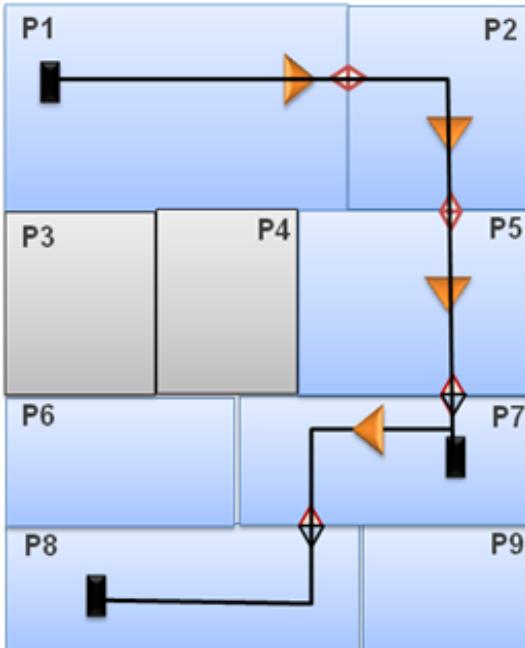
`dont_add_ports_to_module.`

- `insertPtnFeedThrough -dont_add_ports_to_module {P3_hi}`

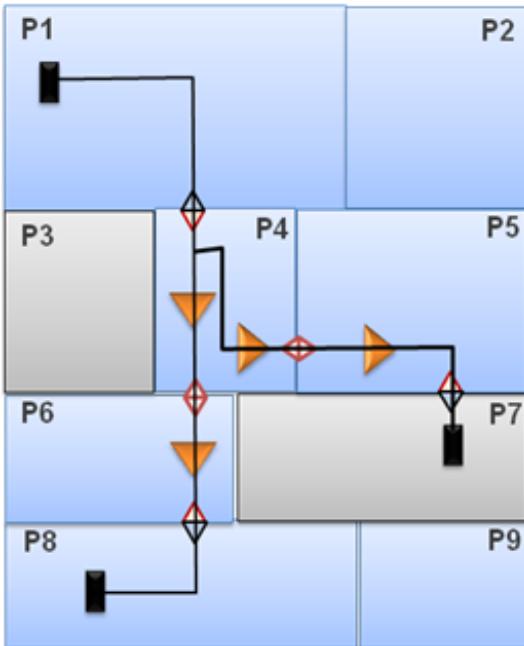


- `insertPtnFeedThrough -dont_add_ports_to_module {P3_hi P4_hi}`

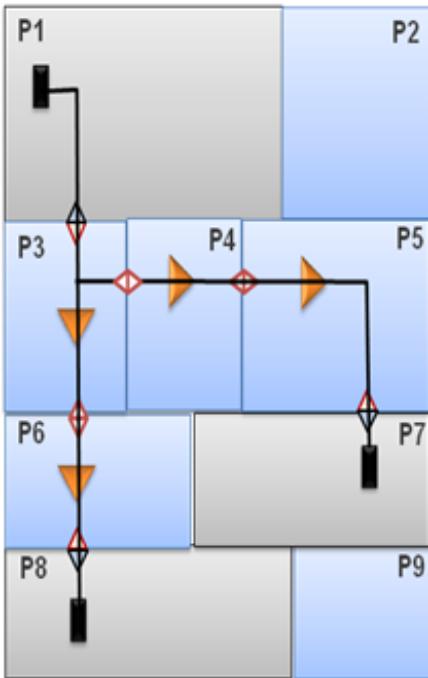
An alternate path is created leaving the partitions mentioned in `dont_add_ports_to_module`.



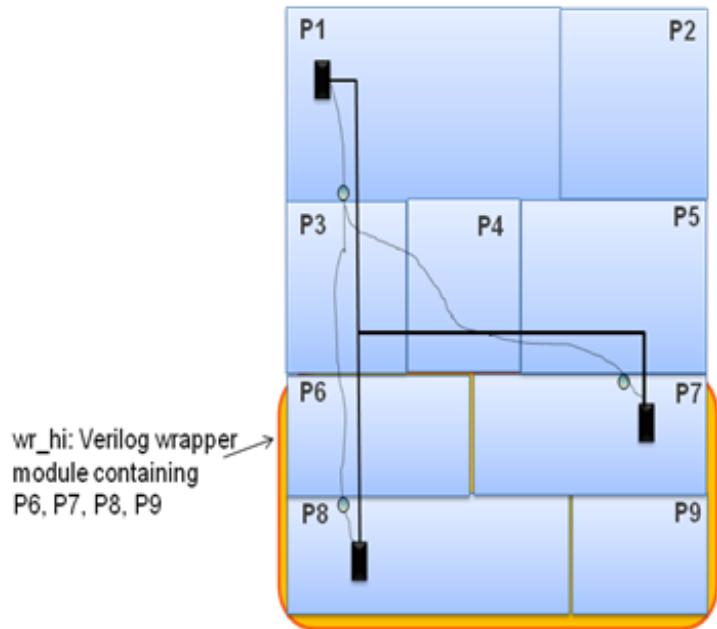
- `insertPtnFeedThrough -dont_add_ports_to_module {P3_hi P7_hi}`



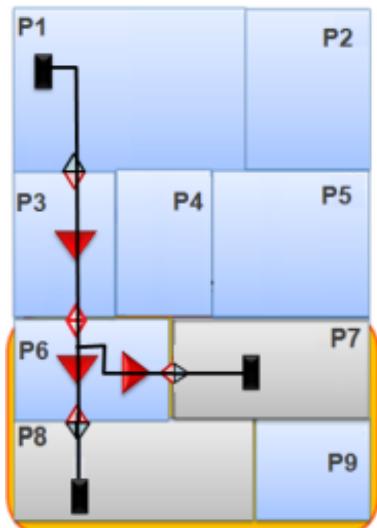
- `insertPtnFeedThrough -dont_add_ports_to_module {P1_hi P7_hi P8_hi }`



Consider another design scenario now, as shown below. Note that `wr_hi` is the verilog wrapper module containing P6 (`wr_hi/P6_hi`), P7 (`wr_hi/P7_hi`), P8 (`wr_hi/P8_hi`), P9 (`wr_hi/P9_hi`).



- `insertPtnFeedThrough -dont_add_module_port {wr_hi wr_hi/P7_hi wr_hi/P8_hi}`



## Abbreviating Lengthy Feedthrough Net Names

You can abbreviate inserted feedthrough net names so that the net names will not extend too long if you run the `insertPtnFeedthrough` multiple times. With the `-useShortName` option, you can eliminate the use of the old net name and partition names, and instead use a running count for the new net names.

For example, if a feedthrough net reset connects two partitions `tt_chiplet` and `video_chiplet`, the feedthrough net name is:

`FE_FEEDX_NET_C_tt_chiplet_video_chiplet_reset`

The net name abbreviation convention for feedthrough buffer insertion when using the `insertPtnFeedthrough - useShortName` command are:

Net Names	<code>FE_FTN_ n</code> , where <i>n</i> is an integer
Buffer Names	<code>FE_FTB_ n</code> , where <i>n</i> is an integer

The net name abbreviation convention for feedback buffer insertion when using the `insertPtnFeedthrough - useShortName` command are:

Net Names	<code>FE_FB_NET_ n</code> , where <i>n</i> is an integer
Buffer Names	<code>FE_FB_BUF_ n</code> , where <i>n</i> is an integer

## Highlighting the Nets for which Feedthrough Buffers Have been Inserted

Once you insert partition feedthrough buffers with the `insertPtnFeedthrough` command, you can highlight these nets with the `hiliteFeedthroughNets` command. The highlighted feedthrough path consists of the nets, the terms that the nets connect to, and the instances that contain those terms. For the `hiliteFeedthroughNets` command to work, the `insertPtnFeedthrough` command must be run with the `-netMapping` parameter. The net mapping file generated with the `insertPtnFeedthrough -netMapping` parameter is used by the `hiliteFeedthroughNets` command to highlight the feedthrough nets.

To dehighlight the feedthrough nets, run the `dehighlight` command.

## Utilizing Pre-defined Feedthrough Pins in Custom Macros

Some designs contain hard macros, which could, for example, be IP blocks or analog blocks. chip-level routing might not be possible without passing over these blocks. Or, in other cases, routing might not meet timing requirements if it detours around these blocks. To facilitate routing these blocks might provide pre-defined feedthrough pins. You can utilize these predefined feedthroughs using the `connectMacroFeedthrough` command. This command automatically connects the feedthrough pins to nets that have wires crossing over these blocks or macros.

## Use Flow

The `connectMacroFeedthrough` command uses the routing topology to connect the pre-defined feedthrough nets. Therefore, the design must be placed and routed before you run the `connectMacroFeedthrough` command. The use flow is as follows:

1. Import the design.
2. Floorplan the design.
3. Perform placement.
4. Run Trial Route.

**Info:** At least one vertical and one horizontal routing layer must be available (that is, not blocked) on the macro(s). Otherwise, there will be no routing over the macro(s). In case the macro has all the layers blocked, manually remove the blockage over one horizontal and vertical layer.

5. Connect the built-in feedthroughs through the [connectMacroFeedthrough](#) command.

**Note:** Before running detailed routing, take care of the unused feedthrough input pins that are left floating. For example, you might want to assign them to tie-high or tie-low. You can save the list of the unused ports with the `connectMacroFeedthrough -floatingPortList` command.

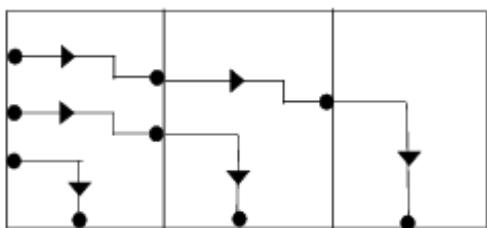
## How the `connectMacroFeedthrough` Command Connects Feedthroughs

The following points illustrate the criteria for feedthrough selection and other important features of the [connectMacroFeedthrough](#) command:

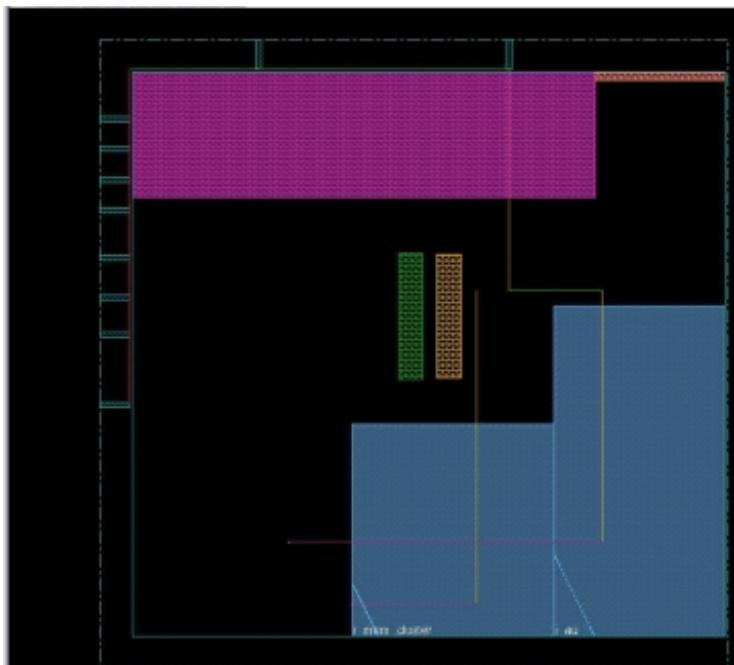
- The [connectMacroFeedthrough](#) command considers all routing on all layers that cross the specified custom macro boundaries.
- The command searches for a feedthrough whose in and out pins lie *on the same sides* of the macro on which the wires enter and exit the macro.
- A feedthrough that has pins that are closer to the intersections has a higher probability of selection. Both input and output pins are considered. Layer information is ignored while evaluating the distance. To consider only pins within a specific distance from the wire crossing, use the `-maxSearchDistance` parameter.
- The command creates new nets and ports as required.
- If multiple feedthrough insertions are performed, the command keeps track of the feedthroughs already used, and does not assign such feedthroughs again.
- The new nets (the nets that connect to feedthrough output pins) have the following naming convention:
  - `FE_FTM_x_ netName`  
where `x` is a unique numeric identifier and `netName` is the name of the original net.
- You can select only specific nets for or exclude specific instances or nets. You can also specify the distance till which the command will search for a connected feedthrough. The feedthrough connectivity is described through a mapping file, which is described in the section [Mapping File For Describing Feedthrough Connectivity](#).

## ***Feedthrough Connection for Abutted Macros***

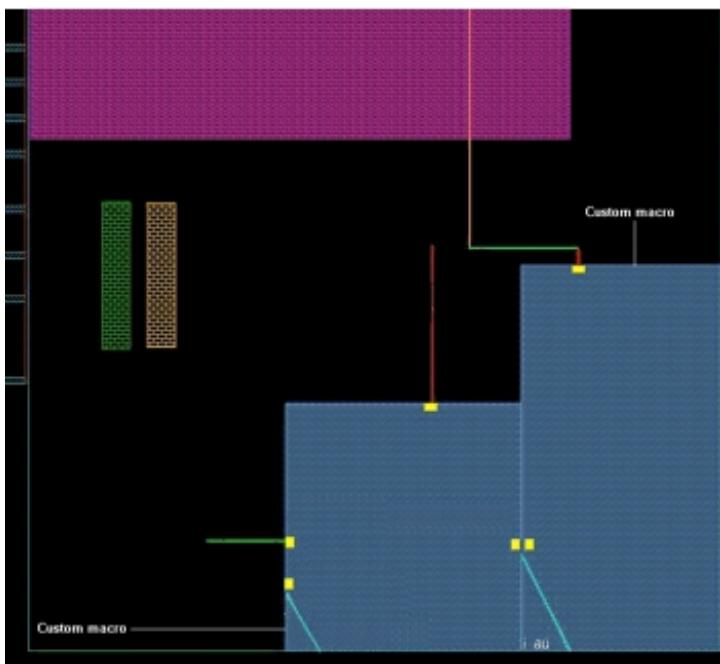
For abutted custom macros, the `connectMacroFeedthrough` command detects the paths formed by the abutted feedthrough pins. The Innovus software considers only the end points of the detected paths, and picks those feedthroughs that will give good results. The following figures show how Innovus selects the feedthroughs for insertion in the abutted custom macros.



The following figure shows pre-defined custom feedthroughs in the design.



The following figure shows how these feedthroughs are utilized by the `connectMacroFeedthrough` command. Notice the feedthrough pins, represented by yellow squares, that are added at the intersection of the macro boundary and the pre-defined nets.



## Mapping File For Describing Feedthrough Connectivity

The feedthrough connectivity is defined through a mapping file that is provided as a parameter to the. If a mapping file is not specified with the `connectMacroFeedthrough` command, Innovus assumes that a file with the name `portmap` in the current directory is used by default.

The syntax of the file is as follows:

```
MACRO MacroName
```

```
    Macro definition section
```

```
END MACRO
```

The definition of the macro is provided in the *Macro definition* section, which can contain one or more feedthrough sections. The name of the feedthrough section is optional.

**Note:** The definitions for all custom macros to be used in the design should be in a single portmap file.

The syntax of the Feedthrough section is as follows. The name of the feedthrough is optional.

```
Feedthrough [ FeedthroughName ]
```

```
Pin Section
```

```
END FEEDTHROUGH
```

Each Feedthrough section contains one section for the input pin and one section for the output pin.

**Note:** Multi-fanout feedthrough sections are not supported.

The syntax of the pin section is as follows:

PIN *PinName*

END PIN

**Note:** All the predefined macro feedthrough pins should be floating pins.

Here is an example of a mapping file:

```
MACRO RAMXXX
FEEDTHROUGH feedthrough1
PIN feedthrough1_in
END PIN
PIN feedthrough1_out
END PIN
END FEEDTHROUGH
```

```
FEEDTHROUGH feedthrough2
```

```
PIN feedthrough2_in
END PIN
PIN feedthrough2_out;
END PIN
END FEEDTHROUGH
END MACRO
```

## Limitations

The [connectMacroFeedthrough](#) command has the following limitations:

- Multi-fanout feedthroughs are not supported.
- Routing blockage and congestion are not considered. However, because topology is derived from routing, this should not be a concern.
- Bidirectional pins (INOUT) are not supported.
- The topology is derived from the routing results. Therefore, you might need to specify certain Trial Route options (for example, options to block or unblock certain routing tracks) to get the desired routing results.
- Floating module ports connected to a net are not supported because there is no routing to the floating module ports.
- Rectilinear hard macros are not supported.

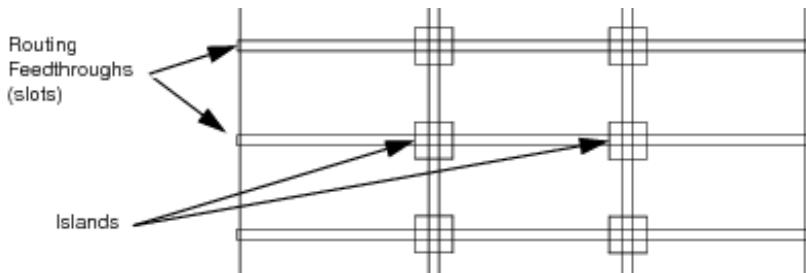
## Inserting Routing Feedthroughs

Routing feedthroughs and hole punch buffers reserve a portion of the partition area for top-level use. Because the partition's netlist does not change, no new ports are created for the partition. Buffers are inserted in top-level netlist but occupy space within the partition's fence. Partition feedthroughs are used to indicate the top-level partition's concession within the partition fence.

Partition feedthroughs should be defined before running the Partition program, which automatically generates appropriate placement and routing blockages within the partition and in top-level view to reflect the real estate ownership scheme. For example, a routing feedthrough with *Meta/6* will generate a *Meta/6* routing blockage for the partition, and an opening in the *Meta/6* blockage in the top level.

**Note:** The partition feedthrough discussed in this section is a floorplan object. It affects a partition only physically (not logically) and does not affect partition feedthrough buffer cells.

A *routing feedthrough* (slot) within the partition's fence is used by the top-level partition's routing, and an *island* within the partition's fence can be used by the top-level partition's placement, as shown in the following figure:



**Note:** Routing feedthroughs can be used without placement islands.

To create a channel-type feedthrough, use the [Create Physical Feedthrough](#) tool widget. After adding a partition feedthrough to the design, you can use the Attribute Editor to change its layers. The specified routing layers are reserved for top-level use, and the partition uses the other layers. You can create an island type partition feedthrough in a similar way, but all layers are deselected.

To insert routing feedthroughs and hole punch buffers, complete the following steps:

1. Create routing feedthroughs using one of the following methods:

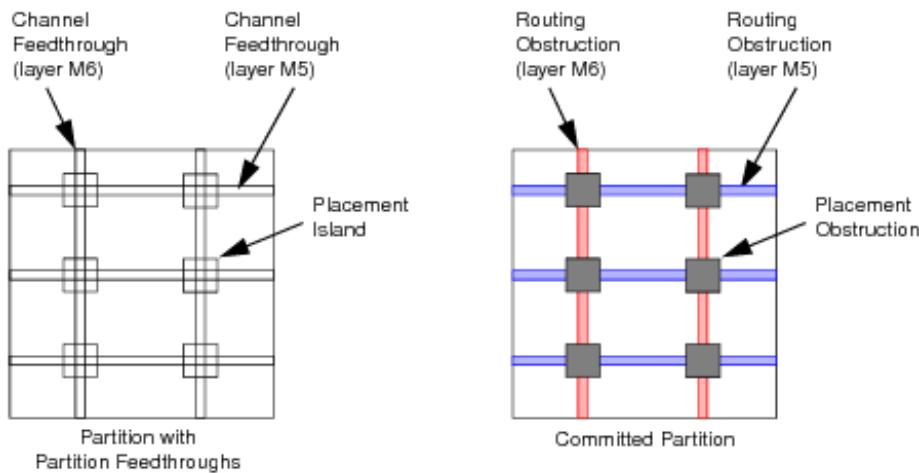
**Method 1:** Use the *Create Physical Feedthrough* widget to create the physical feedthrough on the partition. Select the feedthrough and open the [Attribute Editor](#) form, specify the metal layer, and click *OK*. This creates the channel for the routing on the specified layers at the top level, and pushes down appropriate routing blockages at the block level.

- Method 2:** If you want to specify narrow feedthroughs or several of them on a given partition, choose *Partition-Create Physical Feedthroughs* to open the Create Physical Feedthrough form. To specify which partition you want, click on the partition in the design display area, then click *get selected*. Complete the form and click *OK*.
2. (Optional) if you have hole punch buffers, create an island to specify where the holes are to be punched in the partition.

To do this, use the *Create Physical Feedthrough* widget to create a routing feedthrough and then deselect all layers after double-clicking on the physical feedthrough. This creates the island for buffer placement at the top level, and pushes down the appropriate routing and placement blockage at the block level by the partition command. At the top-level design, buffers can be placed into these created islands by IPO or buffer insertion.

3. Run Partition.

This automatically creates routing blockages for the channel feedthroughs, and placement blockages for the placement island, as shown in the following figure:



## Generating the Wire Crossing Report

You can display and write a file of wires that physically cross over partitions using the `showPtnWireX` text command or the *Partition - Show Wire Crossing* menu command.

The results are saved to a `designName.wirecrossing` file that reports nets that cross each partition in a design. For any net that crosses more than one partition, you can use it as a starting point for generating a list of nets for feedthrough insertion.

**Tip:** Edit the *designName*.wirecrossing file to exclude high fanout nets, clock nets, and nets that are connected to two or more glue logic standard cells to avoid timing and routing problems on these nets. You can use the resulting file with the `insertPtnFeedthrough` text command's -selectNet option. Note that the Innovus software determines the buffer tree topology, so not all specified nets will receive inserted feedthroughs. For example, nets that connect directly between adjacent partitions are not candidates for feedthrough insertion.

In designs with master clone partitions, the `showPtnWireX` command creates three additional output files. These files can be used by the `insertPtnFeedthrough -selectNet filename` command for path based feedthrough insertions in master and clone designs.

For example, when you specify,  
`showPtnWireX -outFile abc`

The following files are generated:

- `abc.mc_connected_nets.txt` : For all nets connected to master/clone partitions.
- `abc.mc_crossing_nets.txt` : For all nets with wires over master/clone partitions (but not in the above file) and not intra partition nets.
- `abc.non_mc.txt` : For all remaining nets which are not in above files and are not intra partition nets.

The default outfile file, `abc.wirecrossing`, is only valid for non master/clone designs and has no significance for master and clone designs. The `abc.wireCrossing` file does not include wire crossings over clone partitions.

These files contain just the net names. Their syntax is as follows:

```
NetA
NetB
NetC
NetD
...
...
```

To do path based feedthrough insertion for master clones nets, do the following:

- For `abc.mc_connected_nets.txt` and `abc.mc_crossing_nets.txt` files, use the following command without using the `-routeBased` parameter:
 

```
insertPtnFeedthrough -selectNet abc.mc_connected_nets.txt or,
insertPtnFeedthrough -selectNet abc.mc_crossing_nets.txt
```
- For `abc.non_mc.txt` file, use the following command with or without using the `-routeBased` parameter:

```
insertPtnFeedthrough -selectNet abc.non_mc.txt
```

## Interpreting the Wire Crossing Report

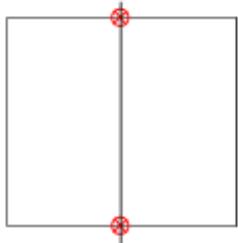
The wire crossing report section lists the nets, their wire lengths, in micrometers, and the shape of the wire in relation to the partition. For example, the following report segment is for a partition module named ptn01:

```
#####
# Nets that cross partition module ptn01
# Box (335 335) (833 567)
# Format: Net <netName> <wireLength> <shape>
#####

Net A 65 I
Net B 80 L
Net C 1050 T
Net D 132 X
...
```

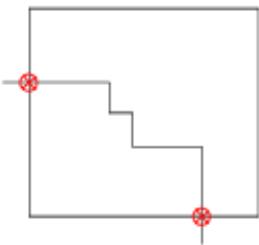
The first net in the report, **A**, has a wire length of 65 micrometers in an `I' shape, which indicates that the net crosses the partition on opposite sides, as follows:

```
Net A 65 I
```



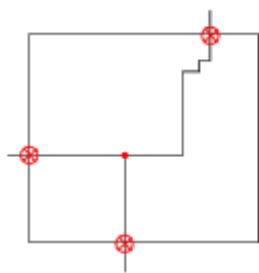
The second net in the report, **B**, has a wire length of 80 micrometers in an `L' shape, which indicates that the net crosses the partition on adjacent sides, as follows:

```
Net B 80 L
```



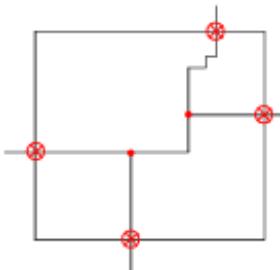
The third net in the report, C, has a wire length of 105 micrometers in an 'T' shape, which indicates that the net crosses the partition on three sides, as follows:

Net C 105 T



The fourth net in the report, D, has a wire length of 132.30 micrometers in an 'x' shape, which indicates that the net crosses the partition on all four sides, as follows:

Net D 132 X



In the report, you can also include the total length of the wire crossing the block in the horizontal X direction and total length of the wire crossing the block in the vertical Y direction using the -delta option of the showPtnWireX command. For example, the following report segment is for the same partition module named ptn01 using the - delta option:

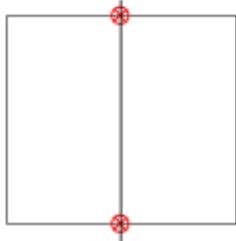
```
#####
# Nets that cross partition module ptn01
# Box (335 335) (833 567)
# Format: Net <netName> <wireLength> <shape> <deltaX> <deltaY>
#####
Net A 65 I 0 65
```

Net B 80 L 38 47

...

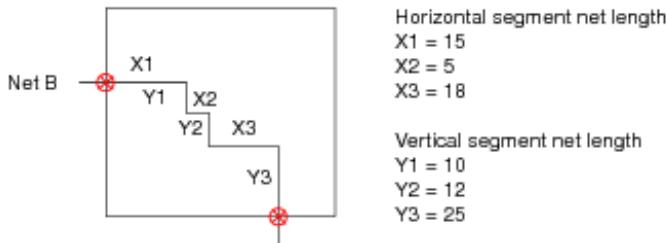
The first net in the report, A, has a wire length of 65 micrometers in an 'I' shape, with a total of 0 length in the horizontal X direction, and 65 in the vertical Y direction:

Net A 65 I 0 65



The second net in the report, B, has a wire length of 80 micrometers in an 'L' shape, with a total of 38 length in the horizontal X direction, and 47 in the vertical Y direction:

Net B 80 L 38 47



In the above example, the 38 length in the X direction is calculated for the X direction net segments ( $X_1 + X_2 + X_3$ ), and the 47 in the Y direction is calculated for the Y direction net segments ( $Y_1 + Y_2 + Y_3$ ).

## Estimating the Routing Channel Width

For committed partitions and blackboxes with assigned pins, channel width estimation uses the current pin assignment. If partition pins are not assigned, they are placed at the lower-left corner. In this case, the Innovus software issues a warning message because the estimator cannot produce a good result.

For uncommitted partitions, channel width estimation runs the Partition program, assigns pins, estimates the channel widths, and runs the Unpartition program. For blackboxes without assigned pins, it assigns pins and estimates the channel widths.

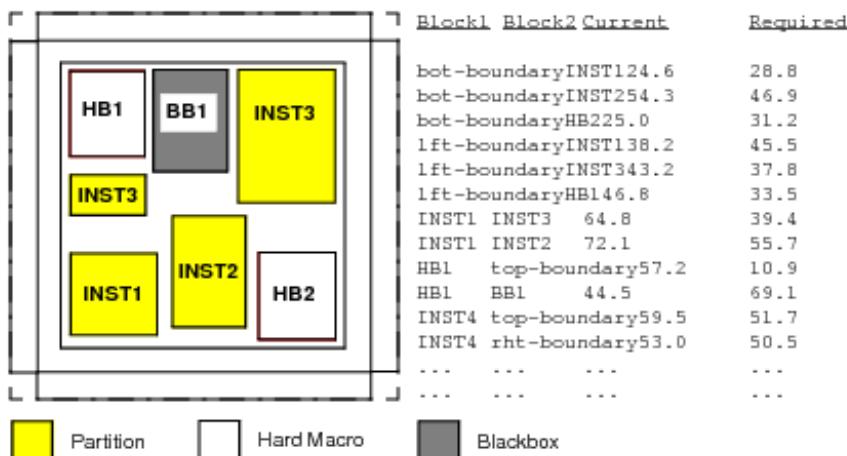
Channel width estimation also considers topology constraints to drive block placement. These

constraints are block-to-block boundary, block-to-block distance, block order and alignment, block aspect ratio, net weight (from global `trialRoute`), and block halo. The channel width estimator also respects these constraints so that their top-level block floorplans are not dramatically changed. If there is conflict between a specified constraint and the minimum required channel spacing, the Innovus software honors the minimum required channel spacing.

This feature produces a report containing the following information:

- Estimated required spacing, in micrometers, between partitions, blackboxes, and hard macros.
- Estimated required spacing surrounding each partition based on its pins (the relative distance between partition blocks required for top-level routing).
- Estimated distance between blocks and core boundaries (top, bottom, left, right).

The following figure shows an example of how the channel estimation report relates to the design:



## Running the Partition Program

The Partition program creates the partitions in the top-level design. This changes the module's status from a fence to a block and generates pins if routing data exists from running Trial Route. When the Partition program is run, the Trial Route data is deleted because the current placement and route data are not suitable for further work at the top level. The partition pin guide (floorplan) object can be used to determine the location of the pins, and nets or buses will be assigned to the partition pin guide objects.

If the partitions are changed, then the placement and Trial Route programs must be rerun. To change the status of the partition from being a hard block, you must run Unpartition to flatten the partition.

**Info:** After you run the Partition program and save the partition data, you should exit the session and

start a new session for the top-level design and for each partition in their newly created UNIX directories.

**Note:** Running the Partition program creates a blockage on an OVERLAP layer even though the OVERLAP layer is not defined in the technology section of the LEF file. As a result, the partition LEF file cannot be loaded into either the Innovus software or any standalone tools. If your design has rectilinear partitions or feedthroughs, the OVERLAP layer must be defined in the technology section of the LEF file.

**Info:** If a partitioned design is unpartitioned and then partitioned again, it will lose the original routing and timing information. The routing and timing information are not preserved during the unpartition-partition process.

To restore the timing information, Save your routing data before partitioning. If you unpartition later, run the `restoreRoute` text command to get the routing information, then run `extractRC`, and then `buildTimingGraph`, to restore timing information.

**Info:** To preserve the existing power/ground pins during partitioning and create additional pins based on the power structure that crosses partitions in the floorplan, use the `partition` command with the `-keepPGPin` parameter.

You can save the partition data in an OpenAccess database. For more information, see [Working with OpenAccess Database](#).

## Creating a Top-Level Partition

1. Run the Partition program.
2. Run Trial Route on the top-level partition.
3. Check for routing congestion.  
If there is no congestion, you are done. If there is congestion, continue to step 4.
4. Run the Unpartition program and add more routing resources to the congested area.
5. Rerun the Partition program.

Repeat steps 1 - 5 until there is no routing congestion.

## Block-Level Partition

To create a block-level partition, complete the following steps:

1. Run the Partition program.
2. Check to see if each partition size is suitable.  
If it is, you are done. If it is not, continue to step 3:
3. Run the Unpartition program.

4. Increase the size of the block.
5. Rerun the Partition program.

Continue with the steps above until you have reached suitable partition sizes.

## Pushing Down Signal Routes

During partition program, you can use the `-pushRoute` parameter of the `partition` command to push down signal routes to the respective partitions.

**Info:** Before running the partition `-pushRoute` command, you can check the hierarchy violations for nets on the partitions with the `checkHierRoute` command.

Here's the pushdown behavior with the `-pushRoute` parameter of the `partition` command:

- The following routes are pushed down:
  - Intra-partition nets routed completely within the routed boundary.
  - Inter-partition nets that cross the partition boundary only once *and* that pass through the partition pin location.
- Top nets that are routed completely in the top channels are retained at the top
- All other nets are deleted.

For nets that have a hierarchy violation, only the wire segments that have a hierarchy violation on the nets are discarded. The other wire segments are retained.

## How Top-level Stripes Are Pushed Down

This section explains how stripes on the top level are pushed down into the partition when you run the partitioning program. The following scenarios are discussed:

- The Default Behavior
- Behavior with the `-stripStayOnTop` Option

## The Default Behavior

The following table summarizes the default behavior.

Stripe Position	How Stripe Is Pushed Down
-----------------	---------------------------

Stripe is completely inside partition boundary	<ul style="list-style-type: none"> <li>● Top-level: The stripe is removed from the top.</li> <li>● Block-level: The stripe is pushed down as two pins and one stripe.</li> <li>● Block Abstract:           <ul style="list-style-type: none"> <li>● On layers reserved for partition, two pins are created on the boundary.</li> <li>● On layers not reserved for partition, one big LEF pin is created.</li> </ul> </li> </ul>
Stripe is partially inside partition boundary.	<ul style="list-style-type: none"> <li>● Top-level: The stripe is retained at the top.</li> <li>● Block-level: The stripe is pushed down as two pins and one stripe.</li> <li>● Block Abstract: The stripe is pushed down as a big LEF pin.</li> </ul>
Stripe is outside but close to partition boundary	<ul style="list-style-type: none"> <li>● Top-level: The stripe is retained at the top.</li> <li>● Block-level: The stripe is retained at the top and is copied as a routing blockage (same size as stripe) with a +PUSHDOWN attribute.</li> <li>● Block-abstract: No effect.</li> </ul>

## Behavior with the `-stripStayOnTop` Option

The `-stripStayOnTop` parameter in the `partition` command specifies that stripes that are not on a layer reserved by the partition are retained at the top level and are also copied into the partition. The following table explains how the stripes on the top level are pushed down to the partition when you run the partitioning program with the `-stripStayOnTop` parameter.

Stripe Position	How Stripes Are Pushed Down
-----------------	-----------------------------

Stripe completely inside partition boundary	<ul style="list-style-type: none"> <li>● Top-level:           <ul style="list-style-type: none"> <li>● On layers not reserved for partition, the stripe is retained at the top and is copied to the block-level design.</li> <li>● On layers reserved for partition, the stripe is pushed down to the block-level design.</li> </ul> </li> <li>● Block-level:           <ul style="list-style-type: none"> <li>● On layers not reserved for partition, the stripe is retained at the top and is copied as two pins and one stripe.</li> <li>● On layers reserved for partition, the stripe is pushed down as two pins and one stripe.</li> </ul> </li> <li>● Block Abstract:           <ul style="list-style-type: none"> <li>● On layers not reserved for partition, two pins are created at the edges.</li> <li>● On layers reserved for partition except the topmost layer, two pins are created at the edges.</li> <li>● On the topmost layer reserved for partition, one big LEF pin is created.</li> </ul> </li> </ul>
Stripe is partially inside Partition boundary.	<ul style="list-style-type: none"> <li>● Top-level: The stripe is retained on the top and is copied to the block-level design.</li> <li>● Block-level: The stripe is retained at the top and is copied as two pins and one stripe.</li> <li>● Block Abstract: The stripe is retained at the top and is copied as a big LEF pin.</li> </ul>
Stripe is outside but close to boundary	<ul style="list-style-type: none"> <li>● Top-level: Stripe is retained at the top.</li> <li>● Block-level: Stripe is retained at the top and is copied as a routing blockage (same size as wire) with a +PUSHDOWN attribute.</li> <li>● Block-abstract: No effect.</li> </ul>

## How Bumps, Routes, and Area I/O Cells Are Affected

This section illustrates how bumps and routes are handled when the design uses hierarchical partitioning with flip chip RDL routing and 45-degree routes. This information pertains to the [partition](#) command.

**Note:** In the releases of Innovus prior to 6.1, the area I/O cells had to be part of the top-level netlist; otherwise, DRC violations were reported during block implementation. From the 6.1 release onwards, the area I/O cells can be at the top level or be a part of the partition netlist. This section describes the behavior for both the cases.

After the partition, LEF obstruction is cut against the overlapping bumps at the top. This is done for all the bumps (power/gnd/signal/unused). Similarly the routing blockages inside the partition is cut against the pushed down bump.

The following scenarios are discussed:

- [Area I/O Cells are Part of the Top-level Netlist](#)
- [Area I/O Cells are Part of the Partition Netlist :](#)
  - Bumps and Routing are on Top Routing Layer--Behavior with the stripStayOnTop parameter
  - Bumps and Routing are on Reserved Routing Layer--Behavior with the stripStayOnTop parameter
  - Bumps and Routing are on Top Routing Layer--Default Behavior
  - Bumps and Routing are on Reserved Routing Layer--Default Behavior

### Area I/O Cells are Part of the Top-level Netlist

When area I/O cells are part of the top-level netlist, signal bumps and routes remain bumps and wires at the top level, but become routing blockages at the partition level. This allows routing at the block level while preserving the space for the signal bumps and routes.

Power and ground bumps and routes are copied and pasted (duplicated) from the top level to the partition. This allows power analysis at the block level. When the design is flattened, the duplicate power and ground bumps and routes are removed from the block level.

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

Signal bump	Signal bump	Placement and Routing Blockage
Signal route	Signal Route	Routing blockage
Power and ground bump	Bump	Bump (copied and pasted)
Power and ground route	Route	Route (copied and pasted)

## Area I/O Cells are Part of the Partition Netlist

When area I/O cells are part of the partition netlist, the pushdown behavior depends on:

- Whether the `stripStayOnTop` parameter has been specified with the [partition](#) command.
- Whether the bumps and routing are on the top routing layer or the reserved routing layer

**Info:** In this case (that is, area I/O cells are part of the partition netlist), the behavior applicable to area I/O cells is also applicable to any other instance to which the bump is logically connected.

If the area I/O cell and the bump connection pass through a partition pin, the pin will not be assigned when you assign partition pins. These partition pins are assigned only when you run the [partition](#) command. If the bump overlaps the partition, a partition pin is created, with a geometry similar to that of the bump. If the bump does not overlap the partition, the pin is created during special route pushdown. The pin is created on the partition boundary where the routes between the bump and the area I/O cross the partition boundary.

For floating partition pins that are connected to a bump, the [assignPtnPin](#) command will check if the bump physically overlaps with the partition. If so, the command will not assign the pin and a partition pin is created, with a geometry similar to that of the bump, only when the [partition](#) command is run. Otherwise, the pin is assigned on the partition boundary by [assignPtnPin](#) command.

The following sections discuss the behavior for the following cases:

- [Bumps and Routing are on Top Routing Layer--Behavior with the stripStayOnTop parameter](#)
- [Bumps and Routing are on Reserved Routing Layer--Behavior with the stripStayOnTop parameter](#)
- [Bumps and Routing are on Top Routing Layer--Default Behavior](#)
- [Bumps and Routing are on Reserved Routing Layer--Default Behavior](#)

**Note:** For all the listed scenarios, the push down behavior for signal routes is similar to the behavior described in the [How Top-level Stripes Are Pushed Down](#).

## **Bumps and Routing are on Top Routing Layer--Behavior with the `stripStayOnTop` parameter**

The following table summarizes the behavior when the bumps and the routing are on the top routing layer and you run the `partition` command with the `-stripStayOnTop` parameter.

Object Type	Top Level	Partition Level
Area I/O cell	An pin equivalent pin to the area I/O pin is created in the partition LEF file. This pin has the same size, location, and metal layer as the area I/O pin.	Area I/O cell is retained in the partition netlist
Signal bump	Signal bump stays on top and, additionally, an equivalent pin is created in the partition LEF file.	<ul style="list-style-type: none"><li>If the bump overlaps fully or partially with the partition, and connects to the partition: An equivalent pin for the signal bump is created in the partition LEF file. This pin has the same size, location, and metal layer as the bump.</li><li>If the bump overlaps with the partition but is <i>not</i> connected to the partition: The signal bump is pushed down as a routing blockage.</li></ul>

Signal Routes	<p>Signal Routes routed on the top routing layers stays at top.</p>	<ul style="list-style-type: none"> <li>● If the signal route overlaps the partition and is also connected to a area I/O cell inside the overlapping partition and a signal bump at the top, the signal route is copied and pasted to the partition. The pushed down net will be the internal net in the partition and will be named based on the partition port it is connected to inside the partition.</li> <li>● If the signal route overlaps the partition to which it is not connected (that is, it is not connected to any instance inside the partition but to a bump at top), these routes are copied and pasted as routing blockages inside the overlapping partition.</li> </ul>
---------------	---	--

## ***Bumps and Routing are on Reserved Routing Layer--Behavior with the stripStayOnTop parameter***

The following table summarizes the behavior when the bumps and the routing are on the reserved routing layer and you run the `partition` command *with* the `-stripStayOnTop` parameter.

Object Type	Top Level	Partition Level
Area I/O cell	Not applicable because area I/O cell is already part of the partition netlist.	Area I/O cell is retained in the partition netlist.
Signal bump	Signal bump stays on top and, additionally, an equivalent pin is created in the partition LEF file.	Bumps get pushed down to the partition as an equivalent pin in the partition DEF file.
Signal route	Signal routes are removed from the top.	Routing gets pushed down inside the partition block

## **Bumps and Routing are on Top Routing Layer--Default Behavior**

The following table summarizes the default behavior when the bumps and the routing are on the top routing layer. The default behavior in this context refers to the behavior that occurs when you run the `partition` command *without* the `-stripStayOnTop` parameter.

Object Type	Top Level	Partition Level
Area I/O cell	Not applicable because area I/O cell is already part of the partition netlist.	Area I/O cell is retained in the partition netlist
Signal bump	Bump Stays at Top. An additional Bump pin is created in partition LEF file.	Bumps get pushed down inside the partition block as an equivalent pin in partition DEF file.
Signal route	Signal routes are removed from the top.	Routing gets pushed down inside the partition block The routes on top routing layer are cut from the top and pasted inside the partition. For details, please refer to <a href="#">How Top-level Stripes Are Pushed Down</a> .

## **Bumps and Routing are on Reserved Routing Layer--Default Behavior**

The following table summarizes the default behavior when the bumps and the routing are on the reserved routing layer.

Object Type	Top Level	Partition Level
Area I/O cell	Not applicable.	Area I/O cell is retained in the partition netlist
Signal bump	Signal bump stays at top. An additional bump pin is created in the partition LEF file.	Bumps get copied down inside the partition block as a pin.

Signal route	Signal routes are removed from the top.	Routing gets pushed down inside the partition block.
--------------	---	--

## Limitations

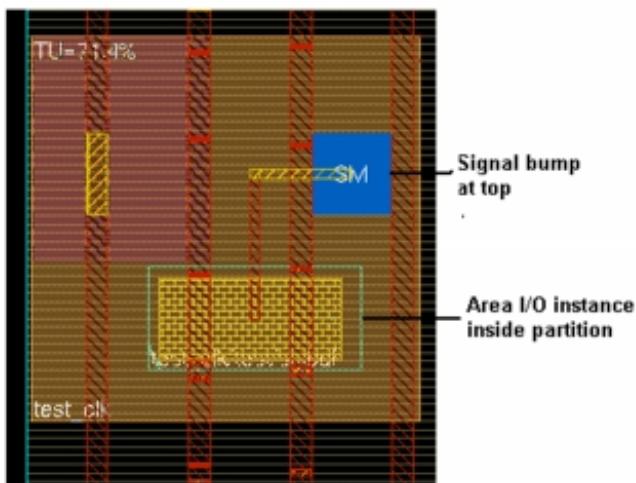
- The pushdown of the signal bumps as an equivalent pin inside the partition is not supported for the non-rectangular shapes of the bump cell.
- If the pushed down area I/O cell has pin shapes on the top routing layers, the blockages created on the top routing layers are not cut against these component pins.
- If the signal routes are pushed down to the partition, any routes that do not overlap with the partition but lie close enough to the partition boundary and may thus result in spacing violations at chip assembly, will be pushed down as blockage inside the partition. This may result in some blockages being pushed down to the partition but outside the partition box.

The following examples illustrate the behavior:

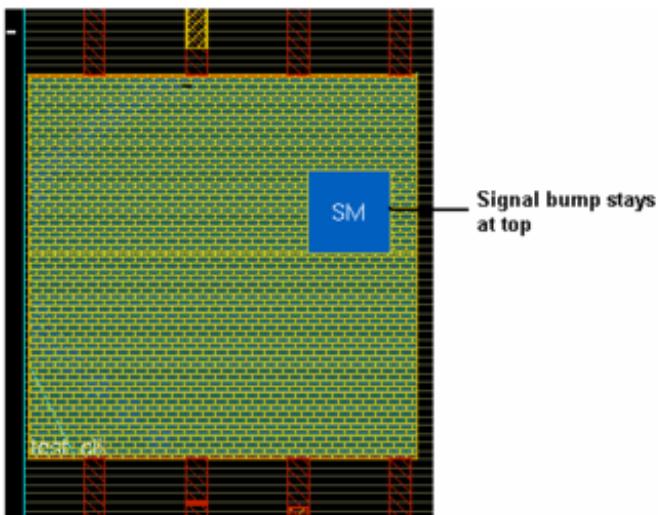
- [Case 1: All Routing Layers Reserved for the Partition](#)
- [Case 2: Top Layer Not Reserved for Routing](#)

### ***Case 1: All Routing Layers Reserved for the Partition***

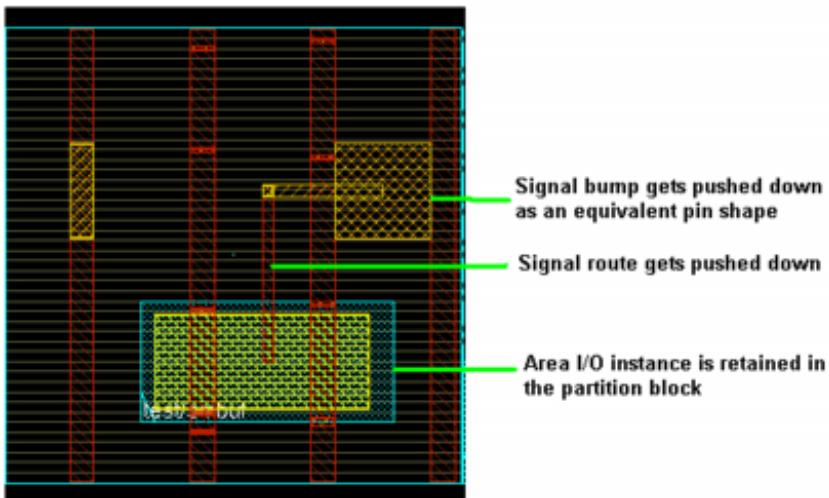
The design has six routing layers. All the layers are reserved for the partition. Signal Bump SM is connected to area I/O cell inside the partition. The following diagram shows the floorplan view before partitioning.



The following figure shows the view at top after partitioning.



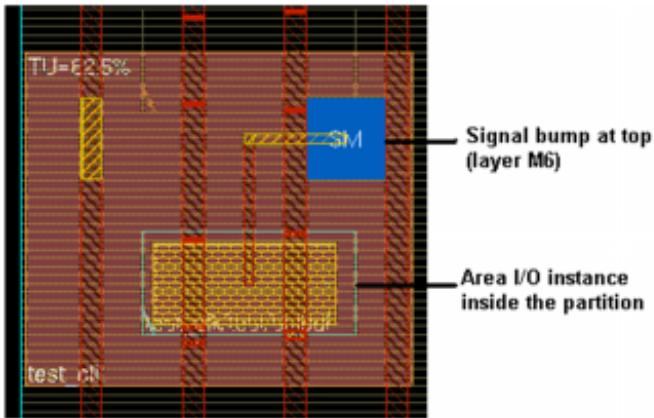
The following figure shows the view inside the partition



## ***Case 2: Top Layer Not Reserved for Routing***

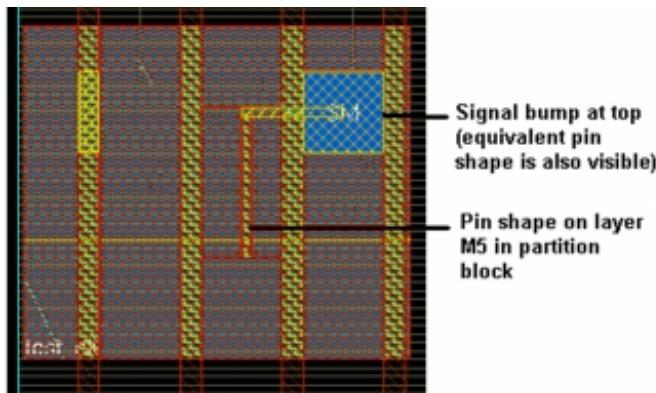
The design has six routing layers. Layers M1-M5 are reserved for the partition. M6 is the top routing layer.

The following diagram shows the floorplan view before partitioning.

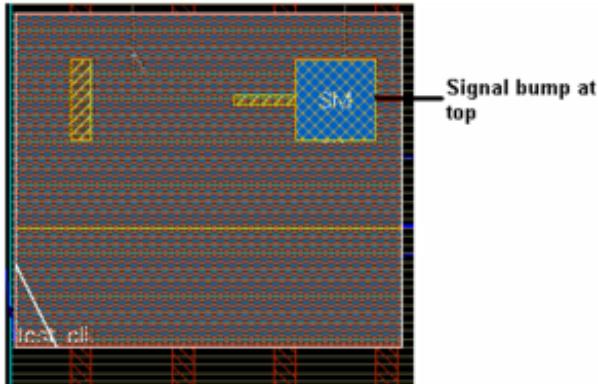


The following diagram shows the view at the top after partitioning with the `-stripStayOnTop` parameter specified.

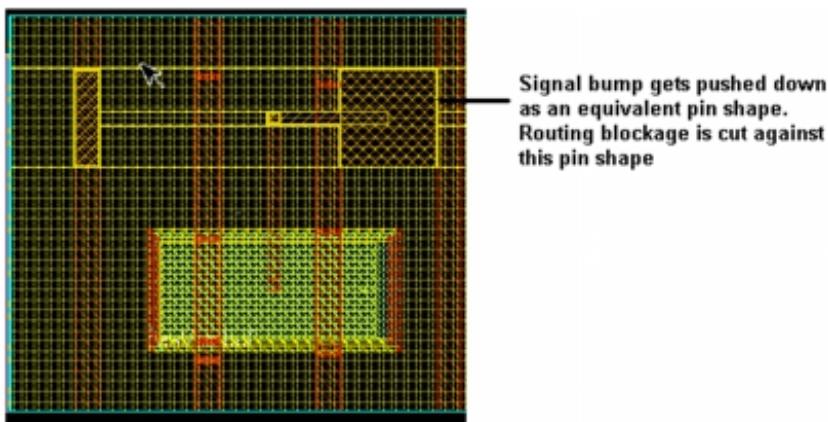
The following figure shows the view on the top after partitioning with the pins visible.



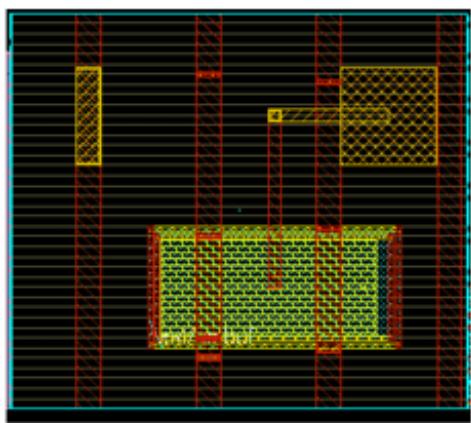
The following figure shows the view on the top after partitioning *without* the -stripStayOnTop parameter specified.



The following figure shows the view on the top after partitioning with visible routing blockages on layer M6.



The following figure shows the view inside partition with the display of the routing blockages turned off



## Restoring the Top-Level Floorplan with Partition Data

1. Import the entire design from the top-level directory that was created or updated when you saved the partition.

If any portion of the design (top level or any partition) was changed by running scan optimization, CTS, or IPO, the changed netlist of the entire design is imported, not the original netlist. This changed netlist is usually created by concatenating each of the partition netlists to the top-level netlist. To do this, use a text editor to manually edit it, or use the Design Import form to create a single Verilog netlist of the entire design (see [Concatenating Netlist Files of a Partitioned Design](#)).

**Info:** If a tool changes the partition netlist, you must update the full chip netlist. Some routers, such as NanoRoute™, might modify the partition netlist. The Innovus software requires that the full chip netlist, loaded during Design Import, be consistent with the routed partition netlist.

2. Load the top-level floorplan used to partition the entire design.
3. Set the Partition forms with *Perform Pin Assignment* deselected and partition the design.
4. Load the top-level placement data from the top-level directory.
5. Choose *File - Load - Partition*.  
This opens the Load Partition form.
6. In the Load Partition form, enter the directory name where the partition data was saved.
7. Click *OK*.

**Tip:** In place of steps 5, 6, and 7, you can use the setTopCell command to restore the partition and top-level data for the entire design. This is especially useful for restoring placement data from a DEF or TDF file.

**Note:** To perform a full chip analysis or a timing budget refinement analysis, use the Unpartition form to flatten the design.

## Concatenating Netlist Files of a Partitioned Design

To create a single Verilog netlist of the entire design, including the top level and all the partitions, complete the following steps:

1. Start a new Innovus session.
2. Choose *File - Import Design* to open the Design Import form, and click the *Basic* tab if it is not selected.
3. In the *Verilog Files* field, enter each netlist filename in the order from top-level netlist followed by the partition netlist files.

**Note:** The partition netlist are read from each of the partition's work directories.

4. Click *OK*.
5. Choose *File - Save - Netlist* to open the Save Netlist form.
6. Enter a Verilog file netlist name in the *Netlist File* field.
7. Click *OK*.
8. Use the saved Verilog file to restore the top-level floorplan with partition data (see [Restoring the Top-Level Floorplan with Partition Data](#)).

## Saving Partitions

You can save partition results, including the top-level partition, to their own subdirectories so that each partition can be worked on concurrently. Each partition directory contains all files necessary to run the Innovus software. Files necessary to run back-end tools in DEF, PDEF, and TDF formats can be selected when saving partitions.

To save a partition, use the [Save - Partition](#) form or the `savePartition` text command.

**Warning:** *Do not use the Save Design form to save a partition.*

You can also save partitions in the OpenAccess database format. For more information, see [Working with OpenAccess Database](#).

## Loading Partitions

After completing the design work for each partition and the top level, you can restore a partitioned design to the top level, which includes loading all the partition design directories and its data. To restore a saved partition design, use the [Load - Partition](#) form.

## Unpartitioning with Routing Data

When loading a partition, it is important that the loaded routing results correctly correspond to the new netlist. To ensure that the netlist and routing file are consistent, you need to unpartition with the routing data using the following steps:

1. Load the original flat design.  
This is the original design before running the partition steps.
2. Specify the partition and save to a file (to be loaded later in step 7).
3. Run partitioning with pin assignment.
4. Save the partitions and the top level.
5. For each partition and the top-level, run the block-level implementation with the following commands:
  - innovus
  - restoreDesign (for the block or top level)
  - trialRoute **OR** nanoroute **OR** wroute
  - saveRoute
6. Load the original design (the same design loaded in step 1).  
If the netlist has been modified after step 1 (for example, in the case where a netlist is modified after in-place optimization or running NanoRoute) use the updated netlist instead.  
To specify the updated netlist, you must first specify top-level netlist, then the block-level netlists in the *Verilog Files* field of the Design Import form's *Design* page.  
For example, top.v block1.v block2.v ...

**Info:** The netlist and routing must be consistent when loading a partition with routing data, be sure you load the design with floorplanning, placement, and routing data that is consistent with the data saved in step 4.

7. Load the partition file (specified in step 2).
8. Run partitioning without pin assignment.
9. Load the partition data.  
For each partition, select the partition, then change the partition view (using the *Partition - Change Partition View* menu command) and load all the data for the viewed partition. You can use either the DEF file, or the .fp, .place and .route files.
10. Reset the view back to the top level (using the *Partition - Change Partition View* menu

- command).
11. Load the top-level data.  
You can read in the top-level physical information by either using the DEF file or the placement (.place) and routing (.route) file. You must not read in the floorplan (.fp) file again because the floorplan information was already read in at the very beginning.  
**Note:** Top-level physical information can only be loaded using DEF.
  12. Unpartition the design (`flattenPartition`).

## Working with OpenAccess Database

You can save and load designs using the OpenAccess database. The following commands and parameters are used for OpenAccess database designs.

- The `savePartition` command can save files in OpenAccess database format:
  - `-ptnLib`  
Specifies an OpenAccess directory library name where the top-level and the block-level designs will be saved.
  - `-ptnView`  
Specifies a view name for the top view and the partition view.
  - `-refLibs`  
Specifies a list of reference libraries.
- The `assembleDesign` command supports assembling the saved OpenAccess format files.
  - `-topDesign`  
Specifies the top-level name.
  - `-block`  
Specifies the block names.

The general flow for designs that use an OpenAccess database is the same as described throughout this chapter.

The following command saves the partition information/files in the OpenAccess database format. The information for the top and the block level designs (all blocks) will be written in the `libForOA` directory view with the view name `ptnView1`.

```
savePartition -ptnLib libForOA -ptnView ptnView1
```

The following command assembles the design after bringing back information from the top-level cell

DTMF and block-level cells TDSP\_CORE and TDSP\_ARB.

```
assembleDesign -topDesign libForOA DTMF ptnView1 -block libForOA TDSP_CORE ptnView1 -  
block libForOA TDSP_ARB ptnView1
```

## Parallel Job Processing

With parallel processing, you can distribute jobs using a remote shell (rsh) or load sharing facility (LSF), specify host names for running jobs, and specify job information, such as block working directories and their run scripts. The following procedure provides the most common steps for parallel job processing:

1. Import the design.
2. Floorplan the design.
3. Assign pins.
4. Run Timing Budgeting.
5. Partition the design.
6. Save the partition
7. Run parallel job processing to implement the blocks.

For more information, see:

- [Set Multiple CPU Usage](#) in the "Options Menu" chapter of the *Innovus Menu Reference*.
- [trialRoute](#)
- [setAllowedPinLayersOnEdge](#)

## Focused Methodologies

In addition to generic flow methodologies, there are some specific requirements for various design styles. They are covered as under:

- [Correcting Pin Illegality On Selected Pins](#)
- [Selecting Pins Using a File](#)
- [Assigning Pins of a Net](#)
- [Assigning Pins in Pre-feedthrough Netlist](#)
- [Doing Pin Prioritization](#)
  - [Prioritizing Few Pins in a Selected Pin Assignment Flow](#)

- Speeding Up Interactive Pin Assignment
- Deciding the Closest Legal Location to a Selected Position

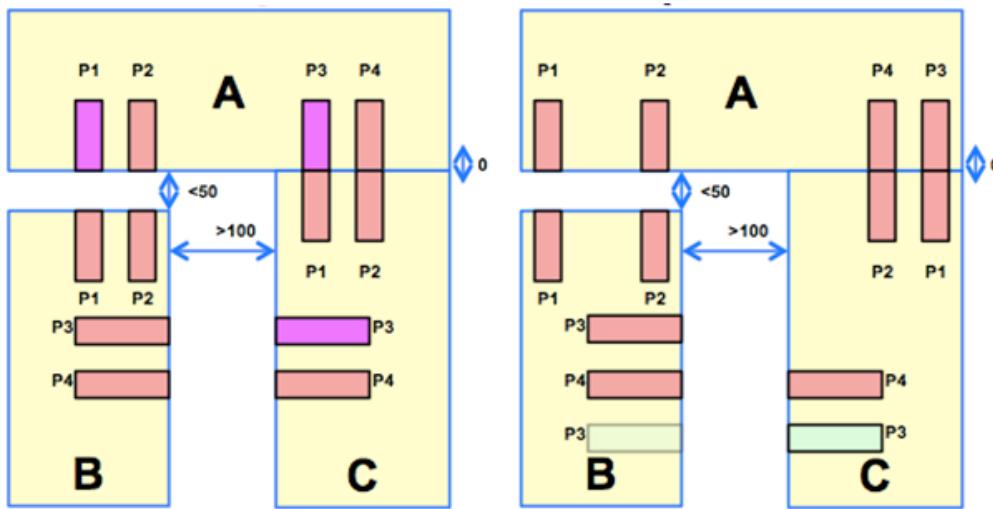
## Correcting Pin Illegality On Selected Pins

You can use the `legalizePin` command to automatically correct all violations on the design. When the `checkPinAssignment` command suggests an error on a pin, this pin is part of a net that has one pin emanating from the partition containing the source and the other pin concluding at partition with sink. In order to avoid bends in the route, it is always better to move pins on both partitions. This is particularly important when a design has a narrow or no channel in between the two partitions. For abutted designs (no channels), even if you select a pin for legalization (`legalizePin -ptn ptnName -pin pinName`) the Innovus will move its connected pin to a location where both pins will be legal.

The behavior of moving pins in pairs (for a selected pin) is controlled by distance between two partitions. With the use of the `-auto_pairing` parameter, the `legalizePin` command automatically calculates the channel width or distance between pins in a design which will be considered for pairing.

Alternatively, for a distance (in microns) between the pins, to move corresponding connected pins of 2-pin-connection nets, you can use the `setPinAssignMode -max_distance_pairing` command. The `-max_distance_pairing` parameter behavior extends to partitions which are less than 50 microns apart (default value) and can be easily used to set the value of the distance (lower or higher than 50) for which the pins should be moved in pairs automatically.

The following example illustrates this behavior:



In the above figure, the design has no channel between Partition A and Partition C. It has <50 microns channel between Partition A and Partition B and a channel of >100 microns between Partition B and Partition C. The pink colored pins (P1, P3 on partition A and P3 on Partition C) are illegal in the design.

Specifying the following command will only move - P1 and P3 of Partition A, P1 of Partition B, and P1 of Partition C:

```
legalizePin -ptn A -pin {P1 P3}
```

and specifying the following command will only move P3 on Partition C:

```
legalizePin -ptn C -pin {P3}
```

In order to move pin P3 of Partition B along with P3 of Partition C, you must use the following command:

```
setPinAssignMode -max_distance_paring 100
```

Now, P3 of Partition B will also move to the shaded green location (as shown above) along with P3 of Partition C.

**Note:** Even though the pins P1 of Partition B or Partition C are legal, they will still be moved to get an alignment with their corresponding connected pin.

You can avoid moving the corresponding pin for any channel width (routing bends are acceptable in channels) by using the distance value for the `-max_distance_paring` parameter as 0. In such a case, whatever may be the distance between the channel, but the pins will not move in pairs and the legal pins will remain on their original position.

```
setPinAssignMode -max_distance_paring 0
```

## Selecting Pins Using a File

The assignIoPins, assignPtnPin, checkPinAssignment, legalizePin commands can work on selected pins also.

For example,

```
assignIoPins -pin {out1 out2 out3}
assignPtnPin -ptn A -pin {in1 in2 in3} -ptn B -pin {pina pinb pinc}
checkPinAssignment -ptn A -pin {in1 in2 in3}
legalizePin -ptn A -pin {in1 in2 in3}
```

This selection of pins can also be achieved using a list of pins, per partition, from a file.

In the pin file, the pins are specified in the following format:

Partition: *PtnName/BlockName*

*Pin1*

*Pin2*

..

*PinN*

Partition: *PtnName/BlockName*

*Pin1*

*Pin2*

..

*PinN*

Nets:

*netA*

*netB*

..

*netN*

For example, consider the following use model:

```
assignIoPins -pin_file pin.list
assignPtnPin -pin_file pin.list
checkPinAssignment -pin_file pin.list
legalizePin -pin_file pin.list
```

where, pin.list contains:

```
Partition: top
out1
out2
out3
```

Partition: A

in1

in2

in3

Partition: B

pina

pinb

pinc

Now, the `checkPinAssignment` and `legalizePin` commands will work for block: top, Partition: A and B.

All pins present in the pin.list file will be considered for selection.

Now,

For `assignIoPins` command, all Io pins will be selected, under the section `Partition:top`.

For `assignPtnPin` command, all pins of all partitions in the file will be considered, except the pins under the section `Partition:top`.

For `checkPinAssignment` and `legalizePin` commands, pins of all partitions and block (block: top, Partition: A and B) will be considered.

**Note:** To skip one of the partitions from selection, you can use the `-exclude_ptn` parameter. However this parameter is not applicable for the `assignIoPins` command.

For example:

- The following command ignores the partition B mentioned in the pin file pin.list.

```
assignPtnPin -pin_file pin.list -exclude_ptn {B}
```

The select pins are similar to the ones selected with `assignPtnPin -ptn A { in1 in2 in3}` command.

- The following command only considers pins of partition A . pins of block Top and of partition B are ignored.

```
checkPinAssignment -pin_file pin.list -exclude_ptn {top B}
```

The select pins are similar to the ones selected with `checkPinAssignment -ptn A -pin {in1 in2 in3}` command.

- The following command considers pins of block Top and of partition B. It ignores pins of partition A mentioned in the pin file pin.list

```
legalizePin -pin_file pin.list -exclude_ptn {A}
```

The select pins are similar to the ones selected with:

```
legalizePin -ptn top -pin { out1 out2 out3}
```

```
legalizePin -ptn B -pin {pina pinb pinc }
```

## Assigning Pins of a Net

In the selected pin assignment flow, using the pin file for selecting pins offers an additional benefit. Instead of defining a set of pins which are connected to a net and is part of different partitions, you can select the net name itself. As a result, all the pins connected to different partitions will be derived automatically.

Partition: *PtnName/BlockName*

*Pin1*

*Pin2*

Nets:

*netA*

*netB*

..

*netN*

For example, consider a Net *n1* which connects to 3 partition pins, pin *x* on partition A, pin *y* on partition B, and pin *z* on partition C.

The following command will do selected pin assignment of just net *n1*.

```
assignPtnPin -ptn a -pin {x} -ptn b -pin {y} -ptn c -pin {z}
```

Alternatively, you can use the *-pinFile* parameter to do the pin assignment for net *n1* using the following command:

```
assignPtnPin -pinFile a.list
```

where, *a.list* contains:

Net:

*n1*

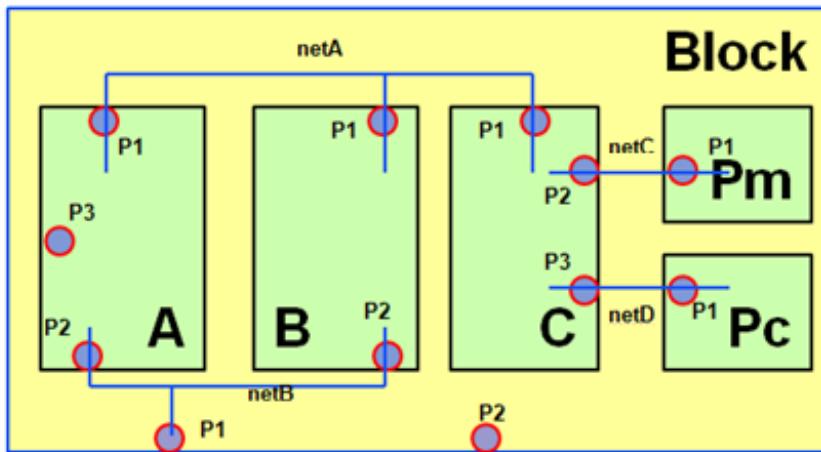
All net names specified under the Net section of the pin file will have all their pins assigned.

**Note:** You can use the *-exclude\_ptn ptnName* parameter to ignore the pins of the specified partition.

## Using pins and nets in the same pin file

The pin file supports explicit pin names for each partition but the section for net names is implicit of pins of more than one partition

### Use Model: Design where the master and clone are interacting with each other



The pin file `p.list` contains:

Nets:

`netA`

`netB`

`Ptn:Block`

`P2`

`Ptn:A`

`P3`

- The following command will place block pin P1, which has been implicitly mentioned in the pin file, through netB and block pin P2, which has been explicitly mentioned in the pin file.

```
assignIoPins -pin_file p.list
```

- The following command will place pins on nets, netA and netB, connected to all partitions (implicit mention in the file) pin P1 of PTN C, PTN B, PTN A and pin P2 of PTN B, PTN A. Pin P3 of PTN A is mentioned explicitly in the pin file.

```
assignPtnPin -pin_file p.list
```

- The following command will place pins on nets, netA and netB, connected to PTN B (pin P1 and P2) and PTN C (pin P1) but will not place Pin P1 (netA) and Pin P2 (netB) on PTN A (implicit mention in file) and Pin P3 on PTN A (explicit mention in file)

```
assignPtnPin -pin_file p.list -exclude_ptn {A}
```

- The following command check and legalize all pins including blocks pins P1 of PTN C, PTN B, PTN A and pin P2 of PTN B and PTN A, P3 pin on PTN A, P1 and P2 pin on block.

```
checkPinAssignment -pin_file p.list ;  
legalizePin -pin_file p.list
```

- The following commands will check and legalize pin P1 on PTN B and pin C and pin P2 on PTN B but will ignore pin P1, P2 of Block and pins P1, P2, P3 of PTN A

```
checkPinAssignment -pin_file p.list -exclude_ptn {A Block} ;  
legalizePin -pin_file p.list -exclude_ptn {A Block}
```

*For master and clone connections:*

If pin file pc.list contains:

Nets:  
*netC*

- The following command will place pin P1 on PTN P (both master and clone) and pin P2 only on PTN C.

```
assignPtnPin -pin_file pc.list
```

If pin file pd.list contains:

Net:  
*netD*

- The following command will place pin P1 on PTN P (both master and clone) and pin P3 only on PTN C.

```
assignPtnPin -pin_file pd.list
```

- The following command will place pin P1 on both master and clone.

```
assignPtnPin -pin_file pd.list -exclude_ptn {C}
```

If pin file pe.list contains:

Ptn:P  
*P1*

- The following command will place pin P1 on PTN P (both master and clone).

```
assignPtnPin -pin_file pe.list
```

If pin file pf.list contains:

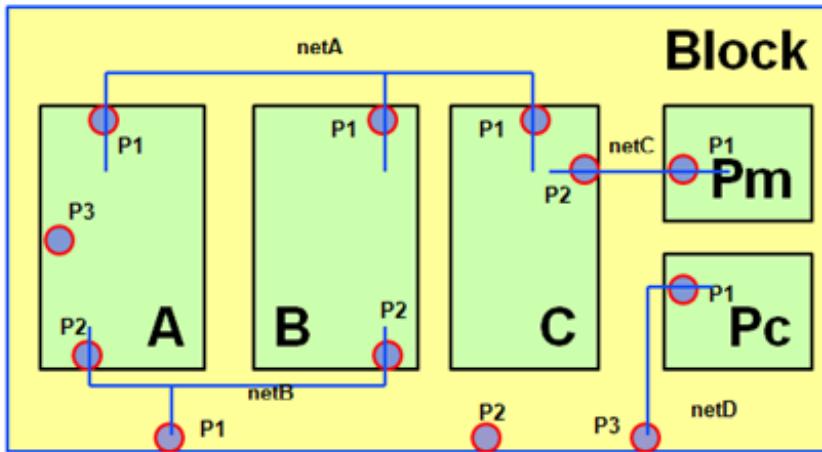
Ptn:C  
P2

- The following command will only place pin P2 on PTN C.

```
assignPtnPin -pin_file pf.list
```

**Note:** The pin selection for commands assignIoPins, checkPinAssignment, and legalizePin will be the same.

**Use Model: Design where the master and clone are interacting with top**



If pin file pc.list contains:

Nets:  
netC

- The following command will consider pin P1 on partition P (both master and clone) and only consider pin P2 on PTN C.

```
assignPtnPin -pin_file pc.list
```

- The following command will consider pin P1 on partition P (both master and clone)

```
assignPtnPin -pin_file pc.list -exclude_ptn {C}
```

- The following command will only consider pin P2 on PTN C

```
assignPtnPin -pin_file pc.list -exclude_ptn {P}
```

If pin file pd.list contains:

Nets:

*netD*

- The following command will consider pin P1 on partition P (both master and clone)

```
assignPtnPin -pin_file pd.list
```

- The following command will assign pin P3 of block.

```
assignIoPins -pin_file pd.list
```

If pin file *pe.list* contains:

Ptn:P

*P1*

- The following command will place pin P1 on PTN P (both master and clone).

```
assignPtnPin -pin_file pe.list
```

If pin file *pf.list* contains:

Ptn:C

*P2*

Ptn:Block

*P3*

- The following command will only place pin P2 on PTN C.

```
assignPtnPin -pin_file pf.list
```

- The following command will assign pin P3 on Block.

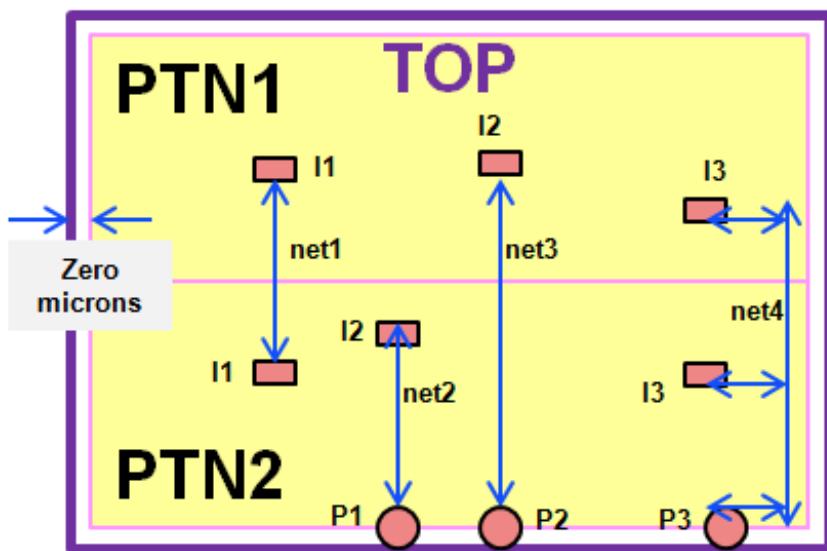
```
assignIoPins -pin_file pd.list
```

**Note:** The pin selection for commands `assignIoPins`, `checkPinAssignment`, and `legalizePin` will be the same.

## Assigning Pins in Pre-feedthrough Netlist

Abutted designs have no channels between partitions. You must use the `insertPtnFeedthrough` command on such designs before pin assignment so that ports are inserted into the partitions which lie in between the partitions that contain source and sinks. This makes the design routable by avoiding any violations in the hierarchy.

The following example illustrates this behavior:



The design shown in the figure above is a fully abutted design with block TOP and partitions PTN1 and PTN2.

It has the following 4 nets:

- net1 connects inst I1 of PTN1 to inst I1 of PTN2
- net2 connects inst I2 of PTN2 to TOP (block) port P1
- net3 connects inst I2 of PTN1 to TOP port P2
- net4 connects inst I3 of PTN1 to inst3 of PTN2 and top port P3

For nets net1 and net2 a feedthrough is not required as the net routing will not cross over unrelated partitions. However, nets net3 do require a feedthrough as the route would cross over unrelated partition PTN2.

for and net4, feedthrough is required because PTN2 requires 2 ports (for entry and exit points of routes) but it just has the entry point port.

When you use the `assignIoPins`, `assignPtnPin`, `pinAlignment` commands, they will give the results shown in Figure 1. They cannot put pins for nets net3 and net4, because it will create illegal pins. The `checkPinAssignment` command will report abutment violations on pins if you manually try to keep pins as shown in Figure 2. The `legalizePin` command will not touch them, as it does not have any legal pin location for such pins

image2014-1-23 12:33:49.png

Thus to resolve these issues and get legal pin locations, the feedthrough of these nets is required.

To get pins (of nets that would require feedthrough) assigned in pre feedthrough netslist stage use the following command:

```
setPinAssignMode -strict_abutment false
```

This would:

- Enable the `assignIoPins`, `assignPtnPin`, `pinAlignment` commands to keep pins with abutment violations.  
**Note:** Pin P3 of PTN2 can come at 2 locations: aligned with P3 of PTN1 or can be aligned to top port P3 (shown in light color in the figure)
- Enable the `checkPinAssignment` command to avoid reporting abutment violation on these nets.
- Enable the `legalizePin` command to move such pins in order to remove other type of issues.

**Note:** The `checkPinAssignment` command will continue to flag genuine abutment violations. For example, if pin P1 of PTN1 and PA of PTN2 are placed at not aligned location, then the `checkPinAssignment` command will report abutment violation for them, even with the strict abutment mode is set to false. To avoid reporting such violations, you can use the `checkPinAssignment -ignore {pin_abutment}` command to ignore the abutment check all together.

## Doing Pin Prioritization

As there may be limited slots which ensure less congestion and a better route length resulting in an improved timing result, you may want to prioritize putting some pins first in the pin assignment flow. You can use the following methods to prioritize pins and get better pin locations for certain pins:

## Method 1

Use the `specifyNetWeight` command to specify the priority weighting of a net. Pins that connect to a net that has a higher net weight are assigned before the pins that connect to a net with a lower net weight.

For example, the following commands set the priority net weighting for nets net1 to first and net2 to second. All pins on net1 will have more slots to choose from:

```
specifyNetWeight net1 5
specifyNetWeight net2 4
assignPtnPin
```

Pins with net weights are prioritized after putting:

- Pins with location constraints, set using the `setPinConstraint` command
- Pin/net grouped/guided pins
- Master and clone pins

**Note:** If a slot has been assigned to a grouped pin in the default pin assignment flow, this slot cannot be assigned to any pin of a net by setting the net weight on it.

## Method 2

Select only a few pins for pin assignment to ensure that these pins have maximum slots to choose from and decide on the best pin locations.

For example, the following commands place the pins `in[1:5]` (`in1`, `in2`, `in3`, `in4`, and `in5`) first. The pins `out[1:5]` (`out1`, `out2`, `out3`, `out4`, and `out5`) will be placed later. The `out[1:5]` pins will not move pins `in[1:5]` from their positions, thereby ensuring `out[1:5]` will occupy from the rest of the pins.

```
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}
assignPtnPin -ptn A -pin {out1 out2 out3 out4 out5}
```

**Note:** You can also do the pin selection using the `pin_file` flow to achieve the same results.

**Note:** If the selected pins are part of a pin group, then all the pins of the group will be placed in the flow.

For example, If pin `in2` is part of pin group `GRP` having pins `{data reset in2}` then the following command will place all pins including pins `in1`, `in2`, `in3`, `in4`, `in5`, `data`, and `reset`.

```
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}
```

However, you can do the following to avoid placing the whole group of pins:

1. Use the `assignPtnPin -ignore_group_pins` command.

then the following command will place only the pins in1, in3, in4, and in5 .

```
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5} -ignore_group_pins
```

2. Use the `createPinGroup PinGroup -optimizeOrder` command to specify that pins in the *PinGroup* are reordered to optimize wire length.

Now pin in2 is part of pin group GRP having pins {data reset in2} with `-optimisedOrder`, then the following command will place pins in1, in2, in3, in4, in5

```
createPinGroup PinGroup -cell {A} -pin {data* reset in2} -optimizeOrder  
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}
```

**Note:** If the selected pin is of a master or clone, then pin on both the master and clone will be placed (and not the pin of the other connected partitions). However, connectivity to other partition for both master and clone will be considered.

The following example illustrates this behavior:

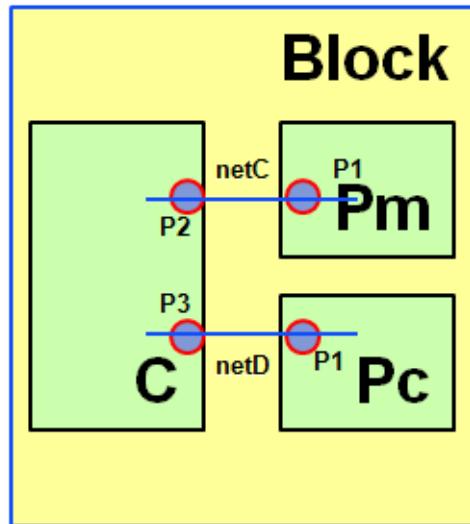


Figure 1

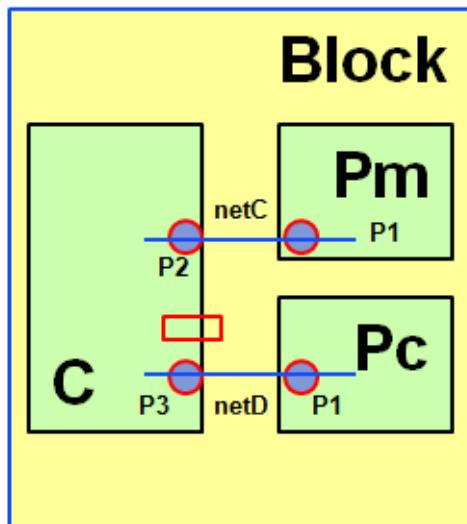


Figure 2

As shown in Figure 1, specifying the following command will place pins of both master Pm and clone Pc while considering pins P2 and P3 of partition C (part of net netC and netD).

```
assignPtnPin -cell P -pin {P1}
```

In Figure 2, there is a pin blockage on partition C. Now, pin P1 of partition P will be placed in

such a way that they consider connection of both nets connecting to (master/clone) partition pin P1. Hence, the pins will be placed at location which is optimized with position of pins P2 and P3 of partition C. In the figures above, the pins of partition C are shown for illustration only and will not be placed by command `assignPtnPin -cell P -pin {P1}`.

**Note:** Both the methods of pin prioritization honor constraints on other connected pins or partitions.

For example, consider a net which has 2 pins. The first pin going to partition A through pin in1 and the second pin connected to partition B though pin out1. Pin out1 of partition B has a location constraint.

In this case while doing selected pin assignment of partition A using the following command  
`assignPtnPin -ptn A -pin {in*}`)

the position of pin in1 of partition A will be chosen considering the location constraint of pin out1 of partition B (even though pin out1 of partition B has not yet been placed).

However, if the following command was specified such that the pin in1 was not in the original selection list

`assignPtnPin -ptn A -pin {out*}`)

then it is not guaranteed that pin in1 will be at the same location (that would be best for it, considering the location constraint on its corresponding pin (out1 of partition B)), because any of the selected {out\*} pin can now occupy that location.

## Prioritizing Few Pins in a Selected Pin Assignment Flow

You can combine both the methods described above to select pins and then set priority among those selected pins.

For example, you can use any of the following use models to select the same set of pins - `{ in1 in2 in3 in4 in5} {out1 out2 out3 out4 out5}`. These selected pins may not have priority among them.

### Model 1

```
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}  
assignPtnPin -ptn A -pin {out1 out2 out3 out4 out5}
```

### Model 2

```
assignPtnPin -pin_file a.in.list  
assignPtnPin -pin_file a.out.list
```

where, the pin files contain

a.in.list	a.out.list
Partition: A	Partition: A
in1	out1
in2	out2
in3	out3
in4	out4
in5	out5

In both these models, Innovus does not choose to favor certain pins over others. However, if you want that among the list of selected pins, priorities should be set, then you can put *netWeight* on the nets of chosen pins.

For example, after selecting pins using the above commands, you may specify:

```
specifyNetWeight net5 6
assignPtnPin -pin_file a.in.list
```

Now pin in5 of net net5 will be considered first among the other in\* pins out of all the pins of partition A.

## Speeding Up Interactive Pin Assignment

When you use the `editPin` batch command or the Pin Editor in the interactive pin assignment flow, internal data structures are created to work on a given solution space. When you give a sequence of commands for pin assignment, the solution space (floorplan objects that impact pin assignment) is not changed. But each of the specified commands creates these data structures individually, to speed up the interactive pin assignment flow, you can choose to reuse these data structures.

To enable pin-editing in batch mode, you can use the `setPinAssignMode -pinEditInBatch true` command. In the batch mode, multiple `editPin` commands reuse internal data structures for all pins without recreating them for each individual command.

For example:

```
setPinAssignMode -pinEditInBatch true
editPin -pin {P1} -layer 2 -assign X1 Y1
editPin -cell {A} -pin {P2} -layer 3 -assign X2 Y2
```

```

editPin -cell {A} -pin {P3} -layer 4 -assign X3 Y3
editPin -cell {B} -pin {P4} -layer 4 -assign X4 Y4
editPin -cell {B} -pin {data*} -layer 4 -spreadType start -start 0 0 -side r
editPin -pin {data*} -layer 4 -spreadType start -start 0 0 -side r
editPin -cell {A} -pin {data*} -layer 4 -spreadType start -start 0 0 -side r
...
...
...
setPinAssignMode -pinEditInBatch false

```

**Note:** Once the interactive pin assignment is complete, it is important to set the mode to false. This can be done using the `setPinAssignMode -pinEditInBatch false` command.

**Note:** Floorplan changes (involving but not limited to pin/routing blockages, pin guides, pre routes, macro placement, size/shape of fences/design) are not allowed and should not be carried out during the batch mode.

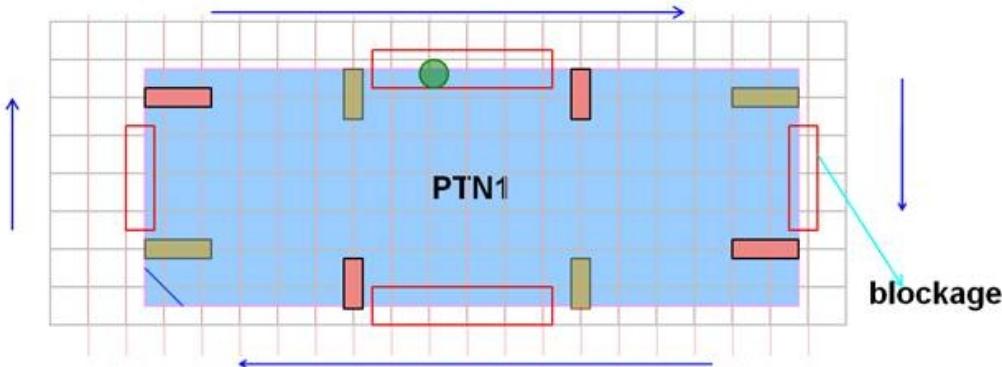
## Deciding the Closest Legal Location to a Selected Position

To assign a pin on a selected position on a partition on a layer's top side, you use the following command

```
editPin -cell cellName -pin pinName -assign {x y} -layer layerID -side top
```

For example, the following command assigns pin A of partition Ptn1 at xy position on the top side with layer 2.

```
editPin -cell PTN1 -pin A -assign X Y -layer 2 -side top
```



However, if the selected position is not a legal position, as shown by the green dot in the figure above, then by default the pin will go to the right. This is shown by the red pin in the figure.

The default spread direction is clockwise for all edges.

- Top: Left to Right
- Bottom: Right to Left

**Note:** In the above figure, all red pins depict the chosen location (always clockwise for all edges) to keep the pin in case the provided location is not feasible. All yellow pins depict the location to keep the pin in counter clock wise direction to provided unfeasible location

In order to get the pins in the opposite direction, you can use the `editPin -spreadDirection {counterclockwise}` parameter. For example,

```
editPin -cell PTN1 -pin A -assign X Y -layer 2 -side top -spreadDirection {counterclockwise}
```

You can also use the `-spreadDirection {both}` parameter to automatically choose the closest legal location in both directions. This enables the `editPin` command to decide the closest legal location to desired `x y` location in both direction and place the pin there.

In the figure shown above, the pin will be placed on the left side (yellow pin), which is 1 tracks away (rather than going to right side (red pin), which is 3 tracks away from the green dot (desired `x y`), on the top edge.

[Example 1 Simple case](#) Refer files for example 1 (3 files)

[Example 2 Multifanout case](#) Refer files for example 2 (3 files)

[Example 3 Reuse for simple case of example 1](#) Refer files for example 3 (3 files)

[Example 4 Reuse case for multi-fanout case of Example 2](#) Refer files for example 4 (3 files)

[Example 5 Complex case for loopback net](#) Refer files for example 5 (3 files) [Example 6 Complex case for multi-fanout net](#) Refer files for example 6 (3 files)

## Timing Budgeting

- [Overview](#)
- [Is My Design Ready for Budgeting?](#)
- [Deriving Timing Budgets](#)
  - [Budgeting Using the GUI](#)
  - [Budgeting Using Text Commands](#)
  - [Top-Level Budgets Derived by Using Active Logic View](#)

- Deriving Preliminary Budgets in Early Design Phase
- Calculating Timing Budgets
- Master Clone Budgeting
- Constraints Adjustment
- Analyzing Timing Budgets
  - Resolving Conflicts with Path-Based Exceptions
  - Budgeting Clock Latency in Propagated Mode
- Budgeting Libraries
  - Resolving Conflicts with Path-based Exceptions
  - Defining Clocks Inside the Partition
- Customizing Budget Generation
- Fixing Budget
  - Recommendations for Fixing Budget
  - Fix Budget Example
- Modifying Budgets
- Reading the Justify Budget Report
  - Design Example
  - SDC Constraints for Design Example
  - Generated Report for Design Example
  - Generate Summarized Report of Budget Data
- Reading the Justify Exception Report
  - Design Example
  - SDC Constraints for Design Example
  - Generated Report for Design Example
- Support for Distributed Processing in Budgeting
- Constraints Support in Budgeting
- Warning Report
  - Pin Constraint Values Greater than Available Time
  - Warning Report Example

## Overview

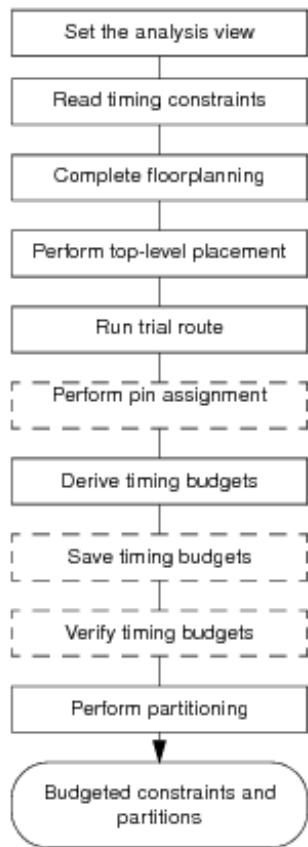
In hierarchical design flows, chip-level timing constraints must be mapped correctly to corresponding block-level constraints. The Innovus™ Implementation System (Innovus) software does this automatically to produce predictable timing convergence.

The software apportions budgets to blocks using a path-based method, which might not have a direct relationship to the size of the blocks themselves. Innovus supports two ways to perform timing budgeting in hierarchical designs:

- Without Trial IPO  
Timing optimization is not run before generating budgets at the port boundaries.
- With Boundary Trial IPO (the default)  
Timing optimization fixes are done for the top level nets and the interface nets.

MMMC mode is active if you have specified `set_analysis_view`. In the MMC mode, timing budgeting is run per-view, and generates view files for partition implementation and top-level implementation (not chip assembly).

The following flowchart shows how timing budgeting is performed within the overall design flow.



**Note:** You can set the analysis view at any time before performing timing analysis, but should be done before reading the constraint file.

## Is My Design Ready for Budgeting?

In order to close timing on the hierarchical level, you must be able to close timing on the flat design first. If the fully flat placement of a design (based on the partition fences, pin placement, and so on) does not meet timing before partitions are committed, then it is unlikely that timing will close after the partitions are committed and the budgets generated.

To help you decide whether the design is ready for budgeting, run virtual IPO on the flat design. This builds a timing graph, which allows you to examine the design for failing inter-partition paths. When you find these paths, you can use the information to determine why the problems occur and how you can fix them. In a hierarchical implementation, you might need to iterate top-level floorplanning and virtual IPO several times before creating a floorplan that can meet timing.

When you are convinced that the timing will close at this stage of the design process, you can then run timing budgeting.

## Deriving Timing Budgets

You can generate timing budget constraint files for each top-level partition using either the partition graphical user interface (GUI) or the various text commands.

### Budgeting Using the GUI

To generate the constraints files using the GUI, complete the following steps:

1. Read in the pin assigned data having timing constraint file during design import.
2. Derive timing budgets. Select *Derive Timing Budget* from the *Partition* menu, and specify the options you need on the Derive Timing Budget form.
3. (Optional) Save timing budgets. Select *Design - Save - Save Timing Budget*.
4. Partition the design. Select *Partition - Partition*.
5. Save the *Partition* data. Select *Design - Save - Partition* to save the top and partitions data in a directory

This directory has sub-directories *topCellName* for the top-level and *partitionName* for the partitions.

For each partition, this procedure creates a timing constraint file named *partitionName.constr.pt*. These files are located in each partition directory. The library model files, *partitionName.lib*, are created in the top-level directory. The constraints file *top\_level.top.constr* is created for the top level.

The result is budgeted constraints and partitions.

### Budgeting Using Text Commands

To generate the constraints files using the text commands, complete the following steps:

1. Load pin assigned database having timing data, using source `pin_assign.enc` command.
2. Derive the timing budgets using the `deriveTimingBudget` command .
3. (Optional) Save the derived budgets using the `saveTimingBudget` command.
4. Partition the design using `Partition` command.
5. save the partition data using `savePartition -dir dir_name`" command.

The result is budgeted constraints and partitions.

## Top-Level Budgets Derived by Using Active Logic View

The software provides a top-level interface timing analysis flow to perform partitioning and budgeting on a trimmed-down version of the timing graph: a virtual partition. This flow saves memory usage and provides faster run time on large designs. The results of the flow are comparable to results of partitioning and budgeting with the full timing graph.

To perform partitioning and budgeting using top-level timing analysis, complete the following steps:

1. Load the hierarchical design, having timing data and pin assigned, into the database.  
`source init.enc`
2. Use the `createActiveLogicView` command to identify interface logic for all the partitions.
3. `createActiveLogicView -type flatTop`

This command marks the interface logic in all the partitions in the chip-level design. When you build the timing graph after this step, the software masks the core logic inside the partitions and ignores it to reduce the run time and memory usage.

4. Derive timing budgets.

`deriveTimingBudget`

The command rebuilds the timing graph as part of its operation.

5. (Optional) Evaluate the budgeting results.

`justifyBudget`

6. Save the timing budgets.

`saveTimingBudget`

7. Clear the virtual partition marking after you save the timing budgets.

`clearActiveLogicView`

## Deriving Preliminary Budgets in Early Design Phase

The software provides a flow for deriving timing budgets in the early stages of the design to obtain a preliminary estimate of the budgets. It uses the net delay and net load that you specify using the `setBudgetingMode` command. To perform the preliminary budgeting, you use the `-preliminary` parameter of the `deriveTimingBudget` command. The software does not use trialIPO operations for calculating budgets for this flow.

The software uses the top-level net delays that you specify using the `-topLevelDelayPerLen` and `-topLevelMinDelayPerNet` parameters of the `setBudgetingMode` command. It calculates the total delay value using the value of the `-topLevelDelayPerLen` parameter and the total length of the net. If the total delay value that the software calculates is less than the `-topLevelMinDelayPerNet` parameter, it uses the value of the `-topLevelMinDelayPerNet` parameter. Otherwise it uses the value of the `-topLevelDelayPerLen` parameter. It assumes the block-level delays are zero.

The software uses the lump capacitance that you specify using the `setBudgetingMode -overrideNetCap` command for both top-level and block-level nets. It uses this value to calculate the cell delay.

It assumes all the net delay is on the top-level and does not perform boundary net adjustments. Therefore, when you run the `justifyBudget` command after generating the preliminary budgets, the `Adjustment by Net Delay` value is zero in the budgeting report.

The software proportions the budget according to the calculated values. If you do not use the `-preliminary` parameter in the `deriveTimingBudget` command, it adds three extra buffer delays to the delay of the positively slacked path that has positive slack. If you use the `-preliminary` parameter, the software distributes the positive slack equally between source block, destination block, and top-level. Therefore, the value of following fields in the budgeting report is zero:

- Virtual buffering adjustment
- External buffering adjustment

For negative slack, the software uses the `-freezeTopLevelNegPathOnly` parameter of the `setBudgetingMode` command.

To perform preliminary budgeting, complete the following steps:

1. Load the hierarchical design, having timing data and pin assigned, in the database.  
`source init.ENC`
2. Set the delay values to be used during budgeting.

```
setBudgetingMode -topLevelDelayPerLen value  
                  -topLevelMinDelayPerNet value  
                  -overrideNetCap value
```

3. Derive timing budgets.

```
deriveTimingBudget -preliminary
```

4. Verify the budgets.

```
justifyBudget
```

5. Save the budgets

```
saveTimingBudget -dir dirName
```

## Calculating Timing Budgets

Innovus proportions timing budget for partitions based on the path segment length, with a slight difference in calculation when the slack on a path is positive or negative.

For paths with negative slack, the proportioning formula for a setup check (max budgeting) is:  
 $SD/TD * AT = BB(\text{neg})$

For paths with negative slack, the proportioning formula for a hold check (min budgeting) is:  
 $SD/TD * (AT + HT) = BB(\text{neg})$

**Note:** If  $AT + HT$  is less than zero, the software does not use the proportioned value. The software uses the timing analysis values for input or output delays.

For paths with positive slack, the proportioning formula for a setup check (max budgeting) is:  
 $SD + SD/TD * PS = BB(\text{pos})$

For paths with positive slack, the proportioning formula for a hold check (min budgeting) is:  
 $SD - SD/TD * PS = BB(\text{pos})$

where:

- $SD$  is the delay through a path segment.
- $TD$  is the total delay of the path.
- $AT$  is the total available time. This could be the number of clock periods for multicycle paths, or the clock period minus the fixed delays.
- $HT$  is the hold time.

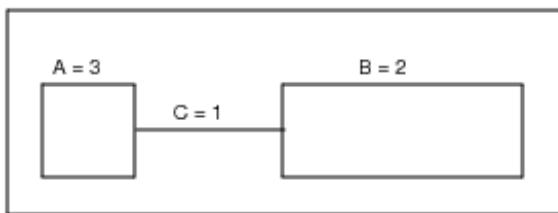
**Note:** For max budgeting, hold time is not same as setup time. Setup time is represented as

an extra delay for the path. On the other hand, hold time is equivalent to required time, that is the amount of time a signal cannot change.

- $BB$  is the baseline budget
- $PS$  is the positive slack

**Note:** For a positively slacked path, budgeting adds virtual buffer delays to the path. The software usually adds three virtual buffer delays. In case of abutted designs, budgeting adds two virtual buffer delays. In case of feedthrough paths, budgeting distributes three buffer delay through all segments of the path.

### Example 26-1 Negatively Slacked Path



In this example, block A is connected to block B via top-level net C. The budget of the top-level net is not fixed. When placed and routed, the path segment through block A needs 3 ns, path segment through block B needs 2 ns, and net C requires 1 ns. The available time to be budgeted is 5 ns.

The software calculates the following values:

Budget for block A =  $3/6 * 5 = 2.5$  ns

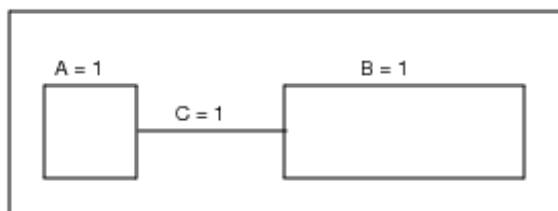
Budget for block B =  $2/6 * 5 = 1.67$  ns

Budget for net C =  $1/6 * 5 = 0.83$  ns

Output delay at A = Budget for block B + Budget for net C = 2.5

Input delay at B = Budgets for blocks A + Budget for net C = 3.33

### Example 26-2 Positively Slacked Path



In this example, the path segment through blocks A and B, and net C require 1 ns each. The total delay is 3 ns. The total available budget is 5 ns. Therefore, positive slack is 2 ns.

The software calculates the following budget values:

Budget for A, B, and C = 1 + 1/3 \* 2 = 1.66 ns

## Master Clone Budgeting

Master-clone partitioning/budgeting involves a trade-off between design implementation time and optimized timing for the complete design. When a master is replicated as multiple clones, a single partition is created that represents the worst of all the partitions for each pin and clock. Master-clone budgeting provides you with the flexibility to generate a single unified timing budget constraint file and timing model (.lib) for repeated partitioned modules referred to as master and clones. The data checked for includes:

- **Physical characteristics** - The following physical characteristics are compared for each interface pin across all partitions: drive resistance, internal capacitance, and external capacitance. The merged physical data includes the electrical data that contributes to the timing of every pin of the instance. When the merged data is stored, it stores the largest data for each pin. For example, drive resistance of the master for a pin = largest of the drive resistances for the pin in instances.
- **Timing characteristics** - The following timing characteristics are compared for each interface pin across all partitions: network latencies, user delay in each pin, and slack in each pin for each clock arriving at it based on input delay and output delay. The merged timing characteristics require comparison of the slack at a pin due to each clock arriving at it across instances and saving the worst among them. For example, the clock `C1k` has a slack `s1` in `Inst1` and `s2` in `Inst2`. If `s1>s2`, then the data in `Inst1` is taken for that pin for that clock.
- **Clock data** - All the data in each pin is merged per clock. Since different clocks can arrive at different instances, the clocks in the overall master are merged. This is done by looking at the clock list on each instance and adding the ones not present in the overall master to it.

The following command derives a timing budget for hierarchical partitions based on the worst-case data per-pin for the master/clone set containing the partitions:

```
setBudgetingMode -mergeClones {true|false}
```

In master clone budgeting, the software takes `set_input_delay` (SID) or `set_output_delay` (SOD) for a port of a repeated partition in such a way that the budget inside the partition is the smallest. This feature prevents underconstraining of some of the full chip paths going through the repeated partitions.

The software computes `set_input_delay` and `set_output_delay` for various feedthrough and non-feedthrough paths during master clone budgeting using the following methods:

- **Non-feedthrough path:** Give min budget inside the partition, that is, choose max of `set_input_delay` and `set_output_delay`.

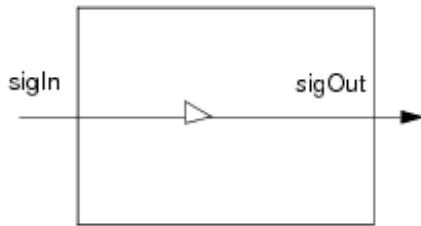
```
max(SIDm, SIDc1, SIDc2, ....),  
max(SODm, SODc1, SODc2, ....)
```

For example, the input delay value of 2.0 is stored for the master and the value of 3.0 and 4.0 is stored for the two clones, respectively:

```
set_input_delay -max -clock CLK1 2.0 { IN1 }  
set_input_delay -max -clock CLK1 3.0 { IN1 }  
set_input_delay -max -clock CLK1 4.0 { IN1 }
```

Choose the max of `set_input_delay` 4.

- **Pure feedthrough path:** Choose budgets from the master/clone having the most critical path. `set_input_delay` and `set_output_delay` will come from the same instance.



For example, consider there are two instances of the same cell, where `M` is the master and `C` is the clone and a clock cycle of 6ns. In this case, instance based budgeting creates the following constraints:

For `M`:

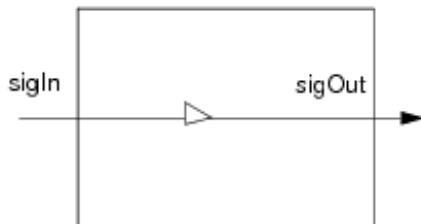
```
set_input_delay 3 sigIn  
set_output_delay 1 sigOut
```

For `C`:

```
set_input_delay 2 sigIn  
set_output_delay 3 sigOut
```

You will choose budgets from the master/clone whose SID+SOD is the biggest, that is, choose the clone (c).

- **Pure feedthrough but two paths having the same normalized slack:** Take both `set_input_delay` and `set_output_delay` from the same instance such that the budget inside the instance is min.



For example, consider there are two instances of the same cell, where M is the master and C is the clone and a clock cycle of 6ns. In this case, instance based budgeting creates the following constraints:

For M:

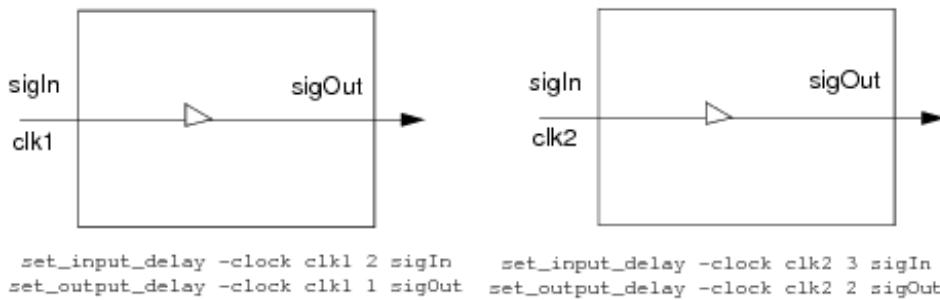
```
set_input_delay 3 sigIn
set_output_delay 2 sigOut
```

For C:

```
set_input_delay 1 sigIn
set_output_delay 4 sigOut
```

In this case, both feedthrough paths have the same normalized slack. You should ensure that both SID and SOD come from the same instance, that is, either SID = 3, SOD=2 or SID=1, SOD=4.

- **Pure feedthrough with paths starting/ending with different clocks:** In this case, different clocks in the master/clone cannot be merged. As a result, invalid clock paths come in the design. For example, a path from `sigIn` with `clk1` ending at `sigOut` with `clk2`.



To eliminate this problem, use false paths. The following example illustrates how SDC is being generated by using `set_false_path`:

```

set_input_delay -clock clk1 2 sigIn
set_input_delay -clock clk2 3 sigIn
set_output_delay -clock clk1 1 sigOut
set_output_delay -clock clk2 2 sigOut
set_false_path -from clk1 -through sigIn -through sigOut -to clk2
set_false_path -from clk2 -through sigIn -through sigOut -to clk1

```

## Constraints Adjustment

The timing budgeting process produces a `.lib` file for a partition that will be used during top-level implementation, and a SDC file for partition implementation. When top-level and block-level implementations are run in parallel, the timing model and the SDC files must match in order for chip assembly to succeed.

To ensure that the files match, timing budgeting makes adjustments for the following constraints:

- Capacitance

For each partition input pin, the tool produces the following output:

- In the `.lib` file, a specification of the pin's capacitance.
- In the SDC constraint file, a `set_max_capacitance` constraint.

If a `max_capacitance` constraint in the SDC file is greater than the capacitance specified in the `.lib` file, this could lead to timing violations during the reassembly. The partition optimization might change the load of a partition input pin to a value such that the buffer, chosen at the top level with respect to the small capacitance specified in the `.lib` file,

would not be able to drive the load.

The correlation adjustment done by budgeting ensures that the `pin_capacitance` specification in the `.lib` file and the `set_max_capacitance` constraint in the partition SDC to be very nearly the same.

- Transition

For each partition output pin, the tool produces the following output:

- In the `.lib` file, a lookup table describing the pin transitions with respect to load.
- In the SDC constraint file, a `set_max_transition` constraint.

If the `.lib` lookup table indicates a range of transition values that are all less than the `set_max_transition` value used to constrain partition implementation, it could be possible for you to perform top-level implementation assuming that the transition will be, for example, 500 ps, while the partition implementation can pass with a transition of 1 ns on the same port. This situation could result in problems after reassembly.

The correlation adjustment done by budgeting ensures that the `set_max_transition` constraint in the partition SDC is within the lookup table in the partition `.lib`.

- Load and `max_cap`

For each partition output pin, the tool produces the following output:

- In the `.lib` file, a `max_capacitance` DRV for the pin.
- In the SDC constraint file, a `set_load` constraint.

A `max_capacitance` constraint in the `.lib` greater than the `set_load` constraint in the SDC can lead to timing violations during reassembly. The top-level optimization might change the load of the partition output pin to an unrealistic value for the buffer implemented within the partition, and chosen with respect to the small `set_load` constraint.

The correlation adjustment done by budgeting ensures that the `max_capacitance` in the `.lib` file and the `set_load` constraint in the partition SDC file are nearly the same.

## Analyzing Timing Budgets

To analyze timing budgets, you must first identify all the boundary pins of the partitions. For each partition pin, the software generates timing constraints in the form of timing `set_input_delay` or `set_output_delay` if the pin is an input or output pin, respectively. The software divides the total available budget among all partitions involved, where their boundary pins constitute part of the path.

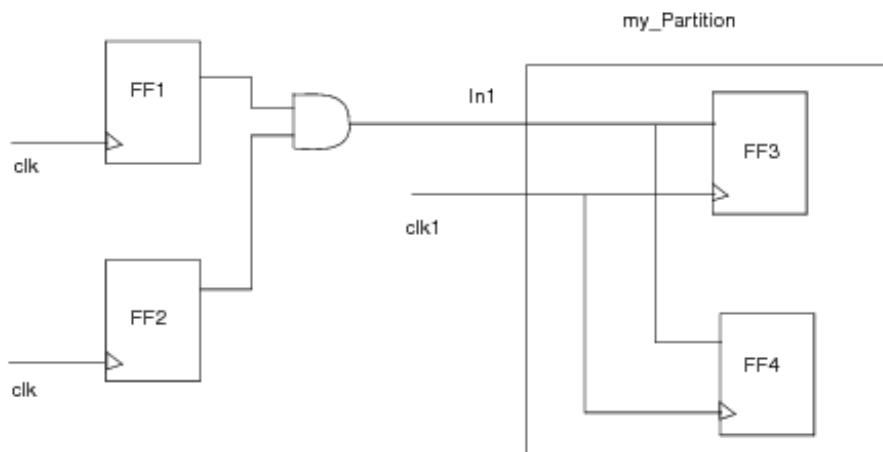
A pin might have multiple paths going through it. Multiple paths through the same port are handled by CTE budgeting. In case of multiple paths related to the same clocks and the same number of clock cycles, the tool automatically chooses the best path for deriving the budgets.

## Resolving Conflicts with Path-Based Exceptions

Budgeting generates one input or output delay assertion for each group of paths controlled by the same group of path-based exceptions. For `set_input_delay` generation at a partition port, the path group is the union of paths originating from the same clock phase that is controlled by the same group of exceptions at the partition port. For `set_output_delay` generation, the path group is the union of paths with the same clock phase at the end point that is controlled by the same set of exceptions traversing backward at the partition port.

## Examples

All of the following examples use the same design:



## Case 1

ONE virtual clock ( along with one original clock) will be created for each same group of path-based exceptions during budgeting.

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF3/D
```

- For single cycles path and `clk`:

```
set_input_delay -clock clk numberln1
```

(based on single path through `ln1`)

- For multicycle path and `clk`:

```
set_input_delay -clock clk_v0 numberln1
```

(based on worst path from `FF1`)

```
set_multicycle_path 2 -from clk_v0 -through ln1 -setup -to FF3/D
```

## Case 2

Two virtual clocks are created for each same group of path-based exceptions.

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF4/D
```

```
set_false_path -from FF1 -to my_partition/FF3/D
```

```
set_multicycle_path 2 -setup -from FF2
```

- For multicycle paths from `FF1`:

```
set_input_delay -clock clk_v0 numberln1
```

(based on path from `FF1` to `my_partition`)

```
set_multicycle_path 2 -setup -from clk_v0 -through ln1 -to FF4/D
```

- For false path from `FF1`:

```
set_false_path -from clk1_v0 -through ln1 -to FF3/D
```

- For multicycle path from `FF2`:

```
set_input_delay -clock clk_v0 numberln1
```

(based on worst path from FF2)

```
set_multicycle_path 2 -from clk_v0 -through in1 -setup
```

## Case 3

Two virtual clocks are created.

- Chip-level exceptions:

```
set_multicycle_path 3 -setup -from FF1 -to my_partition/FF3/D
```

```
set_multicycle_path 2 -setup -from FF2 -to my_partition/FF4/D
```

- For multicycle 3 path and clk:

```
set_input_delay -clock clk_v0 number In1
```

(based on worst of paths from FF1)

```
set_multicycle_path 3 -from clk_v0 -through in1 -setup -to FF3/D
```

- For multicycle 2 path and clk:

```
set_input_delay -clock clk_v1 number In1
```

(Based on worst path form FF2)

```
set_multicycle_path 2 -from clk_v1 -through in1 -setup -to FF3/D
```

## Case 4

Two virtual clocks are created:

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF4/D
```

```
set_false_path -from FF1 -to my_partition/FF3/D
```

```
set_multicycle_path 2 -setup -from FF2
```

- For multicycle 2 path from FF1:

```
set_input_delay -clock clk1_v0 number In1
```

(based on path from FF1 to my\_partition/FF4)

```
set_multicycle_path 2 -setup -from clk1_v0 -through In1 -to FF4/D
```

- For false path from FF1:

```
set_false_path -from clk1_v0 -through in1 -to FF3/D
```

- For multicycle 2 path from FF2:

```
set_input_delay -clock clk2_v0 number in1
```

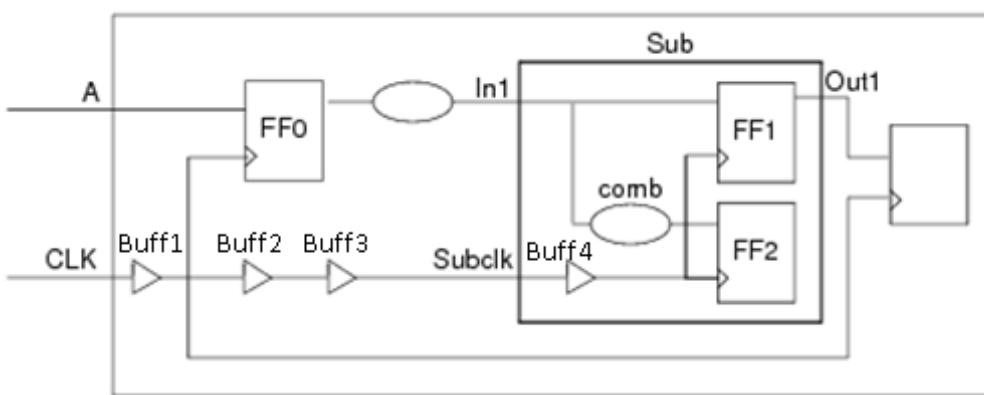
(based on worst of paths from FF2)

```
set_multicycle_path 2 -from clk2_v0 -through in1 -setup
```

## Budgeting Clock Latency in Propagated Mode

Innovus includes clock latency in the constraints generated for clocks in propagated mode. The clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The clock latency is added to `set_input_delay` and subtracted from the `set_output_delay`. This feature is useful when a clock tree is present in your design.

For multiple paths, both source and propagated clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The software adds the `-source_latency_included` and `-network_latency_included` constraints in the `set_input_delay` and `set_output_delay` constraints for all inputs and outputs related to clocks in propagated mode. Consider the following figure.



The `deriveTimingBudget` command result in the following constraints for the `Sub` partition:

```
create_clock -clock subclk -waveform...
set_clock_latency -source (top_source + delay Buff1 + delay Buff2 + delay Buff3 )
subclk
set_input_delay -clock subclk (Input delay (with skew) + top_source + delay Buff1)
```

```

-source_latency_included -network_latency_included In1
set_output_delay -clock subclk (output_delay (with skew) - top_source - delay Buff1)
-source_latency_included -network_latency_included Out1

```

Where,

*top\_source* = source latency of the clock at the top level  
*delay Buff\** = delay through each buffer in the clock network

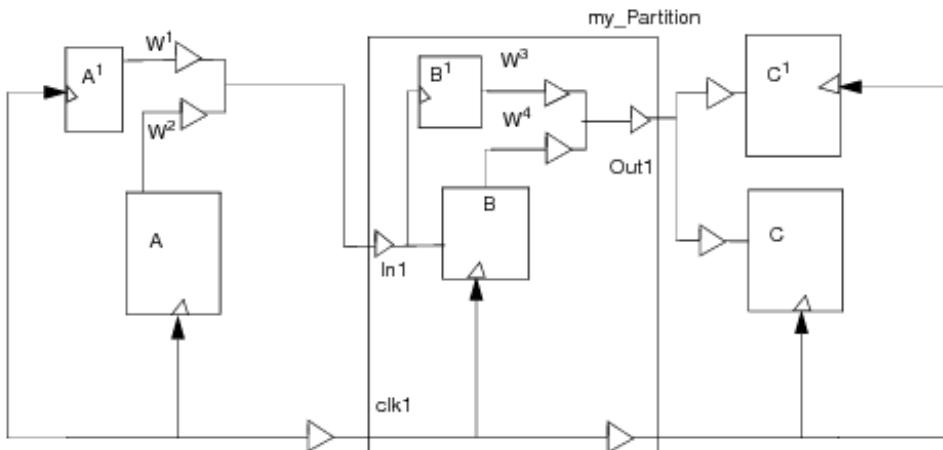
## Budgeting Libraries

To enable design analysis at the top level, budgeting generates black box models of the partitions in the .lib format.

## Resolving Conflicts with Path-based Exceptions

Budgeting generates internal pins and combinational arcs to model the budgets of each group of paths controlled by the same group of path-based exceptions. For input ports, the path group is the union of paths with the same clock phase at the end point that is controlled by the same set of exceptions traversing backwards towards the input port. For output ports, the path group is the union of paths originating from the same clock phase that is controlled by the same group of exceptions at the partition port.

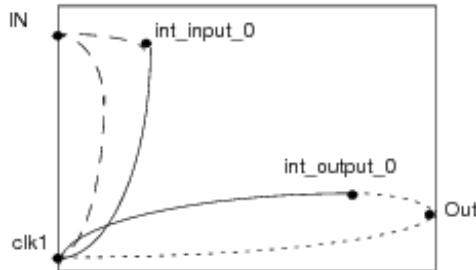
## Case 1: A single cycle path and a multicycle path through the partition port



- Chip-level exceptions:

```
set_multicycle_path 2 -from A1/ck -to my_Partition/B/D  
set_multicycle_path 2 -from my_Partition/B/ck -to C1/ck
```

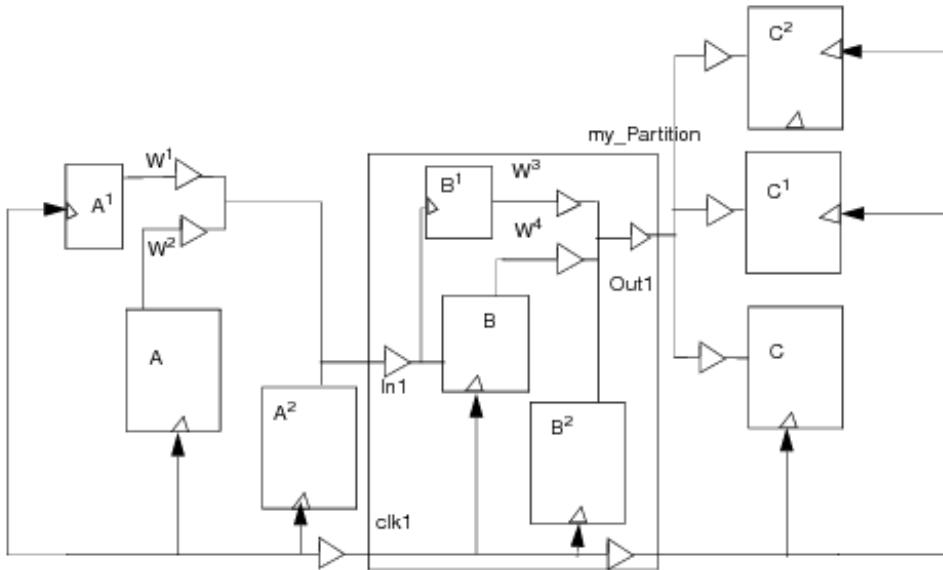
MCP delays are modelled on the combinational arcs between input and output ports, and the internal pins. In the following .lib representation, MCP delays are modelled on arcs IN -> int\_input\_0 and int\_output\_0 -> Out.



- Top-level exceptions:

```
set_multicycle_path 2 -setup -from [list [get_pins {A1/CK}]] -through [list  
[get_pins {partition/in}]] -to [list [get_pins {partition/int_input_0}]]  
set_multicycle_path 2 -setup -through [list [get_pins {partition/int_output_0}]]  
-through [list [get_pins {partition/out}]] -to [list [get_pins {C1/D}]]
```

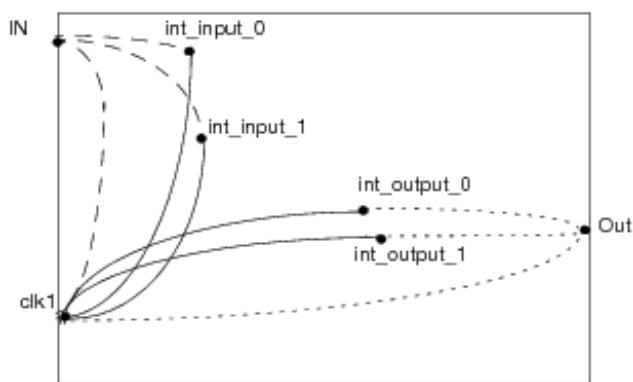
## Case 2: A single cycle path and two different multicycle paths through the partition port



- Chip-level exceptions:

```
set_multicycle_path 2 -from [get_pins {A1/CK}] -to [get_pins {my_partition/B/D}]
set_multicycle_path 4 -from [get_pins {A/CK}] -to [get_pins {my_partition/B1/D}]
set_multicycle_path 2 -from [get_pins {my_partition/B/CK}] -to [get_pins {C1/D}]
set_multicycle_path 4 -from [get_pins {my_partition/B1/CK}] -to [get_pins {C/D}]
```

In this case because of two different MCP constraints, budgeting generates two internal pins to model the effect of multicycle paths, and a normal arc is modelling the effect of the single cycle path, at both input and output ports.



- Top-level exceptions:

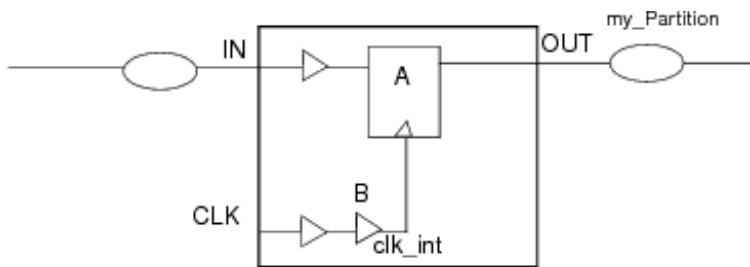
```

set_multicycle_path 2 -setup -from [list [get_pins {A1 /CK}]] - through [list
[get_pins {my_partition/in}]] -to [list [get_pins {my_partition/int_input_0}]]
set_multicycle_path 2 -setup -through [list [get_pins {my_partition/int_output_1}]]
] -through [list [get_pins {my_partition/out}]] -to [list [get_pins {C1 /D}]]
set_multicycle_path 4 -setup -from [list [get_pins {A/CK}]] - through [list
[get_pins {my_partition/in}]] -to [list [get_pins {my_partition/int_input_1}]]
set_multicycle_path 4 -setup -through [list [get_pins {my_partition/int_output_0}]]
] -through [list [get_pins {my_partition/out}]] -to [list [get_pins {C/D}]]

```

## Defining Clocks Inside the Partition

Budgeting generates additional internal pins to model the effect of the clocks defined inside the partition at the top level.

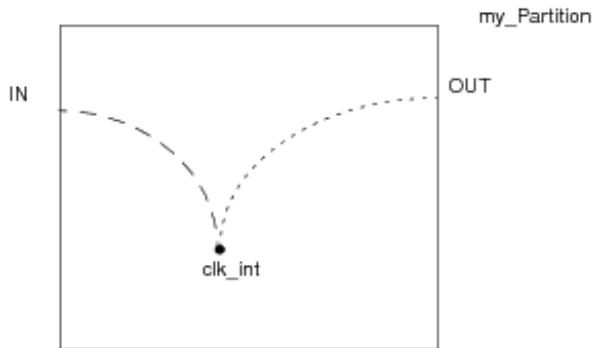


- Chip-level constraint:

```
create_clock -name clk -period 10 [get_pins {my_Partition/B/clk_int}]
```

For the above constraint, an internal pin is created in the partition. At the top level, a constraint is generated for this internal pin.

In the following .lib representation, the B/clk\_int pin is created, and sequential and check arcs are defined with the internal pin.



- Top-level constraint:

```
create_clock -name clk -period 10 [get_pins {my_Partition/B/clk_int}]
```

## Customizing Budget Generation

You can customize budget generation according to the design stage and timing requirements. To customize budget generation, use the following commands in Innovus:

- The `-freezeTopLevel` parameter of the `setBudgetingMode` command fixes the top-level timing budget and proportions the timing budget for the partitions. The commands consider blocks that are not being budgeted as fixed. If the top-level design has no buffers or glue logic, using the `-freezeTopLevel` parameter might not make much difference in the generated budgets.
- The `setBudgetingMode [-ignoreDontTouch | -noIgnoreDontTouch]` command is used to consider `don't_touch` blocks. The `-noIgnoreDontTouch` parameter considers `don't_touch` as fixed delay. The `-ignoreDontTouch` parameter does not consider `don't_touch` as fixed delay. The budgeting results change based on whether fixed delay is considered during trial IPO.
- At the top level, you can set the `set_input_delay` and `set_output_delay` constraints on the hierarchical ports (or partition ports). The software generates budgets for the hierarchical ports based on the set constraints.

## Fixing Budget

You can fix budget for a particular path segment between partitions and start/end points only for the following segments:

- Chip In -> Partition In
- Start Point -> Partition In
- Partition In -> Partition Out
- Partition In -> End Point (FF D Pin) inside same partition
- Start Point (FF Q Pin) inside same partition -> Partition Out
- Partition Out -> Chip Out
- Partition Out -> End Point

To fix budget for a path segment, use the `setFixedBudget` command in Innovus.

## Recommendations for Fixing Budget

Following are the recommendations for fixing budget:

- To perfectly control the delay being distributed in RAK design, “`-virtualOptEngine {none}`” is used.
  - This avoid changing of delays inside a partition based on virtual buffers inserted by virtual optimization engine is called by `deriveTimingBudget` internally.
- Examples: `setFixedBudget -from P5/in -to P5/out -slack 0`
  - Delay of this segment in RAK design is =0.284
  - With virtual optimization more buffers can get added and delay values may change, hence changing the budgets for each of the partitions (we can't accurately predict the number of buffers and hence the delay numbers are unknown before virtual optimization is done).
- Example: `setFixedBudget -from P5/in -to P5/out -delay .284`
  - This can fix the delay of partition `ptn_fd` and override the effects of optimization
  - However results may be inaccurate as paths are not optimized for delays/DRVs (partition delay value can be retrieved by running `timeDesign -preCTS`)
- For channel based design (top portion of net has some delays), top segments are also considered for slack distribution. `setFixedBudget` should be applied to any segment whose

slack distribution needs to be controlled (including top segments discussed in the statement)

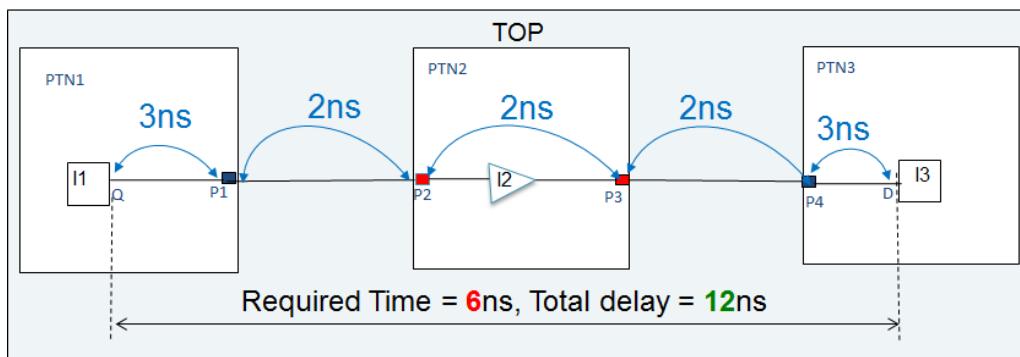
- Set “-bufferDelayAdjustment 0” to prevent `deriveTimingBudget` from automatically adding buffer delays to partition for whom we are fixing the budgets (may lead to unexpected results).

## Fix Budget Example

- Generated budget constraints with required time is 6ns and total path delay is 12ns.

Budget = required time \* (segment delay/total delay)

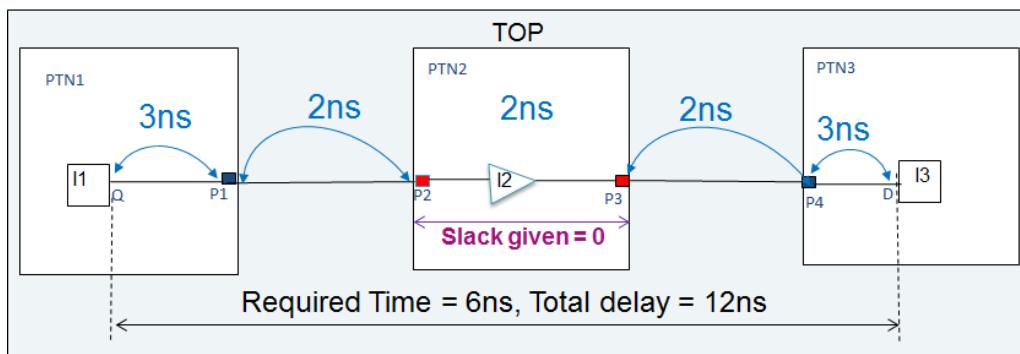
$6*(3/12)=1.5\text{ns}$	$6*(2/12)=1\text{ns}$	$6*(2/12)=1\text{ns}$	$6*(2/12)=1\text{ns}$	$6*(3/12)=1.5\text{ns}$
-------------------------	-----------------------	-----------------------	-----------------------	-------------------------



- setFixedBudget –from PTN2/P2 –to PTN2/P3 –slack 0

Budget = required time \* (segment delay/total delay)

$4*(3/10)=1.2\text{ns}$	$4*(2/10)=.8\text{ns}$	FIXED 2ns	$4*(2/10)=.8\text{ns}$	$4*(3/10)=1.2\text{ns}$
-------------------------	------------------------	-----------	------------------------	-------------------------



- No slack will be distributed to segment 3, delay will remain equal to segment delay = 2ns
- For other segments
  - New required time is  $6-2=4\text{ns}$

- New total delay is 12-2=10ns

## Modifying Budgets

You can modify budgets after they have been saved using the `saveTimingBudget` command. To modify budgets, you must create an input budget file containing constraints that look like an SDC file. It only supports the `set_input_delay` and `set_output_delay` constraints, as shown below:

```
set_input_delay 1.5 -clock [get_clocks {clk1_setuphold}] -max -fall [get_ports {sub_in}] -add_delay
set_input_delay 1.9 -clock [get_clocks {clk1_setuphold}] -max -rise [get_ports {sub_in}] -add_delay
```

The clock names in these constraints are same as the ones in the generated budget file. You can provide a bus name instead of a pin name to apply the same budget to all pins of the bus.

To use `modifyBudget`, you must specify the following command before `deriveTimingBudget`:

```
setBudgetingMode -enableReportBudget true
```

A sample Tcl script is given below:

```
setBudgetingMode -enableReportBudget true
deriveTimingBudget -noTrialIPO -justify
saveTimingBudget -dir Budget
reportBudget -pin "*" SUB -file rpt_original -view default_analysis_view_setup
modifyBudget -file modify.tcl -ptn SUB -view default_analysis_view_setup
saveTimingBudget -dir MOD
```

In this example, `modify.tcl` is the input budget file containing the constraints.

When you change the budgets on a port using the `modifyBudget` command, the `justify` budget report displays the modified budget and the justification for that (user applied or derived from user applied), thereby ensuring accuracy.

For manual budgeting, the budget of the port can change in the following scenarios:

- When you directly apply a constraint on the port, the `justify` report shows the location of the file from which the constraint has been taken, the applied constraint, and the port on which it was applied.

Partition: block

Port: sub\_in

Budgeted constraint type: set\_input\_delay(setup rise)

Virtual Clock: clk1\_setuphold

**Budget Applied by modify budget statement (*../../../../modify.tcl:3*) :**

```
set_input_delay 1.9 -clock [get_clocks {clk1_setuphold}] -max -rise [get_ports {sub_in}] -add_delay
```

**Applied constraint = 1.900**

**Start clock: clk1 clock edge: rise**

**End clock: clk1 clock edge: rise**

- When you apply a constraint on a connected port, the budget of that port also changes. The output port will be impacted due to modification at the input port as it is one continuous path. Budgets are modified at the output port by adjusting the required time of the port with the extra delta delay given to (or taken away from) the connected port. Therefore, the modifications at the input port is reflected in the justify report for both the input port and the output port, wherein, the budgets are being recalculated for the output port based on the modification. The justify report has information about the port, the delta and how that delta affected the constraint value at the given port.

**Budget Impacted by modify budget statement (*location of the file in which the applied constraint is specified*) :**

```
set_input_delay 1.9 -clock [get_clocks {clk1_setuphold}] -max -rise [get_ports {sub_in}] -add_delay
```

**Impact of modify budget(modDelta) = 0.911**

**Available budget after adjustment(AvailTime)=(10.000 - 2.982) - (0.911) = 6.107**

## Reading the Justify Budget Report

You use the `deriveTimingBudget -justify` command to generate a budget report per view containing the debug data to justify the timing budget for a pin. For a negatively slacked path, the software distributes the total available time (in a simple clock period case) proportionally between ports of instances along the path. For a positively slacked path, the software usually adds some buffer delays to the generated delay values (built in positive slack).

The report generated contains the following fields:

- Adjustment for budget available time

Derived as follows:

Path Fixed Delay + Fixed Delay For Feedthrough Blocks - Clock Skew + Value of Constraint for the Receiving Register (HoldTime)

Where, Fixed Delay For Feedthrough Blocks is the two buffer delay distributed between all feedthrough blocks.

- Fixed Delay on the Path (pathFixDel)

Specifies the delay that cannot be modified. The fixed delay adjustment includes: `set_input_delay`, `set_output_delay`, all cell delays for the cells marked as `dont_touch` if `-ignoredonttouch` is not used, delays of top level segments if `-freezeTopLevel` is used, any snapped delays calculated by using `setBudgetingMode -snapFdBudgetTo` or `-snapInputBudgetTo` or `-snapOutputBudgetRatio`. If, during timing analysis, the path segment delay used to generate a budgeting constraint for the port falls below specified threshold value the delay segment is snapped to the specified value and is considered as fixed delay during budget allocation.

- Virtual clock adjustment

Specifies a special adjustment to map the virtual clock into clocks pertaining to partitions. This number is generated when you use the `saveTimingBudget -noVirtualClock` command.

- Top Level Adjustment

Specifies the top-level delay value. The top-level delay value cannot be less than the minimum percentage of total available budget specified using the `-topLevel` parameter of the `setBudgetingMode` command.

- RC Adjustment (RC)

Specifies the input delays. During timing analysis the input delays are adjusted by the delay due to input port drive cell that was added by budgeting as a `set_drive` command in the generated constraint file. The `Adjustment by RC` number is subtracted from the delay value in budgeting so that this effect is not counted twice in the budget.

- Adjustment by clock latency

Specifies the clock latency of the driving object.

- Total Delay (totDel)

Specifies the total path delay.

- Initial Slack

Initial Slack = (Data Required Time - fixed delay) - (Path segment number1 delay + Path segment number 2 delay).

- Virtual Buffering Adjustment

Specifies the total extra delays added to the positive slacked path. This number is usually three extra buffer delays. In case of abutted designs, the number is two extra buffer delays.

**Note:** In case of feedthrough paths, three buffer delay is distributed through all segments of the path.

- Slack after Virtual Buffering Adjustment (slack)

The software takes out three buffers worth of delay from positive slack to safeguard minimum partition budget. This adjustment is used only for positive slacks.

- External Buffering Adjustment

Specifies the extra delay that is external to partition port. This is usually equivalent to two buffer delays. This is part of the virtual buffering adjustment. This delay is added to the input delay for the input ports and output delay for the output ports.

- Budgeted constraint

Budget = Adjustment for budget available time \* Delay for path outside the partition / Absolute total delay + Adjustment by fixed delay + Adjustment by virtual clock + Adjustment by clock latency - Adjustment by RC + External Buffering Adjustment

- External segment delay

Delay of the path segment outside of the partition.

- This block's segment delay

Delay of the path segment inside of the partition.

- Fixed delay through feedthrough

Amount of extra delay allocated to the path feedthrough segments.

- External Segment Fixed Delay from Budget Snap

The fixed delay for the path segment external to the partition contributed by using setBudgetingMode - snapFdBudgetTo or - snapInputBudgetTo or - snapOutputBudgetRatio.

- Total External Segment Fixed Delay  
Fixed delay of the path segment outside of the partition.
- External Segment Extra Delay From Budget Snap  
The extra delay added to the external path segment when you use `setBudgetingMode -snapOutputBudgetRatio` and `-snapInputBudgetRatio` and if external segment path delay is below user defined threshold.
- Clock Skew  
The path clock skew.

**Note:** The report precision (the number of digits printed after the decimal point) is 3.

## Design Example

```
module Top(in1, clk1, clk2, out);
    input in1;
    input clk1;
    output out;
    input clk2;
    wire c0, c1, c2;

    bfx0 buf0(.A(in1), .Z(c0));
    SUB      i_sub1(.sub_in(c0),
    .sub_clk(clk1),
    .sub_out(c1));

    bfx0 buf1(.A(c1), .Z(c2));

    SUBn      i_sub2(.sub_in(c2),
    .sub_clk(clk2),
    .sub_out(out));

endmodule // TOP

module SUB (sub_in, sub_clk, sub_out);
    output sub_out;
    input sub_in;
    input sub_clk;
    df1qx1 sub_FF (.D(sub_in), .CP(sub_clk), .Q(sub_out));
endmodule // SUB
```

```
module SUBn (sub_in, sub_clk, sub_out);  
output sub_out;  
input sub_in;  
input sub_clk;  
df1qx1 sub_FF (.D(sub_in), .CP(sub_clk), .Q(sub_out));  
endmodule // SUBn
```

## SDC Constraints for Design Example

```
current_design Top  
create_clock -name clk1 -period 1 -waveform {0 0.5} [get_ports {clk1}]  
set_input_delay 0.2 -clock clk1 [get_ports {in1}]  
set_multicycle_path 2 -from [get_pins {i_sub1/sub_FF/CP}] -to [get_pins  
{i_sub2/sub_FF/D}]  
  
create_clock -name clk2 -period 1 -waveform {0 0.5} [get_ports {clk2}]  
set_output_delay 0.1 -clock clk2 [get_ports {out}]
```

## Generated Report for Design Example

To validate the budgets with positive slack in the design example, "[Design Example](#)", type the following command:

```
justifyBudget -inst i_sub2 -pin sub_in
```

The following report was generated:

```
HInstance: i_sub2  
Port: sub_in  
Budgeted constraint type: set_input_delay(setup rise)  
Virtual Clock: clk1_V0  
Initial budget available time + clock skew = 2.000  
  
One Buffer Delay for Adjustment(cell bfx2): 0.198  
Fixed Delay for Feedthrough Paths(fixFdThru)= 0.000  
External Segment Fixed Delay From Budget Snap(snapExtFixedDel) = 0.000  
Total External Segment Fixed Delay(extFixDel) = 0.000  
This Block's Segment Fixed Delay from budget snap(snapIntFixedDel) = 0.000  
Total This Block's Segment Fixed Delay(intFixDel) = 0.000  
External Segment Extra Delay From Budget Snap (snapExtDelExtra) = 0.000
```

This Block's Extra Delay From Budget Snap (snapIntDelExtra) = 0.000  
Path Extra Delay From Budget Snap (snapExtraDel) = (0.000 + 0.000) = 0.000  
Fixed Delay on the Path(pathFixDel) = (0.000 + 0.000 + 0.000 + 0.000) = 0.000  
Fixed Delay Adjustment(fixDel)= 0.000  
Clock Skew(clkSkew) : 0.000

Adjustment for budget available time= -(pathFixDel + fixFdThru - clkSkew + snapExtraDel)  
= -(0.000 + 0.000 - 0.000 + 0.000) = -0.000  
Available budget after adjustments(AvailTime)= (2.000 - 0.000) = 2.000

External Segment Delay(extSegDel): 0.638  
This Block's Segment Delay(segDel): 0.273  
Total delay(totDel): 0.638 + 0.273 = 0.910  
Initial Slack = AvailTime - totDel  
Initial Slack = 2.000 - 0.910 = 1.090  
Virtual Buffering Adjustment: (3 x 0.198) = 0.594  
Slack after Virtual Buffering Adjustment(slack): 1.090 - 0.594 = 0.496

External Virtual Buffering Adjustment(extVirBuf)= 0.396  
Top Level Adjustment(topLev): 0.000  
Virtual Clock Adjustment(virClk): 0.000  
RC Adjustment(RC): 0.009  
Budgeted constraint = extSegDel + slack \* extSegDel / totDel + extVirBuf + topLev + fixDel + virClk + startClkLat - RC  
Budgeted constraint = 0.638 + 0.496 \* 0.638 / 0.910 + 0.396 + 0.000 + 0.000 + 0.000 + 0.000 - 0.009 = 1.372

Path 1: MET Setup Check with Pin i\_sub2/sub\_FF/CP  
Endpoint: i\_sub2/sub\_FF/D (^) checked with rising edge of 'clk2'  
Beginpoint: i\_sub1/sub\_FF/Q (^) triggered by rising edge of 'clk1'

<b>Other End Arrival Time</b>	0.000
- Setup	0.269
+ Phase Shift	1.000
+ Cycle Adjustment	1.000
= Required Time	1.731
- Arrival Time	0.641
= Slack Time	1.090
Clock Rise Edge	0.000
= Beginpoint Arrival Time	0.000

Instance	Arc	Cell	Delay Time	Arrival Time	Required	Slew	Load	Instance Location
<hr/>								
	clk1 ^			0.000	1.090	0.000	0.007	
i_sub1/sub_FF	CP ^ -> Q ^	df1qx1	0.182	0.182	1.272	0.063	0.005	(43.12, 366.80)
buf1	A ^ -> Z ^	bfx0	0.455	0.637	1.727	1.003	0.040	(43.96, 398.16)
i_sub2/sub_FF	D ^	df1qx1	0.004	0.641	1.731	1.003	0.040	(43.12, 766.64)
<hr/>								

**Note:** The `justifyBudget` command honors the `report_timing_format` global, which allows you to customize the timing report according to the user-specified columns. You must set this global before specifying `justifyBudget` or `deriveTimingBudget -justify` to get the report in the desired format.

## Generate Summarized Report of Budget Data

To generate a script friendly summary of what budgets have been produced for a specific port or ports, you can use the `reportBudget` command. For reporting the budgeting data for buses, specify the bus name as the pin name.

The use model of the `reportBudget` command is given below:

1. Enable the collection of reporting data - Before running `deriveTimingBudget`, run the following command:

```
setBudgetingMode -enableReportBudget true
```

If this parameter is not specified, the `reportBudget` command does not produce any results.

2. Report budgets - This command does the actual reporting, and should be issued after the budget has been saved by using the `saveTimingBudget` or `savePartition` command.

```
reportBudget
```

The output of the `reportBudget` command is stored in the output file in the following format for pins:

Pin: Port Name

SDC Data

Clock: Clock Name

SDC Source Budget(Rise,Fall): rise value, fall value

SDC Destination Budget(Rise,Fall) : rise value, fall value  
Exception(s) : shows the exceptions written for the port in the resultant SDC file, if any  
SDC Warnings: shows the budgeted warnings for the port, if any  
Justify Source Budget(Rise, Fall): rise value, fall value  
Justify Destination Budgets(Rise,Fall): rise value, fall value  
Justify Slack (Rise, Fall) : rise value, fall value  
Library Data  
Reference Pin: the other end pin name  
Delay Rise : rise delay value  
Delay Fall : fall delay value

The above block is repeated for various pins which are requested through the command (or for all pins , if \* is specified).

For input pins:

- Source Budget is equal to the constraint (SID)
- Destination Budget is equal to the budget given to the partition

For output pins:

- Source Budget is equal to the budget given to the partition
- Destination Budget is equal to the constraint (SOD)

Bus reporting goes through all bus bits and collects the minimum, maximum, median and average data. The "constraint" on the output port is defined as the output delay on that port, which corresponds to the Destination Budget of the port. Hence, more the destination budget, more is the output port constrained.

The output of the reportBudget command is stored in the output file in the following format for bus pins:

Bus: Bus Name

Most Constrained Bit : bit name  
SDC Data  
Clock: Clock Name  
SDC Source Budget(Rise,Fall): rise value, fall value  
SDC Destination Budget(Rise,Fall) : rise value, fall value  
Exception(s) : shows the exceptions, if any  
SDC Warnings: shows the warnings if any  
Justify Source Budget(Rise, Fall): rise value, fall value  
Justify Destination Budgets(Rise,Fall): rise value, fall value

```
Justify S lack (Rise, Fall) : rise value, fall value
Least Constrained Bit: bit name
SDC Data
Clock: Clock Name
SDC Source Budget(Rise,Fall): rise value, fall value
SDC Destination Budget(Rise,Fall) : rise value, fall value
Exception(s) : shows the exceptions, if any
SDC Warnings: shows the warnings if any
Justify Source Budget(Rise, Fall): rise value, fall value
Justify Destination Budgets(Rise,Fall): rise value, fall value
Justify S lack (Rise, Fall) : rise value, fall value

Median Constrained Bit: bit name
SDC Data
Clock: Clock Name
SDC Source Budget(Rise,Fall): rise value, fall value
SDC Destination Budget(Rise,Fall) : rise value, fall value
Exception(s) : shows the exceptions, if any
SDC Warnings: shows the warnings if any
Justify Source Budget(Rise, Fall): rise value, fall value
Justify Destination Budgets(Rise,Fall): rise value, fall value
Justify Slack (Rise, Fall) : rise value, fall value

Average Source Budget (rise,fall in ns) : average budget across all bus bits(rise and
fall)
Average Destination Budget (rise,fall in ns): average budget across all bus
bits(rise and fall)
```

## Reading the Justify Exception Report

You can use the `justifyException` command to provide a debugging or justification mechanism of exceptions on ports. This command allows you to debug exception constraints and justify all exceptions for the specified instances or partitions of the design.

To justify exceptions:

1. Set the `-enableJustifyException` parameter of the `setBudgetingMode` command to `true` to save the justify exception data during the `deriveTimingBudget` process.

## 2. Perform exception justification using one of the following two ways:

- Specify the `deriveTimingBudget` command with the `-justify` parameter.

When the `-justify` parameter is specified, the exception file names are generated as

`<partition|inst>_<view name>.justifyExcp`. If the same view is used as both setup and hold, then the file names are generated as `<partition|inst>_<view name>_<setup|hold>.justifyExcp`. These files are saved in their respective partition/inst directory in the `budget_justify` directory by default, if the `setBudgetingMode -justifyBudgetDir` parameter is not specified.

**⚠** You can run the `deriveTimingBudget` command both with and without the `-justify` parameter. When the `-justify` parameter is not specified, the data is collected only for exception justification, and will neither be written in a file nor will be displayed on screen.

- Specify the `justifyException` command to justify all exceptions on pins specified by `-pins` or `-pin`. If you do not specify an output file using the `-outfile` parameter, the exception data will be displayed on the screen.

### Flow example with `deriveTimingBudget -justify`

```
restoreDesign 120.des.dat top
createFence partition 1 2 3 4
definePartition -hinst partition
setBudgetingMode -enableJustifyException true -justifyBudgetDir budget_justify_with_dtb
deriveTimingBudget -ptn {partition} -justify
```

### Flow example without `-justify` but with the `justifyException` command

```
restoreDesign 120.des.dat top
createFence partition 1 2 3 4
definePartition -hinst partition
setBudgetingMode -enableJustifyException true -justifyBudgetDir
budget_justify_ptn_with_command
deriveTimingBudget -ptn {partition}
justifyException partition -pin out1 -view default_analysis_view_setup # this will only print
data on screen
justifyException partition -pin out1 -view default_analysis_view_setup -outfile
block_default_analysis_view_setup.jusifyExcp #this will print data in the specified file
```

The report generated contains the following fields:

- Partition  
Specifies the partition or instance for which the exception justification is done.
- Port  
Specifies the partition/instance's port for which the exceptions have been justified.
- Budgeted constraint type  
Specifies the delay type applicable on this port. It can be either `set_input_delay` or `set_output_delay`. It also provides the information of setup/hold check for rising/falling edge of clock.
- Virtual Clock  
Virtual clock with which setup/hold check is done.
- Exception  
Specifies the exception type applied on the port. It can be one of the following:
  - `set_multicycle_path`
  - `set_max_delay`
  - `set_min_delay`

**Note:** `set_false_path` is not reported because only constrained paths are justified.
- Applied exceptions  
Specifies the path exception information picked up from the Common Timing Engine. It is the same information that is printed in the path exception section of the timing report when "`report_timing -path_exception all`" is issued on the path that has been used for budgeting of the port.
- Translated exception  
Specifies the full chip exceptions translated to partition/instance.

## Design Example

```
module partition (
    in,
    clk,
    out,
    out1);

    input in;
    input clk;
    output out;
    output out1;

    BUFX12 B21 (.A(clk), .Y(o2));
    BUFX12 B22 (.A(o2), .Y(o4));
    BUFX12 B23 (.A(o4), .Y(o5));
    DFFHQX1 D21 (.D(in), .CK(o5), .Q(ox));
    BUFX12 BMC (.A(ox), .Y(out1));
    BUFX12 BMD (.A(ox), .Y(out1));
endmodule

module top (
    in,
    clk,
    out);

    input in;
    input clk;
    output out;

    BUFX12 B1 (.A(in), .Y(s1));
    BUFX12 B2 (.A(clk), .Y(s2));

    partition partition (.in(s1), .clk(s2), .out(s3), .out1(s4));
    BUFX12 B3 (.A(s4), .Y(out));
endmodule
```

## SDC Constraints for Design Example

```
current_design top
create_clock -name clock -period 15 [get_ports {clk}]
```

```
set_input_delay 2 -clock clock in
set_output_delay 3 -clock clock out

set_multicycle_path 2 -through partition/BMC/Y
set_multicycle_path 3 -through partition/BMD/Y
```

## Generated Report for Design Example

Partition: partition

Port: partition/out1

Budgeted constraint type: set\_output\_delay(setup rise)

Exception: set\_multicycle\_path

Applied exceptions:

Through	Late	View Name
partition/BMC/Y	cycles 2	default_analysis_view_setup

Translated exception:

```
set_multicycle_path 2 -setup -through [list \
[get_ports {out1}]] -to [list \
[get_clocks {clock}]]
```

Translated exception:

```
set_multicycle_path 3 -setup -through [list \
[get_ports {out1}]] -to [list \
[get_clocks {clock}]]
```

Partition: partition

Port: partition/out1

Budgeted constraint type: set\_output\_delay(setup fall)

Exception: set\_multicycle\_path

Applied exceptions:

Through	Late	View Name
partition/BMC/Y	cycles 2	default_analysis_view_setup

Translated exception:

```
set_multicycle_path 2 -setup -through [list \
[get_ports {out1}]] -to [list \
[get_clocks {clock}]]
```

Translated exception:

```
set_multicycle_path 3 -setup -through [list \
[get_ports {out1}]] -to [list \
[get_clocks {clock}]]
```

## Support for Distributed Processing in Budgeting

Innovus supports distributed time budgeting for MMMC designs to improve the overall run-time for large designs. In the distributed mode, processing of views are distributed across CPUs of a local machine or multiple machines, and the view data is collated using the `saveTimingBudget` command.

Distributed processing supports two modes:

- **Local Mode** - you can specify the number of CPUs to use on local machine. In this mode, you can distribute views across multiple CPUs of the same machine.

```
setDistributeHost -local
```

```
setMultiCpuUsage -localCpu integer
```

- **Remote Mode** - you can specify one or more CPUs to use on network hosts. In this mode, you can distribute views to multiple machines.

```
setDistributeHost -rsh -add {host1 host2 host3}
```

```
setMultiCpuUsage -remoteHost integer
```

The use model for distributed MMMC budgeting is as follows:

1. Set up the multi-host environment. Specify the `setMultiCpuUsage` command to set the maximum number of hosts and the `setDistributeHost` command to set the multiple-CPU processing configuration for distributed processing.
2. Specify the `deriveTimingBudget` command. This command internally creates distribution scripts for all views and runs these scripts on different CPUs/machines per the multi-host environment. The logs and distribution scripts for different views are located in the `.tbMMMCdistributed` directory for later reviews.
3. Specify the `saveTimingBudget` command. This command internally collates the data for multiple views from different runs and saves it to the specified directory.

**Note:** All parameters of the `saveTimingBudget` command, except `-dir`, are ignored. You can set these parameters using the `setBudgetingMode` command.

## Example

```
setMultiCpuUsage -remoteHost 4
setDistributeHost -lsf -queue lnx64 -resource "select [mem>10000 && tmp>400 && swp >1000
&& OSNAME==Linux && SFIARCH==OPT64]
deriveTimingBudget -noTrialIPO -inst i_top_0
saveTimingBudget -dir Budget -pt
```

## Constraints Support in Budgeting

- `group_path`

This constraint is supported in timing optimization and timing analysis. In budgeting, it is not pushed-down inside the partition and top-level SDC. This will affect timing budgets, because the constraint affects chip-level timing analysis.

- `create_clock`

If a top-level clock `CK` is inverted, then while generating the budgets for a partition a new negative clock `CK_B_ENC` is created for the partitions connected to the negative clock.

For example, if `CK` is defined as:

```
create_clock -name CK -period 7.500 -waveform { 0.000 3.750 } \
[list [get_ports {clk}]] ]
```

The negative clock is:

```
create_clock -name CK_B_ENC -period 7.500 -waveform { 3.750 7.500 } \
[list [get_ports {losdclk0_rp}]] ]
```

Where, `losdclk0_rp` is the clock port of the partition.

- `create_generated_clock`
- `set_clock_latency`

The `set_clock_latency` constraint is generated when you use the

`setAnalysisMode -skew true` command. The clock latency is not budgeted between the partitions. The `setAnalysisMode -clockPropagation sdcControl`, along with `set_clock_propagation` constraint, do not cause the network delay through the clock tree to be budgeted for the partitions. The same clock latency is assigned to all the partitions if specified in the top-level clock constraints.

- `set_clock_uncertainty`
- `set_input_delay`
- `set_output_delay`
- `set_input_transition`
- `set_load`
- `set_drive`
- `set_driving_cell`
- `set_max_transition`
- `set_max_capacitance`
- `set_multicycle_path`
- `set_false_path`

Innovus timing analysis requires that the `set_false_path` and `set_multicycle_path` constraints have valid startpoints and endpoints for the `-from` and `-to` options.

Valid startpoints are:

- - Input ports
  - Input part of bidirectional ports
  - Clock pins of sequential cells
  - Pins associated with `set_input_delay`
  - Pins associated with `set_path_delay -from`

Valid endpoints are:

- Output ports
- Output part of bidirectional ports
- Data pins of sequential cells
- Pins associated with `set_output_delay`
- Pins associated with `set_max_delay -to`

During budgeting, the software generates valid budgets for partitions based on invalid constraints at the top. For example, if `set_multicycle_path 2 -from SUB/IN1` is set at the top level, it is ignored during timing analysis, because a hierarchical pin is not a valid startpoint for `set_multicycle_path` constraint. However budgeting generates `set_multicycle_path -from IN1` for partition which is valid when the constraints are sourced for the partition because `IN1` is a top-level port for partition and a valid start point.

- `set_case_analysis`
- `set_max_delay`
- `set_min_delay`
- `set_logic_zero`
- `set_logic_one`

Partition ports could be left unconstrained, which means that there are some ports missing `set_input_delay` or `set_output_delay` constraints in the constraint file. Several factors can cause a partition I/O being unconstrained. For instance, `set_false_path`, `set_case_analysis`, `set_disable_timing` in an input constraint file can effectively cut paths through a port. The `Set_input_delay` constraint at the top-level, without a reference clock is another possibility which can cause some partition ports being unconstrained. Missing timing arcs in cell timing model can also cut timing paths. If a constant signal (`1'b0`, `1'b1`) is assigned to a net leading to a partition port in Verilog®, the constant signal can also cause that port to be left unconstrained.

- Min and `-hold`

The following commands are supported:

- `Set_clock_latency -min`
- `Set_clock_transition -min`

- Set\_clock\_uncertainty -hold
  - Set\_drive -min
  - Set\_driving\_cell -min
  - Set\_input\_delay -min
  - Set\_output\_delay -min
  - Set\_false\_path -hold
  - Set\_load -min
  - Set\_min\_delay
  - Set\_multicycle\_path -hold
- set\_timing\_derate

This constraint is pushed down to a separate file with the extension `.nonsdc.constr` in the push down directory.

## Warning Report

The `saveTimingBudget -ptn` command generates a warning report (`partition_or_instance.warn`) for each partition and stores these reports in the partition subdirectories.

## Pin Constraint Values Greater than Available Time

The `.warn` report contains a section entitled "Pin constraint values greater than available time."

The software checks whether generated `input_delay/output_delay` budgets are less than a maximum allowed time in the clock period for the delay. The maximum allowed time is defined as a delay between active edge of the starting clock and the sampling edge of the sampling clock. This time may vary based on phase shift and multicycle path directives. If `deriveTimingBudget tsConsCheck` is specified, budget checking will use more conservative value for available time:

The tool subtracts `clk2q` delays from available time when checking `output_delay` statements and setup time when checking `input_delay` statements.

The following command reports all partition ports that have slack less than <value> in  
*partition\_dir/partition\_name/partition\_name/constr.warn:*

```
savePartition/saveTimingBudget -rptNegSlackOnPorts value
```

For example:

```
*.warn file:  
....  
/* Start Section: Instance ports with slack < 0.020 */  
/* End Section: Instance ports with slack < 0.020 */
```

## Warning Report Example

The warning report format is as follows:

```
*****  
* Timing constraint sanity check File  
*****  
  
/* Start Section: Pin constraint values greater than available time */  
/* End Section: Pin constraint values greater than available time */  
  
/* Start Section: Ports without constraints */  
Port: sub1_out1 (setup)  
Port: sub1_out2 (setup)  
Port: sub1_out3 (setup)  
Port: sub1_out4 (setup)  
Port: sub1_out1 (hold)  
Port: sub1_out2 (hold)  
Port: sub1_out3 (hold)  
Port: sub1_out4 (hold)  
  
/* End Section: Missing constraints due to constant signals at ports */  
  
/* Start Section: Missing constraints due to false path assignments at ports */  
/* End Section: Missing constraints due to false path assignments at ports */  
  
/* Start Section: List of ports w/o constraints with added false_path assertions */  
Port: sub1_out1 (setup)  
Port: sub1_out2 (setup)  
Port: sub1_out3 (setup)  
Port: sub1_out4 (setup)  
Port: sub1_out1 (hold)  
Port: sub1_out2 (hold)  
Port: sub1_out3 (hold)  
Port: sub1_out4 (hold)  
  
/* End Section: List of ports w/o constraints with added false_path assertions */  
  
/* Start Section: Toplevel constraints applied to ports */  
/* End Section: Toplevel constraints applied to ports */  
  
/* Start Section Unconnected Ports */  
  
/* End Section Unconnected Ports */  
  
/* Start Section: Missing constraints due to constant signals at ports */  
Port sub1_in1 set to logical ZERO, 6 inversions  
Port sub1_in2 set to logical ONE, 0 inversions  
Port sub1_out1 set to logical ONE, 9 inversions  
Port sub1_out2 set to logical ZERO, 3 inversions  
/* End Section: Missing constraints due to constant signals at ports */
```



# Using ART in Hierarchical Designs

- [Overview](#)
- [Types of Active Logic Views](#)
  - [Flat Top](#)
  - [Critical](#)
- [Creating an Active Logic View](#)
- [The flexILM PreCTS Closure Flow](#)

## Overview

Active-logic Reduction Technology (ART) is a technique that is used to activate certain portion of a logic in a design and masking the other logic, while maintaining full physical design database in memory. In ART, an active logic view contains only the active portion of the logic.

ART can be applied to any timing-related command, such as timing budgeting or timing optimization to reduce run time and memory usage. In timing operations, an active logic view contains only the set of timing paths exposed to the specific operation. When applied to timing optimization, active logic views enable cross-hierarchical optimization while preserving the full hierarchical view of the design after optimization is complete.

## Types of Active Logic Views

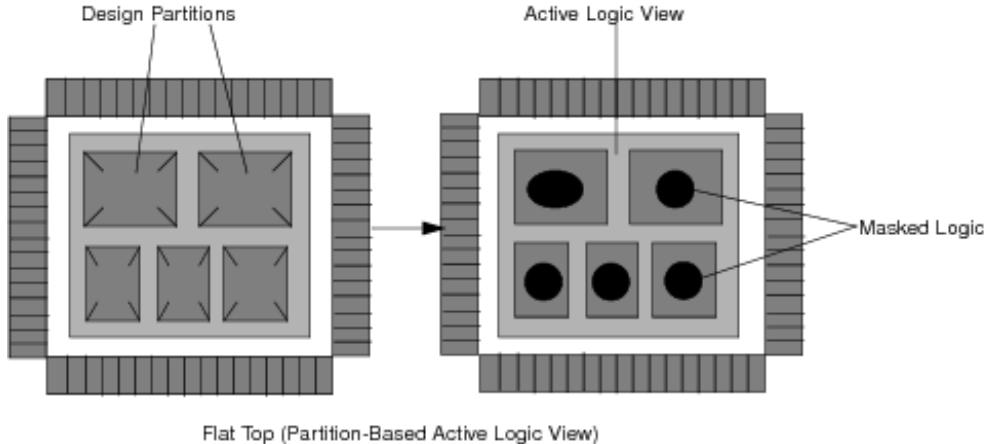
The tool creates an active logic view based on the partition boundaries, set of critical timing paths, block module boundaries, and physical area. There are two types of active logic views:

- [Flat Top](#)
- [Critical](#)

## Flat Top

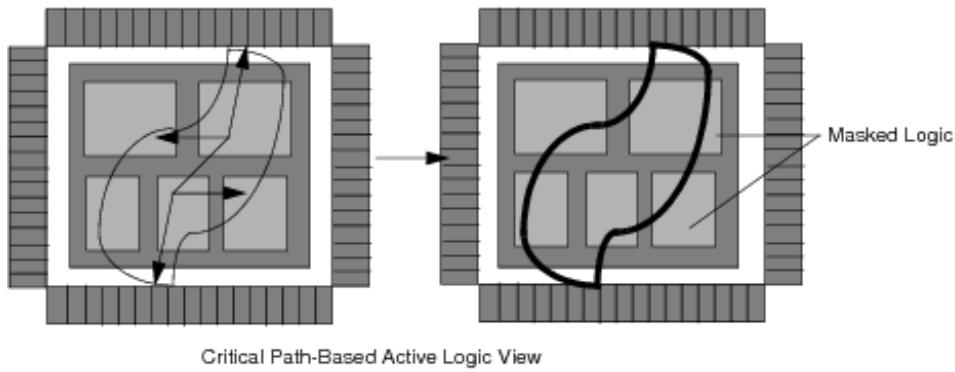
A flat top is a partition-based active logic view that activates the top-level paths and the interface path of partition blocks. The logic inside the partition blocks is excluded from the timing database.

The following figure shows the flat top active logic view:



## Critical

The critical active logic view activates all paths in a design that have a negative slack. All other logics in the design is masked.



## Creating an Active Logic View

To create an active logic view, load the entire chip as a design in the Innovus software, specify the partition, and then run the `createActiveLogicView` command with an appropriate option.

- ⓘ An active logic view cannot be saved as a database or a file. Run the `createActiveLogicView` command to create an active logic view.

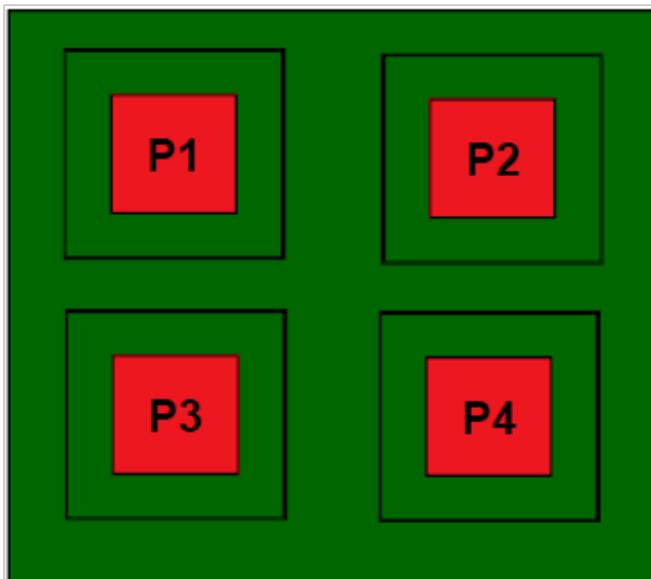
**Note:** The Innovus software considers MMMC settings while creating an active logic view.

## Example of Active Logic View Creation

To create an active logic view, run the following commands:

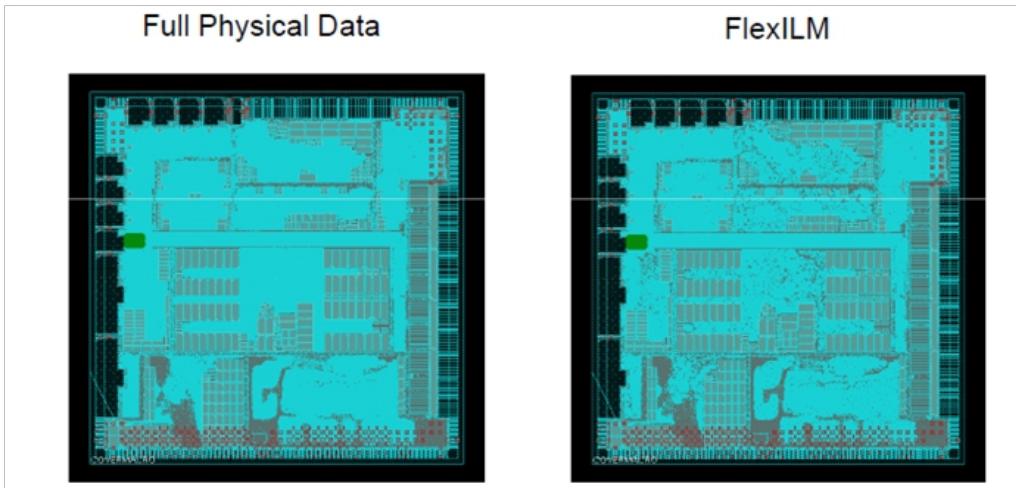
```
assembleDesign -topDir top.enc.dat -blockDir block1.enc.dat  
createActiveLogicView -type flatTop
```

## The flexILM PreCTS Closure Flow



FlexILM creates reduced Netlist by trimming logic on internal path and keeping logic on interface path, and saves Netlist and DEF. In terms of Netlist and Placement ECO, FlexILM commits change on interface paths to original block data via the ECO process. It also contributes to memory reduction in timing graph and physical data.

## Preliminary Results on Large Design: Floorplan



## Preliminary Results on Large Design: Memory and TAT

Testcase: 12 Partitions						
Flow	FlexILM		ART		Pure Flat	
Instances	1.2M		7.6M		7.6M	
Index	CPU	Peak Mem	CPU	Peak Mem	CPU	Peak Mem
Data	9:44:24	25G	16:01:39	47G	52:03:01	72G

Note: All flows achieved timing closure.

For detail about flexILM flow, you can refer to the [Top-level Timing Closure Methodologies](#) chapter in the *Innovus User Guide*.

# Top-level Timing Closure Methodologies

- [Using Interface Logic Models \(ILM\)](#)
  - [Overview](#)
  - [General ILM Flow](#)
  - [Creating ILMs](#)
  - [Specifying ILM Directories at the Top Level](#)
  - [ILMs Supported in MMMC Analysis](#)
  - [ILMs Supported in SI](#)
  - [ILM Model](#)
  - [Interactive Use of ILMs](#)
  - [Handling Interactive Constraints](#)
  - [Using Flexible Interface Logic Models \(FlexILM\)](#)
  - [General FlexILM Flow](#)
  - [FlexILM Model Creation](#)
  - [Top-Level Optimization](#)
  - [Full Chip Assembly](#)
  - [FlexILM Model Data](#)

## Using Interface Logic Models (ILM)

Instead of using a blackbox at the top level, you create an ILM at the block level and use it as you would use a blackbox.

The advantages of using ILMs are as follows:

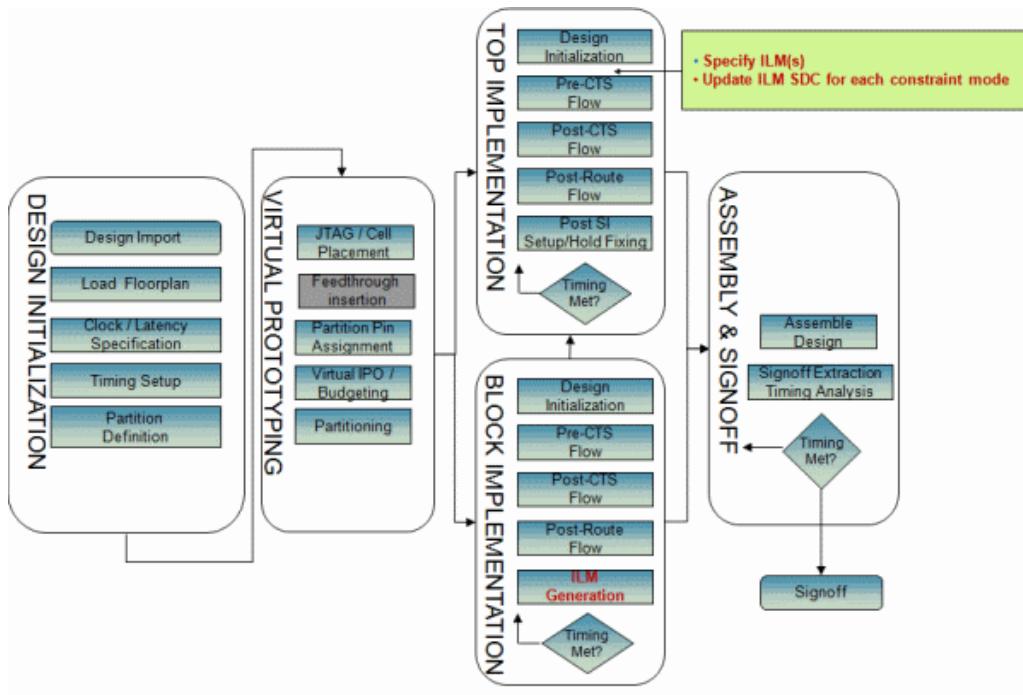
- More accurate analysis than a blackbox flow
  - More SI aware than combined `.lib` or `.cdb` approach
  - Can model clock generator inside block
  - More accurate timing and SI reduces the number of design iterations to close timing and SI.
- No need to characterize blocks
  - Works on actual design data
- Can be used in the initial prototyping stage for very big designs, when loading full design data

is not feasible.

- Allows you to modify only top-level data
  - Fully preserves implemented partitions- Uses the original constraint file for top-level analysis
  - No abstraction for timing exceptions

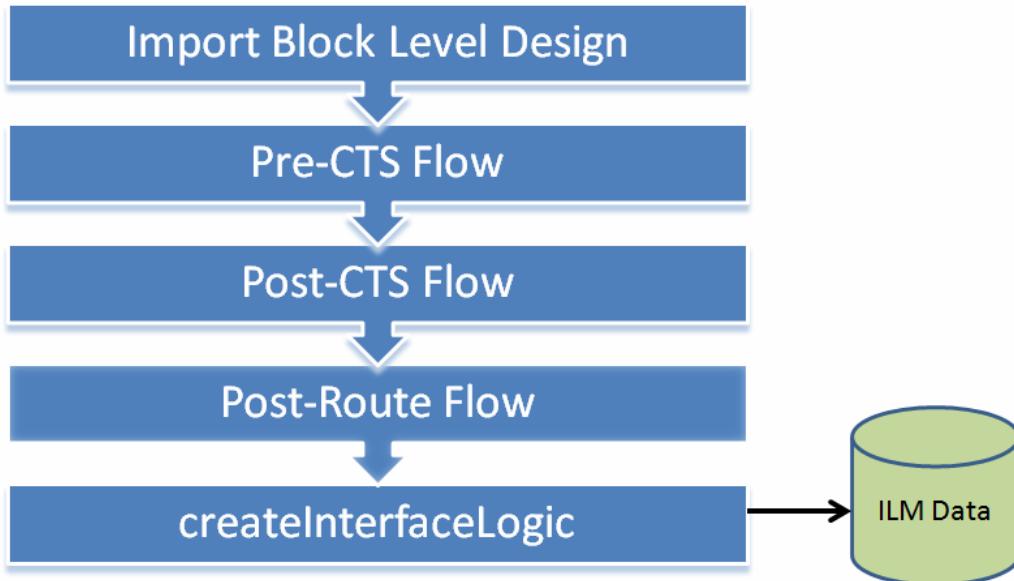
## General ILM Flow

ILM is used for top-level timing closure. Models will be generated during the block implementation stage and will be used at the top implementation stage. Following is the general ILM foundation flow:



## ILM Generation at Innovus Partition Block-level Design

Use the `createInterfaceLogic` command to generate ILM at the Innovus partition block-level design. Two types of models can be generated, ILM model and SI model. The ILM model will be used for timing analysis and clock tree synthesis, and the SI model can be used at postRoute stage for SI analysis. The SI model can only be generated if the block-level design is already SI-aware optimized.



Here is an example flow script for ILM generation:

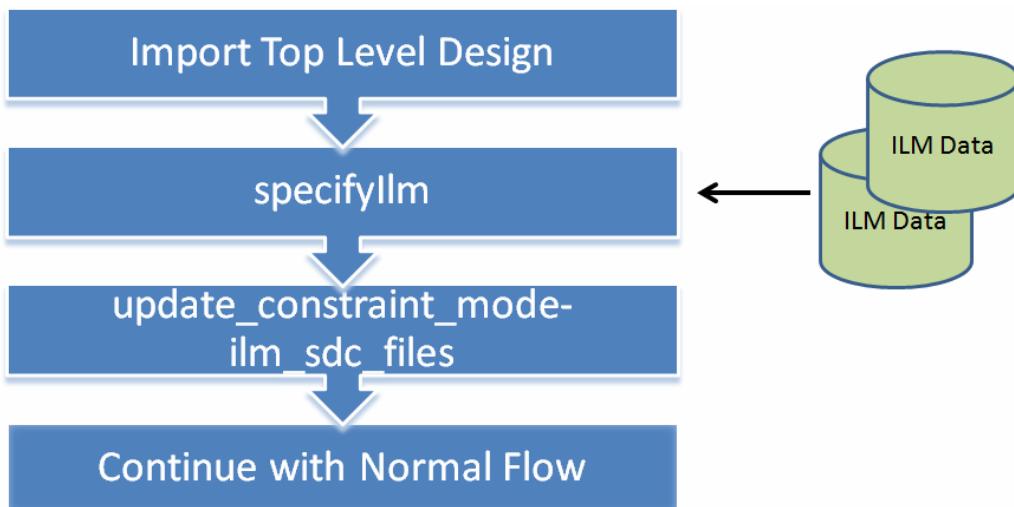
```
source DBS/init.enc
createActiveLogicView -type flatTop
deriveTimingBudget
clearActiveLogicView
partition
savePartition -lib -dir PTN -pt -def
set ptnNameList {tdsp_core ram_128x16_test ram_256x16_test}
# Create partition blocks
# Cd to each partition block directory, and create ILMs
foreach ptnName $ptnNameList {
    cd PTN/${ptnName}
```

```
restoreDesign . ${ptnName}  
  
placeDesign  
  
optDesign -preCTS  
  
clockDesign  
  
routeDesign  
  
saveDesign ${ptnName}_data.enc  
  
setAnalysisMode -analysisType onChipVariation  
  
optDesign -postRoute  
  
createInterfaceLogic -dir ilm  
  
cd ../../  
}
```

**Note:** You can also generate an ILM model using the third-party data with the [import\\_ilm\\_data](#) command. However, it is recommended to use Innovus-generated models for better timing correlation.

## ILM Integration at Top-level Design

Once the models have been generated, they can be specified at the top-level design for timing closure:



1. Specify a block as an ILM using the [specifyIlm](#) command.

```
set ilmNameList {tdsp_core ram_128x16_test ram_256x16_test}

foreach ilmName $ilmNameList {

    specifyIlm -cell ${ilmName} -dir ../../PTN/${ilmName}/ilm
}
```

2. Specify the ILM SDC constraint file for each constraint mode using the full chip SDC file (not the top-level SDC file).

```
update_constraint_mode -name setup_func \
    -ilm_sdc_files $dataDir/mmmc/dtmf_recv_core_func.sdc

update_constraint_mode -name setup_test \
    -ilm_sdc_files $dataDir/mmmc/dtmf_chip_testmode.sdc
```

3. Continue with the normal flow.



- ILM SDC file(s) are used in the flattened ILM view.
- Timing results are not available if the ILM SDC file is not defined.
- Top-level SDC file is used when ILMs are not specified.

Here is an example flow script for top-level timing closure using ILM:

```
restoreDesign . dtmf_recv_core

# Specify ILMs at top-level design

set ilmNameList {tdsp_core ram_128x16_test ram_256x16_test}

foreach ilmName $ilmNameList {

    specifyIlm -cell ${ilmName} -dir ../../PTN/${ilmName}/ilm
}

# NEED to specify SDC constraints for ILMs using FULL-chip SDC file
update_constraint_mode -name setup_func \
    -ilm_sdc_files $dataDir/mmmc/dtmf_recv_core_func.sdc

update_constraint_mode -name setup_test \
    -ilm_sdc_files $dataDir/mmmc/dtmf_chip_testmode.sdc

flattenIlm
```

```
placeDesign  
checkPlace  
trialRoute  
timeDesign -preCTS  
saveDesign DBS/place_route.enc  
optDesign -preCTS  
timeDesign -preCTS  
saveDesign DBS/optDesign.enc
```

## Creating ILMs

In the hierarchical design flow, you create a detailed block-level implementation of a block, then specify the `createInterfaceLogic` command to create an ILM for the block. This command creates the specified directory containing ILM files.

You can also create ILMs for blocks that are in an intermediate stage of design, then use the data at the top level of the design for preliminary timing optimization.

-  An ILM created for an incomplete block is not as accurate as an ILM created for a complete block. Always use ILMs for complete blocks to complete the top-level design.

The software generates ILM data for CTS, signal integrity, and other design stages (preCTS, postCTS, postRoute)

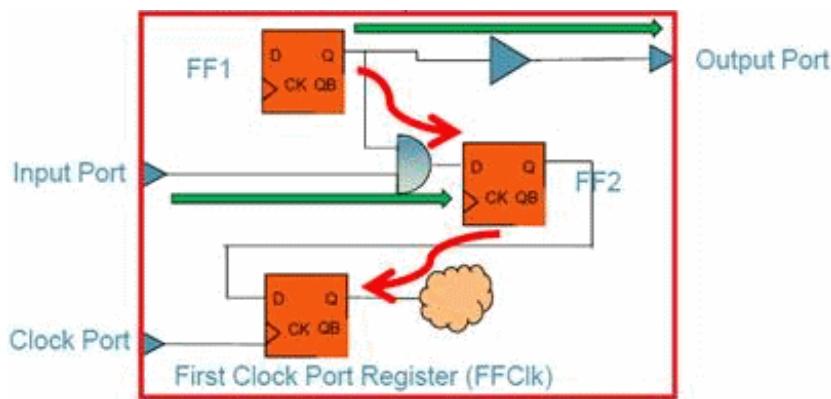
- ILM data for preCTS, CTS, postCTS, and postRoute

The model contains the netlist of the circuitry leading from the I/O ports to interface sequential instances (that is, registers or latches), and from interface sequential instances to I/O ports. The clock tree leading to the interface registers is preserved.

In case of CTS, the timing and CTS models have been merged to reduce the disk usage of an ILM model. The CTS data is limited to the worst clock sinks and instances/nets leading to those sinks.

ILMs do not contain information about the following:

- Internal paths (if `-noInterClockPath` is used): Internal paths controlled by a different clock, or clocks connected to the ILM module through different ports. If the logic between the I/O ports is pure combinational, it is preserved in an ILM.
- Internal register-to-register paths: if internal logic is not part of the interface path. However, in special cases some internal paths will be kept, as shown in the following example:



- ILM data for SI

The model includes all the above, plus aggressor drivers or nets which affect I/O paths. It also includes the timing window files in the ILM model directory.

Use `createInterfaceLogic -writeSDC` to generate block-level constraints that can be used:

- During a bottom-up design flow to build a top-level constraint file from the block constraints manually. The generated block-level `.sdc` file contains references to the block instances or pins or nets that made it into the ILM model netlist.
- To validate a model at the block level. For example, an ILM netlist and the `.sdc` file can be read in a separate Innovus session and timing analysis can be run on all paths. Then, the results can be compared against timing for the same path during full-block implementation.

**Note:** When `createInterfaceLogic` is called, all views are generated for multi-corner, multi-mode (MMMC) analysis.

## Example ILM Creation

The following method creates a model that can be used in the top-level implementation flow by both `timeDesign` and `optDesign` for setup effort, including postRoute SI optimization. This model is also used during `clockDesign` or `ccopt_design`.

```
createInterfaceLogic -dir block_A.ILM
```

## Sample Summary Report

The following is a sample summary report generated at the end of the `createInterfaceLogic` command:

```
-----  
createInterfaceLogicSummary  
-----
```

Model	Reduced Instances	Reduced Registers
ilm_data	7153/7621 (93%)	174/285 (61%)
si_ilm_data	6793/7621 (89%)	160/285 (56%)

In this report, the reduction ratio in the `ilm_data` model is 93 percent which means that 7153 out of the total 7621 instances for this block have been eliminated. Only 468 instances are written to the Verilog netlist for the `ilm_data` model out of which 111 instances are registers.

This summary report applies to a block using MMMC. Therefore, views with worst reduction ratio are displayed for each model.

## ILM Generated Data

ILM Generation provides the following output data:

File Extension	Description
.v	One netlist file

.sdc	One per constraint mode
.spf	One per corner
.place	One Innovus place file
.xtwf	One per analysis view. This file is only available with SI model

## Preserving Selected Instances in ILMs

You can force the selected instances and nets to be included in the ILM model by using the `createInterfaceLogic -keepSelected` parameter.

1. Select instances or nets using the `selectInst` or `selectNet` commands.
2. Specify `createInterfaceLogic -keepSelected`.

## Creating ILMs for Shared Modules

You can use the same sub-block module in different ILM blocks, enabling reuse of versatile modules. The `createInterfaceLogic` command considers constant propagation, so that only the enabled parts of a module are considered when creating ILMs for the reused modules. Because the Innovus database cannot handle the same module name in different circuits, the software automatically modifies the module names with the following rule:

`topModuleName+timestamp+$+moduleName`

As an example, one ILM block (`ModuleA`) uses an ALU module (ALU) as an unsigned ALU, and a second block (`ModuleB`) uses the ALU as a signed ALU. You can change the input signal to use the ALU differently, setting one ALU as sign enabled and the other to off. When you run the `createInterfaceLogic` command, the software considers only the enabled parts of the ALU when creating ILMs for `ModuleA` and `ModuleB`. The software also ensures that the name of the ALU module in `ModuleA` and the name of the ALU module in `ModuleB` are different.

## Creating ILMs Without Using Innovus Database

If you do not have an Innovus database for an implemented block but have a Verilog netlist, constraints, and SPEF for that block, then use the `import_ilm_data` command to store data in the ILM format.

Following is the usage of the `import_ilm_data` command:

```
import_ilm_data -cell blk1 -dir A -model_type timing -verilog V1.v \
                 -spef 1.spef -rc_corner corner1 -sdc sdcfile1 -timing_view view1

import_ilm_data -cell blk2 -dir A -model_type si -verilog V2.v \
                 -spef 2.spef -rc_corner corner2 -sdc sdcfile2 -timing_view view2

import_ilm_data -cell blk1 -model_type timing -cell_view designLib \
                 -verilog V1.v

import_ilm_data -cell blk1 -dir A -model_type timing -verilog V1.v \
                 -spef 1.spef -rc_corner corner1

import_ilm_data -cell blk1 -dir A -model_type timing \
                 -sdc sdcfile1 -timing_view view1
```

-  ● `-cell`, `-modelType`, `{-cellview | -dir}` are required options. When you import an ILM model from a third party, you must specify the cell that you want to transfer to Innovus, the model that will be put inside the Innovus ILM model, and the directory where the ILM data will be stored.
- `-timing_view` requires `-sdc`, and `-rc_corner` requires `-spef`. If these options are not provided together, the tool will give an error message.

**Note:** The `import_ilm_data` command does not merge timing and CTS data into one model. ILMs created using this command will have separate models for timing, CTS, and SI.

## Specifying ILM Directories at the Top Level

Use `specifyILM` for ILM data of a block at the top partition level rather than using the default `.lib` model. You can run `specifyILM` multiple times in the same session. Each time you run this command, the software overwrites the previous setting for the same block. If master/clones exist in the design, the cell name will have the name of the master partition.

**Note:** You can use this command (and `unspecifyILM`) only if the ILMs are unflattened (`unflattenILM`). You cannot change ILM settings in flattened ILM view.

Use `unSpecifyILM` to revert to the `.lib` model of the block.

## Example Top-level Implementation Flow with ILMs

1. Before you start the Innovus tool, prepare the top-level Verilog file, if needed.

If you use the Innovus hierarchical flow in a previous Innovus session, then the `savePartition` command automatically creates the top-level data. Else, you need the following in the top-level directory:

- A Verilog netlist that includes dummy modules for the blocks (ILM or Liberty) in the design.
- A view definition file since ILMs are supported only in the MMMC mode. If you have a non-MMMC design, create or load a view definition file that contains the following:  
`set_analysis_views -setup {model1_slowCorner} -hold {model1_fastCorner}`

2. Start an Innovus session from the top-level module directory within the directory where the partitions are saved.
3. Load the design, including the top-level netlist, ILM directory name, `ilm_blocks.lib` (optional if using ILM), `stdcells.lib`, and `.lef` for the block and chip-level constraints.

```
specifyILM -cell block_A -dir ../block_A/block_A.ILM  
specifyILM -cell block_B -dir ../block_B/block_B.ILM
```

4. Load the floorplan.

```
loadFPlan top_floorplan
```

**5. Place the design.**

```
placeDesign
```

**6. Run preCTS timing optimization.**

```
optDesign -preCTS
```

**7. Build the clock tree.**

```
clockDesign
```

**8. Run postCTS timing optimization.**

```
optDesign -postCTS
```

```
optDesign -postCTS -hold ;#optional
```

**9. Route the design.**

```
routeDesign
```

**10. Run SI Aware inside postrouteOpt and postrouteOpt\_hold.**

```
setDelayCalMode -SIAware true  
optDesign -postRoute  
optDesign -postRoute -hold
```

If you want to create an ILM of the resulting block for use in the next level up in the hierarchy, run the following steps with the above-mentioned flow:

**1. Enable SI aware delay calculation.**

```
setDelayCalMode -SIAware true
```

**2. Perform timing analysis.**

```
timeDesign -postRoute
```

**3. Create ILM.**

```
createInterfaceLogic -dir block_parent
```

## ILMs Supported in MMMC Analysis

Cadence strongly recommends that you use ILMs in the MMMC mode. If you have a non-MMMC design, create and load a view definition file that contains the following:

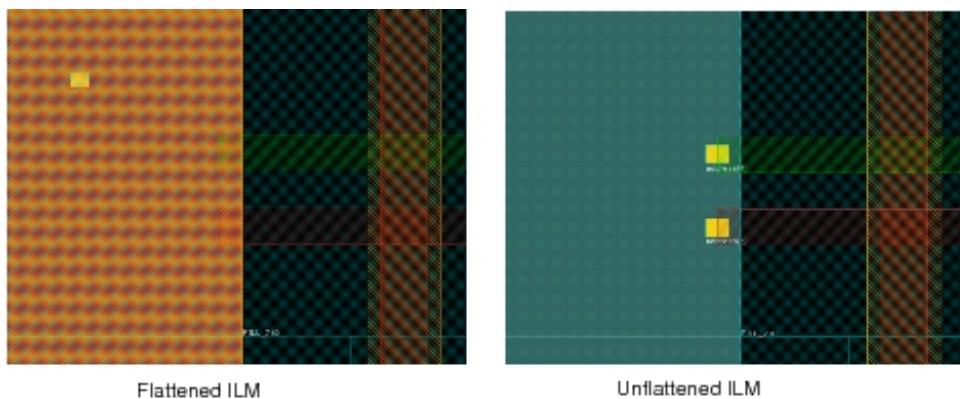
```
set_analysis_views -setup {model_slowCorner} -hold {model_fastCorner}
```

The MMMC analysis for designs including ILMs is identical to MMMC analysis for black box designs except for the following considerations:

1. Views, modes, and corners at the top and partition levels must have same names.
2. When you use `create_constraint_mode` or `update_constraint_mode` to specify constraints for MMMC, you must specify the ILM constraints using the `-ilm_sdc_files` parameter (that is, timing in the presence of ILMs gets constraints from the `-ilm_sdc_files` parameter, not the `-sdc_files` parameter). The .sdc files specified with the `-ilm_sdc_files` parameter should be the constraint file of the full-chip flat netlist where it allows referencing nets or pins internal to the ILM model.

If you want to see the LEF pins of the ILM in GUI, the design must be in the unflattened mode.

The following figure shows the flattened and unflattened ILM. The LEF pins of the ILM are visible after unflattening the ILM.



## ILMs Supported in SI

ILM supports the SI aware for `optDesign` and `timeDesign`. These commands automatically run `setIImType -model si` before calling `flattenIIm` such that the SI ILM model is used. Therefore, your present postRoute optimization scripts should run successfully in the presence of ILMs (without any additional changes).

The following command can be used to get timing reports containing the SI push-out delays on nets using the `-setIImType -model` command:

```
# Reflattens to SI model, then does not unflatten (All other design
# commands unflatten upon exit, regardless of the flattened/unflattened
# state before invocation)
setDelayCalMode -SIAware true
timeDesign -postroute

# Adds incremental delay column (for SI push-out delays) in timing output:

set_global report_timing_format {instance arc cell fanout load slew delay incr_delay
arrival}

# Minimizes the width of the report such that it easily fits into the screen
# without wrapping

set_table_style -name report_timing -no_frame -indent 0

report_timing
```

**Note:** You can also invoke the Global Timing Debugger (*Timing - Debug Timing - Generate*).

## SI Model Generation

To enhance the accuracy for top-level SI analysis, SI models are supported only when the RC database has coupling capacitance information. This information is needed for correct SI analysis.

- i In the SPEF flow, ensure that SPEF has coupling capacitance data for the SI model that is to be generated. In the extraction flow, the `extractRCM extractRCMode` settings determine whether the SI model is generated (the extraction mode will not be changed internally to generate the SI model).

## ILM Model

ILM model contains both timing and CTS information instead of having a separate model for each of them to reduce the disk usage. In this model, the timing and CTS models have been merged by only the:

- interface paths from the timing model.
- best/worst latency registers that are to be kept as a part of the CTS model. All other registers are excluded.
- worst inter-clock paths for the timing model.

## Interactive Use of ILMs

- When `setILMMode -keepFlatten` is set to true, the flow will work under the flattenILM mode. As a result:
  - The number of flatten/unflattenILM calls inside super commands, such as `placeDesign`, `timeDesign`, `optDesign`, `ccopt_design`, is reduced.
  - Super commands except `assembleDesign` will return in the same ILM mode as it was.
  - Once the design is flattened, it will be kept in the flattened state until you run unflattenILM. This will help reduce the number of flatten/unflattenILM operations during the timing closure where you can run `timeDesign`, interactive debugging, setting additional SDC constraints, manual ECO in the flattenILM mode.  
Note: For multi-stage scripts, you need to set this option only once.
  - For non-related timing commands, Innovus will automatically unflatten the design, run the command, and flatten the design back. However, it is recommended that you explicitly run `unflattenILM` for a group of physical commands, such as `verifyConnectivity` and `verifyGeometry` to improve the run time.

The list of commands for which Innovus will do automatic unflattenILM/flattenILM when `setILMMode -keepFlatten` is set to true is given below:

<code>addAIORow</code>	<code>flattenTile</code>
<code>addBumpToArrayGrid</code>	<code>flattenTileRow</code>

addChannelDensityControl	floorPlan
addDeCap	freeDesign
addEndCap	generateTracks
addHaloToBlock	generateVias
addRing	insertPtnFeedthrough
addRoutingHalo	legalizeFPlan
addStripe	legalizePin
addViaFill	loadFPlan
addWellTap	loadPtnPin
alignPtnClone	ObsfHV_Ob
amoebaPlace	ObsfSV_Ob
assignBump	ObsrPReqt_Ob
assignIoPins	optCritPath
assignPGBumps	optFanout
assignPtnPin	pinAnalysis
assignSigToBump	placeSpareModule
assignSigToTilePin	preassignPin
autoGenRelativeFPlan	relativeFPlan
balanceSlew	restoreDesign
checkBump	runCCAR
checkDrc	runQRC
checkDrcInVisibleArea	saveFPlan
checkFPlan	savePartition

checkHierRoute	savePlace
checkPinAssignment	savePtnPin
checkRoute	saveRelativeFPlan
ciopCreateBump	saveRoute
clearCutRow	saveRouteGuide
clearRelativeFPlan	spaceBondPad
clonePlace	specifyBlackBox
clusterPlace	specifyILM
commitPartition	specifyPartition
convertBlackBoxToFence	specifySpareGate
convertFenceToBlackBox	sroute
convertFenceToLef	summaryReport
create_constraint_mode	trimMetalFill
createPinBlkg	uniqualifyNetlist
createPinGuide	unloadPtnPin
createPtnCut	unplaceAllBlocks
createPtnFeedthrough	unplaceAllGuides
createPtnPinBlk	unplaceAllInsts
createRegion	unplaceGuide
createRouteBlk	unplaceGuideConstraints
createRow	unplaceJtag
createShifterRows	unspecifyBlackBox
createSoftGuide	unspecifyILM

createTileInst	unspecifyJtag
cutCoreRow	verifyConnectivity
cutPowerDomainByOverlaps	verifyCutDensity
cutRectilinearInst	verifyGeometry
cutRow	verifyMetalDensity
defIn	verifyProcessAntenna
defOut	verifyTracks
deleteMetalFill	verifyWellTap
fcroute	write_lef_abstract
fillNotch	wroute
fixDRCViolation	
fixMinCutVia	
fixMinStepVia	
flattenCoverCell	

## Handling Interactive Constraints

You cannot specify the interactive constraints when ILMs are not flattened (`unflattenilm`). In the flatten mode (`flattenilm`), you can specify both interactive and modeless constraints and these constraints are used during various cycles of `unflattenilm` to `flattenilm`. During `saveDesign`, these constraints are honored.

To specify additional constraints while running `unflattenilm`, set the following:

```
set_global timing_defer_mmmc_object_updates true
set_interactive_constraint_modes [all_constraint_modes -active|or
your_own_list_of_modes]
foreach mode {list_of_modes_to_be_updated} {update_constraint_modes -name $mode
-ilm_sdc_files \
[concat get_constraint_modes -name $mode -ilm_sdc_files] additional.sdc}
```

}

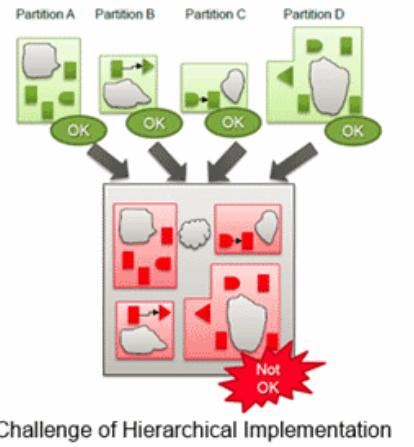
Include all mode-less constraints such as `timing_derates` and `group_path` in a separate file, and run:

```
source <mode-less_constraint.sdc>
set_global timing_defer_mmmc_object_updates false
set_analysis_view -update_timing
```

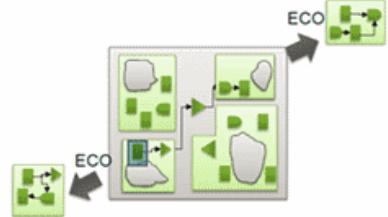
## Using Flexible Interface Logic Models (FlexILM)

### Overview

In the top-level timing closure, timing budgeting is not accurate and inter-partition critical paths are hard to close. After partition, blocks are implemented in a number of days and are modeled as ILM blocks at the top-level design for the top-level timing closure. ILM blocks are read-only, and as a result, top-level timing issues are not fixed easily, especially in channel-less designs where there is no room for buffer insertion. Designers may need to do at least two or three passes of hierarchical flow to close timing. To address this challenge, a single-pass hierarchical solution with Flexible Interface Logic Models (FlexILM) can be used. FlexILM is a reduced netlist where logics on interface paths are kept and logics on internal paths are removed. At the top-level design, interface paths of FlexILMs can be optimized, and netlist and placement changes can be ECO back to partition blocks automatically. FlexILM also reduces the memory in timing graph and physical data where removed instances are replaced by placement blockages to avoid violations with the newly added optimized logics. Additionally, routing of removed nets will be replaced by RC grids to improve RC extracted correlation.



Challenge of Hierarchical Implementation



FlexILM Single-pass Hierarchical Implementation

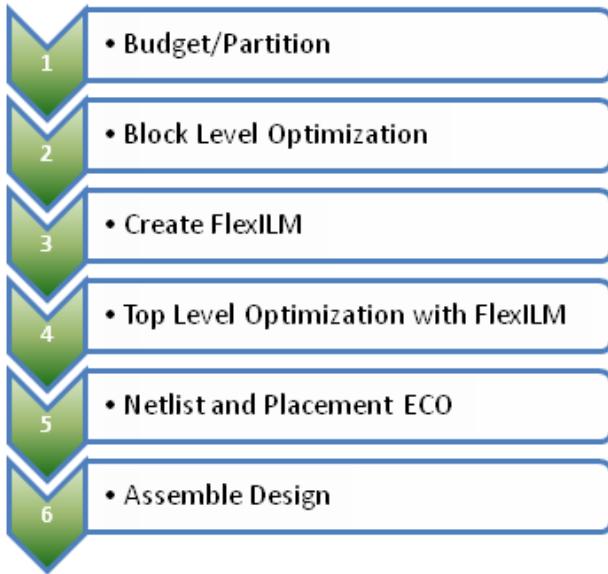
The advantages of using FlexILM are as follows:

- Enables concurrent top and block optimization for interface paths
  - Fixes inter-partition timing critical paths early in the flow where interface logics can be touched
- Does not need accurate timing budget
- Is the only solution for a channel-less design

Currently, FlexILM does not provide support for master/clone, route, and CCopt with useful skew, and there is minor delay or RC correlation due to a different routing pattern.

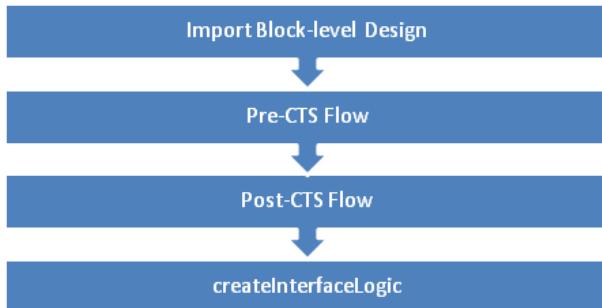
## General FlexILM Flow

FlexILM is used for the top-level timing closure. Models should be generated during the block implementation stage and will be used at top-level design for timing closure. Following is the general FlexILM flow:



## FlexILM Model Creation

FlexILM can be generated at the Innovus partition block-level design using `createInterfaceLogic -useType flexILM`. FlexILM supports the preCTS and/or postCTS optimization stage. It converts reduced instances to placement blockages and reduced net wires to RC Grid.



Following is an example flow script for generating FlexILM:

```
# Cd to <savePtn_working_dir>/>ptn_name>

cd PTN/${ptnName}

restoreDesign . ${ptnName}

placeDesign

optDesign -preCTS
```

```
saveDesign ${ptnName}_data.enc

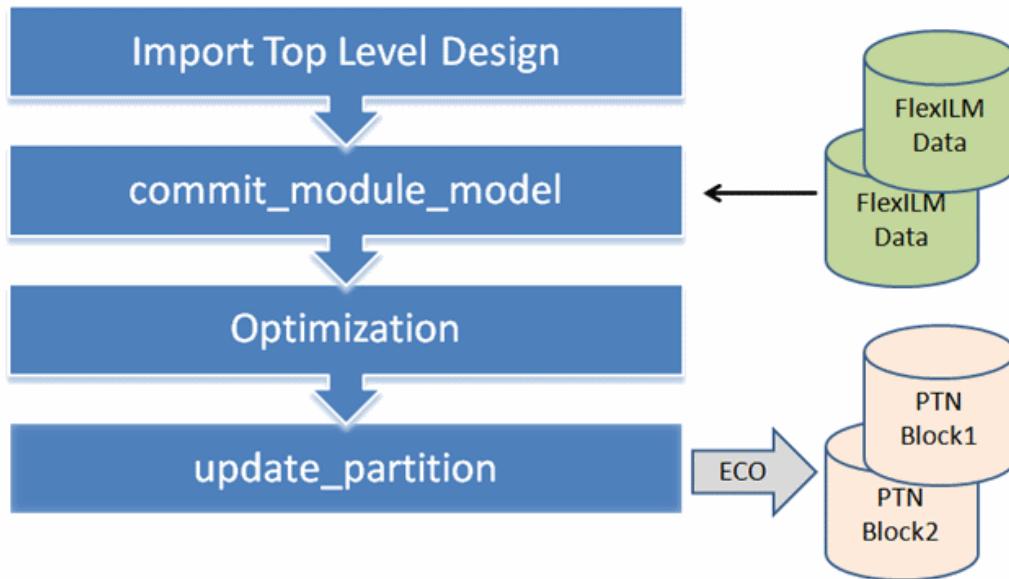
createInterfaceLogic -dir flexILM -useType flexILM \
    -optStage preCTS
```

`-optStage` is mentioned to specify the stage at which the flexILM model will be used, preCTS or postCTS.

**⚠** When `createInterfaceLogic` is called, all the views are generated for multi-corner, multi-mode (MMMC) analysis.

## Top-Level Optimization

After creating FlexILM model from each partition, these models can be specified and committed at the top level using the `commit_module_model` command. Once the design is optimized, changes can be ECO back to block-level design using `update_partition`.



Following are the main steps of the top-level optimization flow:

1. Restore the design for the top level.

```
restoreDesign . dtmf_recv_core
```

2. Commit FlexILM model for all the blocks.

```
set flexIImList {tdsp_core ram_128x16_test ram_256x16_test}
set dirs ""

foreach name $flexIImList { append dirs " -flex_ilm ${name} \
../${name}/${flexIImDir}" }

eval commit_module_model $dirs -mmmc_file \
$dataDir/dtmf_recv_core.enc.dat/viewDefinition.tcl
```



- The full chip MMMC view file should be specified with the `-mmmc_file` option when running `commit_module_model`.
- You can use the following command to get back to the full block after you have committed the FlexILM model at the top level.

```
commit_module_model -full_block
```

3. After committing FlexLM model, you optimize the design to improve timing by following the steps given below:

```
foreach name $flexIImList {setPtnPinStatus -cell ${name} \
-pins * -status unplaced}
```



- It is recommended to delete the existing partition pins to give `optDesign` more flexibility to fix timing. Once the design is optimized, re-route the design to re-assign partition pins.

a. Ensure that you keep logic ports during optimization.

```
setHierMode -optStage preCTS
```

b. Run `optDesign` to optimize the design.

```
optDesign -preCTS
```

c. Run `assignPtnPin` to re-assign partition pins and update the pin location back to the block design.

```
trialRoute
```

```
timeDesign -preCTS  
assignPtnPin  
checkPinAssignment
```

d. Timing budget based on the updated netlist.

```
setTrialRouteMode -honorPin true  
trialRoute  
deriveTimingBudget
```

4. Run update\_partition to create a new partitioned output that contains the optimized blocks and top. It also performs ECO on the original block. The ECO output is saved as a new Innovus database in the partitioned block.

```
set dirs ""  
  
foreach block $flexIImList { append dirs " -goldenBlockDir \  
${ptnDir}/${block}/${block}_data.enc.dat" }  
  
eval update_partition -flexIImECO $dirs -flexIImDir ${postOptDir} \  
-postECOSuffix postECO -pinLocation
```

After update\_partition, ECO outputs information for the top level, and all the block-level designs will be saved within the “-flexIImDir \$postOptDir” directory. -postECOSuffix will add suffix to DB saved from ECO output on all the blocks. If you do not set this option, the default option name is <block\_name>\_eco\_<timestamp>.enc.dat. -pinLocation will update partition pins in the “golden” block DB with new assigned pin locations.

Following is an example flow script:

```
set dataDir [pwd]/design  
set libfir [pwd]/libs  
set flexIImDir flexIIm  
set ptnDir ../../PTN  
set postOptDir ./flexIIm_postOpt  
  
setMultiCpuUsage -localCpu 4  
  
cd PTN/dtmf_recv_core  
restoreDesign . dtmf_recv_core  
  
# Specify flexILMs at top-level design
```

```
set flexIImList {tdsp_core ram_128x16_test ram_256x16_test}
set dirs ""
foreach name $flexIImList { append dirs \
" -flex_ilm ${name} ../${name}/${flexIImDir}" }

eval commit_module_model $dirs -mmmc_file \
$dataDir/dtmf_recv_core.enc.dat/viewDefinition.tcl

foreach name $flexIImList {setPtnPinStatus -cell ${name} \
-pin * status unplaced}
setTrialRouteMode -honorPin false -handlePartitionComplex true
setHierMode -optStage preCTS
setOptMode -handlePartitionComplex true
optDesign -preCTS
saveDesign flexIIm_preCTS.enc

# Re-assign pins based on new netlist changes
trialRoute
timeDesign -preCTS
assignPtnPin
checkPinAssignment
setTrialRouteMode -honorPin true
trialRoute
deriveTimingBudget
set dirs ""
foreach block $flexIImList { append dirs " -goldenBlockDir
${ptnDir}/${block}/${block}_data.enc.dat" }
eval update_partition -flexIImECO $dirs -flexIImDir ${postOptDir} \
-postECOSuffix postECO -pinLocation
```

## Full Chip Assembly

After net list changes had been ECO back to block-level designs, block designs with ECO changes can be brought back to the top-level design for chip assembly to check timing.

Following is the sample script:

```
set dataDir [pwd]/design
set libDir [pwd]/libs
set flexIIm_postOpt_dir ../flexIIm_postOpt
set topCell dtmf_recv_core
```

```

cd PTN/${topCell}
set dirs "-topDir ${flexILm_postOpt_dir}/${topCell}"
set blockList "ram_256x16_test ram_128x16_test tdsp_core"
foreach block $blockList {
    set dir [ glob \ ${flexILm_postOpt_dir}/${block}/${block}_${postEcoSuffix}.enc.dat ]
    append dirs " -blockDir $dir"
}
eval assembleDesign $dirs -defMerge \
-mmmcfFile ${dataDir}/${topCell}.enc.dat/viewDefinition.tcl
dbDeleteTrialRoute
setTrialRouteMode -honorPin true -handlePartitionComplex true
trialRoute
createActiveLogicView
timeDesign -preCTS
clearActiveLogicView

```

## FlexILM Model Data

FlexILM generation provides the following output data:

File Extension	Description
.v	Netlist file
.sdc	One per constraint mode
.def	One DEF placement file
.rcg	One RC Grid file
.blk	One placement blockage file

## Extracting Timing Models

- [Overview](#)
- [ETM Generation](#)
- [ETM Generation for MMMC Designs](#)
- [ETM Generation Flows](#)

- Basic Elements of Timing Model Extraction
  - 
  - Simple Clock Gating with AND or Logic
  - Clock Gating with Blackbox or Unknown Logic
  - No Clock Gating Logic
  - Annotate Delays, Load, and Slews
  - Design Rules
  - Handling Typical Clock Definitions
- Clocks with Sequential and Combination Arcs
- Secondary Load Dependent Networks
- Characterization Point Selection
- Constraint Generation during Model Extraction
  - Clock Definitions
  - Arrival and Required Time
  - Global Constraints
- Slew Propagation Mode in Model Extraction
- Parallel Arcs in ETM
- Latency Arcs Modeling
- Latch-Based Model Extraction
- Stage Weight Extraction in ETM
  - Stage weight setting in ETM
  - AOCV Derating Mode
  - Merging model with stage\_weight attribute
- PG Pin Modeling During Extraction
  - Various Type PG Modeling in ETM
  - ETM Merging Requirements for Power/Ground Aware ETMs
- Extracting Timing Models with Noise (SI) Effect
- Merging Timing Models
- Limitations of ETM
- Validation of Generated ETM
  - Validation Flow for Non-MMMC Designs
  - Validation Flow for MMC Designs
  - Validation Reports
- ETM Extremity Validation

- Limitation/Implications of EV-ETM

## Overview

Timing model extraction is the process of abstracting the interface timing of hierarchical blocks in a timing library. Typically, model extraction has the following advantages:

- Reduces the memory requirements by generation of extraction timing models (ETMs) for respective blocks, which may be huge in size.
- Reduces the static timing analysis (STA) runtime.
- Hides the proprietary implementation details of the block from a third-party.

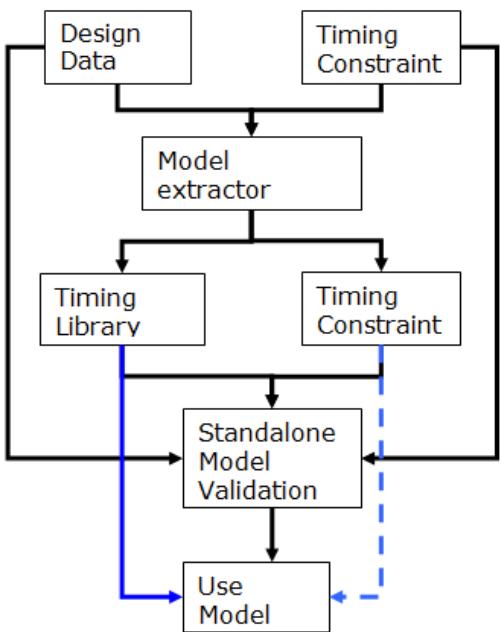
The `do_extract_model` command is used to build a timing model of a digital block to be used with a timing analyzer. The automatic derivation and extraction of the "actual" timing context for a lower-level module instance is required for achieving timing convergence with large design databases. The software has the following advantages for timing extraction:

- Provides a fast timing engine that allows for quick derivation of a module's timing context.
- Extracts data correctly across multiple clock domains.
- Allows merging of various models in various modes for their respective blocks.

You can start with model extraction with the following data loaded in the software:

- Design netlist.
- Timing libraries.
- Block context (i.e. constraints like clocks, path exceptions, operating conditions etc.)
- RC data of the nets.

The following diagram illustrates this flow:

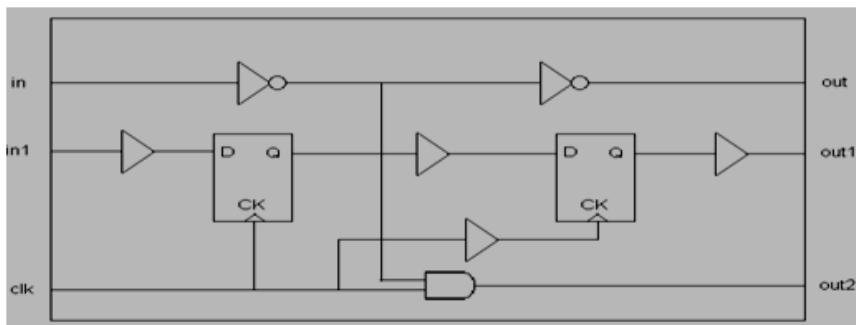


## ETM Generation

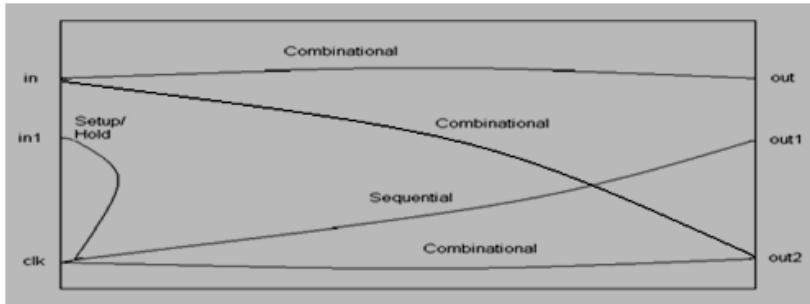
ETM represents the timing for interface paths. Input to flop/gate, input to output and flop/gate to output paths of a design are represented as timing arcs in its ETM. If there are multiple clocks capturing the data from an input port then an arc with respect to each input port is extracted.

Flop to flop type of paths are not considered in the extraction process as these do not affect the interface paths timing.

The following figure shows the original netlist



The following figure shows the Extracted Model:



The timing model extracted is context independent as it gives the correct value of arcs/delays, even with varying values of input transitions, output loads, input delays, or output delays. In such cases, it is not required to re-extract the model if some of the context gets changed at a later stage of development.

However, the model depends upon the operating conditions, wire load models, annotated delays/loads, and RC data present on internal nets defined in the original design. So if these elements change at the development stage of design then we need to re-extract the model for correlation with the changed scenario.

## ETM Generation for MMMC Designs

In MMMC configuration, for generating ETM, only one view should be active. If you need to generate ETM for a specified view, however, the view is not active in setup as well as hold, then the software will error out. It is a prerequisite that the view should be active for both setup and hold analysis. You can use `set_analysis_view -setup {$viewName} -hold {$viewName}` command before running the `do_extract_model` command to achieve this.

After model extraction for MMMC designs, the dumped assertion files will be added with  `${viewName}` suffix, and the dumped derate related assertion files will be added with  `${delayCornerName}` prefix. You need to choose the corresponding file while validating the extracted models.

## ETM Generation Flows

There are two types of extraction models which are currently existing. They can be controlled by using following variable:

```
set timing_extract_model_compatibility_mode {true/false}
```

When set to `true`, the extracted model is based on graph-based flow.

When set to `false`, the extracted model is based on the `report_timing` based flow. This document

highlights the `report_timing` based model extraction method.

## Basic Elements of Timing Model Extraction

The following are the various basic elements which are relevant to model extraction:

- Nets – internal and boundary nets
- Check and delay paths – in2reg, in2out, and reg2out paths
- Minimum pulse width and periods checks
- Path exception – false, multicycle, min delay, max delay, disable timing
- Constants
- Unconstrained paths
- Clock gating checks
- Annotated delays, slews, and loads
- Design rules checks
- Clocks
- Generated clocks

These are explained below.

### Nets

Nets can mainly classify into two types – boundary nets and internal nets. The nets which are directly connected to the input or output ports of the block are termed as the boundary nets. The nets which drive and are driven by some internal instance pin of the block are termed as the internal nets. Model extraction treats both the nets differently as the boundary nets are always related and connected to the context.

## Boundary Nets

The boundaries are connected to the context. So the RC data for them may change if the context of the block changes at a later stage. To be better context independent the model should not consider the SPEF or DSPEF data for their delay calculation and a separate SPEF/DSPEF file for the boundary nets can be stitched to the top level data while instantiating the extracted model.

The software by default considers the RC data while extracting the model. The software provides a command line option which can be used to ignore the RC data for boundary nets by using following option:

```
set_delay_cal_mode -ignoreNetLoad true
```

**Note:** Currently, the software does not dump the SPEF file for boundary nets which can be used at the top level. To solve this problem, you need to create the top level SPEF file for boundary nets of this block or use the default behavior to consider the boundary nets RC data for model extraction.

## Internal Nets

As the internal nets connect the pins for the internal instance of the block, they are in no way dependent on the context environment. So the RC data defined for the internal nets is used as it is for delay calculation and is added in the extracted paths. If no RC is defined for these nets, then the wire load models are used to calculate their delays.

However, if a load or resistance is annotated using the `set_load/set_resistance` command then it will override the annotated SPEF or the applied wire load model.

If the nets are annotated with the delays using the `set_annotationed_delay` command or SDF annotation, then the annotated delay is used instead of the calculated delays. In case of incremental delays the addition of calculated and incremental delays is used.

## Timing Paths

Timing paths are broadly divided in four types:

- In2reg
- reg2reg
- in2out
- reg2out

**Note:** The reg2reg paths are not relevant to the model extraction process.

## In2Reg Paths

An in2reg path is a setup/hold check that starts from an input port and is captured at a flop or gating element by a clock. So the in2reg types of paths are captured in an equivalent setup or hold checks. These arcs contain the delay calculated on the basis of the delay from an input port to the flop, the setup/hold value of the library cell and the delay from a clock source to the clock pin of the flop. The equations can be written as follows.

Setup arc delay = delay (input to flop) + delay (setup value of flop) – delay (clock source to clk pin)  
 Hold arc delay = delay (input to flop) - delay (hold value of flop) – delay (clock source to clk pin)

The delay for these arcs is the function of the transition on the input port and the transition at the clock source. If a flop is captured by multiple clocks then separate setup/hold arcs are extracted with respect to each clock source.

## Reg2out Paths

The reg2out paths are paths starting from a register and ending up on an output port. These paths are a combination of trigger arcs (of starting register) and the combinational delay from sink of trigger arc to the output port. So for such paths an equivalent trigger/sequential arc is modelled in the extracted model. The delay for the arc will be equal to:

Sequential arc delay = delay (clock source to CK of register) + delay (register CK pin to out port).

The delay for these arcs is a function of the slew at the clock source and the capacitance at the output port. As the extracted model can be used for max as well as min analysis, two arcs are preserved in the model to represent the longest and the shortest path. Different types (rising\_edge / falling\_edge) of arcs are extracted for the different valid clock edges.

## In2Out Paths

The in2out paths are the path starting from an input port and ending up on an output port. These paths are pure combinational paths. So for such paths an equivalent combinational arc is modelled in the extracted model. The delay for the arc will be equal to:

Combinational arc delay = delay (delay of all elements in the path)

The delay for these arcs is a function of the slew at the input port and the capacitance at the output port. As the extracted model can be used for max as well as min analysis, two arcs are preserved in the model to represent the longest and the shortest combinational path. In case if no path exists for a particular transition (rise/fall), half unate (combinational\_rise /combinational\_fall) arcs will be extracted. The timing sense for the arc will depend on the function of worst (early/late) paths.

## Minimum Pulse width and Period

The min\_pulse\_width and period constraint defined at the CK pin of the registers are transferred to the clock source pins during model extraction.

There may be several different type of registers in the fanout of a clock source. So while transferring the min pulse width to the clock source you can use the worst values present on the fanout registers. The pulse width is calculated as:

`pulse width = MAX(Arrival time of Launch clock Path - Arrival time of Capture Clock Path + pulse width at clock pins from library)`

The `min_pulse_width` check is modelled as arc, in case following variable is set to `true`, that is,

```
set timing_extract_model_write_clock_checks_as_arc true
```

In this case the resulting arc is the function of rise/fall slew of CLK port. While creating the `min_pulse_width` check, the delay and slew propagation of the clock network for rise and fall transitions is considered. The minimum period for any clock pin is calculated in the same way as the minimum pulse width. The minimum period is calculated as:

`Min Period = MAX(Arrival time of Launch clock Path - Arrival time of Capture Clock Path + Min Period at clock pins from library)`

## Path exceptions

Timing extraction flow honors path exceptions defined in the given constraints' file. Typically, the following path exceptions are used:

### **set\_false\_path**

The false path exceptions are handled by Modeling non-false paths in the ETM. Any path on which false path is applicable is not extracted in the timing model.

### **set\_multicycle\_path**

In case minimum clock period is applied through pins/ports, then during extraction, worst path through them is mapped to IO ports and exceptions through these IOs are dumped in the assertion file.

### **set\_min\_delay/set\_max\_delay**

Minimum or maximum delay assertions applied on the ports or on the timing check arcs are preserved in the assertion file during ETM extraction.

## Constants

The case analysis and the constants are propagated while extracting the model. In case of `set_case_analysis` command, we can control how constants propagated at o/p or bi-di ports are modelled in ETM through following variable.

```
set timing_extract_model_case_analysis_in_library {true/false}
```

When set to `false`, this global variable specifies the propagated or applied constants, using `set_case_analysis` command to output ports in the generated constraints file (generated using `-assertion` parameter in the `do_extract_model` command).

When set to `true`, this global variable specifies that propagated constant to output ports are written as pin function in extracted model.

## Unconstrained Paths in ETM

The unconstrained end points exist when proper launch/capture clock phase is not propagated at the desired endpoint due to any exceptions. The unconstrained input ports due to false path exceptions are ignored and are not modelled during ETM extraction. The unconstrained combinational IO paths due to false path exceptions are ignored and are not modelled during ETM extraction. The unconstrained trigger arcs are calculated during ETM extraction. You can even validate the unconstrained paths in the `compare_model_timing` using the following global variable:

```
set timing_extract_model_validate_unconstrained_paths {true/false}
```

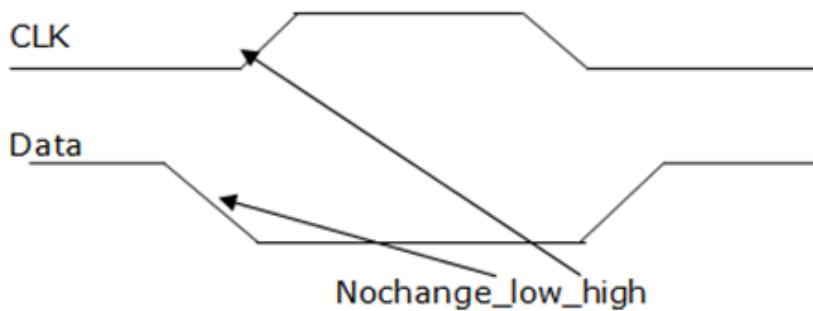
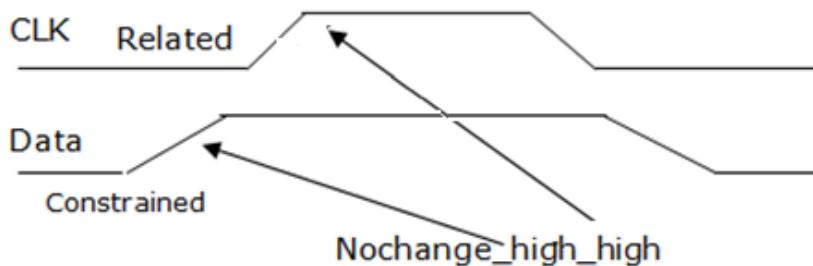
The purpose of the global is to report unconstrained arcs specified, using the `write_model_timing` command, so that unconstrained paths can be validated automatically between pre-ETM and post-ETM sessions when set to `true`.

## Clock Gating Checks

The clock gating checks are modelled as `no_change` arcs between a clock pin and its enabling signal pin. If you wish to extract these checks as regular setup/hold checks, you can set the `timing_extract_model_gating_as_nochange_arc` global variable to `false`.

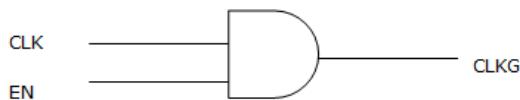
Depending on the type of clock gating situation, no change checks are inferred as follows:

The following waveform shows how `no_change` checks are modelled:



## Simple Clock Gating with AND or Logic

The following diagram shows clock gating AND/OR logic:



A simple AND gate check will be as follows:

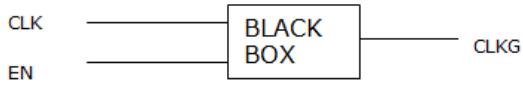
```

timing () {
    timing_type: nochange_high_high;
}
timing () {
    timing_type: nochange_low_high;
}
    
```

**Note:** After extracting the clock gating check arc the signal downstream the gate output is not propagated to the clock pins of the registers.

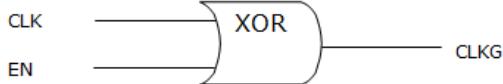
## Clock Gating with Blackbox or Unknown Logic

In case of clock gating with unknown logic, as shown below, no\_change checks will be checked with respect to both the rising edge and the falling edge of the clock signal.



## No Clock Gating Logic

There are some gates, such as multiplexers or XOR gates, where a clock signal cannot control the clock gate, as shown below.



In such cases, no checks are inferred so you need to explicitly set the gating check if you want these interface paths to be extracted in the ETM. To know where static timing analyzer will not infer the gating in such situations, you can use the following command:

```
check_timing -check_only clock_gating_controlling_edge_unknown -verbose
```

## Annotate Delays, Load, and Slews

Model extraction uses the back annotated delay slews and the load information during the extraction process and reflect the effect in the extracted model. Here is a small description how these are used.

### Annotated Delays

The annotated delays using SDF or `set_annotated_delay` command override the calculated delay (from library or RC data) value for the arc; however, the output slew for the arcs is used as calculated. In case of incremental delays the delta delay is added to the calculated arc delay.

### Annotated Slews

Annotated slews are ignored during timing model extraction. These constraints are dumped in assertions file generated with the `-assertions` parameter in `do_extract_model` command.

### Annotated Load

Annotated load will override the pin capacitance defined in the library, and will be used for the C-eff and hence the delay calculation.

**Note:** The annotated delays are generated with a particular context and remain true for that context (transition times) only. So annotating the delays/slews makes the model context dependent. So to create a context independent model it is recommended not to annotate the SDF.

## Design Rules

Model extraction uses the design rules defined in the library. The worst (smaller for max design rules and vice-versa) among all the fanout pins (topologically first level) is used for the input ports and the worst of all the fanin pins (topologically first level) is used for the output ports.

If design rule limits are defined at the pin and library level, the design rule from the pin is chosen, rather than the worst of the pin and library level, because the pin level information overrides the cell level, which in turn overrides the library level information. If design rule violations (DRV) are coming from the constraints , then you can choose them based on the following global variable:

```
set timing_extract_model_consider_design_level_drv {true/false}
```

When set to `false`, this global variable specifies that the user-asserted design level DRVs should not be considered while Modeling DRVs in the extracted timing model.

When set to `true`, the global specifies that the user-asserted design level DRVs should be considered while Modeling DRVs into the extracted timing model

## Create Clocks

All internal pins with the `create_clock` statement will be preserved. The pins will be renamed to the clock name asserted on them and hence the ETM will have the internal pins with same name as that of the clock asserted on it.

### Generated Clocks

Generated clocks defined in a hierarchical block are supposed to be the part of the block itself and do not come from the top level. So ETM preserves generated clocks inside the model itself using Liberty `generated_clock` construct. Here are the rules how model extraction handles the `create_generated_clock`.

- The pin asserted with a generated clock is preserved as an internal pin.
- The name of this pin is changed to the generated clock name.
- The `-source` pin of the generated clock is moved to the master clock root. In case there is an inverter between the master clock root and `-source` pin, then a new pin is preserved with the name `genclk_name_SRC_pin`. This pin is connected to the master clock root via a negative

unit zero delay arc and with the generated clock root via the latency arc.

- In case of generated clocks having multiple masters, the software only picks one of them as master. When the ETM is read, the software gives a warning that you need to correct the clock definition.

For example, if the original netlist contains the following generated clock statement:

```
create_generated_clock -name gclk -source [get_ports clk] -divide_by 2 [get_pins buf1/A]
```

During the extraction process the pin “buf1/A” will be preserved as an internal pin in the library with name `gclk`. The model will look as follows:

```
generated_clock (gclk) {
    master_pin : clk;
    divided_by : 2;
    clock_pin : "gclk ";
}
pin (gclk ) {
    clock: true ;
    direction: internal ;
.
.
}
```

## Handling Typical Clock Definitions

### Multiple Clocks on Same Pin

There are some design scenarios when multiple generated clocks are defined on a single pin with the `create_generated_clock -add` parameter. This asserts multiple clock definitions on the pins. In this case, during model extraction the pin is duplicated with the name of the clock. For example, in a netlist if there are two clock definitions, such as,

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK -
divide_by 2 [get_pins buf1/A]

create_generated_clock -name gclk2 -source [get_ports clk] -add -master_clock CLK -
divide_by 2 [get_pins buf1/A]
```

During the extraction process the pin `buf1/A` will be preserved as an internal pin in the library with a

name gclk1 and gclk2. The model will be as follows:

```
generated_clock (gclk1) {
    master_pin : clk;
    divided_by : 2;
    clock_pin : "gclk1 ";
}
generated_clock (gclk2) {
    master_pin : clk;
    divided_by : 2;
    clock_pin : "gclk2 ";
}
pin (gclk1) {
    clock: true;
    direction: internal;
.....
}
pin (gclk2) {
    clock: true;
    direction: internal;
.....
}
```

## Generated Clocks on Multiple Pins

When a generated clock is defined on multiple pins, the software asserts a single clock definition on multiple pins. To handle this scenario, all the pins asserted with this clock are preserved as internal pins in the model with name [clock\_name\_<count>] and the `generated_clock` construct is written in the model with all the pin names in the `clock_pin` attributes with one space between them. For example, in a netlist having clock definitions on multiple pins.

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK -
divide_by 2 [get_pins {buf1/A buf2/A}]
```

During the extraction process the pin buf1/A will be preserved as an internal pin in the library with name gclk1\_1; the pin buf2/A will be preserved as an internal pin with name gclk1\_2. The model will be as follows:

```
generated_clock (gclk1) {
    master_pin : clk;
    divided_by : 2;
```

```

clock_pin : "gclk1_1 gclk_2 ";
}
pin (gclk1_1) {
clock: true;
direction: internal;
...
}
pin (gclk1_2) {
clock: true;
direction: internal;
.....
}

```

As seen above, if the extracted model is loaded in the default mode then the generated clock definitions will not match one to one with the original netlist.

- In first scenario the second `gen_clock` construct will overwrite the first one, hence creating only last generated clock.
- In the second scenario one generated clock will be created on two internal clock pins.

Due to the above issues, it becomes difficult to have a one to one mapping with the original netlist, and you are required to modify the top level constraints to fit this model. This flow issue can be resolved by setting the following global variable to false

```
set timing_library_genclk_use_group_name false
```

When set to true, the `timing_library_genclk_use_group_name` global variable directs the software to use the `generated_clock` group name as for naming the generated clock. So if the design is as per the above scenario then the following global variable should be set to `true` while loading the model:

```
set timing_prefix_module_name_with_library_genclk true
```

When set to `true`, the software appends the instance name to the clock pin name when creating a generated clock. When set to `false`, the software uses only the clock pin name when creating a generated clock. For example, assume that the cell for instance a/b in the timing library contains the following generated clock group:

```

generated_clock(genclk2) {
clock_pin : clk1;
master_pin : "int/clk";
multiplied_by : 2;

```

```
duty_cycle : 50.0;
}
```

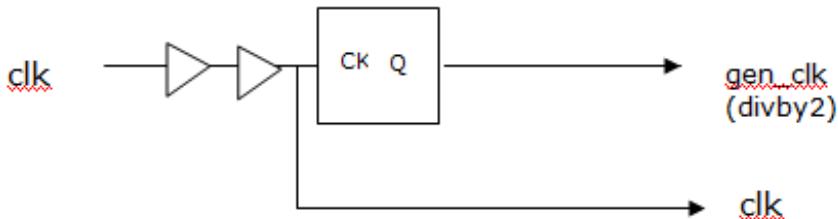
By default (`true`), the software creates a generated clock with the following name:  
`a/b/int/clk`

If you set the above global variable to `false`, the software creates a generated clock with the following name:  
`int/clk`

**Note:** You will need to ensure that the constraints are applied with respect to the generated clock naming, if required. You can remove these library generated clocks and then apply the specified generated clocks so that minimal changes are required in the constraints.

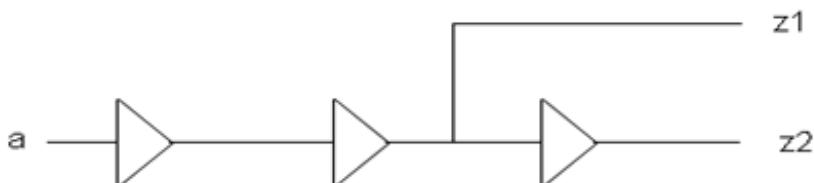
## Clocks with Sequential and Combination Arcs

If there are both sequential and combinational arcs for the same master clock, the clock pin is duplicated with two pins having `_SEQ_pin` and `_COMB_pin` suffix. All the combinational arcs starting from this clock root are bound to the duplicated pin having “`_COMB_pin`” suffix and all sequential arcs are bound to the duplicated pin having “`_SEQ_pin`” suffix.



## Secondary Load Dependent Networks

If there is a timing path from one output pin to another pin, the model extractor considers both primary output loading and secondary output loading when output-to-output delays are computed. In the following figure there is an output-to-output path from `z1` to `z2`:



The extracted timing model for this block consists of two delay arcs - one from `a`  $\rightarrow$  `z1` and another from `a`  $\rightarrow$  `z2`. But now the delay of arc `a`  $\rightarrow$  `z2` also depends on the secondary loading, that is, at port

z1.

This gives rise to a delay arc from a to z2 which is dependent on two loads and one input slew. So this arc will be dumped as a three dimensional delay table because the delay depends on the primary output loading at z2, as well as a secondary output loading at z1.

The table template will be as follows:

```
lu_table_template (lut_timing_1) {
variable_1: input_net_transition;
index_1 ("0...0 1.0 2.0 3.0 4.0 5.0");
variable_2: total_output_net_capacitance;
index_2 ("0...0 1.0 2.0 3.0 4.0 5.0");
variable_3: related_out_total_output_net_capacitance;
index_3 ("0...0 1.0 2.0 3.0 4.0 5.0");
}
```

**Note:** Please note that this is not a good design practice. You should always buffer the port to avoid 3D dependencies. If there is more than a couple of load dependencies, then there will be accuracy loss since model extraction cannot accurately model beyond 3D arcs accurately.

## Characterization Point Selection

Choosing Characterization during point extracting models depends upon the following selection criteria:

If no input\_slews/clock\_slews/output\_load is specified as argument in the do\_extract\_model command then tool find out these characterization points from the design's interface elements. As it is only the interface elements which are to be supplied with the external world's slew/load. This can be understood as follows.

All the check arcs (e.g. clk->in) will be characterized for the slew points as follows:

- Reference slew points will be taken from the slew index of the first element just after the “clk” port.
- Signal slew points will be taken from the slew index of the first element just after the “in” port.

All the Sequential (e.g., clk->out) arcs will be characterized as follows:

- Input slew will be taken from the slew index of the first element just after the \*clk\* port.
- Output load will be taken from the load index of the last element just before the \*out\* port.

Consider following topology in Netlist:

in -----> buf1 -----> ff1 ----->buf2 -----> out

clk -----> clk\_buf ---- ->ff1 ----->buf2 -----> out

A snapshot of the original timing library for the interface elements (i.e., BUF and CLK\_BUF) is as follows:

```
Cell (BUF)
{
  timing ()
  index_1 ("0.0500, 1.4000, 4.5000");
  index_2 ("1.0500, 6.5000, 10.0000");
}

Cell (CLK_BUF)
{
  timing ()
  index_1 ("1.0500, 2.4000, 3.5000");
  index_2 ("0.0500, 4.5000, 5.0000");
}
```

If the `do_extract_model` command is used, then the extracted model will be characterized as follows:

```
Check Arc clk->in
index_1 ("0 0.0500, 1.4000, 4.5000");
```

Here the signal slew is taken from buf1 slew-index in original libraries.

```
index_1 ("0 1.0500, 2.4000, 3.5000");
```

Here the reference slew is taken from clk\_buf slew-index in original libraries.

```
Seq. Arc clk->out
index_1 ("0 1.0500, 2.4000, 3.5000");
```

Input Slew is taken from clk\_buf slew-index in original libraries.

```
index_2 ("0 1.0500, 6.5000, 10.0000");
```

Output load is taken from buf2 slew-index in original libraries.

**Note:** A value of 0 is always added to the characterization points.

# Constraint Generation during Model Extraction

The extracted models need to be validated before they are transferred to other designers to be used for a top level analysis or optimization flow. For validation purposes, we need a subset of the original timing constraints and further a subset of these constraints is needed for fitting the model in a top level netlist.

The model extraction process will output a timing library and two constraints file containing the subset of original constraint. One of these constraint files will be used for a standalone validation of the model compared to the original netlist. By default model.asrt and model.asrt.latchInferredMCP will be written in CWD. if the `do_extract_model -assertions` parameter is not specified, the software will use stand alone validation. If `do_extract_model -assertions test.asrt` command is specified, then 3 assertion files are generated, named `-test.asrt`, `test.asrt.latchInferredMCP`, and `top_model.asrt`. The other constraint file (`top_model.asrt`) will be used when the extracted model will be stitched to a top level netlist. This assertion is generated only when the `do_extract_model -assertions` parameter is specified.

Two separate constraint files are required because:

- For a standalone validation we need all the context parameters of the design.
- For STA or optimization flow when stitched to a top level environment, the context parameter will be automatically coming from the top level. So some of the constraints need to be filtered.

This is what is achieved with an automated procedure executed with every model extraction.

The following are the applications of these constraints in a model extraction flow:

## False Path Exceptions

- If `set_false_path` is applied through some internal pins/ports, then those paths/arcs are removed during extraction.
- If `set_false_path` is applied between clocks, then these paths are removed and these exceptions are saved in a “`top_model.asrt`” file.

## Multi-Cycle Path Exception

If multi-cycle paths through pins/ports are applied, then during extraction, worst paths through them mapped to IO ports are calculated and exceptions through these IOs are dumped in the `top_model.asrt` file.

**Note:** You should examine this constraint file (“`top_model.asrt`”), as these constraints are just indicative of what is needed to be modelled at the top level. In certain situations, it may not be possible to extract all possible exceptions for top level due to limitations of liberty to model them

correctly for a given path in case of conflicts.

### **set\_disable\_timing**

The `set_disable_timing` constraints are handled in timing graph, as the arcs being disabled are not used for the extraction purpose. This makes them abstracted in the model and handling of these constraints is not required. So these constraints are not dumped in any of the constraint files. Also, the path broken by them will not be extracted during extraction.

### **set\_case\_analysis**

The constant values are handled during extraction, as all the arcs connected to such pin/ports behave like a disabled arc. Therefore, case analysis constraints are not needed for any of the two constraint files. The case analysis can be modelled in ETM or in SDC. Please refer to section on Clock Gating Checks.

## **Clock Definitions**

### **create\_clock**

The pins/ports having `create_clock` definitions on them are preserved in the timing model. If a clock is created on some internal pin then the pin name needs to be modified to reflect the instance name of the extracted model. This is achieved using the same variable approach.

```
[get_pins "$ETM_CORE/pin1 $ETM_CORE/pin2 $ETM_CORE/pin3"]
```

When the extracted model is stitched to some top level netlist, then the clock definitions are supposed to come from the top level netlist itself. So that such clock definitions are not needed to be dumped in the top level constraint file, but need to exist in the assertion file for standalone validation.

### **create\_generated\_clock**

The pins having the `create_generated_clock` defined on them are preserved in the extracted model as internal pins. As generated clocks are very much intended for the block itself and are not supposed to be coming from some top level, liberty provides syntax to define the generated clocks in the timing model. You can use the following use model:

```
generated_clock ( gclk ) {
    Clock_pin : gclk ;
    master_pin : clk ;
    multiplied_by : 2 ;
}
```

While loading a model, the software names the generated clocks after their target pin names, so while dumping the generated clocks in the library the target pin name is modified to the clock name itself. This facilitates the same name of generated clock names in the original netlist and the

extracted model. This name changing cuts down the need to modify other constraints (clock uncertainty/path exceptions) related to this clock. In this case it is not required to dump these generated clocks as a separate constraint.

## Arrival and Required Time

The constraints like `set_input_delay` and `set_output_delay` lie in this category. Though these constraints can be applied on ports as well as some internal pins. Such constraints applied on some internal pins is of no use for the extracted models, as in an extracted model only interface timing is considered.

Constraints applied on the ports need not any change as the port will be preserved with the same name. So these constraints are dumped out in the assertion file for validation and do not have any impact on the generated ETM. These need not to be dumped out for the top level constraint file as in a top level the data must be coming to the port from some other top level register or port. So the input delay value is supposed to be the delay of path from top level register/port to this port.

## Global Constraints

### **`set_max_transition` / `set_max_capacitance`**

If these constraints exist in the SDC file for block and the `timing_extract_model_consider_design_level_drv` global variable is set to `false`, then these constraints will not have any impact on the generated model.

If variable “`timing_extract_model_consider_design_level_drv =true`”, then these constraints are considered for while generating timing model. The conservative value of `max_transition`, between defined at library cell associated with port or `set_max_transition` defined in SDC is chosen for all ports. In case of `max_capacitance` conservative value between defined at library cell associated with port or `set_max_capacitance` defined in SDC is chosen for all output ports.

**Note:** Please note that `set_max_capacitance`/`set_max_transition` defined in the SDC is always dumped in side constraints file to be read during validation.

### **`set_load`/`set_resistance` / `set_annotated_transition`**

These constraints can be applied on pins as well as ports. Such constraints applied on internal pins are handled in the extraction flow during delay calculation. So these are included in the model itself. But these constraints applied on the port are treated as the out context environment and need to be dumped out in the constraints file.

### **`set_annotated_delay` / `set_annotated_check`**

This constraint is applied on the internal arcs as incremental or absolute values. Annotated

delays/checks are handled in the extraction flow during delay calculation. So these are included in the model. This constraint is not dumped out in any of the constraints file.

### **set\_input\_transition/set\_driving\_cell**

These constraints are for out context environment and are applied to ports only. As ports are the object list for these constraints you do not need to make any changes in these constraints. These are dumped out in side constraints file.

## **Slew Propagation Mode in Model Extraction**

Model extraction supports two methods of slew propagation - worst slew propagation vs. critical slew propagation (also known as path slew propagation).

### **Worst Slew Propagation**

In worst slew propagation mode, the worst slew of all the incoming arcs at a converging point is taken to propagate further, e.g., if At A input pin of 2 input AND gate, slew index was {1 3 5} and at B pin the slew index was {1 2 6}, so the resulting slew at output will correspond to slew index {3 5 6}, which is worst of all slew index.

### **Path based slew propagation**

In path based slew propagation mode, the actual slew for the path elements is propagated for extracted the arcs, e.g., if At A input pin of 2 input AND gate , slew index was {1 3 5} and at B pin the slew index was {1 2 6}, so the resulting slew index for path through A will be {1 3 5} & through B will be {1 2 6}.

Slew propagation is controlled by the following global variable:

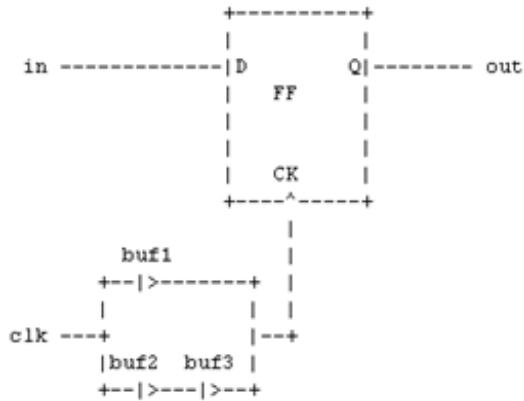
```
set timing_extract_model_slew_propagation_mode {worst_slew | path_based_slew}
```

The default slew propagation mode is `worst_slew`.

## **Parallel Arcs in ETM**

If a mode is extractedl in BC-WC or OCV mode, then timing is taken care fo through early and late path analysis as is done in `report_timing`.

Consider the following design:



In the above design there are two paths  $\text{in} \rightarrow \text{clk}$  (check path) and  $\text{clk} \rightarrow \text{out}$  (Sequential path).

Here the check (setup) path will be captured by the early path of clock (i.e.,  $\text{clk} \rightarrow \text{buf1} \rightarrow \text{FF/CK}$ ). But the sequential path will be using the late clock path (i.e.,  $\text{clk} \rightarrow \text{buf2} \rightarrow \text{buf3} \rightarrow \text{FF/CK}$ ).

The extraction of early/late attributes is differentiated through the attribute `min_delay_arc`. So in the extracted model for combinational/trigger/latency arc, there exist both rise/fall max and min arcs.

## Latency Arcs Modeling

Latency arcs are between a generated clock and its respective master clock. Actual paths are considered while extracting, e.g., consider a generated clock with `-divide_by 2`. The edge relationship of master and generated clock will be “R  $\rightarrow$  R” and “R  $\rightarrow$  F”. No “F  $\rightarrow$  F” or “R  $\rightarrow$  F” arc will be dumped, even if such paths are present in design for data propagation.

In case of Ideal master clock, arcs between master and generated clock source, is controlled through the following global variable:

```
set timing_extract_model_ideal_clock_latency_arc {false/true}
```

When set to `true`, zero delay arc from master clock to generated clock is considered in the extracted model. When set to `false`, this arc is not modelled.

## Latch-Based Model Extraction

Extraction model handles transparent latches during model extraction based on arrival times defined on input ports and specified clock periods, whether a path from an input port to a latch causes borrowing or not.

- If the path causes borrowing then path traversal should continue through the latch until either

a non borrowing latch or edge-triggered flip-flop is encountered. If the path does not cause borrowing then path tracing should be stopped and a setup arc should be extracted relative to the closing-edge of the first (interface) latch.

- The borrowing behavior of latches does not impact the extracted hold arc. So hold arcs should always be extracted relative to the close edge of the first interface register (latch or flip-flop) encountered while tracing paths from input ports.
- Borrowing can result in paths being traced through latches from an input port to an output port. In this case, a delay (comb) arc should be extracted from the input port to output port. In scenarios where path becomes of more than 1 cycle, inferred multicycle paths are dumped in a separate file named as “model.asrt. latchInferredMCP”.
- When tracing paths from interface registers to output ports, transparent latches will be handled by tracing through borrowing latches backwards and generating a sequential delay arc from a clock port to an output port.

**Note:** As all such latch traversal arcs are true for setup analysis, there will be a `set_false_path -hold` statement in the assertion file for any such arcs.

Please also note that file “model.asrt. latchInferredMCP” is an indicative of what assertions you may need to use at the top level. They should be audited before use MCP cannot be inferred in all the cases.

## Stage Weight Extraction in ETM

The AOCV derating and stage weight calculation are supported during model extraction. These stage weights and AOCV derates are used during TOP level analysis with ETM by correctly computing the stage counts and derates of the paths related to ETM. The PBA feature in ETM can be enabled using the following global variable:

```
set timing_extract_model_slew_propagation_mode path_based_slew
```

Before using this global variable, it is mandatory that the AOCV libraries are read in and AOCV analysis is enabled (by using `set_analysis_mode -aocv true`).

## Stage weight setting in ETM

This feature can be enabled by using below command line option:

```
do_extract_model -include_aocv_weights
```

Using this command line option will enable the software to write stage weights on individual arcs.

The weights written in the ETM are not meant to derate the model. The model will be containing the derated delays. The stage weight extracted on arcs will be used to calculate the stage count of chip level paths going across the ETM.

Extraction of stage weight will be done by tracing the minimum stage count path on arc by arc basis. A path-based cell stage count will be printed on all sequential and combinational arcs. The extraction strategy for computing stage weight will be as follows:

The stage weights in the extracted model are dumped as user-defined attributes. This will require defining three attributes at arc level.

```
define ("aocv_weight","timing","float");
define ("clock_aocv_weight","timing","float");
The first attribute will be used for combinational and trigger arc and will be dumped
at arc level
as below.
timing () {
timing_type : combinational;
timing_sense : positive_unate;
aocv_weight : 4;
.
.
.
}
```

The check arcs will be dumped with two separate stage weights for data and clock:

```
timing () {
timing_type : setup_rising;
aocv_weight : 4;
clock_aocv_weight : 3;
.
.
.
}
```

## AOCV Derating Mode

The AOCV derating mode can be specified to path-based or graph-based or none using the following global variable:

```
set timing_extract_model_aocv_mode {graph_based|path_based|none}
```

The default is none.

You can set this global variable as per your requirements. The global values are explained below:

- graph\_based: Delays in ETM are derated using the graph-based stage counts.
- path\_based: Delay in ETM are derated using total path stage count of worst path between those pin pairs.
- none: Models derated delays that contain the effect of user-derate but does not contain effect of AOCV derates.

Before setting this global variable to `graph_based/path_based` it is mandatory that the AOCV libraries are read in and AOCV analysis is enabled (by using `set_analysis_mode -aocv true`).

## Merging model with stage\_weight attribute

Merging timing models containing `stage_weight` attribute on them, are considered pessimistically, so the resulting merged liberty contains min `stage_weight` defined between various input library files.

## PG Pin Modeling During Extraction

The software provides extraction model power/ground information in multiple layers (library / cell / pin) to identify the association of each pin with a power and ground source. Besides, CPF also models power/ground information and overrides in some cases the information available in the dotlib or specifies the information that is missing in the dotlib.

Pre-requisites of PG-based flow are:

- Input CPF and libraries in the flow for generation of power / ground aware ETM generation.
- UPF cannot be directly fed to the software but can be converted into CPF.
- Signal Integrity (SI) analysis uses commands like `set_dc_sources` and may fall back to CDB (whenever the relevant power rail/voltage information is missing in CPF/dotlib) for getting supply information. The CPF or input dotlib will have sufficient information to correctly model ETM.

When `-pg` is used with the `do_extract_model` command, power/ground related pin information will be extracted inside the dotlib.

## Various Type PG Modeling in ETM

### Modeling of voltage maps

Voltage maps definitions are typically modeled inside the Liberty .lib format at library scope level. It specifies the voltage value associated with the power rail name. It is possible that these voltage values can be overridden through CPF modeling. User may also add additional power / supply nets.

Model extractor would look at the voltage maps in the original library set and in CPF that user has fed in to the tool and would extract this information from there and model them at library scope level in the ETM.

```
voltage_map (<voltage_name>, <voltage_value>);
voltage_map(<voltage_name>, <voltage_value>);
```

#### 16.1.2 Modeling of Power / Ground Pins

At cell level, pg\_pin groups will be modeled in the ETM in the following fashion:

```
pg_pin (<pg_pin_name_p1>) {
    voltage_name : <voltage_name_p1>;
    pg_type : <type _value>
}
```

The pg\_pin group will be written using using the power rail voltage associated with the net connected to the port. PG pin name formulation will be formed as follows:

`powerDomainName_VoltageRailName_VoltageValue`

### Modeling associated power and ground pins of a logic pin

This information will be written for all the logic input /output / in-out pins that are written in the ETM. For internal pins that are preserved inside the ETM dotlib (e.g., a pin on which generated clock was asserted), no such information will be retained.

```
pin (<signal_pin>) {
...
related_power_pin: <power_rail_name>
related_ground_pin: <ground_rail_name>
}
```

For ports, that are written as pins in ETM, the associated pin (driver or receiver of a net) will be checked and printed in the same way as related\_power\_pin and related\_ground\_pin information.

## ETM Merging Requirements for Power/Ground Aware ETMs

- Voltage maps from two libraries will be merged successfully as long as the voltage rail names are distinct
- pg\_pins will be merged successfully as long as power rail names and the pg\_type match.
- Related\_power\_pins and related\_ground\_pins will be merged as long as they are available in all the libraries and refer to the same power rail name.

## Extracting Timing Models with Noise (SI) Effect

To generate a timing model with SI effects contained within a block, you can use the following flow for the block under consideration

- Load the database and relevant information for performing STA/SI analysis.
- Perform SI analysis, or if you have an incremental SDF from the previous SI run just load the incremental SDF.
- Run model extraction. During this phase, the incremental delays will be added to the computed base delays for characterization of the dotlib table. When the final dotlib is written, it contains the effect of the SI analysis in terms of characterized delays.
- Run `save_cdb` command to generate a CDB for the block under consideration. It generates a SPICE file and block TCL file which can be given as inputs to `make_cdb` for generating a cdb noise library.

The block ETM can then be instantiated in the top level flow and the respective CDB file can be loaded for performing the top level analysis with ETM.

## Merging Timing Models

The software supports merging library models through `merge_model_timing` command. You can generate ETM in various modes in a single library that can be used for Timing Optimization during design flow.

Some Considerations while merging models as given below:

- All the input timing models should have the same cell name.

- If the timing arcs are not comparable i.e. the number of index values differ, then they are written as separate timing arcs with separate mode definitions.
- Boundary pins of single mode libs should be equivalent in number and name while merging. An error message will be flagged if that is not the case.
- If we have generated clock with same name, as well as same definition in ETM libraries, we preserve just this clock in the merged lib, without issuing any ERROR message.
- If you have generated clock with same name, but different divided\_by /multiply\_by / master\_pin / clock\_pin definition, we issue ERROR message stating that “this genclk is defined with different specification, so we cannot merge them.
- If there is a mismatch in timing arcs among the input libraries, then the merged library will contain the union of all the timing arcs with respective mode. e.g., if the first library has only setup arcs and the second library has only hold arcs then the merged library will have both setup and hold arcs with respective modes appended.
- If there are mismatch in internal pins among input libraries i.e., if a particular internal pin is missing in one or more libraries then that internal pin and its timing arcs will be part of merged library with respective mode.
- For both max and min DRV, use the most conservative value. Minimum value from max DRVs will be used and maximum value from min DRVs will be used in merging.
- When same modes are specified in the modes list for two libraries e.g., `merge_model_timing -library_files "setup.lib hold.lib" - modes "M M" -mode_group "MG"`, then the arcs from the two libraries will be assigned same mode i.e., “M” while merging.

## Limitations of ETM

- Transformation of path exceptions to ports in certain cases may not be possible and hence you are advised to review them before use at top level.
- Slew/delay dependency on side inputs will be lost.

In path based slew propagation, only the path based behavior is preserved. For example, after model extraction, the path delay from input i1 to output z does not depend on slews or delays at side input i2, as shown below.

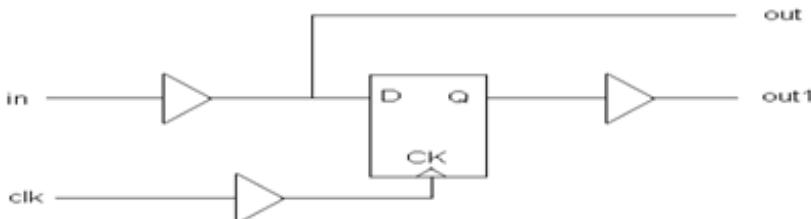


Also, in worst slew propagation mode, the slew at the convergence point is calculated as max assuming the same slew at the input points. See the figure below:



Assuming slew (in1) =slew (in2) and computing max (slew (in1->x), slew (in2->x)).

- Output load dependent check paths lose the load dependency in the extracted model. The check value for the arc is extracted at the current load value. See the figure below:

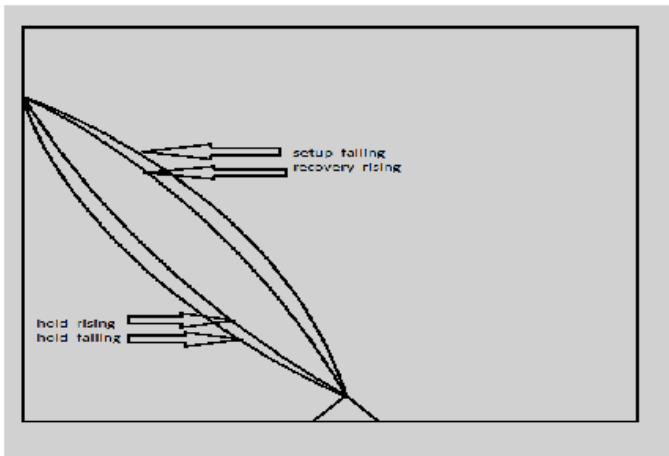


In this figure, the slew at point D will be depending on the loading at port out. Hence, the check arc value will essentially depend on the out loading. Extracted model currently does not support the 3D arcs for setup checks. In this case the check arc value will be calculated using the current loading at the out port.

- Three state enable or disable arcs are not preserved as the three state arcs. As far as three state enable and disable arcs and conditional arcs are concerned, the extracted model is a snap shot of the original block with respect to the constants that are supplied to its inputs. For the given set of constraints, the three state enable or disable expressions are evaluated and three state arcs with transitions to and from the high impedance state Z (0->Z, 1->Z, Z->0, or Z->1) are transformed to combinational arcs with transitions 0->1, 1->0, 1->0, or 0->1, respectively.
- In cases where multiple Input/output delays are applied, the worst arrival path will be same for all output delays. However if some MCP applied through some pin and to one of the output delay clock then there can be two different paths. ETM will only capture the worst slack path and for other path slack might not match when stitched to top.

- If “- reference\_pin” option in set\_input\_delay/set\_output\_delay, reference pin is not preserved, thus latency value of this reference pin is lost.
- If set\_min\_delay/set\_max\_delay/set\_input\_delay/set\_output\_delay is applied on internal pins, then these internal pin will become start & end points, which is not a good way to model them, as top cell is not interacting with these internal start/end points.
- By default, the clock-gating arcs will be extracted as nochange arcs. nochange arc is a combination of one setup and one hold arc. So every rise\_constraint of hold\_falling should have counterpart fall\_constraint of setup\_rising. Cases where the counterpart of hold clock gating is not found (due to false path/disable timing or any other reason), such arcs are written as normal hold arcs. This may cause different phase shift in some cases.
- If ETM extracts partial hold arcs due to false path on setup or selection of recovery checks due to worst slack. These partial checks should be checked considering the same edge for setup, However if the same port have a setup arc of opposite edge, these partial hold checks end up using the opposite edge of the setup arc. Below is the design topology for more details:

If you extract the model for the above scenario you will get an ETM as shown below:



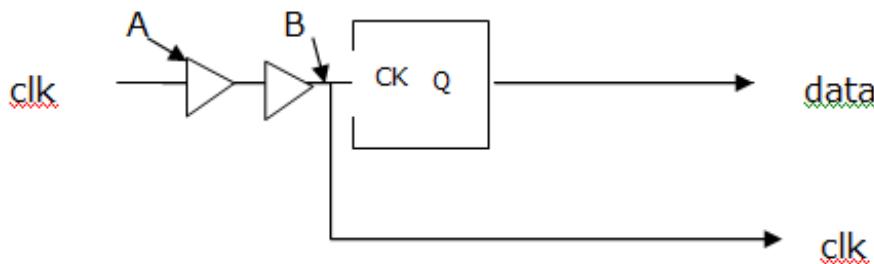
Here is the ETM for max type check for setup\_falling and recovery\_rising check arc between the data and reference pin.

For min type check there are hold\_rising and hold\_falling check arcs.

When this ETM is evaluated by the timer, since there is no rising edge setup between these two pins, the phase shift for hold\_rising arc will be calculated based on the falling edge of setup, hence

giving a wrong phase shift. Similar scenario may occur, in case of clock\_gating hold checks.

- There are some design scenarios, as described below where a generated clock is going out & data is launched wrt this clock only. Now while extracting model, we extract definition point of generated clock (A here), & latency arcs are extracted wrt this point only. In this process Adjusting CPPR calculation till point B gets lost.



- In AOCV mode for pin arcs, even if mode is set to PBA you need to write the GBA count, as there will be more than one path from this pin. This again will cause pessimism.
- For stage weight in combinational/trigger arcs, worst of rise/fall of arc will be taken for analysis, thus adding some pessimism.
- When `timing_extract_model_write_clock_checks_as_arc` global variable is set to `false`, scalar modelling will be done for clock-style checks (pulse-width/period checks). If checks corresponding to both rise and fall transitions are present in the design, they will be modeled as duplicate entries in ETM due to limitations of scalar modeling, as shown below:

```
pin (clk3_clk) {
    clock : true ;
    direction : input ;
    min_period : 2325.3999;
    max_period : 2315.5000;
    min_pulse_width_low : 538.7003;
    max_pulse_width_high : 865.9001;
}
```

To avoid duplicate entries we can model these checks as arcs, where rise and fall transition checks can be modeled as two different arcs. These arcs will be honored by the timer depending on the context, and thus will be more accurate.

```

pin (clk3_clk ) {
    clock : true ;
    direction : input ;
    timing() {
        timing_type : minimum_period ;
        rise_constraint (lut_timing ){
            values(\"
                " 2325.7998, 2325.5999, 2325.7000, 2325.6001, 2325.5000" \
            );
        }
        fall_constraint (lut_timing ){
            values(\"
                " 2315.7000, 2315.6001, 2315.5000, 2315.7998, 2315.8003" \
            );
        }
        related_pin :" clk3_clk ";
    }
    timing() {
        timing_type : min_pulse_width ;
        rise_constraint (lut_timing){
            values(\"
                " 764.3000, 763.9999, 763.0001, 760.6000, 755.8000" \
            );
        }
        fall_constraint (lut_timing){
            values(\"
                " 358.9002, 359.2000, 359.9000, 362.2003, 367.1001" \
            );
        }
        related_pin :" clk3_clk ";
    }
}

```

Arc-based modeling of clock-style checks can be enabled by setting `timing_extract_model_write_clock_checks_as_arc` global variable to `true`.

- There is a possibility of difference in arcs across various versions of the software. Here are some of the reasons for missing arcs:
  - Clock gating checks are modelled as nochange arcs in new versions, whereas earlier they were modelled as regular setup checks.

- Latency arcs are modelled differently across different versions.
- There is a possibility of difference in arcs across various versions of the software. Here are some of the reasons for missing arcs:
  - Clock gating checks are modelled as nochange arcs in new versions, whereas earlier they were modelled as regular setup checks.
  - Latency arcs are modelled differently across different versions.. .

## Validation of Generated ETM

### Validation Flow for Non-MMMC Designs

Extracted models need to be validated before stitching to top level for their accuracy and coverage. The software provides an automated flow for validating the extracted models.

Block-level validation can be achieved using the extracted timing model against the original gate-level netlist, or compare any two valid timing models in the model extraction system, by using the following commands:

- write\_model\_timing
- compare\_model\_timing

The following is the validation flow:

**Step1:** Generating timing model/validation module (using `do_extract_model`) and write model timing report from original session as described below.

```
loadConfig design.conf
do_extract_model top1.lib -verilog_shell_file test.v -verilog_shell_module test -
assertions test.asrt
write_model_timing -type slack ref.rpt
saveConfig test.conf
exit
```

**Step2 :** Loading validation module containing ETM, generating `write_model_timing` report for the ETM session and then comparing reports with reference session using `compare_model_timing` as described below.

```
loadConfig test.conf
setVar rda_Input(ui_timelib) "./data/cell_w.lib"
setVar rda_Input(ui_timelib,max) "./top1.lib"
```

```

setVar rda_Input(ui_timelib,min) "./top1.lib"
setVar rda_Input(ui_netlist) "./test.v"
setVar rda_Input(ui_topcell) {test}
setVar rda_Input(ui_timingcon_file) "./test.asrt"
commitConfig
write_model_timing -type slack comp.rpt
compare_model_timing -ref ref.rpt -compare comp.rpt -outFile final.rpt -
percent_tolerance 3 -absolute_tolerance [expr 0.30/$timeUnit]

```

The details of all these steps are described below:

### **Model Extraction for validation**

Model extraction for validation is done by following command:

```
do_extract_model top.lib -verilog_shell_file test.v -verilog_shell_module test-
assertions test.asrt
```

The following are the outputs:

- verilog\_shell\_file (test.v): verilog having block instantiated in it. In case the block constraints file have command `set_driving_cell`, then cell used there will be instantiated in test.v
- verilog\_shell\_module(test): this will be the module name of test.v
- top.lib: this is resulting ETM
- test.asrt: this will be the resulting exception file to be used in validation flow.

### **write\_model\_timing**

The `write_model_timing` command writes a report on the interface timing of a specified netlist or model. This report includes the worst-case slacks or timing arc values for various types of paths, the transition times at the output ports, the lumped and extracted capacitance on all ports, and the design rules on the ports. It considers the input, output, and combinational paths, but not register-to-register paths. The use-model is as follows:

```
write_model_timing -type slack ref.rpt
```

### **compare\_model\_timing**

The `compare_model_timing` command compares two reports generated by the `write_model_timing` command. This command should be used to specify the reference file, the comparison file, and the allowed tolerance levels that trigger comparison failures. If the timing parameter values in the two files are the same or within the specified tolerance, the result is pass. Otherwise, the result would be reported as fail. The use model is as follows:

```
compare_model_timing -ref ref.rpt -compare comp.rpt -outFile final.rpt -
```

```
percent_tolerance 3 -absolute_tolerance [expr 0.30/$timeUnit]
```

## Validation Flow for MMMC Designs

### Extraction

Considering one MMMC design has two active analysis views, you can use the following flow.

1. Load the design using the following command.

```
loadConfig case1.conf (viewDefinition.tcl is defined in the conf file)
```

2. The following command puts the model extractor in the worst slew propagation mode. It is the default slew propagation mode for model extraction. User can also set the slew propagation mode as path based.

```
set timing_extract_model_slew_propagation_mode worst_slew
```

3. In MMMC configuration, for generating ETM, only one view should be active, which is done by::

```
set_analysis_view -setup {$viewName} -hold {$viewName}
```

4. The model is extracted using the do\_extract\_model command with various relevant command line options.

```
set_analysis_view -setup {view1} -hold {view1}
do_extract_model my_view1.lib -lib_name extracted_model -cell_name extracted_cell -
tolerance 0.0 -verilog_shell_file top.v -verilog_shell_module test_top -view view1
write_model_timing -type slack netlist_view1.rpt -view view1

set_analysis_view -setup {view2} -hold {view2}
do_extract_model my_view2.lib. -lib_name extracted_model -cell_name extracted_cell -
tolerance 0.0 -verilog_shell_file top.v -verilog_shell_module test_top -view view2
write_model_timing -type slack netlist_view2.rpt -view view2
exit
```

The last step is the validation report generation for each analysis view. The write\_model\_timing command with the -view option will generate a report file that will contain the interface timing information for this design for the specific view. First you should save the timing for the original netlist and then this will be used to compare with the report file generated with the model instantiation.

## Validation Flow

For the validationflow, you need to prepare two configuration files with the same settings for the time unit/cap unit/ default values, same as the original design. For the first .conf file, the verilog file will be “top.v”, the library file will be “my\_view1.lib”, the top cell name will be “test\_top”, and the timing constraint file will be “model.asrt.view1”.

For the second .conf file, the verilog file will be “top.v”, library file will be “my\_view2.lib”, the top cell name will be the “test\_top”, and the timing constraint file will be “model.asrt.view2”.

Now, these configuration files will be loaded in a new software session. The `write_model_timing` command is run so that the interface timing for the model can also be dumped out. The use model is as follows:

```
loadConfig top1.conf
write_model_timing -type slack model_view1.rpt
freeDesign
loadConfig top2.conf
write_model_timing -type slack model_view2.rpt
```

You need to compare the timing information for both the original design as well as the extracted model. The comparison is carried out using the `compare_model_timing` command as follows.

```
compare_model_timing -ref netlist_view1.rpt -compare model_view1.rpt -ignore_view -
outFile view1.diff -percent_tolerance 2 -absolute_tolerance 0.003
compare_model_timing -ref netlist_view2.rpt -compare model_view2.rpt -ignore_view -
outFile view2.diff -percent_tolerance 2 -absolute_tolerance 0.003
```

## Validation Reports

### **write\_model\_timing report**

A `write_model_timing` report contains the following timing view properties to capture the timing characteristics of design being extracted as timing model. A `write_model_timing` report for a sample design below would look like.

The report has the following components:

#### **Slack or Arc Value**

This section reports the worst-case slack or arc value for each path from input port to clock, from clock to output port, and from input port to output port.

#### **Transition Time**

This section reports the actual transition time at each port for the four delay types: min\_fall, min\_rise, max\_fall, and max\_rise.

## Capacitance

This section reports the maximum total (lumped) capacitance at each port, and if available, the effective capacitance.

## Design Rules

This section reports all design rules that apply to each port, including maximum capacitance, minimum capacitance, maximum transition time, maximum fan-out (for input ports), and fan-out load (for output ports).

```
#####
# write_model_timing report
#####
# Section 1 : Worst Case Arcs
# Analysis Mode : setup
# Description : worst case arcs
#
# From To Arc Type Transition Arc-Delays
#####
in out max_combinatorial fall/fall 9.857
in out max_combinatorial rise/rise 9.612
in out2 max_combinatorial fall/fall 4.811
in out2 max_combinatorial rise/rise 4.111
clk out2 max_combinatorial fall/fall 3.210
in1 clk setup fall/rise 1.200
in1 clk setup rise/rise 1.782
clk out1 max_sequential rise/fall 3.412
clk out1 max_sequential rise/rise 3.202
#####
# Section 2 : Transition time
# Analysis Mode : setup
# Description : Actual transition times on the ports
#
# Port RiseTran FallTran
#####
in 0.120 0.120
in1 0.120 0.120
clk 0.120 0.120
out 0.056 0.051
```

```

out1 0.056 0.051
out2 0.056 0.051
#####
# Section 3 : Capacitance
# Analysis Mode : setup
# Description : Total Cap on the ports
#
# Port CTotal(Rise) CTotal(Fall)
#####
in 0.002 0.002
in1 0.002 0.002
clk 0.002 0.002
out 0.000 0.000
out1 0.000 0.000
out2 0.000 0.000
#####
# Section 4 : Design Rules
# Analysis Mode : setup
# Description : Design Rules on the ports
#
# Port MaxCap MinCap MaxTrans MaxFanout Fanout Limit
#####
in NA NA 4.500000 NA NA
in1 NA NA 4.500000 NA NA
in1 NA NA 4.500000 NA NA
out NA NA 4.500000 NA NA
out1 NA NA 4.500000 NA NA
out2 NA NA 4.500000 NA NA

```

## **compare\_model\_timing report**

The `compare_model_timing` report shows the comparison results for individual paths, ports, and timing parameters.

By default, the `compare_model_timing` command compares all the parameters contained in the timing interface reports. The resulting comparison report has the same sections as the interface timing report: slack, or arc value, transition time, capacitance, design rules.

There are two tolerance settings, called the

- absolute tolerance

- percentage tolerance

The absolute tolerance setting specifies the absolute amount of difference for slack and arc value comparisons, in library time units such as nanoseconds.

The percentage tolerance setting specifies the percentage amount of difference for slack value and arc value comparisons. A setting of 1.0 means a difference of 1 percent.

### Absolute Tolerance

In an example if the absolute tolerance value specified is 0.2 time units. This means that the result of subtracting the comparison value from the reference value must be between -0.2 and +0.2 in order to pass the comparison test. For example, if the slack in the reference file is 5.0 time units, the slack in the comparison file must be less than 5.2 and more than 4.8 time units for the comparison to pass.

### Percentage Tolerance

In an example if the percentage tolerance is set to a single number, 1.0, which represents plus or minus 1.0 percent of the slack value or arc value in the reference file. If the value in the comparison file is outside of this range, the result is a comparison failure. For example, if the arc value in the reference file is 10.0, the arc value in the comparison file must be between 9.9 and 10.1 for the comparison to pass.

A sample report is shown below:

```
#####
# compare_model_timing report
#####
#####
#Section 1: Worst-case Arc-Delay
#Tolerance Percent: 1%
#Tolerance Absolute: 0.005
#NOTE: The various columns list values in the form 'X[Y]', where:
# X = value that is compared.
# Y = reference value.
#NOTE2: If a comparison fails (ie. status = FAIL), then the clauses
# (i.e. 'X[Y]' pairs) that caused the failure are identified with
# a '*' in front of them.
#From To Arc-Type Transition Arc-Delay abdDiff %Diff Status
#####
clk out1 max_sequential rise/fall 3.412[3.412] 0.000 0.000% PASS
clk out1 max_sequential rise/rise 3.203[3.202] 0.001 0.031% PASS
clk out2 max_combinational fall/fall 3.210[3.210] 0.000 0.000% PASS
in out max_combinational fall/fall 9.854[9.857] -0.003 -0.030% PASS
in out max_combinational rise/rise 9.613[9.612] 0.001 0.010% PASS
in out2 max_combinational fall/fall 4.815[4.811] 0.004 0.083% PASS
```

```
in out2 max_combinatorial rise/rise 4.112[4.111] 0.001 0.024% PASS
in1 clk setup fall/rise 1.200[1.200] 0.000 0.000% PASS
in1 clk setup rise/rise 1.782[1.782] 0.000 0.000% PASS
#####
#Section 2: Pin-capacitances
#Tolerance: 1%
#NOTE: The various columns list values in the form 'X[Y]', where:
# X = value that is compared.
# Y = reference value.
#NOTE2: If a comparison fails (i.e. status = FAIL), then the clauses
# (i.e. 'X[Y]' pairs) that caused the failure are identified with
# a '*' in front of them.
#
#Pin C(total) [rise] C(total) [fall] Status
#####
clk 0.002[0.002] 0.002[0.002] PASS
in 0.002[0.002] 0.002[0.002] PASS
in1 0.002[0.002] 0.002[0.002] PASS
out 0.000[0.000] 0.000[0.000] PASS
out1 0.000[0.000] 0.000[0.000] PASS
out2 0.000[0.000] 0.000[0.000] PASS
#####
#Section 3: Transition-times
#Tolerance: 1%
#NOTE: The various columns list values in the form 'X[Y]', where:
# X = value that is compared.
# Y = reference value.
#
#NOTE2: If a comparison fails (i.e. status = FAIL), then the clauses
# (i.e. 'X[Y]' pairs) that caused the failure are identified with
# a '*' in front of them.
#
#Pin Rise Fall Status
#####
clk 0.120[0.120] 0.120[0.120] PASS
in 0.120[0.120] 0.120[0.120] PASS
in1 0.120[0.120] 0.120[0.120] PASS
out 0.056[0.056] 0.051[0.051] PASS
out1 0.056[0.056] 0.051[0.051] PASS
out2 0.056[0.056] 0.051[0.051] PASS
#####
#Section 4: Design Rules
```

```
#Tolerance: 1%
#NOTE: The various columns list values in the form 'X[Y]', where:
# X = value that is compared.
# Y = reference value.
#NOTE2: If a comparison fails (i.e. status = FAIL), then the clauses
# (i.e. 'X[Y]' pairs) that caused the failure are identified with
# a '*' in front of them.
#
#Pin MaxCap MinCap MaxTrans MaxFanout FanoutLoad Status
#####
in NA[NA] NA[NA] 4.500000[4.500000] NA[NA] NA[NA] PASS
in1 NA[NA] NA[NA] 4.500000[4.500000] NA[NA] NA[NA] PASS
in1 NA[NA] NA[NA] 4.500000[4.500000] NA[NA] NA[NA] PASS
out NA[NA] NA[NA] 4.500000[4.500000] NA[NA] NA[NA] PASS
out1 NA[NA] NA[NA] 4.500000[4.500000] NA[NA] NA[NA] PASS
out2 NA[NA] NA[NA] 4.500000[4.500000] NA[NA] NA[NA] PASS
#####
# SUMMARY
#
# Check-Type Passed Failed N/A Total
#####
Worst-Slacks 9 0 0 9
Capacitances 6 0 0 6
Trans-Times 6 0 0 6
Design-Rules 6 0 0 6
```

## ETM Extremity Validation

The ETM models the timing arcs in the design for a range of discrete input-slews/output-loads. The slew/load asserted using SDC at the block-level during ETM characterization represents the anticipated context of the ETM instance at the top-level. However, the current validation of the ETM is done only for the input-slews/output- loads that have been asserted on the design during the ETM extraction. Therefore, the current validation of ETM is incomplete in the sense that the validation is not being done for a complete range of slew/load points for which ETM has been characterized.

Additionally, in GBA mode the problem of ETM validation is computationally more difficult. In GBA mode, the assertion of a slew on a particular input port may have an impact on the timing of a path starting from some other input port. Therefore, different combinations of input slews at the input ports will result in different timing behavior of the block in GBA mode. Since there can be exponentially large number of combination of slews at the input ports, validation of ETM in GBA mode is computationally more difficult.

The “Extremity-Validation” of ETM (also referred as EV\_ETM) validates ETM at the minimum/maximum value of the slew at the input ports and minimum/maximum value of the output load at the output ports, thereby validating the ETM to a greater extent of the possible scenarios at the top. The minimum/maximum slew at the input port is defined as the minimum/maximum slew for which that input port has been characterized in the ETM library. Similarly, the minimum/maximum load at the output port is defined as the minimum/maximum load for which that output port has been characterized in the ETM library.

The minimum/maximum value of the slew/load will yield four corners of the ETM context, namely:

1. Corner 1 (Fast): the minimum slew at the input ports and the minimum load at the output ports
2. Corner 2: the minimum slew at the input ports and the maximum load at the output ports
3. Corner 3: the maximum slew at the input ports and the minimum load at the output ports
4. Corner 4 (Slow): the maximum slew at the input ports and the maximum load at the output ports

The validation flow in exhaustive mode, is same as that in non-exhaustive mode, that is,

1. `do_extract_model`
2. `write_model_timing` at block level
3. read ETM at top level
4. `write_model_timing` at top level
5. `compare_model_timing`

This flow is controlled by the `timing_extract_model_exhaustive_validation_mode` global variable.

Additional files generated in EV-ETM (but not generated in normal validation) will be stored in the current working directory by default. However, location to place these files can be controlled by the `timing_extract_model_exhaustive_validation_dir` global variable. These files are written as hidden files.

Also note that the `timing_extract_model_exhaustive_validation_dir` global should point to the same directory at the block and top level. When this global is not used at the block level, then the EV-ETM files (in step 1 and 2 above) will be dumped in the working directory at the block level. If the working directory at the top level is different than that at the block level, then at the top level `timing_extract_model_exhaustive_validation_dir` global variable should point to the block level working directory.

The `compare_model_timing` will compare the `write_model_timing` report written at the block level with the corresponding top-level report. The output of `compare_model_timing` command, corresponding to four corners is written to a file named: `<output_file_name>_ev`, where `<output_file_name>` is the name of the output-file specified using

the `compare_model_timing` command. The results of four different corners are written in four different columns as shown below:

Slack (SS) / Status Transition	Slack (SF) / Status Arc-Type	Slack (FS) / Status From To	Slack (FF) / Status
#####	#####	#####	#####
7.511[7.511]/PASS rise/rise	7.511[7.511]/PASS setup CLK1	7.466[7.466]/PASS CLOCK1	7.466[7.466]/PASS
9.666[9.665]/PASS fall/rise	9.666[9.665]/PASS setup CLK3	9.336[9.321]/PASS CLOCK2	9.336[9.321]/PASS

In this mode, there is no change in the use models of `do_extract_model`, `write_model_timing`, and `compare_model_timing` commands. The behavioral difference (between default validation and EV\_ETM) is the creation of additional four files corresponding to the four corners.

If the slew propagation is set to worst, then the `do_extract_model` and `write_model_timing` commands will issue a warning.

## Limitation/Implications of EV-ETM

The EV-ETM flow will have the following limitations/implications:

- The runtime of validation will appreciably increase.
- The EV-ETM will be supported only in the path\_based mode and not in worst\_case mode.
- The EV-ETM, though validates the ETM at the corners, cannot be considered as fully rigorous. For example, the proposed methodology does not validate the timing obtained by interpolation of the tables characterized in ETM.

# Prototyping Flow Capabilities

---

- Creating An Initial Floorplan Using Automatic Floorplan Synthesis
- Using Trial Route for Congestion and Timing Analysis
- What-If Timing Analysis
- Fast Slack Timing Analysis
- Prototyping Methodologies



# Creating An Initial Floorplan Using Automatic Floorplan Synthesis

- [Overview](#)
- [Data Preparation](#)
- [Importing the Design](#)
- [Setting Automatic Floorplan Synthesis Global Parameters](#)
- [Creating an Initial Floorplan](#)
- [Creating Floorplan for Hierarchical Design](#)
- [Creating Multiple Alternative Floorplans](#)
- [Analyzing the Floorplan](#)
- [Adjusting Macro Placement](#)
- [Saving the Floorplan](#)

## Overview

Automatic Floorplan Synthesis (previously called Masterplan) is a set of natively-integrated automatic floorplan capabilities that can create a quick, prototype floorplan. Given a gate-level netlist and design physical boundary, Automatic Floorplan Synthesis can analyze the signal flow and generate a floorplan that includes automatic module and macro placement for large chips.

By default, Automatic Floorplan Synthesis takes timing constraints into account during floorplan generation. The advantage of using Automatic Floorplan Synthesis is that you can quickly create multiple alternative floorplans. You can then test the floorplans to find the one that gives you the best placement and routing results, and use it as a starting point for making the final floorplan.

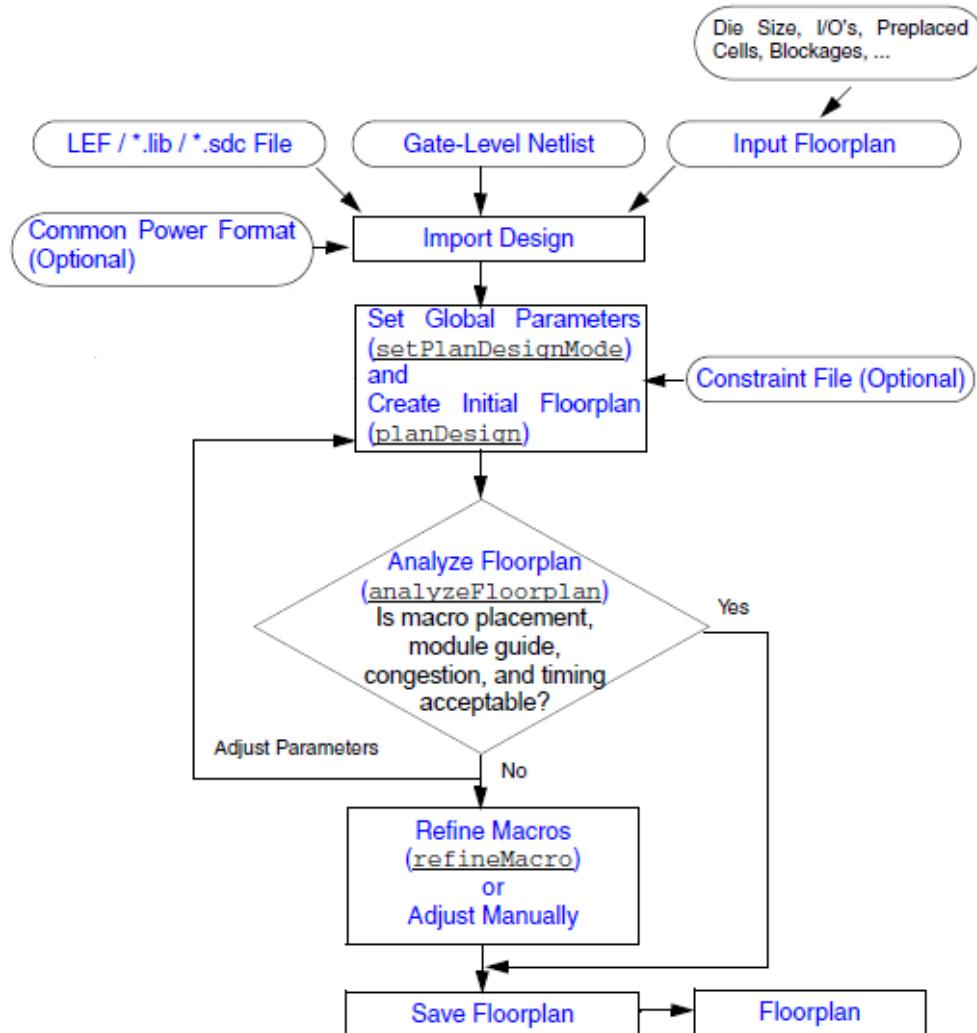
Ideal designs for use with Automatic Floorplan Synthesis are:

- Designs with hierarchical logical netlists
- Large designs that contain more than 50 hard macros
- Designs in which no more than 50 percent of the chip area is made up of hard macros

## Automatic Floorplan Synthesis Flow

The following figure, shows the typical task flow for using Automatic Floorplan Synthesis to create an initial floorplan:

### Automatic Floorplan Synthesis Flow



## Data Preparation

The Automatic Floorplan Synthesis uses the following input files when creating floorplans:

- Gate-level netlist
- LEF file that contains models for standard cells, hard macros, I/O pads
- Floorplan file (or DEF file) that defines:
  - Die size and core area
  - Fixed I/O pad and pin locations
  - Any preplaced hard macros or placement blockages
  - Power stripes and rings (optional)

**Note:** Cells with direct I/O connections should be preplaced around the chip boundary before running the Automatic Floorplan Synthesis feature. These include Jtag cells, boundary scan cells, and I/O interface logic cells. Automatic Floorplan Synthesis automatically ignores high fanout global nets, such as clock, reset, scan, and enable.

Use the following commands to preplace Jtag and boundary scan cells:

`specifyJtag`

`placeJtag`

- Common Power Format (optional)
- Constraint file (optional)

## Selecting Seeds

Seeds are typically design blocks that represent functional units. For example, USB controllers, PCI controllers, Cache sub blocks, and FPUs make good seed candidates. For most designs, hierarchical modules make good seed candidates if they represent the functional units in the design. Hierarchical modules (soft seeds) are not the only candidates for seeds; a seed can also be a hard macro (hard seed) or an instance group made up of strongly connected instances.

Automatic Floorplan Synthesis analyzes the data flow between seeds (design blocks) based on their connectivity and their location. It then places the seeds in the core area in a way that minimizes wire length and congestion. You should select seeds that identify the data flow in the design so that when `planDesign` is run, Automatic Floorplan Synthesis can optimize your data flow to reduce overall interconnect and placement.

Seed selection and seed location also influence how Automatic Floorplan Synthesis places hard macros. Seeds eventually become module guides in the Automatic Floorplan Synthesis generated

floorplan.

There are two methods for selecting seeds:

- Automatic seed selection
- User-Specified seed selection

## Automatic Seed Selection

By default, Automatic Floorplan Synthesis selects seeds using the following methods in the specified order:

1. Chooses modules that fit an internally calculated size range
2. Chooses large hard macros as seeds
3. Groups small modules or single standard cells to fit an internally calculated size range

However, you can force automatic seed selection to favor a certain constraint during the seed selection process by specifying one of the following `setPlanDesignMode` parameters: -

`setSeedHierLevel`, `-numSeed`, or `-seedSize`.

For example, some designs might not require the Automatic Floorplan Synthesis feature to look through their entire hierarchies to select seeds. Going down two to three levels from the top in the hierarchy might be enough to select good seeds. In this case, you can force Automatic Floorplan Synthesis to favor logic level when selecting seeds by typing the following commands:

```
setPlanDesignMode -setSeedHierLevel 3  
planDesign
```

**Note:** After automatic seed selection, Automatic Floorplan Synthesis writes out a seed file called `MP_seed` under the current directory.

## User-Specified Seed Selection

You can provide your own choice of seeds to influence the macro placement and module guide generation of Automatic Floorplan Synthesis. You can select seeds based on a module's function, size, connectivity, or any combination of the three.

For most designs, the optimal number of seeds to select is between 20 to 50. You should not select fewer than 10 seeds, or more than 100. You do not have to provide a complete list of seeds for the

design. Automatic Floorplan Synthesis will select the remainder of the seeds required for the design size, and will attempt to select seeds that match the size of the ones you selected.

To provide Automatic Floorplan Synthesis with your choice of seeds, you must create a seed section in the constraint text file. This seed section lists the names of the hierarchical modules, hard macros, and instance groups that you picked as seeds. You also can specify utilization values for individual modules or instance groups.

You can specify a seed section using the following format:

```
BEGIN SEED
name=seedname [util=utilization_value]
[createFence=true] [minWHRatio=value]
[maxWHRatio=value] [minFenceToFenceSpace=value]
[minFenceToCoreSpace=value] [minFenceToInsideMacroSpace=value]
[minFenceToOutsideMacroSpace=value] [minInsideFenceMacroToMacroSpace=value]
[master=seedname] [cloneOrient={R0|MX|MY|R180}]
END SEED
```

Where:

name=seedname	Specifies the name of the hierarchical module, hard macro, or instance group that you want to choose as a seed.  If you specify the <code>createFence=true</code> then the seedname must be a hierarchical module.
util=utilization_value	Specifies the utilization value for the specified module or instance group.
createFence=true	Indicates that a fence should be generated for the hierarchical module.
minWHRatio=value	Specifies the minimum width to height ratio.  <i>Type:</i> float
maxWHRatio=value	Specifies the maximum width to height ratio.  <i>Type:</i> float

minFenceToFenceSpace=value	
	Specifies the minimum amount of spacing, in microns, allowed between fences.
minFenceToCoreSpace=value	
	Specifies the minimum amount of spacing, in microns, allowed between a fence and the core boundary.
minFenceToInsideMacroSpace=value	
	Specifies the minimum amount of spacing, in microns, allowed between the fence boundary and the macros that belong to the fence.
minFenceToOutsideMacroSpace=value	
	Specifies the minimum amount of spacing, in microns, allowed between the fence boundary and the macros that do not belong to the fence.
minInsideFenceMacroToMacroSpace=value	
	Specifies the minimum amount of spacing, in microns, allowed between macros that belong to the same fence.
master=seedname	
	Specifies the name of the master seed. The clone seed inherits the same constraints (except orientation) as the master seed.
cloneOrient={R0   MX   MY   R180}	
	Specifies the orientation of the clone seed.

**Note:** From the software's 11.1 release onwards, `planDesign` only supports VERSION 1 format of Floorplan constraint file.

Just make sure to add "VERSION 1.0" in the beginning of the seed file provided during macro placement (`planDesign`).

For example:

```
VERSION 1.0
BEGIN SEED
name=A1/B2 util=0.75
name=C1/D3/M5 createFence=true minFenceToFenceSpace=60
```

```

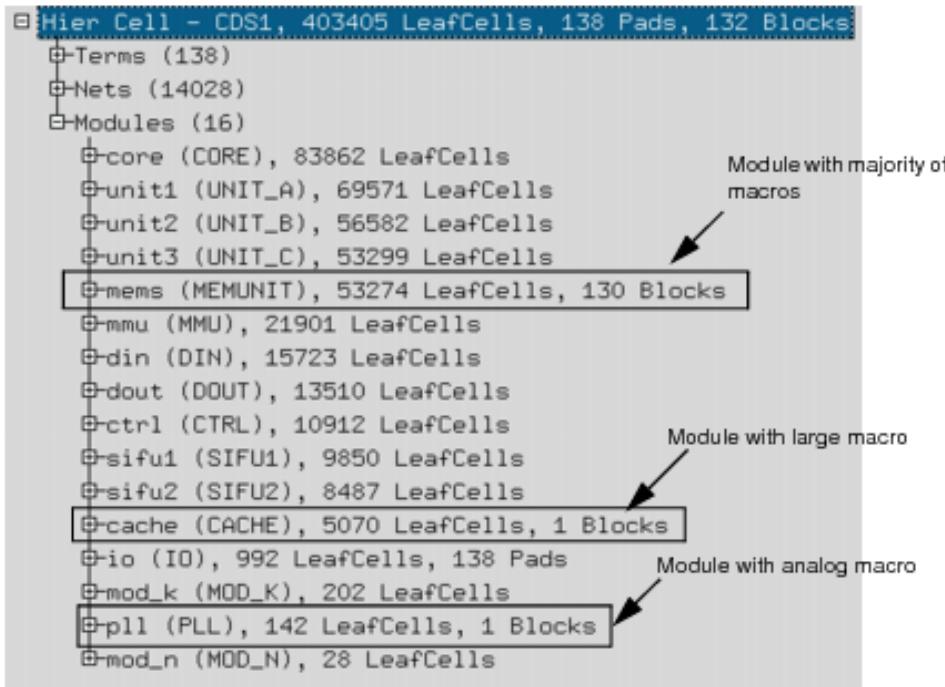
name=E1 minWHRatio=0.25 maxWHRatio=4.0
name=F2 master=E1 cloneOrient={R0 | MX}
name=PDGp1 util=0.6 createFence=true minFenceToInsideMacroSpace=10
# H1 and H2 can be existing fences.
name=H1 minFenceToInsideMacroSpace=10 minFenceToOutsideMacroSpace=10
name=H2 minInsideFenceMacroToMacroSpace=15
END SEED

```

## Creating a Seed Section In the Constraint File

The purpose of creating a seed section in the constraint file is to select seeds that will identify the data flow of your design so as to generate better floorplan results. The following steps show you one way to determine which modules in your design to select as seeds.

1. Import your design.
2. Open the Design Browser and examine the design hierarchy.



Consider the following points when examining the modules:

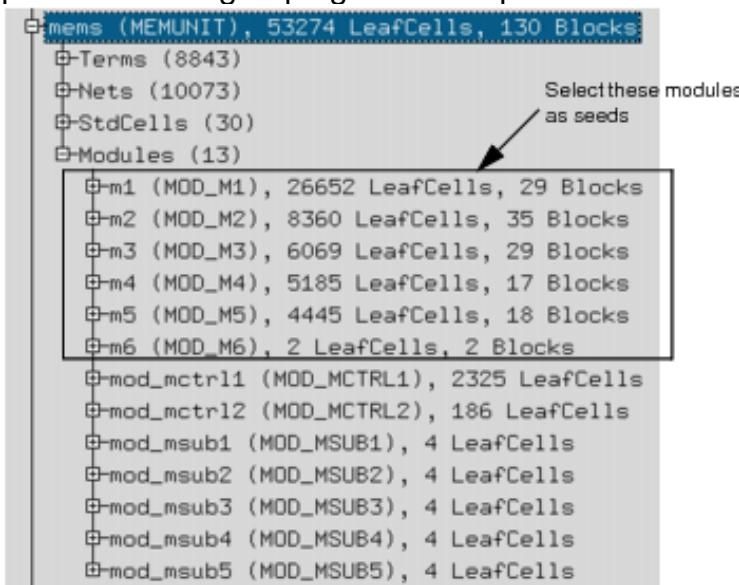
- The relationship between modules

Look at the relationships between modules to find which modules best identify the data flow in your design. The block diagram for your design can give you a good idea as to which modules these are.

- The size and area of modules.
  - Do not select seeds that differ widely in size. The size of a seed is based on an area estimate. The size difference between the largest seed and the smallest seed in the design should be no more than 10x.
  - Select large hard macros with sizes greater than 1/4 of the core area as seeds, except if they are marked as FIXED.
  - Only select small modules or single standard cells as seeds if you know they are important to the global signal flow. At the end of the seed selection process, the Automatic Floorplan Synthesis feature groups small modules and individual standard cells into instance group seeds based on existing seed size and connectivity.
  - Do not select I/O modules that include pad cells as seeds because I/O pads should already be pre-placed.
- The number of macros in a module
  - If your design hierarchy includes a single module that contains almost all of the macros in the design, do not select it as a seed. Instead, examine its sub modules for seed candidates (step 3).
- Any special modules in the design
  - For example, look for modules that include a very large macro, or an analog macro.

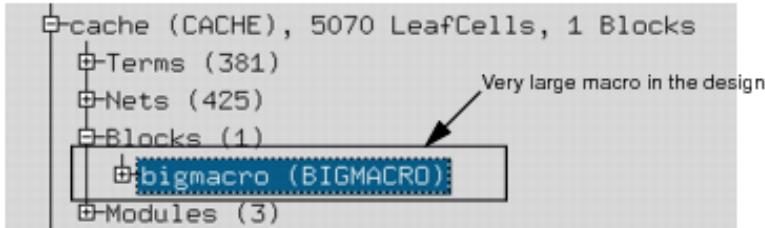
### 3. Examine the sub modules of a module for seed candidates.

In the following figure, you should select the modules at this level as seeds because it will produce better grouping for macro placement.



In the following figure, you should select this macro as a seed because it is larger than 1/4 of the core area.

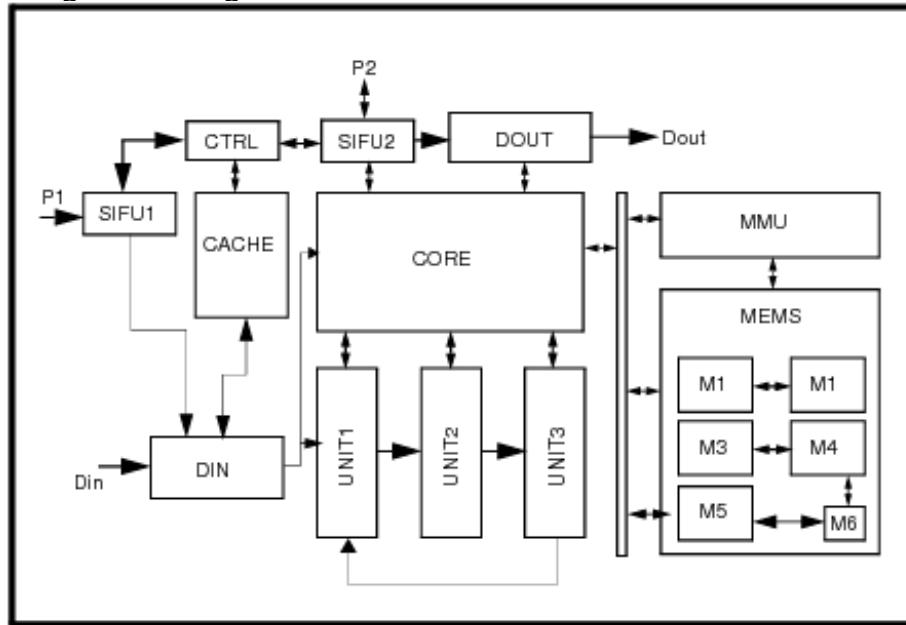
### Large Macro



4. Create a seed section.

For example, the following figure shows the block diagram for the design.

**Design Block Diagram**



Based on this diagram:

1. Examine the module `mems` because it contains the majority of the macros in the design. Select seeds from its sub modules in order to get better macro placement.
2. Examine the module `cache`, and select the macro `bigmacro` because it is very large .
3. Do not select the modules named `mod_k` and `mod_n` as seeds because they are so small that they do not show up in the block diagram.
4. Do not select the module named `io` because it includes I/O pad cells that have already been pre-placed.

The resulting seed section is as follows:

```

core          #Hinst: good for data flow
unit1        #Hinst: good for data flow
unit2        #Hinst: good for data flow
unit3        #Hinst: good for data flow
mmu          #Hinst: good for data flow
din          #Hinst: good for data flow
dout         #Hinst: good for data flow
ctrl         #Hinst: good for data flow
siful        #Hinst: good for data flow

```

```
sifu2          #Hinst: good for data flow
mems/m1        #Hinst: good for data flow and macro grouping
mems/m2        #Hinst: good for data flow and macro grouping
mems/m3        #Hinst: good for data flow and macro grouping
mems/m4        #Hinst: good for data flow and macro grouping
mems/m5        #Hinst: good for data flow and macro grouping
mems/m6        #Hinst: good for data flow and macro grouping
pll            #Hinst: maintain for analog macro
cache/bigmacro #Inst: maintain for big macro
```

## Importing the Design

1. To import a design, type the following command:

```
source design_name.globals  
init_design
```

2. To load a floorplan file, type the following command:

```
loadFPlan design_name.fp
```

## Setting Automatic Floorplan Synthesis Global Parameters

Use the `setPlanDesignMode` command to specify the global parameters of the Automatic Floorplan Synthesis feature. These parameters are used by the `planDesign` command every time you call it to create a floorplan.

You can set the following parameters:

- Hard macro spacing and fence spacing
- Target utilization for floorplan guides
- Place macros close to the chip boundary
- Place macros close to the guide boundary
- Control macro placement status
- Flow control options

## Creating an Initial Floorplan

- Type the following command to generate an initial floorplan:

```
planDesign [-constraints constraint_file]
```

In general, you use the Automatic Floorplan Synthesis feature to create multiple alternative floorplans, which you can then compare. Run the `planDesign` command the first time using automatic seed selection, and analyze the results. For each subsequent run, modify the seed section of the constraint file that was created automatically (`MP_seed`) with several different seed options. Then specify the modified constraint file when you run `planDesign` to produce different results.

**Note:** The `planDesign` command supports all spacing rules and honors them as hard constraints using the parameters defined in the Automatic Floorplan Synthesis Constraints File. The `planDesign` command understands a physical instance group by default and places it once the instance group has been defined as a seed in the Automatic Floorplan Synthesis Constraints File.

By default, Automatic Floorplan Synthesis takes timing constraints into account during floorplan generation, if timing libraries (.lib) and SDC constraint files are loaded in the design. It will not perform timing aware floorplanning if either the timing library or constraint file is not loaded.

When specified, Automatic Floorplan Synthesis performs the following internal functions in order:

- Selects the floorplan objects (seeds) to be placed

Automatic Floorplan Synthesis performs seed selection either automatically, or using the seed section of the specified constraint file, then clusters the netlist.

- Places the seeds

Automatic Floorplan Synthesis calls the placer to place the seeds of zero size in order to find the best relative locations for the seeds. Automatic Floorplan Synthesis gives I/O nets a higher weight (100), and ignores high fanout nets (0 weight).

- Refines the seeds

Automatic Floorplan Synthesis adjusts seed size, location, and aspect ratio to reflect real module and macro size, and to reduce seed-to-seed overlap area. The tool preserves the relative location of seeds throughout refinement.

- Places the macros

Automatic Floorplan Synthesis firsts groups and packs hard macros from the same seed that

are of the same type, or are similar in size and aspect ratio. Macro packs furthest from the center of the core area are moderately pushed out toward the chip or block boundary.

Automatic Floorplan Synthesis determines a macro's location and orientation based on a combination of many factors, including parent seed location, size, aspect ratio, chip or core aspect ratio, wire length and congestion optimization, macro pin layer, and the preferred routing layer direction of the layer. Automatic Floorplan Synthesis calculates the space between adjacent macros based on pin density, track estimation, and metal layer pitch. You can also specify a spacing value for the tool to use instead.

After placement, macros are marked `PLACED` in the database.

## Creating Floorplan for Hierarchical Design

Given a hierarchical top-level floorplan that contains fences or power domains that represent predefined partitions , Automatic Floorplan Synthesis can be used for:

- **Macro Placement**

Places the macros within the predefined partition fences.

**Note:** If a floorplan has pre-placed fences, you can select sub-modules under the fence module as fence candidates and put them in the constraint file. After reading the constraint file, `planDesign` automatically creates fences for sub-modules within the existing fence boundary. Macros are automatically placed within their lowest level fence boundary.

If no predefined fences or power domains are present in hierarchical top-level floorplan, Automatic Floorplan Synthesis can be used for:

- **Full-chip Floorplan**

Creates fences and places macros at chip level.

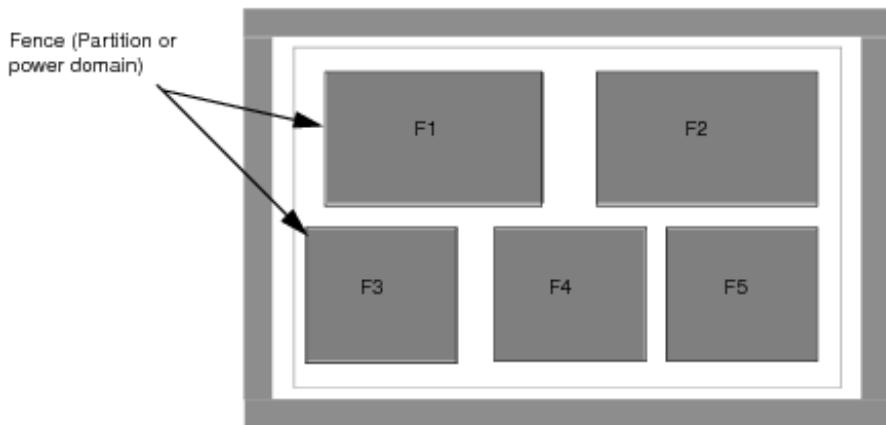
- **Power-Domain Aware Floorplan**

Reads CPF, creates power domains, and places macros.

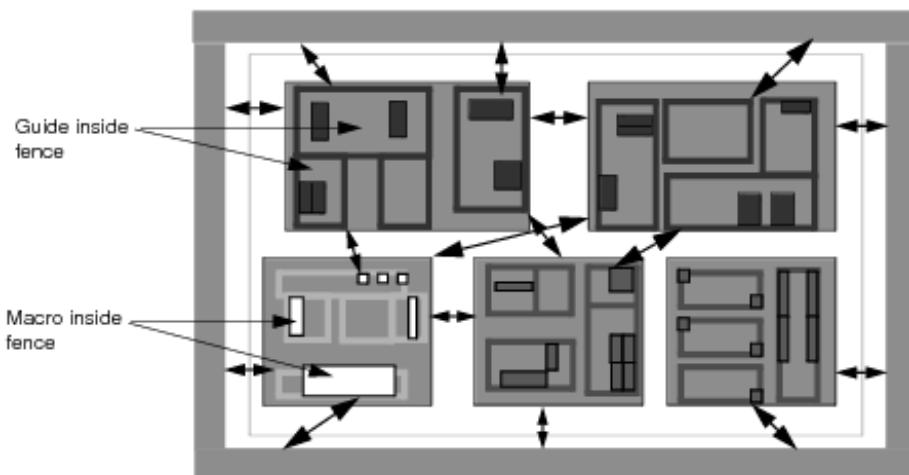
## Macro Placement

Automatic Floorplan Synthesis places hard macros, and generates guides for the sub modules within the fence boundary and also on the top level, considering the global connectivity.

### Hierarchical Top-Level Floorplan



### Automatic Floorplan Synthesis Result



**Note:** The following guidelines when performing hierarchical floorplanning:

- If you specify a seed section in a constraint file (-constraints), the fence module should not be specified in it as a seed, because Automatic Floorplan Synthesis does not touch any fences in the floorplan.

- The macro placer looks at nets globally, but does not see hierarchical ports on partition boundaries because the ports have not been assigned yet.
- In order for the macro placer to produce good results, you must set appropriate fence utilization and aspect ratio values in the input floorplan by manually stretching or reshaping them.

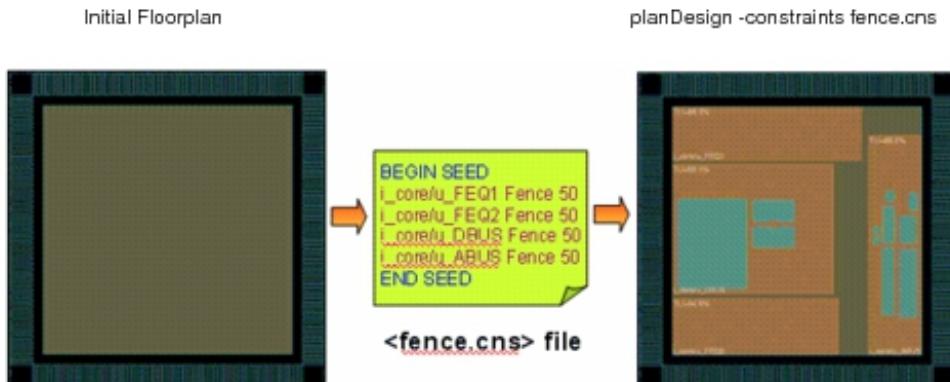
## Full-chip Floorplan

Floorplanning a top-level hierarchical design or creating power domain physical boundary for a MSV design requires rectilinear module fence generation. Automatic Floorplan Synthesis generates a quick top-level initial floorplan for such designs.

For Automatic Floorplan Synthesis to create fences and place macros at chip level, you must specify the fence module(s) in the constraint file using the `planDesign` command.

```
planDesign -constraints constraint_file
```

The results of the `planDesign` command is as follows:



## Power-Domain Aware Floorplan

The `planDesign` command is power-domain aware and supports placement of power domains. If your design uses the [Common Power Format \(CPF\) flow](#), you can run the `planDesign` command to automatically create fences around the power domains and place the power domains along with other hard blocks in the design, without having to create any seed definitions in the constraint file. The `planDesign` command honors power domain attributes, such as the minimum gap (

`modifyPowerDomainAttr -minGaps`) if they are defined before running the command.

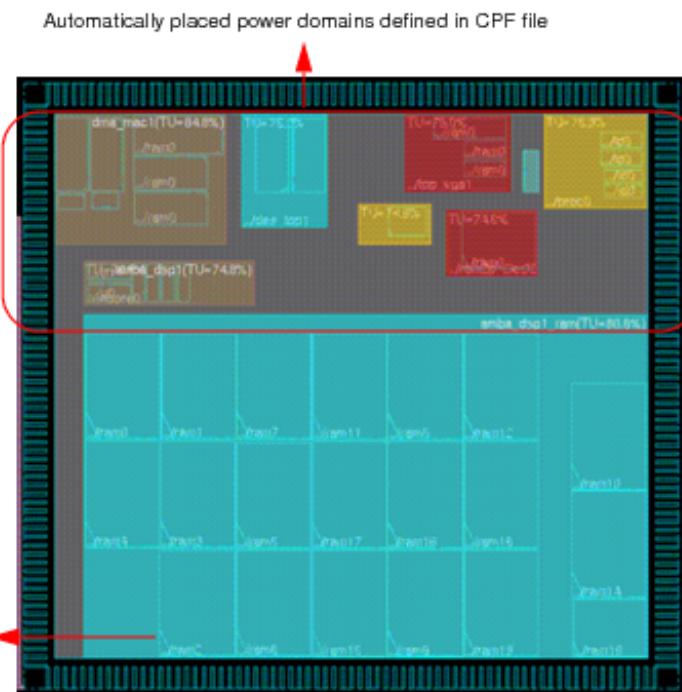
You can also perform congestion aware power-domain placement by specifying `setPlanDesignMode -congAware true`, which is also applicable for standard cell placement.

**Note:** The `planDesign` command places power domains inside the core boundary without any overlaps with other flexModels or macros (especially rectilinear macros), thus improving the QoR of power domain placement.

**Note:** When you set `setPlanDesignMode -congAware true` and call `planDesign`, you get a floorplan and an Automatic Floorplan Synthesis generated congestion map. To control the visibility of this congestion map in the floorplan view, select the *Congestion Label* checkbox in the [View-Only](#) page of the *Color Preferences* form (*Options - All Color*). You can clear the checkbox to clear the congestion map.

The following example describes the output of the `planDesign` command which has automatically placed 8 power domains and hard macros in the design:

Automatic placement of power domains and hard macros for a sample design with 8 power domains



## Creating Multiple Alternative Floorplans

Use the `multiPlanDesign` command to create multiple alternative floorplans by running different variations of the `setPlanDesignMode` and `planDesign` commands. The Automatic Floorplan Synthesis feature then analyzes the resulting floorplans, and ranks their usability using estimated wire length, congestion, and timing criteria.

The `multiPlanDesign` command generates floorplans based on the parameter settings you specify. For some parameters, you can instruct Automatic Floorplan Synthesis to always perform the functionality (on), or to perform or not perform the functionality on floorplans in a predefined order (on\_off).

You can run `planDesign` jobs sequentially on one machine, or in parallel on multiple host machines. To run `multiPlanDesign` in parallel, you must first use the `setDistributeHost` and `setMultiCpuUsage` commands to set up the configuration for the distributed processing.

For example:

```
setDistributeHost {-local | -rsh -add {machine1 machine2}}  
setMultiCpuUsage -remoteHost 2  
multiPlanDesign -autoTrials 2
```

Floorplan rankings are automatically displayed in a separate results form after all of the planDesign jobs are completed. Additionally, Automatic Floorplan Synthesis saves a text report of the ranking results to a user-specified file.

If you run this command on a master instance blackbox with non-R0 orientation, it automatically converts the new orientation to R0. For more information, see [Handling of Blackboxes with Non-R0 Orientation](#) in the "Partitioning the Design" chapter of the *Innovus User Guide*.

## Analyzing the Floorplan

After generating a floorplan, analyze the results in order to assess whether improvements are needed:

- Run the [analyzeFloorplan](#) command to analyze the floorplan, and report basic design information.
  - Visually check the floorplan for macro placement issues, such as:
    - Macros placed far away from their related modules or connected I/Os.
    - Too many wire cross-overs among modules, macros, and I/Os.
    - Too many macro-to-macro, or module-to-module, overlaps.
    - Too many dead areas in the design (that is, holes in the middle of placed macros).
    - Inadequate macro orientation or spacing.
- Note:** Use the *Preferences* form and the display control panel on the Innovus® Implementation System (Innovus) main window to control the information displayed when visually checking the floorplan.
- Read the log file to see the intermediate results of Automatic Floorplan Synthesis, and error and warning messages that might indicate to more serious issues.  
Check the following items in the log file:
    - Seed Report
      - Do the selected seeds match those specified in the seed section of the constraint file?
      - How many seeds exist in the design after clustering? Are there too many seeds, or too few?

The following example shows a typical seed report in the log file:

```
----- Clustering summary -----
*** Clustered Preplaced Objects ***
# of preplaced io is: 0
# of preplaced hard macro is: 1
# of preplaced standard cell is: 0

*** Clustered FPlan Seeds ***
# of hard macro seeds is: 1
# of standard cell seeds is: 0
# of soft module seeds is: 9
# of instance group seeds is: 1

*** Normal Netlist Clustering ***
# of clustered instances is: 0
----- End of Clustering summary -----
```

- Macro Placement

- Read the options reported back by the macro placer.
- Are there any reported overlaps between PLACED macros?
- What total wire length did the macro placer report?

## Adjusting Macro Placement

Typically after generating a floorplan, macro adjustment is required to improve the layout. There are two ways you can refine the macro placement:

- Manually adjust the macros using interactive commands
- Use `refineMacro` to adjust the macro locations

## Manual Macro Adjustment

The Innovus main window includes widgets that allow you to interactively adjust macro placement. Actions you can perform include move, align, flip, rotate, redo, and undo.

For more information on the toolbar and tool widgets, see "Toolbar Widgets" and "Tool Widgets" in the [The Main Window](#) chapter of the *Innovus Menu Reference*.

You can also use the *Edit Floorplan* submenu on the *Floorplan* menu to perform many of the same actions.

For more information, see "[Edit Floorplan](#)" in the Innovus Menu Reference.

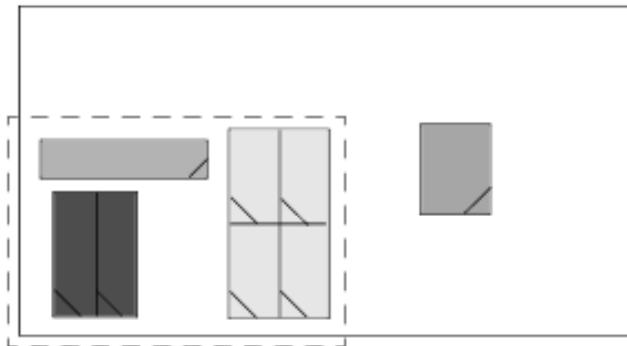
## Automatic Floorplan Synthesis Macro Adjustment

You can also use the `refineMacro` command to adjust the placement of macros in the floorplan. By default, the command performs global incremental macro adjustment. You can also adjust the placement of specific macro packs, or all of the macros in a specific area of the design.

## Adjusting the Placement of a Specific Macro Pack

1. Select the macro pack you want to adjust in the main window. A macro pack is a set of macros from the same seed that have similar sizes and aspect ratios and have been grouped together. You can select one macro to select the entire pack.

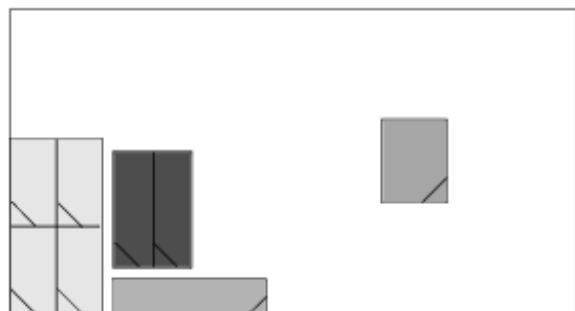
**Info:** Macro pack information is stored temporarily in the data base. If you quit the Innovus session, the information is lost.



2. Type the following command:

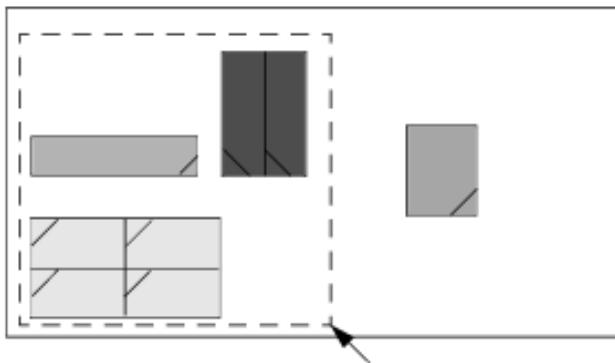
```
refineMacro -permutePack
```

Automatic Floorplan Synthesis adjusts the macros' locations to reduce empty area between them.

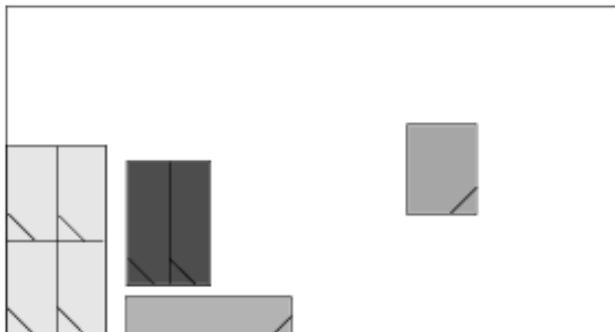


## Adjusting Macro Placement Within a Specified Area

1. Choose *Floorplan - Automatic Floorplan - Refine Macro Placement*.
2. Select the *Window Refine* option.
3. Click the *Draw* button to enable drawing mode.
4. Hold down the left mouse button and drag the cursor to draw a box around the area of the design in which you want to adjust macro placement.

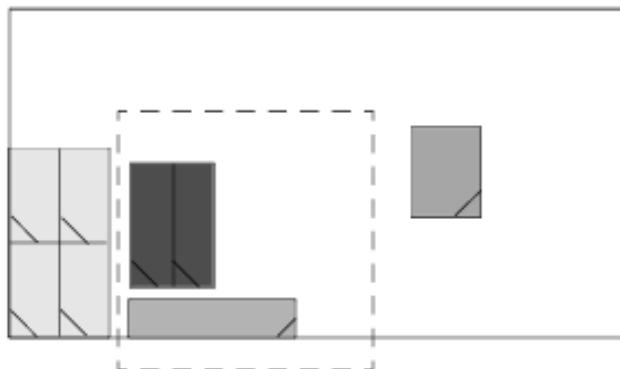


5. Click *Apply* to perform the adjustment.

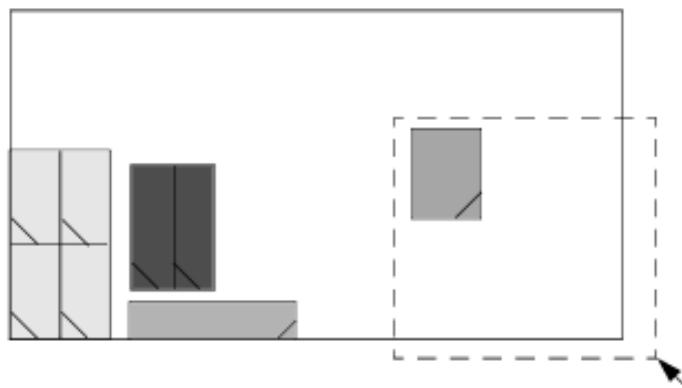


You can also move selected macros to a specified area and adjust them along with any macros already within the area.

1. Select the macros you want to move in the main window.



2. Open the Refine Macro Placement form, then select *Include Selected Macro*.
3. Click the *Draw* button and draw a box around the area of the design to which you want to move the selected macros.



Alternatively, if you know the coordinates of the area to which you want to move the macros, you can perform one of the following steps:

- Specify the coordinates in the *Path1* and *Path2 X/Y* text fields on the *Refine Macro Placement* form and click *Apply*.
  - Type the following command:  
`refineMacro -selected -area llx lly urx ury`
4. Click *Apply* to perform the adjustment.



## Marking Refinement Steps

Automatic Floorplan Synthesis considers each adjustment that you make to the floorplan a step. By default, Automatic Floorplan Synthesis saves all incremental steps done with the `refineMacro` command and lists them in the log file with a number. The tool can store up to 1,000 steps in the database before overwriting them. Automatic Floorplan Synthesis does not save steps done through manual refinement (such as move, rotate, and flip).

To mark manual refinement steps (or any other important step that you want to save), type the following command:

```
refineMacro -markStep
```

Automatic Floorplan Synthesis assigns a number to the mark (as well as a step number) and stores the step in the database. It saves up to 20 marks before overwriting them.

## Restoring Refinement Steps

- To return the design to the state it was in after the specified intermediate non-marked step, type the following command:  
`refineMacro -restoreStep step_number`
- To return the design to the state it was in after the specified marked step, type the following command:  
`refineMacro -restoreMark mark_number`
- To return the design to the state it was in after the previous adjustment, type the following command:

```
refineMacro -restoreStep -1
```

You can find the mark and step numbers listed in the log file.

## Saving the Floorplan

To save the floorplan, type the following command:

```
saveFPlan fileName.fp
```

# Using Trial Route for Congestion and Timing Analysis

- [Overview](#)
- [Data Preparation](#)
- [Routing a Flat Design](#)
- [Routing a Partitioned Design](#)
- [Routing Two-Metal Layer Designs](#)
- [Routing Using the NanoRoute Global Router](#)
- [Loading and Saving Route Data](#)
- [Analyzing Route Data](#)
- [Improving Route Congestion](#)
- [Using Bus Guides](#)
- [Trial Route Behavior For Hierarchical Flow](#)
- [Additional Information](#)

## Overview

Trial Route performs quick global and detailed routing for estimating routing-related congestion and capacitance values. It also incorporates any changes made during placement, such as scan reorder. You can use Trial Route results to estimate and view routing congestion, and to estimate parasitic values for optimization and timing analysis. When used during prototyping, Trial Route creates actual wires, so you can get a good representation of RC and coupling for timing optimization at an early stage in the flow. Trial Route also produces a congestion map you can view to get early feedback on whether the design is routable. Trial Route results can also be used for pin assignment when you commit partitions.

**Note:** Trial Route does not guarantee DRC-clean routing results. Do not perform signal integrity analysis on a design that has been routed using Trial Route, because the routes are only used to estimate parasitic values for timing analysis. Route designs with NanoRoute or WRoute, if you want to perform signal integrity analysis.

You can use Trial Route during virtual prototyping, hierarchical floorplanning, block implementation, and top-level implementation.

## Data Preparation

The design must be successfully placed and it must be loaded into the current Innovus session.

**Note:** If you make any changes after running Trial Route that affect the placement data--for example, floorplan changes--you must run placement before rerunning Trial Route.

The following optional input files are only required as necessary:

- DEF file
- Top Design Format (TDF) file containing routing data
- Routing guide file

## Routing a Flat Design

### Initial Design Routing

Run Trial Route for the first time to gauge the routability of the design. You can then examine the congestion map and congestion distribution report to identify congested areas that might cause routing problems later in the design session.

1. Choose *Route - Trial Route*. This opens the *Trial Route* form.
2. Select the *Prototyping* effort level and click *OK*. You can also issue the following command:

```
trialRoute -floorplan
```

**Note:** The prototyping ( *-floorplan* ) mode runs Trial Route quickly, which is important when prototyping large designs. However, note that components in your design might not be routed at legal locations.

### Post Clock Tree Synthesis Routing

Run Trial Route after clock-tree synthesis to recheck the routing congestion and to estimate parasitic values for timing analysis.

1. Choose *Route - Trial Route*.
2. Select the *Medium Effort* (default) or *High Effort* effort level on the Trial Route form.
3. (Optional) Select the *Use Routing Guide* option, and specify the name of a guide file in the

corresponding field. Trial Route follows the routing regions defined in the guide file and honors the specified pre-routed nets.

4. Click *OK*.

Alternatively, you can issue the following command:

```
trialRoute -highEffort -guide my_chip.rguide
```

## Routing a Partitioned Design

In a flat design, Trial Route can route through guides, regions, and fences, as long as there are no routing blockages or hard blocks. However, fences are often defined as partitions, which become blocks after the design becomes hierarchical. Once partitions become blocks, the routes are no longer allowed, unless they use a proper feedthrough mechanism, such as inserted buffers or routing feedthroughs.

In channel-based routing designs, all top-level routing use channels to route around partitions. In a partitioned design in which the partitions have not been committed, you can use the Trial Route `-handlePartition` and `-handlePartitionComplex` parameters to force the routing into channels, simulating a channel-based design.

Issue one of the following commands:

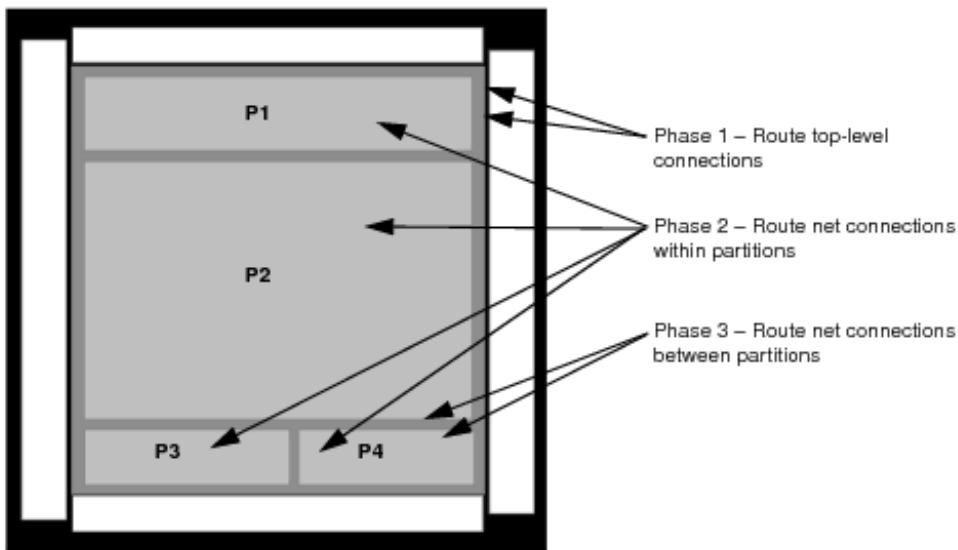
```
trialRoute -highEffort -handlePartition or  
trialRoute -highEffort -handlePartitionComplex
```

Use the `-handlePartition` parameter to route nets that are only connected within partitions. Use the `-handlePartitionComplex` to route nets that belong to more than one partition, so that the routing does not violate partition boundaries.

When the `-handlePartition` or `-handlePartitionComplex` parameter is specified, Trial Route works in three phases:

- Phase 1 - Routes the top-level connections
- Phase 2 - Routes net connections within partitions
- Phase 3 - Routes net connections between partitions

For example, the design in the figure below has five metal layers, a top-level partition and four flattened partitions: P1, P2, P3, and P4. P1 reserves the *metal1*, *metal2*, and *metal3* layers for partitions, and P2, P3, and P4 reserve layers *metal1* through *metal5* for partitions.



If you specify `-handlePartition` or `-handlePartitionComplex`, Trial Route performs the following tasks to route the net connections:

1. Trial Route applies all of the metal blockages defined for each partition. The layers metal1, metal2, and metal3 are blocked for P1, and metal1 through metal5 are blocked for the other partitions.
2. Trial Route then routes nets connecting only at the top level. No connections to partitions or cells within the partitions are made.
3. Trial Route blocks all areas outside of the defined partitions on all routing layers. For P1, Trial Route applies a routing blockage for layers metal4 and metal5.
4. Trial Route routes all nets within P1, P2, P3, and P4 on the available routing layers. This means that even a cell at the lower-left corner of P2 that connects to a cell at the upper-right corner of P2 is routed within P2, regardless of any congestion.
5. Trial Route removes the routing blockages and routes the net connections between partitions. If an I/O at the top level connects to P2:
  - a. If you specify `-handlePartition`, Trial Route uses all metal layers to route through P1.
  - b. If you specify `-handlePartitionComplex`, Trial Route uses only layers metal4 and metal5 to route through P1.

## Routing Two-Metal Layer Designs

Trial Route can route designs with only two metal layers defined in the LEF file. Trial Route automatically detects when a design is M1/M2 only, and uses wires from the M1 and M2 layers for routing. Trial Route can also perform two-layer routing on designs that have more than two metal layers defined in the LEF file by setting the `trialRoute` or `setTrialRouteMode -maxRouteLayer` parameter to 2. Trial Route then uses wires from the M1 and M2 layers for routing.

All pins of the nets to be routed must be on layers M1 and M2.

## Routing Using the NanoRoute Global Router

**Note:** This is a limited-access feature. This feature has been internally qualified at Cadence but has had only limited customer testing. The limited access features are enabled by a variable specified through the `setLimitedAccessFeature` command. To use this limited access feature, please contact your Cadence representative to qualify your usage and make sure it meets your needs before deploying it widely.

Trial Route can route designs using the NanoRoute global router technology in place of the current trialRoute technology by setting the `setTrialRouteMode -useNanoRoute` parameter to true. When the `-useNanoRoute` parameter is enabled, the routing correlation between the preRoute and postRoute stage is improved, which results in reduced timing jumps between preRoute and postRoute timing. The resulting congestion analysis report uses the same formatting and provides the same level of information as the one from global detailed routing phase. You can check the obstructions and congestion in the design graphically by analyzing the generated congestion map.

**Note:** The `setTrialRouteMode -useNanoRoute` does not support designs with partitions and blobs. When the router detects partitions in the design, it automatically reverts to the trialRoute technology to route the design.

For more information on congestion maps, see Using the Congestion Map in the "[Using the NanoRoute Router](#)" chapter of the *Innovus User Guide*.

## Loading and Saving Route Data

After initially running the Trial Route program, you can load or save Trial Route data at any time during an Innovus session.

- To load the route data, use the `restoreRoute` command.

- To save the route data select the Save Routing to option on the Trial Route form and specifying a filename.

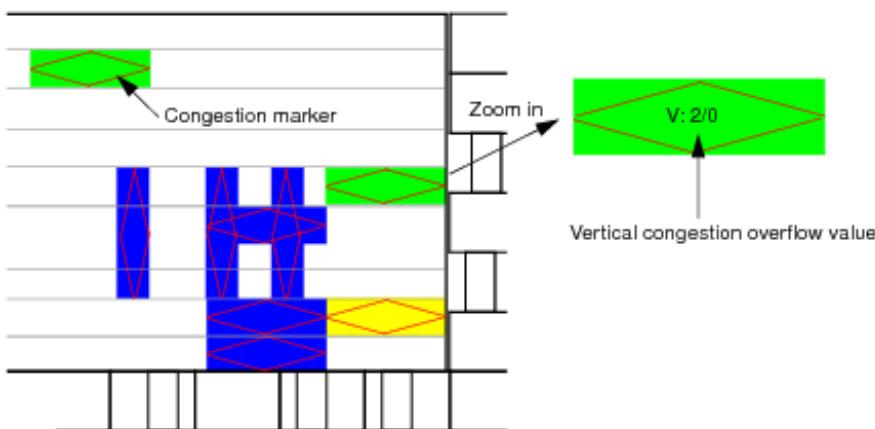
## Analyzing Route Data

After running Trial Route, you can analyze the results to check if your design is routable for back-end detailed routing tools.

1. Visually check the route congestion markers. The red diamond-shaped congestion markers should not be very dense in a local area. These markers contain an overflow value to identify the number of tracks required for that grid, and the actual number of tracks available. See Congestion Markers in the Display for more information.
2. In the log file, inspect the Trial Route contents in the congestion distribution table. See Congestion Distribution Report for more information. When these two tests are satisfactory, the design is routable by a detail router.

## Congestion Markers in the Display

You can visually check the Trial Route congestion statistics in the design display area of the main Innovus window to identify the tight clusters of congestion markers. Check the design display area to make sure there are no markers grouped closely together. These usually occur around blocks or between large blocks. The indicators are diamond shaped and red by default. Zoom into the area to display the vertical and horizontal congestion overflow values, as shown in the following figure.



Congestion markers contain a vertical or horizontal overflow value to identify the number of tracks required for that grid, and the actual number of tracks available. For example, in the above illustration, the vertical overflow is 2/0, which indicates that two additional tracks are required, and 0 tracks are available. Congestion marker values are based on an integer number of adjacent gcells that are grouped together to form a "super gcell." Horizontal congestion super gcells are tall, narrow boxes that typically have a height of four gcells and a width approximately equal to the height of a vertical congestion super gcell. Vertical congestion super gcells are short, wide boxes that typically have a height of one gcell and a width approximately equal to the height of a horizontal congestion super gcell.

Vertical and horizontal overflow values are calculated separately for better accuracy. The overflow value is the amount by which the track demand exceeds the track supply. The required track value is calculated by totalling the number of required tracks in the super gcell. That is, the value is the sum of the number of required tracks in all of the adjacent gcells that form the super gcell. The available track value is calculated by totalling the number of available tracks in the super gcell.

**Note:** Congestion markers can display different congestion information than that contained in the default congestion distribution report. The information in the congestion distribution report is based on the congestion of each gcell instead of the super gcells. To create a congestion report based on the congestion of the super gcells, use the [describeCongestion](#) command.

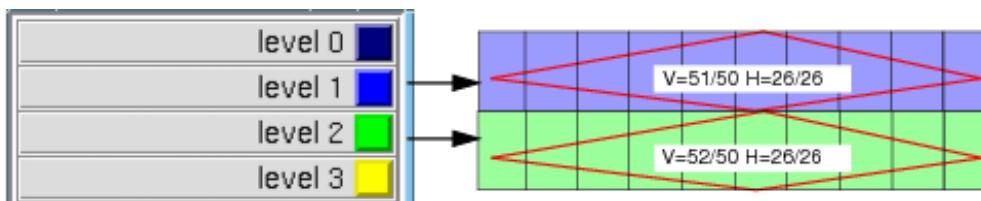
To change the size of super gcells, define the following variable:

```
set rdaSuperGcellSize n
```

The value you specify for *n* must be greater than or equal to 0 and less than or equal to 10. If you specify a value of 1, a super gcell becomes a regular gcell, and the displayed congestion marker information matches the congestion information provided in the report. If you specify a value of 0, the super gcells become square.

## Congestion Marker Color Boxes

By specifying the *HCongest* and or *VCongest* colors in the *Color* panel, you can also add a color box to the congestion marker that indicates the severity of the overflow level (that is, the number of overflow tracks in a one-unit area). Usually, a one-unit area contains 10 global cells (gcells) horizontally. If there are 50 vertical tracks available in that area, and Trial Route requires 51 vertical tracks, the congestion marker color box is blue (by default), indicating a one-track overflow. If Trial Route requires 52 vertical tracks, the congestion marker color box is green (by default), indicating a two-track overflow. An example of this is shown in the following figure.



The following table shows the default congestion marker colors and their corresponding overflow values:

Level	Color	Overflow Value
level 1	Blue	1 (One more track required)
level 2	Green	2 (Two more tracks required)
level 3	Yellow	3 (Three more tracks required)
level 4	Red	4 (Four more tracks required)
level 5	Magenta	5 (Five more tracks required)
level 6 and higher	Gray to white	6 or greater (Six or more tracks required)

For more information, see Multicolor Layers in the [The Main Window](#) chapter of the *Innovus Menu Reference*.

## Congestion Distribution Report

After Trial Route completes, a congestion distribution report is created in the innovus.logv file. The congestion distribution report provides usage and routing overflow percent values, as well as gcell overflow information (that is, the internal supply and demand for each gcell). There are two types of congestion distribution reports that can be generated: a default congestion distribution report; and a detailed congestion distribution report.

**Note:** The congestion information contained in the congestion distribution report can differ from the congestion information displayed in congestion markers in the Innovus window. For more information, see Congestion Markers in the Display.

## Default Congestion Distribution Report

By default, Trial Route generates a congestion distribution report that summarizes congestion information for the entire chip.

## Usage and Routing Overflow

The following example illustrates the section of the congestion distribution report that summarizes the usage and routing overflow percent values:

```
Phase 1f route (0:00:02.0 105.9M) :
Usage: (24.2%H 35.8%V) = (8.695e+06um 1.314e+07um) = (1734514 486668)
Overflow: 192 = 1 (0.0% H) + 191 (0.07% V)
```

The `Usage` statement summarizes horizontal and vertical tracks used in gcells. In the above example, there are 1,734,514 horizontal tracks used in all gcells and 486,668 vertical tracks used in gcells. Of the available horizontal tracks, 24.2 percent were used for horizontal routing (that is, wires), and 35.8 percent were used for vertical routing. The total horizontal wire length used equals 8.65e+06 μm, and the total vertical wire length used equals 1.314e+07 μm.

The `Overflow` statement summarizes all overflowed gcells. In the example, there is one horizontally overflowed gcell, and 191 vertically overflowed gcells.

## Gcell Overflow

The following example illustrates the section of the congestion distribution report that summarizes gcell overflow information. A gcell has overflow if its demand exceeds its supply. Supply is the available routing resource, and demand is the amount of routing resource assigned to the gcell. Typically, the supply is the number of unobstructed tracks crossing the gcell, and the demand is the number of wires assigned to it.

Remain	cntH	cntV	
<hr/>			
-6:	0	0.00%	1
-5:	2	0.00%	0
-3:	10	0.00%	26
-2:	510	0.03%	830
-1:	8100	0.47%	17618
<hr/>			
0:	78504	4.59%	178501
1:	102934	6.02%	214588

2:	76165	4.46%	185168	11.03%
3:	72832	4.26%	179080	10.67%
4:	81555	4.77%	180443	10.75%
5:	92704	5.42%	158498	9.44%
6:	106167	6.21%	137707	8.20%

The following table defines the columns in the congestion report:

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

The following line from the example shows that there are 8,100 gcells (.47 percent of the total number of gcells) where the demand exceeds the supply by one track in the horizontal direction, and 17,618 gcells (1.05 percent of the total number of gcells) where the demand exceeds the supply by one track in the vertical direction:

-1: 8100 0.47% 17618 1.05%

The following line shows that there are 78,504 gcells where the track supply is equal to the track demand in the horizontal direction, and 178,501 gcells where the track supply is equal to the track demand in the vertical direction:

0: 78504 4.59% 178501 10.63%

## Detailed Congestion Distribution Report

You can create a detailed congestion distribution report that writes out information about sections of the chip. These sections can be either fixed quadrants defined by the middle horizontal and vertical lines, or user-defined sections specified by rows and columns. To create a report for quadrants, specify `trialRoute -printSections` or `setTrialRouteMode -printSections true`. The report formats the information section-by-section, with each section containing congestion information for each layer in the section.

To create a report for user-defined sections, define the following two variables before running `trialRoute -printSections` or `setTrialRouteMode -printSections true`:

```
set trgNrSecRows number  
set trgNrSecCols number
```

These variables define the number of equal-size sections into which to divide the chip when reporting the congestion information. For example, the following two variable definitions divide the chip area into 3 x 2 sections of equal size:

```
set trgNrSecRows 2  
set trgNrSecCols 3
```

The detailed congestion distribution report is divided into six categories of information for each section of the chip:

- Virtual (global) wire length
- Range of tracks in a gcell
- Number of gcells with remaining tracks, including blocked gcells
- Number of gcells with remaining tracks, excluding blocked gcells
- Track usage in gcells
- Real wire length

These categories are described below.

### Virtual (global) wire length

This section summarizes the number of tracks used, the estimated wire length, the percentage of overflow, and the percentage of blocked gcells for each layer. The following example illustrates this section of the congestion distribution report:

```
***Virtual wire length:  
M2: tracks used = 17.8% = 7754505/43513490 est wire length = 1.707e+07um  
      overflow = 0.0% (0.0%) = 369/13471232 blk = 65.5%  
M3: tracks used = 19.1% = 4452746/23334471 est wire length = 1.959e+07um
```

```

overflow = 0.0% (0.0%) = 515/13471232 blk = 65.1
M4: tracks used = 13.1 % =14991505/114837671 est wire length = 3.298e+07um
    overflow = 0.0% (0.0%) = 237/13471232 blk = 0.4%

```

## Range of tracks in a gcell

This section summarizes the range of the number of tracks used in a gcell on each layer, and the average number of tracks used per gcell on the layer. The following example illustrates this section of the congestion distribution report:

```

Range of number of tracks in a Gcell in layer M2: [5:10], avg: 3.2
Range of number of tracks in a Gcell in layer M3: [1:29], avg: 1.7
Range of number of tracks in a Gcell in layer M4: [1:29], avg: 8.5
Range of number of tracks in a Gcell in layer M5: [3:6], avg: 5.0

```

The first line of this example shows that there are between 5 and 10 tracks used in a gcell on layer *metal2*, and that the average number of tracks used in a gcell is 3.2.

## Number of gcells with remaining tracks, including blocked gcells

This section summarizes the number of gcells, including blocked gcells, with remaining tracks (or the internal supply and demand for gcells) for each layer. A gcell has overflow if its demand exceeds its supply. Supply is the available routing resource and demand is the amount of routing resource assigned to the gcell. Typically, the supply is the number of unobstructed tracks crossing the gcell, and the demand is the number of wires assigned to it.

The following example illustrates this section of the congestion distribution report:

Table for number of Gcells with remain tracks (includes blocked Gcells) :

Remain	M2	M3	M4	M5	...
<hr/>					
-6:	0	0.00%	0	0.00%	0
-5:	0	0.00%	0	0.00%	3
-4:	0	0.00%	0	0.00%	5
-3:	0	0.00%	0	0.00%	37
-2:	10	0.00%	28	0.00%	264
-1:	355	0.00%	476	0.00%	1254
<hr/>					
0:	8855290	65.73%	8858305	65.76%	70715
1:	91903	0.68%	269941	2.00%	124321
2:	166878	1.24%	441388	3.28%	192300
3:	250040	1.86%	559047	4.15%	347255
					0.94%
					3.42%
					7.81%
					8.23%

4:	319497	2.37%	734307	5.45%	738688	5.48%	2369132	17.59%
----	--------	-------	--------	-------	--------	-------	---------	--------

The `Remain` column is the track supply minus the track demand. When this value is a positive number, it is the number and percentage of gcells where the track supply exceeds the track demand on each layer. When it is a negative number, it is the number and percentage of gcells where the track demand exceeds the track supply on each layer. When it is 0 (zero), it is the number and percentage of gcells where the track demand is equal to the track supply on each layer.

The following line from the example shows that there are 8,855,290 gcells (65.73 percent of the total number of gcells) where the track supply is equal to the track demand on layer metal2, and 8,858,305 gcells (65.76 percent of the total number of gcells) where the track supply is equal to the track demand on layer metal3.

Remain	M2	M3	M4	...		
-	-	-	-	-		
0:	8855290	65.73%	8858305	65.76%	70715	0.52%

The following line from the example shows that there are 91,903 gcells (.68 percent of the total number of gcells) where the track supply exceeds the track demand on layer metal2, and 269,941 gcells where the track supply exceeds the track demand on layer metal3.

Remain	M2	M3	M4	...		
-	-	-	-	-		
1:	91903	0.68%	269941	2.00%	124321	0.92%

## Number of gcells with remaining tracks, excluding blocked gcells

This section summarizes the number of gcells, excluding blocked gcells, with remaining tracks for each layer, and follows the same format as the table that reports the number of gcells with remaining tracks, including blocked gcells. The following example illustrates this section of the congestion distribution report:

Table for number of Gcells with remain tracks (excludes blocked Gcells) :

Remain	M2	M3	M4	M5	...	
-	-	-	-	-	-	
-6:	0	0.00%	0	0.00%	0	0.00%
-5:	0	0.00%	0	0.00%	0	0.00%
-4:	0	0.00%	0	0.00%	0	0.00%
-3:	0	0.00%	0	0.00%	0	0.00%
-2:	10	0.00%	28	0.00%	6	0.00%
-1:	355	0.00%	476	0.00%	229	0.00%
0:	34536	0.74%	83715	1.78%	18240	0.14%
1:	91903	1.98%	269941	5.75%	124321	0.93%

```
2: 166878 3.59% 441388 9.40% 192300 1.43% 1052688 7.83%
3: 250040 5.38% 559047 11.90% 347255 2.59% 1109275 8.25%
```

## Track usage in gcells

This section summarizes the percentage of tracks used for routing per gcell for each layer. It also reports the average number of tracks used in a gcell on each layer. The following example illustrates this section of the congestion distribution report:

Table for track usage in Gcells

Usage (%)	All M	M2	M3	M4	M5	...
#Gcells avg trks:		3.2	1.7	8.5	5.0	
<hr/>						
0%-50%	97.31%	30.20%	29.25%	93.68%	88.09%	
	13108628	4068834	3940947	1260035	11866924	
50%-60%	1.64%	0.33%	3.00%	1.81%	7.37%	
	221249	44376	404372	243607	9933345	
60%-70%	0.78%	1.81%	0.08%	2.08%	0.16%	
	104771	243711	10795	279931	22125	
70%-80%	0.24%	1.24%	1.85%	1.20%	3.34%	
	32323	166753	249401	161193	450375	
80%-90%	0.03%	0.68%	0.05%	0.71%	0.07%	
	3796	91903	6908	95516	9724	

The fourth line from this example shows that 70 percent to 80 percent of the tracks in 1.24 percent of the gcells (out of a total of 166753 gcells) on layer *meta/2* are used.

## Real wire length

This section summarizes the total real wire length used and number of vias for each layer in the section. The following example illustrates this section of the congestion distribution report:

```
***Real Wire Length:
Total: 1.396e+08um, total number of vias: 3547270
M1 (V): 9.076e+04um
M2 (H): 1.490e+07um, number of M1/M2 vias: 1560437
M3 (V): 1.652e+07um, number of M2/M3 vias: 1273925
M4 (H): 3.241e+07um, number of M3/M4 vias: 360881
M5 (V): 4.233e+07um, number of M4/M5 vias: 242047
M6 (H): 2.492e+07um, number of M5/M6 vias: 75288
```

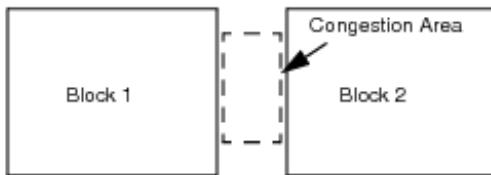
M7 (V) : 5.399e+06um, number of M6/M7 vias: 26668

M8 (H) : 3.013e+06um, number of M7/M8 vias: 8034

## Improving Route Congestion

For a module or submodule that has route congestion, complete one of the following actions to improve congestion, depending on the type and severity of the violation:

1. Change the block pin orientation. Route congestion usually occurs around blocks that have their pins facing incorrectly. You can identify these blocks by clicking on them in the design display area to see where the flight lines terminate. When the block pins are visible, you can rotate or flip the block(s) to alleviate the congestion. For information on changing block orientation, see [Flip/Rotate Instances](#) in the "[Floorplan Menu](#)" chapter of the *Innovus Menu Reference*.
2. Add density screens. You can use density screens to control standard cell placement density in certain areas where there is high routing congestion. Use the Add Density Screens tool to create a screen over the highly congested area. Congestion is more severe if it spans between two blocked areas, as illustrated in the following figure.



This figure represents a vertically congested area between two blocks that are placed close to one another. This routing bottleneck is more severe than local congestion. Assigning a density screen alleviates this congested area.

For information on adding density screens, see *Add Partial Placement Blockage* under in the *Floorplan Widgets* section of *Tool Widgets*.

## Using Bus Guides

Trial Route uses bus guide information created with the bus planning feature to better control the routing of nets and nets groups. Bus guides can be created to control routing by area, layer, and net names. Trial Route automatically checks the bus guide information, then routes nets and net groups based on their defined bus guide constraints. Trial Route with bus guide routing can be run before or after assigning pins for a hierarchical partition or a black box design. When you run Trial Route on a design containing a bus guide, all the routing nets belonging to the specified net group are highlighted.

Bus guide routing enables groups of nets to be routed, through larger areas. This requires the gcells allowed for the nets to cover a larger area, based on the bus guide width. Nets can go outside of a specified bus guide, but must be at least partially within the gcells covered by the bus guide. If the bus guide area is blocked, or too narrow, nets can go completely outside of the bus guide area. In this case, the Innovus software issues a warning message. To display warning messages, set `setTrialRouteMode -printWiresOutsideBusguide` parameter to true (or specify `trialRoute -printWiresOutsideBusguide`).

For more information on creating bus guides, see the "[Bus Planning](#)" chapter in the *Innovus User Guide*.

The following limitations exist for the Trial Route bus guide feature:

- Bus bit ordering is not guaranteed; it depends on the pin ordering and topology requirements.
- Bus guides must be complete, and must cover the pins of the net they are to guide.

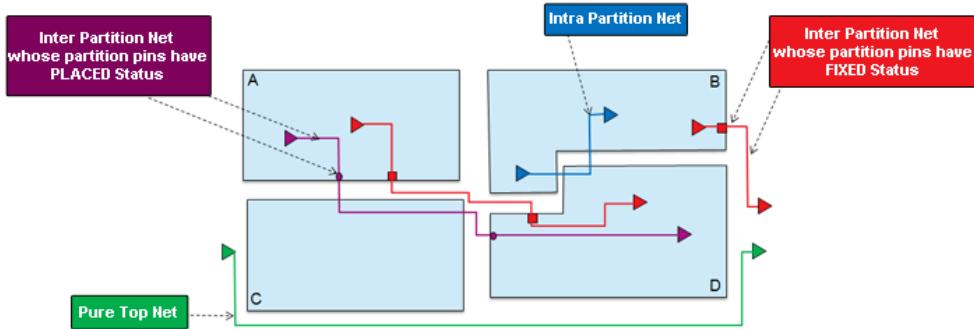
## Trial Route Behavior For Hierarchical Flow

In hierarchical flow, trial route is used to estimate the congestion and hot spots in the design. It is especially helpful to estimate congestion in thin channels between partition boundaries, and between partition and hard macro boundaries. It is also used as a useful guidance for hierarchical flows steps of pin assignment and timing budgeting.

It is vital to be able to use trial route - with its various command options - correctly and as per the needs of the hierarchical flow step.

- Standard Trial Route Behavior
- Trial Route Behavior with HonorPin

**Note:** The illustrations used in this section to describe the behavior of `trialRoute` in the hierarchical flow use the following color coding:



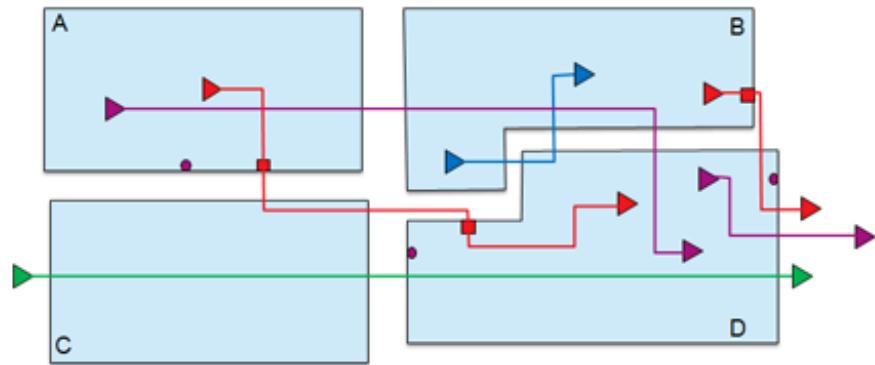
Color	Net Type
RED	Inter Partition Nets with Fixed Pins
MAGENTA	Inter Partition Nets with Placed Pins
BLUE	Intra Partition Net
GREEN	Pure Top Nets

## Standard Trial Route Behavior

- Trial Route: Default Behavior
- Trial Route: HandlePartition Behavior
- Trial Route: HandlePartitionComplex Behavior

## Trial Route: Default Behavior

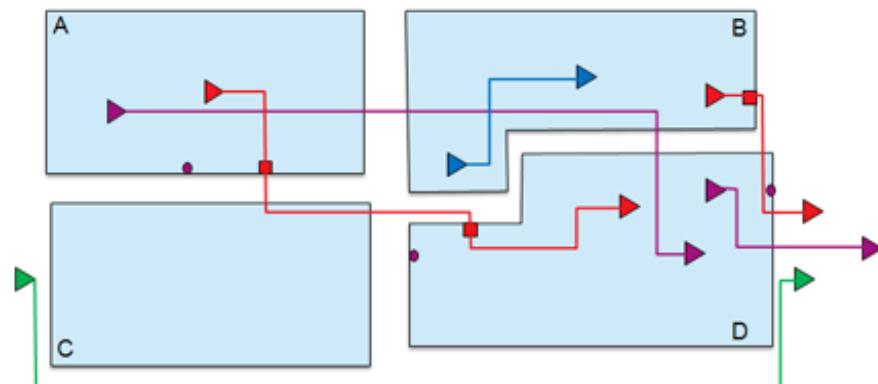
The following tables describes the default behavior of `trialRoute` in hierarchical flow



<b>Intra Partition Net:</b>	Routed freely (as if there were no partitions).
<b>Pure Top Net:</b>	
<b>Inter Partition Nets with Placed Pins</b>	Routed freely.
<b>Inter Partition Nets with Fixed Pins</b>	<p>Routed through the partition pin, such that:</p> <ul style="list-style-type: none"> <li>• The subnets inside partitions are routed within respective partition fences.</li> <li>• The subnet in top, is routed freely, but without entering into source or sink partitions.</li> </ul> <p>It is allowed to enter other partitions (like C).</p>

## Trial Route: HandlePartition Behavior

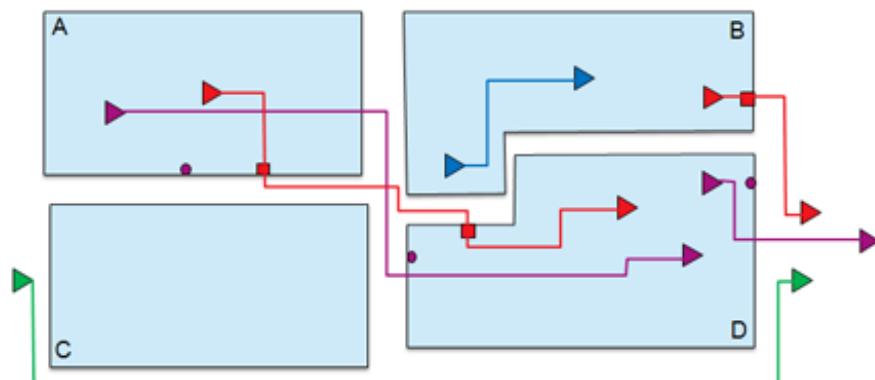
The following table describes the behavior of `trialRoute -handlePartition true` in hierarchical flow.



<b>Intra Partition Net</b>	Routed completely inside its partition.
<b>Pure Top Net</b>	Routed completely avoiding all partitions.
<b>Inter Partition Nets with Placed Pins</b>	Routed freely.
<b>Inter Partition Nets with Fixed Pins</b>	<p>Routed through the partition pin, such that:</p> <ul style="list-style-type: none"> <li>• The subnet inside partitions are routed within respective partition fences</li> <li>• The subnet in top, is routed freely, but without entering into source or sink partitions.</li> </ul> <p>It is allowed to enter other partitions.</p>

## Trial Route: HandlePartitionComplex Behavior

The following table describes the behavior of `trialRoute -handlePartitionComplex true` in hierarchical flow.



<b>Intra Partition Net</b>	Routed completely inside its partition.
<b>Pure Top Net</b>	Routed completely avoiding all partitions.
<b>Inter Partition Nets with Placed Pins</b>	Routed through the partition pin.
<b>Inter Partition Nets with Fixed Pins</b>	Routed through the partition pin, such that: <ul style="list-style-type: none"> <li>• The subnet inside partitions are routed within respective partition fences.</li> <li>• The subnet in top, is only in top region, avoiding all partitions.</li> </ul>

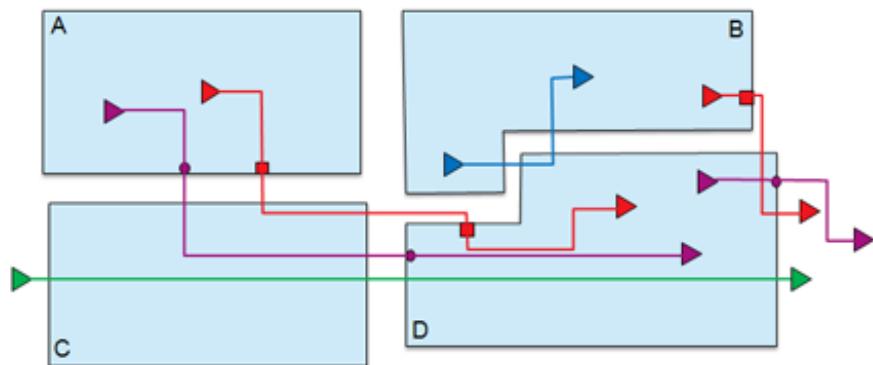
## Trial Route Behavior with Honor Pin

With the `-honorPin` option enabled, `trialRoute` honors pre-assigned pins (pins marked FIXED) and assigned pins (pins marked PLACED).

- Trial Route: Default Behavior with Honor Pin
- Trial Route: HandlePartition Behavior with Honor Pin
- Trial Route: HandlePartitionComplex Behavior with Honor Pin

## Trial Route: Default Behavior with Honor Pin

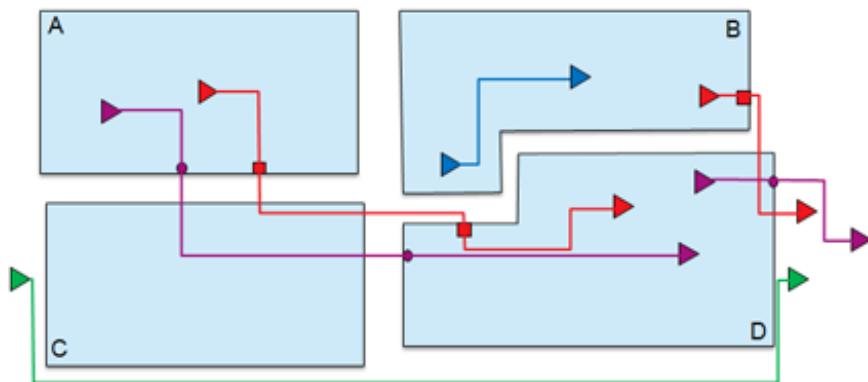
The following tables describes the default behavior of `trialRoute -honorPin true` in hierarchical flow



<b>Intra Partition Net</b>	Routed freely (as if there were no partitions).
<b>Pure Top Net</b>	
<b>Inter Partition Nets with Placed Pins</b>	Routed through the partition pin, such that: <ul style="list-style-type: none"> <li>The subnets inside partitions are routed within respective</li> </ul>
<b>Inter Partition Nets with Fixed Pins</b>	

## Trial Route: HandlePartition Behavior with Honor Pin

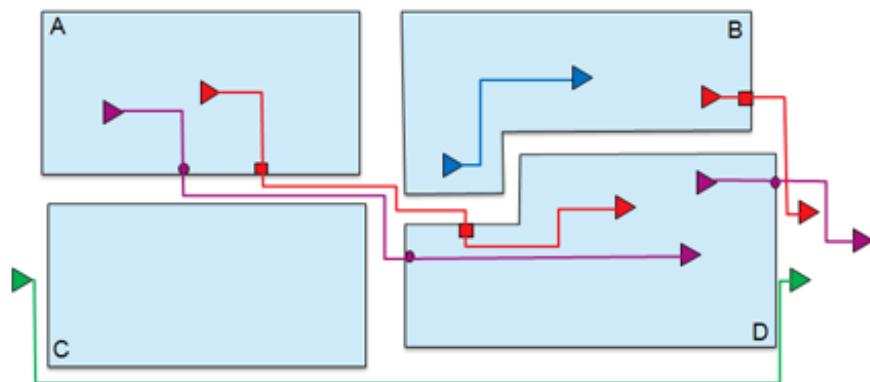
The following table describes the behavior of `trialRoute -handlePartition true -honorPin true` in hierarchical flow.



<b>Intra Partition Net</b>	Routed completely inside its partition.
<b>Pure Top Net</b>	Routed completely avoiding all partitions.
<b>Inter Partition Nets with Placed Pins</b>	Routed through the partition pin, such that: <ul style="list-style-type: none"> <li>The subnets inside partitions are routed within respective</li> </ul>
<b>Inter Partition Nets with Fixed Pins</b>	

## Trial Route: HandlePartitionComplex Behavior with Honor Pin

The following table describes the behavior of `trialRoute -handlePartitionComplex true -honorPin true` in hierarchical flow.



<b>Intra Partition Net</b>	Routed completely inside its partition.
<b>Pure Top Net</b>	Routed completely avoiding all partitions.
<b>Inter Partition Nets with Placed Pins</b>	Routed through the partition pin, such that: <ul style="list-style-type: none"> <li>The subnets inside partitions are routed within respective</li> </ul>
<b>Inter Partition Nets with Fixed Pins</b>	

## Additional Information

## Wire Overlap

In certain situations, wire overlap can occur when using Trial Route. A wire can overlap a routing blockage boundary if the blockage only partially covers the gcell. The covered tracks are counted as blocked tracks that are not available during global routing. However, Trial Route does not record the exact track location, which can result in wires being placed on a track which is already occupied by a routing blockage.

When using nondefault rules, wire width and space are considered during the global routing phase for congestion calculation, before track assignment. Because they are not considered during track assignment, overlapping nondefault wires can occur. However, because spacing was considered during congestion calculation, the routing congestion information is correct.

# What-If Timing Analysis

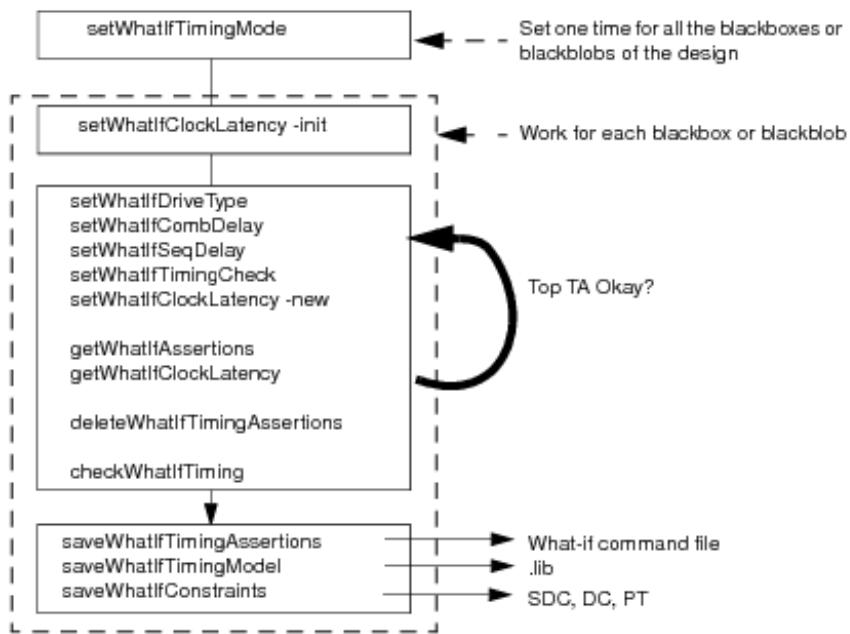
- Performing What-If Timing Analysis
  - Prerequisite
  - Timing Models Supported for What-If Timing Analysis
  - Using the What-If Timing Commands

## Performing What-If Timing Analysis

You use blackboxes in large designs containing hierarchical flows when gate-level details are not available at the beginning of the design cycle. You can easily modify the timing model of a blackbox at the top level because it is not a hard macro. Using the Innovus™ Implementation System, you can make quick modifications to the timing model of a blackbox, and run timing analysis to check the impact of the modifications. This feature is known as what-if timing budgeting. The Innovus software provides what-if timing commands to support what-if timing budgeting. For more information on what-if timing commands, see the chapter [What-if Timing Commands](#), in the *Innovus Text Command Reference*.

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

The following diagram shows the what-if budgeting flow.



## Prerequisite

Prior to using what-if timing commands, you must load the what-if timing models into the database because the what-if timing commands simulate the modifications of the timing arcs.

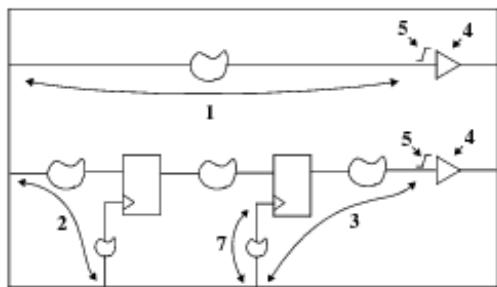
If you do not have timing models in the early design phase, you can use the `setWhatIfClockPort` command to create clock ports. You can then use the clock port to create timing arcs.

## Timing Models Supported for What-If Timing Analysis

The Innovus software supports two timing models for what-if timing analysis: intrinsic and normalized. You can select only one mode at a time.

Figure 10-1 shows the intrinsic timing model.

**Figure 10-1 Intrinsic Timing Model**



The data types associated with the numbers in the Figure 10-1 and the corresponding commands that you use to specify that data are as follows:

#	Data Type	Command
1	Combinational delay from an input port to the input of the driver	<a href="#">setWhatIfCombDelay</a>
2	Delay from the clock input port to the data input port	<a href="#">setWhatIfTimingCheck</a>
3	Sequential delay from the clock input port to the input of the driver	<a href="#">setWhatIfSeqDelay</a>
4	Type of Driver	<a href="#">setWhatIfDriveType</a>
5	Driver input slew	
7	Clock insertion delay to internal registers	<a href="#">setWhatIfClockLatency</a>

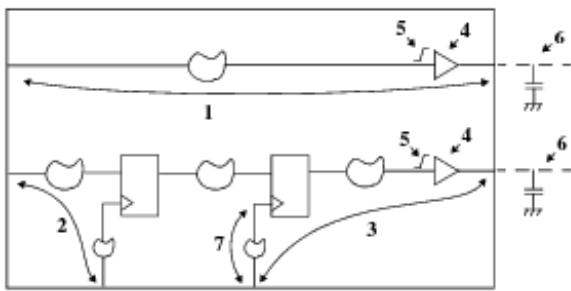
An intrinsic timing model uses the following formula for timing arcs ending on output ports:

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

If you do not use slew specifications in an intrinsic timing model, the timing arc is a 2-D timing table containing input slew and output capacitance dependencies. With slew specifications, the timing arc is only load dependent.

Figure 10-2 shows the normalized timing model.

### Figure 10-2 Normalized Timing Model



The data types associated with the numbers in Figure 10-2, and the corresponding commands that you use to specify that data is as follows:

#	Data Type	Command
1	Combinational delay from an input port to the output port. It includes the driver delay	<a href="#">setWhatIfCombDelay</a>
2	Delay from the clock input port to the data input port	<a href="#">setWhatIfTimingCheck</a>
3	Sequential delay from the clock input port to the data output port. It includes the driver delay	<a href="#">setWhatIfSeqDelay</a>
4	Driver type	<a href="#">setWhatIfDriveType</a>
5	Driver input slew	
6	Total driver output net capacitance	
7	Clock insertion delay to internal registers	<a href="#">setWhatIfClockLatency</a>

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

$\text{Delay} = \text{constant delay} - \text{driver delay}^* + \text{driver delay}$  (look-up table)

Where,

$\text{constant delay}$  = Timing arc delay including driver delay

$\text{driver delay}^*$  = Constant delay considering an input slew and an output capacitance

$\text{constant delay} - \text{clock latency}$  must be greater than  $\text{driver delay}^*$

In a normalized timing model mode driver input slew is always required. In this mode, timing arcs are only load dependant. If you do not specify the driver total output net capacitance, the software

takes real net capacitance into account.

## Using the What-If Timing Commands

You can perform the following tasks with the what-if timing commands:

- Selecting Timing Model

Use the following command to select the timing mode:

- [setWhatIfTimingMode](#)

- Defining generated clocks on internal pins:

Use the following command to create an internal pin and to define a generated clock on the pin.

- [createWhatIfInternalGeneratedClock](#)

- Set the following values on the what-if ports, if required:

- Capacitance
- Maximum capacitance
- Maximum transition
- Maximum fanout

Use the following command to set these values on the what-if ports:

- [setWhatIfPortParameters](#)

By default, the parameters specified with the `setWhatIfPortParameters` command are applied to all ports in the what-if timing analysis model. If you want to apply the values for a particular port, specify the port name with the `setWhatIfPortParameters -port` parameter.

- Selecting the precedence between the values set by `setWhatIfDriveType` command and the values set by the `setWhatIfPortParameters` command

On output ports, parameters such as capacitance value, maximum capacitance values, maximum transition value, or the maximum fanout value can come from the driver (`setWhatIfDriveType` command) or they can be set through the `setWhatIfPortParameters` command.

Use the following command to define which of these values will take precedence in case of a conflict.

- [setWhatIfTimingMode](#)

- **Modifying Timing Arcs**

While what-if commands are the same for both intrinsic and normalized timing models, the delay value specified in the commands for the combinatorial and the sequential timing arcs has different meaning. The driver output net capacitance is a characteristic of the normalized timing model only. Whenever you create or modify a timing arc, the timing graph is updated automatically. The Innovus software recomputes the entire timing arc whenever any of the parameter such as clock insertion delay, timing arc delay or driver type is modified.

**Note:** The timing sense of the driver is taken into account in the combinatorial what-if timing arc description--while applying the drive type, the timing sense of the combinatorial arc is replaced by the timing sense of the driver's timing arc. For sequential arcs, the timing sense is always set to `non_unate`.

Use the following commands to modify timing arcs:

- [setWhatIfDriveType](#)
- [setWhatIfCombDelay](#)
- [setWhatIfSeqDelay](#)
- [setWhatIfTimingCheck](#)
- [setWhatIfClockPort](#)
- [setWhatIfClockLatency](#)

- **Getting Timing Arcs Assertions**

Use the following command to get what-if timing arc assertions:

- [getWhatIfTimingAssertions](#)

- **Saving Timing Arcs Assertions**

Use the following command to save what-if timing arc assertions:

- [saveWhatIfTimingAssertions](#)

- **Deleting Timing Arcs Assertions**

Use the following command to delete the what-if timing arc assertions:

- [deleteWhatIfTimingAssertions](#)

- **Checking Timing Assertions**

Use the following command to check the what-if timing assertions:

- [checkWhatIfTiming](#)

- **Generating what-if timing Models**

After modifying the what-if timing model (in memory) using the what-if command, you can generate an updated timing model (.lib).

Use the following command to generate an updated .lib file:

- [saveWhatIfTimingModel](#)

- **Generating What-If SDC constraints**

The Innovus software generates the what-if timing constraints considering the top-level environment of the blackbox or blackblob. It provides a higher convergence for a top-down flow. The software generates drive, load and transition as IN context. The software generates the input and output delays as OUT context taking into account the last modifications done when you use the what-if commands.

Use the following command to save the What-If constraints:

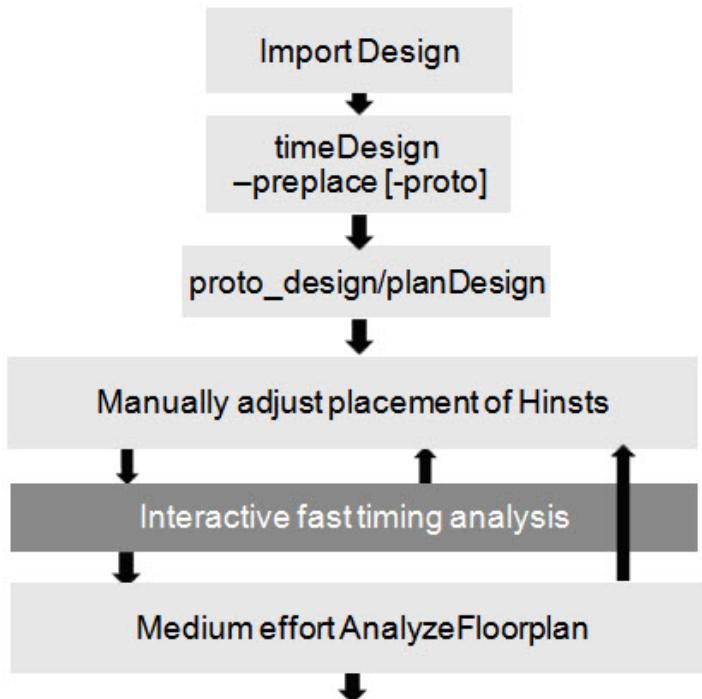
- [saveWhatIfConstraints](#)

# Fast Slack Timing Analysis

- Performing Fast Slack Timing Analysis

## Performing Fast Slack Timing Analysis

The Fast Slack Timing Analysis provides a fast and reliable way to interactively check timing between FlexModels and/or selected modules using GUI. The timing report is based on the reliable psPM model and the slack is calculated between two models. Once the timing between models is initialized, slack of timing path between models is updated if locations of these models are changed.



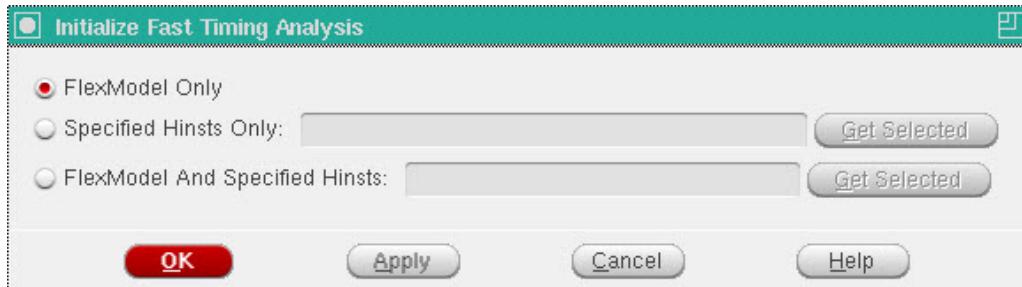
**Note:** It can be used for chip planning stage of a prototyping or a hierarchical implementation flow.

## Initializing Fast Slack Timing Analysis

You can initialize the fast timing analysis by clicking the *Floorplan - Generate Floorplan - Initialize Fast Timing Analysis* menu and choose one of three usage models based on your design.

**Note:** The psPM model should be available before invoking fast slack timing analysis.

Once initialization is done, slack is automatically updated if the location of a model is changed.

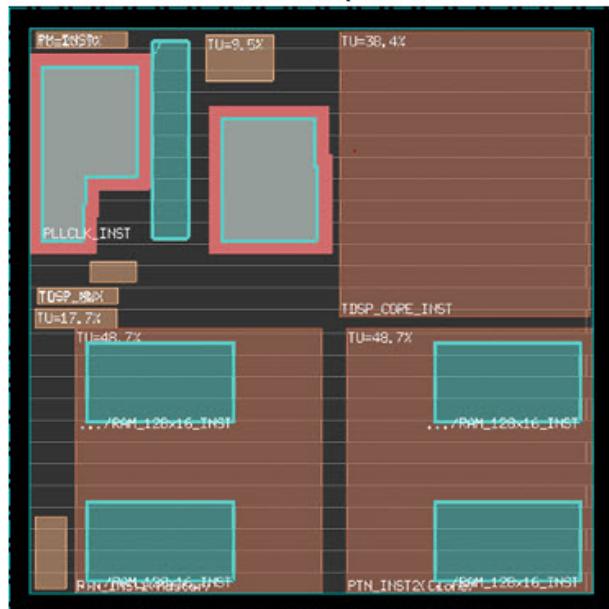


## Examples of Fast Slack Timing Analysis

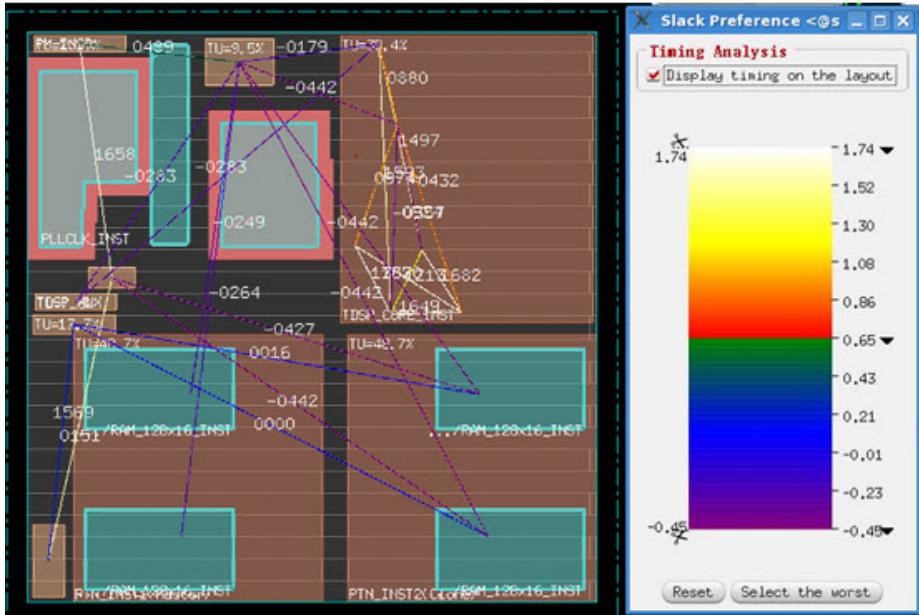
### Example 1

When you choose *Floorplan - Generate Floorplan - Initialize Fast Timing Analysis - Flexmodel only*

The initial floorplan view



### The floorplan view with slack value

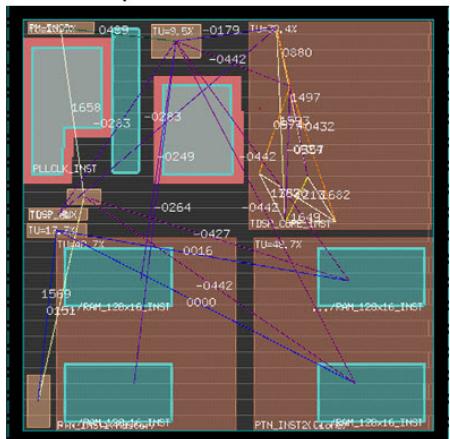


### Example 2

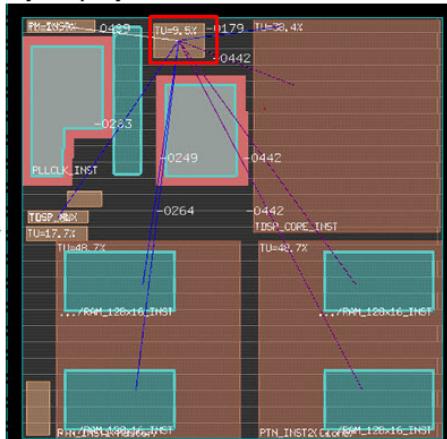
When you choose *Floorplan - Generate Floorplan - Fast Slack Analysis/Display - Display slack for*



The floorplan view with slack value



Only display slack for the selected model



**When the model is moved, slack value is automatically updated**



# Prototyping Methodologies

- [Overview](#)
- [Prototyping Methodologies](#)
  - [Using SoC Architecture Information \(SAI\) for Prototyping Without Netlist](#)
  - [Using FlexModel for Prototyping](#)
  - [Advantages of Using FlexModel Methodology](#)
  - [Creating Hierarchical FlexModels](#)

## Overview

The prototyping methodologies have been designed to handle growing design sizes. It allows you to do productive chip planning and concurrently handle multiple design objectives. This flow has the following capabilities:

- Capacity - This flow handles upto 50 million instances in concurrent timing and congestion-driven mode.
- Turnaround Time - This is a progressively converging flow and enables you to run:
  - Global placement of modules
  - Incremental macro placement
  - Detailed standard cell placementBy creating abstracts of the design to ten times fewer instances than the full netlist, you get ten times faster turnaround.
- GUI - Simplified GUI is provided to obtain abstract timing information relevant to the global context.
- Productivity - It provides a unified correct by construction flow for partitioning, pin assignment, macro placement, feedthrough insertions, and time budgeting.
- Flexible abstractions - This flow contains:
  - Fine grain abstraction for the right mix of capacity and accuracy.
  - The innovative flexfiller connectivity modeling for reasonable placement utilization, routing congestion, and timing estimates.

- Optimization - By using the prototyping foundation flow, you can:
  - Run real optimization on interface paths of the flexModels to improve accuracy.
  - Create models only once and use it multiple times to refine global placement. This makes model generation linearly scalable with an additional CPU.
- Creative heuristics - The flow provides:
  - Lightweight prototype timing engine
  - Timing-driven proto\_design and trialRoute
- Analysis - The prototyping foundation flow provides a simplified design analysis. You can filter large amount of data and visualize the data using various forms of graphical representations.

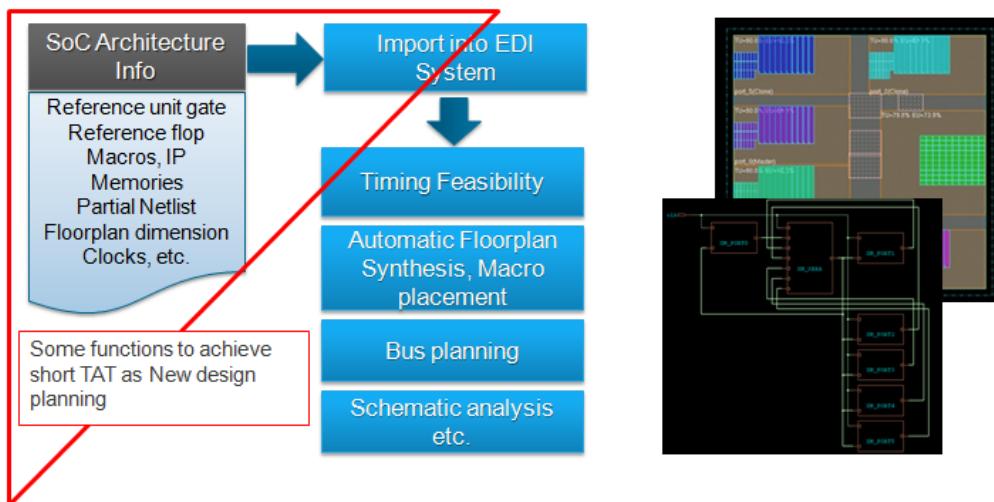
## Prototyping Methodologies

You can use the following methodologies:

- [SoC Architecture Information \(SAI\)](#)
- [FlexModels](#)

## Using SoC Architecture Information (SAI) for Prototyping Without Netlist

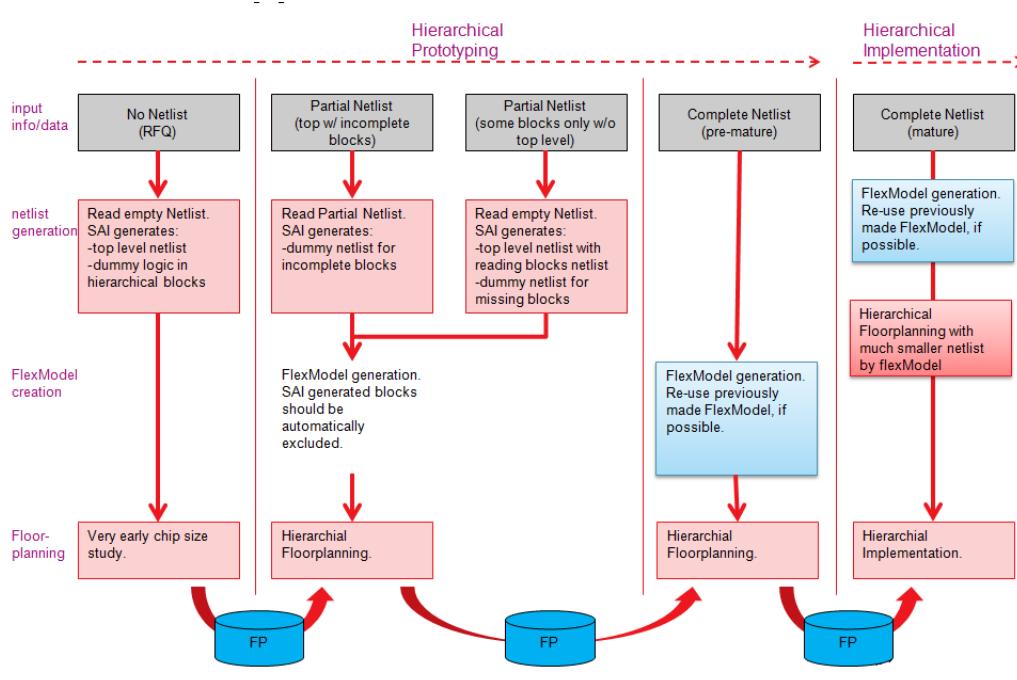
The SoC Architecture Information (SAI) methodology was added in the software's 14.1 release. This feature allows floorplanning and analysis much earlier in the design process when a netlist is not available. SAI methodology allows exploring design feasibility and creates schematics and floor plan with foundry, IP, and die size target information. Additionally, you can turn block diagram information into a simple format, and create a netlist to enable Innovus floor planning. The SAI architecture is shown below:



The SAI file contains the following details:

Chip Architecture and IPs information is captured: Includes reference unit “gate” from the foundry specification, reference flop, macros or special IPs, memory with different sizes and ports, bus connection, soft modules (partitions), existing partial netlist, clocks, and floorplan dimension. Additionally, the software can also convert the block diagram language into a simple format.

## Possible Application of SAI/FlexModel Flows



## Enabling SAI Mode

To enable/disable the SAI mode, you can use the following commands:

- `read_sai`
- `end_sai`

For more information on SAI commands, refer to the "SAI Commands" chapter in the *Innovus Text Command Reference* manual.

## Enabling SAI in Batch Mode

To enable SAI in batch mode, you can use the following commands:

```
setUserDataValue init_lef_file ...
setUserDataValue init_mmmc_file ...
setUserDataValue init_verilog ... #partial netlist
setUserDataValue init_top_cell <my_top_cell_name>
read_sai demo.sai
```

In the above example, `read_sai` enables you to interactively use SAI commands on the prompt or in turn, use it within a run script to execute SAI steps. In the batch mode, you can directly use `read_sai demo.sai`. The content of `demo.sai` (used to build design content through SAI commands) is shown in the section [Sample SAI File](#) below.

Once completed, you can end the SAI session with the `end_sai` command.

## Enabling SAI in Interactive Mode

For interactive session, you can just enter `read_sai` to invoke SAI capabilities and follow up with individual SAI commands on the prompt to build the database.

```
read_sai
```

Type in any SAI related commands you want to use. For example `add_module`, `connect`, `add_macro`, ...

You can also use the `-help` option to check the SAI command options. Once completed, you can end the SAI session with the `end_sai` command.

### Notes:

- `end_sai` terminates the SAI input in both interactive or SAI script based commands.
- You can use the `timeDesign -prePlace` parameter to check if timing is ready. Then you can open GUI to floorplan modules.

## Creating Automatic Netlist with SoC Architecture Information (SAI)

Based on the SAI, you can create a netlist to enable Innovus floorplanning. This provides the following features:

- Parse a partial netlist and the SAI file and to create a netlist.
- Add dummy cells to mimic the size of the define modules (RFQ).
- Add dummy flop to mimic the boundary connection from module to module.
- Add dummy memory or the real memory in order to assist sub-chip floorplan.
- Add pipe line stage registers per the specification in the connection file.
- Create SDC timing constraints file for the defined boundary connection and read into Innovus.
- Define the die size and read into Innovus.
- Create all the net groups and pipeline net groups.
- Handle master-clone, where connection model is single-driver but multiple receiver.

## Partial Netlist Support

The SAI commands can be used to generate top level netlist or add new modules or modify the existing ones. Consider the following examples:

- Top level netlist is ready but consists of some incomplete modules

Incomplete modules might have only port definition or have some logic inside. The SAI command `add_module` can be used to fill those modules with pseudo logic incrementally, as shown below:

```
add_module xcmUnit1 -cell XM_PORT$i -gate_count 5000
-memor y_bit_count 8192 -util 0.4 -aspect_ratio_range {0.5 2.0}
-flop_count 450 -flop_ref_clock clk
```

You can create a net connection by using the following command:

```
connect xcmUnit1/out -to xsy/in -clock clk -bus_width 256 -pipeline_stages 2
```

Module ports can be created on demand when the `connect` command is used.

- There is no top level netlist but some modules for partition blocks are ready. The SAI command `add_module` will generate top level netlist and add existing modules.

For more information on SAI commands, refer to the "SAI Commands" chapter in the *Innovus Text Command Reference* manual.

## Sample SAI File

```
set_sai_version 1.0
set_ref_flop DFFSRX4 ;# define a Flop
set_ref_gate CLKBUFX8 ;# define the basic "unit gate" cell
set_ref_memory mem1 -bit_size 1024 -area_per_bit 1.1 -aspect_ratio 0.75
for {set i 0} {$i<6} {incr i} {
    add_module iport$i -cell XM_PORT$i -gate_count 5000 -memory_bit_count 8192 \
    -util 0.4 -aspect_ratio_range {0.5 2.0} -flop_count 450 \
    -flop_ref_clock clk
}
add_module xsw -cell XM_XBAR -gate_count 8000 -memory_bit_count 4096 -util 0.5 \
-aspect_ratio_range {0.5 2.0}
add_macro -cell XM_PORT0 {ram_128x16A 10 rom_512x16A 20}
add_macro -cell XM_PORT1 {ram_128x16A 10 rom_512x16A 10}
add_macro -cell XM_PORT2 {ram_128x16A 10 rom_512x16A 20}
add_macro -cell XM_PORT3 {ram_128x16A 10 rom_512x16A 10}
add_macro -cell XM_PORT4 {ram_128x16A 10}
add_macro -cell XM_PORT5 {ram_128x16A 10 rom_512x16A 10}
add_macro -cell XM_XBAR -memory mem1
for {set i 0} {$i<6} {incr i} {
    connect iport$i/out -to xsw/in$i -clock clk -bus_width 256 -pipeline_stages 2
    connect xsw/out -to iport$i/in -clock clk -bus_width 256
}
add_clock clk -period 6.0 -waveform {0.0 3.0} -buffer CLKBUFX8
set_floorplan -aspect_ratio 1.0 -util 0.2 -side_spacing 80.0
```

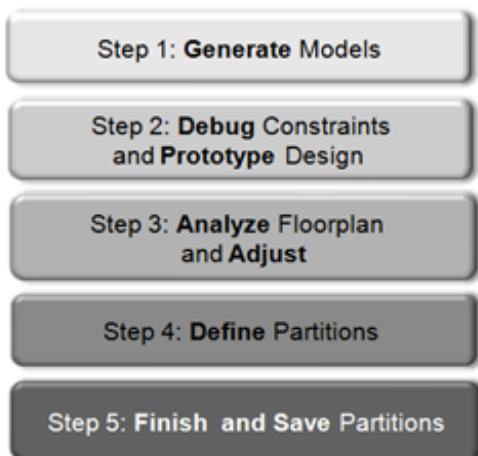
## Using FlexModel for Prototyping

The prototyping foundation flow runs in the following stages:

- Generating Models
- Debugging Constraints and Prototype Design
- Analyzing Floorplan and Adjust

- Defining Partitions
- Finishing and Saving Partition

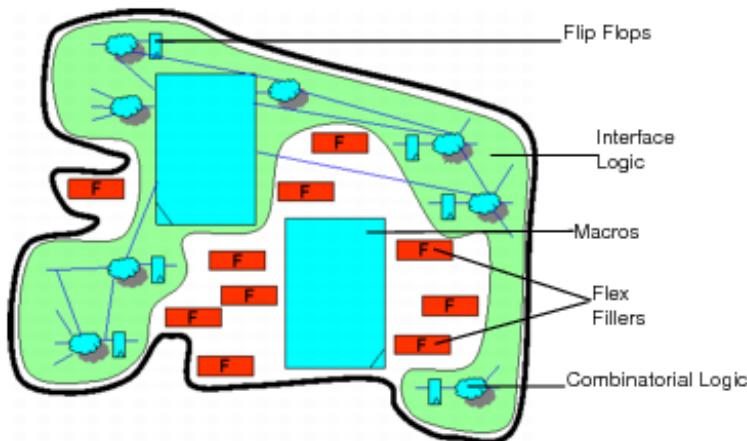
The following diagram shows the various stages of prototyping foundation flow.



## Generating Models

This is the first step in the prototyping foundation flow. In this step, you identify, create, and replace existing modules in the design with physical and timing models on disk for each identified model.

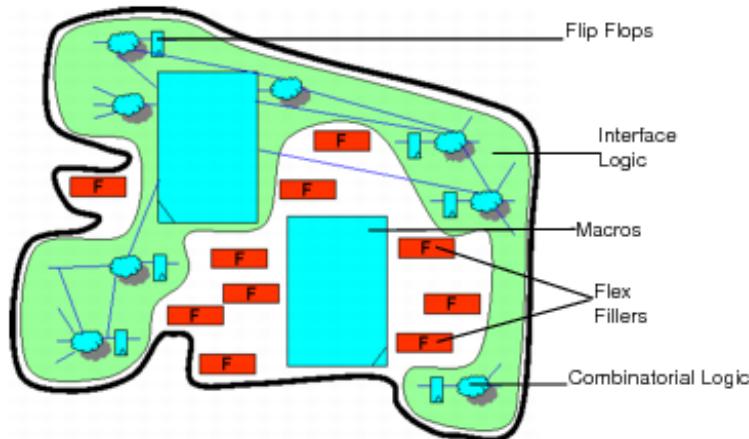
A flexModel is a Verilog netlist that can contain macros, interface standard cells, and flex fillers. The flex fillers reserve space for the removal of internal register-to-register logic. The flex fillers have no timing model associated with them and they connect such that the model holds together during placement. So the placer will place them together in one group.



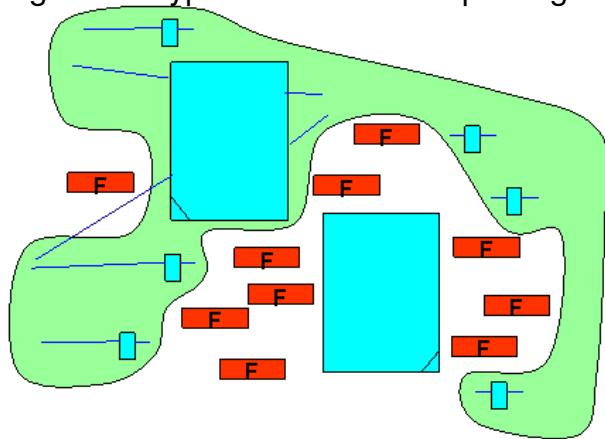
A flexModel netlist is usually one tenth the number of instances of its full netlist. It is used during

early design planning to reduce the run time and memory while accurately modeling the timing and area.

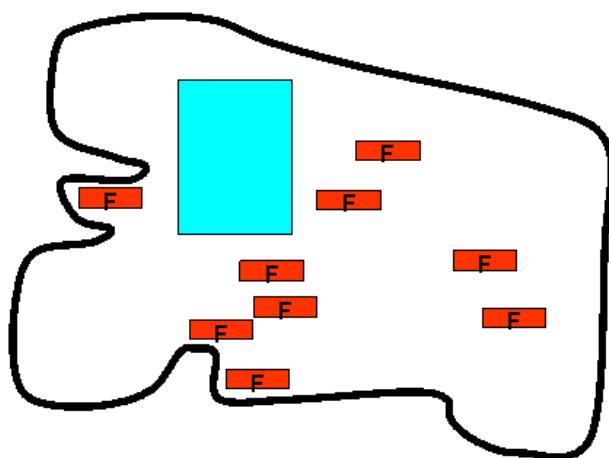
A flexModel can be created even in early stages of the design where the netlist is in its early stages. If your netlist is a full/partial netlist which has combinatorial logic, then the flexModel will have the interface logic, the macros as well as the flex fillers. This will help in accurate timing and area estimation.



If your netlist is a skeleton netlist (that is it contains only flops at the model's interface ports), then your flexModel's interface logic will only consist of flops (no gates). There will no combinatorial logic. This type of netlist will help find gross timing problems.



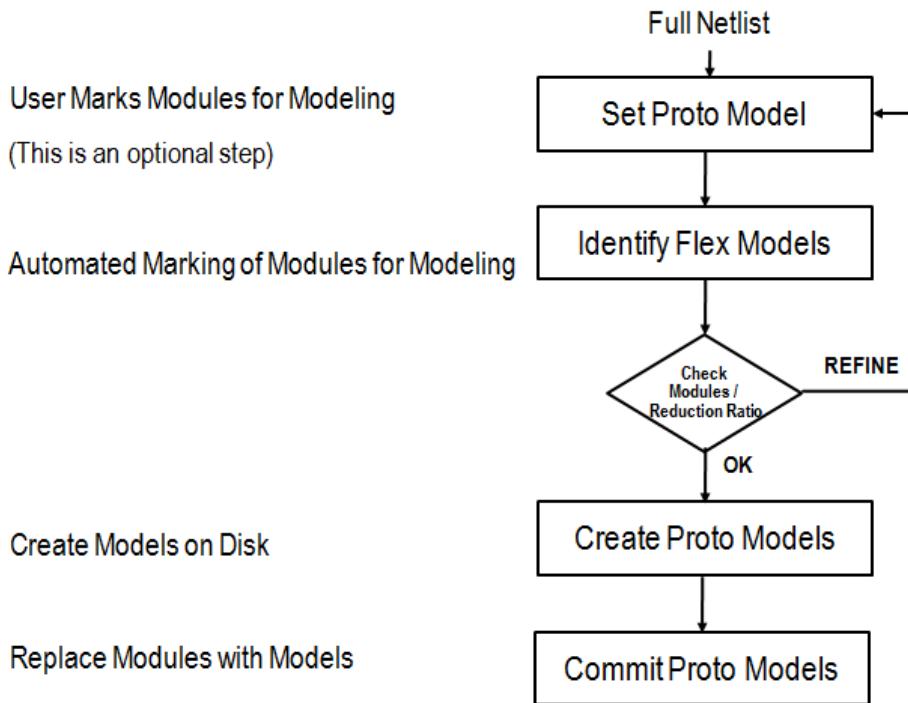
In case of empty netlist, your flexModel will have only the macros and flex fillers. Using empty netlist, you can solve only the congestion problem but you cannot estimate timing.



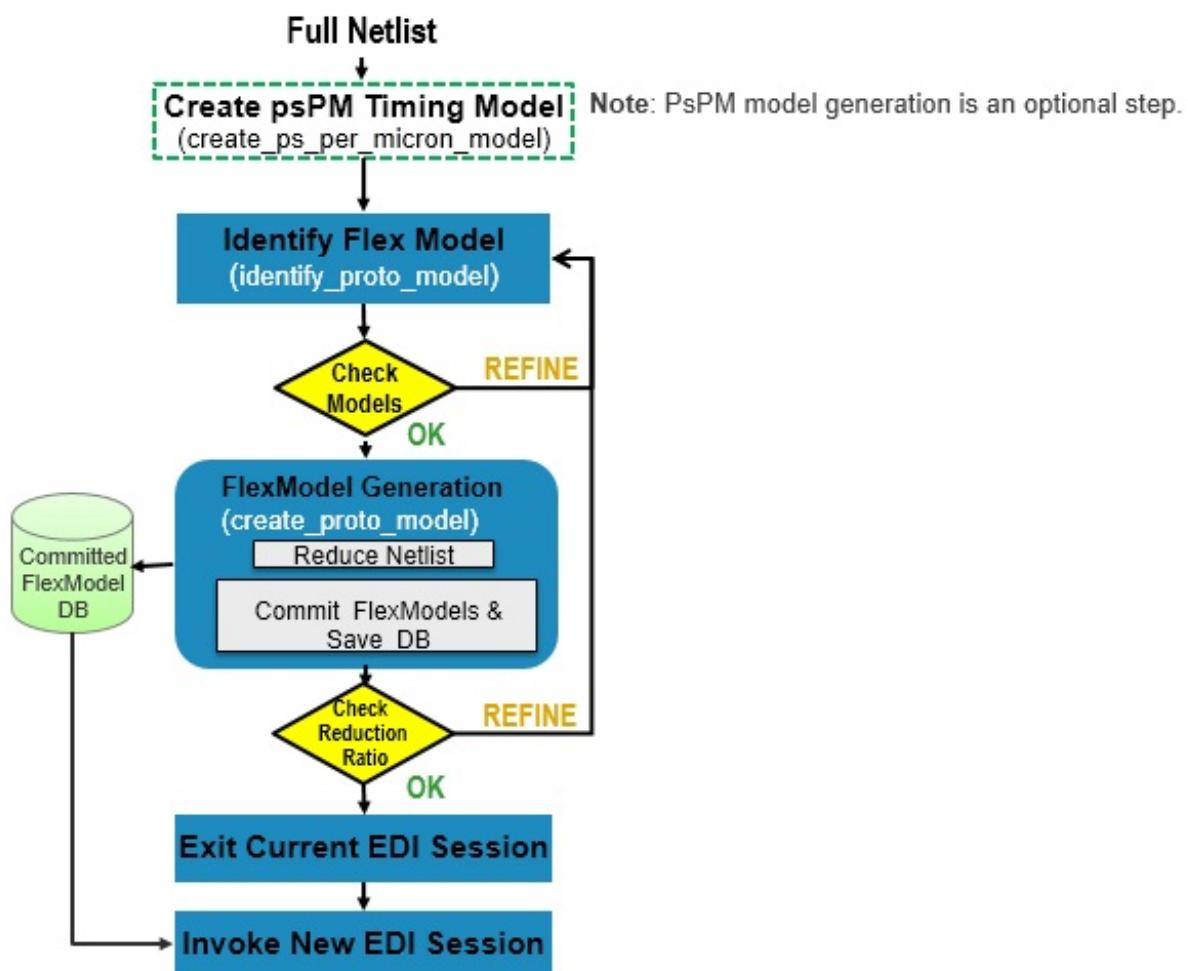
Following is the comparison table between different supported Innovus abstractions:

	LEF/.LIB	LEF/ILM	BBOX/.LIB	FlexModel
<b>Physical Model</b>				
Flexible shape			✓	✓
Flexible amoeba				✓
Auto-pin place			✓	✓
Pin place anywhere				✓
Models internal congestion				✓
Placeable internal macros				✓
Placeable interface stdcells				✓
<b>Timing Model</b>				
SDC reference internal pins?		✓		✓
Accuracy		✓		✓
<b>Memory (&lt;1/10 of full chip)</b>				
Runtime (<1/10 of full chip)	✓	✓	✓	✓

## The Model Generation Flow



**Note:** If Active Logic Reduction technique (ART) is used for FlexModel generation then "Commit Proto Models" step is not needed since Innovus will automatically replace original full netlist with new FlexModel information and save a committed FlexModel design. Once ART based FlexModel generation is done, you should exit the current Innovus session and restore the saved committed FlexModel design with a new Innovus session.



ART based FlexModel generation flow support to define a partition as a FlexModel since software's 14.2 release.

- In the software's 14.1 release, add the following setting into /enc.tcl file:  
`set mib::allow_partition_as_flexModel 1`
- In the software's 14.2 release, use the following command to turn on this feature:  
`set_proto_mode -allow_partition_as_flexModel true`

After the above setting, add the FlexModel settings before running `create_proto_model` as following:

```
set_proto_model -model <ptn1> -type flex_module
set_proto_model -model <ptn2> -type flex_module
```

Or

```
foreach ptn [dbGet -e top.ptns.master.cell.name] {
```

```
set_proto_model -model $ptn -type flex_module
}
```

**Note:** `identify_proto_model` must not be used when defining partition as FlexModel.

## Examples

- The following example shows the use of `set_proto_model` command using which you can mark the modules for modeling.

```
set_proto_model -model prog_bus_mach -type flex_module
set_proto_model -model port_bus_mach -type flex_module
set_proto_model -model mult_32 -type flex_module
set_proto_model -model mult_32 -create_total_area 9770
get_proto_model -all

module      type      source  create_total_area
prog_bus_mach  flex_module user
mult_32      flex_module user   9770
port_bus_mach  flex_module user

get_proto_model -all -tcl
{{name prog_bus_mach} {type flex_module} {source user}}
{{name port_bus_mach} {type flex_module} {source user}}
{{name mult_32}      {type flex_module} {source user} {create_total_area 9770}}
get_proto_model -type_match {flex_module} -name -tcl
prog_bus_mach port_bus_mach mult_32
```

- The following example shows the use of `identify_proto_model` command that is used for automated marking of modules for modeling.

```
set_proto_mode -identify_min_insts    205
set_proto_mode -identify_max_insts    4190
identify_proto_model
```

- The following example shows the use of `create_proto_model` command that is used to create models on the disk.

```
setMultiCpuUsage -remoteHost 2 -cpuPerRemoteHost 1
setDistributeHost -local
```

```
create_proto_model  
report_proto_model -created
```

- Following example shows the use of ART based model generation. The committed FlexModel design is saved into DBS/model\_gen.enc.dat directory

```
create_proto_model -out_dir DBS/model_gen.enc
```

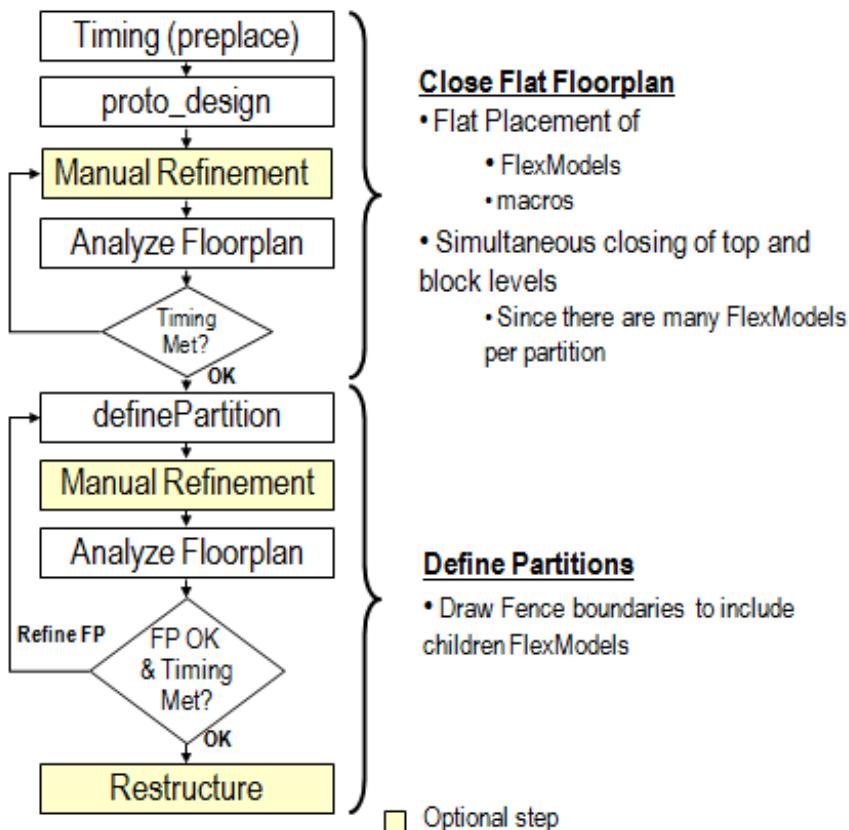
- The following example shows the use of `commit_proto_model` command that replaces modules with the prototyping models. This step should be NOT be run for ART based FlexModel generation.

```
commit_proto_model  
report_proto_model -committed
```

For more information about the prototyping foundation flow commands, see the [Prototyping Foundation Flow Commands](#) chapter of the *Innovus Text Command Reference*.

## Debugging Constraints and Prototype Design

This is the second stage of prototyping foundation flow. In this stage, first you need to close timing on the flat floorplan by placing the flexModels. Since there are many flexModels per partition, so you will also simultaneously close the block-level placement of the flexModels. Second, you need to define the partitions and draw fences around the already placed flexModels.



## Closing the Flat Floorplan

You close a flat floorplan by:

- Running `timeDesign`
- Running `proto_design`

### ***Running timeDesign***

To begin closing the flat floorplan, you first need to perform `timeDesign` pre-placement. This is done to check the initial timing and constraints.

```
timeDesign -proto -preplace
```

The `timeDesign -proto` parameter allows you to call the prototype timing if flexModels are present.

By default prototype `timeDesign` assumes:

- No detours
- No pin or wire overload
- Best routing layers used (that is the lowest RC delay)
- Runs very fast and is useful for debugging constraints

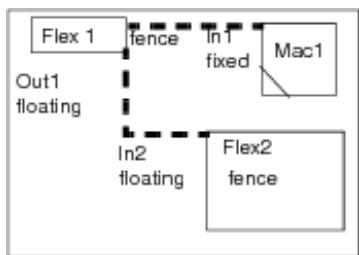
The prototyping foundation flow accepts `set_proto_mode -timing_ps_per_micron`. This parameter represents the amount of total delay (buffer and wire) that a technology can drive a signal at a certain distance. If value of `-timing_ps_per_micron` is not specified Innovus will automatically create a pico-second per micron model (`psPM.model`) that is based on net length and layer for timing estimation.

**Note:** This `psPM.model` correlates well with `optDesign`.

The `timeDesign -proto` parameter always uses the best intrinsic gate delays.

## Examples

- Consider a design in which `flexModel`, `Flex1`, has terminal pins `Out1` and `In1`. If you run `planDesign` without running standard cell placement, then the standard cell driving `Out1` within `Flex1` is unplaced. If the macro `In1` terminal location is known, then the distance is calculated as the minimum manhattan distance between the `Flex1` fence and the `In1` terminal.



If a standard cell containing `In2` within `Flex2` standard cell is unplaced, then the distance is calculated as the minimum distance between the two flexModels.

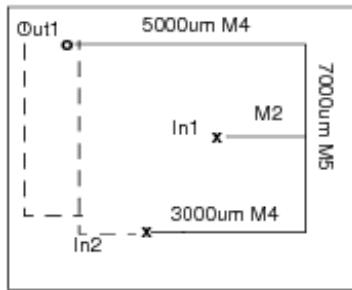
The delay is zero for any case where one of the terminals is of an unplaced standard cell which does not have a parent guide/region/fence (that is, neither the standard cell nor its parents have any placement information).

- If `placeDesign` is run, and if
  - the value of `get_proto_mode -timing_net_delay_model` is `best_delay_no_detour`

(default value), and

- set\_proto\_mode -timing\_ps\_per\_micron is set to {M1:M5 0.25 M6:M7 0.14} then, timeDesign -proto assumes no detour on the least delay layers. In such a case, the Out1 to In2 delay is calculated as:

M6delay + M7delay  
 $0.14 \times 2000 + 0.14 \times 7000 = 1260\text{ps}$



If value of -timing\_net\_delay\_model is use\_actual\_wire, and if no routing exists, the delay calculation is same as with best\_layer\_no\_detour value. Else, actual routing distances and layers are used for calculating the delay:

+ M5delay  
 $0.25 \times 8000 + 0.25 \times 7000 = 3750\text{ps}$

## ***Running proto\_design***

In the next step, you need to run timing-driven proto\_design. This command internally call planDesign to place flexModels.

To do this, set the following:

```
setPlanDesignMode -effort medium -useFlexModel fence
setPlanDesignMode -timing true
```

The -useFlexModel fence parameter:

- forces planDesign to keep flexModels from overlapping.
- forces planDesign to place a flexModel's hard macros within the flexModel fence.

- forces `placeDesign` to place standard cells within the `flexModel`.

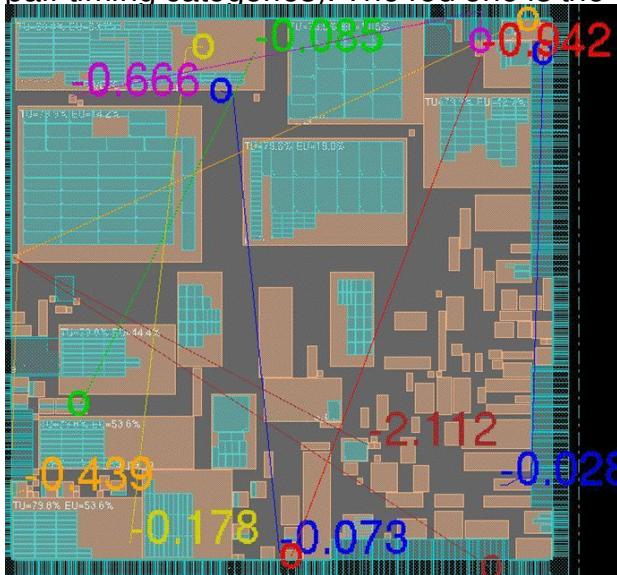
## Analyzing Floorplan and Adjust

After `proto_design`, some manual refinement is usually necessary, followed by running these commands:

```
checkFPlan  
analyzeFloorplan -fp
```

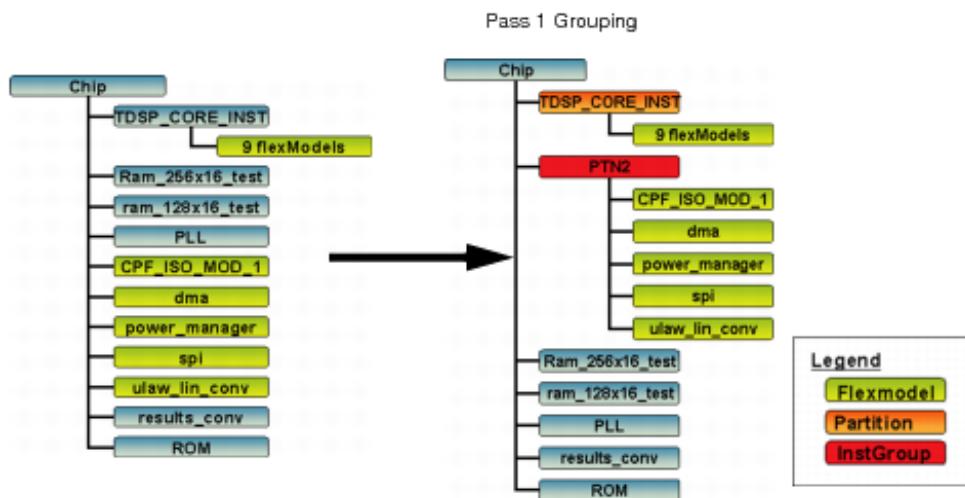
The prototyping timing flow is fast as the calls to `trialRoute` and `optDesign` are not required. Buffering is also not needed since long wire delay is estimated using the `timing_ps_per_micron` parameter or `psPM.model` information. The flow is accurate since the `flexModel`'s interface logic has been fully optimized once during model creation, that is, even the short interface paths of a `flexModel` are optimized to be as short as possible.

The `load_timing_debug_report -proto` command is automatically called by `analyzeFloorplan`. It creates timing categories for each `flexModel` pair. These timing categories are color-coded and thousands of timing paths are condensed into few paths. For example, in the figure below, thousands of timing paths are condensed only into eight paths (the top path from the eight `flexModel` pair timing categories). The red one is the worst timing path.



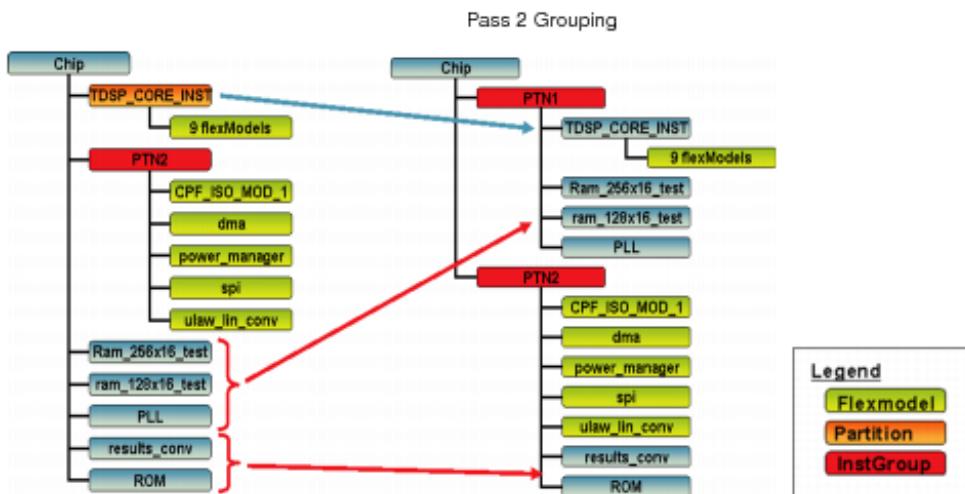
## Defining Partitions

After running `planDesign` and getting a flat flexModel placement that closes timing, you need to define partitions. In the first pass for defining a partition, it is recommended to use the modules from the original design hierarchy. For example, in the design hierarchy shown below, the `TDSP_CORE_INST` partition has nine flexModels. There are other flexModels also. Since there are many flexModels at the top level of the design example below, to create a partition to hold them, an instance group can be created, which a restructuring step will change the netlist by changing that instance group into a module. So you define an instance group named `PTN2` and define all the flexModels within that instance groups.



Once the instance group `PTN2` is created, the next step is to draw its fence using `generate_fence` command. To get a report of the flexModels and their hierarchical names to decide if an instance group is right for your design, use the `report_proto_model -identified` command.

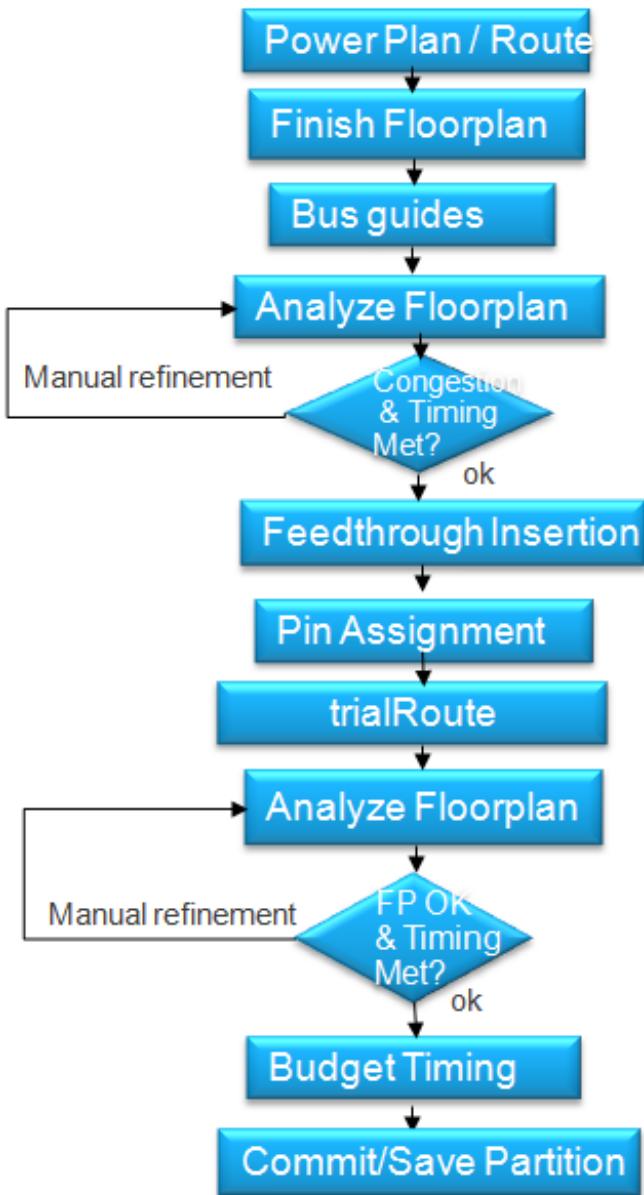
In the second pass, you define the partitions and perform manual refinement. Here you delete the `TDSP_CORE_INST` partition and create a new instance groups `PTN1`.



Finally you restructure your netlist using `runRcNetlistRestruct -proto`

## Finishing and Saving Partition

This is the last stage of running the prototyping foundation flow. In this stage, you analyze the floorplan to make sure that design is routable and has met timing requirements. To begin, you first create the power structure of a design and finish the floorplan so that you can perform detailed placement. You can also create the bus guides to guide the routing of critical nets. After doing this, you should run `analyzeFloorplan -congestion -timing_aware_route` to check that there are no congestion problems and your design meets the timing requirements. After the floorplan is verified, then you need to perform feedthrough insertion, pin assignment and run trialRoute honoring the assigned pins. Then you need to check the timing again before generating the timing budget for the block. Finally, you save your design.



The finish and save partition stage can be divided into the following tasks:

- Finishing the Floorplan
- Analyzing the Floorplan
- Feedthrough Insertion
- Pin Assignment
- Running timeDesign
- Budget Timing

## Finishing the Floorplan

While finishing the floorplan, you can create the power structures using the `sroute` command. To prepare the floorplan for detailed placement and feedthrough insertion, you can add the block halo for your macros, add standard cell row blockages around partition fences to reserve the area for feedthrough insertion, and add partial placement blockages between macros and boundaries to handle congestion.

## Analyzing the Floorplan

The analyze floorplan step with `-congestion` and `-timing_aware_route` comprises of three steps:

- Cell placement

Following are the placement settings:

```
setPlaceMode -fp false -doRPlace true  
placeDesign -noPrePlaceOpt
```

- Timing-driven trialRoute

- Honors partition pin constraints while routing the top and inter-partition nets. The default partition-to-partition routes skip every second track on the boundary.

Assign net weight for critical nets

- Default is 2
  - Critical is 4

- Mark critical nets to be routed on layers no lower than specified by user (that is, reserves faster layers for critical nets)

- Use `set_proto_mode -preferred_bottom_layer` parameter

- Route higher weighted/critical nets

- minimize detouring on critical nets

- Invoke `timeDesign` for timing reports

- Use actual routes for delay calculation

See Also [Calculating Delay Using Timing-Driven trialRoute](#)

- Display congestion report: Run `describeCongestion` to aggregate the congestion information into horizontal congestion and vertical congestion super gcells, and output a congestion report.

<b>-congestion</b>	<b>-timing_aware_route</b>	<b>Behavior</b>
On	Off	<ul style="list-style-type: none"> <li>Default placeDesign</li> <li>Default trialRoute</li> <li>Use actual wire for delay calculation</li> </ul>
Off	On	<ul style="list-style-type: none"> <li>Four iterations of timing-driven trialRoute</li> <li>Use actual wire for delay calculation</li> </ul>
On	On	<ul style="list-style-type: none"> <li>Default placeDesign</li> <li>Four iterations of timing-driven trialRoute</li> <li>Use actual wire for delay calculation</li> </ul>

To do this, there are two use models:

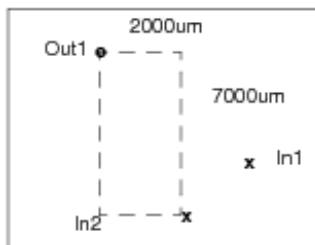
- First Model: Run `analyzeFloorplan` with `-congestion` to check congestion and then run `analyzeFloorplan -timing_aware_route`
- Second Model: Run `analyzeFloorplan -congestion -timing_aware_route`

**Note:** The first model is recommended since you can find congestion problems and fix them early without waiting for full run time.

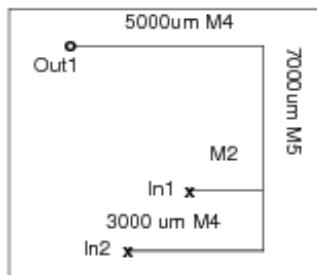
## ***Calculating Delay Using Timing-Driven trialRoute***

In a typical design described in the following example, you might see that the initial trialRoute results have shown the worst negative slack of `-1ns` versus `0ns` for the timing-driven trialRoute due to the following reasons:

- Before trialRoute, `timeDesign -proto` calculates the delay from `Out1` to `ln1` by:
  - finding the bounding box that contains two terminals.
  - multiplying half a perimeter by the best delay factor.
 For example,  $0.14 * (2000 + 7000) = 1260\text{ps}$  as shown below

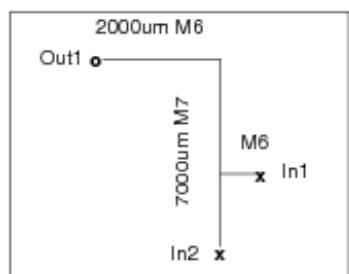


- assuming best and non-detouring layers
- After running the initial iteration of trialRoute, then the actual routing layer is used and the layer length is multiplied by its delay factor. This results in more delay as compared to the earlier stage.  
For example, M4 delay ( $0.25 * 8000$ ) + M5 delay( $0.25 * 7000$ ) = 3750ps



- Next, the tool automatically determines the critical nets and weigh them so that they can be routed first to minimize the tour or no-detour, and also set the correct bottom routing layer to use the routing layer with less delay. So by the fourth iteration, the delay is similar as you predicted early in the flow.

It is calculated as: M6 delay ( $0.14 * 2000$ ) + M7 delay ( $0.14 * 7000$ ) = 1260 ps

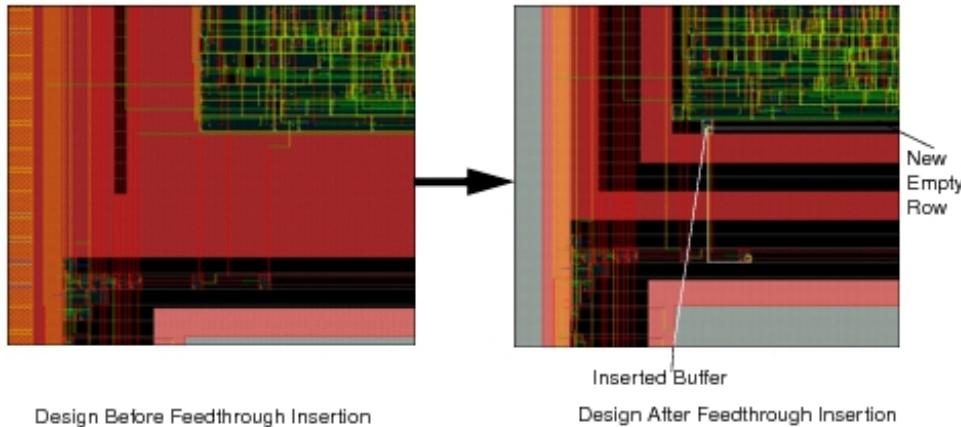


## Feedthrough Insertion

After solving the congestion problems, you perform feedthrough insertion. In this step, you:

- Delete two rows of placement blockages around partition fence (Two inner row and two outer row blockages) to provide space for inserted buffers.
- Run `insertPtnFeedthrough`
- Run `trialRoute` with `-keepExistingRoutes` to re-route old/new feedthrough nets and any existing top and/or inter partition nets that are changed by `refinePlace` during feedthrough

insertion.



For example,

```
createPlaceBlockage -allPartition -innerRingBySide {2 2 2 2} -outerRingBySide {2 2 2 2}

set feedThruFile [get_proto_mode -tmpDir -quiet]/InsertFeedThru_netMap.txt
insertPtnFeedthrough -routeBased -netmapping $feedThruFile -doubleBuffer

#re-route new nets created by inserted buffers:
trialRoute -keepExistingRoutes -routeBasedBBPin
deselectAll ;#since -keepExistingRoutes selects nets to be ECO routed.
```

## Pin Assignment

After running trialRoute, you perform pin assignment. To do this:

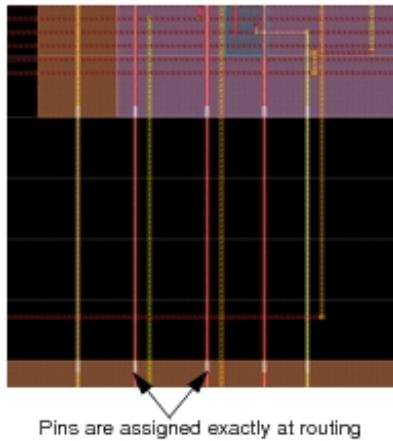
- Assign pin locations exactly where routing crosses the partition boundaries.
- Check assigned partition pins.
- Reassign pins which are overlapping or short.
- Report unaligned partition pins.

For example,

```
assignPtn -enforceRoute
checkPinAssignment

legalizePin -keepLayer -verbose
reportUnalignedNets -ptnToPtn {unaligned} -noTopToPtn \
-rptFile [get_proto_mode -tmpDir -quiet]/ptnToPtn_unaligned.pins
```

```
reportUnalignedNets -ptnToPtn {layerMismatch} -noTopToPtn \
-rptFile [get_proto_mode -tmpDir -quiet]/ptnToPtn_layerMismatch.pins
```



## Running timeDesign

In this step, do the following:

- Run final trialRoute with `-handlePartitionComplex` by honoring partition pins (`-honorPin`) to check congestion.
  - Honor partition pin constraints while routing top and inter-partition nets.
- Run `timeDesign` to get the final timing report.

For example,

```
setTrialRouteMode -handlePartitionComplex true -honorPin true
trialRoute
saveDesign assignPin.enc
timeDesign -proto -expandedViews
load_timing_debug_report -proto
```

## Budget Timing

To perform timing budgeting, you have a choice to do psPM timing budgeting or based on optDesign. Recommend to run psPM budgeting for quick turn-around time.

- Run `timeDesign -proto`
- Derive timing budgets for partitions in a design.

For example,

```
timeDesign -proto; load_timing_debug_report -proto
set tbgMMCUseSameTGraph 1
set_global timing_support_hierarchical_pin_constraints true
set tbgWriteLLatencyPerClock 1
set tbgPrintExceptionInfoInJustify 1
setBudgetingMode -noHoldView true
setBudgetingMode -trialIPO false
deriveTimingBudget -justify
```

## Save Design

In this step, the flexModel setting will be saved in the saved design directory if a design has flexModels. This information will be restored by `restoreDesign` command to load flexModel netlists from directory specified with `set_proto_mode -create_dir` parameter and to set relevant information for flexModels.

## Commit Partition

To commit a partition, convert partition fences to partition blocks and push down top-level floorplanning data into partition block level designs for accurate block implementation using `partition` command.

**Note:** If a partition module has flexModel(s), then the partition block-level design will still have flexModels.

## Save Partition

During this step, data is generated for the top-level and block-level implementation. If there are flexModels at the top-level, then the saved top-level netlist will contain flexModels. If there are no flexModels outside partitions, then the top-level netlist will be the original full netlist.

Similarly, if a partition has flexModels, then its partition netlist will contain flexModels.

## Restore Design

A flexModel design can be restored with `restoreDesign` command. Besides restoring the existing data, it will also restore flexModel setting information to load appropriate flexModel netlists and the required model information. To have correct setting for a FlexModel design, load the `procs.tcl` script. For example,

```
source SCRIPTS/PROTO/procs.tcl
restoreDesign model_gen.enc.dat dmtf_recvrx_core
```

To restore a flexModel design to original full netlist, use the following steps:

1. Use `set_proto_model` command to change flexModel netlist type to full for each flexModel module.
2. Use `commit_proto_model` command to commit the change(s).

Example: Partition block design PTN2 has four flexModel modules. They are `ulaw_lin_conv`, `CPF_ISO_MOD_1`, `spi`, and `dma`:

```
restoreDesign DBS/PTN_L1/PTN2.enc.dat dmtf_recvrx_core
set_proto_model -model ulaw_lin_conv -type full
set_proto_model -model CPF_ISO_MOD_1 -type full
set_proto_model -model spi -type full
set_proto_model -model dma -type full
commit_proto_model
```

**Note:** Cadence recommends you to convert models to a full netlist at the partition-level block design instead at the full chip level before partitioning.

## Advantages of Using FlexModel Methodology

Following are the advantages of using the prototyping foundation flow:

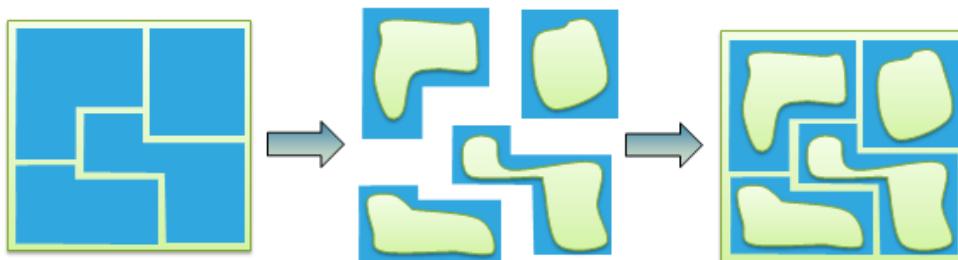
- FlexModels reduce the instance count to 1/10th that of full netlist.
  - Allow designs upto 100 million instances
  - Significantly improve the run time and memory
- Using flexModels, you can perform the floorplanning of top level and partitions simultaneously. Many flexModels per partition provide visibility into a partition's macros for congestion analysis and internal timing details.
- This flow allows accurate chip-level timing.
  - A flexModel is created with all of its I/O paths optimized to be as fast as possible

- Short paths are optimized
  - Eliminates re-spin of models with different budgets
  - Models are created only once and then used for many turns of the floorplan
  - No top-level optimization is needed. It is done only once during the model creation.
- This flow minimizes the partition/channel resizing.

## Creating Hierarchical FlexModels

For generating FlexModels you need to load a full netlist which utilizes a lot of memory. To improve memory usage, Innovus provides the ability to create FlexModels for each partition block netlist, if the information is already available. Flexmodels for each partition can be loaded back to the top level design.

The following figure shows how FlexModels can be created at each partition block and loaded into the top level design to reduce memory usage:

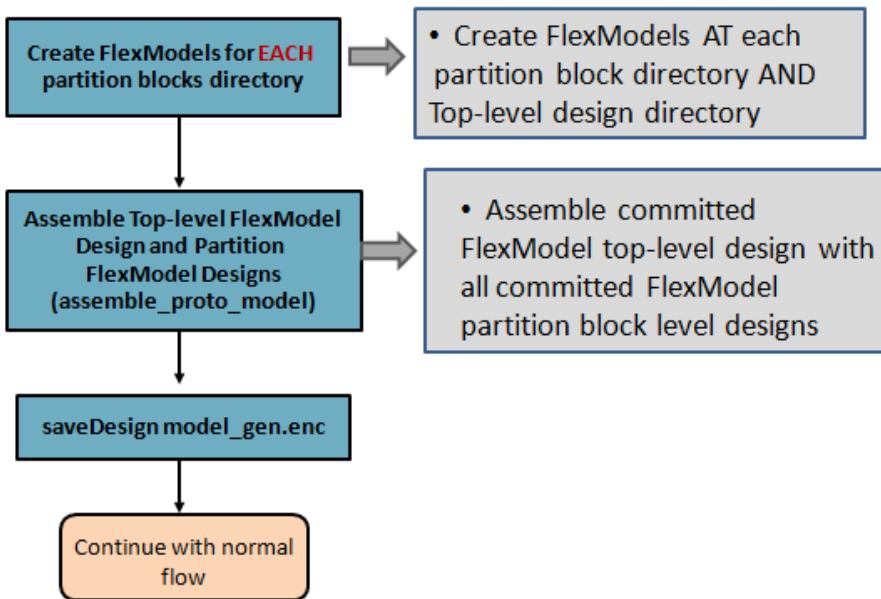


## Hierarchical FlexModel Generation Flow

You can load a full chip FlexModel netlist of a design using the following flow:

1. Create FlexModels for partition block using an existing model generation flow at the partition block.
2. Create FlexModels for top-level netlist, if required.
3. Run the `assemble_proto_model` command at the current run directory where full chip FlexModel netlist is to be loaded.

The following diagram shows the hierarchical FlexModel generation flow:



## Sample Script for Creating Models at Partition Blocks

To improve run time, you can use multiple CPUs to run block model generation in parallel:

```

set ptnName "tdsp_core"
set minInst 100

cd PTN_dir/$ptn
restoreDesign . $ptn
create_ps_per_micron_model
set_proto_mode -identify_min_inst ${minInst}
identify_proto_model
create_proto_model -out_dir fm.enc

cd ../..
exit
  
```

## Sample Script for Assembling All Models

```

assemble_proto_model -topdir {PTN_dir/DTMF_CHIP/fm.enc.dat
PTN_dir/DTMF_CHIP/proto_model} \
-blockdir { PTN_dir/tdsp_core/fm.enc.dat PTN_dir/tdsp_core/proto_model} \
-blockdir { PTN_dir/arb } \
-mmmmc_file viewDefinition.tcl
  
```

```
saveDesign DBS/model_gen.enc
report_proto_model -created

timeDesign -preplace -proto
# Continue with normal flow
...
```

# Analysis Capabilities

---

- RC Extraction
- Calculating Delay
- Timing Analysis
- Debugging Timing Results
- Power and Rail Analysis
- Power Analysis and Reports
- Analyzing and Repairing Crosstalk

# RC Extraction

- Overview
- Before You Begin
  - Results
  - Specifying Temporary File Locations
- Extraction Flow in Innovus
- PreRoute Extraction
- PostRoute Extraction
  - Native Detailed
  - tQuantus
  - tQuantus versus IQRC
  - Sign-Off Extraction Using Quantus QRC
  - Inputs for Quantus QRC Sign-Off Extraction
- Scale Factor Setting
- Generating a Capacitance Table
  - Inputs for Generating a Capacitance Table
  - Capacitance Table Generation Flow
  - Generating Capacitance Table with Specified Scale Factors
- Reading a Capacitance Table
- Reading a Quantus Techfile
- PreRoute Extraction Flow without Capacitance Table Data
  - For designs above 32nm
- Correlating Native Extraction With Sign-Off Extraction
  - Correlating SPEF Files Using the Ostrich Utility
  - Defining the Scale Factor
- Distributed Processing
  - Setting-up Distributed Processing
  - Generating a Capacitance Table in Multi-CPU Mode
- Using Advanced Virtual Metal Fill
  - Setting-up the Advanced VMF Rules

## Overview

You can perform two types of extraction in Innovus™ Implementation System (Innovus):

- PreRoute Extraction

Provides quick parasitic extraction for design prototyping. For more information, see [PreRoute Extraction](#).

- PostRoute Extraction

Generates more accurate parasitics for cross-coupling and signal integrity analysis, timing and SI optimization flow, or obtain sign-off quality detailed parasitic extraction. The postRoute extraction engine has four variants that allow selection based on the performance versus accuracy needs.

Following is a list of extraction engines in increasing order of accuracy:

- Native Detailed
- tQuantus
- Integrated QRC (IQRC)
- Standalone Quantus QRC

For more information, see [PostRoute Extraction](#).

The following table summarizes the types of extraction used during the design process.

Extraction Type	When	Quantus QRC License Required
PreRoute	Used during optimization both before and after clock tree synthesis.	✗
PostRoute	Native Detailed	Used during postRoute and SI optimization flow in older technologies.
	tQuantus	Used during postRoute optimization flow in newer technologies.
	IQRC	Used after ECO and for near signoff. <b>Note:</b> For IQRC, Quantus QRC-XL license is required
	Standalone Quantus QRC	Used during chip assembly and timing sign-off processes.

## Before You Begin

Before running extraction, you optionally enter the RC scale factor values in the [Edit RC Corner](#) form for the MMMC design environment. Scale factor values provide better correlation between the Innovus estimated parasitics and the signoff extraction results by multiplying the extracted resistance and capacitance. For example, a capacitance scale factor of 1.1 increases the extracted values by ten percent.

Use of scale factors is recommended for preRoute and native detailed extraction engines, and is optional for tQuantus. The IQRC scale factor also allows for optional fine tuning for optimal correlation with third party signoff extractor. In addition, all scale factors allow you to use simple scaling for non-typical corners as an alternative for the recommended use of corner-specific capacitance tables (captables) and Quantus technology files. Scaling for engine with `effortLevel` set to `signoff` is not supported.

The following files are required for Innovus extraction:

- Capacitance table - Used by preRoute extraction engine and native detailed extraction engines for above 32nm technology, and for 32nm and below technology only when Quantus technology files are not specified. For more information, see the section titled, "Generating a Capacitance Table".
- Quantus technology file(s) - Used by preRoute extraction engine for 32nm and below technology, tQuantus/IQRC, and Standalone Quantus QRC engines.

**Note:** One captable and a matching `qrcTechFile` is required per process corner.

## Results

- Binary RC Database (RCDB) is created. It contains parasitics for each process corner.
- An ASCII SPEF file can be generated from the parasitics database for the specified process corner, if required.

## Specifying Temporary File Locations

You can specify a temporary file location for tQuantus and IQRC extraction. The temporary file location is chosen based on the following order of precedence:

1. If you specify a directory using the `FE_TMPDIR` environment variable, the software uses that directory as the temporary file location.
2. If you specify a directory using the `TMPDIR` environment variable, the software uses that directory

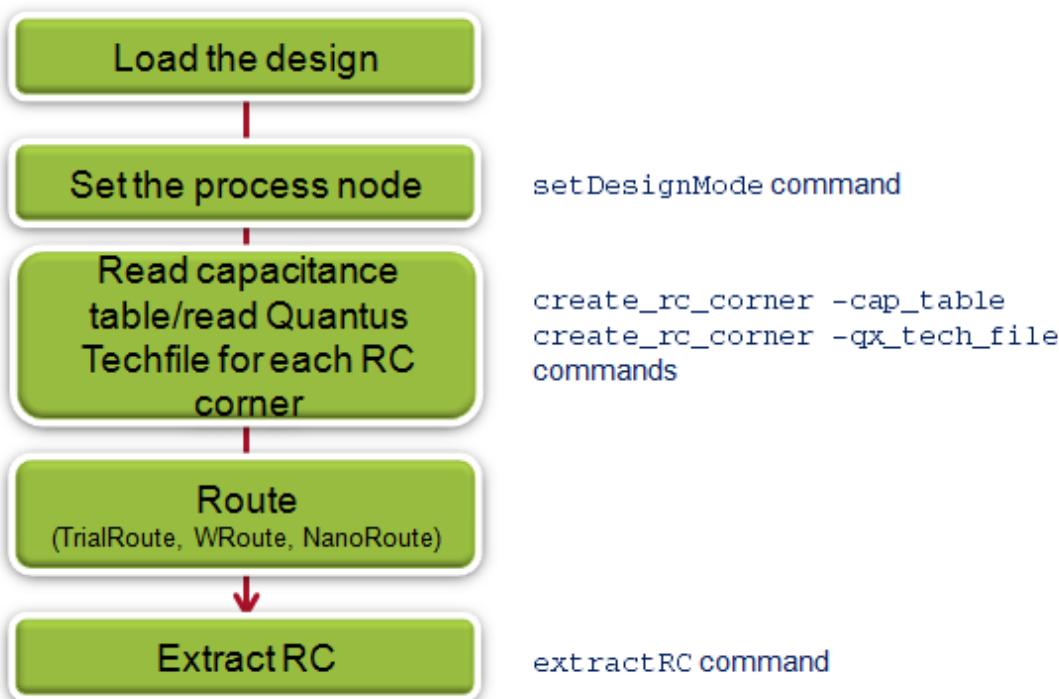
as the temporary file location.

3. Saves the files to the current directory (if writable).
4. Saves the files to the /tmp directory.

**Note:** For IQRC/tQuantus extraction in the distributed processing mode using different machines, do not store the cache or temporary data to the /tmp directory. The temporary data must be visible on all machines used for distributed processing. Either use the current directory, or specify a directory using the TMPDIR or FE\_TMPDIR environment variable.

## Extraction Flow in Innovus

The following figure shows the extraction flow.



To perform extraction, perform the following steps:

1. Load the design.
2. Specify the process technology value, to automatically set the technology node dependent parameters, by using the following syntax:  
`setDesignMode -process processnode`

**Note:** For maximum accuracy and optimal automatic threshold setting, use the `-process processnode` parameter of the `setDesignMode` command prior to running the `extractRC` command.

3. Read in the capturable file(s) for extracting interconnect capacitance values with `preRoute` and `postRoute -effortLevel low` engine choices for above 32nm technology, and for 32nm and below technology only when Quantus Techfiles are not specified. For more information, see the section titled, "Correlating Native Extraction With Sign-Off Extraction".

Optionally, when using `preRoute` extraction (for 32nm and below), tQuantus, IQRC, or Standalone Quantus QRC extraction, read in the Quantus Techfile that contains the interconnect models used by these engine choices. For more information, see "Reading a Quantus Techfile".

**Note:** If the design is half node and the layout scale value is not specified in the capturable, you can specify it using the `setShrinkFactor` command.

4. Use the `setExtractRCMode` command to set up extraction parameters and to specify the extraction engine to be used for subsequent extraction.
5. Use the `extractRC` command to perform extraction. You can also use the Specify RC Extraction Mode and Extract RC GUI forms to perform extraction. An RCDB is created. This database contains extracted parasitics.
6. Optionally, use the `rcOut` command to retrieve a SPEF files (corresponding to each active RC corner) with the parasitics results. Timing and SI analysis commands use the RC database directly.
7. Use the `spefIn` command to load the existing SPEF files with the parasitic data.
8. For hierarchical designs, use the `read_parasitics` command for reading both the top-level and block-level parasitic data in either the RCDB format or the SPEF format, or a mix of both. This command generates a flat-level RCDB for the complete design after performing hierarchical stitching of the RC data.

## PreRoute Extraction

In preRoute extraction mode, the total capacitance for each net is calculated based on the net geometry and the local wire density. In this mode, the software does not calculate separate coupling capacitance. This mode uses the signal wire geometries provided by TrialRoute, and the clock net geometries provided by Clock Tree Synthesis (CTS).

**Note:** The preRoute mode is only used for static timing analysis (STA).

## PostRoute Extraction

You can perform postRoute extraction using the following postRoute engine variants:

### Native Detailed

In native detailed mode:

- RC values that are generated can be used for both STA (including cross-coupling) and SI analysis to provide more accurate results for a particular process technology.
- The software calculates the coupling capacitance component for each segment by considering the actual geometries of neighboring nets on the same metal layer and on the adjacent metal layer when a full capturable is provided during design import.

To invoke native detailed extraction, use the following command:

```
setExtractRCMode -engine postRoute -effortLevel low
```

Note: Usage of native detailed extraction is not recommended for 32nm and below nodes. tQuantus and IQRC should be used instead.

### tQuantus

The tQuantus extraction engine is an advanced extraction engine that is enabled by default for postRoute -effortLevel medium extraction.

```
setExtractRCMode -engine postRoute -effortLevel medium
```

The tQuantus engine is an improvement on the TQRC extraction engine and as compared to TQRC, it is much faster, consumes less memory, is deterministic, and provides results that are equivalent to signoff QoR. In addition, it is tightly integrated with NanoRoute and drives track assignment-based timing and SI optimization/postRoute optimization, and timing-driven routing. The use model is detailed

below.

The tQuantus flow is enabled by default (`setExtractRCMode -tQuantusForPostRoute true`). In this mode, the software instructs `optDesign / timeDesign -postRoute` to use the tQuantusModel file for optimization. When this parameter is set to `false`, the software will use the TQRC model. You can specify a file name for the tQuantus model file by using the `-tQuantusModelFile` parameter of the `setExtractRCMode` command. If this file is not specified, it is generated automatically by the software.

TQRC will be obsoleted in a subsequent release but for now it can still be used, if desired, by setting the following commands:

```
setExtractRCMode -tQuantusForPostRoute false
setExtractRCMode -engine postRoute -effortLevel medium
```

For a complete behavior of `timeDesign` and `optDesign -postRoute` commands with respect to tQuantus, see the table below.

### tQuantus Usage

Setting1	Setting2	Outcome
<code>setExtractRCMode -tQuantusForPostRoute true</code>	<code>setExtractRCMode -effortLevel medium</code>	Use tQuantus for <code>timeDesign / optDesign -postRoute</code> and <code>extractRC</code> . If the <code>setExtractRCMode -tQuantusModelFile</code> is not specified, it will be generated by the software.
<code>setExtractRCMode -tQuantusForPostRoute true</code>	<code>setExtractRCMode -effortLevel other</code>	Use "other" extraction for <code>timeDesign/optDesign -postRoute</code> and <code>extractRC</code> .
<code>setExtractRCMode -tQuantusForPostRoute false</code>	<code>setExtractRCMode -effortLevel medium</code>	Issue a warning stating that TQRC is no longer the default engine and will be obsoleted in a future release. However, the software will proceed with TQRC for <code>timeDesign/optDesign -postRoute</code> .
<code>setExtractRCMode -tQuantusForPostRoute false</code>	<code>setExtractRCMode -effortLevel other</code>	Use "other" extraction for <code>timeDesign/optDesign -postRoute</code> and <code>extractRC</code> .

The tQuantus extraction engine supports the following `setExtractRCMode` parameters:

- capFilterMode relAndCoup

**Note:** It does not support the `relOnly` and `relOrCoup` parameters of the `-capFilterMode` parameter as they are for old nodes only.

- relative\_c\_th
- total\_c\_th
- coupling\_c\_th

-extraCmdFile  
-coupled

It does not support the following parameters of the `setExtractRCMode` command:

-hardBlockObs  
-lefTechFileMap

## tQuantus versus IQRC

tQuantus and IQRC are more accurate as compared to native detailed extraction. These are based on the same technology as the Standalone Cadence signoff extraction tool, Quantus QRC. The tQuantus extraction engine is recommended for the implementation phase because it is optimized for performance with a small tradeoff for accuracy. The IQRC extraction engine is recommended for the ECO flow, as it has near-signoff accuracy.

In IQRC, there are two modes for RC extraction:

- **Full-chip extraction**

Forces full extraction on the complete design.

- **Incremental extraction**

Enables incremental extraction on the design. The software recognizes the changes that have taken place since the last extraction. If the changes are less than the pre-defined threshold, the software incrementally extracts the changed regions of the design and then stitches the new data with the previously extracted parasitic data.

By default, the incremental mode is enabled. In tQuantus there is no incremental extraction. This is not needed due to the high speed of this extraction engine.

**Note:** The performance and accuracy of tQuantus falls between that of native detailed extraction and IQRC. IQRC provides superior accuracy compared to tQuantus. Both IQRC and tQuantus support distributed processing. tQuantus does not require a Quantus QRC license while IQRC requires a Quantus QRC-XL license.

To invoke IQRC, use the following command:

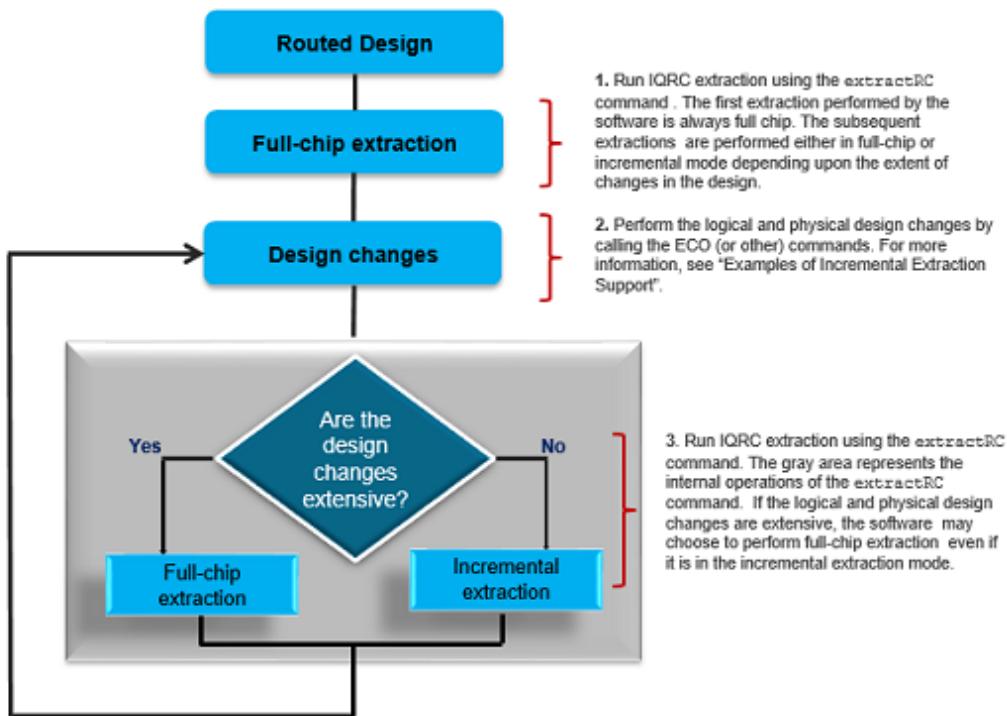
```
setExtractRCMode -engine postRoute -effortLevel high
```

To invoke tQuantus, use the following command:

```
setExtractRCMode -engine postRoute -effortLevel medium
```

The following figure shows the IQRC extraction flow.

## IQRC Extraction Flow



## Examples of Incremental Extraction Support

Following are some of the examples for incremental extraction flow in Innovus:

- **Example of Interactive ECO Commands:**

```

setExtractRCMode -engine postRoute -effortLevel high
extractRC
ecoAddRepeater -net netName -cell cellName
...
ecoPlace
ecoRoute
extractRC
  
```

- **Example of Wire Edit Commands:**

```

setExtractRCMode -engine postRoute -effortLevel high
extractRC
  
```

```
editSelect -area areaValue -net netName
editMove y distance
...
extractRC
```

- **Example of Timing Optimization Command:** In the postRoute optimization cycle, extraction is called multiple times. Depending upon the type of changes, the extraction called may be either full-chip or incremental.

```
setExtractRCMode -engine postRoute -effortLevel high -incremental true
optDesign -postRoute
```

**Note:** Incremental extraction is not supported for all designs or for all types of changes in the design. For example, incremental extraction will not take place if the design contains 45-degree wire(s). In such cases, the software automatically goes into the full-chip extraction mode after printing an appropriate message citing the reason for selecting full-chip extraction.

## Sign-Off Extraction Using Quantus QRC

Standalone Quantus QRC extraction is accessible through Innovus for generating detailed signoff quality parasitics. You can perform signoff extraction after the detailed routing phase.

**Note:** Standalone Quantus QRC extraction requires a separate license and requires installation of the EXT software package.

You can run Quantus extraction by using the `extractRC` command after setting the Quantus extraction mode using the `-effortLevel signoff` option of the `setExtractRCMode` command.

By default, Standalone Quantus QRC extraction is a Design Exchange Format (DEF)-based flow. However, you can specify the OpenAccess (OA)-based flow for invoking Quantus extraction. For this, set the `setExtractRCMode -useQRCAInterface` option to `true`.

**Note:** For TSV designs, you are required to provide the name of the layer map file for IQRC/tQuantus and Standalone Quantus QRC.

## Inputs for Quantus QRC Sign-Off Extraction

When you perform signoff extraction through the Innovus interface, either a routed DEF is created or an OA database is saved automatically. The Quantus technology file is required before you can start signoff extraction:

- **Quantus technology file:** Contains the process-dependent model files and manufacturing effects used by the extractor to calculate resistance and capacitance.
- **Quantus command file:** Providing a Quantus command file is optional. It contains commands and variables that define the extraction environment (technology filename, library name, and so on), specifies which net(s) to extract and how to extract them, controls the resistance and capacitance extraction, and specifies the extraction outputs. This is applicable only for partial and custom command files.

## Scale Factor Setting

To better correlate native extraction results, both in preRoute and postRoute stages with sign-off extraction, Innovus allows you to set scale factors for total capacitance, cross-coupling capacitance, and resistance. As the accuracy of the different engines varies, engine-dependent scale factors can be entered when using different extraction engines in the flow. For more information, see the section titled, [Correlating Native Extraction With Sign-Off Extraction](#).

To generate scale factors, you can also use the `generateRCFactor` command.

## Generating a Capacitance Table

A capacitance table is needed for extraction by the `preRoute` and `postRoute -effortLevel low` extraction engines for above 32nm technology, and for 32nm and below technology only when Quantus technology files are not specified. The table contains three parts:

- **Header:** Contains process information and manufacturing effects. The information in the header is used for resistance extraction and to correct the extracted values for the specified manufacturing effects. For preRoute extraction, the header section also provides default values for scale factors.
- **Basic Captable Part:** Contains the coefficients used by the preRoute capacitance extraction engine. This part contains the area, fringe, and lateral coupling capacitance coefficients organized per conductor layer for wires with different width and spacing. The basic capturable part is presented in a readable tabular format.
- **Extended Captable Part:** Contains the coefficients used by the preRoute capacitance extraction engine and the `postRoute -effortLevel low` capacitance extraction engine. This part is much larger compared to the basic capturable part because the coefficients are generated on more

complex profiles, which account for geometries on multiple layers. The extended capturable part is an ASCII dump of the binary stored data.

Each technology requires one capacitance table. To consider process corner variations, you must generate a capacitance table for each process corner.

-  The capacitance table must be generated before running extraction.

If the capacitance table is not defined before extraction, Innovus generates a basic capacitance table using default process parameters and using heuristic equations for calculation. It is strongly recommended to provide a capturable for the technology used in the design to ensure maximum accuracy. Even if tQuantus or IQRC are used as postRoute extraction engines, the capturable is still needed for preRoute extraction and postRoute extraction engines for above 32nm technology, and for 32nm and below technology only when Quantus technology files are not specified.

## Inputs for Generating a Capacitance Table

To generate a capacitance table, you need an ICT file and a technology LEF file (optional). The technology LEF file provides the capacitance generated with design specific widths and spacings used in non-default routing rules. In addition, it provides information on the actual spacing between regular wires as defined by the `PITCH` statement. The use of a LEF file increases the simulation points and consequently reduces the need of the extractor to use interpolation. The LEF file is used for the extended capturable part only.

Fabrication process information in the ICT file can consist of the following:

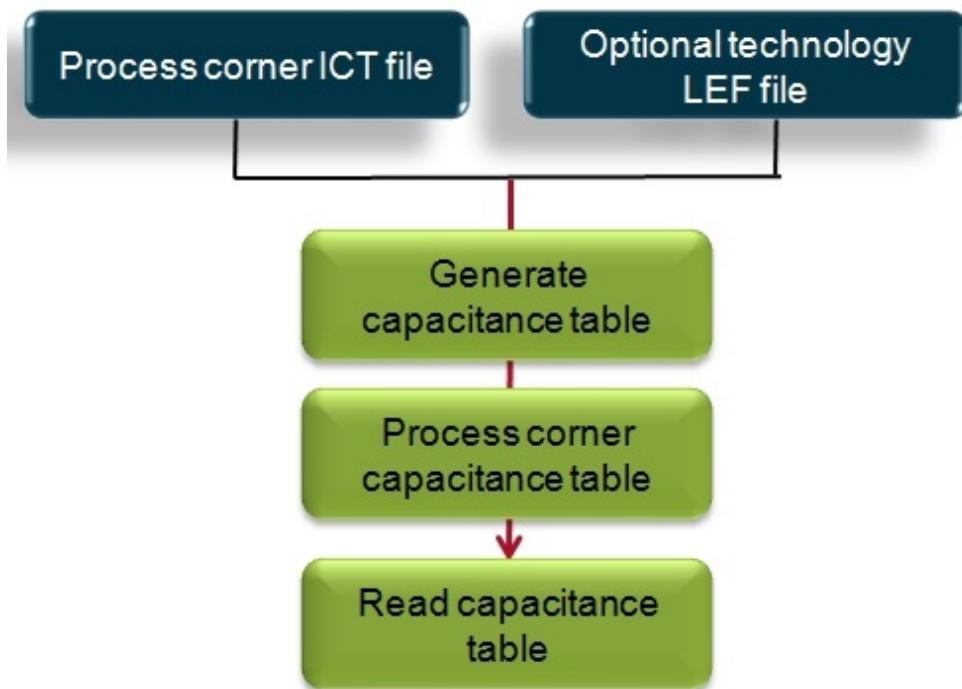
- The minimum spacing and minimum width of the conductors as specified in the design rules for the conductor layers.
- The thicknesses of the conductor layers.
- The heights of the conductor layers above the substrate (measuring height from the field) or as a delta from a previously-defined lower-level conductor layer.
- The resistivities of the conductor layers: The ICT file can contain a constant sheet resistance value, a width-dependent sheet resistance vector, or a resistivity (`rho`) table.
- The interlayer planar dielectric constant, its height above the substrate (measuring height above the field), and its thickness.
- The names of the top conductor layer of a via, the bottom conductor layer of the via, and the

contact resistance of the via with their associated cut resistance.

For more information on the syntax of the ICT file, see the chapter titled, [Creating the ICT File](#).

## Capacitance Table Generation Flow

The following figure shows the flow for generating a process corner-specific capacitance table:



**Note:** You can also use the scale factor to convert a typical corner capacitance table to a different corner capacitance table. For more information, see the section titled, [Generating Capacitance Table with Specified Scale Factors](#).

To generate a capacitance table, perform the following steps:

1. Generate an ICT file for each process corner. For an example of an ICT file, see the chapter titled, [Creating the ICT File](#).
2. Generate a capacitance table for each ICT file. Use the `generateCapTbl` command within Innovus or the `generateCapTbl` standalone executable.

**Note:** Generating a capacitance table is CPU-intensive and can take several hours to run for newer technologies. The `generateCapTbl` standalone executable which can be found in the `bin` directory of your Innovus hierarchy runs independent of Innovus. It has the same syntax as the `generateCapTbl` command.

## Capacitance Table Examples

### Example 1: Capacitance Table

```
PROCESS_VARIATION ...

LAYER M1

MinWidth 0.09000
MinSpace 0.09000
# Height 0.54000
Thickness 0.20260
TopWidth 0.12100
BottomWidth 0.09300
WidthDev 0.00000
ThermalC1 2.65000e-03
ThermalC2 -2.64100e-07
WireEdgeEnlargement
WeeWidths 0.107 0.127 0.152 0.197 0.287 0.377 0.467 0.557 0.647 0.917 1.017 2.017 3.017
4.517 7.517 12.017
WeeSpacings 0.073 0.093 0.118 0.163 0.253 0.343 0.433 0.523 0.613 0.883 0.983 1.483 1.983
2.483 2.983 4.983
WeeAdjustments -0.001 -0.003 -0.003 -0.009 -0.009 -0.009 -0.01 -0.01 -0.01 -0.011 -0.016 -0.018 -0.018
-0.019 -0.019 -0.019 -0.019
0.003 -0.002 -0.003 -0.009 -0.009 -0.01 -0.01 -0.011 -0.016 -0.018 -0.018 -0.019 -0.019 -
0.019 -0.019 -0.019
0.008 0.003 0 -0.009 -0.009 -0.01 -0.01 -0.011 -0.016 -0.018 -0.018 -0.019 -0.019 -0.019 -
0.019 -0.019
...
0.027 0.019 0.011 -0.009 -0.009 -0.01 -0.01 -0.011 -0.016 -0.018 -0.018 -0.019 -0.019 -
0.019 -0.019 -0.019

Rho
RhoWidths 0.09 0.11 0.135 0.18 0.27 0.36 0.45 0.54 0.63 0.9 1 2 3 4.5 7.5 12
RhoSpacings 0.09 0.11 0.135 0.18 0.27 0.36 0.45 0.54 0.63 0.9 1 1.5 2 2.5 3 5
```

```
RhoValues 0.0301 0.0288 0.0272 0.0257 0.0236 0.0225 0.0218 0.0216 0.0216 0.0216  
0.0216 0.0216 0.0215 0.0215  
0.0294 0.0286 0.0272 0.0257 0.0235 0.0224 0.0218 0.0216 0.0216 0.0216 0.0216  
0.0216 0.0215 0.0215  
0.0286 0.0279 0.0269 0.0257 0.0235 0.0224 0.0218 0.0216 0.0215 0.0216 0.0216  
0.0215 0.0215 0.0215  
...  
0.0264 0.0262 0.0259 0.0257 0.0235 0.0224 0.0218 0.0216 0.0215 0.0215 0.0215  
0.0214 0.0213 0.0213
```

WireThicknessRatio

WtrMinThicknessRatio 0.914688

WtrMaxThicknessRatio 1.03875

WtrTileWidth 100 100

WtrStepperWindowWidth 50 50

WtrMaxSpacing 5

WtrWidthRanges 0.09 0.18 12

WtrDensityPolynomialOrder 4

WtrWidthPolynomialOrder 4

WtrPolynomialCoefficients

{

0 7180.07 -3744.08 625.29 -33.3498

0 -8418.26 4361.73 -720.647 37.5707

0 3011.8 -1560.96 257.063 -13.1704

0 -369.306 193.11 -31.9649 1.58144

0 -2.73935 -0.912962 0.476445 0.0054143

}

{

-0.00355249 0.0134349 -0.377017 2.18449 -1.14899

0.0086884 0.0669357 0.00360917 -3.62062 1.91715

```
-0.0108639 -0.126014 0.84772 1.42329 -0.92238
0.00784181 0.0469798 -0.580639 0.11264 0.0642999
-0.00204508 -0.00330229 0.125923 -0.179971 0.100731
}

END

LAYER M2
...
...
...
VIA VIA1
TopLayer M2
BottomLayer M1
ThermalC1 7.81500e-04
ThermalC2 -2.57400e-06
Resistance 3.00000
END
...
END_PROCESS_VARIATION
BASIC_CAP_TABLE ...
M1
width(um) space(um) Ctot(Ff/um) Cc(Ff/um) Carea(Ff/um) Cfrg(Ff/um)
0.090 0.072 0.3549 0.1313 0.0502 0.0123
0.090 0.090 0.3115 0.1248 0.0502 0.0147
0.090 0.270 0.1803 0.0237 0.0502 0.0418
0.090 0.450 0.1728 0.0074 0.0502 0.0541
0.090 0.630 0.1721 0.0023 0.0502 0.0587
0.090 0.810 0.1720 0.0007 0.0502 0.0602
0.090 0.990 0.1720 0.0002 0.0502 0.0607
0.090 1.170 0.1720 0.0001 0.0502 0.0608
```

```
0.270 0.072 0.3797 0.1114 0.1267 0.0151
...
9.000 0.990 4.2967 0.0002 4.2226 0.0369
9.000 1.170 4.2967 0.0001 4.2226 0.0370
M2
width(um) space(um) Ctot(Ff/um) Cc(Ff/um) Carea(Ff/um) Cfrg(Ff/um)
0.100 0.080 0.3330 0.1275 0.0458 0.0161
0.100 0.100 0.2659 0.0919 0.0458 0.0182
....
END_BASIC_CAP_TABLE
EXTENDED_CAP_TABLE ...
# SolverExe: coyote
# Solver Type: coyote
1.02 8 8 t 1
0.5385 0.2046 3.9 0 0
0.017 0 3 2 1
3 0
....
END_EXTENDED_CAP_TABLE
```

### **Example 2: Rho (Resistivity Table) Included in the Capacitance Table**

```
Rho
RhoWidths 0.14 0.28 10
RhoSpacings 0.14 0.28 1
RhoValues 0.0351 0.02 0.0176
0.0451 0.03 0.01
0.0702 0.04 0.02
```

### **Example 3: Wire Edge Enlargement - Resistance Included in the Capacitance Table**

```
WireEdgeEnlargementR
WeeWidths 0.12 0.16 0.24
```

```
WeeSpacings 0.108 0.17 0.24
```

```
WeeAdjustments 0 0 0.002
```

```
0.011 0.007 0.002
```

```
0.022 0.012 0.002
```

#### **Example 4: Wire Edge Enlargement - Capacitance Included in the Capacitance Table**

```
WireEdgeEnlargementC
```

```
WeeWidths 0.12 0.16 0.24
```

```
WeeSpacings 0.108 0.17 0.24
```

```
WeeAdjustments 0.01 0.01 0.02
```

```
0.01 0.07 0.02
```

```
0.02 0.02 0.02
```

## **Generating Capacitance Table with Specified Scale Factors**

You can specify scale factors to convert a corner-specific capacitance table into another capacitance table for a different process corner. Use the `generateCapTbl` command to input a capacitance table and to specify the scale factors. Use the following parameters of the `generateCapTbl` command:

- `-incaptable fileName`  
Specifies the name of an existing capacitance table in ASCII format.
- `-cap totalCapFactor`  
Specifies the capacitance scale factor.
- `-xcap crossCouplingFactor`  
Specifies the cross-coupling capacitance scale factor.
- `-res resistanceFactor`  
Specifies the resistance scale factor.

## Reading a Capacitance Table

To consider process corner variations in MMMC design setup, you must read multiple capacitance tables. For each corner created by the `create_rc_corner` command, use the `-cap_table` parameter to specify the appropriate capturable file to be used for a specific corner. Information related to the specified corner can be modified using the `update_rc_corner` command. This information is saved in the `viewDefinition.tcl` file, which is read in the subsequent Innovus sessions during design import.

For more information, see the [Configuring the Setup for Multi-Mode Multi-Corner Analysis](#) section of the *Importing and Exporting Designs* chapter of the *Innovus User Guide*.

Examine the command log for any possible error messages. The number of metal layers specified in the ICT and the LEF file (if used) at the time of capacitance table generation must either match or be higher than the actual number of layers used in your design (current LEF/DEF).

The capacitance table contains standard names for metal layers (`M1, M2...`). It does not contain names used in the LEF file.

**Note:** You must read the capacitance table before specifying the extraction mode.

## Reading a Quantus Techfile

Quantis Techfiles are required by preRoute (32nm and below), tQuantus, IQRC, and signoff Quantus extraction engines. Use the `-qx_tech_file` parameter of the `create_rc_corner` command to read in the Quantus Techfile for each process corner.

```
create_rc_corner -name rcCornerName -qx_tech_file fileName
```

This information is saved in the `viewDefinition.tcl` file, which is read in the subsequent Innovus sessions during design import.

For more information, see the [Configuring the Setup for Multi-Mode Multi-Corner Analysis](#) section of the *Importing and Exporting Designs* chapter of the *Innovus User Guide*.

## PreRoute Extraction Flow without Capacitance Table Data

Quantis Techfiles are recommended for performing RC Extraction on designs at 32nm or below nodes. When Quantus Techfiles are provided, capturable files are ignored. In this scenario, preRoute extraction uses Quantus Techfiles (instead of capturable files) and postRoute extraction invokes "`-engine postRoute -effortLevel medium`" (i.e. tQuantus).

The use model is detailed below.

## Extraction Flow without Capacitance Table Data

	Design – 32nm and Below		Design – Above 32nm		
	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
PreRoute Extraction	Captable not needed	Captable Present	Captable Absent	Captable Present	Captable Present
	Quantus Techfile Present	Quantus Techfile Absent	Quantus Techfile Present	Quantus Techfile Absent	Quantus Techfile Present
	Flow without Captable	Captable-based Flow	Flow without Captable	Captable-based Flow	Captable-based Flow
PostRoute Extraction	Detailed extraction is not allowed.  tQuantus extraction engine is run.	Detailed extraction is run.	tQuantus engine is run.	Detailed extraction is run.	TQuantus extraction engine is run for 65nm and below – but detailed extraction is allowed by explicit setting.  For process nodes greater than 65nm, detailed extraction is run.

For designs at 32nm or below nodes

**Note:** The design mode is set in Innovus using the `setDesignMode -process` command.

- Captable files are not needed and Quantus Techfiles are specified - preRoute extraction uses Quantus Techfiles even when capturable files are provided. PostRoute extraction uses tQuantus extraction engine and detailed extraction is not allowed. This is shown as Scenario 1 in above table.
- Captable files are specified and the Quantus Techfiles are not specified - preRoute extraction uses capturable files. PostRoute extraction is called with `-effortLevel low` (detailed extraction) that uses capturable files. This is shown as Scenario 2 in above table.

## For designs above 32nm

- Captable files are not specified and Quantus Techfiles are specified - preRoute extraction uses Quantus Techfiles. PostRoute extraction uses tQuantus extraction engine. Detailed extraction is not allowed. This is shown as Scenario 3 in above table.
- Captable files are specified and the Quantus Techfiles are not specified - preRoute extraction uses the capturable files. PostRoute extraction is called with `-effortLevel low` (detailed extraction)

that uses capturable files. This is shown as Scenario 4 in above table.

- Capturable files and Quantus Techfiles are specified - preRoute extraction uses the capturable files. PostRoute extraction uses tQuantus extraction engine for designs at 65 nm and below nodes. Detailed extraction is run for designs above 65 nm. This is shown as Scenario 5 in above table.

**Note:** When neither capturables nor Quantus Techfiles are specified, the software gives an error.

## Correlating Native Extraction With Sign-Off Extraction

The software accommodates an extraction flow that uses process-dependent scale factors to generate extraction values that are close to the signoff extraction values. With these scale factors, the results generated by the native extraction correlate to the results of signoff extraction. The run time for the native extraction flow is much less than that for signoff extraction.

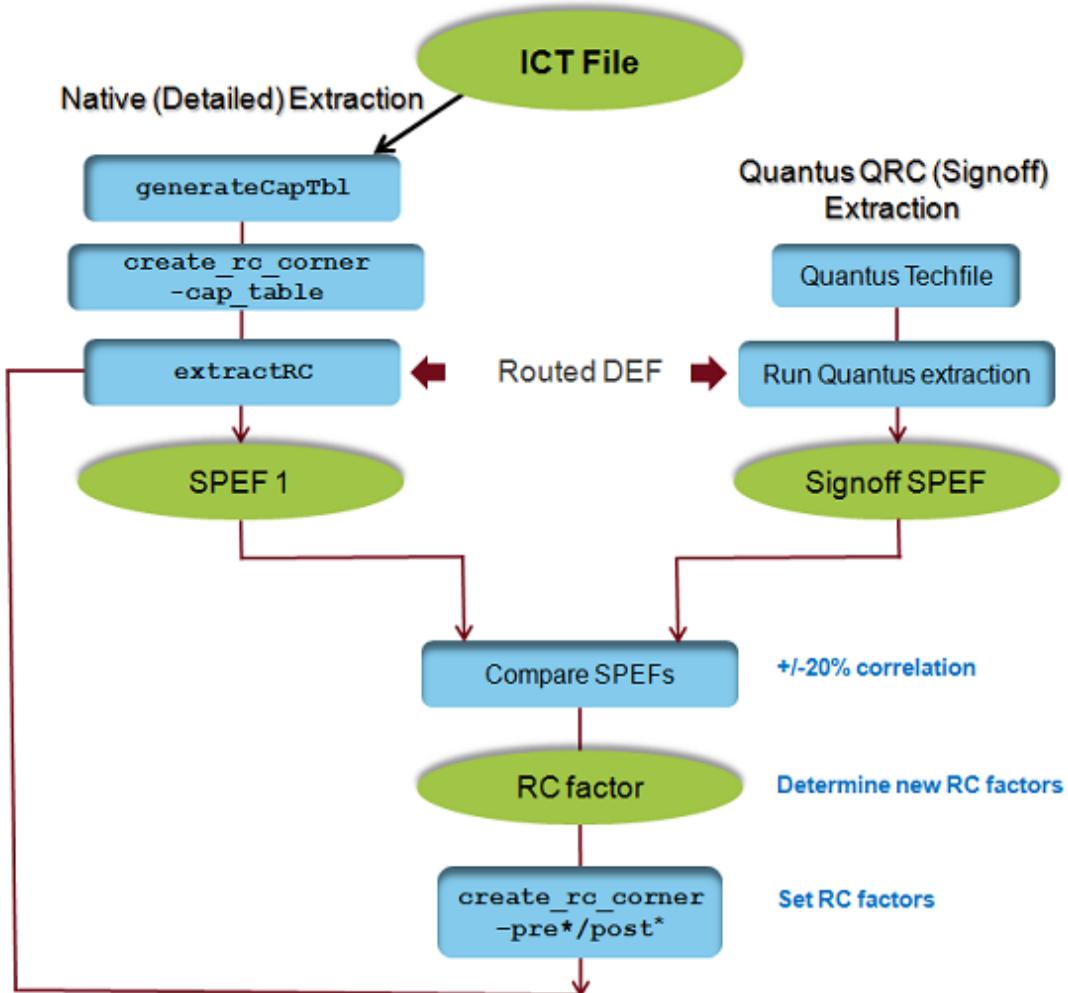
To generate RC scale factors, use the [generateRCFactor](#) command. Alternatively, complete the following steps to generate them to correlate native extraction results with sign-off extraction:

1. From the routed DEF, generate a SPEF file using the Sign-Off Extraction engine. For more information on generating a SPEF file, see [Sign-Off Extraction Using Quantus QRC](#).
2. Specify the process technology value to automatically set the technology node dependent parameters, by using the following syntax:  
`setDesignMode -process processnode`
3. Read in the capacitance table file(s) for extracting interconnect capacitance values with preRoute and postRoute -effortLevel low engine choices.  
Optionally, when using preRoute (32nm and below), tQuantus or IQRC, read in the Quantus Techfile that contains the interconnect models used by these engine choices. For more information, see "Reading a Quantus Techfile".
4. Generate native extraction SPEF file(s) using the [extractRC](#) and [rcOut](#) commands.

For preRoute mode, extraction should be run in the preRoute design stage and not on the final routed design. This way the scale factors to improve correlation will also take into account the difference between trial routes and final routes. For postRoute mode, extraction should be run on the same routed design that was used for the signoff extraction SPEF file generation.

5. Compare the SPEF file from native extraction with the SPEF file from signoff extraction using the Ostrich parasitics correlation utility. Use the correlation utility to generate RC factors (scale factors) for total capacitance, cross-coupling capacitance, and resistance. For more information, see [Correlating SPEF Files Using the Ostrich Utility](#).
6. Specify these scale factors using the `-pre*` and `-post*` parameters of the `create_rc_corner` command before future runs of native extraction. For more information, see the section titled, Defining the Scale Factor.
7. Rerun the `extractRC` command to generate a new SPEF file. This file contains capacitance and resistance values that correlate to the values in the Quantus signoff SPEF file.

The following figure shows the flow for generating RC scale factors.



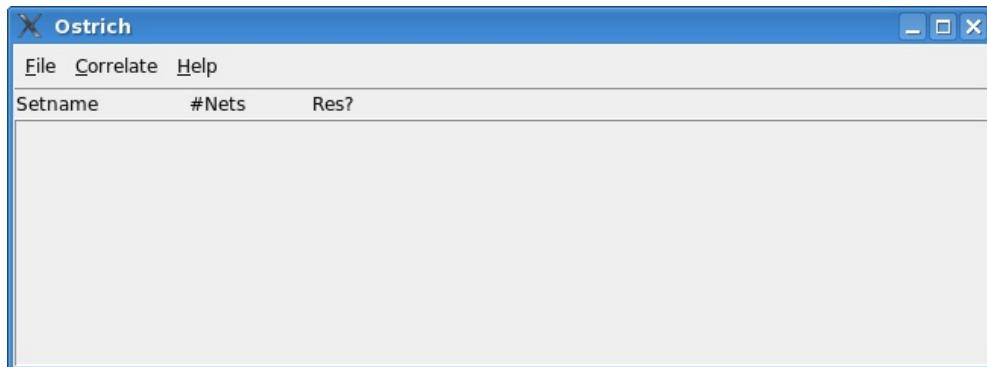
## Correlating SPEF Files Using the Ostrich Utility

Use Ostrich to correlate the SPEF files generated using native (`postRoute -effortLevel low`) extraction and signoff extraction. Ostrich is a standalone utility in Innovus. Ostrich generates the scale factors after correlating the SPEF files. You can then set the scale factors for the next extraction cycle.

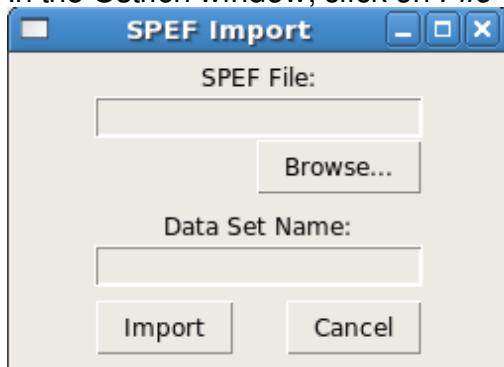
To correlate the SPEF files, complete the following steps:

1. Type `ostrich` at the Innovus prompt. This opens the main Ostrich window.

**Note:** As an alternative for the GUI, you can also use the command line mode by adding `-nowin` as an argument when starting the tool.

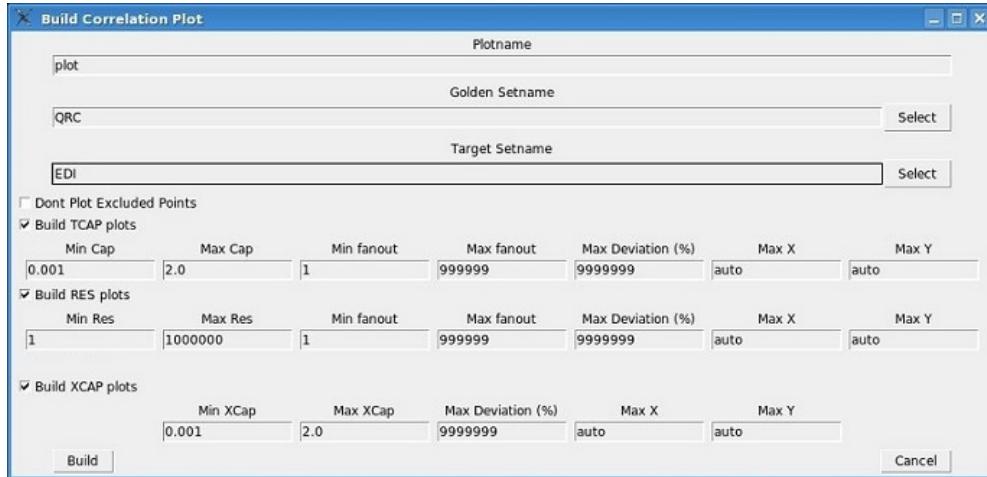


2. In the Ostrich window, click on *File - Import - SPEF*. This opens the SPEF Import form.

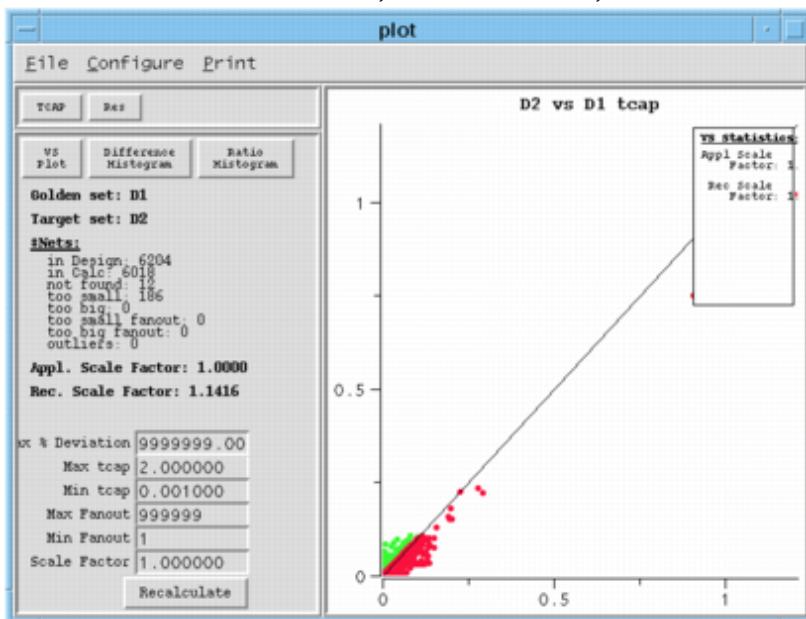


3. In the SPEF Import form, specify the name of the sign-off SPEF file and a name in the *Data Set Name* field. Next, click on the *Import* button to add the SPEF values in the Ostrich window.
4. Similarly, import the native extraction SPEF file using the SPEF Import form.

- Click on *Correlate - Build Plot* option in the Ostrich Window. This opens the Build Correlation Plot window.



- Select the *Golden Setname* and *Target Setname* corresponding to the sign-off SPEF, and native extraction SPEF respectively.
- Select *Build TCAP plots*, *Build RES plots*, and *Build XCAP plots* options. Click *Build*.
- In the Ostrich main window, click *Correlate*, and then *Draw Plot*. This opens the plot window.



The plot window displays the suggested scale factor.

## Report File Example

#Ref. Cap file:	lambda_aftercrosstalkfix.spf			
#Cmp. Cap file:	lambda_detailrc.spf			
#-----#				
# Total Capacitance Statistics #				
#-----#				
#Total Capacitance range considered:	0.0100 - 5.0000 [pF]			
#Total number of nets:	418399			
#Total number of nets with nonzero lumped Cap:	213095			
#Total number of nets discarded (small lumped Cap):	152272.00			
<b>#Suggested Capacitance Scale Factor</b>				
0.9577 <---- (Ref. = FE(Cmp.)* 0.9577)				
#Mean (Cap Scalar):	1.04			
#Total Sum of residual square:	22.89			
#Variance:	0.00			
#Standard deviation:	0.02			
#Coefficient of variation:	1.90%			
#Normal distribution range (sigma):	1.00 to 1.08			
# -----				
# Correlation results:	#of Nets	%		
# -----				
# Nets [..., 0.1]:	0	0.00		
# Nets [0.1, 0.2]:	0	0.00		

# Nets [0.2, 0.3]:	0	0.00	
# Nets [0.3, 0.4]:	0	0.00	
# Nets [0.4, 0.5]:	0	0.00	
# Nets [0.5, 0.6]:	0	0.00	
# Nets [0.6, 0.7]:	0	0.00	
# Nets [0.7, 0.8]:	0	0.00	
# Nets [0.8, 0.9]:	800	0.38	
# Nets [0.9, 1.0]:	20004	9.39	
# Nets [1.0, 1.1]:	24729	11.60	
# Nets [1.1, 1.2]:	10048	4.72	
# Nets [1.2, 1.3]:	3224	1.51	
# Nets [1.3, 1.4]:	1164	0.55	
# Nets [1.4, 1.5]:	454	0.21	
# Nets [1.5, 1.6]:	227	0.11	
# Nets [1.6, 1.7]:	90	0.04	
# Nets [1.7, 1.8]:	54	0.03	
# Nets [1.8, 1.9]:	11	0.01	
# Nets [1.9, 2.0]:	10	0.00	
# Nets [2.0, ...]:	8	0.00	
# Nets discarded :	152272	71.46	

## Defining the Scale Factor

You can specify the scale factors in the following ways:

- Change the technology file.

You can change the `ScaleFactor` in the technology file. This scaling is used for each technology.

Use the `-pre*` and `-post*` parameters of the `create_rc_corner` command.

You can set scale factors for the resistors and capacitors that are extracted in either preRoute or postRoute extraction mode. You can set different postRoute scale factors for the postRoute engine variants by specifying them as duplets and triplets. For example,

```
{value1 value2 value3}
```

Where:

- Single value: If you specify one value, the scale factor applies to effort level low. Scale factor value of 1 is used for medium and high by default.
- Duplet: If you specify two values, the first value is used for effort level low and the second value is used for medium. Scale factor value of 1 is used for high by default.
- Triplet: If you specify three values, the first value is used for effort level low, the second value is used for medium, and the third value is used for high.

## Example

```
create_rc_corner -postRoute_xcap {1.1 1.05} -postRoute_cap 1.2 -postRoute_res 1.1 -  
preRoute_cap 1.3 -preRoute_res 1.4 -preRoute_clkcap 1.11
```

**Note:** When saving the design with the `saveDesign` command, the scale factors that are specified with the `create_rc_corner` command are saved in the `viewDefinition.tcl` file, which can be later restored.

Note: The default value for all scale factors, except clock net scale factors, is 1.0. The default value for all clock net scale factors is a symbolic value 0. This indicates that the value of the specific clock net scale factor follows the matching signal net scale factor.

## Distributed Processing

Multiple CPUs can be used to improve the overall turn-around time of extraction. The run-time improvement may vary depending on multi-CPU configuration, design size and type. Generally, performance improvement will start to diminish beyond 8 CPUs.

## Setting-up Distributed Processing

The distributed processing is supported with two modes:

- Local Mode: In this mode, you can specify the number of CPUs to use on local machine.

```
setDistributeHost -local  
setMultiCpuUsage -localCpu 8
```

- Distributed Mode: In this mode, you can specify one or more CPUs to use on network hosts.

```
setDistributeHost -rsh -add {host1 host2 host3}  
setMultiCpuUsage -remoteHost 3
```

**Note:** RC Extraction ignores the `-cpuPerRemoteHost` parameter of the `setMultiCpuUsage` command. You must have rlogin access to remote host machines.

If you run a job in both local (`-localCpu`) and distributed mode (`-remoteHost`), the `-remoteHost` parameter takes precedence.

You can specify LSF and SGE queue or custom job submission script for multi-CPU mode.

```
setDistributeHost  
-lsf [-queue queue_name] [-resource resource_string] [-lsf_args lsf_arguments] |  
-sge [-queue queueName] |  
-custom [-custom_script script]
```

## Generating a Capacitance Table in Multi-CPU Mode

You can use the [Multiple-CPU Processing Commands](#) to generate a capacitance table in parallel mode when you use the `generateCapTbl` command within Innovus. This functionality is not available for standalone capacitance table generation.

## TCL Script to Run the `generateCapTbl` Command in the Distributed Mode

To run the `generateCapTbl` command in the parallel mode on different hosts, specify the following commands:

```
setDistributeHost -rsh -add { host1 host2 host3 }  
setMultiCpuUsage -remoteHost 3  
generateCapTbl -ict sample.ict -output sample.capTbl
```

## TCL Script to Run the generateCapTbl Command in the Local Mode

To run the `generateCapTbl` command with three CPUs on a local machine, specify the following commands:

```
setDistributeHost -local
setMultiCpuUsage -localCpu 3
generateCapTbl -ict sample.ict -output sample.capTbl
```

## Performing IQRC, tQuantus, and Standalone Quantus QRC Extraction in Multi-CPU Mode

IQRC, tQuantus, and Standalone Quantus QRC Extraction engines support distributed processing. You can use the [Multiple-CPU Processing Commands](#) to invoke IQRC, tQuantus, and Standalone Quantus QRC Extraction in the multi-CPU mode.

## TCL Script for IQRC, tQuantus, and Standalone Quantus QRC Extraction Invoked in the Distributed Mode

To run IQRC, tQuantus, and Standalone Quantus QRC Extraction in the parallel mode on different hosts, specify the following commands:

```
setDistributeHost -rsh -add { host1 host2 host3 }
setMultiCpuUsage -remoteHost 3
setExtractRCMode -engine postRoute -effortLevel [ medium | high | signoff ]
extractRC
```

## TCL Script for IQRC, tQuantus, and Standalone Quantus QRC Extraction Invoked in the Local Mode

To run IQRC, tQuantus, and Standalone Quantus QRC Extraction with three CPUs on a local machine, specify the following commands:

```
setDistributeHost -local
setMultiCpuUsage -localCpu 3
setExtractRCMode -engine postRoute -effortLevel [ medium | high | signoff ]
extractRC
```

## Using Advanced Virtual Metal Fill

Quantus, IQRC, and tQuantus support Advanced Virtual Metal Fill (VMF). Advanced VMF gives better metal-fill effect estimation as compared to the original default VMF. To use Advanced VMF with IQRC and tQuantus, an extra command file containing the Quantus QRC CCL commands is required. This file can be specified by using the `setExtractRCMode -extraCmdFile qrc.cmd` command. Include the set-up detailed below in the `qrc.cmd` file.

## Setting-up the Advanced VMF Rules

To use Advanced VMF, specify the external VMF rule files. These files will be used regardless of whether VMF rules are included in the `qrcTechFile` or not.

To use the external VMF rule files, use the combination of the following three commands inside the `qrc.cmd` file:

```
metal_fill -type virtual \
            -enable_advanced_virtual_fill true \
            -vmf_metal_scheme_file <metal_scheme_file_name> \
            -vmf_rule_file <param_file_name>
```

When specified, the above options overwrite the VMF specification in the ICT file. This feature lets you use the specified external VMF rule files at runtime. In this case, if the `qrcTechFile` does not contain the VMF rules, Quantus, IQRC, and tQuantus will use the external VMF rule files. If the `qrcTechfile` contains the VMF rules, the rules in the external VMF rule file will supersede the VMF rules specified in the `qrcTechFile`.

- `-enable_advanced_virtual_fill true`

Enables Advanced VMF feature. The default value is "false".

- `-vmf_metal_scheme_file metal_scheme_file_name`

Specifies the metal scheme file. This file defines the mapping between layer names and layer types. The format of the file is provided below:

Layer: metal layer name in ICT file

Type: layer type (1, x, y, z, ...)

Dir: routing direction

An example of a metal scheme file is provided below:

<b>Layer</b>	<b>type</b>	<b>Directory</b>
M2	M1	H
M2	Mx	V
M3	Mx	H
M4	Mx	V

- `-vmf_rule_file param_file_name`

Specifies the param file. This file defines the VMF rules, such as size, x/y fill-fill spacing, and net-fill spacing, per metal type defined in the scheme file. The format of the file is provided below:

Type: layer type defined in scheme file

L: VMF length

w: VMF width

nf\_sp: minimum net-fill spacing

ff\_sl: fill-fill spacing along with length

ff\_sw: fill-fill spacing along with width

An example of a param file is provided below.

<b>Type</b>	<b>L</b>	<b>W</b>	<b>nf_sp</b>	<b>ff_sl</b>	<b>ff_sw</b>
M1	1.00	0.12	0.60	0.12	0.12
Mx	1.00	0.12	0.60	0.12	0.12
My	1.20.	0.50	0.30	0.30	0.30
Mz	0.80	0.80	0.60	0.40	0.40

**Note:** For Quantus, XL+AA license is required when `-enable_advanced_virtual_fill` is set to `true`. For IQRC, when Innovus Advanced VMF is turned on, XL+AA license is required.

# Calculating Delay

- [Overview](#)
- [Data Preparation](#)
  - [Operating Conditions](#)
  - [Timing Library Format](#)
- [Delay Calculation Modes and Related Controls](#)
- [Running Delay Calculation](#)
- [Calculating Delay in Multi-Thread Mode](#)

## Overview

You can perform delay calculation at various stages of the design flow to continuously validate your timing. Advanced Analysis Engine (AAE) delay calculator is used to perform delay calculation, timing analysis, and signal integrity (SI) analysis. AAE is a single-step flow using one engine as compared to the earlier two-step flow required by SignalStorm and CeltIC for performing delay calculation and SI analysis. This new engine performs incremental delay calculation and multithreading, thereby resulting in substantial runtime gains.

 The feDc and the SignalStorm delay calculation engines are obsolete and have been replaced by AAE, which is default since the 13.1 software release.

## Data Preparation

In order to perform delay calculation, you must have a placed design loaded in Innovus System. During Timing Analysis, delay calculation uses information from the Innovus database including:

- Netlist connectivity of the current in-memory database.
- Parasitics derived either from RC extraction or from loading a SPEF-format resistance and capacitance parasitic file.
- Timing Library data in `.lib` or `ldb` format.
- Timing constraints file in SDC format.
- Any user settings that impact delay calculation.

## Operating Conditions

The Innovus system does not automatically import operating conditions from the SDC constraints. Therefore, you must ensure that operating conditions are specified before running delay calculation. Use the `create_delay_corner` and `update_delay_corner` commands to specify the libraries and operating conditions.

## Timing Library Format

By default, the Innovus System supports NLDM, ECSM, and CCS-based Liberty (.lib) timing libraries for performing delay calculation. To help improve ECSM and CCS library loading times, .lib files may be converted into the `ldb` format using the `write_ldb` command

## Delay Calculation Modes and Related Controls

Delays are calculated differently, depending on the number of terminals (fanouts) a net has and the Elmore time constant. The following table lists the calculation modes and related controls used by the software for delay calculation.

Delay Calculation Modes and Related Controls				
	Fanouts $\geq 1,000$	1,000 > Fanouts $\geq 100$	100 > Fanouts and Delay $< 10\text{ns}$	Delay $> 10\text{ns}$
Algorithm	Uses the default delay parameters.	Uses simplified delay calculation mode.	Uses full RC delay calculation mode.	Uses simplified delay calculation mode.

<b>Cell Delay</b>	Uses lumped C lookup with a default value of 0.5 pF.  The value is controlled by the <code>delaycal_default_net_load</code> global variable.	Uses lumped C lookup from total parasitics on the net.	Uses full RC.	Uses lumped C lookup from total parasitics on the net.
<b>Wire Delay</b>	Uses the default value of 1 ns.  The value is controlled by the <code>delaycal_default_net_delay</code> global variable.	Uses Elmore delay.	Uses full RC.	Uses Elmore delay.
<b>Driver Slew Rates</b>	Uses the default value of 0 ps.  The value is controlled by the <code>delaycal_input_transistor_delay</code> global variable.	Uses lumped C lookup from total parasitics on the net.	Uses full RC.	Uses lumped C lookup from total parasitics on the net.
<b>Interconnect Slew Degradation</b>	None	None	Uses full RC. (Slews are degraded across interconnect.)	None
<b>Controls</b>	Fanout threshold is controlled by the <code>delaycal_use_default_delay_limit</code> global variable. The default value is 1,000.	Lower fanout threshold is controlled by the <code>delaycal_use_elmore_delay_limit</code> global variable. The default fanout value is 100.		Delay threshold controlled by <code>delaycal_use_elmore_delay_upper_threshold</code>

## Running Delay Calculation

The delay calculation engine is called automatically during timing analysis. Interactive debug of delays may be performed using `reportDelayCalculation`. You can choose to write the delays to a standard delay format (SDF) file using the `write_sdf` command.

For example, the following command saves the results to the SDF file named `TOPCHIP_SP.sdf`:

```
write_sdf TOPCHIP_SP.sdf
```

## Calculating Delay in Multi-Thread Mode

Multi-threaded delay calculation is automatically enabled when you configure multiple-CPU processing using `setMultiCpuUsage`. You can invoke multi-threaded delay calculation by running any command that needs timing information (for example, `optDesign`, `timeDesign`, `report_timing`, and so on).

### Example:

```
setMultiCpuUsage -localCpu 4
report_timing
```

# Timing Analysis

- [Overview](#)
- [Timing Analysis Features](#)
- [MMMC-On By Default Functionality](#)
- [Before You Begin](#)
- [Calculating Clock Latency](#)
- [Path Exceptions Priority](#)
- [Specifying Timing Analysis Modes](#)
  - [Definition of Early and Late Paths](#)
  - [Single Timing Analysis Mode](#)
  - [Best-Case Worst-Case \(BC-WC\) Timing Analysis Mode](#)
  - [On-Chip Variation \(OCV\) Timing Analysis Mode](#)
- [Clock Path Pessimism Removal](#)
- [Analyzing Timing Problems](#)
  - [Resolving Buffer-Related Problems](#)

## Overview

The goal of timing analysis is to verify that a design meets timing requirements under a specified set of timing constraints, such as arrival and required times, operating conditions, slew rates, false paths, and path delays. Performing timing analysis lets you determine how fast a design can run without incurring timing violations. You can use the results of timing analysis to fine tune and debug the speed-limiting, critical paths in a design.

You can perform timing analysis using Cadence® and third-party constraint formats and timing libraries .lib.

## Timing Analysis Features

Timing analysis includes the following features and capabilities:

### Static Timing Analyzer (STA)

- Performs setup time analysis, which analyzes violating paths due to timing
- Performs hold time analysis
- Performs analysis in ideal and propagated mode
- Reports asynchronous violating paths
- Reports violating paths after running pre-clock tree synthesis (CTS) skew

### What-If Timing Analysis

Use what-if timing analysis to modify instance cell timing information to reach top level timing requirements, after which you can manually change the timing model of a standard cell or modify the timing arcs of blackboxes. Once you have defined the initial timing model of the blackboxes, you can modify arc definitions and verify the consequences in timing analysis.

### Minimum and Maximum Timing Analysis

To read in libraries with multiple operating conditions for minimum and maximum analysis, you can:

- Create a MMMC configuration file. You can use the Innovus System GUI.
- Specify the operating conditions by using the `create_delay_corner -opcond` command.
- Specify the `setAnalysisMode` command.

## Timing Analysis Ideal and Propagated Modes

setAnalysisMode		Clock Propagation	Clock Latency
-skew false		Forced Ideal	No Effect
-skew true	-clockPropagation forcedIdeal	Forced Ideal	SDCs in Effect
-skew true	-clockPropagation sdcControl	*SDCs in Effect	**SDCs in Effect

\* Both `-clockPropagation` `sdcControl` and `set_propagated_clock` required.

\*\* The closest (`set_clock_latency` or `set_propagated_clock`) assertion to the clock endpoint determines ideal vs. propagated mode.

## MMMC-On By Default Functionality

Starting from 11.1 release of the software, all designs need to be MMC-based to initialize into the 11.1 release of the software. The earlier versions of the software supported the two-corner, single-mode (minimum/maximum) operation. Starting from 11.1 release of the software, Innovus follows the `init_design` based flow. This flow requires a valid MMC specification to provide the necessary timing, SI, constraint, and extraction related data for the system. The default MMC objects are treated as real user MMC objects and are saved/restored from the MMC view definition (`viewDefinition.tcl`) file.

## Before You Begin

Before running timing analysis, read in the timing libraries, timing constraints, and the netlist.

Optionally, you can also set the following conditions:

- Specify the delay calculation and RC extraction data.  
Use the *Timing* and *Power* pages in the Design Import form to specify these values. For more information, see [Design Menu](#) in the *Innovus Menu Reference*.
- Specify the operating conditions to use for timing analysis.  
Use the operating conditions to specify process, voltage, and temperature (PVT) values. Operating conditions are defined in the timing library and read into the Innovus session when you import the design. You can use a single set of operating conditions for setup and hold analysis, or you can specify minimum and maximum conditions.

- Check and report timing libraries by generating the timing library report.
- Check and report cell footprints by generating the cell footprint report.
- Define RC corners for extraction. In the MMMC configuration file you can define three different types of RC corners - typical, best, and worst. Several RC corners can be defined.
- Specify the analysis mode you want to use for timing analysis. There are three types of analysis modes: single, best-case worst-case (BC-WC), and on-chip variation. For more information, see "[Specifying Timing Analysis Modes](#)".

For more information, see "Importing and Exporting Designs" chapter of the *Innovus User Guide*.

## Calculating Clock Latency

The Innovus System software calculates clock latency based on the following two settings:

- Analysis mode set using the `setAnalysisMode` command.
- The `set_propagated_clock` and `set_clock_latency` constraints values.

Depending on these settings, the clock latency can be equal to either 0.0 or the value of the `set_clock_latency` constraint, or the delay computed by propagation along the clock path.

The Innovus System software sets the clock latency for various combinations of analysis mode settings as follows:

- `setAnalysisMode -skew true -clockPropagation sdcControl` (**Default Setting**)
  - Latency is defined by the precedence of `set_propagated_clock` and `set_clock_latency` in the SDC.
  - If both `set_propagated_clock` and `set_clock_latency` are not specified, no clock latency is reported (ideal mode).
- `setAnalysisMode -skew true -clockPropagation forcedIdeal`
  - If `set_clock_latency` command is in the timing constraint file, the clock latency specified in the constraint is used (ideal mode).
  - If `set_clock_latency` is not specified, 0ns clock latency is reported (ideal mode).
- **Note:** The `-clockPropagation forcedIdeal` option forces ideal clock mode, even if `set_propagated_clock` is specified in the constraints file.
- `setAnalysisMode -skew false -clockPropagation sdcControl`

Or,

```
setAnalysisMode -skew false -clockPropagation forcedIdeal
```

- No latency is reported (ideal mode).

**Note:** When you use the `-skew false` parameter, clock latencies are ignored.

## Path Exceptions Priority

The priority between different types of path exceptions is as follows:

- a) `set_false_path` (Highest)
- b) `set_max_delay`/`set_min_delay`
- c) `set_multicycle_path`

The priority between two exceptions of similar type is decided on the following basis:

- a) If all the `from`/`through`/`to` list of both the exceptions are same then the last entry has higher priority

For example:

```
set_multicycle_path 3 -from a
set_multicycle_path 4 -from a <---- Higher priority
```

- b) If one exception has `-from` specified and the second exception does not have a `-from` pin, then the first one with `-from` is given higher priority.

For example:

```
set_multicycle_path 3 -from a -through b <---- Higher priority
set_multicycle_path 4 -through b -through c
```

- c) If one exception has `-to` specified and the second exception does not have a `-to` pin, then the first one with `-to` has higher priority.

For example:

```
set_multicycle_path 3 -through b -to d <---- Higher priority
set_multicycle_path 4 -through b -through c
```

- d) If number of `through` list are matching then the exception with more constraining adjustment will be given higher priority.

For example:

```
set_multicycle_path 4 -through b -through c
set_multicycle_path 3 -through b <---- Higher priority
```

e) If number of from/through/to list are matching and the exceptions differ only by edges, then the more constraining one gets higher priority.

For example:

```
set_multicycle_path 3 -from a <----- Higher priority  
set_multicycle_path 4 -fall_from a
```

f) The clock based exceptions have a lower priority than pin based exceptions.

g) If a multicycle path is specified without -setup|hold option, then a 0 hold is assumed while the specified cycle will be applied as setup.

For example:

```
set_multicycle_path -from a 7
```

is equivalent to:

```
set_multicycle_path -from a 7 -setup  
set_multicycle_path -from a 0 -hold
```

g) In case of hold analysis, the effective multicycle path is derived from the most constraining setup and explicit hold constraint, if specified.

For example:

```
in ----- |>----- |>----- out  
buf1 buf2
```

If there are following constraints specified:

- 1) set\_multicycle\_path 4 -setup -through buf1/A
- 2) set\_multicycle\_path 3 -hold -through buf1/A
- 3) set\_multicycle\_path 2 -setup -through buf2/A <-----
- 4) set\_multicycle\_path 1 -hold -through buf2/A <-----

Then the CTE behavior while doing Hold analysis is:

- a) Most constraining setup constraint will be picked (i.e. constraint 3).
- b) Inferred hold constraint should be derived from setup constraint picked in a). That is, Inferred Hold constraint will be corresponding to constraint 3).
- c) CTE picks the most constraining explicit HOLD constraint (constraint 4).

## Specifying Timing Analysis Modes

The Innovus software provides different timing analysis modes and performs different calculations for setup and hold checks for each mode. The timing analysis modes are divided as follows:

- [Single Timing Analysis Mode](#)

In single analysis mode, only maximum delay values and -max options of constraints are used for both min and max analysis.

- [Best-Case Worst-Case \(BC-WC\) Timing Analysis Mode](#)

Minimum delays from BC and max delays from WC should not be used together to evaluate a timing check because the BC and WC operating conditions or corners are vastly different.

In BC-WC analysis mode the software uses the maximum delays for all paths during setup checks and minimum delays for all paths during hold checks.

- [On-Chip Variation \(OCV\) Timing Analysis Mode](#)

OCV is the small difference in the operating parameter value across the chip. Each timing arc in the design can have an early and a late delay to account for the on-chip process, voltage and temperature variation. These delays are used together in the analysis of each check.

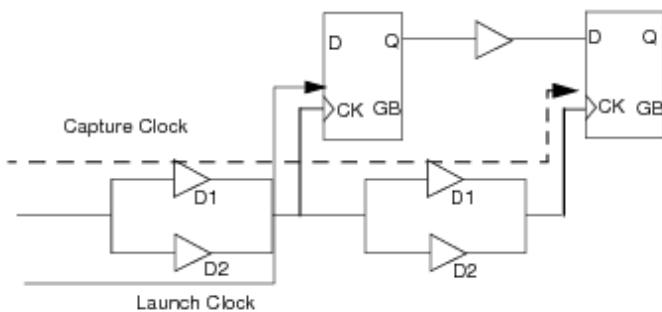
In OCV mode, the software calculates clock and data path delays based on minimum and maximum operating conditions for setup analysis and vice-versa for hold analysis.

## Definition of Early and Late Paths

The timing analysis modes described in this section refer to early and late paths and their usage in slack calculation. The early and late paths are the shortest and the longest paths respectively.

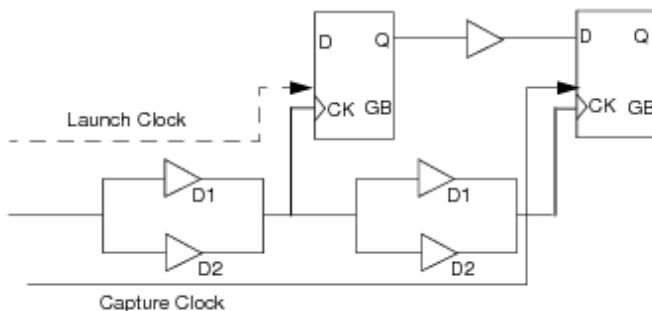
The following figure shows a setup check with late (shown in solid line) launch clock and early capture clock (shown in dotted line).

Figure: **Setup Check**



The following figure shows hold check with early launch clock (shown in dotted line), and late capture clock (shown in solid line).

**Figure 29-2 Hold Check**



## Single Timing Analysis Mode

In this mode, the Innovus software uses a single set of delays (using one library group) based on one set of process, temperature, and voltage conditions. To set the timing analysis mode as single, use the `-analysisType single` parameter of the `setAnalysisMode` command.

### Setup Check in Single Timing Analysis Mode

For setup check, the software checks the late launch clock and late data paths against early capture clock path.

For zero slack value in a setup check, the following condition should be met:

$$\text{launch clock late path} + \text{data clock late path} \leq \text{capture clock early path} + \text{clock period} - \text{setup}$$

## Hold Check in Single Timing Analysis Mode

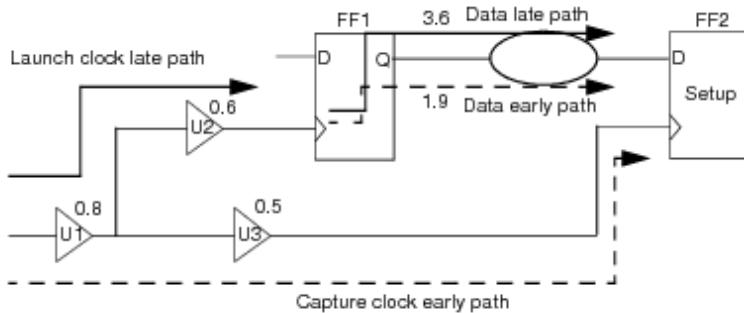
For hold check the software compares the early arriving data against the late arriving clock at the endpoint.

For zero slack value in a hold check, the following condition should be met:

launch clock early path + data clock early path  $\geq$  capture clock late path + hold

### Example 29-1 Setup Check in Single Timing Analysis Mode

The following figure shows the setup check on the path from FF1 to FF2.



The software uses a library to scale all delays at WC conditions. For setup check, the software considers two paths between the two registers, FF1 and FF2. The software considers only the late path delay to calculate slack during setup check.

The following values are assumed in this example:

Data late path delay = 3.6

Data early path delay = 1.9

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock mode = Propagated clock mode

The software computes the slack as follows:

Launch clock late path delay =  $0.8 + 0.6 = 1.4$

Data late path delay = 3.6

Capture clock early path delay =  $0.8 + 0.5 = 1.3$

Setup = 0.2

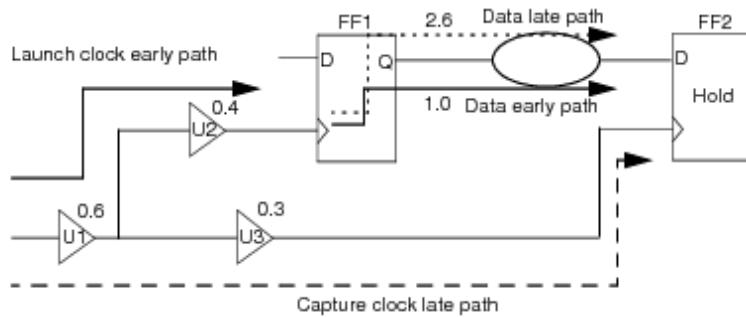
Data arrival time =  $1.4 + 3.6 = 5$

Data required time =  $4 + 1.3 - 0.2 = 5.1$

Slack =  $5.1 - 5 = 0.1$

### Example 29-2 Hold Check in Single Timing Analysis Mode

The following figure shows the hold check on the path from FF1 to FF2.



The software uses a library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock early path delay =  $0.6 + 0.4 = 1.0$

Data early path delay = 1.0

Capture clock late path delay =  $0.6 + 0.3 = 0.9$

Hold = 0.1

Data arrival time =  $1 + 1 = 2$

Data Required Time =  $0.1 + 0.9 = 1$

Slack =  $2 - 1 = 1$

## Performing Timing Analysis in Single Analysis Mode

1. Create a single corner MMMC configuration file and set `init_mmmc_file` variable to point to it:

```
create_library_set -name my_max_library_set
```

```
-timing [list /icd/libs/syn/stdcell/slow/slow.lib]

create_constraint_mode -name my_constraint_mode
    -sdc_files [list ./constraints/design.sdc]

create_rc_corner -name my_wc_corner_worst
    -qx_tech_file /icd/libs/tech/6mlv-wc.tch

create_delay_corner -name my_delay_corner_max
    -library_set my_max_library_set
    -rc_corner my_wc_corner_worst
    -opcond slow

create_analysis_view -name my_wc_analysis_view
    -constraint_mode my_constraint_mode
    -delay_corner my_delay_corner_max

set_analysis_view -setup my_wc_analysis_view -hold my_wc_analysis_view
```

**2. Load the design using the following commands:**

```
source init.globals
init_design
```

**3. Set the analysis mode to single, setup and propagated clock mode.**

```
setAnalysisMode -analysisType single -checkType setup -skew true -clockPropagation
sdccontrol
```

**4. Generate the timing reports for setup.**

```
report_timing
```

**5. Set the analysis mode to hold and propagated clock mode.**

```
setAnalysisMode -checkType hold -skew true -clockPropagation sdcControl
```

**6. Generate the timing reports for hold.**

```
report_timing
```

## **Best-Case Worst-Case (BC-WC) Timing Analysis Mode**

In BC-WC timing analysis mode, the Innovus software considers two operating conditions. The software checks both operating conditions in one timing analysis run.

To set the timing analysis mode as BC-WC, use the `-analysisType bcWC` parameter of the [setAnalysisMode](#) command.

You can use the `set_clock_latency` constraint to set the source latency for a clock in both ideal and propagated mode for setup and hold checks. You can also use the constraint to set the network latency for an ideal clock. The specified source or network latency affects the early and late clock paths for both capture and launch clocks for both min and max operating conditions. The software considers the network latency that you set using the `set_clock_latency -max` or `-min` constraint for ideal clocks only.

## Setup Check in BC-WC Mode

For setup check, the software calculates delay values from the Max library group for data arrival time, and network delay of both launch and capture clocks (in propagated mode).

The software scales the delay values using the operating condition that you specified. The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path (max)	<code>set_clock_latency -source -late -max value</code>
Capture clock early path (max)	<code>set_clock_latency -source -early -max value</code>

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path (max)	<code>set_clock_latency -max value</code>
Capture clock early path (max)	<code>set_clock_latency -max value</code>

## HOLD Check in BC-WC Mode

For HOLD check, the software uses the delay values from the Min library for the data arrival time, and network delay of both launch and capture clocks (in propagated mode). The software scales the delay values using the operating condition that you specified.

Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	<code>set_clock_latency -source -early -min value</code>
Capture clock late path (min)	<code>set_clock_latency -source -late -min value</code>

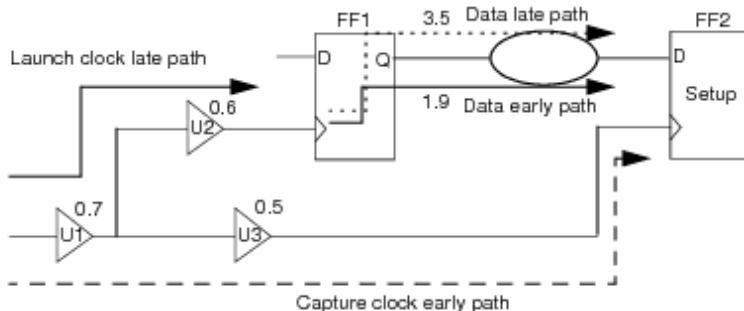
The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	<code>set_clock_latency -min value</code>
Capture clock late path (min)	<code>set_clock_latency -min value</code>

**Note:** You can also use one library containing two operating conditions in this mode.

### Example 29-3 Setup Check in BC-WC Timing Analysis Mode

The following shows the setup check on the path from FF1 to FF2.



The software uses the Max library to scale all delays at WC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock late path delay =  $0.7 + 0.6 = 1.3$

Data late path delay = 3.5

Capture clock early path delay =  $0.7 + 0.5 = 1.2$

Setup = 0.2

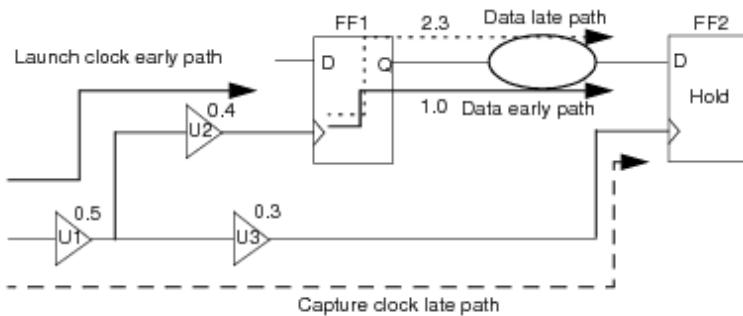
Data arrival time =  $1.3 + 3.5 = 4.8$

Data required time =  $4 + 1.2 - 0.2 = 5$

Slack =  $5 - 4.8 = 0.2$

### Example Hold Check in BC-WC Timing Analysis Mode

The following figure shows the hold check on the path from FF1 to FF2.



The software uses the Min library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock early path delay =  $0.5 + 0.4 = 0.9$

Data early path delay = 1.0

Capture clock late path delay =  $0.3 + 0.5 = 0.8$

Hold = 0.1

Data arrival time =  $0.9 + 1 = 1.9$

Data required time =  $0.1 + 0.8 = 0.9$

Slack =  $1.9 - 0.9 = 1$

## Performing Timing Analysis in BC-WC Analysis Mode

To perform timing analysis in BC-WC analysis mode, complete the following steps:

1. Create a BC-WC MMMC configuration file, and set `init_mmmc_file` variable to point to it:

```
create_library_set -name my_max_library_set
    -timing [list /icd/libs/syn/stdcell/slow/slow.lib]
create_library_set -name my_min_library_set
    -timing [list /icd/libs/syn/stdcell/fast/fast.lib]

create_constraint_mode -name my_constraint_mode
    -sdc_files [list ./constraints/design.sdc]

create_rc_corner -name my_wc_corner_worst
    -qx_tech_file /icd/libs/tech/6mlv-wc.tch

create_rc_corner -name my_bc_corner_worst
    -qx_tech_file /icd/libs/tech/6mlv-wc.tch

create_delay_corner -name my_delay_corner_max
    -library_set my_max_library_set
    -rc_corner my_wc_corner_worst
    -opcond slow
create_delay_corner -name my_delay_corner_min
    -library_set my_min_library_set
    -rc_corner my_bc_corner_worst
    -opcond fast

create_analysis_view -name my_wc_analysis_view
    -constraint_mode my_constraint_mode
    -delay_corner my_delay_corner_max
create_analysis_view -name my_bc_analysis_view
    -constraint_mode my_constraint_mode
    -delay_corner my_delay_corner_min

set_analysis_view -setup my_wc_analysis_view -hold my_bc_analysis_view
```

2. Load the design using the following commands:

```
source init.globals
init_design
```

3. Set the analysis mode to BC-WC, setup and propagated clock mode.

```
setAnalysisMode -analysisType bcwc -checkType setup -skew true -clockPropagation
```

sdcControl

4. Generate the timing reports for setup.

report\_timing

5. Set the analysis mode to hold and propagated clock mode.

setAnalysisMode -checkType hold

6. Generate the timing reports for hold.

report\_timing

## On-Chip Variation (OCV) Timing Analysis Mode

### Setup Check

In OCV mode setup check, the software uses the timing delay values from the late library set and operating conditions for the data and the launch clock network delay. The software uses the delay values from the early library set and operating conditions for the capturing clock network delay assuming that the clocks are set in propagated mode.

**Note:** You can also use one library instead of max and min libraries.

The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path (max)	<code>set_clock_latency -source -late -max value</code> Or, <code>set_clock_latency -source -late value</code>
Capture clock early path (min)	<code>set_clock_latency -source -early -min value</code> Or, <code>set_clock_latency -source -early value</code>

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path	<code>set_clock_latency -max value</code>
Capture clock early path	<code>set_clock_latency -min value</code>

## Hold Check

In OCV hold check, the software uses the timing delay values from the early library set and operating conditions for the data arrival time and launch clock network delay. The software uses delay values from the late library set and operating conditions for the capturing clock network delay assuming that the clocks are set in propagated mode.

The source latency in both ideal and propagated modes for hold checks is defined in the constraints used by various clock paths as follows:

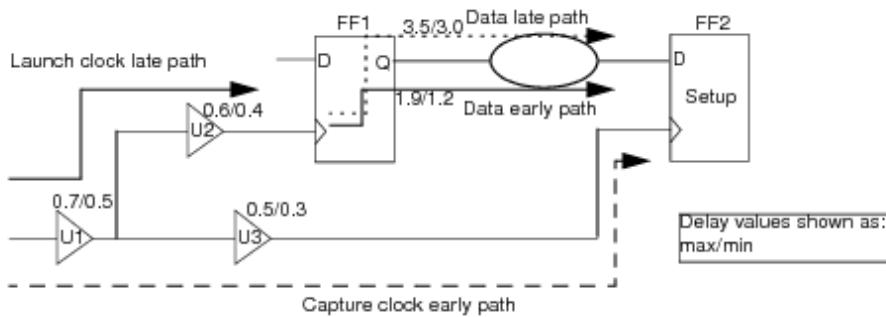
Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	<code>set_clock_latency -source -early -min value</code> Or, <code>set_clock_latency -source -early value</code>
Capture clock late path (max)	<code>set_clock_latency -source -late -max value</code> Or, <code>set_clock_latency -source -late value</code>

The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock early path	<code>set_clock_latency -min value</code>
Capture clock late path	<code>set_clock_latency -max value</code>

### Example 29-5 Setup Check in OCV Timing Analysis Mode

The following figure shows the setup check on the path from FF1 to FF2.



The software uses the Max library for all late path delays and Min library for all early path delays.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock mode = Propagated clock mode

The software computes the slack as follows:

Launch clock late path delay (max) =  $0.7 + 0.6 = 1.3$

Data late path delay (max) = 3.5

Capture clock early path delay (min) =  $0.5 + 0.3 = 0.8$

Setup = 0.2

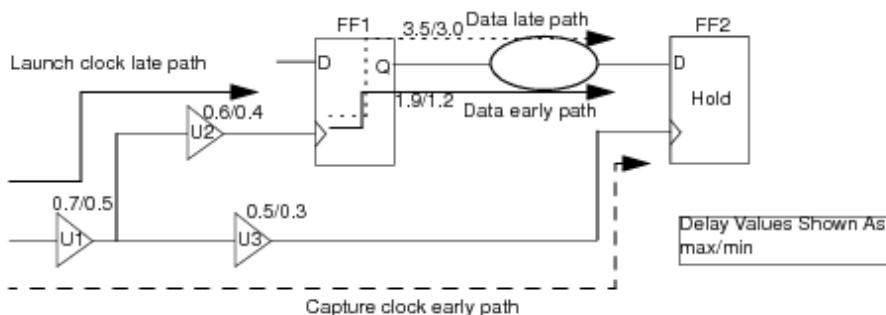
Data arrival time =  $1.3 + 3.5 = 4.8$

Data required time =  $4 + 0.8 - 0.2 = 4.6$

Slack =  $4.6 - 4.8 = -0.2$

### Example 29-6 Hold Check in OCV Timing Analysis Mode

The following figure shows the hold check on the path from FF1 to FF2.



The software uses the Max library to scale all delays at WC conditions and Min library to scale all

delays at BC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock mode = Propagated clock mode

The software computes the slack as follows:

Launch clock early path delay (min) =  $0.5 + 0.4 = 0.9$

Data early path delay (min) = 1.2

Capture clock late path delay (max) =  $0.7 + 0.5 = 1.2$

Hold = 0.1

Data arrival time =  $0.9 + 1.2 = 2.1$

Data required time =  $0.1 + 1.2 = 1.3$

Slack =  $2.1 - 1.3 = 0.8$

## Performing Timing Analysis in OCV Mode with Two Libraries And Operating Conditions

1. Create a MMMC configuration file for OCV analysis. You do not need to create special MMMC configurations, specifically for BC-WC vs. OCV analysis. But if you wish to perform OCV with specific libraries and/or operating conditions for early or late mode, these options should be coded within a single delay corner object.

To create library sets, corners, and modes, use the following set of commands:

```
create_delay_corner -name my_delay_corner_max
    -early_library_set my_max_library_set_1p3_V
    -late_library_set my_max_library_set_1p1_V
    -rc_corner my_wc_corner_worst
    -early_opcond slow_1p3V
    -late_opcond slow_1p1V
```

```
create_analysis_view -name my_wc_analysis_view
    -constraint_mode my_constraint_mode
    -delay_corner my_delay_corner_max
```

```
set_analysis_view -setup my_wc_analysis_view -hold my_wc_analysis_view
```

2. Load the design by using the following commands:

```
source init.globals  
init_design
```

3. Set the analysis mode to OCV and propagated clock mode.

```
setAnalysisMode -analysisType onChipVariation -skew true  
-clockPropagation sdcControl
```

4. Generate the timing reports for setup.

```
report_timing -late
```

5. Generate the timing reports for hold.

```
report_timing -early
```

## Using `set_timing_derate` with OCV Analysis Mode

The `set_timing_derate` command affect the following paths in OCV mode:

Violations	Data	Launch Clock	Capture Clock
SETUP	-late	-late	-early
	-data	-clock	-clock
HOLD	-early	-early	-late
	-data	-clock	-clock

## Clock Path Pessimism Removal

Clock Path Pessimism Removal (CPPR) or clock reconvergence pessimism removal (CRPR) is the process of identifying and removing the pessimism introduced in the slack reports for clock paths when the clock paths have a segment in common.

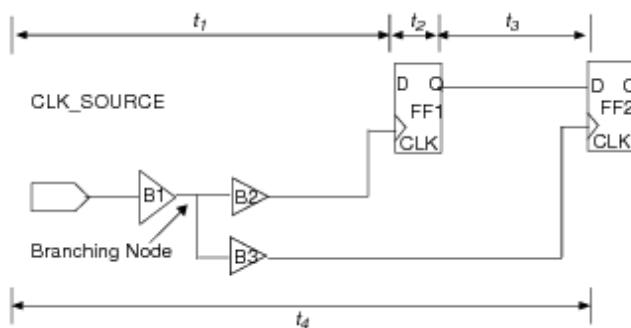
You can introduce early or late delay variations by using the `setAnalysisMode -analysisType onChipVariation` or by using derating factors in BcWc analysis mode. In CPPR mode, the difference between late and early delays is removed for common clock tree segments of the launching and latching devices.

When you use the `setAnalysisMode -analysisType BcWc` and the `set_timing_derate` commands, the software calculates maximum delay value by multiplying the delay by the scale value that you set using `set_timing_derate -late`. Similarly, the software calculates the minimum delay value by multiplying the delay by the scale value that you set using `set_timing_derate -early`.

When you use the `setAnalysisMode -analysisType onChipVariation` command, the software uses the maximum and minimum operating conditions to calculate the minimum and maximum delays. In this case also if you specify one operating condition, the software uses the `set_timing_derate` command.

As shown in Figure 29-3, the setup check at FF2 compares the maximum delay data at the D pin against minimum delay clock at the CLK pin. The maximum delay data at FF2/D consists of a sum of maximum signal delay from FF1/Q to FF2/D, the maximum delay from CLK\_SOURCE to FF1/CLK, and the delay from FF1/CLK to FF1/Q. Similarly, the minimum delay clock arrival time at FF2/CLK is the minimum delay from CLK\_SOURCE to FF2/CLK.

**Figure Example Signal Path**



The setup check equation for the example in Figure 29-3 with pessimism is as follows:

$$t_{1max} + t_{2max} + t_{3max} \leq t_{4min} + t_{pa} - t_{su}$$

where,

$t_1$  = Delay value for launch clock late path

$t_2$  = Delay between FF1/CLK and FF1/Q

$t_3$  = Delay between FF1/Q and FF2/D

$t_2 + t_3$  = Delay value for late data path

$t_4$  = Delay value for capture clock early path

$t_{pa}$  = Period adjustment

$t_{su}$  = Setup time

The setup check equation incorrectly implies that the common clock network, B1, can simultaneously use maximum delay for the launch clock late path (clock source to FF1/CLK) and minimum delay for the capture clock early path (clock source to FF2/CLK). You use the CPPR to remove this pessimism.

Setup check equation using CPPR is as follows:

$$t_{1max} + t_{2max} + t_{3max} \leq t_{4min} + t_{pa} - t_{su} + t_{cppr}$$

Where,

$t_{CPPR}$  = Difference in the maximum and minimum delay from the clock source to the branching node

Similarly, hold check equation using CPPR is as follows:

$$t_{1min} + t_{2min} + t_{3min} + t_{cppr} \leq t_{4max} + t_H$$

Where,

$t_1$  = Delay value for launch clock early path

$t_2$  = Delay between FF1/CLK and FF1/Q

$t_3$  = Delay between FF1/Q and FF2/D

$t_2 + t_3$  = Delay value for early data path

$t_4$  = Delay value for capture clock late path

$t_{CPPR}$  = Difference in the maximum and minimum delay from the clock source to the branching node

$t_H$  = Hold time

For example, you use the following `set_timing_derate` command for setup check delays in [Figure A-29-3](#) in scaled on-chip variation analysis methodology:

`set_timing_derate -max -late 1 -early 0.9 -clock`

With the `set_timing_derate` command, if the delay through B1 is 1ns, the software removes the pessimism of 0.1ns. Without CPPR the analysis tool incorrectly assumes that B1 can have a delay of 1 and 0.9. The common path pessimism time ( $t_{CPPR}$ ) is calculated as follows:

$B1 * Late\ Derate - B1 * Early\ Derate = t_{CPPR}$

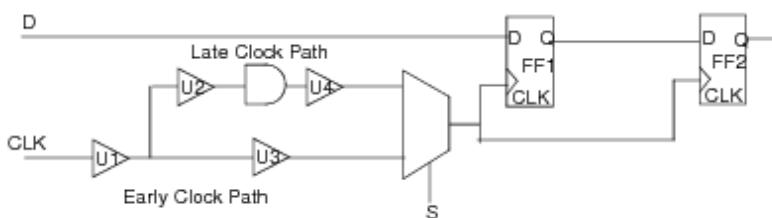
Therefore,

$$t_{CPPR} = 1.0\text{ns} * 1.0 - 1.0\text{ns} * 0.9 = 0.1\text{ns}$$

## CPPR and Reconvergent Logic

If a design contains reconvergent logic on the clock path, the timing analysis software might assume certain pessimism while calculating slack.

The following figure shows a circuit for which timing analysis is done in single analysis mode.



In this case, if `set_case_analysis` is not set at point S of the multiplexer, the timing analysis tools assume different delay values for early and late paths. For example, if early path has delay of 0.5ns, and late path has delay of 1ns, a pessimism equal to 0.5ns is introduced in the design.

The above pessimism is not specific to single analysis mode only; it also applies to best-case/worst-case and on-chip variation methodology.

Innovus uses a default threshold of 20ps during pessimism removal. That means 20ps of uncertainty remains in the analysis. To reset the threshold value you can use the `timing_cppr_threshold_ps` global variable. Setting this global to a specified value means that all the paths might be reported without having their pessimism removed.

## CPPR Flow

To remove this pessimism, use the `-cppr` parameter in the `setAnalysisMode` command.

In Innovus, the following flow supports the CPPR feature:

1. Load the design.
2. Set the analysis mode to setup, propagated clock and CPPR.

```
setAnalysisMode -checkType setup -skew true
-clockPropagation sdcControl -cppr both
```

3. Set the derating values.

```
set_timing_derate -late 1 -early 0.9 -clock
```

4. Generate timing report. You use the `report_timing` command to remove delay pessimism from paths that have a portion of the clock network in common.

```
report_timing
```

## Timing Analysis Results Before and After CPPR

The following example shows a timing report generated before CPPR analysis.

```
Path 1: MET Setup Check with Pin reg_2/CK
Endpoint: reg_2/D (v) checked with leading edge of 'CLK1'
Beginpoint: reg_1/Q (v) triggered by leading edge of 'CLK1'
Other End Arrival Time 0.104
- Setup 0.167
+ Phase Shift 2.000
= Required Time 1.938
- Arrival Time 1.850
= Slack Time 0.088
Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	clk ^			0.000	0.088
ck_0	A ^ -> Y ^	BUFX2	0.091	0.091	0.178
ck_1	A ^ -> Y ^	BUFX2	0.097	0.188	0.275
ck_2	A ^ -> Y ^	BUFX2	0.094	0.282	0.369
ck_3	A ^ -> Y ^	BUFX2	0.092	0.374	0.462
ck_4	A ^ -> Y ^	CLKAND2X2	0.150	0.524	0.612
reg_1	CK ^ -> Q v	DFFRHQX1	0.288	0.812	0.900
t_1	A ^ -> Y ^	BUFX8	0.111	0.923	1.011

t_2	A ^ -> Y ^	BUFX8	0.092	1.015	1.103
t_3	A ^ -> Y ^	BUFX8	0.092	1.107	1.195
t_4	A ^ -> Y ^	BUFX8	0.092	1.199	1.287
t_5	A ^ -> Y ^	BUFX8	0.092	1.291	1.379
t_6	A ^ -> Y ^	BUFX8	0.092	1.383	1.471
t_7	A ^ -> Y ^	BUFX8	0.092	1.475	1.563
t_8	A ^ -> Y ^	BUFX8	0.092	1.567	1.655
t_9	A ^ -> Y ^	BUFX8	0.092	1.660	1.747
t_10	A ^ -> Y ^	BUFX8	0.088	1.747	1.835
t_11	B ^ -> Y ^	NAND2X1	0.066	1.813	1.901
t_12	A ^ -> Y ^	INVX1	0.037	1.850	1.938
reg_2	D v	DFFRHQX1	0.000	1.850	1.938

The following example shows a timing report generated after CPPR analysis:

Path 1: MET Setup Check with Pin reg\_2/CK

Endpoint: reg\_2/D (v) checked with leading edge of 'CLK1'

Beginpoint: reg\_1/Q (v) triggered by leading edge of 'CLK1'

Other End Arrival Time 0.104

- Setup 0.167

+ Phase Shift 2.000

**+ CPPR Adjustment** 0.420

= Required Time 2.358

- Arrival Time 1.850

= Slack Time 0.508

Clock Rise Edge 0.000

= Beginpoint Arrival Time 0.000

**Innovus User Guide**  
**Analysis Capabilities--Timing Analysis**

---

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	clk ^			0.000	0.508
ck_0	A ^ -> Y ^	BUFX2	0.091	0.091	0.598
ck_1	A ^ -> Y ^	BUFX2	0.097	0.188	0.695
ck_2	A ^ -> Y ^	BUFX2	0.094	0.282	0.789
ck_3	A ^ -> Y ^	BUFX2	0.092	0.374	0.882
ck_4	A ^ -> Y ^	CLKAND2X2	0.150	0.524	1.032
reg_1	CK ^ -> Q v	DFFRHQX1	0.288	0.812	1.320
t_1	A ^ -> Y ^	BUFX8	0.111	0.923	1.431
t_2	A ^ -> Y ^	BUFX8	0.092	1.015	1.523
t_3	A ^ -> Y ^	BUFX8	0.092	1.107	1.615
t_4	A ^ -> Y ^	BUFX8	0.092	1.199	1.707
t_5	A ^ -> Y ^	BUFX8	0.092	1.291	1.799
t_6	A ^ -> Y ^	BUFX8	0.092	1.383	1.891
t_7	A ^ -> Y ^	BUFX8	0.092	1.475	1.983
t_8	A ^ -> Y ^	BUFX8	0.092	1.567	2.075
t_9	A ^ -> Y ^	BUFX8	0.092	1.660	2.167
t_10	A ^ -> Y ^	BUFX8	0.088	1.747	2.255
t_11	B ^ -> Y ^	NAND2X1	0.066	1.813	2.321
t_12	A ^ -> Y ^	INVX1	0.037	1.850	2.358
reg_2	D v	DFFRHQX1	0.000	1.850	2.358

# Analyzing Timing Problems

In addition to the detailed timing violation report, the following report commands are helpful in analyzing timing problems:

- [check\\_timing](#)

Performs a variety of consistency and completeness checks on the timing constraints specified for a design. Use the `check_timing` command after setting all constraints, but before any timing analysis commands, such as `report_timing`, to verify that the timing environment is complete and self-consistent.

- [get\\_property](#)

Retrieves timing information for the specified property on the given pin, net, timing arc, or clock.

- [report\\_analysis\\_coverage](#)

Provides information about the timing checks in the design.

- [report\\_annotated\\_check](#)

Reports coverage of annotated timing checks.

- [report\\_annotated\\_delay](#)

Reports SDF design annotations coverage.

- [report\\_annotated\\_parasitics](#)

Reports the back-annotated parasitics of the design.

- [report\\_case\\_analysis](#)

Reports ports and pins with `set_case_analysis` constraint.

- [report\\_cell\\_instance\\_timing](#)

Reports instance pin and delay arc timing information.

- [report\\_clock\\_timing](#)

Generates a clock skew report for the current design.

- [report\\_clocks](#)

Reports clock waveform, clock arrival point and clock uncertainty information.

- [report\\_inactive\\_arcs](#)

Reports all disabled timing arcs and checks.

- [report\\_path\\_exceptions](#)

Reports design path exceptions such as `set_false_path`, `set_multicycle_path`, `set_max_delay` and `set_min_delay`.

- [report\\_timing](#)

Generates a timing report that provides information about the various paths in the design. The report typically contains data on the delay through the entire path. The start node and the end node of each path is identified.

- [reportAnalysisMode](#)

Reports the current setting for building the timing graph. Use `setAnalysisMode` to change the settings.

- [report\\_constraint](#)

Reports constraint information of current design.

- [report\\_fanin](#)

Allows a cone traversal that is not tied to the timing graph, that is, not blocked by `set_disable_timing` and case analysis.

- [report\\_fanout](#)

Allows a cone traversal that is not tied to the timing graph, that is, not blocked by `set_disable_timing` and case analysis.

- [timeDesign](#)

Performs routing, extraction, timing analysis and generates detailed timing reports.

## Resolving Buffer-Related Problems

You may encounter some of the following buffer-related problems when running timing analysis:

- The logical cell or buffer equivalence, based on cell functionality not used during timing optimization, can cause timing optimization to ignore timing violations.
- If an incorrect buffer footprint name was entered for the set of buffers to run timing optimization, use the `reportFootPrint` command to list the current footprint information.

- If the `in_place_swap_mode:match_footprint` statement is in the timing library, then timing optimization matches up all the cells with same `cell_footprint` name, and logical cell or buffer equivalence will not be used.
- If the `in_place_swap_mode:match_footprint` statement does not exist, then timing optimization derives logical cell equivalence based on matching function: `boolean_eq`.
- If you want the logical cell equivalence based on matching, comment out the `in_place_swap_mode:match_footprint` statement in the timing library.

# Debugging Timing Results

- [Overview](#)
- [Timing Debug Flow](#)
- [Generating Timing Debug Report](#)
- [Displaying Violation Report](#)
- [Analyzing Timing Results](#)
  - [Viewing Power Domain Information](#)
- [Creating Path Categories](#)
  - [Creating Predefined Categories](#)
  - [Creating New Categories](#)
  - [Creating Sub-Categories](#)
  - [Hiding path categories](#)
  - [Reporting Path Categories](#)
- [Using Categories to Analyze Timing Results](#)
  - [Analyzing MMMC Categories](#)
  - [Manual Slack Correction of Categories](#)
- [Editing Table Columns](#)
  - [Cell Coloring](#)
- [Viewing Schematics](#)
- [Running Timing Debug with Interface Logic Models](#)

## Overview

Innovus provides the Global Timing Debug feature for debugging the timing results. The various Timing Debug forms provide easy visual access to the timing reports and debugging tools.

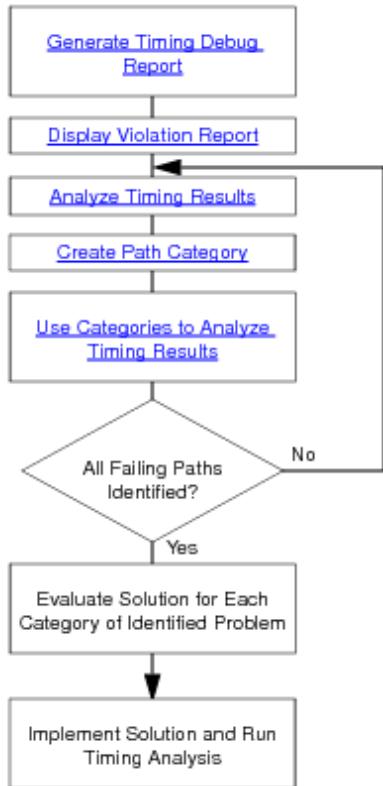
You can group all paths that are failing for the same reason and apply solutions for faster timing closure. You can cross-probe between the timing paths in the timing report and display area in the Layout window.

**Note:** If you have a previously saved timing debug report, you can use the timing debug feature even when the design is not loaded in the Innovus session.

## Timing Debug Flow

You can generate a detailed violation report to list the details of all violating paths. You can then use the timing debug capability to visually identify problems with critical paths in this report. After identifying the problems, you group all paths with the same problem under a single category. You can define several categories to capture all problems related to the violating paths before fixing the problems and running timing analysis again.

Following is the flow for debugging timing results.



**Note:** The file gtd.pref.tcl is loaded automatically at launch.

## Generating Timing Debug Report

Innovus uses a machine readable timing report to display timing debug information. The report is generated in the ASCII format and contains details of all violating paths. By default, the report has .mtarpt extension.

To generate a violation report, use one of the following options:

- Use `report_timing -machine_readable` command

You can also generate text-format report from a machine readable report.

To generate the text report, use the following:

- Use the `write_text_timing_report` command
- Use the *Write Textual Timing Report* form

## Displaying Violation Report

To analyze the timing results, you need to load the machine readable timing report in Innovus.

To display the violation report, use one of the following options:

- Specify the file name in the *Display/Generate Timing Report* form.

**Note:** To select an existing file, deselect the *Generate* option before clicking on the directory icon to the right of the *Timing Report File* field.

By default, the global timing debug engine uses the following command to generate a machine-readable timing analysis report for the GUI display:

```
report_timing -machine_readable -max_points 10000 -max_slack 0.75
```

- Use the `load_timing_debug_report` command.

**Note:** Use the *Append to Current Report* option in the *Display/Generate Timing Report* form to load multiple reports in a single session.

## Analyzing Timing Results

Innovus provides Timing Debug feature to visually analyze timing problems.

You can analyze the following data in the *Timing Debug* form:

- Visual display of passing and failing paths as a histogram. Failing paths are represented in red and passing paths are represented in green color. The goal of timing debug process is to identify paths that fall in red category.
- Details of the critical paths in the Path list. You identify a critical path in this list for further analyses using the Timing Path Analyzer form.

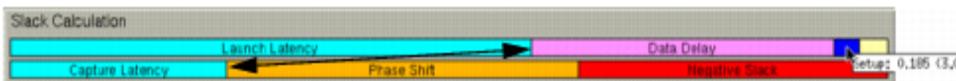
- Visual display of paths reported in different timing reports. When you load multiple debug reports in a single timing debug session, the paths are displayed in different colors corresponding to the report file they are coming from. You can move the cursor over a path to display the name of the report file.

You analyze the following data in the *Timing Path Analyzer* form:

- Slack calculation bars for arrival and required times. You can identify clock skew issues, latency balancing or large clock uncertainty issues using these bars.

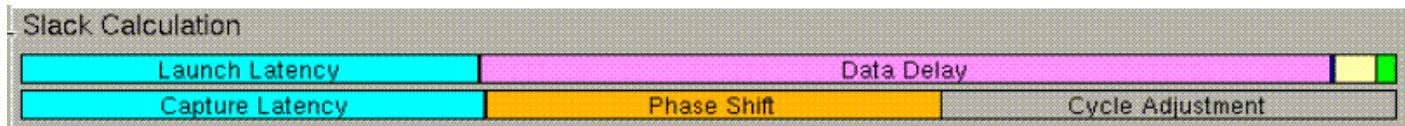
The following examples illustrate the problems that you can identify using the slack calculation bars in the *Timing Path Analyzer* form.

#### Example 9-1



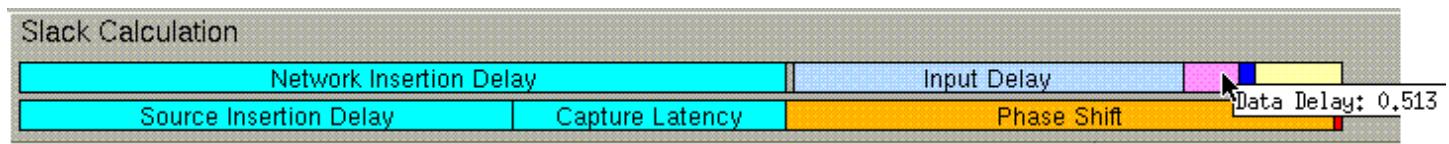
Launch and capture latency components are not aligned. Therefore there can be large clock-latency mismatch in this path.

#### Example 9-2



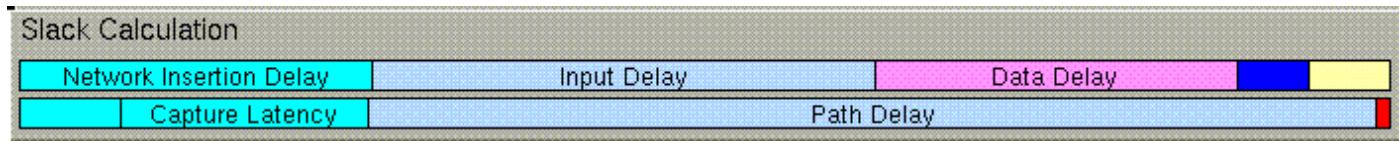
The cycle adjustment bar in the required time indicates presence of multicycle path.

#### Example 9-3



Large input delay in an I/O path is represented by the blue bar in the arrival time.

#### Example 9-4



Path Delay bar in the required time indicates a `set_max_delay` constraint.

- Path details including launch and capture. This information is provided as tabs in the *Timing Path Analyzer* form. You can click on a single path to display it in the design display area.

You can select each element consecutively to trace the entire path in the design display area. This form has a Status column that indicates the status of the path as follows:

Flag	Description
a	Assign net
b	Blackbox instance
c	Clock net
cr	Cover cell
f	Preplaced instance
i	Ignore net
s	Skip route net
t	"don't touch" net
t	instance marked as "don't touch"
T	instance not marked as "don't touch" when the module it belongs to is marked as "don't touch"
u	Unplaced cell
x	External net

- SDC related to the path. The Path SDC tab displays the SDC constraints related to the selected path. The list contains the name of the SDC file, the line number that indicates the position of the constraint in the SDC file, and the constraint definition.

The commands that can be displayed in the Path SDC tab are:

- `create_clock`
- `create_generated_clock`
- `group_path`
- `set_multicycle_path`
- `set_false_path`
- `set_clock_transition`

- set\_max\_delay
- set\_min\_delay
- set\_max\_fanout
- set\_fanout\_load
- set\_min\_capacitance
- set\_max\_capacitance
- set\_min\_transition
- set\_max\_transition
- set\_input\_transition
- set\_capacitance
- set\_drive
- set\_driving\_cell
- set\_logic\_one
- set\_logic\_zero
- set\_dont\_use
- set\_dont\_touch
- set\_case\_analysis
- set\_input\_delay
- set\_output\_delay
- set\_annotated\_check
- set\_clock\_uncertainty
- set\_clock\_latency
- set\_propagated\_clock
- set\_load
- set\_disable\_clock\_gating\_check
- set\_clock\_gating\_check
- set\_max\_time\_borrow
- set\_clock\_groups

When you create a constraint on the command line, the Path SDC tab interactively displays the result of the additional constraint.

**Note:** MMMC views are not displayed interactively.

**Note:** You can create a path category directly from SDC constraints in the Path SDC form. When you right-click a constraint and view the *Create Path Category* form, to see the line number (from the SDC file) and the name of the constraint.

In Innovus, you can also create a path category based on SDC constraint using the -sdc parameter of the [create\\_path\\_category](#) command:

```
create_path_category -name category_name -sdc {file_name line_number}
```

where *file\_name* is the name of the constraint file and *line\_number* is the line number of the SDC constraint.

- Schematic display of the path. The Schematics tab displays the gate-level schematic view of the critical path. For more information, see [Viewing Schematics](#).
- Timing interpretation for the path. This feature provides rule-based path analysis to help you discover sources of potential timing problems in a path. By default, the software performs the following checks on the following rules:
  - Path structure
    - Transparent Latch in Path
    - Clock Gating
    - Hard Macros
    - HVT Cells
    - Buffering List
    - Net Fanout
    - Level Shifters
    - Isolation Cells
    - Net too long
    - Couple capacitance ratio
  - Timing and constraints
    - Large Skew
    - Divider in Clock Path
    - Total SI Delay
    - SI Delay
    - External Delay
  - Floorplan
    - Fixed Cells
    - Distance from start to end

- Distance of repeater chain
- Detour
- Multiple power domains
- DRVs
  - Max transitions
  - Max capacitance
  - Max fanout

You can customize the type of timing information reported. The *Edit Timing Interpretation* GUI Innovus lets you add, modify, or delete rules you want the tool to check and report.

- Timing bar to analyze delays associated with instances and nets in a path. Use this information to identify issues related to large instance or net delays, repeater chains, paths with large number of buffers, and large macro delays. The small bars superimposed on net delays or within element delays show incremental (longer or shorter) delays due to noise effects:

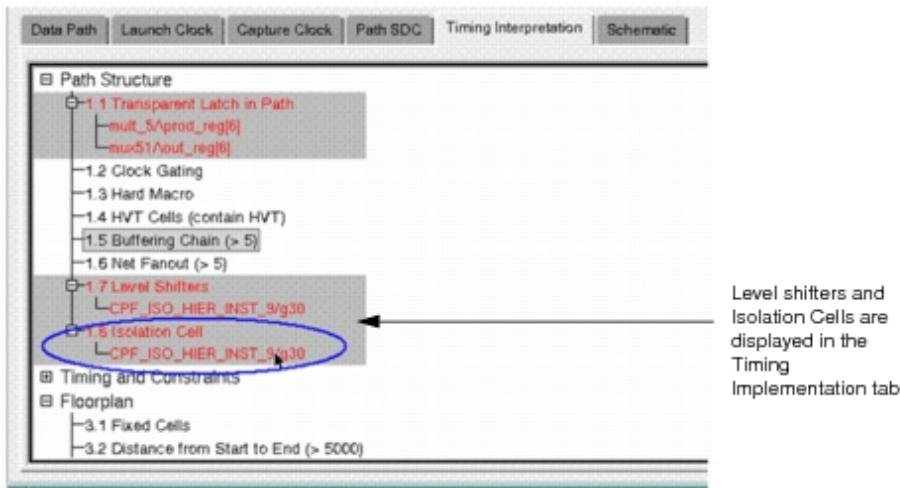


- Hierarchical representation of the path in the Hierarchy View field. This representation of the path-delay shows the traversal of a path through the design hierarchy drawn on the time axis. A longer arrow means that there are more instances on its path. Use this information to see the module where the path is spending more time or to identify inter-partition timing problems.

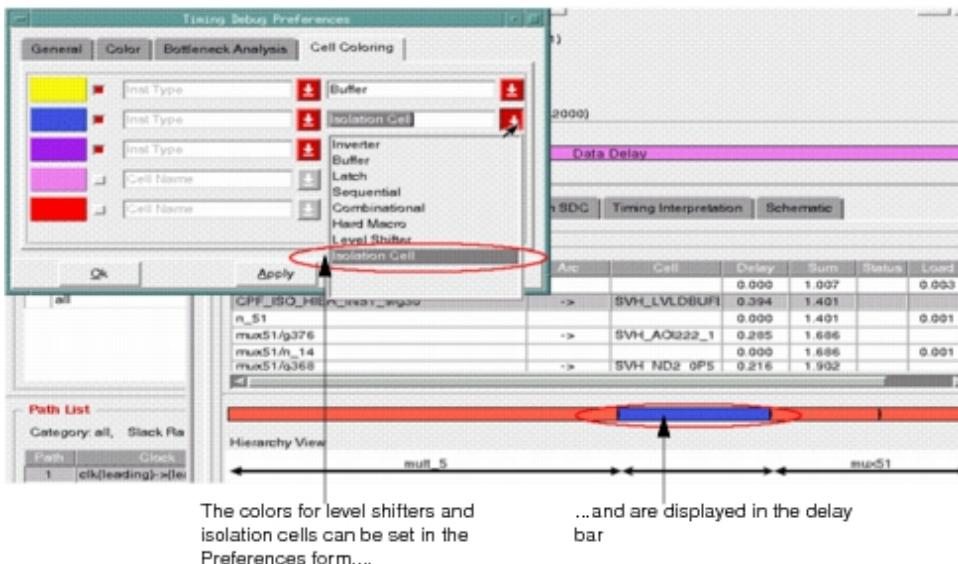
## Viewing Power Domain Information

While debugging critical paths on MSV designs, it is useful to be able to identify power domains and low power cells. The Timing Debug feature displays this information in the following ways:

- The level shifters and isolation cells are listed in the Timing Interpretation tab of the Timing Path Analyzer.



- The delay bar of the Timing Path Analyzer can display the level shifters and isolation cells. You can also use the Preferences form to specify the colors in which the level shifters and isolation cells are displayed.



## Creating Path Categories

After analyzing the paths in the timing report, you identify problems in various paths. Then you create a group of paths such that all paths in that group have the same timing problem and can be fixed at the same time. In timing debug such a group of paths is called a category. In Innovus, you can either define your own category or use predefined categories to group your paths. The categories that you define are then displayed in the *Path Category* field in the *Analysis* tab. The form also displays the paths associated with each category.

## Creating Predefined Categories

There are following predefined categories:

- Basic Path Group  
Creates standard path categories according to basic path groups.

The basic path groups are:

- Register to register
- Input to register
- Register to output
- Input to Output

Registers can be macros, latches, or sequential-celltypes. To create categories by basic path groups, choose the *Analysis* tab - *Analysis* drop-down menu - *Path Group Analysis* option.

- Clock Paths  
Creates categories according to launch clock - capture clock combinations.

Following categories are created:

- Paths with clock fully contained in a single domain, clk1.
- Paths with clocks starting one clock domain and ending at another, clk1->clk2.

To create categories for clock paths, choose the *Analysis* tab - *Analysis* drop-down menu - *Clock Analysis* option.

- Hierarchical Floorplan  
Creates categories according to the hierarchical characteristics of a path. For more information, see [Path Analysis](#) (hierarchical Floorplan)
- View

Creates categories according to the view for which the path was generated.

To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *View Analysis* option.

- **False Paths**

Creates a category with paths defined as False paths.

To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *Critical False Path* option.

- **Bottleneck**

Creates categories based on instances that occur often in critical paths.

To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *Bottleneck Analysis* option.

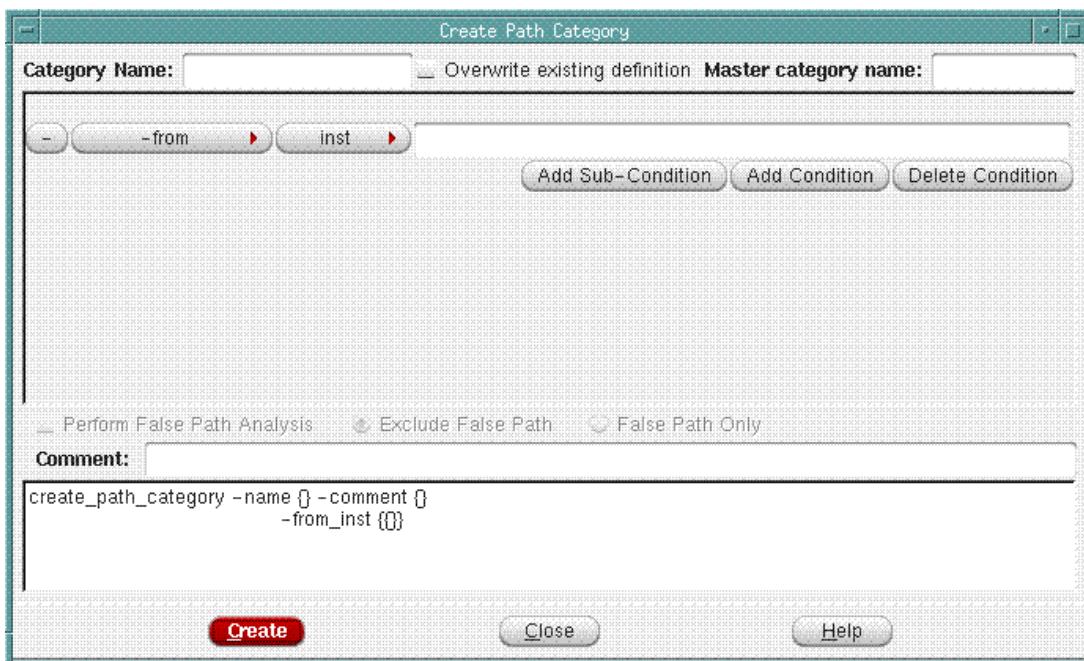
- **DRV Analysis**

Generates or loads a DRV report containing capacitance, transition, or fanout violations. Paths that are affected by the selected DRV types are grouped in a category.

To create or load this report, choose the *Analysis* tab - *Analysis* drop-down menu - *DRV Analysis* option.

## Creating New Categories

To define a new category, use the *Create Path Category* form. The Create Path Category form contains drop-down menus with conditions that you use to define a path category. The conditions are characteristics that a path must have to be added to the named category. You can define multiple conditions that a path must meet to be added to the category.



The category that you create is added in the *Path Category* field in the *Analysis* tab. All paths that meet the conditions set for this category are grouped under the category name. Paths are separated automatically according to MMMC views into different categories, for example:

```
CLOCK1<View_test_mode>
CLOCK2<View_mission_mode>
```

- Double-click on the category name in the *Path Category* field in the *Analysis* tab to display the list of paths in the *Path List* field.

**Note:** You can add a comment in the *Comment* field to record any notes that you would like to include with the category. The comment appears in the category report file.

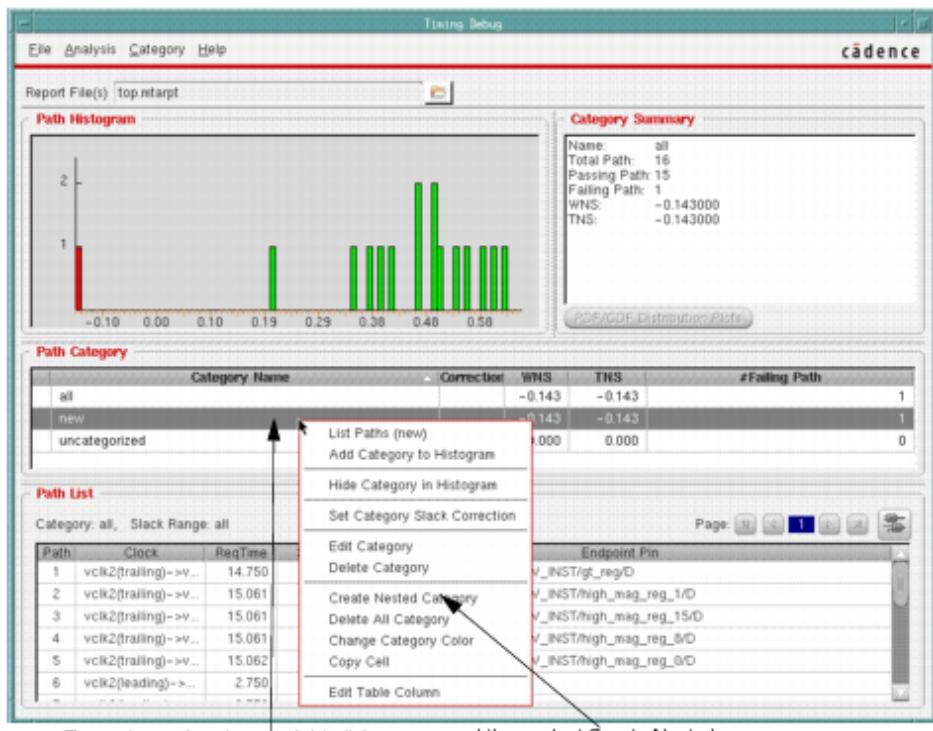
## Creating Sub-Categories

You can create sub-categories based on existing categories. While analyzing a sub-category, global timing debug will traverse the paths in the master category instead of all the paths in the current report.

- [Creating Sub-Categories through the GUI](#)
- [Creating Sub-Categories through Command Line](#)

## Creating Sub-Categories through the GUI

- In the GUI, you can create a sub category as follows:
- Right-click on a path category, and select *Nested Category*.



The *Create Path Category* form is displayed.



- In the *Master category name* field, the name of the category you selected previously is displayed. Create one or more subcategories as explained in [Creating Path Categories](#).

**Note:** You can create nested sub-categories, that is, you can further create sub-categories for a sub-category.

You can also use the Category - Create menu command to bring up the Create Path Category form. In the *Master category name* field, type the name of the category for which you want to create the sub-category, and then create one or more subcategories as explained in [Creating Path Categories](#).

## Creating Sub-Categories through Command Line

Use the *-master* parameter of the [`create\_path\_category`](#) command to create sub-categories. The category created will be a sub-category of the category name specified with the *-master* parameter.

The following other commands also support sub-categories; to run these commands only on the sub-categories of a particular master category, specify the master category name with the *-master* parameter.

- [`analyze\_paths\_by\_basic\_path\_group`](#)
- [`analyze\_paths\_by\_bottleneck`](#)
- [`analyze\_paths\_by\_clock\_domain`](#)
- [`analyze\_paths\_by\_critical\_false\_path`](#)
- [`analyze\_paths\_by\_drv`](#)

- [analyze\\_paths\\_by\\_hierarchy](#)
- [analyze\\_paths\\_by\\_view](#)

**Note:** If the parent category of a sub-category is deleted, the sub-category cannot be edited or changed anymore. However, the sub-category is still displayed in case you want to refer to it.

## Viewing Sub-Categories

The subcategories for a master category are displayed in a hierarchically numbered list below the master category. As an illustration, consider the example shown here:

Path Category	
	Category Name
	all
	master_category1
(1)	nested_category_1a
(2)	nested_category_2
(1)	nested_category_1b
	uncategorized

In this example:

- `master_category1` is the master category
- `nested_category_1a` and `nested_category_1b` are the sub-categories of `master_category1`.  
The prefix (1) is displayed with `nested_category_1a` and `nested_category_1b`.
- `nested_category_2` is the sub-category of `nested_category_1a`.  
The prefix (2) is shown with `nested_category_2`.

## Hiding path categories

To remove a path category from the histogram display, right-click on a path and select *Hide Category*. The category name in the category list is not hidden, but is marked with an "H" as hidden.

## Reporting Path Categories

To generate a report containing information about path categories, use the following options:

- Use the [write category summary](#) command
- Use the Write Category Report File GUI

The text file contains the following information:

- Category name
- Total number of paths
- Number of passing paths
- Number of failing paths
- Worst negative slack
- Total negative slack
- TNS

Sample report:

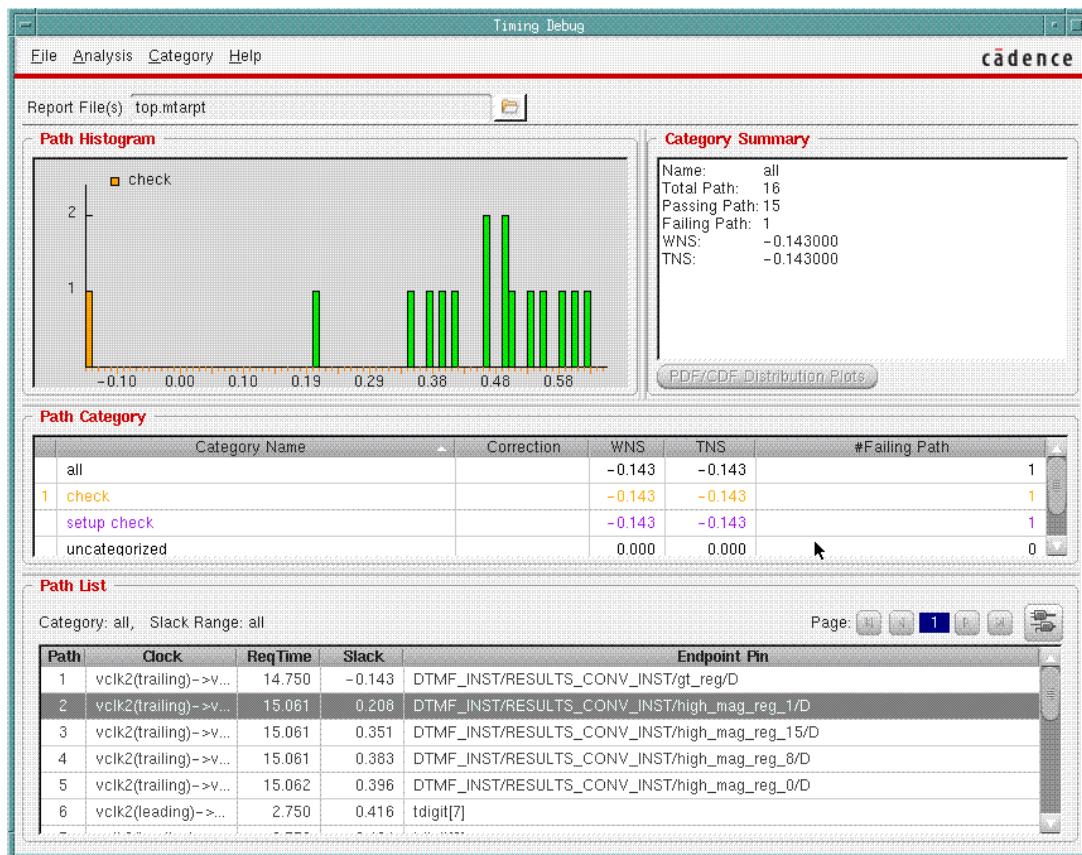
Category name	Total path	Passing Path	Failing path	WNS	TNS
test_clock<view_test_	3869	768	3101	-5.699	-4802.857
100MHz_1.00V>	Clock Domain Analysis				
@->test_clock<view_test	484	55	429	-2.292	-378.432
_100MHz_1.00V>	Clock Domain Analysis				
my_clk-><view_mission	1	2	11	1.931	-1.931
_166MHz_1.08V>	Clock Domain Analysis				
my_clk_2x<view_mission	156	154	2	-0.013	-0.21
_166MHz_1.08V>	Clock Domain Analysis				
cat1875	77	13	64	-3.524	-100.893
Category of paths that cross i_1875 and start with test_clock - need to change the uncertainty value -advised --by Don					

## Using Categories to Analyze Timing Results

You can use the categories that you create to group the timing paths, and display them as histogram in the *Analysis* tab. The *Analysis* tab displays the category details in the *Path Category* field. You can perform the following tasks in the *Analysis* tab to analyze the timing results:

- Double-click on any category to display the details of the paths grouped in that category in the *Path List* field.
- Right-click on the category name and select *Add to Histogram* option. The paths related to the selected category are highlighted in a different color in the histogram. This gives you a visual representation of the number of paths that meet the conditions in that category and can possibly have the same timing problem.

For example, in the following figure the *SetupCheck* category was added to the histogram.

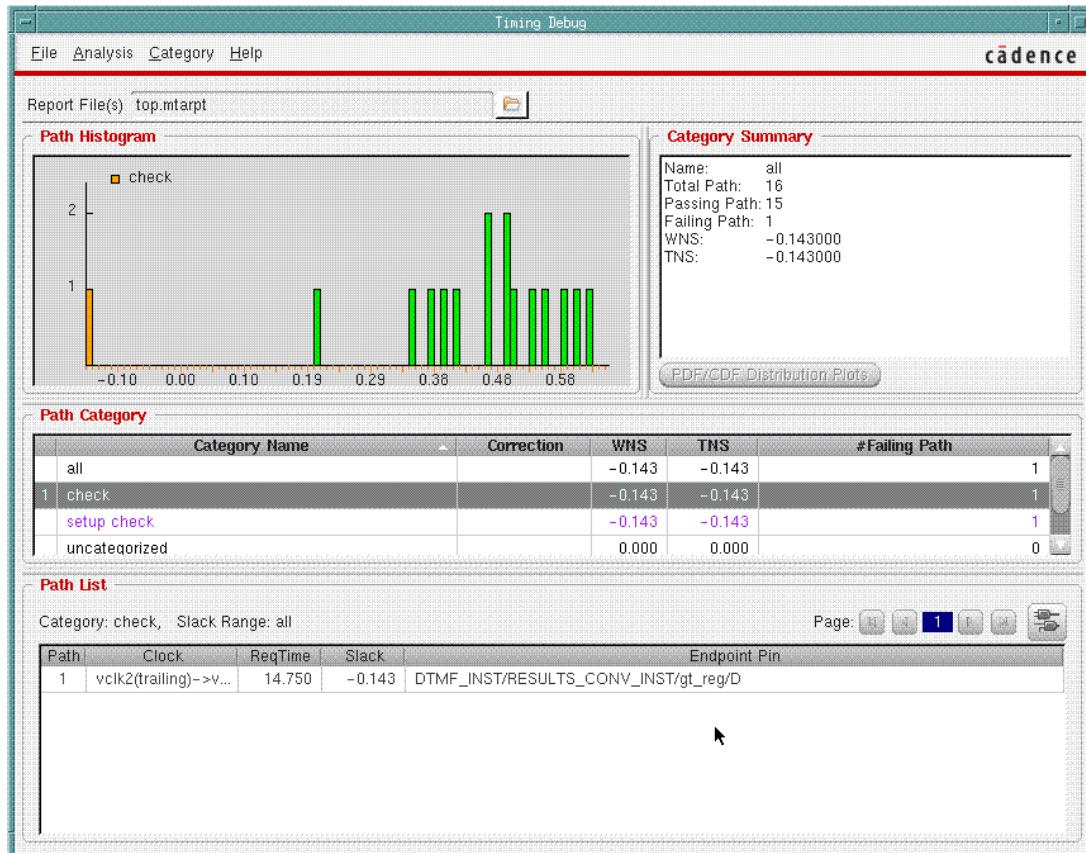


Analyzing the *Analysis* tab gives you information for fixing problems related to larger sets of timing paths. After identifying the problems, you can make the required changes such as modify floorplan, script or SDC files and run timing analysis again for further analysis.

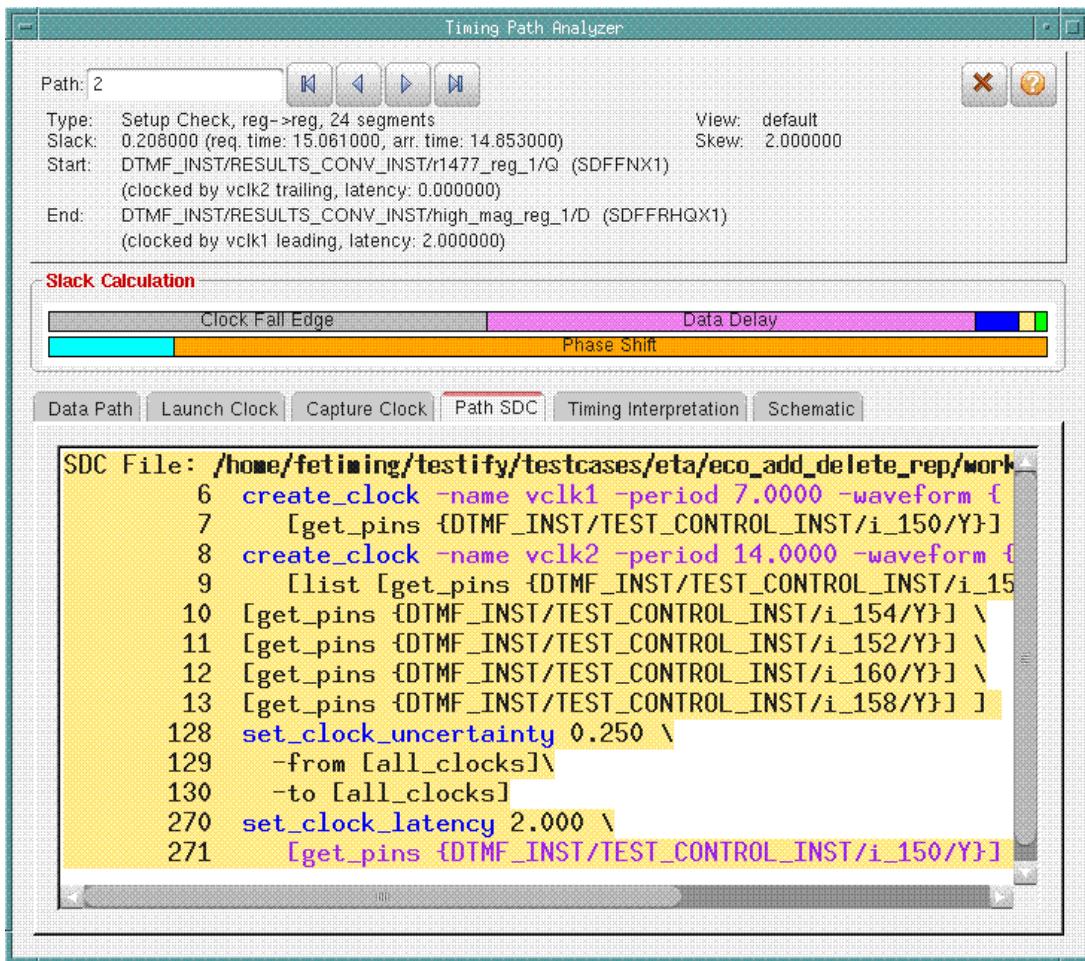
## Analyzing MMMC Categories

Paths are separated automatically according to MMMC views into different categories, for example, the following figure shows two categories based on MMMC views:

1. view\_mission\_140MHz\_1.0V
2. view\_mission\_166MHz\_1.8V



- Right-click on one of `view_mission_140MHz_1.0V` and choose *List Paths*.
- Right-click on one the paths and choose Show Timing Path Analyzer.  
The Timing Path Analyzer is displayed.
- Click on the **Path SDC** tab to display the SDCs:



Note that the SDCs relative to mode mission\_140MHz that produced the path are highlighted.

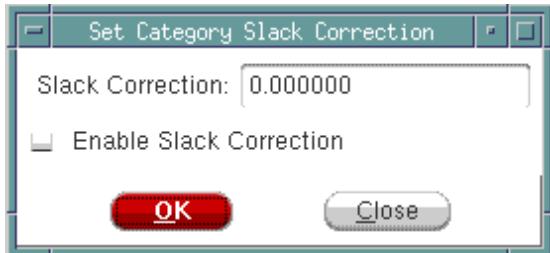
## Manual Slack Correction of Categories

Use the Set Category Slack Correction form to specify the estimated slack correction for the selected category of paths. A slack correction that you apply to a category modifies all the paths in that category. If a path belongs to several categories, all the correction from the categories are added. The worst negative slack and total negative slack values of a category can be affected by the correction applied to another category.

Once you enable the slack correction, the histogram is updated to reflect the slack correction. An asterisk (\*) is added next to the slack value of paths that belong to this category in the *Path List* field in the *Analysis* tab. Paths are reordered based on new specified slack. This allows you to filter out the paths that can be fixed and work on the remaining paths.

To access the Set Category Slack Correction form complete the following steps:

- Click on the Analysis Tab in the main Innovus window.
- Right click on the category name in the *Path Category* field.
- Choose the *Set Category Slack Correction* option.



To disable the set slack correction value:

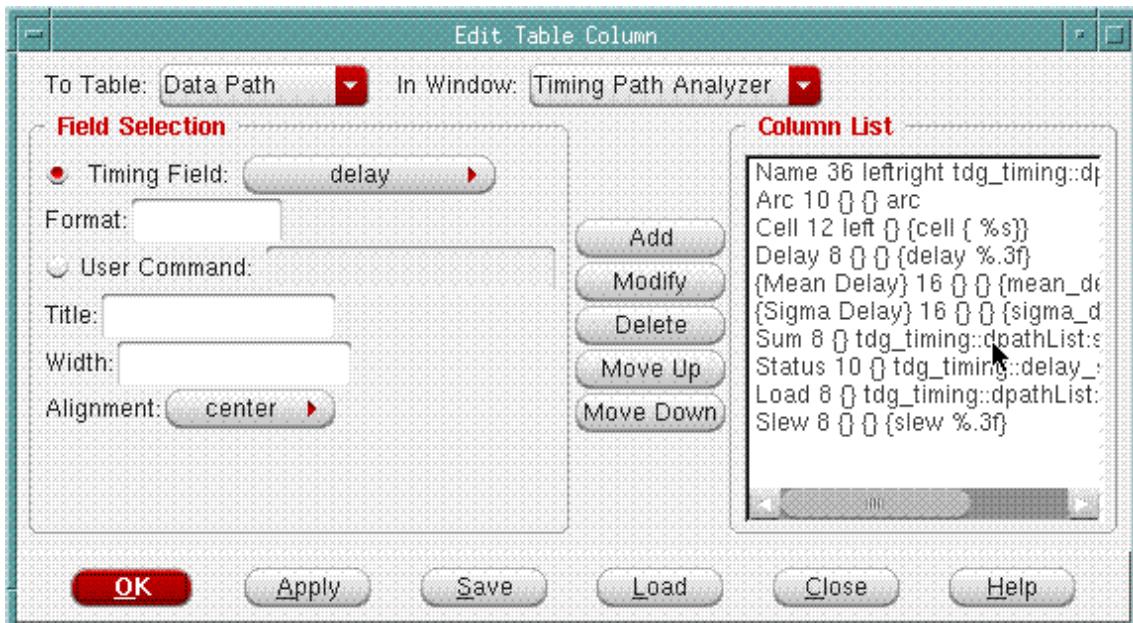
- Right click on the category name in the *Path Category* field.
- Choose the *Deactive Category Slack correction* option.

## Editing Table Columns

You can customize the dimensions and contents of table columns to suit your needs.

- To begin customizing a table, click on the *Analysis* tab, then right-click on a path. A drop-down menu is displayed.
- Select *Edit Table Column*.

The Edit Table Column form is displayed.



- Choose the timing window that contains the table to want to customize.
- Choose the table whose columns you want to customize. The selections change according to the timing window you choose.
- Choose a column item or specify a command.

For commands, specify the procedure you want to use to determine the information you want to include in the column. Source the file containing the procedure before you specify the procedure here.

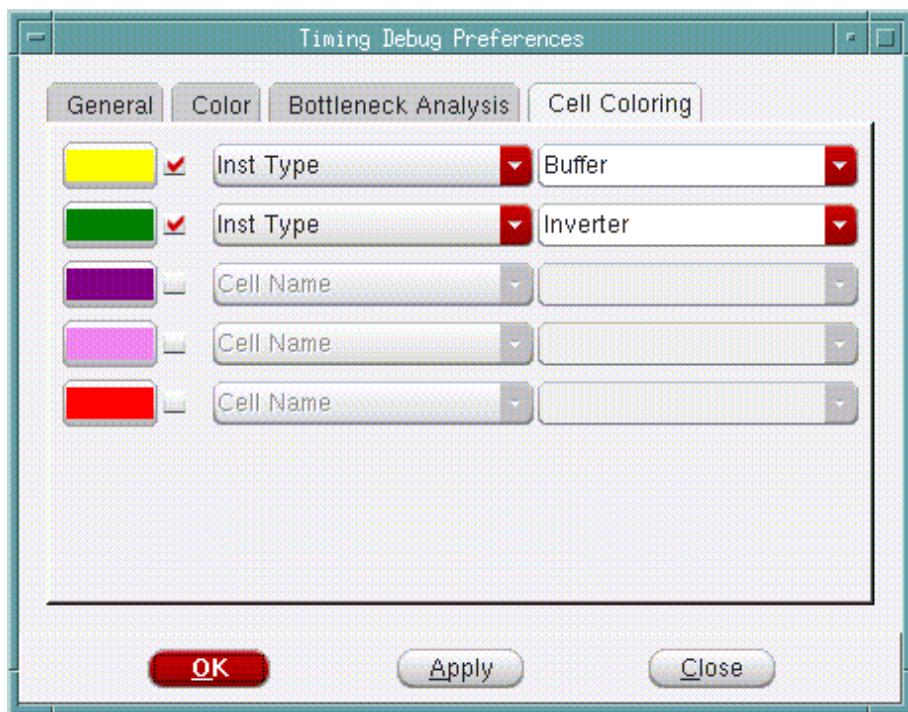
For example:

```
# Combine fedge (from edge) and tedge (to edge) #information into a single field
proc my_get_edge {id var} {
    upvar #0 $var p
    ($ added before p)
    if {p(type) == "inst"} {
        return "$p(fedge) -> $p(tedge)"
    } elseif {$p(type) == "port" } {
        return $p(fedge)
    } else {
        return ""
    }
}
```

- Build the column list.
  - *Add* adds a column to the column list.
  - *Modify* let you modify characteristics. Click on a column in the column list. Edit the information, then click *Modify*.
  - *Delete* removes a column from the column list.
  - *Move Up* moves a column up in the column list. This effectively moves a column to the left in the table.
  - *Move Down* moves a column down in the column list. This effectively moves a column to the right table.
- (Optional) Click *Load*. The opens the GTD (Global Timing Debug) Preferences form. Specify a file name.

## Cell Coloring

Use the *Cell Coloring* page of the Timing Debug Preferences form to choose colors for specific cells in the delay bar.



When you assign colors, this same colors will be restored when you start a new session.

In the *Cell Name Selection Elements* field for each color, you can choose whether you are providing one of the following:

- Cell name
- Instance
- Procedure that you have defined

The procedure is invoked with the full instance name as the argument. You must source the file containing the procedure before you use this feature.

For example:

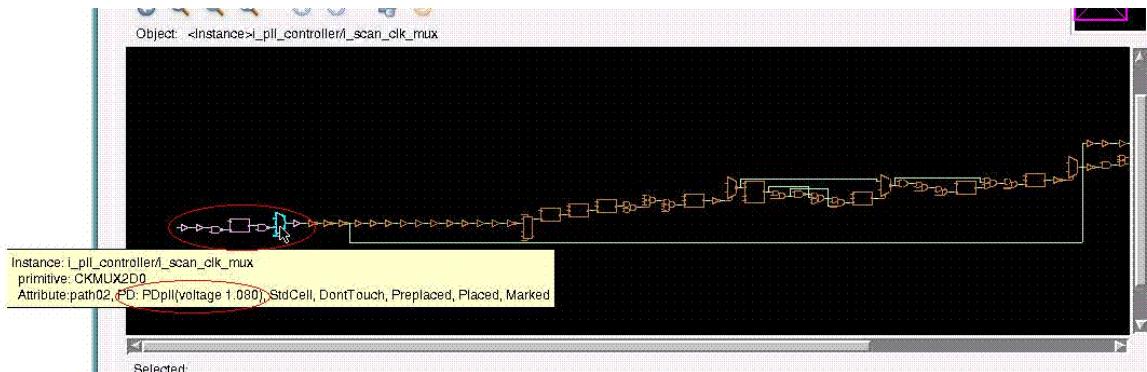
```
# Colors when the instance name contains "core/block1"
proc belongs_to_block1 {inst_name} {
    if [regexp {core/block1} $inst_name] {
        return 1
    } else {
        return 0
    }
}
```

## Viewing Schematics

The Critical Path Schematic Viewer displays the gate-level schematic view of the critical path. To display the Schematic Viewer, click on the schematics icon in the *Path List* field of the *Analysis* tab. You can display additional paths in the Schematic Viewer by using the middle mouse button to drag the path from path list to Schematic Viewer.

You can also display the Schematics by selecting the Schematics tab in the *Timing Path Analyzer* form. The form is displayed when you double-click on a critical path in the Path List in the *Analysis* tab.

On displaying the Schematic Viewer, you can see the power instance colored and the power domain information displayed in a popup message box as well as in terminal.



You can perform the following tasks in the Module Schematic Viewer:

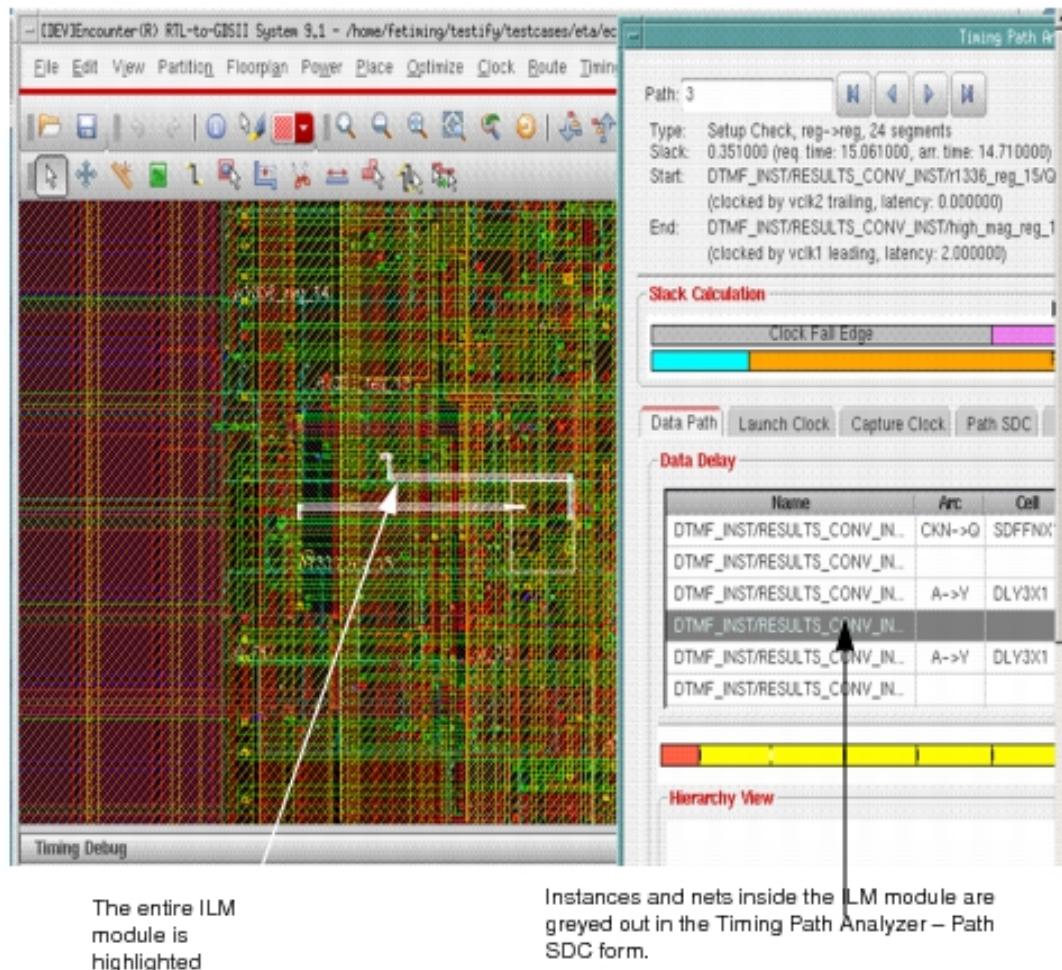
- View the Gate-level design elements.
- Select an element in the schematic.
  - Click on an object in the schematic to select and highlight it. When you move the cursor to an object, the object type and name of the object appear in the information box.
- Scroll over an object to display the object type and name of the object in the *Object* field.
- Cross-probe between the Schematics window and *Path List* field.
  - Select a path and left-click on the Schematics button above the Path List Table. (This is the button at the far-right side, just above the table).
  - To show multiple paths, select another path, and drag and drop it to the Schematics window.
- Use the menu options provided in the Schematic Viewer. To access the menu options, you can either click on the menu bar or right-click on an object in the schematic. You can use the menu options to perform the following tasks:

- Manipulate schematic views of fan-in and fan-out cones.
- Trace connectivity between drivers, objects, and loads.
- Move between different levels of instance views.

## Running Timing Debug with Interface Logic Models

You can use the timing debug feature with designs containing Interface Logic Models (ILMs).

- The Timing Path Analyzer - Path SDC form displays ILM SDCs rather than the original SDCs.
- The software highlights the entire ILM module instead of the instances and nets inside the ILM. The instances and the nets inside the ILMs are greyed out in the Timing Path Analyzer - Path SDC form.



# Power and Rail Analysis

- [Overview](#)
- [Early Rail Analysis](#)
  - [Early Rail Analysis Key Features](#)
  - [Setting up and Running Early Rail Analysis](#)
  - [Viewing Early Rail Analysis Results](#)
- [Signoff-Rail Analysis](#)
- [TCL Command](#)
- [Innovus and Voltus Menu Differences](#)

## Overview

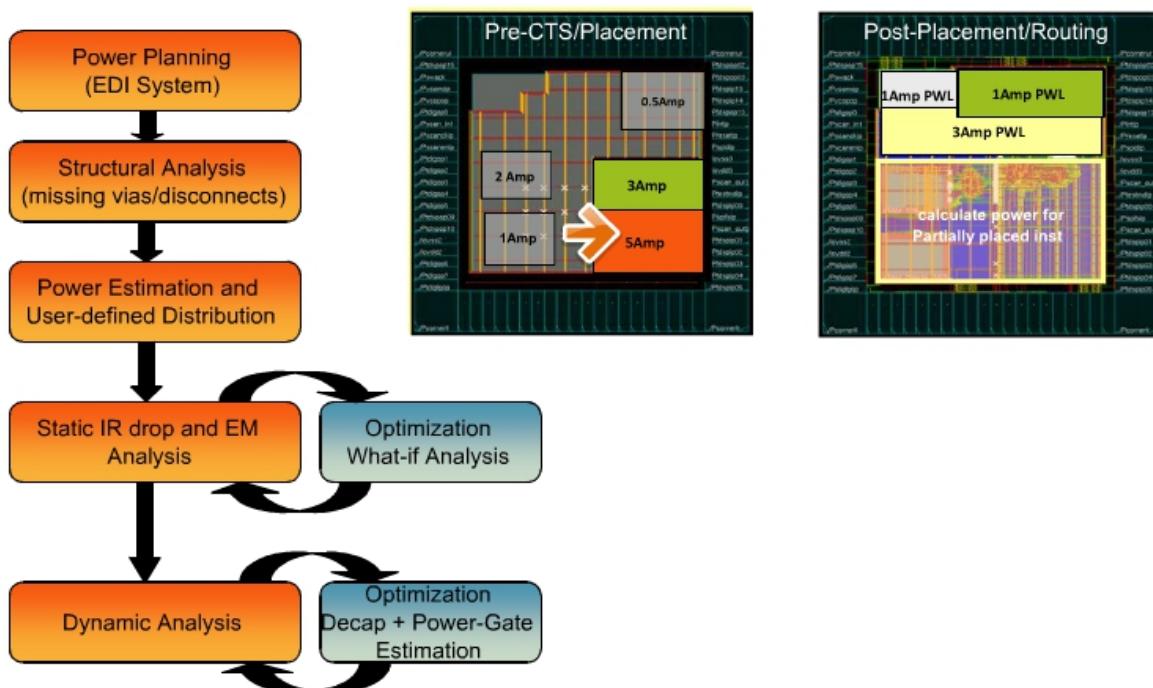
Voltus Power Integrity Solution sign-off power and rail analysis engines are fully integrated in Innovus Implementation System (Innovus). The TCL and GUI use-models are identical in stand-alone Voltus and its integration in Innovus, allowing a smooth transition from early power planning to sign-off power analysis. The power and rail analysis in Innovus is available under the *Power* menu (Voltus is under *Power & Rail* menu). These menus are arranged differently between the two products (See [Innovus and Voltus Menu Differences](#)).

The ERA feature provides rail analysis at the early stage of design with the same use model as Signoff Rail Analysis. Static ERA can also be run with Innovus base license. ERA is not intended for sign-off; if you want to include custom power-grid views or do dynamic analysis, then a Voltus license is required. ERA is intended for early stage analysis and can run on a power-grid floorplan before placement and routing.

## Early Rail Analysis

**⚠ Starting with the 14.2 release, the Early Rail Analysis (ERA) feature inside Innovus works off a new use model using the `set_rail_analysis_mode` and `analyze_rail` commands. The new flow utilizes the same power-grid extraction engine used in Voltus to have seamless transition between early and signoff power-grid analysis. The flow also supports advanced features such as what-if wires, what-if vias, and flexible create current region, along with the ERA flow parameters of the `set_rail_analysis_mode` command. Refer to *Voltus User Guide* for flow details related to the advanced features. The old ERA command in Innovus, `analyze_early_rail`, is obsolete. To ensure compatibility with future releases, use the `set_rail_analysis_mode` command to set up ERA.**

ERA has the ability to analyze power-grid integrity early in the floorplanning stage, after placement, as well as postrouting. It helps fix power-grid problems early in the flow, rather than waiting for when the layout is mostly done and the problems are much more difficult to correct. ERA will take whatever blocks, macros, standard cells, and routing that is available to help improve the accuracy of early rail analysis. The following diagram illustrates the ERA flow:



The following steps describe the ERA flow:

- **Grid Completion:** The ERA engine checks if the followpin routing has been done or not in the design; if not, it creates virtual followpin automatically and drops required virtual vias. Missing

virtual vias between stripes are also dropped by default at this stage.

- **Power Estimation and User-Defined Distribution:** In the ERA static flow, you can specify total power and the ERA power engine will distribute that internally to all placed and unplaced instances in the design. For unplaced instances, current regions are created. You can also selectively assign a specific power value to placed macros, cell or instances using an ASCII file. User-specified power can be enforced for unplaced instances using an explicit current regions file.
- **Static IRdrop and EM Analysis:** Run static IRdrop and electromigration analysis.
- **Dynamic Analysis:** Run dynamic IRdrop analysis. For this, you need to specify explicit dynamic current region file or the dynamic instance current file.

ERA uses Voltus extractor and rail analysis engines. They can be used during power-grid prototyping to analyze power, IRdrop and power-grid integrity. This section includes:

- Early Rail Analysis Key Features
- Setting up and Running Early Rail Analysis

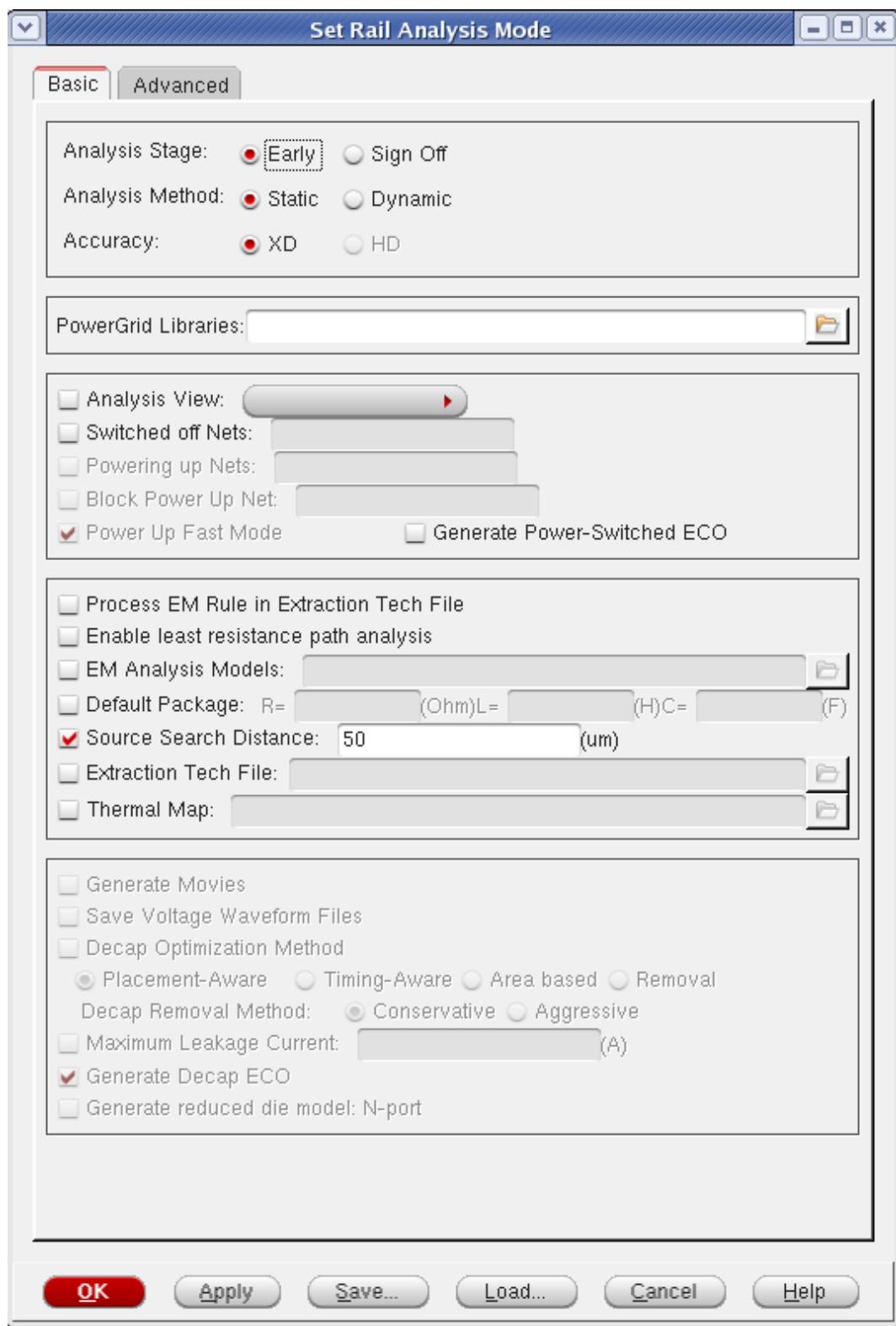
## Early Rail Analysis Key Features

- Static and Dynamic Rail analysis during the floorplanning stage using grid based interactive current specification use-model. Static and Dynamic Rail Analysis requires the VTS-L/VTS-XL license. Static Power Analysis can be run using the Innovus license.
- Interactive current specification use-model to enable static and dynamic rail analysis at the floor-planning stage
- Power and rail analysis during the placement stage using ERA driven virtual follow pin routing and virtual via for grid completion.
- Automatic current region generation accounting for unplaced instance in the design, and automatic distribution of power among placed instances in the design.
- Support of user-specified explicit power value at macro, cell and instance level in an ASCII file format, hence, enabling flexible power distribution.
- PGV library is optional. If not provided, it is generated on the fly using the specified technology file.
- Static Power and Rail Analysis without PGV can also be run using the Innovus license.
- Power and rail analysis on a placed and routed database.
- Early power-switch analysis to refine power-switch placement.
- Support for multi-CPU in static/dynamic power and rail analysis, and static and dynamic power and rail analysis. For information on how to set up multi-CPU analysis, see the "Distributed Processing" chapter in *Voltus User Guide*.
- Support for the unplaced flow during static and dynamic analysis.
- What-if shape analysis to guide power-grid optimization.
- Support for native power-up analysis.

## Setting up and Running Early Rail Analysis

To setup and run early rail analysis, perform the following steps.

1. Select *Power - Rail Analysis - Setup Rail Analysis Mode* menu. The following form appears:



**Note:** The content of the two tabs of this form change depending on the selection of the analysis

method.

2. Set Analysis Stage as *Early* .
3. Set Analysis Method as *Static* or *Dynamic*.

By default, *XD* is selected as the Accuracy mode. *HD* is disabled for early rail analysis.

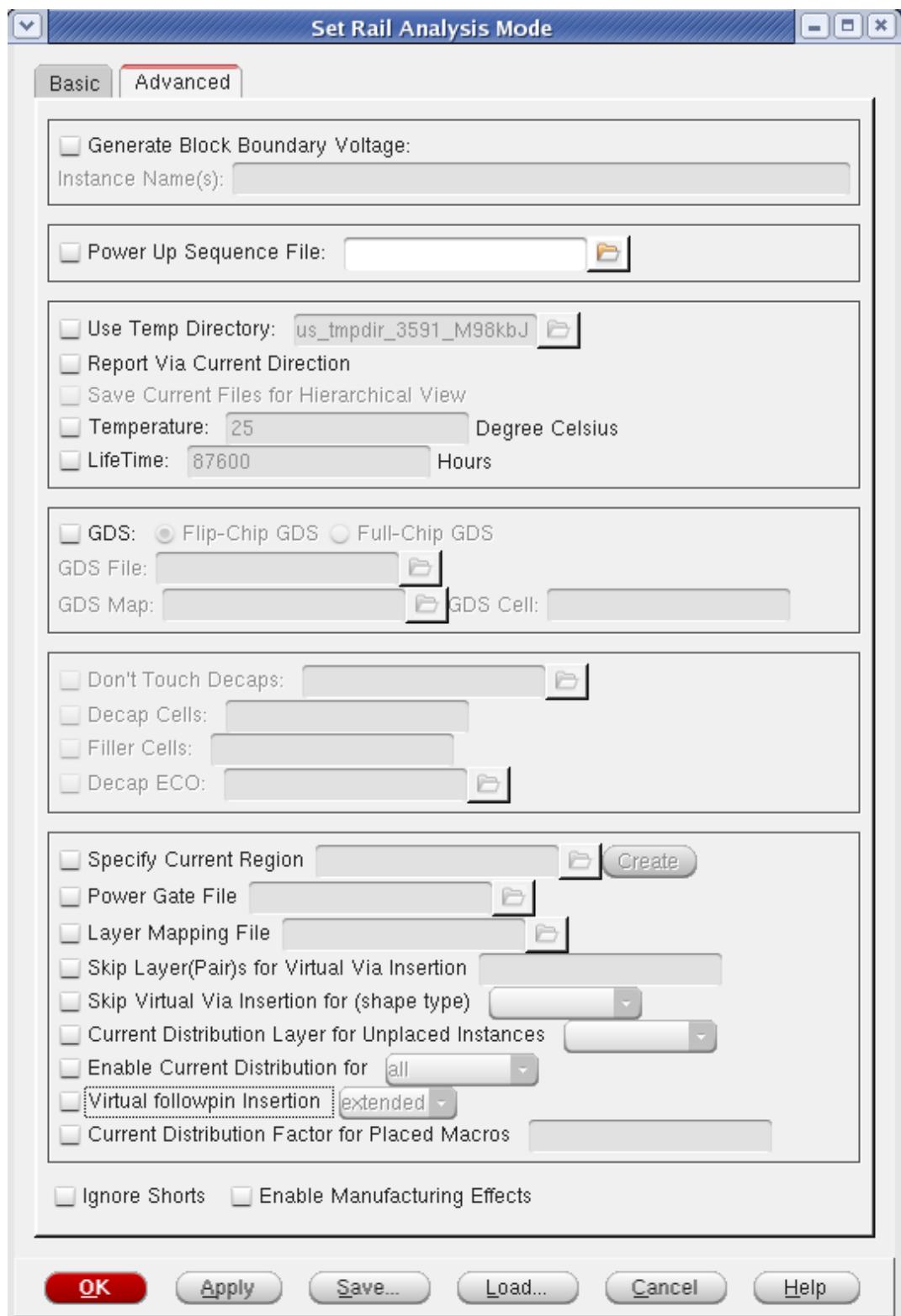
4. Specify *Power-Grid Libraries* or *Extraction Tech File*.

See the "Power-Grid Library Generation" chapter of the *Voltus User Guide* for details on generating power-grid libraries.

5. If CPF is loaded, select *Analysis View*.
6. If it is a power gated design, optionally specify *Switched off Nets*.
7. For optional Electromigration analysis, specify either *EM Analysis Models* or *Process EM rules in Extraction Tech File*.

**Note:** If you do not specify an Electromigration model file, the software will select Electromigration models from the QRC technology file. You can use the command `set_rail_analysis_mode -em_models file` to override these settings.

8. Select the *Advanced* tab of the *Set Rail Analysis Mode* form. The following form appears:



9. You can specify one or more of the following advanced options:

- *Generate Boundary Voltage File* if using hierarchical view for the block
  - *GDS for Flip-chip RDL or Full-chip GDS* if needed.
- Note:** ERA would work only when virtual connections between GDS and DEF are not required.
- *Specify Current Region* to specify a file that includes a list of regions and the amount of current to be distributed within them for the power-grid. When you select this checkbox, the *Create* button gets enabled that lets you create current regions for Static and Dynamic analysis. See "[Creating Regions for Static and Dynamic Analysis](#)".
  - *Power Gate File* to specify a power-gate file to analyze nets which are power-gated. The power-gate file syntax is as follows:

```
CELL cellname SUPPLY unswitched_net_name SWITCHED switched_net_name  
RON r_value IDSAT idsat_value ILEAK ileak_value
```

CELL *cellname* - the name of the cell.

SUPPLY *unswitched\_net\_name* - the name of the unswitched power net. This is the pin that the leakage current is attached to if the power gate is in the off state.

SWITCHED *switched\_net\_name* - the name of the switched power net.

RON *r\_value* - the on-resistance value in ohms.

IDSAT *idsat\_value* - the value of the saturation current in millamps.

ILEAK *ileak\_value* - the value of the leakage current in millamps.

If this file is specified, it is expected that the power-switches are fully connected to the appropriate alwaysOn and switched power nets. ERA will extract the power-grid and perform steady state IRdrop analysis. For information about power gate analysis, see "[Power Gate \(Switch\) Analysis](#)" in the *Voltus User Guide*.

- *Layer Mapping File* to specify a layer map file to generate a techonly view. If a layer map file is not provided, it would be automatically inferred by the tool.
- *Skip Layer (Pair)s for Virtual Via Insertion* to skip via insertion between stripes and non-stripes, on the specified LEF layer pairs.
- *Skip Virtual Via Insertion for Shape Type* to skip a given via type. By default, ERA generates all virtual via layer types.
  - whatif** - vias are virtual vias that have connectivity to user-defined what if shapes.
  - def** - vias are virtual vias between two metal shapes defined in DEF.
  - all** - will skip all virtual via generation.
- *Current Distribution Layer for Unplaced Instances* to specify the layer name for distributing unplaced current in the early rail analysis mode.
- *Enable Current Distribution for* to control the behavior of era current distribution.

`set_power_data` area based power, or `current_region_file` need to be specified for ERA current distribution to work. Placed instances without uti or ascii based power can also be considered for era current distribution.

**Unplaced** : Enable current distribution only for unplaced instances. If `set_power_data` area based power is specified, `-era_current_distribution_layer` will be required.

**Placed** : Enable current distribution for placed instances without any power specified.

**All** : Both unplaced and placed instances without power specified; will have ERA current.

**None** : Disable ERA current distribution.

- *Virtual Followpin Insertion* to generate virtual followpins. The `extended` followpins will create followpins that extend from one stripe to another. The `standard` followpins may extend to previous stripe but does not reach the next stripe.
- *Current Distribution Factor for Placed Macros* to control the current distribution factors for the placed instances, hence power allocated for area-based power calculation. For example, if you specify 0.5, the software will assume all placed instances to be 50% of its actual size and distribute current accordingly.
- *Enable Manufacturing Effects* to honor DFM effects.

## 1. Creating Regions for Static and Dynamic Analysis

You can create regions for Static and Dynamic analysis.

- When you set Analysis Method as *Static*, the following form appears:



- When you set Analysis Method as *Dynamic*, the following form appears:



You can do the following:

1. *Draw Current Regions* to create a *Current Region List*.

Use this when you have an area that has not been placed, but you would like to have its power consumption influence the overall grid. You can specify the coordinates of the region, the layer, and the current.

**Power Domain** lets you specify a power domain based current region. You can use the *Power Domain* field to select a power domain name and click *Get Coordinates* to automatically get the coordinates of the power domain. When the power domain is selected, the label name of the current region will be the power domain name and the boundary box coordinates will be the power domain boundary, and you cannot modify these fields.

**Label** specifies a name for the region. If not specified, Voltus will provide a name (region1, region2...). The **Draw** button lets you draw a window where you want the current to be applied. If you click *Draw*, and then select a box in the main window, the coordinate of this box will be automatically populated in x1 y1 x2 y2. Use the left mouse button to draw the box.

x1, y1, x2, and y2 specifies a rectangular region that the current will be distributed within.

**Rectilinear** specifies the rectilinear current region that the current will be distributed within. You can specify a rectilinear box to add a current region. The rectilinear box enables you to specify multiple x,y points to add current regions in the areas that are not rectangular in shape. To draw the rectilinear box, use the left mouse button and select multiple points. Use the 'Esc' key to the last point of the rectilinear box to finish and capture the box co-ordinates.

**Note:** In the static mode, ERA splits the rectilinear region into several rectangular regions and distributes current to the rectangular regions based on the area.  
Current in rectangular region = Area of the rectangle / Area of the rectilinear

**Layer** specifies the metal layer that the current sink will be placed on.

**Static Current** specifies the current to be attached in the window.

For dynamic current regions, the PWL waveform is specified in time (ns) and current (mA) pairs. In addition to the dynamic current, you can specify loading capacitance and cell intrinsic capacitance which impacts dynamic IRdrop. If this information is not available for the region, you can click the **Estimate** button to populate these values automatically. These capacitance values are derived by calculating loading capacitance of the design using wire-load models and using percentage ratio of loading capacitance to estimate cell intrinsic capacitance in the region.

**Note:** The effect of loading capacitance depends upon on-resistance through which it is connected to the global power-grid. Generally, this on-resistance value is high and limits the effectiveness of loading capacitance. Therefore, specification of loading capacitance is optional and when specified, you must also specify the on-resistance value.

2. Click the *Add* button to add the region to the *Current Region List*. The *Delete* button will delete a selected item on the list. If you click *View* after selecting an item in the list, the selected region will be displayed in the main window.

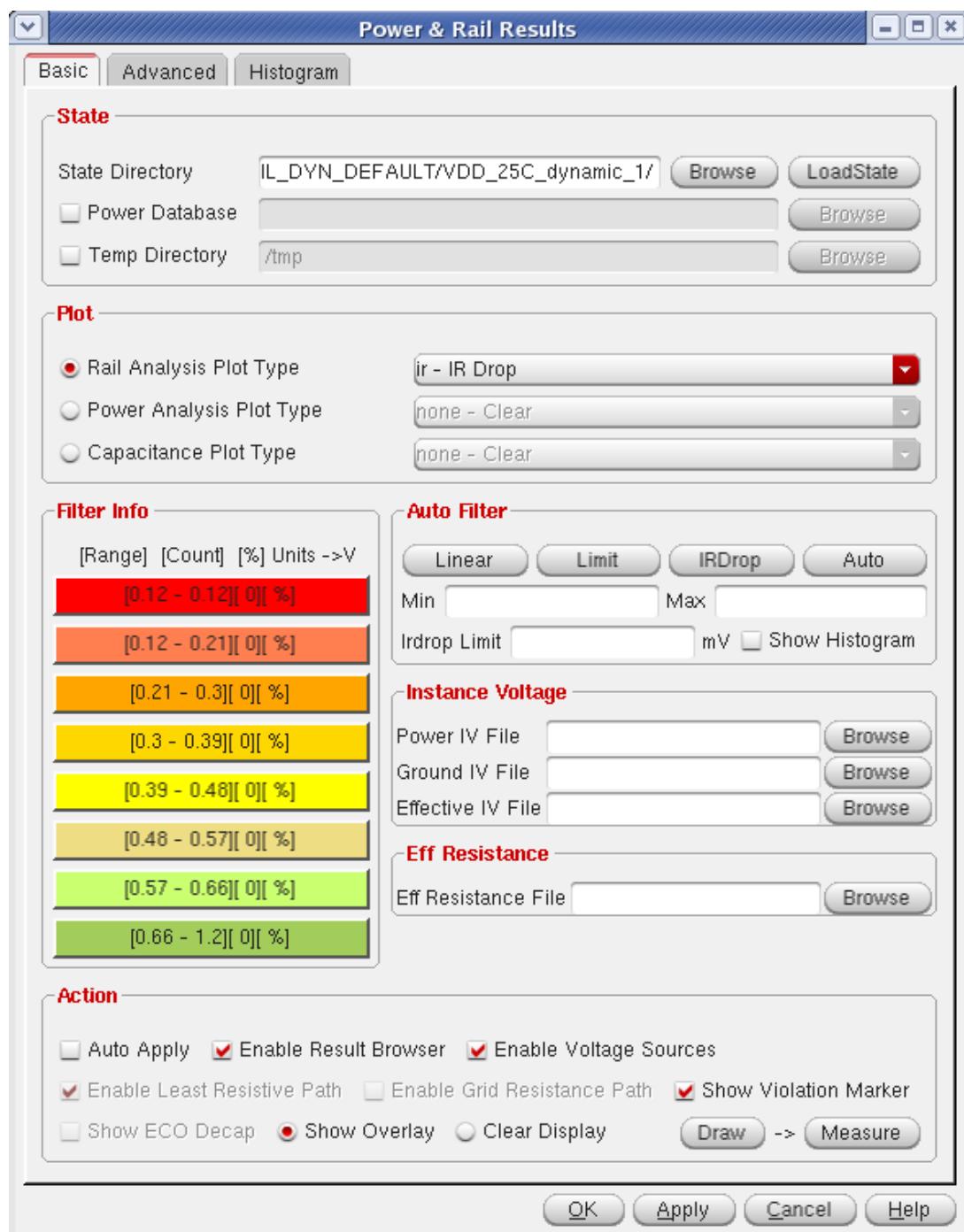
3. Click *OK* or *Apply* and the early rail analysis will be run.

Upon successful completion of the analysis, the Power & Rail Results form appears. In the *Basic* tab, the *State Directory* field is automatically filled with the most recent analysis run. The automatic run naming convention: is vss\_25C\_avg\_2 (VSS rail analysis, at 25 degrees Celsius, average or static power, run number 2). Running VSS analysis again will increment 2 to 3.

## Viewing Early Rail Analysis Results

Selecting *Power - Report - Power & Rail Result* menu item will bring up the following form:

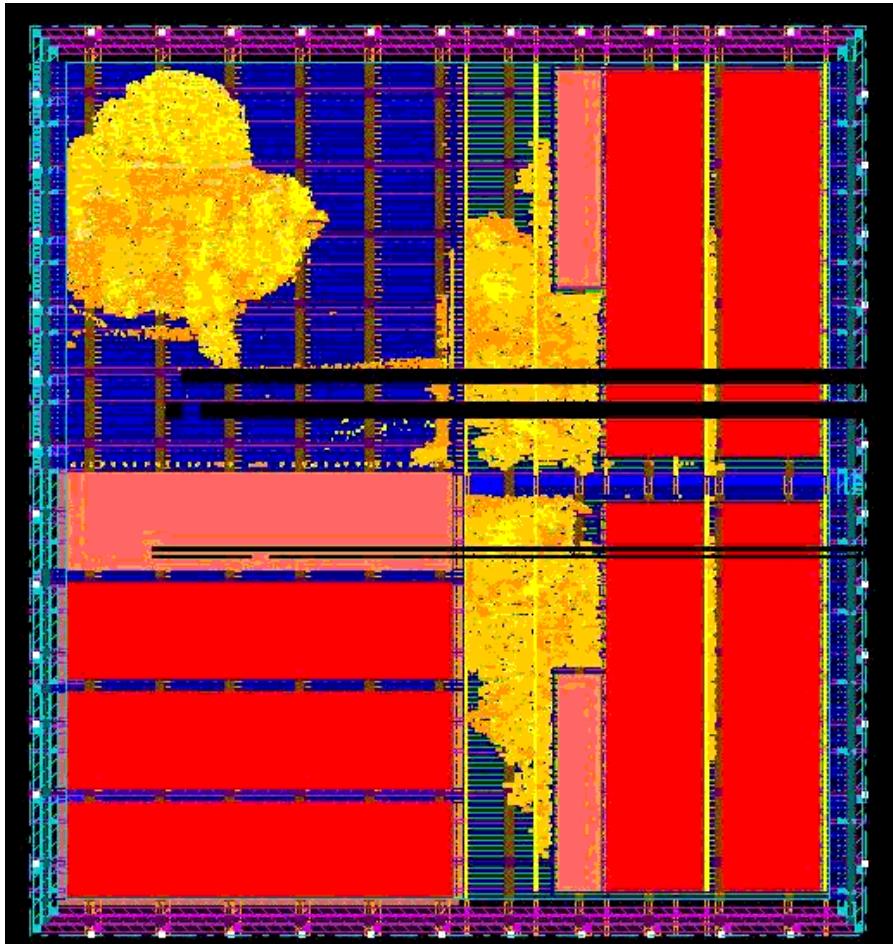
**Figure 37-8 Power & Rail Results**



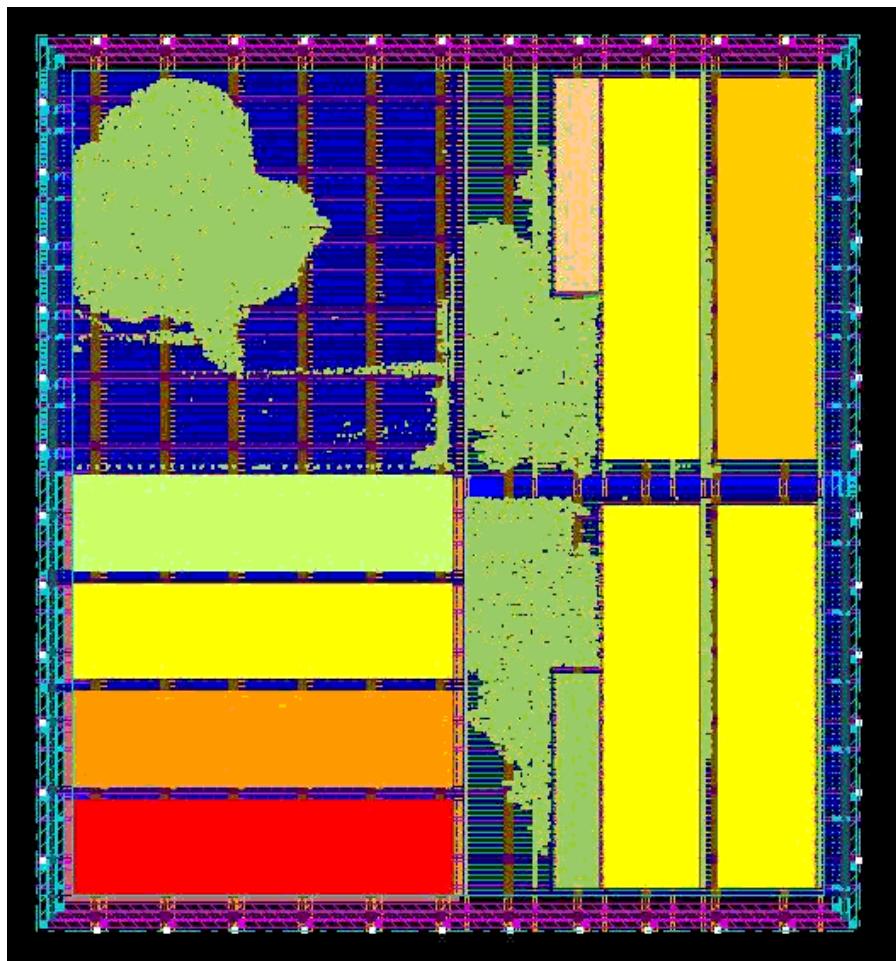
You can use this form to browse to other analysis runs and load them as well to view early analysis results and compare runs. You can specify the type of plot (Rail Analysis, Power Analysis, or Capacitance) and then select the specific plot type. An instance power (ip), load capacitance (load), and irDrop (ir) plot are shown in [Instance Power Plot](#), [Load Capacitance Plot](#), and [irDrop Plot](#), respectively.

For Early Rail Analysis, the viewing of Power & Rail Results is the same as that used for Sign-off Analysis. For additional information on viewing the plots, see "[Static Power Analysis Plots](#)" and "[Static Rail Analysis plotting steps](#)" in the *Voltus User Guide*.

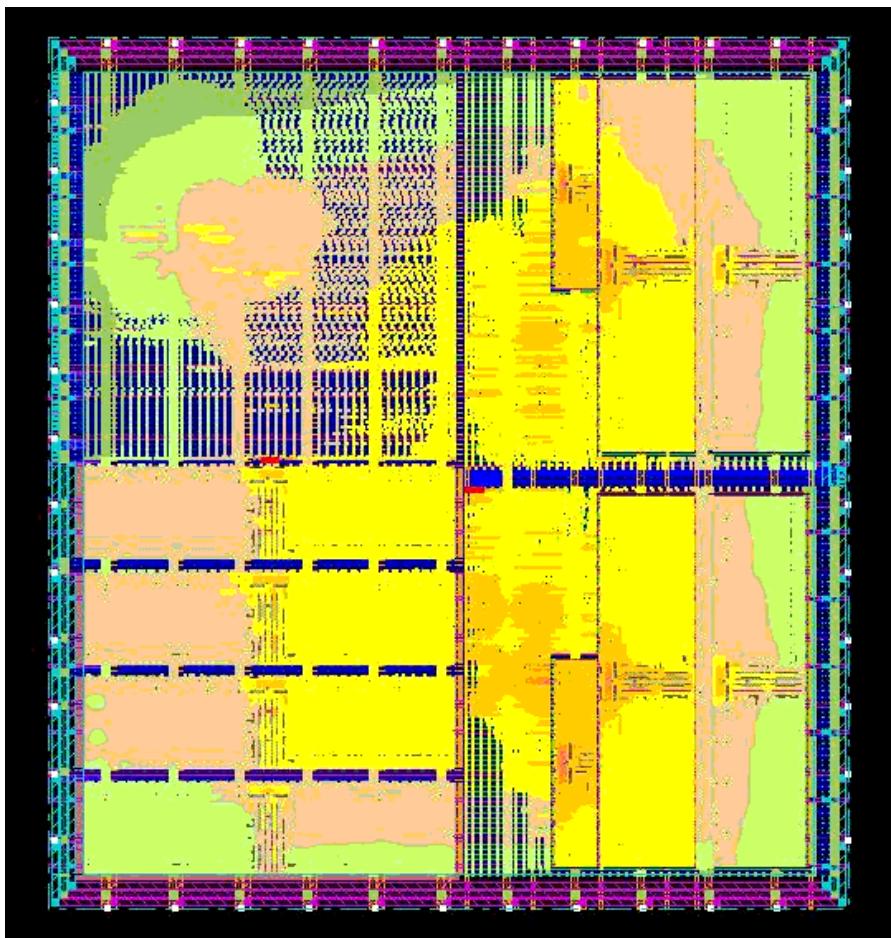
**Figure 37-9 Instance Power Plot**



**Figure 37-10 Load Capacitance Plot**



**Figure 37-11** irDrop Plot



## Signoff-Rail Analysis

For details on running Signoff Power and Rail Analysis within Innovus, see *Voltus User Guide* chapters 5-12.

## TCL Command

An example TCL command for Floorplan stage design grid analysis with current regions is as follows:

```
read_lef -lef design/full.lef
read_verilog design/test.v.gz
set_top_module test
read_def design/full.def
set_rail_analysis_mode -method era_static -accuracy xd -extraction_tech_file
design/tech.tch -era_current_region_file design/current_region
```

```
set_pg_nets -net VDD -voltage 1.1 -threshold 1.067
set_power_pads -net VDD -format xy -file pads/vdd.pp
analyze_rail -type net -results_directory early_vdd VDD
```

An example TCL command for Power Gate Design with area based power distribution is as follows:

```
read_design -physical_data design.dat CHIP
set_pg_nets -net VDD -voltage 1.1 -threshold 0.9
set_power_pads -net VDD -format xy -file design/vdd.pad
set_power_data -bias_voltage 1.2 -power 1.2 -format area
set_rail_analysis_mode -method era_static -accuracy xd -extraction_tech_file
design/tech.tech -era_power_gate_file design/power_gate_file
analyze_rail -type net VDD
```

An example TCL command for MSMV dynamic analysis is as follows:

```
read_verilog design/test.v.gz
set_top_module test
read_def design/full.def
set_pg_nets -net VSS -voltage 0 -threshold 0.18
set_pg_nets -net VDDm -voltage 0.84 -threshold 0.756
set_pg_nets -net VDD -voltage 0.84 -threshold 0.756
set_power_pads -net VDD -format xy -file design/vdd.pp
set_power_pads -net VDDm -format xy -file design/vddm.pp
set_power_pads -net VSS -format xy -file design/vss.pp
set_power_data -format current { instance_current_files/dynamic_VSS.ptiavg
instance_current_files/dynamic_VDD.ptiavg instance_current_files/dynamic_VDDm.ptiavg
instance_current_files/dynamic_VDDlu.ptiavg instance_current_files/dynamic_VDDau.ptiavg}
set_rail_analysis_mode -method era_dynamic -accuracy xd -power_grid_library {
stdcells_accurate/accurate_stdcells.cl lpcells_accurate/accurate_stdcells.cl
memories_accurate/MEM.cl } -off_rails VDDau
set_rail_analysis_domain -name PD -pwnets {VDD VDDm} -gndnets VSS
analyze_rail -type domain PD
```

## Innovus and Voltus Menu Differences

Form	Innovus Menu	Voltus Menu
<i>Set Power Analysis Mode</i>	<i>Power - Power Analysis</i>	<i>Power &amp; Rail</i>

<i>Run Power Analysis</i>	<i>Power - Power Analysis</i>	<i>Power &amp; Rail</i>
<i>Set PG Library Mode</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>Generate PG Library</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>Setup Rail Analysis Mode</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>Analyze ESD</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>Optimize ESD</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>Set Power Network Optimization Mode</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>Run Rail Analysis</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>Run Resistance Analysis</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail</i>
<i>PowerGrid Library Report</i>	<i>Power - Report</i>	<i>Power &amp; Rail - Textual Reports</i>
<i>Power Report</i>	<i>Power - Report</i>	<i>Power &amp; Rail - Textual Report</i>
<i>Power Histograms</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis - Histograms</i>
<i>Power &amp; Rail Results</i>	<i>Power - Report</i>	<i>Power &amp; Rail</i>
<i>Dynamic Movies</i>	<i>Power - Report</i>	<i>Power &amp; Rail - Dynamic Results</i>
<i>Dynamic Waveforms</i>	<i>Power - Report</i>	<i>Power &amp; Rail - Dynamic Results</i>

# Power Analysis and Reports

- [Static Power Analysis Overview](#)
- [Vector-based Average Power Calculation](#)
- [Propagation-based average power calculation](#)
- [Static Power Analysis Flow](#)
- [Static Power Reports](#)
- [Static Power Analysis Plots](#)
- [Viewing and Debugging Static Plots](#)
- [Interactive Queries of Power Data](#)
- [Static Power Histograms and Pie-charts](#)

## Static Power Analysis Overview

### Type of power (internal, leakage, switching)

Static Average Power is consumed in three basic ways in integrated circuits:

1. Switching power, which is the power consumed in the charging and discharging of interconnect capacitances. In most cases, this type of power consumption dominates because of large drivers having to drive large capacitive loads.

$$P = 0.5 * C_L * V^2 * F * A$$

where  $C_L$  is the output capacitive loading,  $V$  is the voltage,  $F$  is frequency, and  $A$  is the average switching activity either from VCD or computed.

2. Internal Power, which is the power consumed in charging and discharging of interconnect and device capacitances internal to cell. Internal power can be divided into two parts:

- Pin Power
- Arc Power

Internal power is calculated by using the internal power tables provided in the .lib, which capture the characterized internal power over a range of input slew rates and external loading. The tables reflect the combination of both the internal switching and internal feedthrough power. Tables are generated as a result of spice simulation during library characterization. If k-factor power scaling parameters (for process, temperature, and voltage) are specified in the .lib file, the power engine will take them into consideration

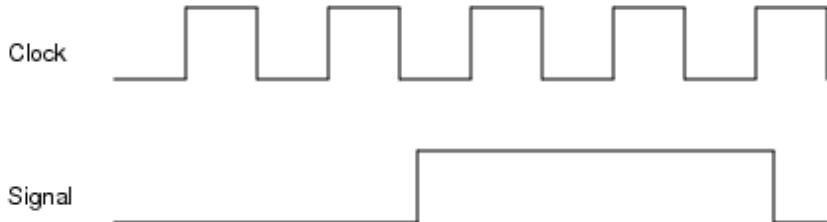
when calculating internal power (Note: timing related scaling factors are not handled by the power engine).

3. Leakage power, which is the power consumed by devices when they are not switching. It includes state-dependent leakage, which is leakage that depends on the state of the gate, that is, whether a transistor is on or off. This value comes from the .lib file if it exists. If k-factor power scaling parameters (for process, temperature, and voltage) are specified in the .lib file, the power engine will take them into consideration when calculating leakage power (Note: timing related scaling factors are not handled by the power engine).

## Definition of activity, duty cycle, and transition density

**Activity** means the probability of all signal nets in design switching from 0 ->1 or 1 -> 0 in one clock cycle.

For instance, if an activity of net or instance is 0.1 then the power engine assumes that net or instance will switch from 0->1 or 1->0 once every ten clock cycles.



For the above diagram,

$$\text{Activity} = (\text{Number of } (0 \rightarrow 1 \text{ or } 1 \rightarrow 0) \text{ transitions} / \text{Number of clock cycles}) = 2/5 = 0.4$$

**Duty Cycle** means the probability that a signal net has the value of 1.

For instance, if signal a net is 1 for 2ns in total simulation time of 10ns then duty cycle of net is 0.2. The duty cycle of the signal in the previous diagram is 0.5 (2.5/5) However, if a signal is Z or X for some time and 0 for rest of time then duty cycle of signal is 0.

**Transition Density** means number of times signal toggle from 0->1 or 1->0 in 1 second.

For the previous diagram and assuming one clock cycle is 4ns, then Transition Density =  $1e+08$  ( $2/20ns$ )

## How PM calculates internal, leakage and switching power including state dependency

The power calculation methods employed inside the power engine are split into 4 components:

1. State dependent internal power associated with input pins
2. State dependent internal power associated with output pins
3. State dependent leakage power
4. Switching Power due to charging or discharging the net loading on the output pins.

When looking at the output power numbers generated by the power engine, the following information is reported in the power file for each instance in the design:

- Instance Name
- Internal Power = sum of (1) and (2) above.
- Switching Power = (4) above.
- Total Power
- Leakage Power = (3) above.
- Cell-type Name

The ordering was chosen to be consistent with previous tools used in the industry. The following sections describe how the power engine calculates the power for each instance based on each of the above 4 components.

### ***State Dependent Internal Power (input pins)***

Inputs can have several sets of power table pairs, each associated with a 'when' clause that specifies the logical condition of inputs that the tables apply to. A call to the table lookup function utilizes a procedure that returns a weighted sum of the energies based on the 'when' clause functions and the signal activity. The weighting of energies is similar to the propagation of static probabilities (duty cycles):

$$y = f(x_0, x_1, \dots, x_n)$$
$$P(y) = P(x_i) * P(y|_{x_i=1}) + (1 - P(x_i)) * P(y|_{x_i=0})$$

The procedure for the weight calculation is similar. However one complication of the energy weighting is that the coverage of the 'when' clauses might not be complete. For example, one set of state dependent tables includes only two 'when' clauses:

```
when : "A & !B";
```

```
when : " !A & B" ;
```

This set of clauses does not account for cases where A and B are either both high or both low. In normal operation, neither of these conditions may appear, so the incomplete coverage may not matter. However, the transition density data is static and lacks signal correlation; the conditions not included in the `when` clauses should be accounted for, or else the internal power will be underestimated. Scaling can resolve this. But it turns out that the most common situation for incomplete clauses is in memories, where assuming the energy to be 0 is the more correct thing to do. As a result, the power engine assumes energy of 0 for missing clauses.

Implementation for state-based internal power on the inputs is straightforward. The implementation for the internal power contribution from a single input port is:

$$inputEnergy = \frac{1}{2} (port \rightarrow riseEnergy() + port \rightarrow fallEnergy())$$

For multiple input ports, each port has a set of energy tables, instead of just one pair, and each pair includes a `when` clause, from which a probability can be calculated:

$$inputEnergy = \frac{\sum_i prob(port \rightarrow when(i)) * (port \rightarrow when(i) \rightarrow riseEnergy() + port \rightarrow when(i) \rightarrow fallEnergy(i))}{2 \sum_i prob(port \rightarrow when(i))}$$

As an example, consider a port with the following data, and with P(A) = 0.25 and P(B) = 0.50:

Index	When	Rise Energy	Fall Energy
0	!A&B	3.0nJ	4.1nJ
1	A&!B	3.2nJ	5.0nJ
2	!A&!B	7.0nJ	9.0nJ

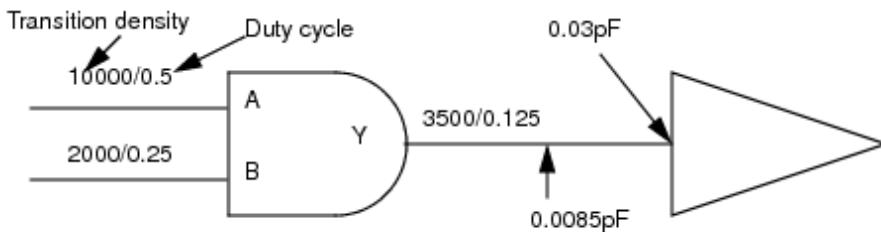
The calculation of energy for a single transition on this input would be:

$$\begin{aligned} inputEnergy &= \frac{prob(!A \& B) * (3.0nJ + 4.1nJ) + prob(A \& !B) * (3.2nJ + 5.0nJ) + prob(!A \& !B) * (7.0nJ + 9.0nJ)}{2(prob(!A \& B) + prob(A \& !B) + prob(!A \& !B))} \\ &= \frac{0.375(7.1nJ) + 0.125 * (8.2nJ) + 0.375(16nJ)}{2(0.375 + 0.125 + 0.375)} = \frac{9.6875}{1.75} = 5.5357\mu J \end{aligned}$$

## ***State Dependent Arc-based Internal Power (output pins)***

Output internal energy is a weighted sum of values extracted from internal power tables that are associated with timing arcs. These tables are indexed by input transition time and output load capacitance. The values for each arc are weighted by the transition density of the corresponding inputs. The resulting energy value is multiplied by the transition density of the output pin to calculate the power.

When state dependent arc-based (output) internal power tables are present, if two or more tables apply to the same arc, they are weighted by the 'when' clauses in the same manner as described in the previous section for "state dependent internal power (input pins)". As an example, we will calculate the output internal power for an AND gate with inputs A and B and output Y. The goal is to calculate the internal power contributed by the output Y. This example does not include any state-dependency.



The first step is to determine the output load capacitance on Y. The output load is the sum of the parasitic net capacitance on the net connected to the Y pin of the instance, plus the pin capacitances of all of the input pins that the net drives. In the example shown, this value is  $0.0085\text{pF} + 0.03\text{pF} = 0.0385\text{pF}$ . Convert this value as required to the units specified in the .lib file for capacitive loads:

```
capacitive_load_unit (1, pf);
```

The next step is to convert the input transition time for each input. The transition time provided is assumed to be extrapolated to 0-100. To convert it, find the bounds in the .lib file:

```
slew_lower_threshold_pct_fall : 10.0;
slew_upper_threshold_pct_fall : 90.0;
slew_lower_threshold_pct_rise : 10.0;
slew_upper_threshold_pct_rise : 90.0;
```

For this case, the time needs to be converted to 10-90. Assuming that in our example that the given transition time for both of the inputs is 0.5ns, the result would be:

$$tt' = \frac{rise - fall}{100} * tt = \frac{90 - 10}{100} * 0.5\text{ns} = 0.4\text{ns}$$

If there are no slew threshold options set in the .lib file, or they are commented out, the default is 20-80.

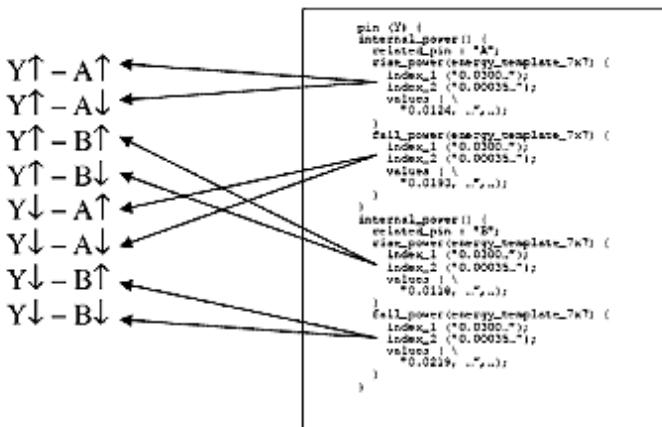
The resulting time should be converted, if necessary, to the time units provided in the .lib:

```
time_unit : "1ns";
```

The next step is to associate the timing arcs with the tables in the .lib file. An AND gate has eight arcs, each corresponding to a change in output logic level in response to a change in an input logic level:

```
Y↑ - A↑
Y↑ - A↓
Y↑ - B↑
Y↑ - B↓
Y↓ - A↑
Y↓ - A↓
Y↓ - B↑
Y↓ - B↓
```

Normally, these eight arcs are represented by four tables in the .lib file. This is the correspondence.



The energy for a transition has two components, rise ( $E_{Y,rise}$ ) and fall ( $E_{Y,fall}$ ). These two values are averaged, because output Y rises as often as it falls, for the total energy for one transition. Note that in the following equations, the rise and fall contributions are averaged as well for the same reason.

$$E_{Y,rise} = \frac{D(A) * \frac{1}{2}(E(Y \uparrow A \uparrow) + E(Y \uparrow A \downarrow)) + D(B) * \frac{1}{2}(E(Y \uparrow B \uparrow) + E(Y \uparrow B \downarrow))}{D(A) + D(B)}$$

$$E_{Y,fall} = \frac{D(A) * \frac{1}{2}(E(Y \downarrow A \uparrow) + E(Y \downarrow A \downarrow)) + D(B) * \frac{1}{2}(E(Y \downarrow B \uparrow) + E(Y \downarrow B \downarrow))}{D(A) + D(B)}$$

In the above equations,  $D(A)$  is the transition density on input A, and  $E(Y^*A^*)$  is the energy value looked up from the corresponding table as described above.

For this example, the energy is:

$$E_{Y,\text{rise}} = \frac{10000s^{-1} * \frac{1}{2}(0.0134pJ + 0.0134pJ) + 2000s^{-1} * \frac{1}{2}(0.0135pJ + 0.0135pJ)}{10000s^{-1} + 2000s^{-1}}$$

$$= \frac{161pJs^{-1}}{12000s^{-1}} = 0.0134167pJ$$

$$E_{Y,\text{full}} = \frac{10000s^{-1} * \frac{1}{2}(0.0244pJ + 0.0244pJ) + 2000s^{-1} * \frac{1}{2}(0.0267pJ + 0.0267pJ)}{10000s^{-1} + 2000s^{-1}}$$

$$= \frac{297.4pJs^{-1}}{12000s^{-1}} = 0.027833pJ$$

The power is:

$$P = \frac{1}{2}(E_{Y,\text{rise}} + E_{Y,\text{full}})D(Y)$$

$$= 0.5(0.0134167pJ + 0.027833pJ)3500s^{-1}$$

$$= 66.85pW$$

## ***State Dependent Leakage Power***

As the leakage component of power dissipation increases, accurate estimation of it becomes more and more critical. Originally, the leakage component of a cell's power dissipation was modeled in Liberty libraries as a single number. However, nowadays it is more common to associate different leakage power values with different input combination ("state-dependence").

The method to compute leakage power is as follows. Extract the state-dependent leakage data from a cell's Liberty .lib description and compute a weighted sum of the leakage values based on the instance's input probabilities.

The state-dependence is expressed as a set of logical functions describing various input conditions. This set of functions may or may not be complete (e.g. covering all possible input combinations). In addition, a generic leakage power value may also be provided. The power engine covers all of the possible combinations of available data.

State-dependent leakage data appears in a Liberty library in the following form:

```
cell_leakage_power : 14.335 ;
leakage_power() {
when : "!A1 !A2" ;
value : 9.120 ;
}
leakage_power() {
when : "!A1 A2" ;
```

```
value : 16.467 ;  
}  
leakage_power() {  
when : "A1 !A2" ;  
value : 12.364 ;  
}  
leakage_power() {  
when : "A1 A2" ;  
value : 19.390 ;  
}
```

Note that the set of `when' clauses are complete; all possible combinations of the inputs A1 and A2 are accounted for. Also note that there is a generic leakage power value that is not associated with any condition.

We assume the following combinations of input data for our approach:

1. Complete clause set with or without generic leakage value
2. Incomplete clause set with generic leakage value
3. Incomplete clause set without generic leakage value
4. Over-complete clause set with generic leakage value (error condition)
5. Over-complete clause set without generic leakage value (error condition)

If the clause set is complete, we expect the sum of the probabilities of all of the clauses to add up to 1.0. Verification of a complete clause set requires logical analysis of the statements. As extensive library verification is not within the functional requirements of the power engine, we base our assessment of the clause set's completeness by the sum of the probabilities. If this sum is equal to 1.0, the set is complete. If the sum is less than 1.0, we will assume the set is incomplete. If the sum is greater than 1.0 (which would indicate over-coverage), we assume that there is an error in the library data. In order to decide which of the above five conditions we have, we need two pieces of information: whether or not we have a generic leakage value and the sum of the clause probabilities.

We find the probability sum as follows:

$$probTotal = \sum_i prob(cell \rightarrow when(i))$$

Then we use the following equations and procedures to detect and calculate the five input conditions:

1. Complete clause set with or without generic leakage value

This condition is assumed if probTotal is equal to 1.0. For this case we calculate the weighted sum of all available clauses:

$$leakSum = \sum_i cell \rightarrow when(i) \rightarrow value() * prob(cell \rightarrow when(i))$$

$$leakagePower = leakSum$$

## 2. Incomplete clause set with generic leakage value

For this case, we use the generic leakage value to "fill in" the missing clauses. We do this by weighting the generic leakage value with 1.0 minus probTotal.

$$leakagePower = leakSum + (cell\_when\_generic\_value () * (1.0 - probTotal))$$

## 3. Incomplete clause set without generic leakage value

For this case we simply scale up the leakSum value to accommodate the missing clauses. We accomplish this by dividing it by probTotal.

$$leakagePower = \frac{leakSum}{probTotal}$$

## 4. Over-complete clause set with generic leakage value (error condition)

This case occurs when probTotal is greater than one, and there is a generic leakage value. For this case, we simply revert to the generic value (a warning is also printed to alert the user that there is a possible problem with the library).

$$leakagePower = cell \rightarrow when\_generic\_value()$$

## 5. Over-complete clause set without generic leakage value (error condition)

Without a generic leakage value, we must use the incorrect leakage data as best as we can. We do this by calculating leakSum and scaling it down. It uses the same equation as condition 3. (A warning is also be printed).

## **Switching Power**

Switching power is calculated using the basic equation:

$$SwitchingPower = CAV^2F$$

Where:

C = Loading net capacitance (SPEF/DSPF or a default load value)

V = Voltage

A = Nodal activity

F = Operating frequency

The product of "A\*F" is the transition density (D) calculated during the activity propagation inside the power engine. Since the calculated transition density includes both rising and falling transitions, the equation is modified for a given power rail as:

$$SwitchingPower = 1/2CV^2D$$

Where a net is driven by multiple outputs, a good example is a clock mesh driven by parallel clock drivers, the capacitance is split or divided amongst the output drivers.

## Vector-based Average Power Calculation

The vector-driven approach uses the VCD or TCF output of a logic simulator to obtain the number of transitions for each net. PM requires gate-level VCD or TCF with good functional coverage for accurate power calculation results.

You can use VCD or TCF information to calculate accurate power consumption figures, if the following conditions apply:

- Gate-level simulation is possible at the full-chip level.
- Gate-level simulation provides sufficient functional coverage for the design.
- The vectors include those that cause the highest power consumption.

The power engine calculates the number of transitions from  $0<->1$ ,  $0/1<->X$  and  $0/1<->Z$ . The  $0<->1$  transition is counted as 1,  $0/1 <-> X$  transitions are counted as 0.5 by default and  $0/1 <->Z$  transitions are counted as 0.25 by default. User can use `-x_transition_factor` and `-z_transition_factor` options of `set_power_analysis_mode` command for changing the default value of  $0/1 <->X$  or  $0/1 <->Z$  transitions.

The power engine also calculates the duty cycle of each net for state dependent internal or leakage power calculation as described previously. The power engine also takes clock definition from VCD or TCF and gives higher priority to clock definition from VCD or TCF if there is discrepancy between clock frequency in SDC or TWF and VCD or TCF.

## Propagation-based average power calculation

The power engine calculates the switching probability, as well as static state probability, of each net in the design. The propagation based approach is vector-independent and provides coverage for all nets in a design.

However, the accuracy depends on good starting values, that is, information about the switching probabilities at the primary inputs in a design. Simple examples are clock and reset or enable inputs. Obtaining an accurate prediction without information about the switching probabilities of these special inputs is difficult, and in most cases an inaccurate prediction causes an over estimation of the power consumption.

## Activity Propagation in the power engine

Activity propagation inside the power engine can be divided into following categories.

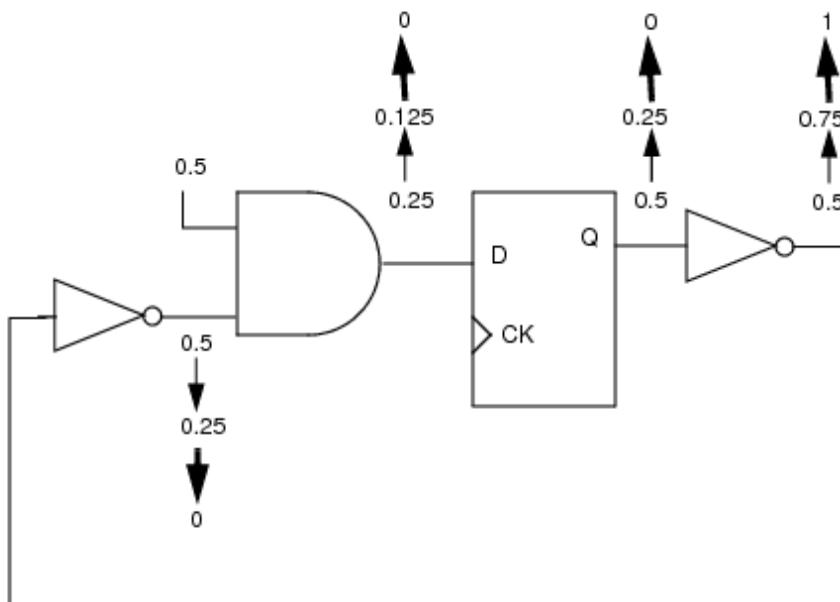
### ***Activity propagation through combinational cells***

Activity propagation through combinational cells is easier. The power engine gets function of combinational cell from .lib and uses the function to propagate activity through combinational cells. The power engine also propagates duty cycle through combinational cells. The only tricky part is when there are combinational loops inside design. In this case the power engine seeds activity at input to break combinational loop. The seeded activity is based upon internal heuristic of power engine and takes into account activity of other neighboring pins.

### ***Activity propagation through sequential Cells***

Activity Propagation through sequential cell is based upon activity of input pin, set or reset pin, and scan enable pin. However most of sequential cells are in sequential loops like state machines which make activity propagation through sequential cell based on heuristics by seeding activity at input of sequential cell. Therefore it is recommended to provide activity at outputs of sequential cells. With the power engine you can use either the `set_default_switching_activity` command to specify the average activity on sequential cells or use RTL, VCD, or TCF for seeding activity at sequential cells.

As seen in the example below, the activity at the output of the sequential cell in the loop can not be resolved using propagation. In this case, iterating the loop to determine the activity at the output Q of the sequential cell will result in diminishing activity towards 0. It proves that using heuristic method to compute activity in sequential loop is an intractable problem for propagation based power calculation. Therefore, in order to get good average power numbers for the design under test, user should always specify average activity at the output of sequential cell using above mentioned command. In addition, user can override this default activity using VCD or TCF.



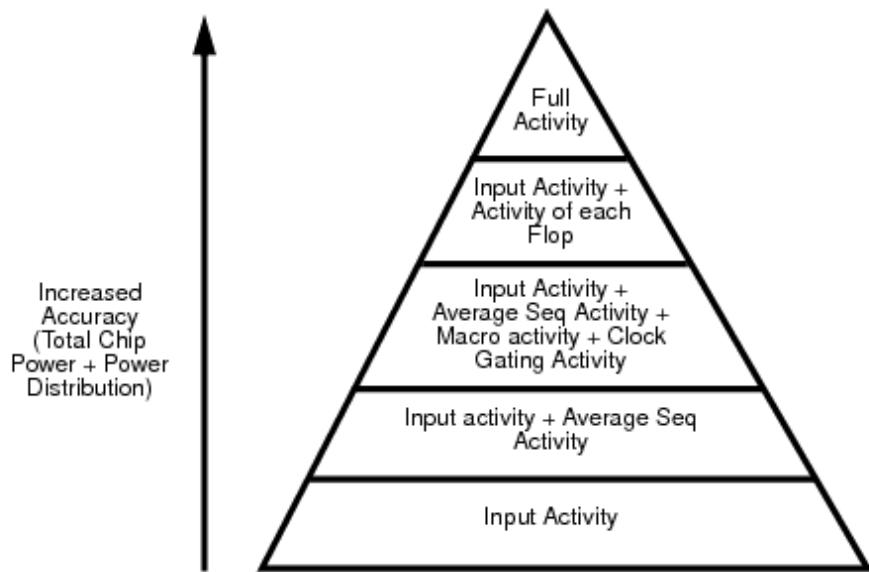
### ***Activity propagation through macros***

The major component of power in macros is internal power. Internal power of macros is highly sensitive to activity on read and writes signals. Small change in activity of read or write signal can cause large change in internal power numbers. Therefore it is recommended that users specify activity at the read and write signals of macros.

### ***Activity propagation through clock network and clock gates***

For accurate propagation through clock network it is important that user specifies TWF file which has clock frequency of generated clocks as well. The activity propagation of clock through clock gating cells depends upon activity of clock enable signal. Since clock enable is a signal net, it will generally have low activity unless specified, which will cause lot of optimism in power calculation. Therefore it is recommended that user should specify activity at enable of clock gating cells for proper propagation through clock network. You can use `set_default_switching_activity` command in Innovus for specifying average activity at enable signal of all clock gating cells.

## Recommended methodology for activity propagation



The above diagram explains the recommended methodology for activity propagation. The accuracy of results increase as you specify more inputs to the power engine. At the very least user should specify the average sequential activity, which is MUST for accurate activity propagation.

Here are some of examples which depict how well the above methodology works:

Design	Input TCF (from VCD)	Input TCF and Avg seq activity (from VCD)	VCD
Testcase1	75.83	96.90	70.75
Testcase2	148.80	230.0	199
Testcase3	279.46	258.45	195
Testcase4	243.48	243.24	170

The table clearly shows that if you specify just activity on inputs then the result vary from -25% from +45% whereas if you specify average sequential activity of design then results are pessimistic by 25-30%, which is expected because combinational activity propagation is meant to be pessimistic.

Here are another two examples which show how well results correlate after specifying activity at Macros and clock gating cells.

Testcase 5:

- Expected power = ~375mW
- Power after specifying default input activity = 225mW

- Power after specifying default input activity + macro activity = 254mW
- Power after specifying default input activity + macro activity + clock gating activity = 370mW

Testcase 6:

- Expected power = 10W
- Power after specifying default input activity = 5.5W
- Power after specifying clock gating activity + Macro activity = 10.8W

## Static Power Analysis Flow

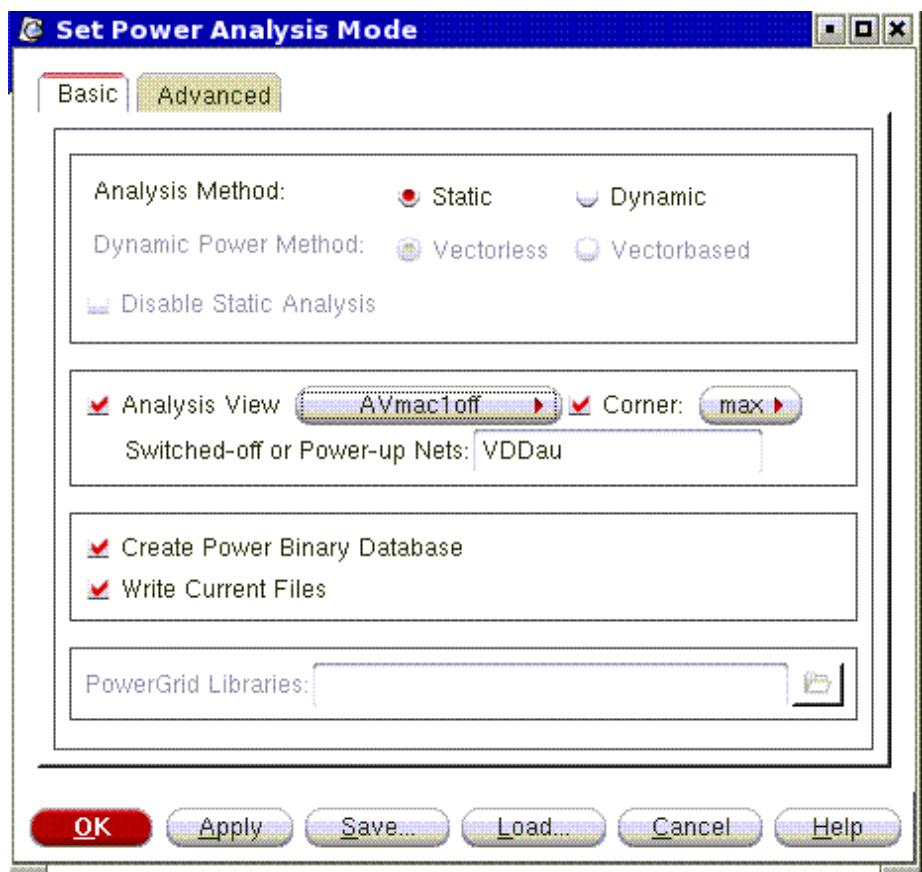
To perform static analysis one must setup the analysis mode and then run power analysis.

### Set Power Analysis Mode

To setup the static power analysis mode do the following steps:

1. Select *Power & Rail - Set Power Analysis Mode* form as shown in [Figure 39-1](#) will appear.
2. Specify *Analysis Method* as *Static*
3. Specify a *CPF Analysis View* or *Corner* to use for power calculation
4. Specify *Switched Off Net* if it is a power-gated design
5. Specify Power-grid library
6. Select *OK* or *Apply*

**Figure 39-1 Set Power Analysis Mode form**



## Run Power Analysis

1. Select *Power & Rail - Run Power Analysis* and the form shown in [Figure 39-2](#) will appear.
2. Select *Basic* tab if not already selected.
3. Specify Primary *Input Activity* (recommended)
4. Specify *Dominant Clock Frequency*
5. Specify *Flop Activity* (recommended)
6. Specify *Clock Gate Enable Activity* (recommended)
7. Specify the *VCD* file (full or partial), if available.
  - a. Select *VCD*.
  - b. Specify the appropriate values in the four fields.
  - c. Select the *Add* button.
  - d. Repeat until all needed scopes are covered.
8. Select the *Activity* tab and the form will appear as is shown in [Figure 39-3](#).
9. Global activity can be assigned for specific parts of the hierarchy.
  - a. Select *Global Activity* and specify a value.
  - b. Select *Hierarchy* if needed and select the *Add* button. Continue until all necessary hierarchy global activities are defined.
10. Specify TCF or SAF file (full or partial), if available.
  - a. Select *Activity* or *Transition Density*, depending on how you want to specify the values.
  - b. Select *Net*, *Pin*, or *Port* to specify the type of activity that you want to specify.
  - c. Specify a *Duty* or *Period*, if needed.
  - d. Select the *Add* Button. Continue until all activities are specified.
11. Select the *Power* tab and the from will appear as is shown in [Figure 39-4](#).
12. Specify *Custom Power* for *Cell*, *Instance Name* if available.
13. Select *Add* and repeat until done.
14. Select *OK* or *Apply*

**Figure 39-2 Run Power Analysis - Basic form**

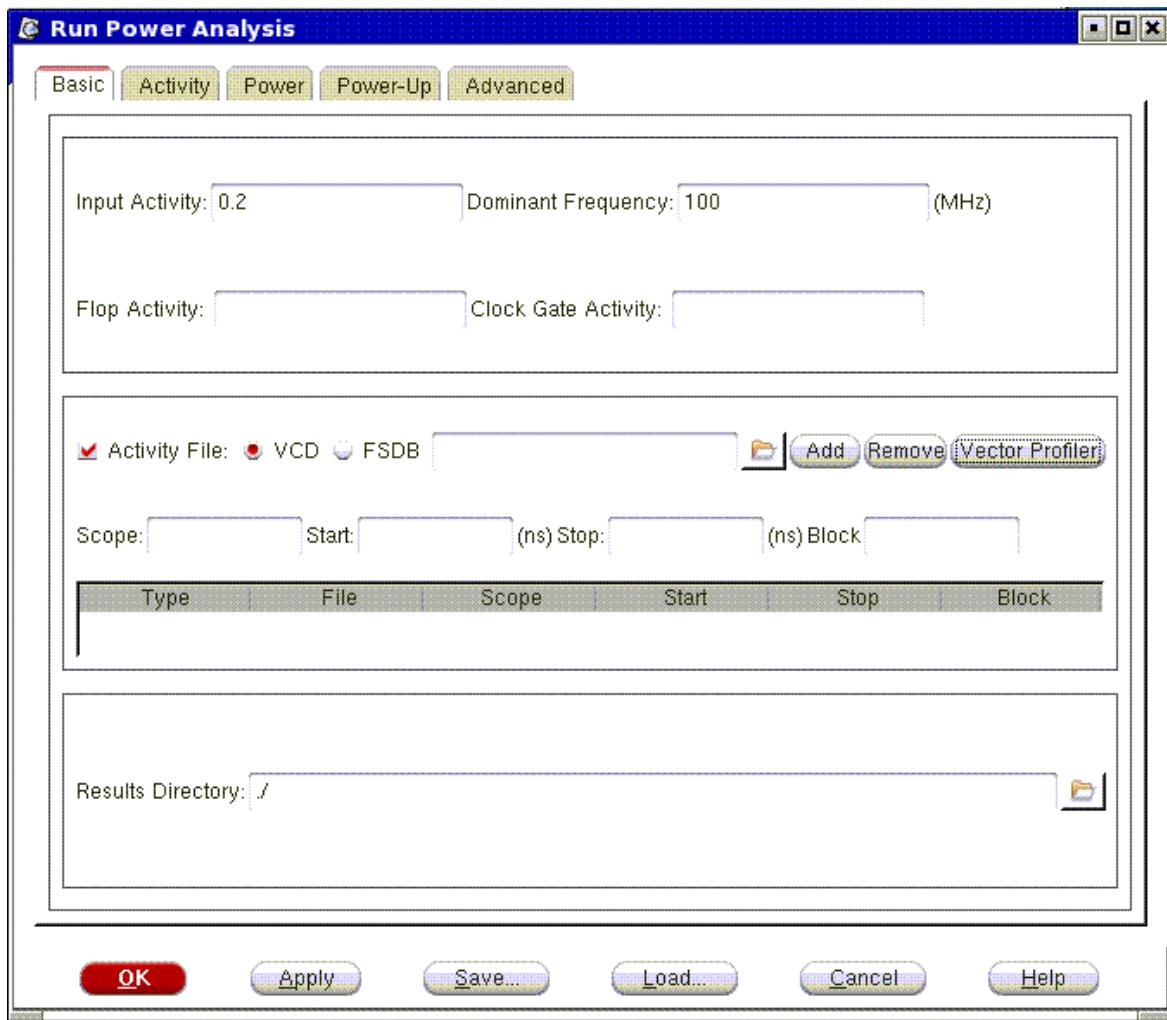


Figure 39-3 Run Power Analysis - Activity form

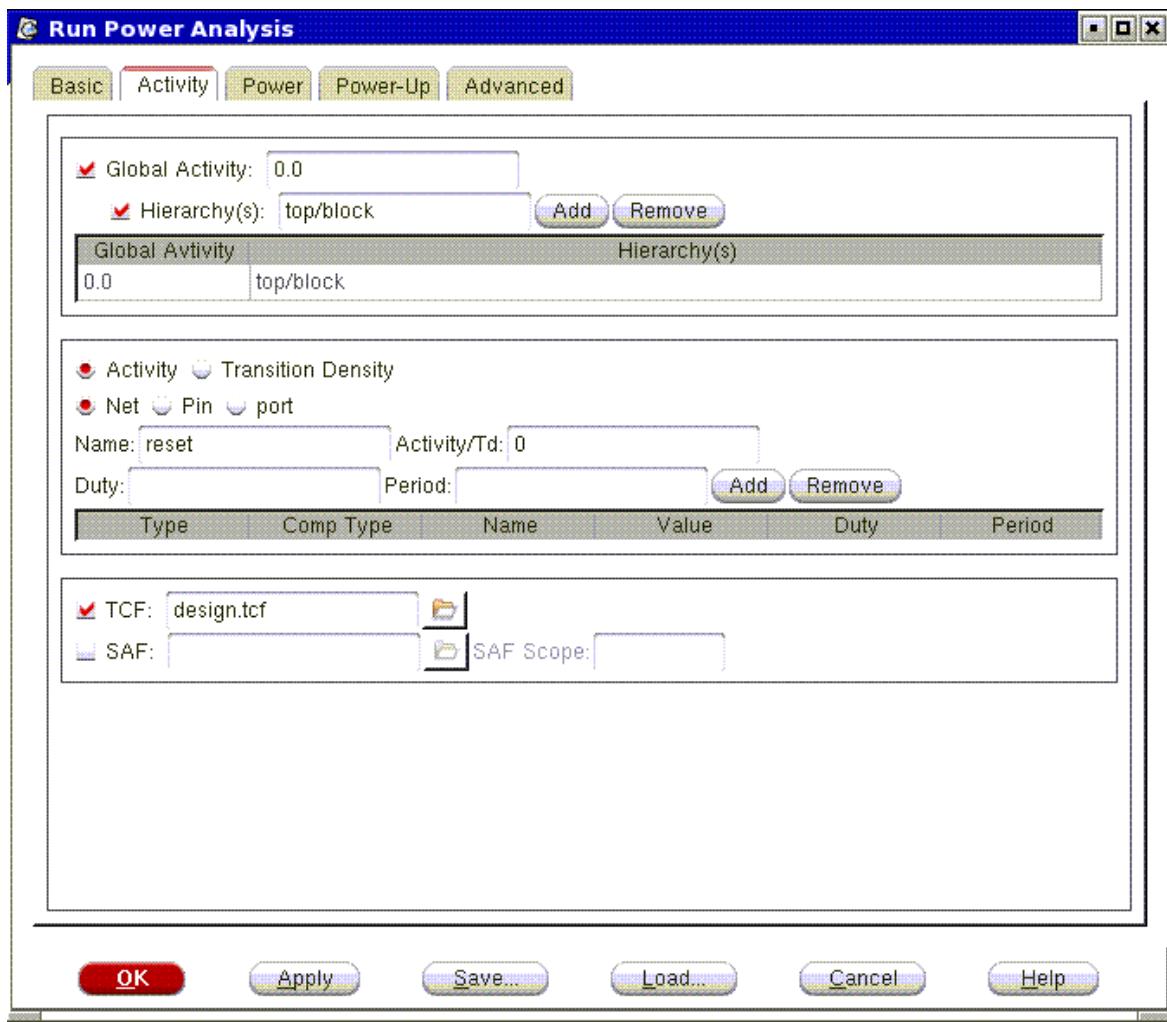
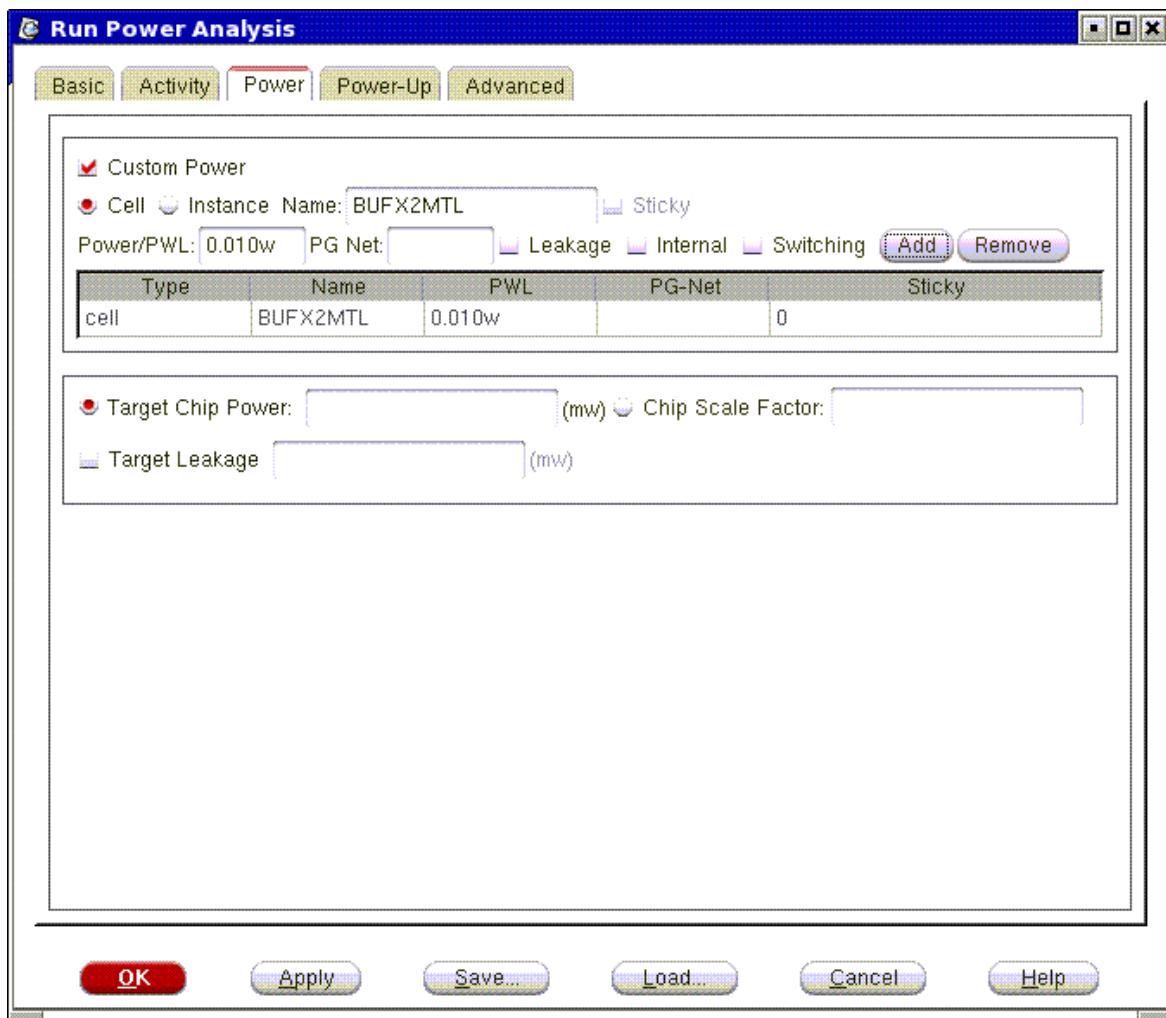


Figure 39-4 Run Power Analysis - Power form



## TCL Command:

The example TCL command for static power calculation is as follows:

**Note:** Commands highlighted in red are optional

```
set_power_analysis_mode -method static -corner max -off_pg_nets VDDau
    -create_binary_db true -write_static_currents true
set_default_switching_activity -input_activity 0.2 -period 10.0 -seq_activity 0.1
read_activity_file -format tcf design.tcf
set_default_switching_activity -global_activity 0.0 -hier top/block
set_switching_activity -activity 0 -net reset
set_power -type cell BUFX2MTL 0.010w
```

```
set_power_output_dir static_power_max  
report_power -outfile design.rpt
```

**Note:** If you do not have a SPEF file, you can use wire load models for load specification which can then be used for static power analysis. The commands to specify wire load models are :

**Note:** If you have the following commands,

```
set_power_output_dir static_power_max  
report_power -outfile design.rpt
```

then `design.rpt` will be dumped to `static_power_max` directory.

If you explicitly specify a directory in `-outfile` option such as below,

```
report_power -outfile my_dir/design.rpt
```

then `design.rpt` is deposited to `my_dir` directory and the path setting by `set_power_output_dir` will be ignored.

## MMMC mode default views

If set to MMMC mode, it is recommended to specify the view by one of the below commands,

```
set_power_analysis_mode -analysis_view viewname
```

or

```
report_power -view viewname
```

If you do not, then the default power view is the first setup or hold view depending on the `checkType` of the `set_analysis_mode` command.

Power analysis runs on only one view at a time, and therefore, the argument to the `-view` parameter must specify only one view. If you give multiple view names to the `-view` parameter of `report_power` or you do not activate the view using `set_analysis_mode`, a warning message will be displayed.

For multiple views, you need to have multiple runs, one for each view. Also, you need to make the view active in order to do power analysis.

If specified as shown below,

```
set_analysis_view -setup {view1 view2} -hold {view3 view4}
```

```
read_spef -rc_corner best design.spef
set_analysis_mode -checkType setup
report_power
```

then the power view in this case is the first setup view which is `view1`. The default for `-checkType` is `setup`. See `set_analysis_mode` in the *Innovus Text Command Reference* for details of this command.

If specified as shown in the next example,

```
set_analysis_view -setup {view1 view2} -hold {view3 view4}
read_spef -rc_corner best design.spef
set_analysis_mode -checkType hold
report_power
```

then the power view in this case is the first hold view which is `view3`.

**Note:** When CPF or MMMC views are loaded, SPEFs must be read after the `set_analysis_view` command as shown in the above examples.

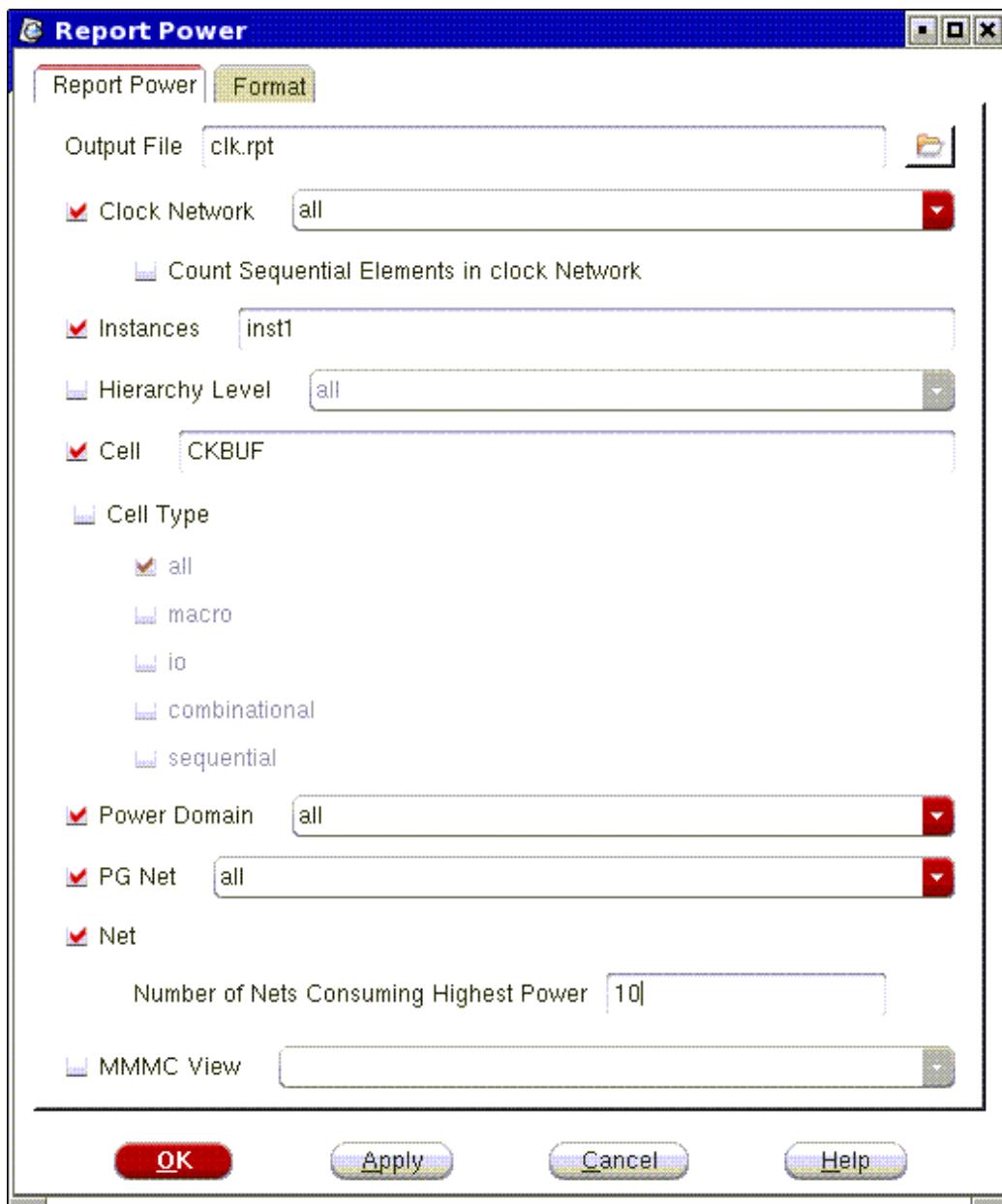
If multiple command types specify the view, the priority is given as follows (from highest to lowest): `report_power`, `set_power_analysis_mode`, `set_analysis_mode`

For additional MMMC details see *Performing Multi-Mode Multi-Corner Timing Analysis* in the *Innovus User Guide*.

## Static Power Reports

The incremental power reports can be generated using *Power & Rail - Report - Power Report* which will bring up the form shown below. Select the items you want to report and *Apply*.

**Figure 39-5 Report Power form**



## TCL Command:

TCL command for generating the incremental report is shown below.

```
report_power -clock_network all -outfile clock.rpt
report_power -instances inst1
report_power -cell CKBUF
report_power -cell_type all
```

```
report_power -nworst 10
```

## Static Power Analysis Plots

The calculated static power can be debugged in the context of physical layout using interactive power analysis plots. To view the static plots you take the steps as follows:

1. Select *Power & Rail - Report - Power & Rail Results* to bring up the form shown in [Figure 39-6](#).
2. If power is calculated during the session, select *Power Analysis Type* radio button and select a power analysis type. See [Viewing and Debugging Static Plots](#) for details of the various plot types and how they can be accessed.
3. If you want to see the power plot overlaid on the layout, select *Show Overlay* and the results of this can be seen in [Figure 39-7](#).

**Figure 39-6 Power & Rail Results form**

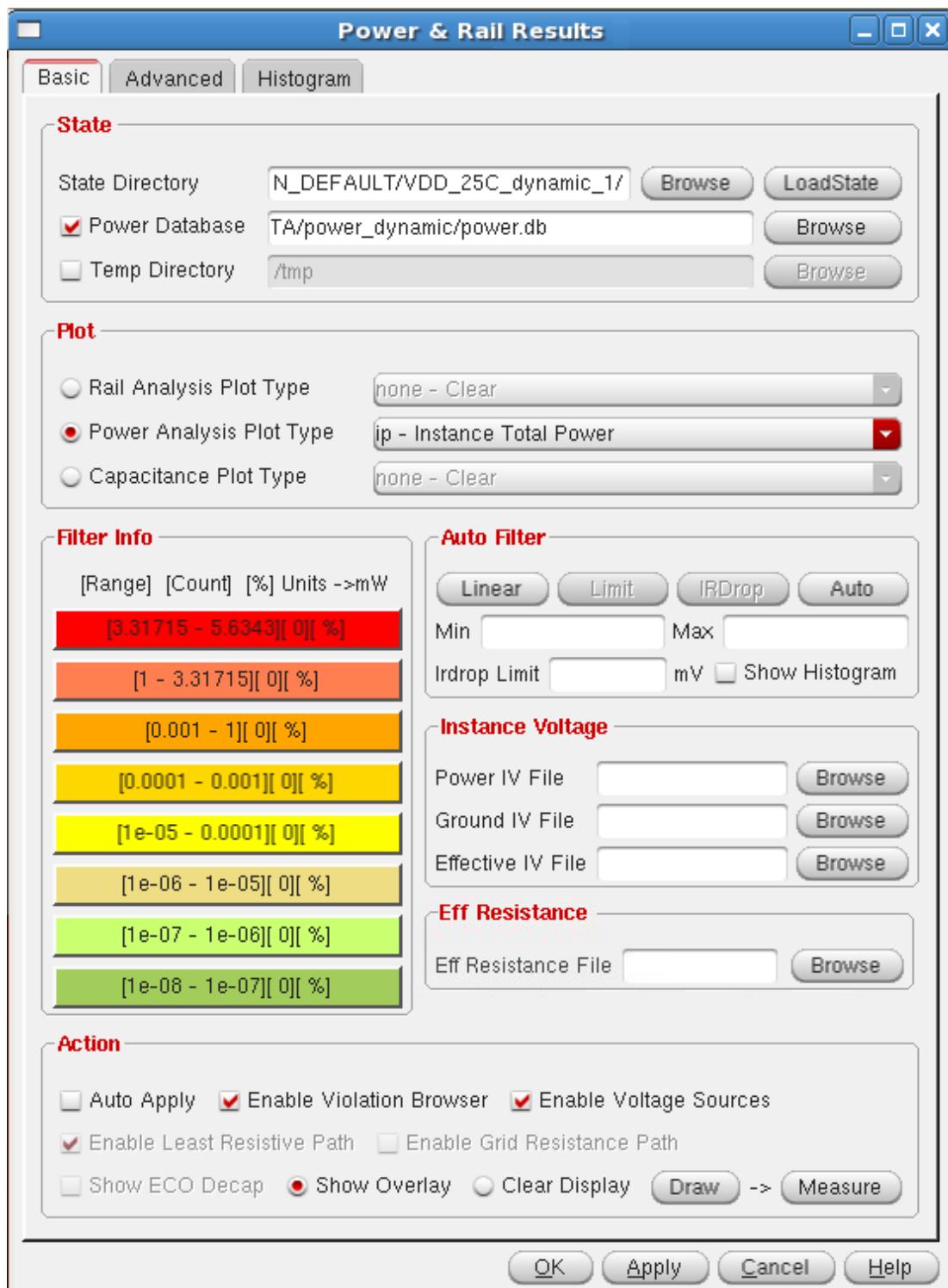
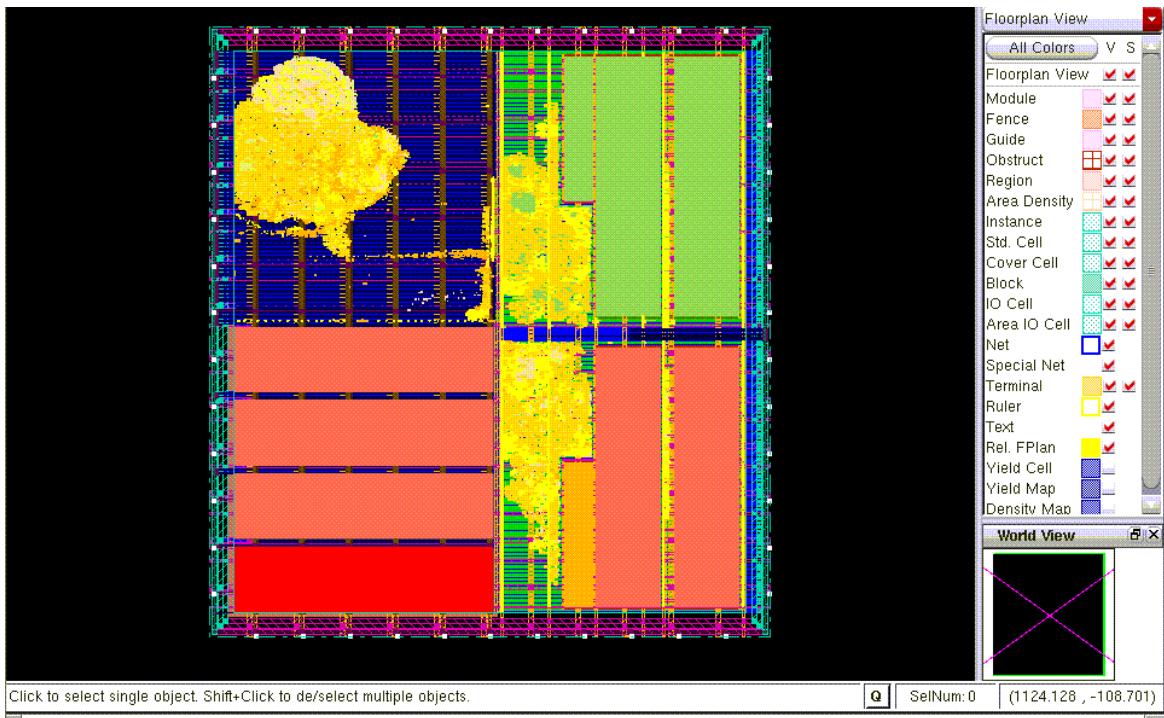


Figure 39-7 Design layout overlaid with power information



## TCL Command:

Some TCL commands for viewing power analysis plots:

```
view_analysis_results -power_db power.db -plot ip  
view_analysis_results -plot ip_s  
view_analysis_results -plot freq  
view_analysis_result -free_data
```

## Viewing and Debugging Static Plots

Static Power can be read in two ways:

- From calculated power stored in memory during the session
- From power.db generated during static power calculation

The following static power plots can be displayed in the Innovus GUI:

- Total Power - ip
- Internal Power - ip\_i
- Switching Power - ip\_s

- Leakage Power - ip\_l
- Frequency Domain - freq
- Transition Density - td
- Loading Capacitance - load
- Slack - slack

- **Total Power - ip**

Total power plots show power distribution of all the instances in the design. It can be used to debug regions of high IRdrop in the design

- **Internal Power - ip\_i**

Internal power is a component of total power and displayed for all instances in the design. It can be used to debug instances that consume unexpectedly a large total power. The internal power is derived from .lib file.

- **Switching Power - ip\_s**

Switching power is a component of total power and displayed for all instances in the design. The switching power plot can be used in conjunction with transition density and loading capacitance plots to understand the switching profile for the design (i.e. high activity and loading capacitance for the instance translates to high switching power).

- **Leakage Power - ip\_l**

Leakage power is a component of total power and displayed for all instances in the design. The leakage power plot can be used to debug high leakage power instances.

- **Frequency Domain - freq**

The frequency domain plot displays the operating frequency of all instances in the design. The instances operating with multiple clock domains are displayed using the fastest clock frequency for the instance. The power is directly proportional to the frequency, so this plot can be used to debug instances with high power consumption.

- **Transition Density - td**

Transition density is the product of frequency and activity. It is plotted for all instances in the design. This plot can be used to debug instances with high power consumption and regions of high activity.

Transition density is plotted for the off domains in power-gated design. However, switching power plot correctly displays that no power is calculated for the off domains

- **Loading Capacitance - load**

Loading capacitance is directly proportional to the power. It is plotted for all instances in the

design. This plot can be used to debug instances that drive large output capacitance resulting in high power consumption.

- **Slack - slack**

Slack for the instance is derived by Common Timing Engine or external Timing Window File. This plot is primarily used to identify time critical instances and can be useful when performing timing aware decap optimization and IRdrop aware timing analysis.

## Interactive Queries of Power Data

The power data can be queried interactively in GUI by selecting an instance using left mouse button and pressing "Q" on keyboard. The Attribute Viewer displays total, internal, switching and leakage power for the instance. In addition it lists frequency domain, transition density and associated power domains of the instance.

## Static Power Histograms and Pie-charts

The calculated static power can be debugged using pie-charts and histograms. To see the histograms you do the following steps:

1. Select *Power & Rail - Report-> Power Histograms* menu. The Power Debug form appears.
2. Double click on hierarchy model (at the right of the pie chart) to explode its power distribution. Use up level to go up one level.
3. Selecting the *Histograms* tab will bring up various histograms that you can view. After this form appears, you can select other tabs to view the histograms with various types of data.

You can search for net(s) in the *By Net*, *Net Toggle*, and *Net Probability* tabs using wildcard entries (\*) and filter any net(s) to display in the histogram when debugging static power. This displays 50 nets at a time, starting with the first 50 nets.

The net that you select in the histogram gets highlighted in the table, and similarly, the net that you select in the table gets highlighted in the histogram.

Information about the selected net is displayed across all the three tabs - *By Net*, *Net Toggle*, and *Net Probability*.

# Analyzing and Repairing Crosstalk

- Overview
- Inputs for SI Analysis
- Setting Up Innovus for SI Analysis
  - RC Extraction Settings
  - Noise Analysis Settings
  - Static Timing Analysis (STA) Settings
  - Advanced Settings for SI Analysis
  - Example of Setting Up Innovus for SI Analysis
- Preventing Crosstalk Violations
- Fixing Crosstalk Violations
  - Data Preparation
  - Using optDesign to Fix Setup Violations with Effects
  - Using RC Data Generated by an External Tool for SI Fixing
  - Using SDF Data Generated by an External Tool for SI Fixing
  - Using optDesign to Fix Hold Violations with Crosstalk Effects
  - Using optDesign to Fix Transition Time Violations with Crosstalk Effects
- Performing XILM-Based SI Analysis and Fixing

## Overview

Crosstalk is the undesired electromagnetic coupling between signal lines that causes functional failures and delay variation. The effects of crosstalk might slow down or speed up the delay depending on the transition direction of the two coupling nets.

The Innovus™ Implementation System supports signal integrity (SI) operations that include crosstalk prevention during detail routing and analysis and repair afterwards. The crosstalk repair features all optimization techniques currently used for regular base timing postRoute optimization.

## Inputs for SI Analysis

The design input files required are the same as needed for regular base postRoute timing optimization:

- Netlist

- SDC (timing information)
- Routed Innovus database or DEF file (placement and routing information)
- LEF file (physical library)
- XILM data (for hierarchical designs)
- Liberty library (.lib)
- Innovus extended capacitance table file
- Quantus QRC standalone extraction technology file and library (optional)

If available you can also supply the following which will be used by the Advanced Analysis Engine (AAE) timing analysis tool for extra accuracy:

- .cdb noise library

## Setting Up Innovus for SI Analysis

- [RC Extraction Settings](#)
- [Noise Analysis Settings](#)
- [Static Timing Analysis \(STA\) Settings](#)
- [Advanced Settings for SI Analysis](#)
- [Example of Setting Up Innovus for SI Analysis](#)

## RC Extraction Settings

The RC extraction settings for SI analysis include the extraction engine and the extraction filters.

### Extraction Engine

You can use one of the following postRoute extraction engines:

- Detail
- tQuantus
- Integrated QRC (IQRC)

- Standalone Quantus QRC

For 65nm technology and above, Innovus software uses detailed extraction for postRoute timing and/or optimization. However, for 65nm and below technology, if Quantus QRC technology files are available, the Innovus software uses tQuantus as the default postRoute extraction engine. For superior correlation with signoff extraction, use of tQuantus and IQRC extraction engine is recommended. The IQRC extraction engine provides the highest accuracy in implementation flow and is particularly recommended at ECO for incremental extraction.

**Note:** IQRC extraction requires a separate QRC license.

To use the tQuantus, IQRC, or the Standalone Quantus QRC extraction engine (the latter comes with a CPU penalty so is not usually recommended), use the following command:

```
setExtractRCMode -engine postRoute -effortLevel [medium | high | signoff]
```

Where:

- medium invokes the tQuantus engine
- high invokes the IQRC engine
- signoff invokes the Quantus QRC engine

## Extraction Filters

Extraction filters enable you to reduce the total number of parasitic capacitors in the design by grounding some net to net coupling capacitance based on total net capacitance, absolute coupling capacitance size, or relative coupling capacitance size compared to total capacitance.

## Effect of RC Extraction Settings on SI Analysis

Extraction coupled capacitance filtering has a significant impact on SI analysis and run time. The Innovus software automatically sets the default values for the RC extraction filters based on the process node specified using the -process parameter of the `setDesignMode` command. To set the process node, use the following command:

```
setDesignMode -process process_node
```

**Note:** For more information on the default values assigned to the filtering parameters with respect to the specified process node, see the `setDesignMode` command.

If you do not want to use the default filtering values for RC extraction, specify the following parameters of the `setExtractRCMode` command to adjust the coupling capacitance filters:

- `-total_c_th`: Specifies the threshold value (femtoFarads) that determines when the extractor lumps a net's coupling capacitance to ground. The software grounds the coupling capacitances

for nets which have a total capacitance value less than the value specified with this parameter.

- `-coupling_c_th`: Specifies the threshold value that determines when the extractor lumps a net's coupling capacitance to ground. The software decouples the coupling capacitance of nets when the total coupling capacitance between the pair of nets is lower than the threshold specified with this parameter.
- `-relative_c_th`: Sets a ratio threshold value that determines when the extractor lumps a net's coupling capacitance to ground. If the total coupling capacitance between a pair of nets is less than the percentage (specified with this parameter) of the total capacitance of the net with the smaller total capacitance in the pair, the coupling capacitance between these two nets will be considered for grounding.

## Guidelines for RC Extraction Settings

Use the following guidelines while setting up extraction:

- Note that the detailed extraction engine is significantly faster compared to tQuantus or IQRC. However, it trades off accuracy of the extraction results for performance. tQuantus is about 30% faster than IQRC. tQuantus and IQRC both support distributed processing and their use is strongly recommended to offset longer runtime. Moreover, the IQRC engine takes advantage of incremental extraction capability in SI optimization flow to reduce runtime in subsequent cycles.
- Ensure that the filters that you are using in the Innovus software for SI fixing are the same as the ones used for SI signoff analysis.
- The default filtering values set by the Innovus software based on the process node (`setDesignMode -process`) attempt to capture the most significant effects of coupling capacitances on SI analysis. It is strongly recommended that you correlate your RCs using these default filters (set by the Innovus software) with the RCs from your signoff extractor.
- While setting the filtering thresholds, ensure that you retain small coupling capacitors because AAE-SI analysis lumps these together into a single virtual attacker model. Multiple small coupling capacitors can result in a significant virtual attacker.
- Exercise caution while setting low-value filters because this can increase the run time significantly.

## Noise Analysis Settings

Noise analysis settings include loading the input noise model, configuring the timing windows, setting the delta delay threshold and specifying the virtual attacker mode.

### Timing Models for SI Delay Calculation

Innovus supports NLDM, ECSM, and CCS-based Liberty (.lib) timing libraries for performing SI delay calculation. To help improve ECSM and CCS library loading times, .lib files may be converted into the ldb format using the [write\\_lDb](#) command. cdB noise libraries are also supported.

**Note:** For hierarchical designs, you need XILM data. For more information on XILM-based SI analysis, see the [Top-level Timing Closure Methodologies](#) chapter in the *Innovus User Guide*.

### Timing Windows

Timing windows are used to filter out signals that are not switching simultaneously. The internal timing engine computes the timing windows and slew rates automatically.

### Delta Delay Threshold

The user can set the delta delay threshold for noise-on-delay analysis using the following command:  
(The value is 0 by default)

```
setSIMode -deltaDelayThreshold va lue
```

If you have a separate delta delay threshold for clock nets, use the following (Note that this will override any value specified by `-deltaDelayThreshold` for the clock nets):

```
setSIMode -clockDeltaDelayThreshold va lue
```

### Guidelines for Noise Analysis Settings

- If you are using Tempus Timing as the Signoff Solution use the following setting to correlate to that: (default setting)

```
setSIMode -analysisType aae
```

- If you are using a third party tool as the Signoff Solution use the following setting to correlate to that:

```
setDesignMode -thirdPartyCompatible true
```

## Static Timing Analysis (STA) Settings

Static timing analysis settings include the timing analysis engine and the analysis conditions.

### Input Timing Library

The primary input for timing analysis is a Liberty (.lib) library.

### STA Engine

Innovus uses the native timing analysis engine to perform static timing analysis. This engine is same as the one used by Tempus Timing Signoff Solution. You can choose to use the built-in timing analysis engine or a third-party tool for performing timing analysis.

### Analysis Conditions

The important analysis conditions for running SI analysis include:

- Setting Up the OCV Analysis Mode & removing common path pessimism

Innovus provides different timing analysis modes and performs different calculations for setup and hold checks for each mode. For SI analysis, set the analysis mode to On-Chip Variation using the `setAnalysisMode` command. It is also recommended to turn on:

```
setAnalysisMode -analysisType onChipVariation
```

OCV mode assumes that capture clock, launch clock, and data path can have maximum or minimum delays in setup and hold analysis. This is the recommended analysis mode for noise analysis and must be set before any AAE analysis can be done.

To remove the common path pessimism which is strongly recommended, use the following command:

```
setAnalysisMode -cppr both
```

The `-cppr` parameter removes pessimism from clock paths that have a portion of the clock network in common between the clock source and clock destination paths. The pessimism is introduced when the timing analysis tools assume that the common path has different delay values for two different paths in case of on-chip variation.

- Enabling Accurate CPPR Analysis When Incremental Delays are Present

To enable accurate CPPR analysis in the presence of incremental delays, set the following

variable:

```
set_global timing_enable_si_cppr true
```

When this variable is set to true, the incremental delay is not included in the CPPR calculation during setup analysis, but is included in the CPPR calculation during hold analysis.

## Advanced Settings for SI Analysis

- [Multi-CPU Processing Settings](#)
- [Path Group Settings for SI Optimization](#)

### Multi-CPU Processing Settings

Innovus supports multi-threaded, distributed, and super-threaded noise analysis. Multi-CPU processing support, in general, reduces the noise analysis run time significantly.

The following command considers the multi-CPU processing settings during noise analysis:

- `optDesign -postRoute [-hold]`
- `timeDesign -postRoute [-hold]`

For information on multi-CPU processing, see [Accelerating the Design Process by Multiple-CPU Processing](#).

### Setting Up Multi-Threading for Noise Analysis

Multi-threading enables you to run noise analysis on multiple CPUs of a single host. The host machine on which you are running Innovus or a different host.

- To setup multi-threaded noise analysis on the same host, use the following command:

```
setMultiCpuUsage -localCpu <number>
```

```
optDesign -postRoute
```

- To setup multi-threaded noise analysis for on a different host, use the following set of commands:

```
setDistributeHost -rsh -add { remote_hostname }
```

```
setMultiCpuUsage -remoteHost 1 -cpuPerRemoteHost <number>
optDesign -postRoute
```

## ***Setting Up Distributed Processing for Noise Analysis***

Distributed processing enables you to divide a noise analysis job between two or more networked computers running concurrently.

- To setup distributed processing using RSH, specify the following commands:

```
setDistributeHost -rsh -add {host1 host2 host3 host4 ... hostN}
setMultiCpuUsage -remoteHost <number>
optDesign -postRoute
```

- To setup distributed processing using LSF, specify the following commands:

```
setDistributeHost -lsf -queue myLSFqueue
setMultiCpuUsage -remoteHost number
optDesign -postRoute
```

When setting up distributed processing, ensure that there is a corresponding host computer for each view definition. For example, if the design has four view definitions, divide these between four host computers.

## ***Setting Up Super-Threaded Noise Analysis***

Super-threading enables you to run a noise analysis job on both distributed processing and multi-threaded modes; that is, two or more networked computers, each with multiple processors, can be called to concurrently run noise analysis.

- To setup super-threading using RSH, use the following commands:

```
setDistributeHost -rsh -add {host1 host2 host3 .... hostN}
setMultiCpuUsage -remoteHost number \
-cpuPerRemoteHost number
optDesign -postRoute
```

- To setup super-threading using LSF, use the following commands:

```
setDistributeHost -lsf -queue myLSFqueue  
setMultiCpuUsage -remoteHost number \  
-cpuPerRemoteHost number  
optDesign -postRoute
```

## **Examples of Setting Up Multi-CPU Processing**

- The following settings distribute the analysis & optimization on host1 and host2 machines using RSH:

```
setDistributeHost -rsh -add { host1 host2 }  
setMultiCpuUsage -remoteHost 2  
optDesign -postRoute
```

- The following multi-threading settings distribute the analysis & optimization on eight threads on the local machine:

```
setMultiCpuUsage -localCpu 8  
optDesign -postRoute
```

- The following settings distribute the analysis & optimization on the 5 specified host machines using RSH:

```
setDistributeHost -rsh -add { host1 host2 host3 host4 host5 }  
setMultiCpuUsage -remoteHost 5 -localCpu 4  
optDesign -postRoute
```

**Note:** When used together, the `-remoteHost` parameter is given preference over the `-localCpu` parameter if the number of remote hosts specified is more than the number of CPUs specified with the `-localCpu` parameter. The `-localCpu` parameter is given preference if the number of CPUs specified with the parameter is higher than the number of remote hosts. In this case, the `-localCpu` parameter is ignored. If you use multiple hosts and multiple threads at the same time during SI optimization, use the `-remoteHost` and `-cpuPerRemoteHost` parameters instead.

- The following super-threading settings distribute the analysis & optimization on host1 and host2 machines using RSH, and run eight threads on each:

```
setDistributeHost -rsh -add { host1 host2 }
setMultiCpuUsage -remoteHost 2 \
-cpuPerRemoteHost 8
optDesign -postRoute
```

**Note:** In this case, a total of 16 CPUs will be used.

## Path Group Settings for SI Optimization

The SI optimization flow accounts for any path groups specified by the user. If none are set it uses the default groups of reg2reg, reg2cgate & default (all other paths).

This enables you to take full advantage of all path group capabilities during SI optimization.

Examples include:

- The `-slackAdjustment` parameter of the [setPathGroupOptions](#) command for slack adjustment of any group.
- The `-weight` parameter of the [setPathGroupOptions](#) command to control the relative optimization priority of each group.

To create, modify, and report path groups, use the following commands:

- [group\\_path](#)
- [reset\\_path\\_group](#)
- [report\\_path\\_groups](#)
- [createBasicPathGroups](#)
- [setPathGroupOptions](#)
- [resetPathGroupOptions](#)
- [reportPathGroupOptions](#)

## Example of Setting Up Innovus for SI Analysis

The following example script provides a summary of the settings that are required for performing SI analysis:

```
## Extraction Settings ##

setDesignMode -process 20
```

```
setExtractRCMode -engine postRoute -effortLevel high

## SI Settings (unless otherwise recommended it is best to use the defaults & skip
this) ##

setSIMode -individual_attacker_threshold 0.01

## STA settings ##

setAnalysisMode -analysisType onChipVariation -cppr both

## CPPR Removal for Incremental delays (optional)

set_global timing_enable_si_cppr true

## Decide what Timing Tool you will be signing off on & set the engine to match (this is the default)

setSIMode -analysisType aae

## Ensure SI Delay Cal is on (this is the default)

setDelayCalMode -SIAware true
```

## Preventing Crosstalk Violations

To prevent crosstalk violations, here are some recommendations:

1. Set reasonable max transition thresholds and ensure they are fixed before detail routing.
2. Fix transition time violations on the Clock tree aggressively.
3. Shield the clock tree root to prevent Clock SI.
4. Run timing and signal integrity driven routing using routeDesign (this is on by default:  
`setNanoRouteMode -routeWithTimingDriven true -routeWithSiDriven true`). You can also use  
the `setNanoRouteMode -routeSiEffort` command.

See *SI Prevention Techniques for PreRoute using Innovus System* application note for more advanced experiments.

## Fixing Crosstalk Violations

- [Data Preparation](#)
- [Using optDesign to Fix Setup Violations with Crosstalk Effects](#)
- [Using optDesign to Fix Hold Violations with Crosstalk Effects](#)

- [Using optDesign to Fix Transition Time Violations with Crosstalk Effects](#)

## Data Preparation

### Extraction:

There are 4 options for postRoute extraction: detailed extraction, Turbo QRC, Integrated QRC & standalone QRC sign-off extraction. To correlate native extraction results with QRC sign-off extraction, compare the SPEF files from basic and sign-off extraction to generate the new scaling factors for total capacitance, cross-coupling capacitance, and resistance. Using these scaling factors, the native extraction results are closer to the sign-off extraction results, while only taking a fraction of the run time required for sign-off extraction. For more information, see "[Correlating Native Extraction With Sign-Off Extraction](#)".

## Using optDesign to Fix Setup Violations with Effects

By default, using the [optDesign](#) -postRoute command will fix both setup base and SI timing at the same time. It will also correct glitch violations caused by incremental delays, which occur due to coupling capacitance.

- If you want to run the base timing optimization alone (without the crosstalk incremental delay), set the following:

```
setDelayCalMode -SIAware false
```

- If you want to run the SI delay Optimization but turn off SI Glitch Optimization, set the following :

```
setOptMode -fixGlitch false
```

- If you want to turn on the SI Slew Optimization, set the following:

```
setOptMode -fixSISlew true
```

For best results, set up Innovus SI analysis to match Sign-off Tool analysis before using the [optDesign](#) command.

## Using RC Data Generated by an External Tool for SI Fixing

To perform Si fixing, load the RC data generated by an external tool by using the [spefIn](#) command in Innovus. RC data needs to be regenerated using the third-party tool to perform SI analysis and generate reports after optimization is complete.

Ensure that you use the [spefIn](#) command to load the parasitic data for each corner.

## Using SDF Data Generated by an External Tool for SI Fixing

You can use the SDF data generated by an external tool in Innovus to do limited SI fixing. SDF data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

### Fixing Setup Violations Using External SDF

Use the [read\\_sdf](#) command to load an SDF file for each view. For each view, two SDFs are required from the external tool; one with base timing only and the other full timing with base and SI Timing.

- Use the following commands to load separate SDF files for two setup views:

```
setDelayCalMode -SIAware false
```

```
read_sdf -view <viewname1> -reset_preexisting_derate <view1_base_timing>.sdf  
read_sdf -view <viewname1> -reset_preexisting_derate -infer_delta_delay  
<view1_full_timing>.sdf
```

```
read_sdf -view <viewname2> -reset_preexisting_derate <view2_base_timing>.sdf  
read_sdf -view <viewname2> -reset_preexisting_derate -infer_delta_delay  
<view2_full_timing>.sdf
```

- Enable the external SDF based SI setup fixing flow:

```
optDesign -postRoute -useSDF
```

## Using optDesign to Fix Hold Violations with Crosstalk Effects

By default using the [optDesign](#) -postRoute -hold command will fix both hold base and SI timing at the same time. If you want to run the base hold timing optimization alone (without the crosstalk incremental delay), set the following:

```
setDelayCalMode -SIAware false
```

## Using RC Data Generated by an External Tool for SI Hold Fixing

You can load the RC data generated by an external tool by using the [spefIn](#) command in Innovus to do limited SI hold fixing. RC data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

Note: Ensure that you use the [spefIn](#) command to load the parasitic data for each corner.

## Using SDF Data Generated by an External Tool for SI Hold Fixing

You can also use SDF data generated by an external tool in Innovus to do limited SI hold fixing. SDF data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

## Fixing Hold Violations Using External SDF for MMMC Designs

Use the [read\\_sdf](#) command to load an SDF file for each view. For each view 2 SDF's are required from the external tool, one with base timing only, the other full timing with base and SI Timing included.

- Use the following commands to load the separate SDF files for a design with one setup view and one hold view:

```
setDelayCalMode -SIAware false
```

```
read_sdf -view <setupview1> -reset_preeexisting_derate <setupview1_base_timing>.sdf  
read_sdf -view <setupview1> -reset_preeexisting_derate -infer_delta_delay  
<setupview1_full_timing>.sdf
```

```
read_sdf -view <holdview1> -reset_preeexisting_derate <holdview1_base_timing>.sdf  
read_sdf -view <holdview1> -reset_preeexisting_derate -infer_delta_delay  
<holdview1_full_timing>.sdf
```

- Enable the external SDF based SI fixing flow:

```
optDesign -postRoute -hold -useSDF
```

## Using optDesign to Fix Transition Time Violations with Crosstalk Effects

To fix SI induced maximum transition violations, set the following:

```
setOptMode -fixSISlew true  
optDesign -postRoute
```

## Performing XILM-Based SI Analysis and Fixing

The model-based flow in Innovus involves generating timing accurate interface logic models (ILMs) for hierarchical blocks. To perform SI analysis, the model data generation process should account for the characterized noise library of the blocks, the timing window information of non-ILM timing path nets inside the blocks, and cross-coupling information. This data, which is an extension to ILMs, is called eXtended Interface Logic Model (XILM). An XILM is built with:

- Cells and RC network (including cross-coupling data) connected from each I/O pin to and from the first latch or flip-flop.
- eXtended Timing Window Format (XTWF) file that contains timing window and slew information of non-ILM timing paths inside the block, which might be aggressors to the nets on the ILM timing path or the top-level nets.

**Note:** For more information on XILM-based SI analysis, see the [Top-level Timing Closure Methodologies](#) chapter of the *Innovus User Guide*.

# Verification Capabilities

---

- Identifying and Viewing Violations
- Verifying Well Pins and Bias Pins
- Integration with LPA and CCP

# Identifying and Viewing Violations

- [Overview](#)
- [Interrupting Verification](#)
- [Verifying Connectivity](#)
- [Verifying Metal Density](#)
- [Verifying Geometry](#)
- [Verifying Process Antennas](#)
- [Verifying Well-Process-Antenna Violations](#)
- [Verifying End Cap Violations](#)
- [Verifying Maximum Floating Area Violations](#)
- [Verifying AC Limit](#)
- [Verifying Isolated Cuts](#)
- [Verifying Tie Cells](#)
- [Viewing Violations With the Violation Browser](#)
- [Saving Violations](#)
- [Clearing Violations](#)

## Overview

The verification commands in the Innovus™ Implementation System check and report on the following types of violations:

- Connectivity: Checks for opens, unconnected wires (geometrical antennas), loops, and partial routing.  
Verify the connectivity of the design after the following design step:
  - Detailed routing
    - For more information, see
    - [Verifying Connectivity](#)
  - `verifyConnectivity` in the "Verify Commands" chapter of the *Innovus Text Command Reference*
- Metal density: Checks that the metal density of the metal layers is within the minimum and maximum metal density values specified by the LEF file or the `setMetalFill` command. Also checks the density of the cut layers.

Verify the metal density after the following design step:

- Inserting metal fill

For more information, see [verifyMetalDensity](#) and [verifyCutDensity](#) in the "Verify Commands" chapter of the *Innovus Text Command Reference*.

- Geometry: Checks the physical layout of the design, including width, length, spacing, area, overlap, enclosure, wire extension, and via stacking violations. If you modify or edit any part of the design, run [verifyGeometry](#) to make sure the design is still DRC clean.

Verify the geometry of the design after the following design steps:

- Placement
- Power routing
- Detailed routing
- Wire editing

For more information, see

- [Verifying Geometry](#)
- [verifyGeometry](#) in the "Verify Commands" chapter of the *Innovus Text Command Reference*. This command is used for 28nm and above designs.
- [verify\\_drc](#) in the "Verify Commands" chapter of the *Innovus Text Command Reference*. This command is used for 20nm and below DRC rules.

- Process antennas and unconnected metal segments (floating areas): Checks the charge that builds up on pins caused by routing that does not have a discharge path to a gate. The [verifyProcessAntenna](#) command checks for pin routing that violates the maximum antenna charge for the pins, and reports violations on pins that have an antenna ratio larger than the maximum allowed antenna ratio specified for the routing layer.

The [verifyProcessAntenna](#) command also checks for unconnected metal segments that violate the maximum area specified in the LEF file. An unconnected (floating) metal segment is a segment that is not connected to diffusion (or a polysilicon gate) through the same layer or a lower layer.

Verify process antenna and maximum floating area violations after the following design step:

- Detailed routing

For more information, see [verifyProcessAntenna](#) in the "Verify Commands" chapter of the *Innovus Text Command Reference*.

- AC limit: Checks for AC current violations on signal nets.

Verify the AC limit after the following design step:

- Detailed routing

For more information, see [verifyACLimit](#) in the "Verify Commands" chapter of the *Innovus Text Command Reference*.

- Lithography hotspots: The software can interpret hotspot interchange format (HIF) files.

For more information, see [loadViolationReport](#) in the "Verify Commands" chapter of the *Innovus Text Command Reference*.

- Placement:

For more information on the types of violations, see [checkPlace](#) in the "Placement Commands" chapter of the *Innovus Text Command Reference*.

- Violation markers

You can use text commands or GUI forms to check the violations and create the reports.

You create violation markers with the Innovus commands, or import markers from another verification tool, such as Assura™ or Calibre, and view the markers with the Violation Browser. The Innovus software saves the markers with the database.

For more information, see the [Viewing Violations With the Violation Browser](#) section.

## Interrupting Verification

The following verification commands support "Interrupt" (Ctrl+C):

- verifyACLimit
- verifyConnectivity
- verifyGeometry
- verifyMetalDensity
- verifyPowerVia
- verifyProcessAntenna
- verify\_drc

If you press `Ctrl+C` while one of the above verification commands is being executed, the verification process stops and the software displays the Interrupt menu. For information on the Interrupt menu, see "Interrupting the Software" in the [Getting Started](#) chapter.

## Verifying Connectivity

Verify the connectivity of your design to detect and report conditions such as opens, unconnected pins, dangling wires, loops, and partial routing. You can use the command to create violation markers in the design window and generate a violation report. There is no database impact from using this command unless you save the design, which saves the violation markers.

For regular wires, the Innovus software checks connectivity by using the center line of the wire segments and center of the vias. For special wires, the command checks the whole geometry. If a via or wire is slightly offset from where it should be, the software reports an error.

The software also detects connectivity loops based on the end points of a regular wire segment center line or the center of a via. It reports geometry loop violations.

**Note:** The Verify Connectivity feature now uses [setMultiCpuUsage](#) and other multi-CPU commands for multi-threading.

For more information, see the [Multiple-CPU Processing Commands](#) chapter in the *Innovus Text Command Reference*.

## Before You Begin

Before you verify connectivity, the following conditions must be met:

- The design must be routed.
- The design must be loaded into the current Innovus session.

## Types of Connectivity Violations Reported

- Antennas (Dangling wires)  
Unconnected wires (dangling wires). For more information, see the [Types of Antenna Violations Reported](#) section.
- Opens  
Parts of nets, such as wires or pins, that are connected to each other but are missing a connection to the net as a whole. Marks each part of a net that is missing a connection as an open and displays a violation marker between the parts.

Violation markers for opens are displayed as polygons that include all wires, pins, and vias that connect to an island.

By default, `verifyConnectivity` checks the connectivity on masterslice layers. If the `-noSoftPGConnect` option is specified, connectivity on these layers is ignored and checking of soft Power/Ground connects is disabled.

- Loops
  - Unconnected pins
- Pins that are not connected to any other objects

**Note:** In releases prior to 7.1, `verifyConnectivity` marked nets with connected pins but without any wiring as unrouted nets. `verifyConnectivity` no longer marks these nets as unrouted, so they do not cause violations.

**Note:** From the 10.1 release, `verifyConnectivity` recognizes the PG pin with DIRECTION OUT or CLASS CORE/BUMP as a strong connection. So, if you define PG PIN DIRECTION as OUTPUT or any PORT with CLASS CORE/BUMP, `verifyConnectivity` would treat that PIN as strongly connected.

For more information, see "*Verify Connectivity*" in the [Verify Menu](#) chapter of *Innovus Menu Reference*.

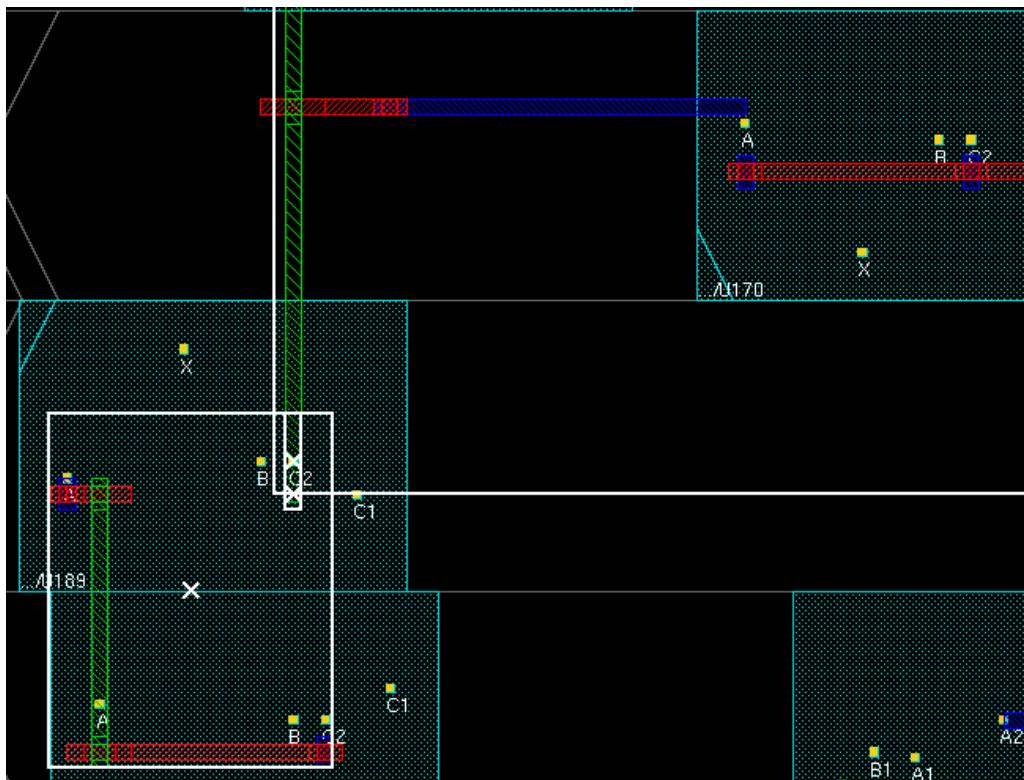
## Results

After verifying connectivity, you can use information in the violation report to repair connectivity violations. You can use the Violation Browser for interactive viewing and highlighting of violation markers. You can see incremental results in the Violation Browser.

## Debugging Opens Interactively

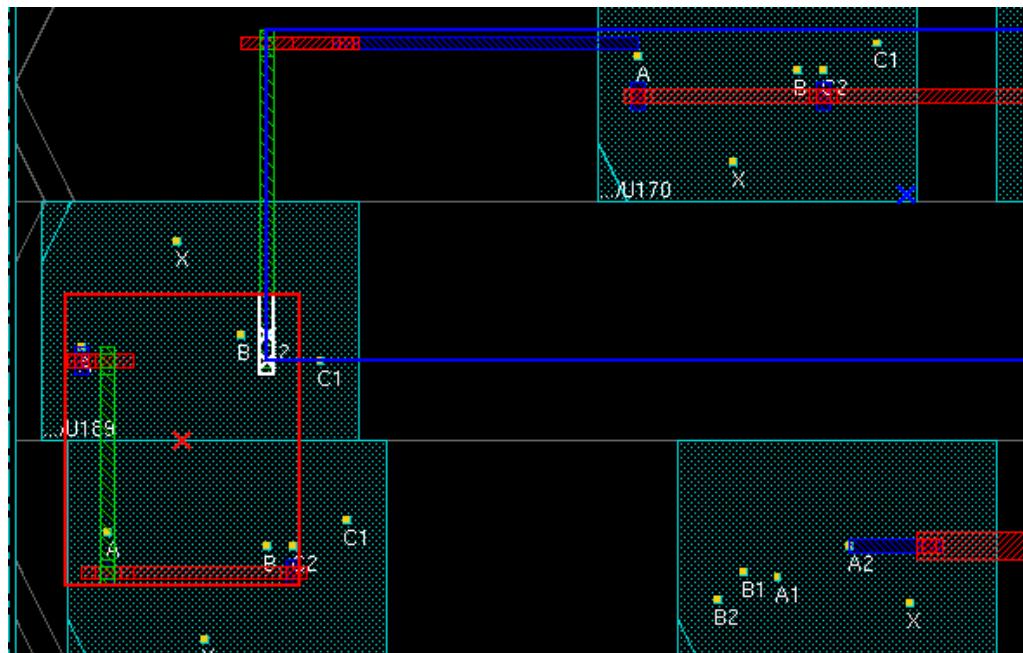
Debugging an open net can be difficult, especially when the island is big. An island is a broken segment of the net including pin, wires, and via. The open markers may overlap each other, making it hard to find the island boundary. Starting from the 14.1 release, you can interactively colorize open net markers. This makes it easier to identify the open island boundary.

Suppose your design has multiple Open violations. Initially, all violations are marked in white in the main window by default.



Follow the steps below to use the colorize feature to identify different open island boundaries:

1. In the Violation Browser, click on an Open violation to auto-zoom to the open marker location.
2. Click the open net marker to which you have zoomed into in the main window. Each open net marker is then automatically assigned with a unique color. In this case, the open net markers are colored red, white, and blue.



**Note:** Clicking any colorized marker again to remove the colors.

3. Press the F12 key or select *Edit > Dim Background* to dim the background and view the open more clearly.  
This helps you identify the cause of the open. Typical causes are missing wires, missing vias, or unconnected pins. You can then take appropriate steps to debug the open net violation.

## Verifying Metal Density

Verify the metal density of the design for each routing layer, to ensure that it is within the minimum and maximum density values specified in the LEF file or by the `setMetalFill` command.

### Before You Begin

Before you verify metal density, the following conditions must be met:

- Metal density values must be specified in the LEF file or by the `setMetalFill` command. The `setMetalFill` setting will overwrite the LEF rules.
- The design must be detail routed.
- The design must be loaded into the current Innovus session.

### Metal Density Statements in the LEF File

The following statements in the Layer (Routing) section of the LEF file define the minimum and maximum metal density and how to analyze the density.

- `MINIMUMDENSITY`
- `MAXIMUMDENSITY`
- `DENSITYCHECKWINDOW`
- `DENSITYCHECKSTEP`
- `FILLACTIVESPACING`

For more information, see the [LEF Syntax](#) chapter in the *LEF/DEF Language Reference*.

## Results

The verification process generates a report file containing information about the metal density that is not within the minimum and maximum density range.

## Verifying Metal Density in Multi-Thread Mode

You can accelerate metal density checking by running the software in multi-thread mode. To do so, run the `setMultiCpuUsage` command before running `verifyMetalDensity`. For example:

```
setMultiCpuUsage -localCPU 4
verifyMetalDensity
```

## Related Topics

- [Accelerating the Design Process By Using Multiple-CPU Processing](#)
- [Multiple-CPU Processing Commands](#) chapter in the *Innovus Text Command Reference*

## Verifying Geometry

Verify the physical layout of the design by checking the width, spacing, internal geometry, and other characteristics of objects. Use the `verifyGeometry` command to specify the checks to perform, disable checking, and set limits for errors and warnings to report.

The disable feature is useful when false violations arise because of discrepancies in the way mask-level data is presented. For example, cell internal obstructions and pins might be represented in a way that causes the verifier to report design rule violations that do not exist in the mask-level layout.

Verify geometry at the following stages in the design flow:

- After placing the design.
- After adding power stripes and rings and running power routing.
- After running detailed routing.

**!** The `verifyGeometry` command does not support 20nm and below advanced rules. Use `verify_drc` instead. `verify_drc` settings can also be specified through `set_verify_drc_mode`.

## Before You Begin

- Ensure the following LEF statements are specified:
  - `CLEARANCEMEASURE`
  - `USEMINSPACING` statements for obstructions and pinsFor more information, see the [LEF Syntax](#) chapter in the *LEF/DEF Language Reference*.
- If you plan to run `verifyGeometry` in multiple-CPU processing mode, use the Innovus multiple-CPU commands or select the appropriate options on the Multiple CPU Processing form. For more information, see the [Verifying Geometry in Multi-Thread Mode](#) section.
- Route the design.
- Set global parameters for `verifyGeometry` using the `setVerifyGeometryMode` command.

**Note:** You can check current global settings for `verifyGeometry` using the `getVerifyGeometryMode` command.

## Verify Geometry Statements in the LEF File

The following statements in the LEF file can be used to define how to verify geometry.

- ADJACENTCUTS
- CORNERFILLSPACING
- CUTCLASS SPACINGTABLE
- ENCLOSURE
- ENCLOSUREEDGE
- ENDOFLINE
- EOLENCLOSURE
- EOLKEEPOUT
- EOLSPACING
- EXCEPTRECTANGLE
- JOGTOJOGSPACING
- MANUFACTURINGGRID
- MAXVIASTACK
- MINSTEP
- MINWIDTH
- OPPOSITEEOLSPACING
- PINMASK
- SPACING
- VOLTAGESPACING
- WIDTH

For more information, see the [LEF Syntax](#) chapter in the *LEF/DEF Language Reference*.

## Verifying Geometry in Multi-Thread Mode

You can accelerate geometry checking by running the software in multi-thread mode. Use one of the following methods:

- On the text command line:

Run the following command before running `verifyGeometry`:

```
setMultiCpuUsage
```

For example,

```
setMultiCpuUsage -localCPU 4  
verifyGeometry
```

- In the GUI:

- On the Verify Geometry - Advanced page, click the *Set Multiple CPU* button to open the Multiple CPU Processing form.
- On the Multiple CPU Processing form, specify the number of local CPUs.
- Optionally, select *Release License*.
- Click *OK* to close the Multiple CPU Processing form and return to the Verify Geometry form.

When you return to the Verify Geometry form, the *Number of Local CPU(s)* option in this form is updated with the value you specified on the Multiple CPU Processing form.

- Run Verify Geometry.

## Related Topics

- [Accelerating the Design Process By Using Multiple-CPU Processing](#)
- [Multiple-CPU Processing Commands](#) chapter in the *Innovus Text Command Reference*
- [Verify Geometry - Advanced](#) in the [Verify Menu](#) chapter of the *Innovus Menu Reference*

## Spacing Violation Checks

- `verifyGeometry` uses the minimum dimension of an object to check for spacing violations. The minimum dimension is the width of the object.
- The command does not detect objects with width greater than `WIDTH` and length greater than `LENGTH` that exist within a distance (`WITHIN`) greater than 10  $\mu\text{m}$  for the `MINIMUMCUT` check in the LEF file.
- The command categorizes spacing violations as `SameNet`, `NonDefault`, and `ParallelRun` violations. If it finds a violation caused by a blockage between two instances of different cells, it treats the violation as a `SameNet` violation because it does not belong to a net.
- The command considers `OBS CUT` layer shapes as within the same metal if they are within the same `OBS ROUTING` layer shape (the layer above or below). This avoids `-sameCellViol` flags on SPACING violations inside the cells.
- To check implant layers for violations, specify an implant rule in the LEF file. To skip implant layer checking, specify the `verifyGeometry -noImplantCheck` parameter.
- To check spacing between cut layers and metal layers, specify a cut-metal spacing rule in the LEF file. For example, the following rule triggers a check of the spacing between `CUTG1` and `MET5` layers:

```
LAYER CUTG1 TYPE CUT ;
  SPACING 0.42 ;
  SPACING 0.28 LAYER MET5 ;
END CUT G1
```

For more information, see the [LEF Syntax](#) chapter of the *LEF/DEF Language Reference*.

## Types of Antenna Violations Reported

`verifyGeometry` flags an antenna violation when it finds an unconnected wire.

- i** In the context of `verifyGeometry` and `verifyConnectivity`, antenna violations are different from process antenna violations. The `verifyProcessAntenna` command checks for process antenna violations, which are caused by pins whose process antenna ratio is larger than the maximum allowed ratio specified in the LEF file for the routing layer. For more information, see `verifyProcessAntenna` in the "Verify Commands" chapter of the *Innovus Text Command Reference*.

To avoid antenna violations, wires and nets must meet the following conditions:

- Regular wires
  - Must terminate on a pin or the center of a via.
  - If a vertical wire intersects a horizontal wire along its axis, the end of the vertical wire must be covered by the horizontal wire.
  - If the ends of a vertical wire and horizontal wire meet, they must meet at their end points.
- Regular net vias
  - Must be covered by a pin.
  - The center of the via must be the start or end point of a wire.
  - The center point of stacked vias must be coincident.
- Special wires
  - A point that is one-quarter of a wire width from the center of the ending edge of the special wire must be covered by a via, pin, or another wire on the same layer.
- Special net vias
  - The metal rectangle of the special net via must overlap with a special wire or via of the same net.

## Support for Via Rules

`verifyGeometry` uses the rules defined in the `VIARULE` section of the LEF file and the drc rules in the tech lef file to check for violations caused by vias.

- The command checks the master via only, and flags violations on only one instance of the via by default. You can run `verifyGeometry` with the `-reportAllVia` parameter to report viaCell violations for all instances.

- The command considers the content of the via during verification when checking for spacing violations.

## Results

`verifyGeometry` creates markers corresponding to geometry violations in the database. Use the [Violation Browser](#) to see the markers.

## Verifying Process Antennas

Verify process antenna violations by checking for routing that violates the maximum charge caused by the process antenna effect (PAE) on pins. The software finds violations when a pin's process antenna ratio is larger than the maximum ratio specified in the LEF file for the routing layer.

The report file lists all the violated nets and includes process antenna information. Optionally, it can also report all other nets.

## Before You Begin

Before performing process antenna verification, complete the following tasks:

- Perform signal routing.
- Ensure the antenna keywords are specified in the LEF file. For example:
  - `ANTENNAAREARATIO` for LEF layers
  - `ANTENNAGATEAREA` and `ANTENNADIFFAREA` for macro pins

**Note:** By default, when `verifyProcessAntenna` (VPA) is run on a design, the value of `AntennaInputGateArea` is added to the gate area of instances connected to the antenna cell. This means that if antenna cells are inserted by NanoRoute, VPA uses a gate area value larger than the value used in NanoRoute antenna calculation. To avoid such an optimistic analysis, you might want VPA to ignore the default `AntennaInputGateArea` for antenna cell. To do so, you can specify `ANTENNAGATEAREA 0.0` in macro pin antenna definition in LEF and set the `gateArea` of the macro's pin to 0.

For more information, see the [LEF Syntax](#) chapter in the *LEF/DEF Language Reference*.

## Verifying PAE

Checks for pin routing that violates the maximum antenna charge for the pins and reports violations on pins that have an antenna ratio larger than the maximum allowed antenna ratio specified for the routing layer. Handles PAE violations on any metal layer on flat or hierarchical designs. Uses a geometry-based approach and does not double count metal areas for vias or wires. Provides a detailed process antenna report including the metal area, diffusion area, and target ratio for each pin. The report file lists all violated nets with process antenna information. Optionally, it can also report all other nets. For more information on PAE, see the [Calculating and Fixing Process Antenna Violations](#) appendix in the *LEF/DEF Language Reference*.

## Results

After verifying process antenna violations, you can use information in the violation report to repair process antenna violations. You can use the *Tools - Violation Browser* command for interactive viewing and highlighting of violation markers.

## Sample Process Antenna Report

The following example shows a section of a detailed process antenna report file:

D1 (2)									
	U0 (BUF) A								
[1]	MET:	Area:	1.10	S.Area:	2.10	G.Area:	100.00	D.Area:	0.00
		Fact:	0.90	PAR:	0.01	Ratio:	0.10	(Area)	
		Fact:	1.00	PAR:	0.02	Ratio:	0.10	(S.Area)	
				CAR:	0.01	Ratio:	0.00	(C.Area)	
				CAR:	0.02	Ratio:	0.00	(C.S.Area)	
[1]	THO:	Area:	0.20	S.Area:	0.00	G.Area:	100.00	D.Area:	0.00
		Fact:	1.00	PAR:	0.00	Ratio:	0.00	(Area)	
				CAR:	0.00	Ratio:	0.00	(C.Area)	
[1]	WMET:	Area:	13.27	S.Area:	51.20	G.Area:	100.00	D.Area:	300.00
		Fact:	1.10	PAR:	0.15	Ratio:	2.50	(Area)	
		Fact:	0.90	PAR:	0.46	Ratio:	0.00	(S.Area)	
				CAR:	0.16	Ratio:	0.00	(C.Area)	
				CAR:	0.48	Ratio:	0.00	(C.S.Area)	

The report uses the following terms:

Area:	Metal area
-------	------------

S.Area:	Metal side area
G.Area:	Gate area
D.Area:	Diffusion area
Fact:	Metal (side) area adjusted factor
PAR:	Partial antenna ratio
CAR:	Cumulated antenna ratio
Ratio:	Target antenna ratio
(Area)	Metal area
(S.Area)	Metal side area
(C.Area)	Cumulated metal area
(C.S.Area)	Cumulated metal side area

## Verifying Well-Process-Antenna Violations

Use the `verifyWellAntenna` command to check for any `CORE` rows that have well-process-antenna violations. Well-process-antenna violations are normally caused by rows with decap cells that do not have any protecting cells in the same row. This command marks with a violation marker any such `CORE` cell. You can then write a Tcl script to put in a well-antenna cell (protection cell) next to the violation marker.

Normally, the decap cells are the only cells that need protection. However, if the standard cell library in your design has one of the `ENDCAP` cells with a well-antenna protection device built-in, and the filler, well-tap, tie-high and tie-low cells are just class `CORE` rather than with the correct `CORE` subclass, you can use the following command:

```
verifyWellAntenna -needToProtect decap* -changeToProtect wellAntEndCap -  
changeToNotProtect {filler* welltap* tie*} -report wellant.rpt
```

## Sample verifyWellAntenna Report

The following example shows a section of a detailed well-process-antenna report file:

```
#####
# Generated by: Cadence Innovus 15.10-b031_1
# OS: Linux x86_64 (Host ID rlno-leenap)
# Generated on: Fri Apr 17 12:25:27 2015
# Design: DTMF_CHIP
# Command: verifyWellAntenna -needToProtect decap* -changeToProtect...
#####
Innovus WellAntenna Verification Report
Design: DTMF_CHIP
```

```
Protects NeedToProtect Name Class [subclass]
1 0 ZLLLN550V15 CORE
1 0 ZHHHN550V15 CORE
1 0 XORSN550V15 CORE
1 0 XORLN550V15 CORE
.....
```

.....

```
0 1 EC_RTC10 ENDCAP
1 0 EC_RTC CORE
1 0 EC_RTE10 CORE
1 0 EC_RTE CORE
1 0 EC_LTE10 CORE
1 0 EC_LTE CORE
1 0 EC_BE16 CORE
1 0 EC_BE8 CORE
1 0 EC_BE4 CORE
1 0 EC_BE2 CORE
1 0 EC_BE1 CORE
1 0 EC_BE CORE
0 1 EC_TE16 CORE [FEEDTHRU]
0 1 EC_TE8 CORE [TIEHIGH]
0 1 EC_TE4 CORE [TIELOW]
0 1 EC_TE2 CORE [SPACER]
0 1 EC_TE1 CORE [ANTENNACELL]
1 0 EC_TE CORE
1 0 EC_LE10 CORE
1 0 EC_RE10 CORE
1 0 EC_LE CORE
1 0 EC_RE CORE
1 0 BCMF001N550 CORE
```

Here, cells like EC\_TE16 CORE [FEEDTHRU] are marked as need to protect cells while cells like EC\_LE10 CORE are marked as protect cells.

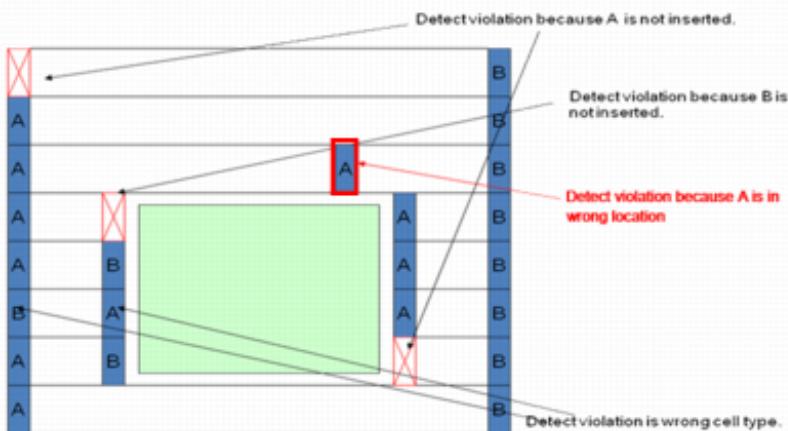
## Verifying End Cap Violations

Use the `verifyEndCap` command to verify whether pre/post cap cells are inserted correctly. By default, the command marks the beginning/end of each site row where specified cap cell is not inserted. The candidate cell lists are specified by `setEndCapMode`. If no pre/post cap cell lists are specified in `setEndCapMode`, `verifyEndCap` ignores the check.

You can also use `verifyEndCap` with the `-wrongLocation` option to check whether the cap cells are inserted in the right location (any location except the beginning/end of the row and the boundary of the design/ block design).

In the following example, cell A has been specified as pre cap cell and cell B has been specified as post cap cell using `setEndCapMode`. As shown in the diagram, `verifyEndCap` marks the following as violations:

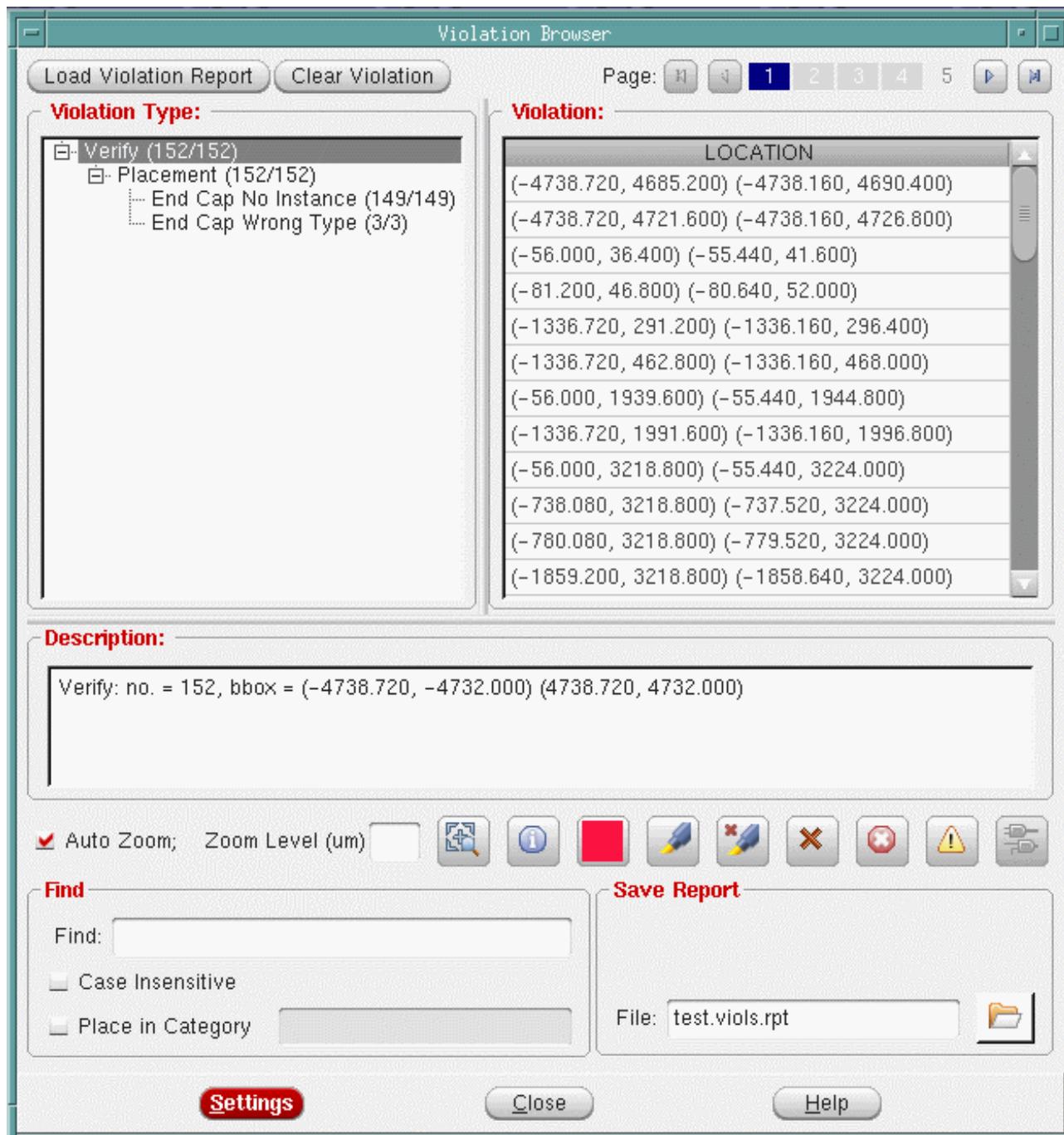
- Specified cap cell is missing. In the diagram, these locations where cap cells were expected but are missing are marked with a red cross enclosed in a rectangle .
- Inserted cap cell is of wrong type, that is cell B is inserted where A is expected and vice versa.
- Cap cell is inserted in wrong location. In the diagram, cell A is inserted in the wrong location in the instance highlighted in red.



You can also use the `verifyEndCap` command to check triple well insertions. Triple well technology is used to isolate p-well from substrate using a deep n-well. `verifyEndCap` marks any location where the specified well cell is not inserted with a violation marker. It only checks the cell lists specified in `setEndCapMode`.

## Results

After verifying end cap violations, you can use information in the violation report to repair such violations. You can use the *Tools - Violation Browser* command for interactive viewing and highlighting of violation markers.



## Sample Verify End Cap Report

The following example shows a section of a verify end cap report file:

```
#####
# Generated by: Cadence Innovus 15.10-b031_1
# OS: Linux x86_64 (Host ID rlno-leenap)
# Generated on: Fri Apr 17 12:25:27 2015
# Design: DTMF_CHIP
# Command: verifyEndCap
#####

Innovus EndCap Verification Report
Design: DTMF_CHIP

Endcap instance is not found at (-4738.720 4726.800 -4738.160 4732.000) on row ROW_0
should: RightEdge

Endcap instance is not found at (-2523.360 4726.800 -2522.800 4732.000) on row ROW_0
should: LeftEdge

Endcap instance is not found at (-2411.920 4726.800 -2411.360 4732.000) on row ROW_0
should: RightEdge

Endcap instance is not found at (-56.000 4726.800 -55.440 4732.000) on row ROW_0
should: LeftEdge

Endcap instance is not found at (55.440 4726.800 56.000 4732.000) on row ROW_0
should: RightEdge

Endcap instance is not found at (2411.360 4726.800 2411.920 4732.000) on row ROW_0
should: LeftEdge
.....
.....
Endcap instance is not found at (55.440 -4732.000 56.000 -4726.800) on row ROW_1819
should: RightEdge

Endcap instance is not found at (2411.360 -4732.000 2411.920 -4726.800) on row ROW_1819
should: LeftEdge

Endcap instance is not found at (2522.800 -4732.000 2523.360 -4726.800) on row ROW_1819
```

should: RightEdge

Endcap instance is not found at (4738.160 -4732.000 4738.720 -4726.800) on row ROW\_1819

should: LeftEdge

## Verifying Maximum Floating Area Violations

Verify maximum floating area violations (unconnected metal segments whose area is greater than the maximum area specified in the LEF file) by using the `verifyProcessAntenna` command. The Innovus software checks for maximum floating area violations by default when you run this command. For more information, see `verifyProcessAntenna` in the *Innovus Text Command Reference*.

The LEF 5.6 property `MAXFLOATINGAREA` specifies the maximum area. The following global properties are also associated with this property:

- `GATEISGROUND`  
Does not check metal layer connected to a polysilicon gate.
- `CONNECTED`  
Checks the sum of areas on the same metal that are connected through a lower metal layer.

For more information, see "[Defining Routing Layer Properties to Create 45 nm and 65 nm Rules](#)" in the "LEF Syntax" chapter of the *LEF/DEF Language Reference*.

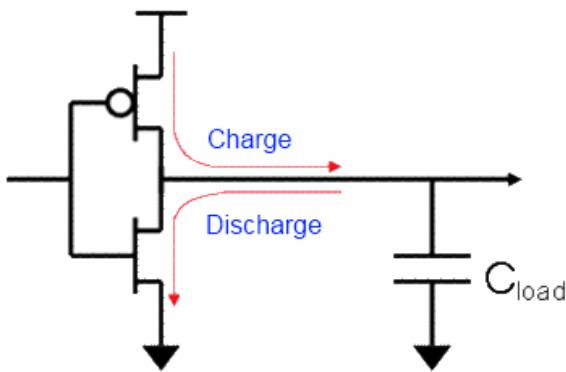
**Note:** To skip maximum floating area violation verification, but run process antenna verification, type the following command:

```
verifyProcessAntenna -noMaxFloatingArea
```

# Verifying AC Limit

## Overview

The charged particles of conductor in an electric field give up kinetic energy when they collide with atomic ions. The increase in this kinetic energy of the ions manifests itself as heat and a rise in temperature. Wire self-heating or Joule Heating describes a phenomenon where high AC currents flowing through the resistive interconnects causes extreme temperature. The following diagram illustrates signal net experiencing AC current as load capacitance is charged and discharged:



To prevent wire self-heating or AC signal electromigration (EM), signal interconnects should be analyzed for their AC current carrying capacity and measured against the AC current limits specified by foundry.

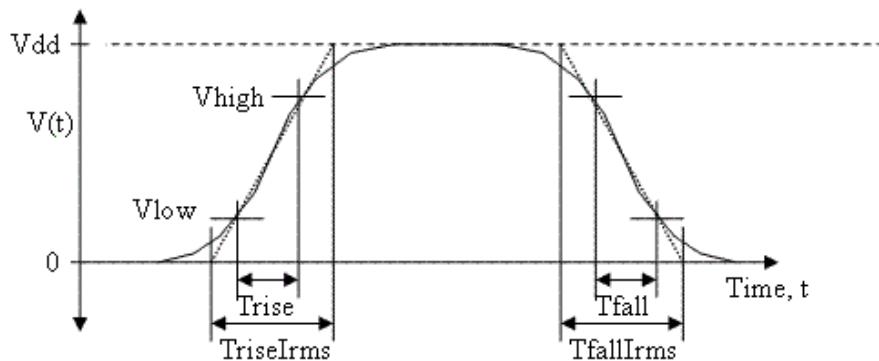
In previous versions, AC current density limits could be defined only in the Technology LEF file and only root-mean-square (RMS) current limit check was supported. From Innovus 11.1 onwards, peak AC current and average AC current limit check are also supported in addition to RMS current limit checks. Peak AC current and average AC limits must be defined in the QRC Technology file. However, RMS current limits can be defined in both Technology LEF and QRC Technology files.

To check peak AC current and average AC current, you must choose delay calculator Advanced Analysis Engine (AAE). To achieve more accurate results, Cadence recommends that you use Effective Current Source Models (ECSM) timing library.

## Calculating $I_{rms}$ Waveform

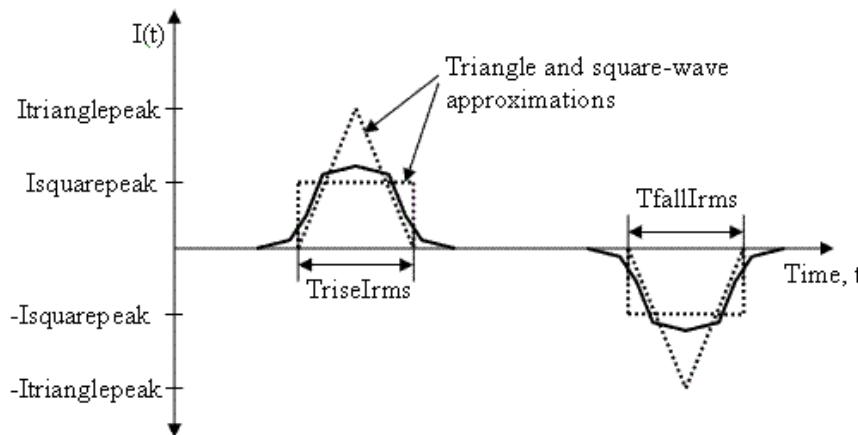
The software estimates the current waveform from the worst-case (fastest) transition time ( $T_{rise}$ ,  $T_{fall}$ ) given by the delay calculator.

The following diagram shows the  $V(t)$  square waveform for a given net:



$Trise$  and  $Tfall$  are computed by normal delay calculation between  $v_{high}$  and  $v_{low}$  thresholds.

The following diagram shows the associated  $I(t)$  waveform with a square and triangle wave approximation super-imposed:



$TriseIrms$  and  $TfallIrms$  are linear projections of the slew for a full transition from 0 to  $v_{dd}$ , or  $v_{dd}$  to 0, respectively.

The choice of triangle and square waveform depends on you. However, if it is based upon empirical data square waveform, it gives the best accuracy, and is used for current estimation by the software.

If

$C_{net}$  = total capacitance on the net

$v_{dd}$  = power supply voltage

$v_{low}$  = 20% of  $v_{dd}$

$v_{high}$  = 80% of  $v_{dd}$

then

$$TriseIrms = Trise / (.8 - .2) = Trise * 1.67$$

$$TfallIrms = Trise / (.8 - .2) = Tfall * 1.67$$

and, in order to match the total current for each transition, we have:

$$Itrianglepeak(rise) = 2 * Cnet * Vdd / TriseIrms$$

$$Itrianglepeak(fall) = 2 * Cnet * Vdd / TfallIrms$$

$$Isquarepeak(rise) = Cnet * Vdd / TriseIrms$$

$$Isquarepeak(fall) = Cnet * Vdd / TfallIrms$$

Then, doing the integral for  $I_{rms}$  for a triangle waveform approximation gives:

$$I_{rms(triangle)} = \sqrt{\frac{4S}{3T_{sw}} \left| \frac{1}{T_{riseIrms}} - \frac{1}{T_{fallIrms}} \right|}$$

where, S = Switching Factor. As charging and discharging the loading capacitor makes one switching cycle, so switching factor is 1.0 in one period for clock net, and for a signal net, you can use `verifyACLimit -toggle` to specify this value.  $T_{sw}$  = Period of one switching cycle (rising + falling). And the effective frequency  $F_{eff} = S/T_{sw}$ .

The difference is a constant  $\sqrt{4/3} = 15\%$ . You can scale the  $I_{rms}$  value during analysis (using the `-scaleCurrent` option) if you prefer the triangle approximation. Spice analysis of some typical 90nm cells shows a square-wave approximation which gives the best correlation.

The  $I_{rms}$  accuracy depends on the delay calculator and delay models used, which is controlled by the `setDelayCalMode` -engine option and contents of the .lib files. `verifyACLimit` with `setDelayCalMode` -engine feDC, will use a square-wave current waveform (current is constant for the slew-time extrapolated to a 100% voltage transition) which is normally slightly pessimistic. For long wires with large resistance, it has been measured up to 20-30% worse than Spice.

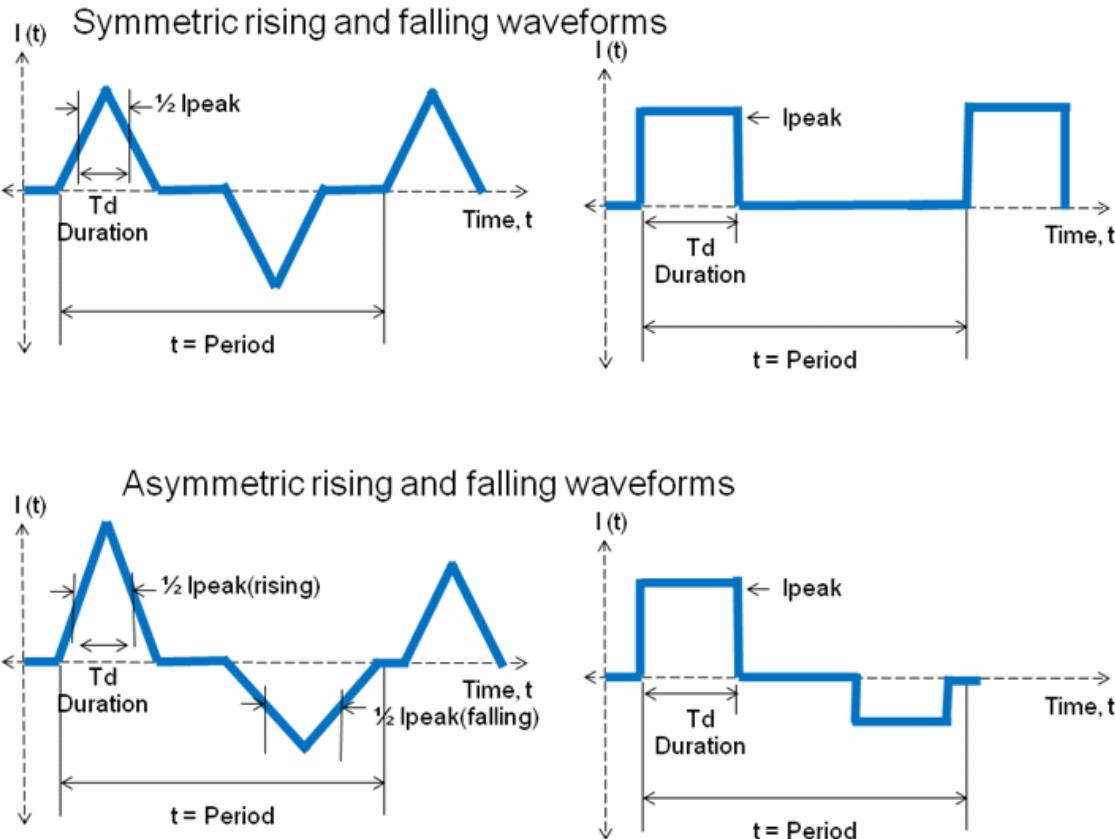
**Note:** For increased accuracy, you should use either SignalStorm® or AAE delay calculator instead of the default feDC engine to calculate  $I_{rms}$ . To enable SignalStorm or AAE calculation, run one of the following command before running `verifyACLimit`:

```
setDelayCalMode -engine signalStorm
setDelayCalMode -engine aae
```

## Calculating $I_{peak}$ Waveform

The software estimates the peak AC current waveform from the AAE delay calculator. The measured peak AC current, used to check against  $I_{peak}$  provided by foundry, is defined as following:

$$\text{peak\_measured} = I_{peak} * \text{sq. root}[r]$$



Where,  $r$  is the duty ratio which is defined as pulse duration divided by the period.

$$r = Td/t$$

The pulse duration  $Td$  can be defined for a transition from when it reaches  $\frac{1}{2} I_{peak}$  value to when it declines to  $\frac{1}{2} I_{peak}$  value as shown in the above figures.

The effective duty ratio for asymmetric rising and falling waveforms is computed as follows:

### Duty ratio $r$ for $I_{peak}$ (rising)

Duty ratio  $r(\text{rising}) = [\text{Td}(\text{rising}) + (\text{Ipeak}(\text{falling})/\text{Ipeak}(\text{rising}))^2 * \text{Td}(\text{falling})] / t$

### Duty ratio $r$ for $\text{I}_{\text{peak}}$ (falling)

Duty ratio  $r(\text{falling}) = [\text{Td}(\text{falling}) + (\text{Ipeak}(\text{rising})/\text{Ipeak}(\text{falling}))^2 * \text{Td}(\text{rising})] / t$

Using the above duty ratios for rising and falling peak waveforms,

$$\text{Ipeak}(\text{rising}) * \text{sq.root}[r(\text{rising})] == \text{Ipeak}(\text{falling}) * \text{sq.root}[r(\text{falling})]$$

Therefore, the software can pick either of these values to compare against the  $\text{I}_{\text{peak}}$  limit provided by the foundry. If the value of the measured peak AC current is larger than the  $\text{I}_{\text{peak}}$  limit, it will be considered as violation. Currently, the  $\text{I}_{\text{peak}}$  limit has to be defined in the QRC Technology file.

According to the DRM from foundry, only signal nets with effective frequency equal or larger than the minimum effective frequency (1MHz by default) will be checked for their peak AC current. If the signal nets have an effective frequency lower than the minimum frequency, the software will issue a warning message stating that the net will be ignored for peak current check. You can use the `verifyACLimit -minPeakFreq` parameter to change the minimum effective frequency. If the duty ratio  $r$  for a signal net is equal or lower than the minimum duty ratio (0.05 by default), then the minimum duty ratio will be used. You can use the `verifyACLimit -minPeakDutyRatio` parameter to change the default value.

## Calculating $\text{I}_{\text{avg}}$ Waveform

The software uses the AAE delay calculator to calculate the average current considering the recovery factor. For a rising (positive) and falling (negative) waveform,  $\text{I}_{\text{avg}}$  is calculated using the following equation:

### Equation1 :

$$\text{I}_{\text{avg.}} = \text{abs. max}(\text{rising}, \text{falling}) - EM\_recovery\_factor * \text{abs. min}(\text{rising}, \text{falling})$$

where, the `EM_recovery_factor` is defined in the QRC Technology file. You can overwrite this value. The default value of the EM recovery factor is 1, that is,  $\text{I}_{\text{avg}}$  will be zero. If you do not specify the EM recovery factor, the software will issue a warning message when `avg` analysis is selected and no EM recovery factor is defined.

For digital CMOS circuits that are charging and discharging a fixed load, by definition,  
 $\text{abs.Iavg(rising)} = \text{abs.Iavg(falling)}$ .

$$I_{avg(falling)} = \frac{S}{T_{sw}} \int_0^{T_{sw}} I_{(falling)}(t) dt$$

$$I_{avg(rising)} = \frac{S}{T_{sw}} \int_0^{T_{sw}} I_{(rising)}(t) dt$$

where, S = Switching Factor

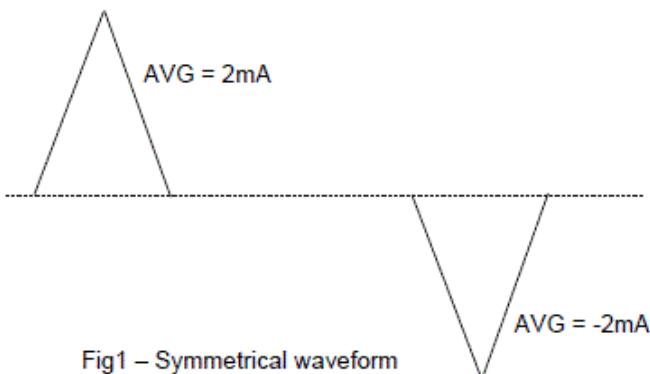
$T_{sw}$  = Period of one switching cycle (rising + falling)

Therefore, according to Equation1:

$$I_{avg} = S/T_{sw} * (C*Vdd - EM\_Recovery*C*Vdd) = F_{eff} * (C*Vdd - EM\_Recovery*C*Vdd)$$

where, C is the total loading capacitance including pin cap and Vdd is the supply voltage

Here is an example:



In this example:

- $I_{avg}(\text{rising}) = 2\text{mA}$
- $I_{avg}(\text{falling}) = -2\text{mA}$
- $em\_recovery\_factor = 0.5$

Based on the equation,  $I_{avg}$  calculation is as follows:

$$I_{avg} = 2 - 0.5*2 = 1\text{mA}$$

## Calculating Effective Frequency

For clock nets,  $T_{sw}$  is already given in the timing constraints, and  $s$ , the switching factor, is always 1.0, so the effective frequency is,  $F_{eff} = 1/T_{sw}$ .

For signal nets, there are two choices:

- You can specify the switching factor (S) for the signal nets using the `verifyACLimit -toggle` parameter. The specified switching factor will be applied to all signal nets and multiplied with the switching frequency ( $1/T_{sw}$ ) to calculate the effective frequency of the signal net. The default switching factor of 1.0 (100%) signifies that the signal net will switch twice at every period of the switching cycle (rising + falling). The switching factor of the clock net will not change with the `verifyACLimit -toggle` parameter as it always equals to 1.0. The switching frequency ( $1/T_{sw}$ ) for the net is derived from the frequency of the associated clock (fastest clock will be used if more than one is associated) using the integrated static timing analysis engine when running this software.

```
# default switching factor of 100% on data network
```

```
verifyACLimit \
```

```
-detailed \
```

```
-toggle 1.0 \
```

```
-report AC.1.0.rpt
```

- If using the global activity on all nets is too pessimistic or optimistic, you can set the effective frequency (frequency \* activity) using the TCF, VCD or FSDB files generated by netlist simulators. Optionally, you can set input and flop activities and propagate them to generate a TCF file as follows:

```
set_default_switching_activity \
```

```
-input_activity 0.2 \
```

```
-seq_activity 0.15 \
-clock_gates_output 2.0 \
-period 7.0
```

```
propagate_activity
```

```
write_tcf <design>.tcf
```

```
read_activity_file \
-format TCF \
```

```
-set_net_freq true \
dma_mac.tcf
```

```
verifyACLimit \
```

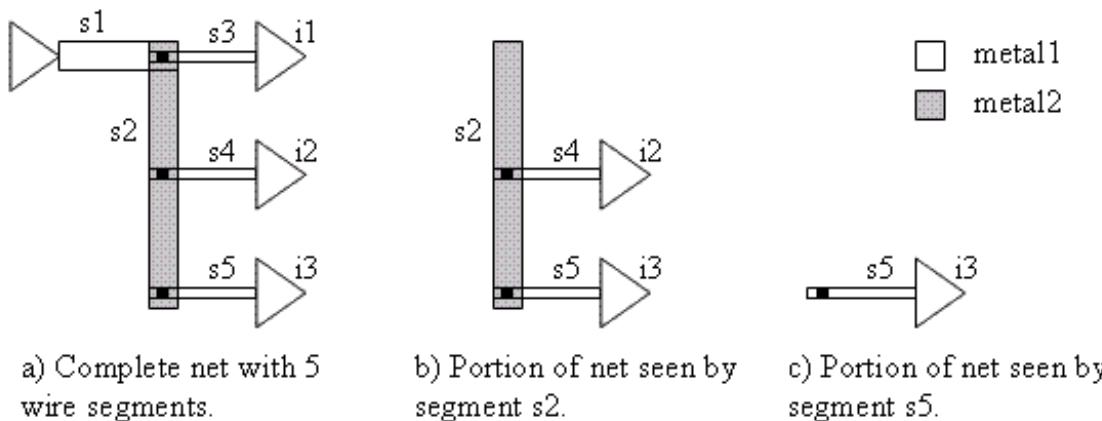
```
-method avg \
-detailed \
-avgRecovery 0.5 \
```

```
-useQRCTech \
-use_db_freq \
-report AVG.tcf.rpt
```

## Computing $I_{rms}/I_{peak}/I_{avg}$ for each Routing Segment

For a routed net, the wire widths can taper as they reach the sink pins, and the routing can branch to multiple sinks. In order to avoid false violations, a local  $I_{rms}/I_{peak}/I_{avg}$  value must be computed for each wire segment of the net.

The  $I_{rms}/I_{peak}/I_{avg}$  value of each wire segment should only be the current required to charge the capacitance of the "downstream" wire segments and sinks. The following diagram illustrates  $I_{rms}/I_{peak}/I_{avg}$  calculation on "downstream" wire segments:



In this diagram, a) shows that the wire segment  $s_1$  must carry current to charge/discharge the entire net. Therefore, the  $I_{rms}/I_{peak}/I_{avg}$  at the driver output is the correct value to check for  $s_1$ . In b), the wire segment  $s_2$  only carries current to charge/discharge  $s_2$ ,  $s_4$ ,  $s_5$  and the pin-caps of  $i_2$  and  $i_3$ . In c), wire segment  $s_5$  only carries current for  $s_5$  and the pin-cap of  $i_3$ .

The total wire capacitance ( $C_{wire}$ ) comes from extraction (SPEF), and the pin-capacitance for each instance pin from the timing libraries. The total  $I_{rms}/I_{peak}/I_{avg}$  will be calculated using the total wire capacitance ( $C_{wire}$ ). The software estimates the "downstream capacitance" that must be

charged/discharged through individual wire segments by using the total area of the "downstream wire segments" relative to the total area of the wire plus the pin-caps of the sinks.

In example b) for  $I_{rms}$ , the result is:

$C_{wire}(\text{total net})$  is already known from extraction

$C_{net}(\text{total net}) = C_{wire}(\text{total net}) + C_{pin}$  for all the sinks (driver's  $C_{pin}$  is not included)

$I_{rms}(\text{total net})$  is computed from  $C_{net}$ , the slew rate and frequency is as described earlier

$\text{Area}(\text{total net}) = \text{Area}(s_1 + s_2 + s_3 + s_4 + s_5)$

$C_{wire}(\text{downstream of } s_2) = C_{wire}(\text{total net}) * [\text{Area}(s_2 + s_4 + s_5) / \text{Area}(\text{total net})]$

$C_{net}(\text{downstream of } s_2) = C_{wire}(\text{downstream of } s_2) + C_{pin}(i_2) + C_{pin}(i_3)$

$I_{rms}$  is proportional to  $C_{net}$ , so we can estimate  $I_{rms}(s_2)$  as:

$I_{rms}(s_2) = I_{rms}(\text{total net}) * C_{net}(\text{downstream of } s_2) / C_{net}(\text{total net})$

## Checking the AC Current Limits

You can verify AC current violations on signal nets by using the `verifyACLimit` command. `verifyACLimit` checks for the following types of AC current violations on signal nets:

- Root-mean-square (RMS) current limit violations ( $I_{rms}$  violations)
- Peak current limit violations ( $I_{peak}$  violations)

In addition, `verifyACLimit` can be used for average current limit ( $I_{avg}$ ) analysis.

AC current limit violations are sometimes also referred as wire self-heat violations. Design Rule Check (DRC) manuals have these current limits to avoid over-heating a signal-net wire with too much AC current. To prevent wire self-heating or AC signal electromigration, signal interconnects should be analyzed for their AC current carrying capacity and measured against the AC current limits specified by foundry.

Use the `verifyACLimit -method` parameter to specify the waveform calculation method to be used (rms, peak, or avg).

Only the Advanced Analysis Engine (AAE) delay calculation engine supports  $I_{peak}$  and  $I_{avg}$  calculation. If you specify `-method as peak or avg` but AAE is not enabled, the tool displays the

following error message:

\*\*ERROR (ENCVAC-92) : verifyACLimit checks for -method peak or avg requires the AAE delay calculation engine. You must use `setDelayCalMode -engine aae` in Innovus, before running verifyACLimit for peak or avg checks.

**Note:** SignalStorm can still be used for  $I_{rms}$  calculations, but it does not support  $I_{peak}$  or  $I_{avg}$  calculations.

## RMS/Peak/Avg Current Limit Violations

By default, verifyACLimit only checks for  $I_{rms}$  violations. The tool calculates the  $I_{rms}$  at the driver output and compares it to the ACCURRENTDENSITY tables in the LEF file that contain the  $I_{rms}$  limits for each routing layer. It generates an error and attaches a violation marker to a net if the calculated  $I_{rms}$  for a net exceeds the ACCURRENTDENSITY  $I_{rms}$  limit for a routing layer or width used by the net.

**Note:** You can use verifyACLimit -useQrcTech to force  $I_{rms}$  checks to use the EM rules defined in the QRC technology file (.ict file) instead of the technology LEF file. If either  $I_{peak}$  or  $I_{avg}$  current limit checks are also turned on, then all checks, including  $I_{rms}$ , will use the EM rules defined in the QRC technology file.

The software support checks the RMS table for all layers. To get access to the width-dependent syntax, we must add an arbitrary single frequency value. Therefore, a typical table might look like:

```
LAYER met1
...
#RMS AC current limits for met1, at 100 C, allowing max temp change of 20 C
ACCURRENTDENSITY RMS
FREQUENCY 1 ;           #syntax needs one frequency value
WIDTH
0.15 0.3 0.6 1.2 15 ;   #values in microns from min to max width
TABLEENTRIES
10.0 9.2 8.8 8.7 8.6 ;   #mA/um for each width
END met1 ;
```

The tables are interpreted as piece-wise-linear tables indexed by width and frequency. In the

example above, a wire of width  $0.15\text{um}$  can carry  $10.0 \text{ mA}/\text{um} * .15\text{um} = 1.5\text{mA}$  of current.

The  $I_{rms}$  limits in the table can also be scaled for other factors like temperature. For example, the  $I_{rms}$  limits table should be computed for a specific "maximum allowed temperature change" ( $\text{maxTchange}$ ). The  $I_{rms}$  limits are normally proportional to the square-root of the  $\text{maxTchange}$ . Therefore, if the table used a  $\text{maxTchange}$  of 20 degrees, and you want a  $\text{maxTchange}$  of 10 degrees, you would use a scale value of  $\sqrt{10/20} = 0.71$ . The scale value would be given with the `-scaleCurrent` parameter of the `verifyACLimit` command.

The software checks the `ACCURRENTDENSITY` tables for the following conditions:

- If more than one frequency is given, `verify_AC_limit` will warn that it is only using width values from the first frequency in the table.
- There must be at least two width values, otherwise `verify_AC_limit` gives an error message and quits.
- The widths must be increasing in value, and the current limits must be increasing in value, otherwise the software gives an error message and quits.
- If no table exists for a routing layer, a warning message is given, and the limit is assumed to be infinite for that layer.

`verifyACLimit` also checks the `ACCURRENTDENSITY` tables for the following conditions and takes the following actions:

- If there is no table for a routing layer, the software gives a warning and assumes an infinite limit for the layer.
- If `PEAK` and `AVERAGE` tables are present, the software ignores them.

**Note:** When AAE is used to do  $I_{rms}$  check, only hold check is supported as the worst  $I_{rms}$  occurs in hold check.

For  $I_{rms}$ , signal EM analysis not only supports the rules defined in the technology LEF file, but also the rules defined in the QRC Technology file. But for  $I_{peak}$  and  $I_{avg}$ , only the rules defined in the QRC Technology file are supported. If no EM rule is defined in the QRC Technology file, the software will stop checking and give an error message.

These rules defined in the QRC Technology file are represented in the form of an equation, as opposed to PWL, which when processed for each wire segment individually results in a more

accurate EM limit calculation.

To perform this analysis, you must load the QRC tech file in Innovus using the `-qx_tech_file` parameter of `create_rc_corner` or `update_rc_corner`. `verifyACLimit` processes the limits defined as polynomials inside the QRC tech file to determine layer-based AC current density limits. The QRC technology file attached to the current timing view (current RC corner) is used.

The QRC tech file is generated from an ASCII input file called the ICT file. For each layer, the ICT file includes `em_model` construct that defines AC and DC current density polynomials. Here's a sample `em_model` construct for a metal layer with current limits as an equation (`EQU`):

```
em_model {  
  
    em_jmax_dc_avg EQU 1 * 1.594 * ( w - 0.003 ) jmax_factor 105 1.434 110 1.000 115 0.704  
    120 0.500 125 0.358 L > 5 W < 0.21  
  
    em_jmax_dc_avg EQU 2 * 1.594 * ( w - 0.003 ) jmax_factor 105 1.434 110 1.000 115 0.704  
    120 0.500 125 0.358 L > 5 W >= 0.21  
  
    em_jmax_dc_avg EQU 4 * 1.594 * ( w - 0.003 ) jmax_factor 105 1.434 110 1.000 115 0.704  
    120 0.500 125 0.358 L <= 5  
  
    em_jmax_dc_peak EQU 18.04 * ( w - 0.003 )  
  
    em_jmax_ac_peak EQU 18.04 * ( w - 0.003 )  
  
    em_jmax_ac_rms EQU sqrt( 12.66 * deltaT * ( w - 0.003 )^2 * ( w - 0.003 + 0.264 ) / ( w - 0.003 + 0.0443 ) )  
  
    em_jmax_ac_avg EQU 2.17 (w - 0.003)  
  
    em_recover 0.5  
  
}
```

The peak, RMS, and average current limits for this layer are computed using the equations for the following three `em_jmax_ac` models.

- `em_jmax_ac_peak`: Peak current limits for this layer.
- `em_jmax_ac_rms`: RMS current limits for this layer.
- `em_jmax_ac_avg`: Average current limits for this layer.

The syntax of `EQU` for these three `em_jmax_ac` equations is a numeric expression ending in a newline. The expression follows normal expression syntax with normal precedence rules:

1. exponentiation (^)
2. \* or /

### 3. + or -

The following reserved names (case-insensitive) can be used in the equations to pass in parameters.

- `w`: Width of the wire being checked (required for metal layers that should be checked)
- `deltaT`: Delta temperature allowed. Normally only used for RMS limits to specify the max change in temperature allowed (optional).
- `jmax_factor`: For temperature based derating (optional).
- `jmax_lifetime`: Maximum years/hours of operation for lifetime derating (optional)

The following built-in functions are supported:

- `^` (exponentiation)
- `*` / (multiply, divide)
- `+` `-` (addition, subtraction)
- `( )` (grouping of expressions for precedence)
- `sqrt`
- `log`

## Specification of derating for lifetime or hours of operation in ICT:

```
jmax_lifetime lifetime1 scale1 [ lifetime2 scale2 .....]
```

`jmax_lifetime` provides the ability to set the scaling factor that applies to the current density limits for different lifetimes. Based on the lifetime value that you specify (using the `em_lifetime_units` parameter defined in the process section of the ICT file), the appropriate multiplication factor is applied to the nominal current density limit. The unit should be specified in accordance with the setting defined by the `em_lifetime_units` parameter defined in the process definition of the ICT file.

Sample extract from the ICT file:

```
process "name" {  
  
[em_lifetime_units hours | years]  
  
}  
  
conductor "M1" {
```

```
em_model {  
  
    em_jmax_ac_avg EQU 1 * 1.594 * ( w - 0.003 ) jmax_lifetime 5 4.3 10 1 15 0.4  
  
}  
  
}
```

## Specification for temperature based derating:

```
jmax_factor temp1 scale1 [ temp2 scale2 ...]
```

`jmax_factor` is an optional scaling factor that can be used at different temperatures compared to the reference temperature (defined by `em_tref` in the process definition). The temperature for `jmax_factor` should be specified in degrees Celsius. Scaling factor is a positive integer: >1 to scale up, <1 to scale down, or 1 for no scale effect.

**Note:** When setting scale factors for multiple temperatures, they should be specified in ascending sequence to enable interpolation.

Sample extract from the ICT file:

```
process "name" {  
  
    [em_tref value_in_degrees_Celsius ]  
  
}  
  
conductor "M1" {  
  
    em_model {  
  
        em_jmax_ac_avg EQU 1 * 1.594 * ( w - 0.003 ) jmax_factor 105 1.434 110 1.000 115 0.704  
        120 0.500 125 0.358  
  
    }  
  
}
```

## Area dependent parameters for via layers:

Area-based EM limits for vias can be defined using a `PWL` pair of *area value*

The `PWL` keyword in the syntax below signifies the use of area based EM limits.

```
em_jmax_ac_peak [ value | PWL value1 area1 | EQU fn(E) ]
```

Sample ICT file for a via layer:

```
via "via1" {  
  
    em_model {  
  
        em_vcwidth 0.32400  
  
        em_jmax_ac_avg PWL 12.308 0.104976  
  
    }  
  
}
```

## Units for EM limits:

You can control the EM units in the ICT file using the following parameters:

```
process "name" {  
  
    # EM_Model parameters  
  
    [em_tref value ]  
  
    [em_output_wlt silicon | drawn ]  
  
    [em_variables variable1 [ variable2 [ ...]]]  
  
    [em_conductor_unit A/cm^2 | mA/um ]  
  
    [em_via_unit A | mA ]
```

```
[em_via_area_unit A/cm^2 | mA/um^2 ]
```

```
[em_lifetime_units hours | years ]
```

```
}
```

The default units are highlighted in bold.

For more information on the ICT file, see the *EM\_Model* section in the "Creating the ICT File" chapter of the *QRC Techgen Reference Manual*.

For MMMC design, if you want to choose one view to do the  $I_{rms} / I_{peak} / I_{avg}$  check, you need to use the [set\\_default\\_view](#) command to specify the view name or else the first defined view will be used by default.

## Before You Begin

Before verifying AC limit, complete the following tasks:

- Perform RC extraction.
- Perform timing analysis.

## Results

After verifying AC limit violations, you can use information in the violation report to repair AC current limit violations. Use the [fixACLimitViolation](#) command to repair the violations. You can use the *Tools - Violation Browser* command for interactive viewing and highlighting of violation markers generated after you use this command.

## Verifying Isolated Cuts

Use the `verifyIsolatedCut` command to check cuts in all or specified cut layers against the spacing rules set by `setIsolatedCutRule`. Verify the isolated cut of the design after the following design step:

- Detailed routing

For more information, see `setIsolatedCutRule` and `verifyIsolatedCut` in the "Verify Commands" chapter of the Innovus Text Command Reference.

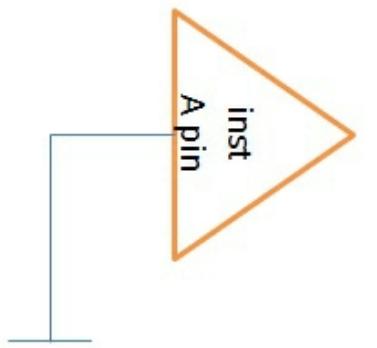
## Verifying Tie Cells

Use the `verifyTieCell` command to check whether or not the tie high/low cells specified with `setTieHiLoMode -cell` are connected to the tie high/low nets.

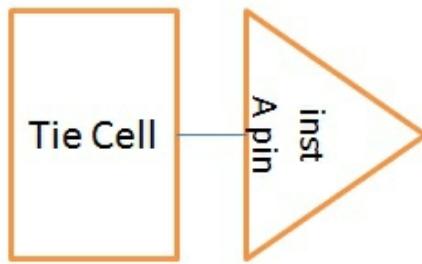
The expected flow is as follows:

1. Specify the tie cell names and other settings with the `setTieHiLoMode` command.
2. Add tie high/low cells with `addTieHiLo`.
3. Run `verifyTieCell` to check whether tie cell connections are correct.

`verifyTieCell` flags any tie high/low nets that are not connected to tie high/low cells.

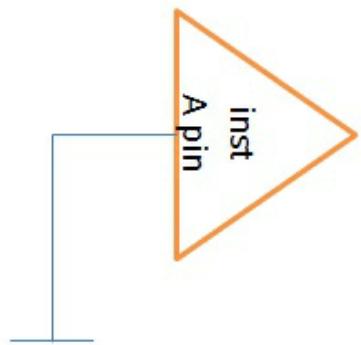


`verifyTieCell` detects violation



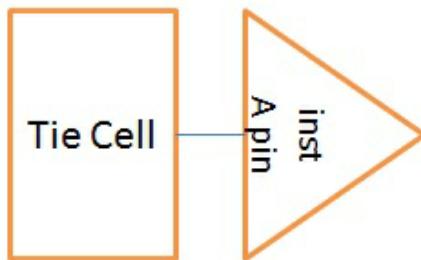
No violation flagged by `verifyTieCell`

`verifyTieCell -noTieCell filename` can be used to verify that pins specified in the file are not connected to tie cells.



No violation flagged by  
verifyTieCell -noTieCell <file>

<file>  
Inst/A



verifyTieCell -noTieCell <file>  
detects a violation

<file>  
Inst/A

## **Viewing Violations With the Violation Browser**

Use the Violation Browser form or the `violationBrowser` text command to view and highlight violation and lithography hotspot markers interactively.

### **Viewing Geometry or Metal Density Violations**

The Violation Browser updates violation markers generated by the `verifyGeometry` and `verifyMetalDensity` commands incrementally in an Innovus session--that is, it displays the markers generated the first time you run either of these commands and adds new markers, or deletes markers, from subsequent runs during the same session. If the software finds violations during a subsequent run that were already found previously, the browser display does not change, as there is no incremental update.

The browser can make the incremental changes because `verifyGeometry` and `verifyMetalDensity` can check a small area of the design and update the database. As a result of this behavior, the Innovus software saves the information from the first verification run.

### **Viewing Connectivity, Process Antenna, or AC Limit Violations**

The Violation Browser overwrites violation markers from the `verifyConnectivity`, `verifyProcessAntenna`, and `verifyACLimit` commands if they are run more than once during an Innovus session. These commands are net-based, not area-based, so the browser does not make incremental updates for connectivity, process antennas, or AC limit. As a result of this behavior, the software does not keep the information from the first verification run.

### **Viewing Violation Markers From Assura, Calibre, PVS, or Other Software Applications**

To view violation markers from Assura, Calibre, or PVS, or other software applications with the Violation Browser, use the following commands or forms:

- `createMarker`

This command creates markers from data imported from Assura or Calibre. For more information, see [createMarker](#) in the *Innovus Text Command Reference*.

- `loadViolationReport` (*Tools - Violation Browser - Load Violation Report*)

This command loads a report file from Assura, Calibre, PVS, or other software applications and converts it to a format that the Innovus software can interpret. For more information, see [loadViolationReport](#) in the *Innovus Text Command Reference*.

- `violationBrowser` (*Tools - Violation Browser*)

This command displays the markers in the Violation Browser. For more information, see [violationBrowser](#) in the *Innovus Text Command Reference*.

## Violation Browser Features

- Click a violation on the violation list on the form to see a description of the violation. The description includes actual and target values for AC limit violations, process antenna violations, and geometry spacing violations.
  - An actual value is the current value
  - A target value is the value defined in the LEF file.
- Click the + or - sign to collapse or expand the listings of each violation type.
- Use the *First*, *Previous*, *Next*, *Last*, *Up*, and *Down* buttons to navigate through the list of violations.
- The browser displays the violations in the following hierarchical order:
  - + tool
    - + type
      - + subtype
  - Description

where the `tool`, `type`, and `subtype` value correspond to the value you specify using the [createMarker](#) command.

- Use cross probing between the design display area and the Violation Browser. To display the details of a violation in the Violation Browser form, double-click the violation marker in the design display area.
- If there are violation markers for overlapped objects, select the top-most marker in the design display area and press the space bar on your keyboard to navigate through the other markers. The type and name of the selected violation is displayed in the lower-left corner of the Innovus main window. Use the `q` keyboard shortcut key to select a violation and highlight it in the Violation Browser form.
- Use the Zoom buttons to change the magnification level of a violation.
- Use any of the following buttons to change the display for a selected violation:
  - *Highlight Color*
  - *Highlight Violations*
  - *De-Highlight Violations*
  - *Delete Violations*
  - *Mark Violations as False*
  - *Mark Violations as True*
- Generate a report file by clicking the *Save* button.  
The report file includes information on the violations shown in the violation browser.
- Limit the number of violations to display by using the *Show Types* panel in the *Settings* page of the form.
- Limit the area to display by using the *Show Area* panel in the *Settings* page of the form.
- Filter the violations to display by using the *Other Filters* panel in the *Settings* page of the form.

## Saving Violations

Choose the *Tools - Violation Browser* menu item and then click the *Save DRC* button to save the violation markers to a file. The DRC file can be read back with the `loadDrc` command. Alternatively, you can load the DRC file by using the GUI form. To do so, choose the *Tools - Violation Browser* menu item and then click the *Load DRC* button.

## Clearing Violations

Choose the *Tools - Violation Browser* menu item and then click the *Clear Violation* button to clear the violation markers in your design.

# Verifying Well Pins and Bias Pins

- [Overview](#)
- [Flow](#)
  - [Adding Information to the Technology and Cell LEF Files](#)
  - [Specifying Connections of Pins to Wells](#)
  - [Validating Connections](#)
  - [Validating Width, Spacing, and Shorts](#)
  - [Exporting the Verilog Netlist](#)
- [Important Considerations for Usage](#)

## Overview

With the increase in usage of multiple power supplies, the need to be able to define well-layer information has increased. The Innovus™ Implementation System (Innovus) now supports defining and verifying well pins and bias pins.

The requirements for verification include:

- The `verifyConnectivity` command should detect floating wells, which are wells with no well tap connection.
- The `verifyGeometry` command should detect shorts between two wells with different connections.
- The `saveNetlist -phys` command should output connections for LVS checking purposes.
- The `sroute` command should ignore masterslice layer during followpin wiring.

## Flow

Following is the high-level flow for verifying well pins and bias pins:

- Adding information to the technology and cell LEF files to identify well taps and well layers, and the ports on those layers
- Specifying connections of pins to wells using `globalNetConnect` (or CPF) command

- Validating connections using `verifyConnectivity` command
- Validating width, spacing, and shorts with `verifyGeometry` command
- Exporting the verilog netlist with `saveNetlist -phys` command for LVS

**Note:** The output verilog physical netlist will contain the port connections to these pins for outside LVS runs.

All the points listed above are detailed in subsequent sections.

## Adding Information to the Technology and Cell LEF Files

Provided below are details of information required to be provided in the LEF files along with examples for the same:

- Add type property to LEF LAYER MASTERSLICE to represent \*WELL layers
  - `PROPERTY LEF58_TYPE "TYPE NWELL ;"`; See Example 1 below
  - `PROPERTY LEF58_TYPE "TYPE PWELL;"`; See Example 2 below

Note: `TYPE PWELL` is optional. It is added, if required, for triple well and substrate modeling.

- Add MACRO/PIN/PORT shapes for \*WELL layers
  - For well tap cells, the layer \*WELL shapes will be in the same PIN/PORT to which the physical connection is made. This could be the existing power/ground pin or a special well bias PIN. See Figure 1.

These PIN/PORTs would need to have connections available to the routing layers. See Figure 2.

- For regular standard cells, the \*WELL layers shapes will be in their own PIN. See Figure 3.

## LEF File Example (Row-Based Checking)

### Example 1: PROPERTY LEF58\_TYPE "TYPE NWELL"

```
LAYER NWELL
```

```
TYPE MASTERSLICE ;
```

```

PROPERTY LEF58_TYPE "TYPE NWELL ; " ;

PROPERTY LEF58_SPACING "SPACING ... ; " ;

PROPERTY LEF58_WIDTH "WIDTH ... ; "]]

END NWELL

```

### **Example 2: PROPERTY LEF58\_TYPE "TYPE PWELL"**

```

LAYER PWELL

TYPE MASTERSLICE ;

PROPERTY LEF58_TYPE "TYPE PWELL ; " ;

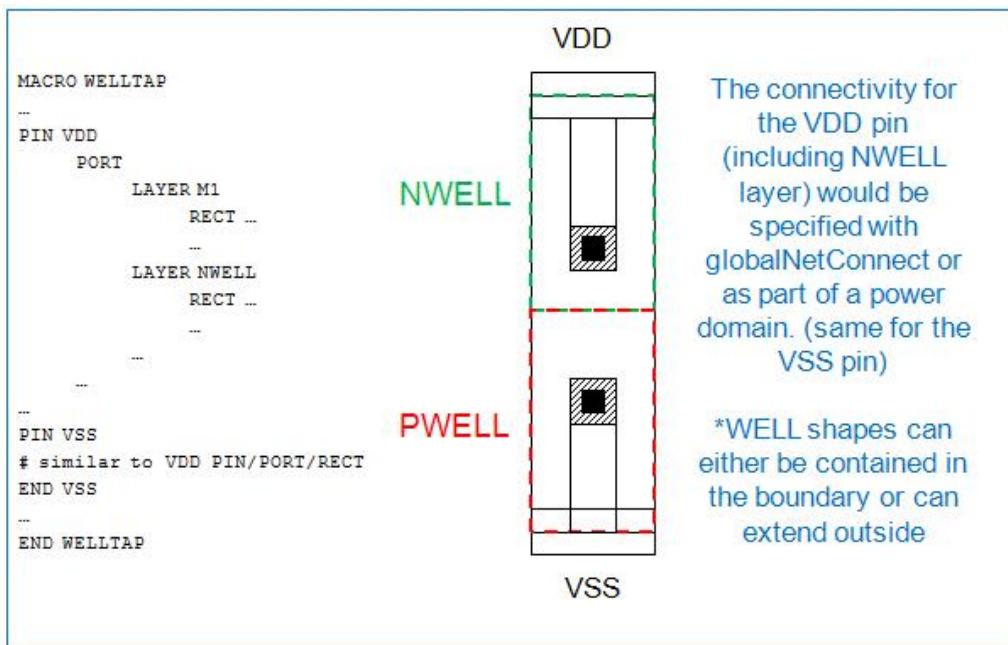
PROPERTY LEF58_SPACING "SPACING ... ; " ;

PROPERTY LEF58_WIDTH "WIDTH ... ; "]]

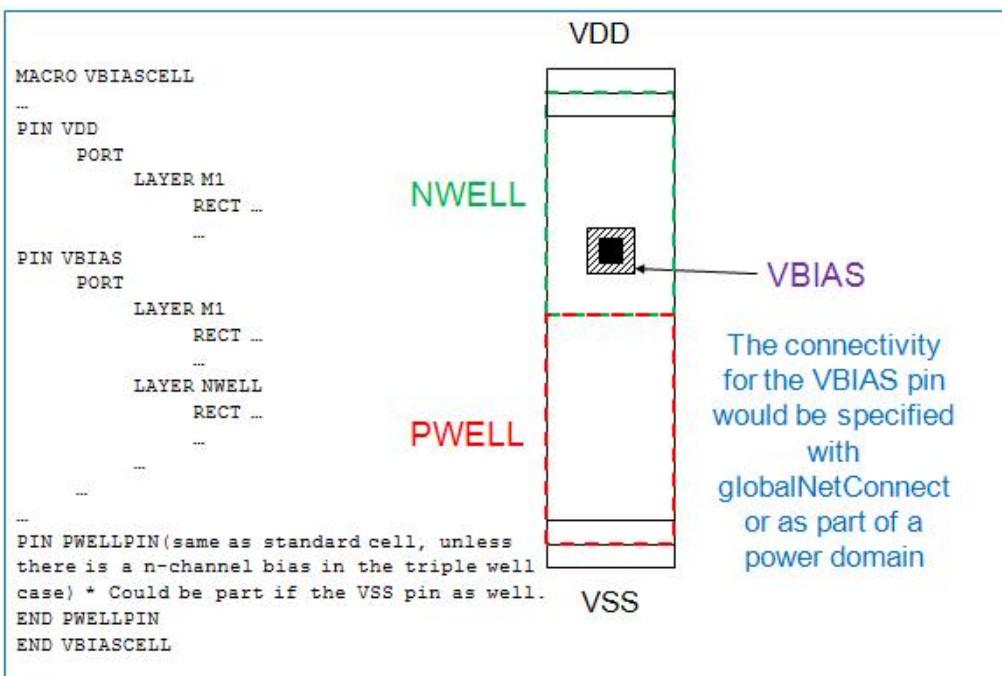
END PWELL

```

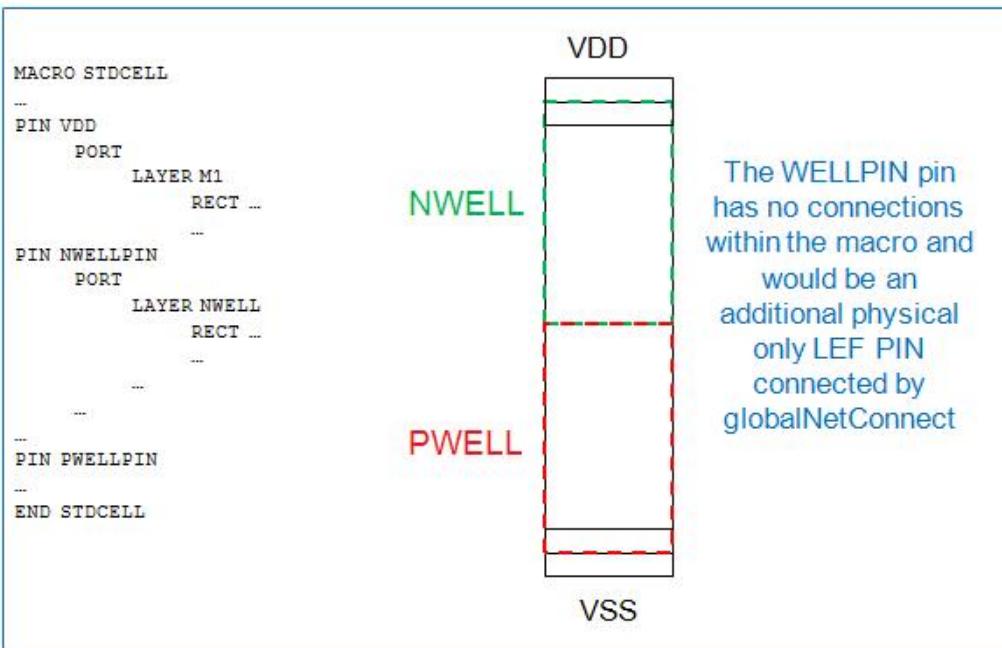
**Figure 1: LEF File Example for WellTap MACRO (CORE)**



**Figure 2: LEF File Example for VBIAS MACRO (CORE)**



**Figure 3: LEF File Example for Standard Cell MACRO (CORE), Including Filler Cells**



**Note:** The names of the NWELLPIN and VBIAS pins should be the same the ones in the Circuit Description Language (CDL) definitions that are used for LVS.

## LEF File Example (Block-Based Checking)

In LEF files for block-based checking, overlapping different wells is considered OK within the same cell. Also, blocks may typically model wells as obstruction (OBS) information, unlike CORE (ROW based) which would use PINS.

```
LAYER NWELLA  
  
TYPE MASTERSLICE ;  
  
PROPERTY LEF58_TYPE "TYPE NWELL ; " ;  
  
PROPERTY LEF58_SPACING "SPACING ... ; " ;  
  
PROPERTY LEF58_SPACING "SPACING ... LAYER NWELLB ; " ;  
  
PROPERTY LEF58_SPACING "SPACING ... LAYER NWELLC ; " ;  
  
PROPERTY LEF58_WIDTH "WIDTH ... ; "  
  
END NWELLA
```

## Specifying Connections of Pins to Wells

Connections of pins to wells are specified using the `globalNetConnect` or (CPF) command. Global net connections connect terminals and nets to the appropriate power and ground nets so that power planning, power routing, detail routing, and power analysis functions operate correctly for the entire design. For more information related to making global connections, see the following:

- "Global Net Connections" section in the Power Planning and Routing chapter in the *Innovus User Guide*.
- "Connect Global Nets" section in the Power Menu chapter in the *Innovus Menu Reference*.

## Validating Connections

After the connections are specified, they are validated using the `verifyConnectivity` command. This command detects conditions such as opens, unconnected wires (geometric antennas), unconnected pins, loops, partial routing, and unrouted nets; generates violation markers in the design window; reports violations.

Also, the `-noSoftPGConnect` parameter of this command is used to check connections that are only complete through the wells. This parameter disables the checking of soft power/ground connects.

For more information about verifying connections, see the [Verifying Connectivity](#) section in Identifying and Viewing Violations chapter in the *Innovus User Guide*.

## Validating Width, Spacing, and Shorts

The width, spacing, and shorts between wells are verified using the `verifyGeometry` command. Use this command to specify the checks to perform, disable checking, and set limits for errors and warnings to report. This command creates and saves violation markers in the design database.

For more information about verifying geometry, see the [Verifying Geometry](#) section in Identifying and Viewing Violations chapter in the *Innovus User Guide*.

## Exporting the Verilog Netlist

The `saveNetlist` command is used to write the netlist file of the design. The `-phys` parameter of this command is used to write out physical cell instances, and insert power and ground nets in the netlist. This command is used to output connections for LVS.

For more information, see the [Importing and Exporting Designs](#) chapter in the *Innovus User Guide*.

## Important Considerations for Usage

Provided below are some aspects for consideration while defining well-layer information:

- The usage is applicable only for power and ground PINs
- Most PINs will be DIRECTION INOUT
- PINs on the \*WELL layers do not need "SHAPE" attribute as that it only used to guide the sroute followpin behavior and the \*WELL layers are connected by abutment only, no additional routing is added
- The \*WELL layer shapes can abut to the boundary of the cell or extend outside
  - Typically derived directly from the layout information (GDSII or OA layout view)
  - In the case of substrate modeling, the PWELL shapes are typically created from an ANDNOT operation from the cell's boundary and NWELL shape
- If all the cells have an implied connection to the substrate and there is no "tap" connection from the topside for the PWELL, then all the PWELL shapes are included in the VSS PINs of

all of the CLASS CORE cells (instead of VSS for the tap and PWELLPIN for the non-tap cells)

# Integration with LPA and CCP

- [Overview](#)
- [Results](#)
- [Before You Begin Running LPA](#)
- [Running LPA from Innovus](#)
  - [Routing Layers Only Mode](#)
  - [Sign-Off Mode](#)
- [Before You Begin Running CCP](#)
- [Running CCP from Innovus](#)
  - [CCP Flow in Innovus](#)
  - [Running CCP in Cadence Model Flow](#)
  - [Running CMP Analysis in TSMC Model Flow](#)
  - [Viewing Hotspots](#)

## Overview

The integration of Litho Physical Analyzer (LPA) and Cadence CMP Predictor (CCP) with Innovus™ Implementation System allows you to perform the foundry-recommended or mandatory lithography and CMP checks at the block and chip level in your design directly from the Innovus GUI, much earlier in the development cycle. You can run LPA during the Routing and Sign-Off phases, and CCP during the Sign-Off phase.

LPA enables you to identify litho hotspots, Design for Manufacturing hotspots, layout optimization opportunities, and predict contours across process windows based on foundry-qualified technology files. It accurately predicts manufacturing variations associated with lithography and etch. Once detected, you can fix these litho hotspots using the NanoRoute routing technology.

**Note:** To learn more about the Cadence Litho Physical Analyzer tool, refer to the *Litho Physical Analyzer User Guide*.

CCP, on the other hand, allows you to identify the potential yield issues that are due to the variations in interconnect thickness caused by Chemical and Mechanical Polishing (CMP). CCP accurately predicts the thickness of the interconnect and dielectric for any design and any manufacturing process that has been calibrated. The resulting prediction is then used to minimize performance loss and to identify thickness-related yield issues.

**Note:** To learn more about the Cadence CMP Predictor tool, refer to the *Cadence Chemical Mechanical Polishing Predictor User Guide*.

You use the *DFM* menu of the main Innovus window to configure LPA and CMP runs on the design. However, the *DFM* menu might not appear on your Innovus window or one of its submenu options might be disabled if the prerequisite conditions are not satisfied (See [Before You Begin Running LPA](#) and [Before You Begin Running CCP](#)).

## Results

The output of an LPA or CCP run is an HIF file containing information about all detected hotspots. You can view this HIF file in the Innovus *Violation Browser* and fix the reported hotspots using NanoRoute.

## Before You Begin Running LPA

Before you can run LPA from Innovus, the following conditions must be met:

- You must have the LPA license to run litho sign-off from Innovus. However, to run LPA in Routing Layers Only mode, the Innovus DFM GXL Option license is sufficient and no separate LPA license is required.
- You must have either the Encounter Advanced Node GXL Option or Innovus DFM GXL Option license.
- You must be able to launch version 9.2 (or a later version) of LPA from Innovus. In other words, the installation path to LPA must be present in your path variable.
- You must have LPA TechFiles that are compatible with the design technology.

## Running LPA from Innovus

The integration of LPA with Innovus allows you to check for litho hotspots and predict contours across process windows earlier in the development cycle, much before the Sign-Off phase. The integration is smooth and easy to configure, and does not require any user intervention to stream out or set up LPA.

You can run LPA from Innovus in two modes:

- [Routing Layers Only Mode](#)
- [Sign-Off Mode](#)

Further, both these modes also support the DRC+ verification methodology from GLOBALFOUNDRIES, in addition to the standard LPA flow. The DRC+ methodology utilizes a

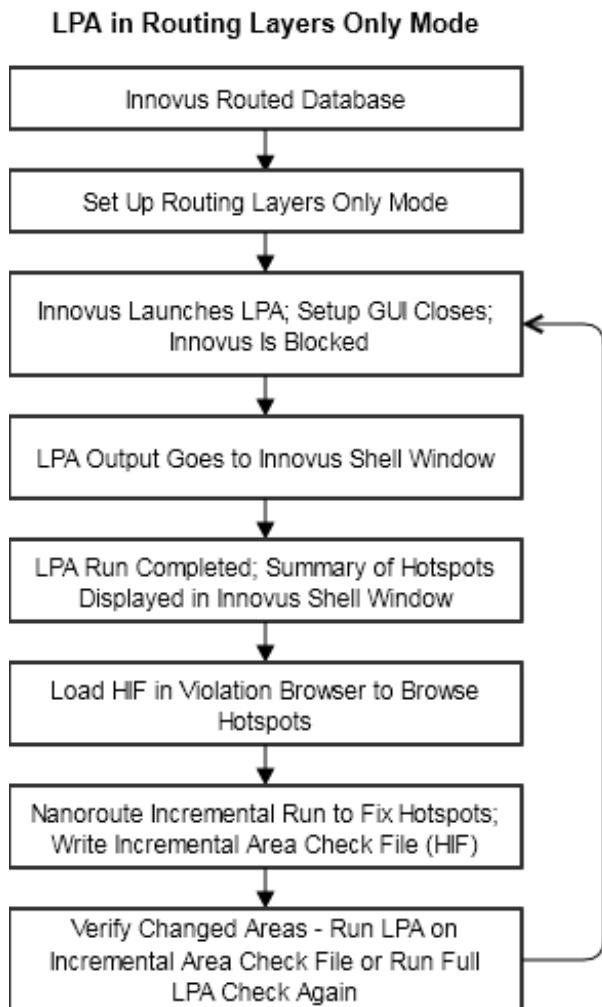
foundry-supplied DRC+ Pattern technology file, but the use model is otherwise identical to the standard LPA flow. For guidance on the choice of technology files, consult with your foundry.

## Routing Layers Only Mode

Routing Layers Only mode is the fast mode of LPA to flag L1 hotspots in a design at the block level during the Implementation phase. In this mode, LPA runs about 100X faster than Sign-Off Mode and helps you identify and fix most hotspots as routing is completed.

**Note:** LPA in Routing Layers Only mode is enabled only when you have the Innovus DFM GXL Option license.

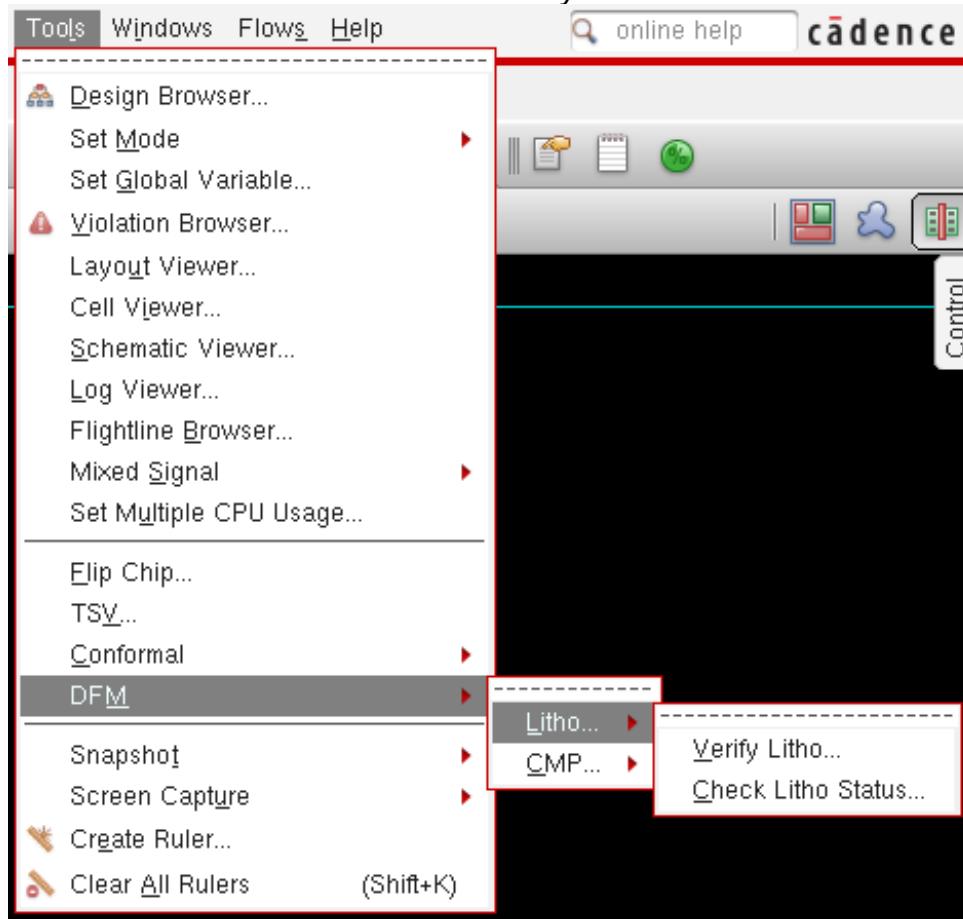
The following diagram shows the design flow for running LPA in Routing Layers Only mode.



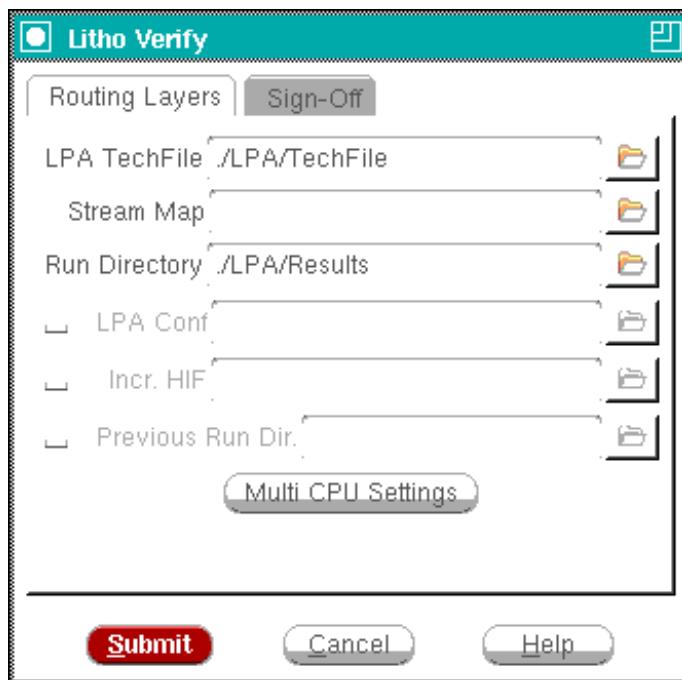
## Running LPA in Routing Layers Only Mode from the GUI

To run LPA in Routing Layers Only mode, launch Innovus by using the `innovus` command and load the design. When Innovus is invoked, it automatically loads all the required LPA files from the LPA installation path. Once the Innovus GUI is displayed, perform the following steps:

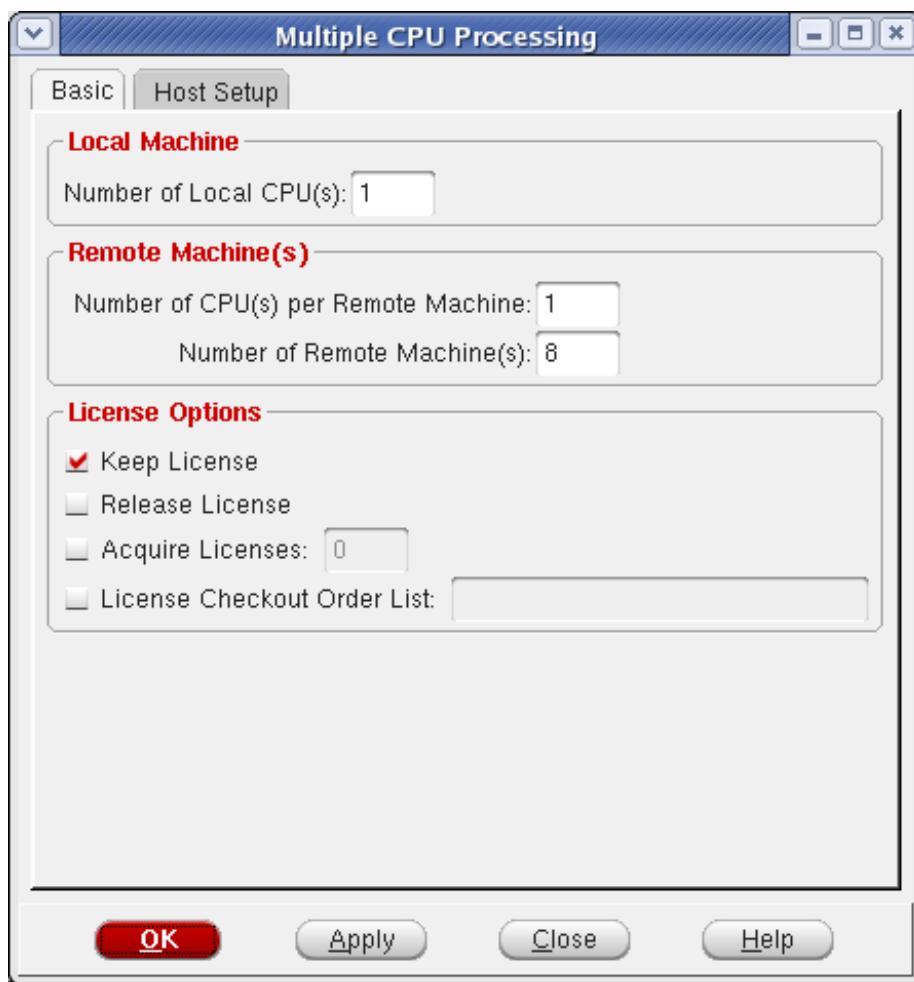
1. Choose *Tools* -> *DFM* -> *Litho* -> *Verify Litho* from the Innovus menu.



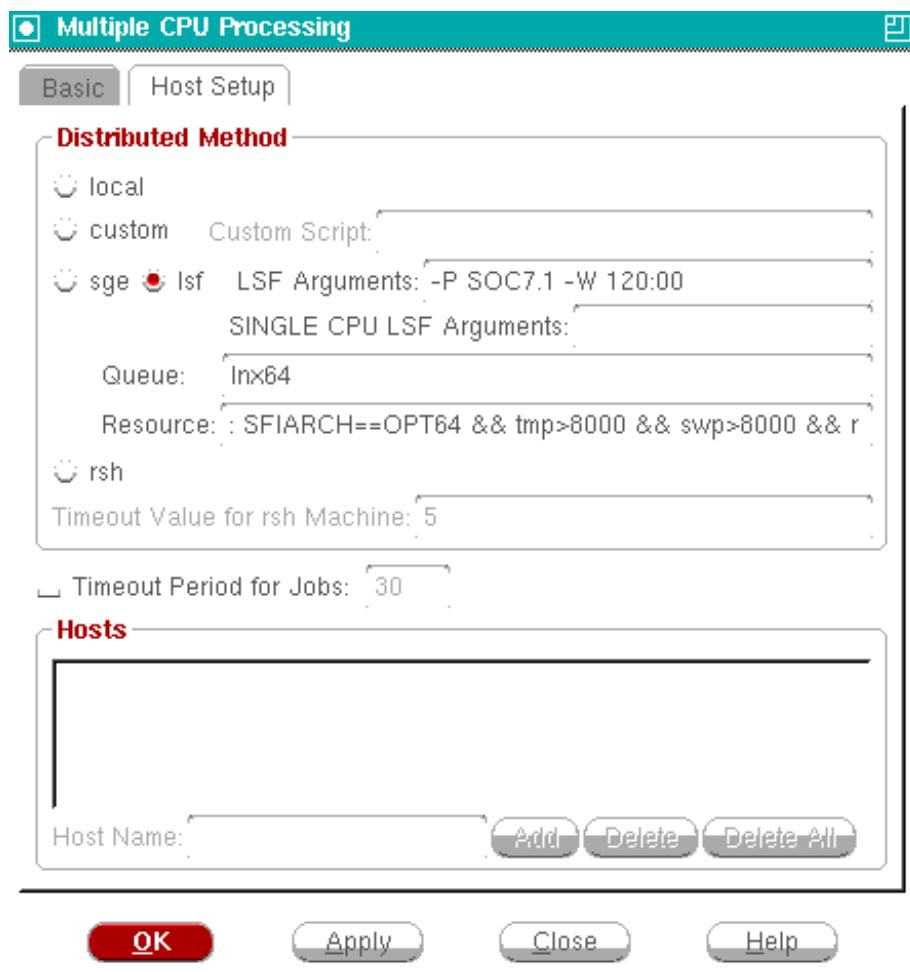
2. This opens the *Litho Verify* form, with the *Routing Layers* tab selected by default. In the *LPA TechFile* field, specify the path and name of the qualified LPA Mx- technology file that includes process-specific hotspot checking options and the LPA model. Alternatively, if you want to enable the GLOBALFOUNDRIES DRC+ flow, you specify the path to the foundry-supplied DRC+ Pattern technology file in the *LPA TechFile* field.



3. In the *Stream Map* field, specify the stream out map file, which maps the GDS stream to the layers in the Innovus database. Ensure that the GDS layer numbers in the stream out layer map matches the numbers in the LPA layer map.
4. In the *Run Directory* field, specify the LPA output directory that will contain one subdirectory for each layer (POLY, M1, etc.) when LPA is run with the configuration file for that layer.
5. *LPA Conf* is an optional field. Select this check box and specify the name and path of the LPA custom configuration file. This file, if specified, controls all run options of LPA.
6. *Incr. HIF* is also an optional field. Select this check box and specify the name and path of the HIF file that you want to use for incremental validation. This HIF file includes the locations that identify the areas affected by each hotspot fix. LPA reads these locations and performs incremental checking only in these areas. This reduces the time for validation.
7. Another optional field is *Previous Run Dir*. You use this option when the design has been changed but no HIF file that identifies the changed areas is available. In this field, you specify the path to the run directory of a previous Verify Litho run that you want to use for XOR-based incremental validation. When this option is selected, the previous run results are compared to the new design and LPA is run only in locations where the layout has changed and where hotspots previously existed, thereby reducing the overall validation run time.
8. By default, LPA uses the LSF settings from the *Multi-CPU Settings* GUI in Innovus. However, you can change the multi-CPU settings for the LPA run by selecting the *Multi CPU Settings* button and specifying the new settings in the *Multiple CPU Processing* form. Note that this will also change the distributed options for all Innovus commands.



9. On the *Basic* tab of the *Multiple CPU Processing* form, if running locally, use the *Number of Local CPU (s)* field to set the number of local CPUs used. If using LSF to run, the field *Number of Remote Machine(s)* field specifies the number of LSF machines that you want to use for the LPA run.
10. On the *Host Setup* tab, select the */sfradio* radio button to set the distribution method as LSF and specify the LSF arguments in the *LSF Arguments* field or select the */local* radio button to use local cpu(s) only.  
**Note:** It is recommended to use local or LSF distribution method for the Innovus-LPA integration.



11. Specify the queue and resource string that is needed for the LSF launch in the *Queue* and *Resource* fields.
  12. Select the *OK* button to confirm the multiple CPU settings and close the form.
  13. Select the *Submit* button in the *Litho Verify* form to launch the LPA run with the specified settings.
- Note:** In Routing Layers Only mode, LPA runs in blocking mode. You cannot perform any operations in the Innovus GUI until the LPA run is completed.

During the LPA run, the output is sent to the Innovus *Shell* window. After LPA is successfully completed, the LPA summary file with hotspot count is presented in the Innovus *Shell* window.

The screenshot shows a terminal window titled "Shell - Konsole". The output of the LPA command is displayed, including warnings about empty layers M10 and M9, distributed processing statistics, runtime information, log files, error files, and hotspot HIF files. A detailed hotspot summary table is shown, comparing layer counts across four categories (L1-L4) for layers M2 through M8. The table shows that only layer M2 has a value in L1 (3), while all other layers have "n/a" in all categories.

```
**Warning** Layer M10 is empty
**Warning** Layer M9 is empty

Distributed Processing
    Number of Clients used (Requested/used): 9/9

Runtime
    Elapsed Runtime (hh:mm:ss): 0:03:15
    Total Runtime (hh:mm:ss): 0:05:05

LPA Detailed Log File: ./LPA/Results/Logs/InShape.log
LPA Error File: ./LPA/Results/LPA.err
DMC Hotspot HIF File: ./LPA/Results/DMC/merged.hif

Total Hotspots : 3

Hotspot Summary:

-----+-----+-----+-----+-----+
Layer | L1 | L2 | L3 | L4 |
-----+-----+-----+-----+-----+
M2 | 3 | n/a | n/a | n/a |
M3 | 0 | n/a | n/a | n/a |
M4 | 0 | n/a | n/a | n/a |
M5 | 0 | n/a | n/a | n/a |
M6 | 0 | n/a | n/a | n/a |
M7 | 0 | n/a | n/a | n/a |
M8 | 0 | n/a | n/a | n/a |
-----+-----+-----+-----+
```

## Running LPA in Routing Layers Only Mode from the Command Line

You can also run LPA in the Routing Layers Only mode from the Innovus command line by using the `verifyLitho -routingLayersOnly` option. You must have the LPA techfile and the streamMap file to run LPA in this mode. The streamMap file is same as the one used by the Innovus `streamOut` command.

To run LPA in the Routing Layers Only mode from the command line, launch Innovus by using

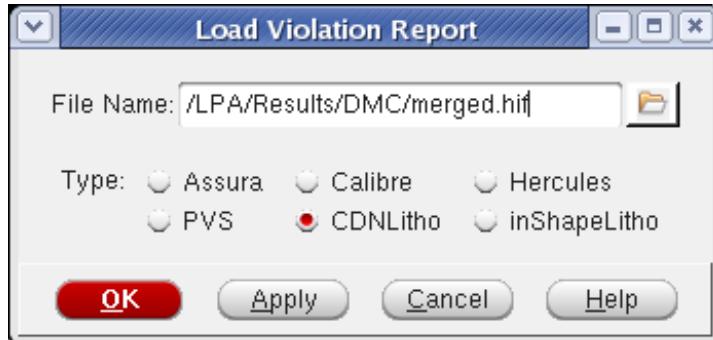
the `innovus` command and load the design. When invoked, Innovus automatically loads all the required LPA files from the LPA installation path. Once the `innovus>` prompt is displayed, run the following command at the prompt:

```
innovus> verifyLitho -routingLayersOnly -techFile LPATechFile_dir -dir ./LPA/Results
```

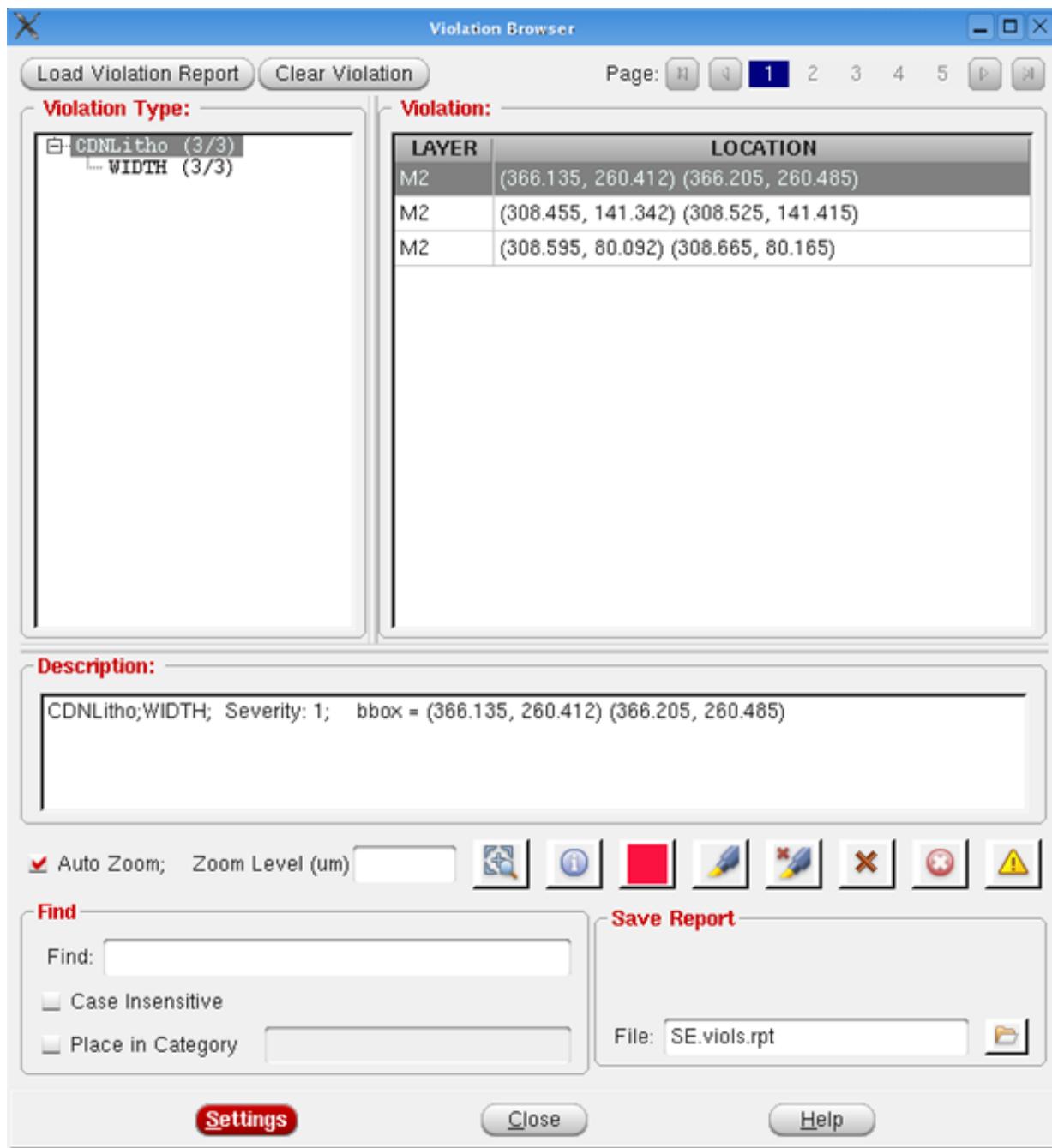
## Viewing Hotspots

You can load the HIF file (created in the output directory by the LPA run) in the *Violation Browser* directly to view the detected litho hotspots. To do this, perform the following set of steps:

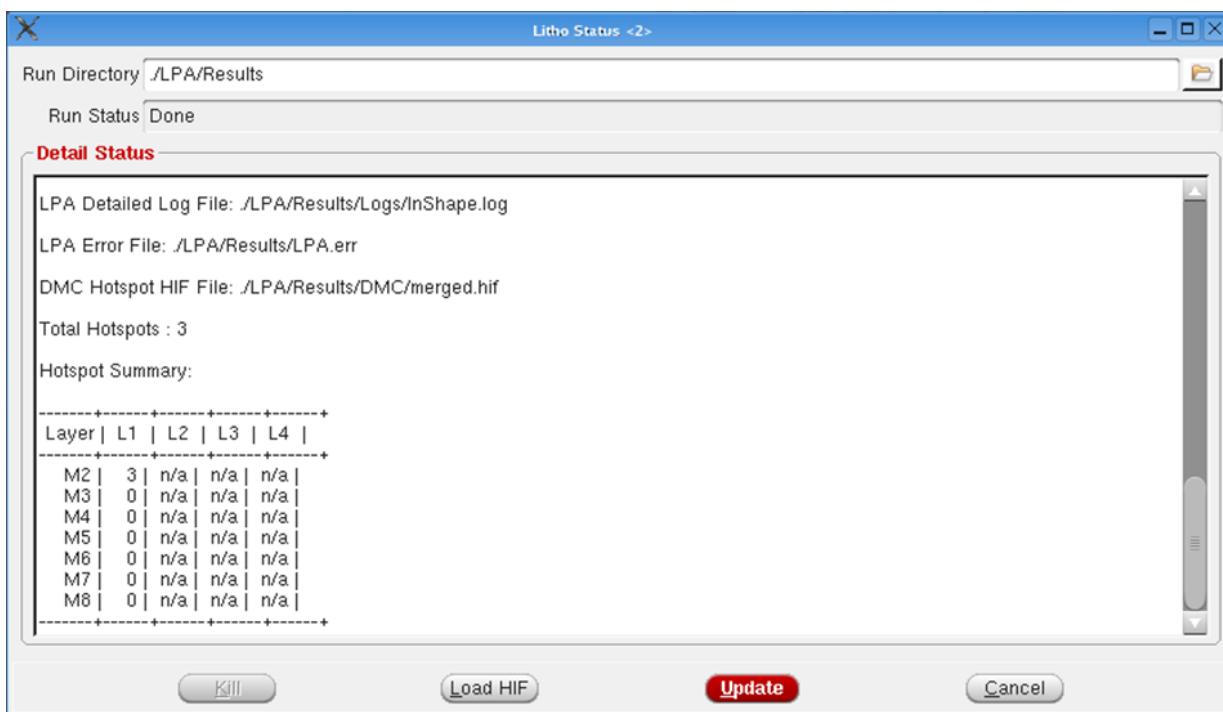
1. Select *Tools* -> *Violation Browser* -> *Load Violation Report*.
2. This opens the *Load Violation Report* form. Specify the path and name of the HIF file in the *File Name* field.



3. Select the *CDNLitho* radio button to specify the HIF format and select the *OK* button to view the hotspots in the *Violation Browser*.



You can also load the hotspots from the *Litho Status* window (*Tools -> DFM-> Litho -> Check Litho Status*) by specifying the results directory name in the *Run Directory* field and selecting the *Load HIF* button. This will open up the *Violation Browser* to show the hotspot details.



**Note:** You can also specify a different run directory name in the *Run Directory* field of the *Litho Status* window and then click the *Update* button to check the output of the specified LPA run.

## Fixing Hotspots

Next, you fix the hotspots identified by the LPA run by using Nanoroute. There can be several ways to fix hotspots depending on the design technology. Please ask your Cadence representative for technology specific applications notes, which may provide advanced methodologies. To use the standard fixing approach, perform the following set of steps:

1. At the Innovus command line, type the following to set up Nanoroute and run it on the design to repair the litho hotspots.

```
innovus> setNanoRouteMode -droutePostRouteLithoRepair true
innovus> setNanoRouteMode -drouteMinimizeTopologyChange true
innovus> globalDetailRoute
```
2. After Innovus has completed `globalDetailRoute`, save the design for verification. This will also save the markers within the design, indicating the areas of change.

```
innovus> saveDesign Litho_Fixed.enc
```
3. The Innovus router marks the areas of change and you can run Litho only on these areas of change rather than on the complete design, thereby saving time. Write out the incremental HIF

file that will be used to run Litho only on the changed areas.

```
innovus> writeHif -file IncrVerify.hif
```

4. Now, run LPA in the incremental area to check for hotspots after Innovus has fixed the litho hotspots. Open the *Litho Verify* form, with the *Routing Layers* tab selected by default. Specify the name of the incremental HIF file in the *Incr. HIF* field and click *Submit* to run LPA only on the changed areas.

If LPA reports no more litho hotspots in this run, the verification task is complete. However, if there are litho hotspots reported, repeat the steps to load the HIF file (created in the output directory by the latest LPA run) in the *Violation Browser*, and fix the hotspots using Nanoroute. The design is marked as verified when the LPA run on the incremental HIF file reports zero litho hotspots.

## Selecting and Excluding Nets

While fixing hotspots, you can choose to exclude certain nets or areas in the design. `verifyLitho` allows you to specify many options for selecting or excluding areas within a design by area, layer identification, or cell. These options can be added to a configuration file which can be passed to `verifyLitho` by file with the `-config config_file_name` option. For information about these options, refer to the *Litho Physical Analyzer User Guide* in the MVS distribution.

Besides the native `verifyLitho` options for selection and exclusion, you can use Innovus commands to specify which nets should be affected by routing. To prevent a net from being moved during routing, simply specify that the net should not be touched during routing by using the following command:

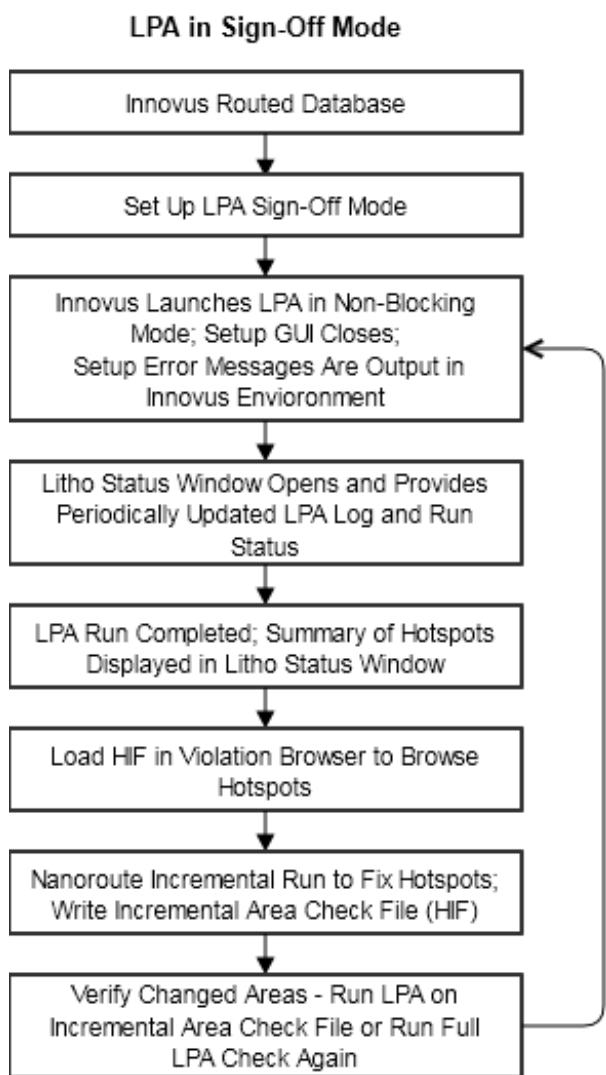
```
setAttribute -net net_name -skip_routing true
```

This will prevent the net from being modified during detail route. Be sure to turn this attribute to false after litho fixing is complete.

## Sign-Off Mode

You run LPA in Sign-Off mode for Litho sign-off, as mandated by foundries. Unlike Routing Layers Only mode where no user input is required, Sign-Off mode requires you to provide the LPA configuration file containing the Techfile and other settings. You must run LPA Sign-Off mode before handing off the design to the top-level or tape-out.

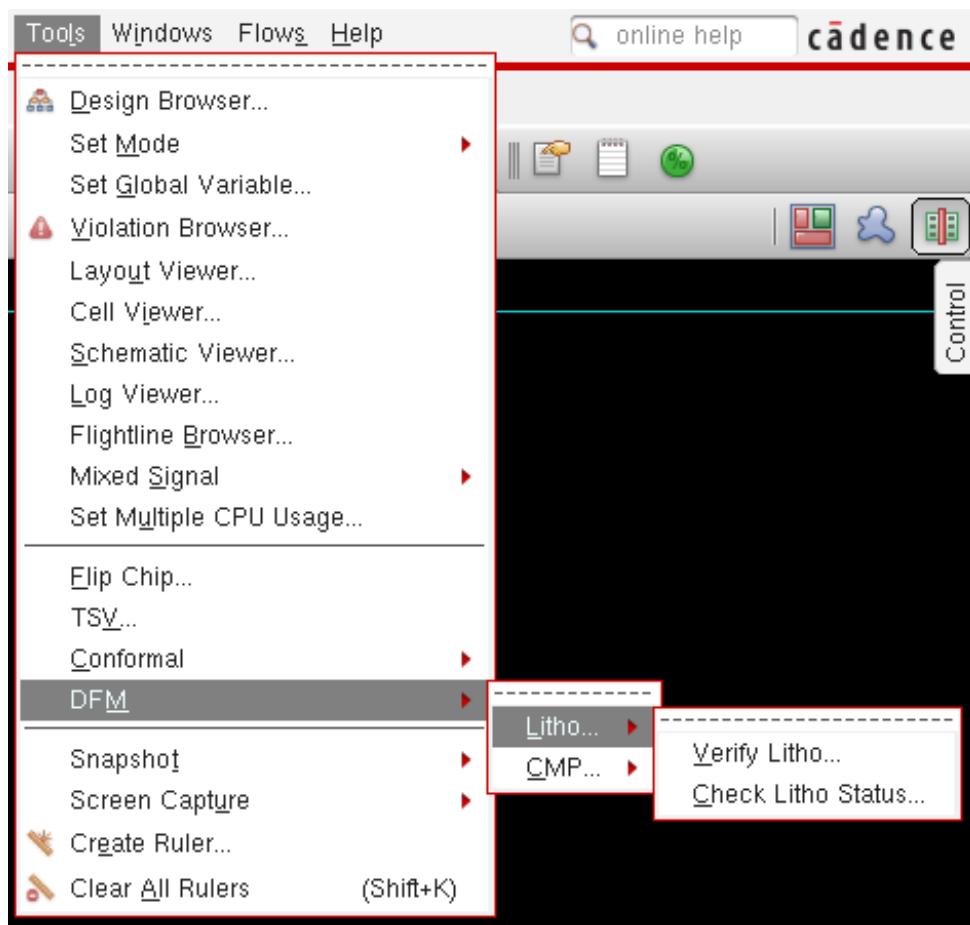
The following diagram shows the design flow for running LPA in Sign-Off mode.



## Running LPA in the Sign-Off Mode from the GUI

To run LPA in Sign-Off mode, launch Innovus by using the `innovus` command and load the design. When Innovus is invoked, it automatically loads all the required LPA files from the LPA installation path. Once the Innovus GUI is displayed, perform the following steps:

1. Choose *Tools -> DFM -> Litho -> Verify Litho* from the Innovus menu.



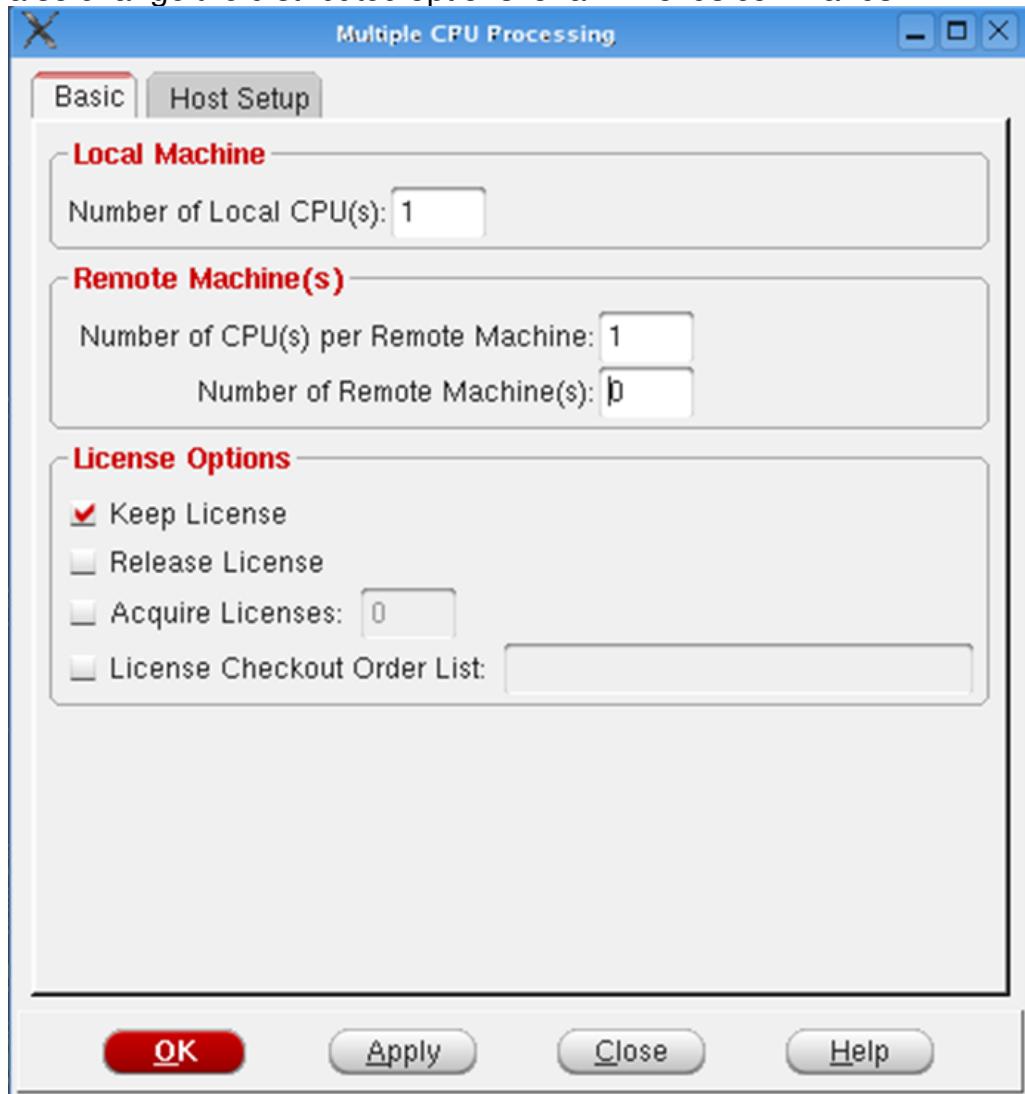
2. This opens the *Litho Verify* form, with the *Routing Layers* tab selected by default. Select the *Sign-Off* tab.



3. Specify the path and name of the LPA output directory in the *Run Directory* field. All output data from the LPA run will be stored under this directory.
4. Specify the name and path of the LPA configuration file in the *LPA Conf* field. This configuration file should contain the Techfile location for all layers and any additional LPA commands that you want to execute during the LPA run.  
Alternatively, if you want to enable the GLOBALFOUNDRIES DRC+ flow, specify the name and path of the configuration file that points to the foundry-supplied DRC+ Pattern technology file.
5. In the *Additional CPUs* field, specify the number of additional CPUs you want to use for the current sign-off LPA run. This number is in addition to the total number of CPUs specified in the *Total CPUs* field. A higher number of additional CPUs results in decreased run time.
6. Optionally, you can specify the name and path of the GDS list file and Stream Out Map file in the *GDS List File* and *Stream Out Map* fields, if you want to run Poly, Diffusion, or Metal1. The GDS list file is a text file containing the list of GDS files for LEF abstracts. The Stream Out Map file is created by Innovus to map the GDS stream to the layers in the Innovus database. By default, LPA runs on the interconnect layers in Sign-Off mode but if the GDS List file and Stream Out Map file are specified, LPA runs on the potential IP blocks defined in these files.
7. If you have already run LPA in Sign-Off mode once and are running it again to check if all detected hotspots have been fixed, specify the name and path of the HIF file that you want to use for incremental validation in the *Incr. HIF* field. This HIF file includes the locations that identify the areas affected by each hotspot fix. LPA reads these locations and performs

incremental checking only in these areas. This reduces the time for validation.

8. If you have already run LPA in Sign-Off mode once and are running it again, but there is no HIF file that identifies the changed areas, you use the *Previous Run Dir* field to specify the path to the run directory of the previous Verify Litho run for XOR-based incremental validation. When this option is selected, the previous run results are compared to the new design and incremental checking is performed only in locations where the layout has changed and where hotspots previously existed, thereby reducing the overall validation run time.
9. By default, LPA uses the LSF settings from the *Multi-CPU Settings* GUI in Innovus. However, you can change the multi-CPU settings for the LPA run by selecting the *Multi CPU Settings* button and specifying the new settings in the *Multiple CPU Processing* form. Note that this will also change the distributed options for all Innovus commands.

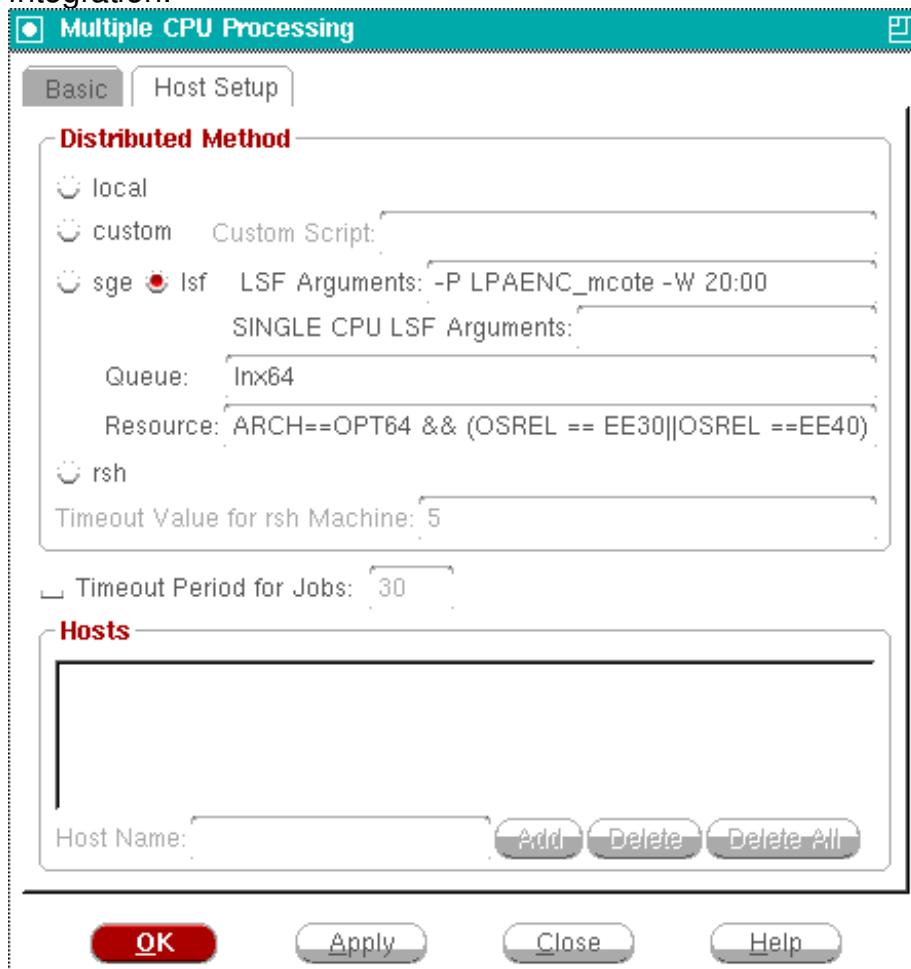


10. On the Basic tab of the *Multiple CPU Processing* form, if running locally, use the *Number of*

*Local CPU(s)* field to set the number of local CPUs used. If using LSF to run, the *Number of Remote Machine(s)* field specifies the number of LSF machines that you want to use for the LPA run.

11. On the *Host Setup* tab, select the the */lsf* radio button to set the distribution method as LSF and specify the LSF arguments in the *LSF Arguments* field or select the *local* radio button to use local CPUs only.

**Note:** It is recommended to use local or LSF distribution method for the Innovus-LPA integration.



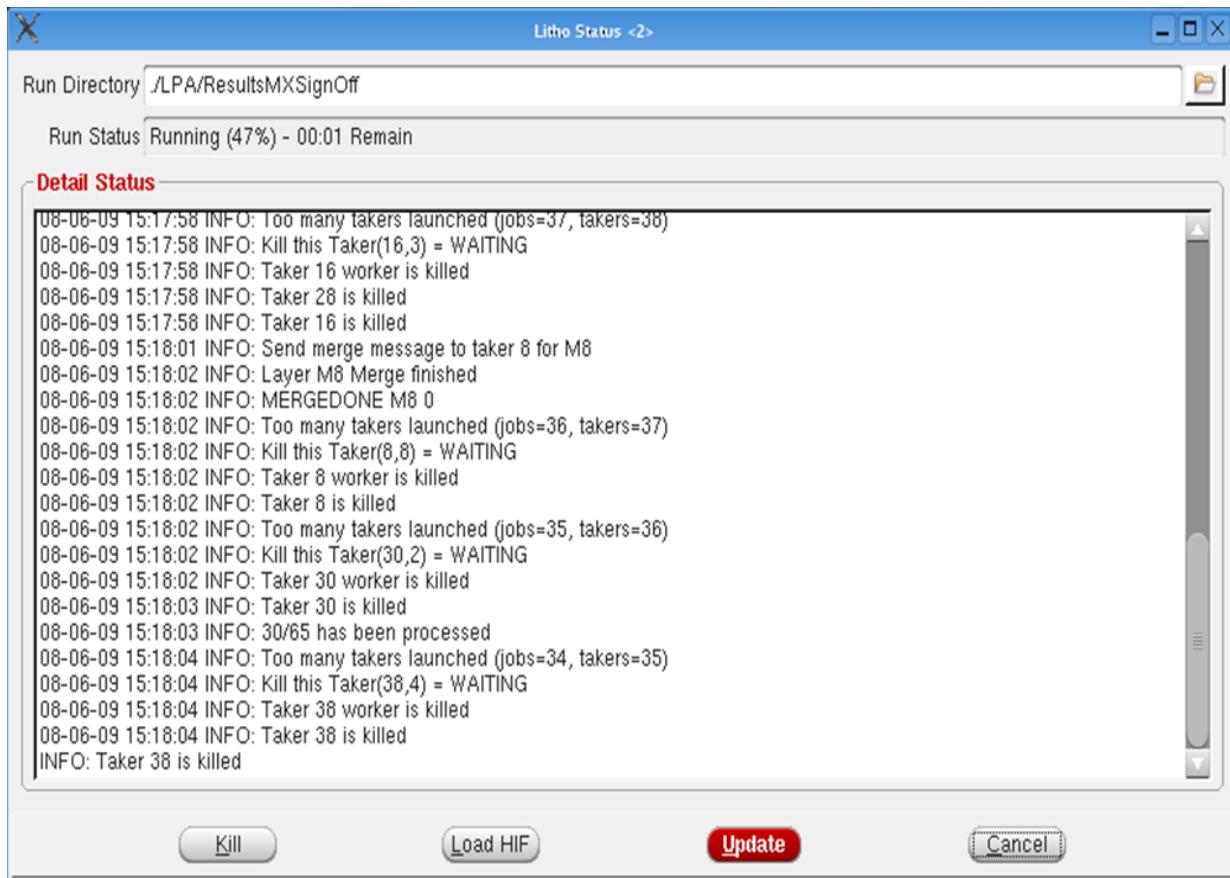
12. Specify the queue and resource string that is needed for the LSF launch in the *Queue* and *Resource* fields.
13. Select the *OK* button to confirm the multiple CPU settings and close the form.
14. Select the *Submit* button in the *Litho Verify* form to launch the sign-off LPA run with the specified settings.

**Note:** In Sign-Off mode, LPA runs in non-blocking mode. You can continue working with the

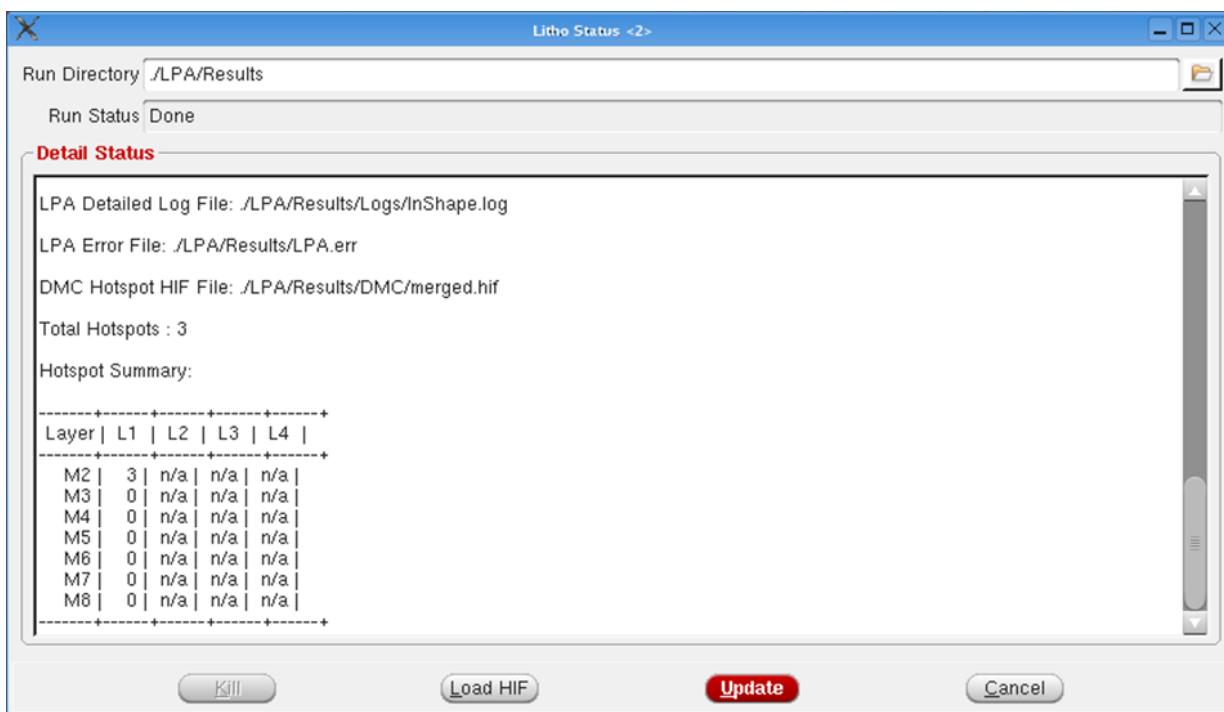
Innovus GUI and perform design activities in parallel with Litho sign-off. If you exit Innovus during this period, a warning is displayed informing you that LPA is still running and you can load the results later.

15. On the successful launch of LPA, the *Litho Verify* form closes and the *Litho Status* window is automatically displayed. The *Run Directory* field of this window is automatically populated from the *Litho Verify* form.

The *Litho Status* window provides updated information about the status of the LPA run. The *Run Status* field provides information on the completion percentage and approximate remaining time of the LPA run. The run status is periodically updated in the *Detail Status* area. You can click the *Update* button to manually update the status. To terminate the current LPA run, click the *Kill* button.



On completion of the LPA run, the *Litho Status* window displays the summary of the hotspots detected by the run. You can close the *Litho Status* window anytime during the LPA run and open it again by selecting *Tools* -> *DFM*-> *Litho* -> *Check Litho Status*.



**Note:** You can also specify a different run directory name in the *Run Directory* field of the *Litho Status* window and then click the *Update* button to check the output of the specified LPA run.

## Running LPA in the Sign-Off Mode from the Command Line

You can also run LPA in the Sign-Off mode from the Innovus command line by using the `verifyLitho -signOff` option. You must have the LPA techfile and the GDS to LPA layer map file to run LPA in this mode. By default, LPA runs on the interconnect layers in Sign-Off mode but if the GDS List file and Stream Out Map file are specified, LPA runs on the potential IP blocks defined in these files.

To run LPA in the Sign-Off mode from the command line, launch Innovus by using the `innovus` command and load the design. When invoked, Innovus automatically loads all the required LPA files from the LPA installation path. Once the `innovus` prompt is displayed, run the following command:

```
innovus> verifyLitho -signOff -dir ./LPA/Results -cpu 1 -config lpa.conf -gdsList
<leaf/*.gds> -mapFile innovusGds.map
```

The GDS to LPA layer map file should have path of the lpa configuration file (lpa.conf). Its syntax is as follows:

```
#userDefined name GDSlayer FAB_techFile_layer_name
```

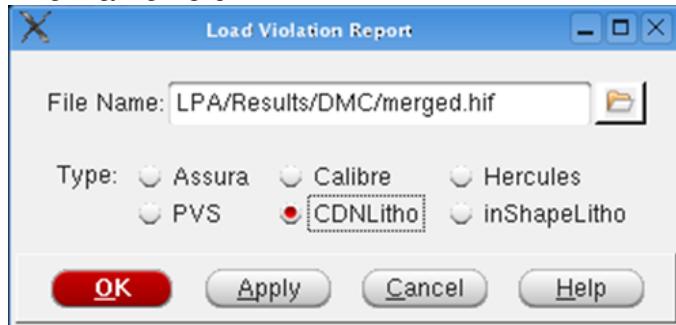
M1 31:0 M1  
M2 32:0 M2

For more details, refer to the *Litho Physical Analyzer User Guide*.

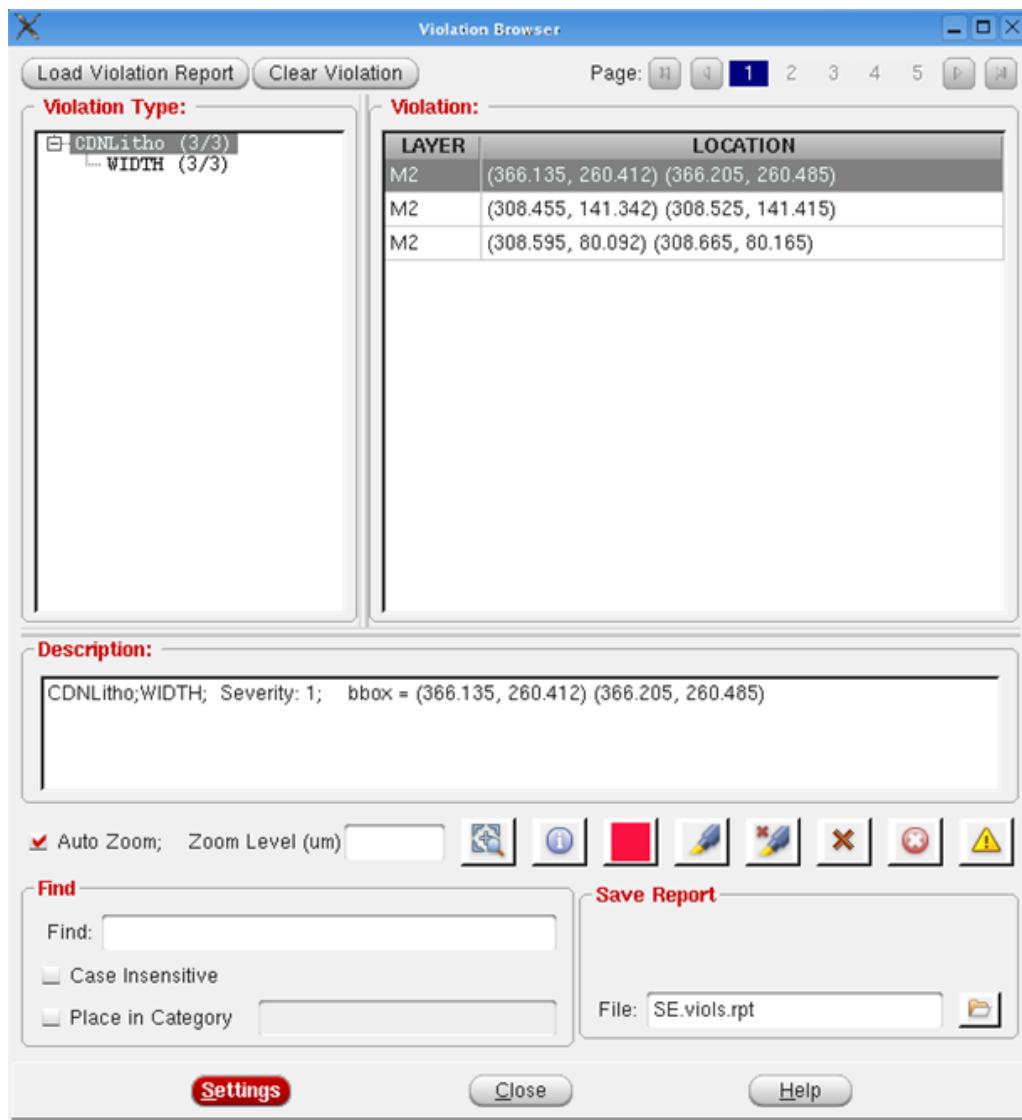
## Viewing Hotspots

You can load the HIF file (created in the output directory by the LPA run) in the *Violation Browser* directly to view the detected hotspots. To do this, you perform the following set of steps:

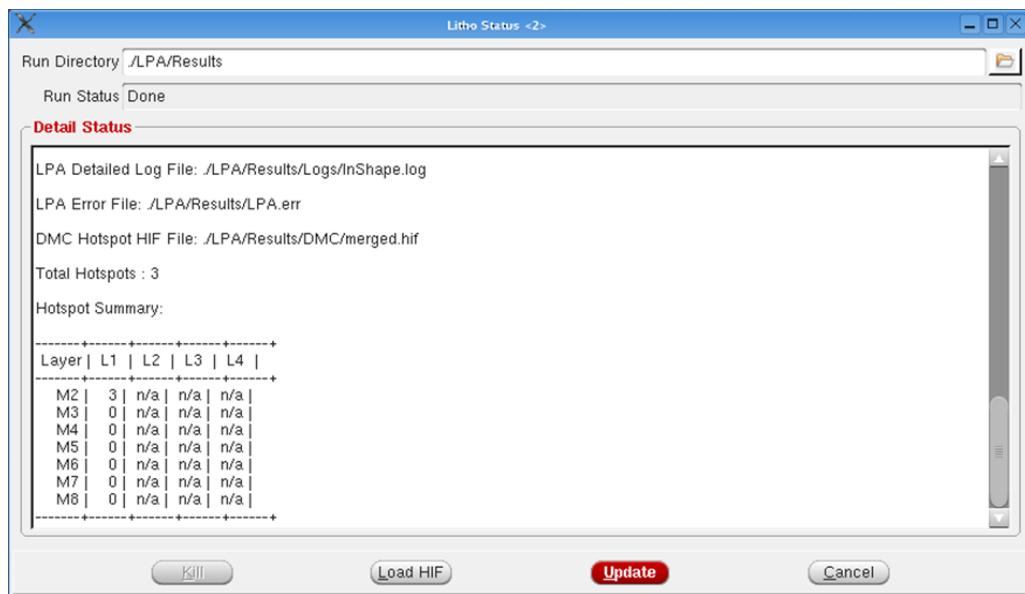
1. Select *Tools -> Violation Browser -> Load Violation Report*.
2. This opens the *Load Violation Report* form. Specify the path and name of the HIF file in the *File Name* field.



3. Select the *CDNLitho* radio button to specify the HIF format and select the *OK* button to view the hotspots in the *Violation Browser*.



4. You can also load the HIF file from the *Litho Status* window (*Tools -> DFM-> Litho -> Check Litho Status*) by specifying the results directory name in the *Run Directory* field and selecting the *Load HIF* button. This will open up the *Violation Browser* to show the hotspot details.



## Fixing Hotspots

Next, you fix the hotspots identified by the LPA run by using Nanoroute. The Innovus router marks the areas of change in the design, which you can use to write out the incremental HIF file. The steps to perform this task are similar to the steps performed in the Routing Layers Only mode (see [Fixing Hotspots](#)). Once you create the incremental HIF file, use it to run LPA only on the changed areas rather than on the complete design, thereby saving time. Here, you run LPA from the *Sign-Off* tab by specifying the name of the incremental HIF file in the *Incr. HIF* field.



## Before You Begin Running CCP

Before you can run CCP from Innovus, the following conditions must be met:

- You must have the CCP license to run CCP from Innovus.
- You must have either the Encounter Advanced Node GXL Option or Innovus DFM GXL Option license.
- You must be able to launch version 9.2 (or a later version) CCP from Innovus. In other words, the installation path to CCP must be present in your path variable.
- You must have CCP TechFiles that are compatible with the design technology.
- You must have TSMC VCMP DDK 1.2 kit or a later version for TSMC CMP checks.

## Running CCP from Innovus

The integration of CCP with Innovus allows you to check for the potential yield issues that are caused by variations in interconnect thickness for any design and any manufacturing process that has been calibrated. You can run CCP to identify L1 hotspots on full chip or on blocks larger than 1mm x 1mm.

Following are the main advantages of running CCP from within Innovus.

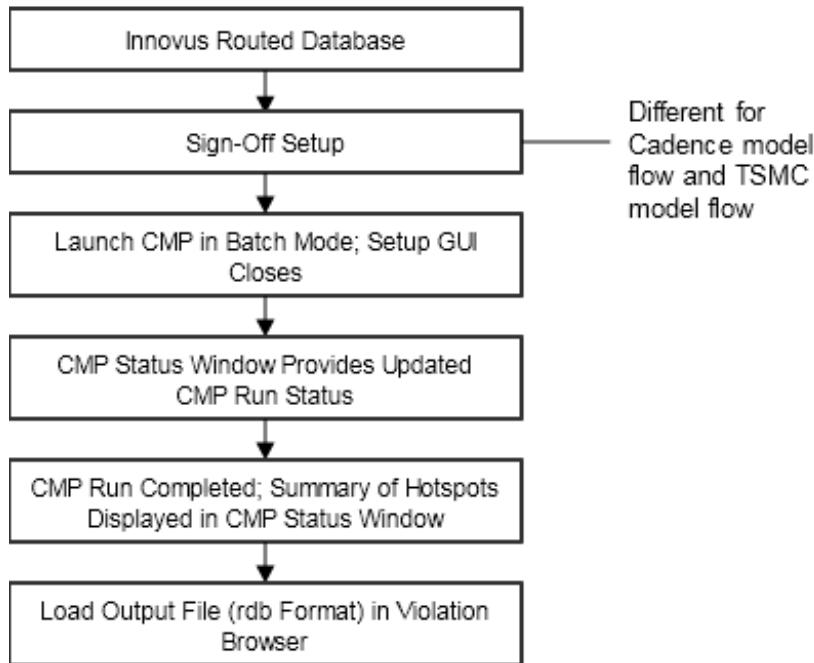
- Supports manufacturability checks of routed blocks
- Supports fast check and sign-off verification
- Eliminates translation and setup hassles
- Runs with default out-of-the-box setup

Depending on the Fab, you use either the Cadence model flow or the TSMC model flow to verify your design. The only difference in the procedure to run CCP in both these flows is in the CMP setup. For the TSMC model flow, you additionally need to set up the VCMP engine and process file.

## CCP Flow in Innovus

The following diagram shows the design flow for running CCP in Sign-Off mode.

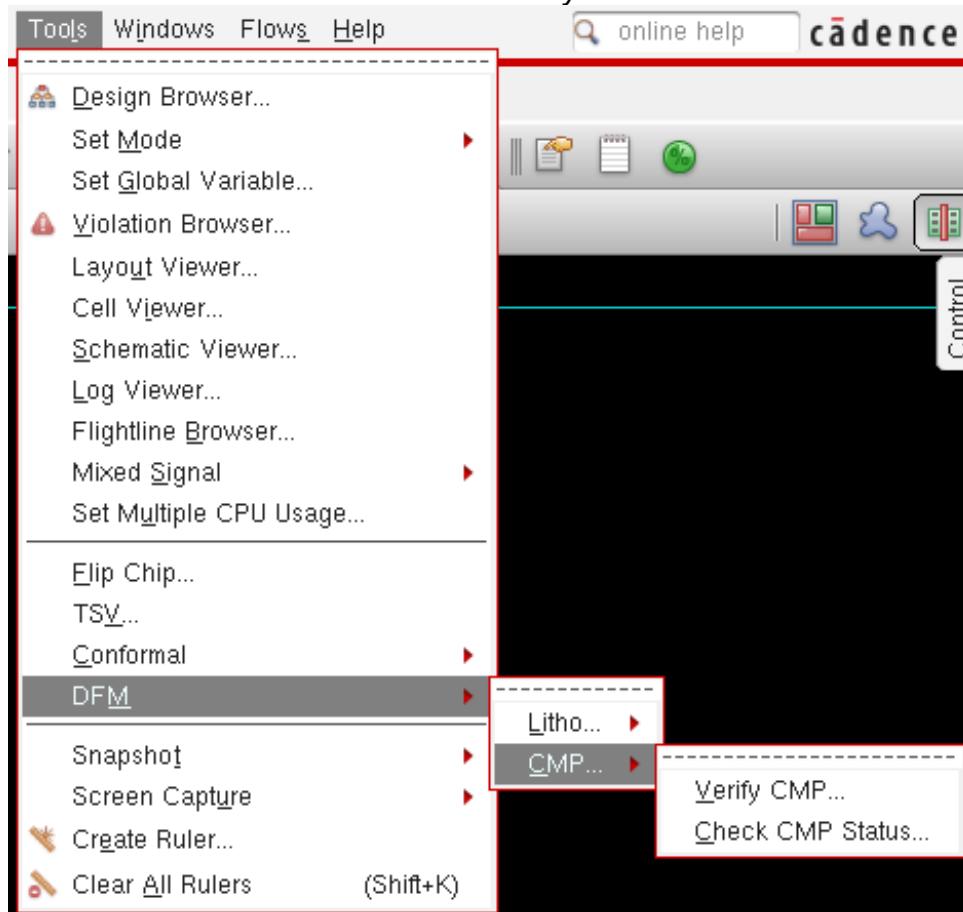
### CCP in Sign-Off Mode



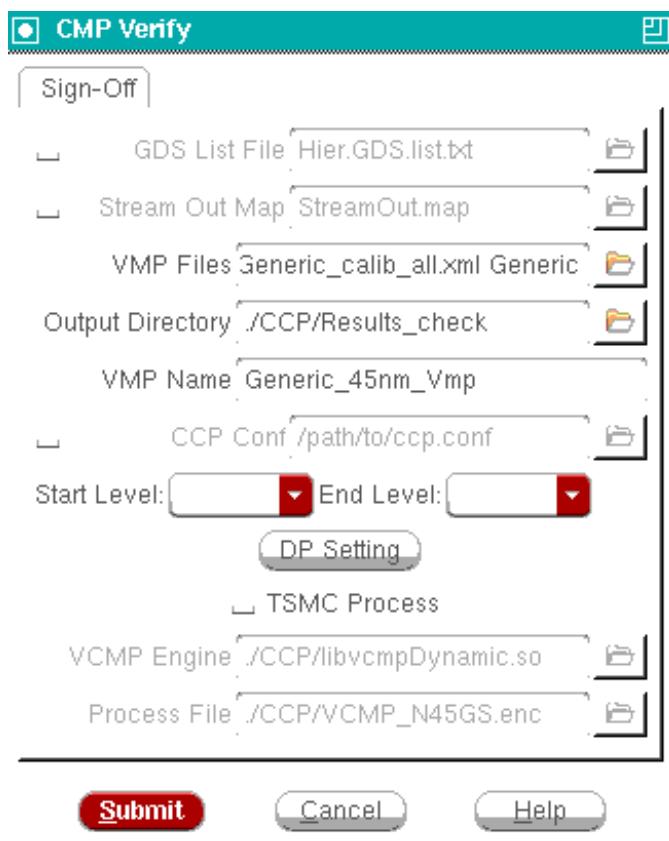
## Running CCP in Cadence Model Flow

To run CCP analysis on your design using the Cadence model flow, launch Innovus by using the innovus command and load the design. When Innovus is invoked, it automatically loads all the required CCP files from the CCP installation path. Once the Innovus GUI is displayed, perform the following steps:

1. Choose *Tools* -> *DFM* -> *CMP* -> *Verify CMP* from the Innovus menu.



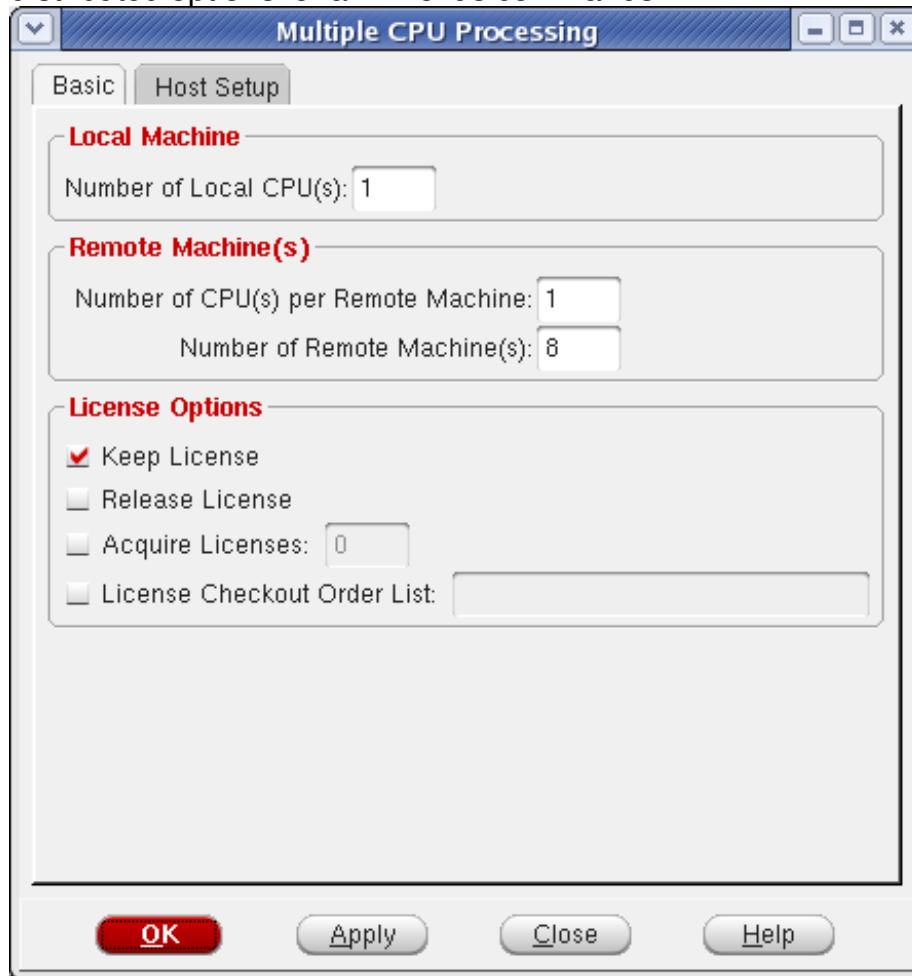
2. This opens the *CMP Verify* form, with the *Sign-Off* tab selected by default. In the *VMP Files* field, specify the name and path of the `vmp.xml` file to be used for the CMP run. These files can be downloaded from the foundries. These files are usually packaged in a DDK kit. The extraction and prediction results of the CMP analysis are based on the process specifications in your `vmp` file.



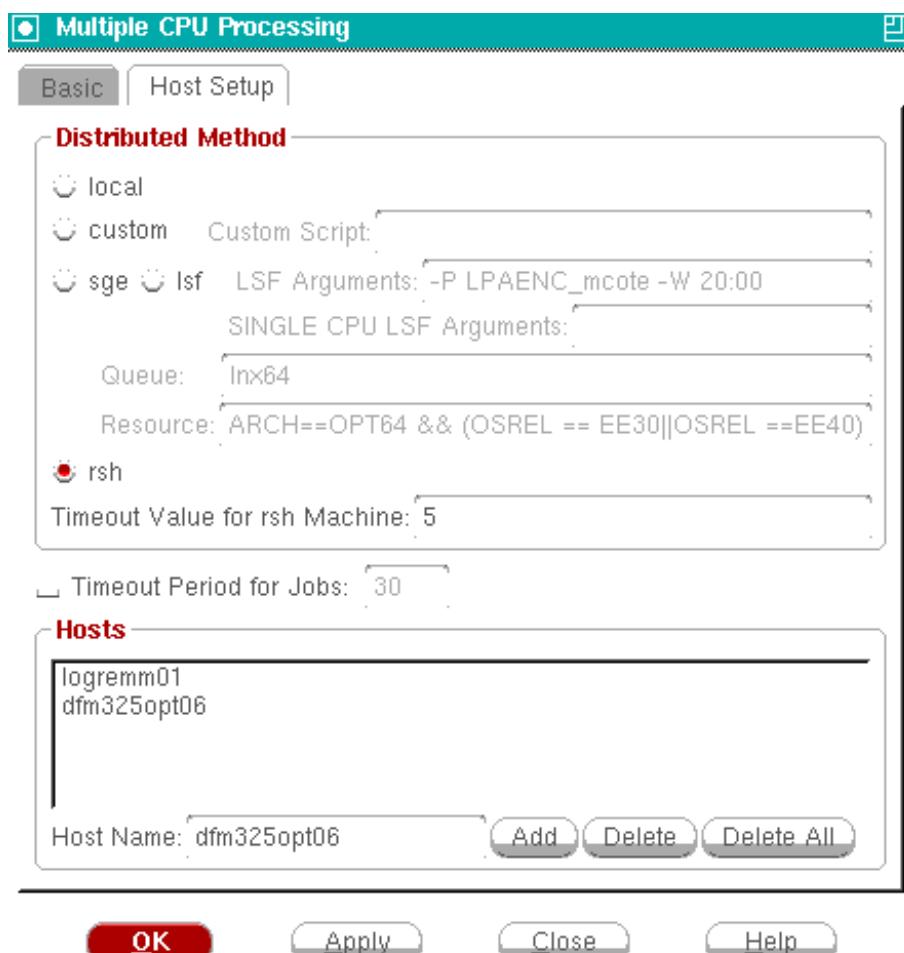
3. In the *Output Directory* field, specify the CMP output directory that will contain the extraction and prediction results of the CMP run.
4. In the *VMP Name* field, specify the name of the metal scheme from the `vmp.xml` file.
5. Optionally, you can specify the name and path of the CMP configuration file in the *CCP Conf* field. This file, if specified, controls all run options of the CMP simulation.
6. The *Start Level* and *End Level* fields allow you to control the number of layers for prediction. These fields are optional. You can specify the starting metal level and end metal level in the *Start Level* and *End Level* fields for the CMP simulation. You can select them from the drop-down menus associated with these two fields.
7. *GDS List File* and *Stream Out Map* fields are also optional. You can specify the name and path of the GDS list file and Stream Out Map file in these fields. The GDS list file is a text file containing the list of GDS files for LEF abstracts. The Stream Out Map file is created by Innovus to map the GDS stream to the layers in the Innovus database. CMP runs on the interconnect layers by default, but if the GDS List file and Stream Out Map file are specified, CMP runs on the potential IP blocks defined in these files.
8. By default, CMP uses the multiple CPU settings from the *Multi-CPU Settings* GUI in Innovus. However, you can change the multi-CPU settings for the CMP run by selecting the *DP*

*Settings* button and specifying the new settings in the *Multiple CPU Processing* form. For example, if you enter 8 in the *Number of Remote Machine(s)* field, eight jobs will be farmed out to LSF.

**Note:** If you modify the multi-CPU settings for CMP analysis, this will also change the distributed options for all Innovus commands.



9. On the *Host Setup* tab, select the *rsh* radio button to set the distribution method as RSH and add the RSH host name to the *Hosts* section.



10. Select the **OK** button to confirm the multiple CPU settings and close the form.
11. Select the **Submit** button in the *CMP Verify* form to launch the CMP run with the specified settings.  
**Note:** CMP runs in non-blocking mode. You can continue working with the Innovus GUI and perform design activities while the CMP simulation is running. If you exit Innovus during this period, a warning will be displayed informing you that CMP is still running and you can load the results later.  
On the successful launch of the CMP simulation, the *CMP Verify* form closes and the *CMP Status* window is automatically displayed. This window provides updated information about the status of the CMP run.



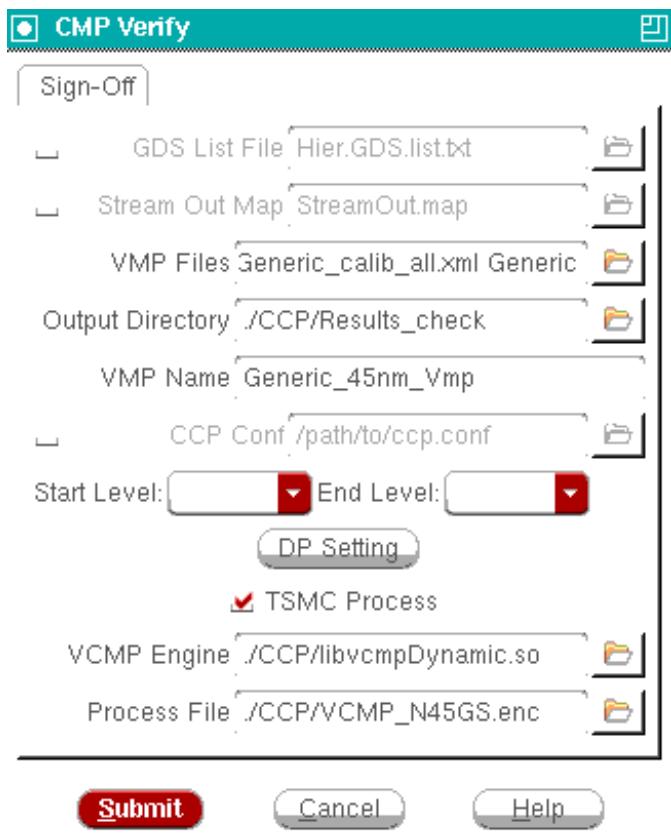
On completion of the CMP run, the *CMP Status* window displays the summary of the hotspots detected by the run and an `rdb` format output file is generated in the results directory.

You can close the *CMP Status* window anytime during the CMP run and open it again by selecting *Tools -> DFM-> CMP -> Check CMP Status*.

**Note:** You can also specify a different run directory name in the *Run Directory* field of the *CMP Status* window to check the output of another CMP run.

## Running CMP Analysis in TSMC Model Flow

The steps to run CMP analysis on your design using the TSMC model flow are similar to the steps in running CMP using the Cadence flow ([Running CCP in Cadence Model Flow](#)). The only difference is in the CMP setup. For the TSMC model flow, you need to set the following additional fields in the *CMP Verify* form.



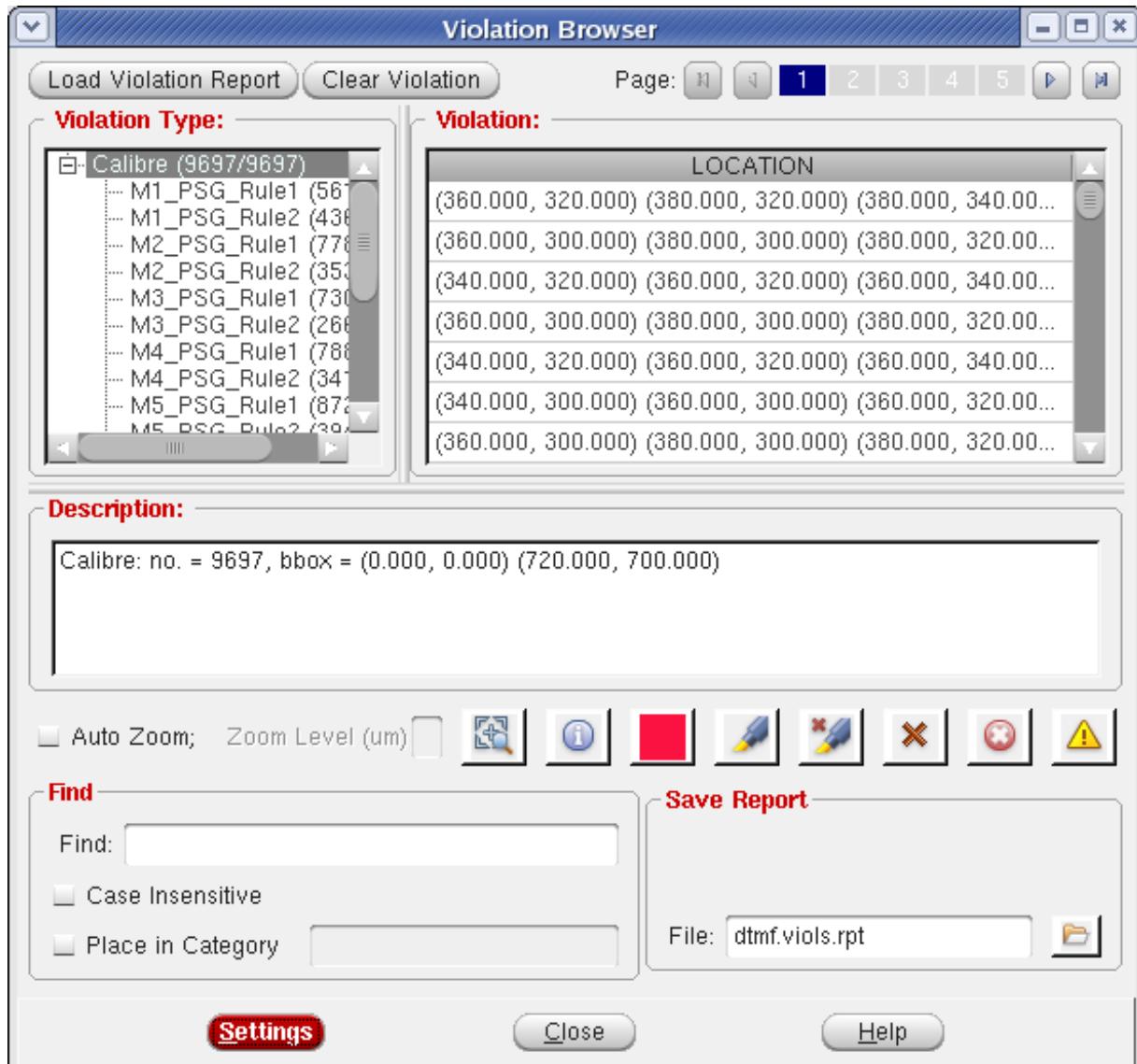
1. Select the *TSMC Process* check box if you want to use the TSMC model instead of the Cadence model.
2. In the *VCMP Engine* field, specify the location and name of the VCMP engine that is to be used for running CMP.
3. In the *Process File* field, specify the location and name of the VCMP process file to be used for the CMP simulation.

The rest of the steps to run CMP analysis using the TSMC model flow are the same as that in the Cadence model flow.

## Viewing Hotspots

You can load the HIF file (created in the output directory by the CMP run in the `rdb` format) in the *Violation Browser* directly to view the detected hotspots. To read how to load the HIF file in the *Violation Browser*, refer to [Viewing Hotspots](#).

You can also load the HIF file from the *CMP Status* window (*Tools -> DFM-> CMP -> Check CMP Status*) by specifying the results directory name in the *Run Directory* field and selecting the *Load HIF* button. This will open up the *Violation Browser* to show the hotspot details.



# ECOs and Interactive Design Editing

---

- ECO Flows
- ECO Directives
- Interactive ECO
- Editing Wires

# ECO Flows

This appendix summarizes the variety of Engineering Change Order (ECO) flows possible with Innovus, and outlines the current approach for each flow.

- [Overview](#)
- [Pre-Mask ECO Changes from a New Verilog File](#)
- [Pre-Mask ECO Changes from a New DEF File](#)
- [Pre-Mask ECO Changes from an ECO File](#)
- [Post-Mask ECO Changes from a New Verilog Netlist \(Using Spare Cells Flow\)](#)
- [Post-Mask ECO Changes from a New Netlist \(Using Gate Array Cells Flow\)](#)
- [Post-Mask ECO Changes from a New Verilog Netlist \(Using Gate Array Filler Cells Flow\)](#)

## Overview

Many types of ECO flows are possible. The ones outlined in this appendix cover the most common cases. You can use these flows directly, or you can modify them for your design.

For an example ECO file, see the [ECO Directives](#) chapter in the *Innovus User Guide*.

## Assumptions

The ECO flows in this appendix assume the following:

- The old Verilog netlist and the new Verilog netlist have already been unqualified so that no Verilog module is instantiated more than once.
- Your design uses an existing floorplan.
- Your old placement, special routing, and routing information is saved in one of the Innovus formats, DEF, and so forth.

## Flows

This appendix describes various types of ECO flows:

- Pre-Mask ECO Changes from a New Verilog File  
If you make changes to the netlist, use this flow to use to load the new netlist and restore all the physical data from the previously saved design.
- Pre-Mask ECO Changes from a New DEF File  
Allows you to make external changes that include new cell placements from a DEF file, while preserving your previous placement, optimization, and optionally, previous routing information; for example, a clock tree with placements and specialized buffer insertion with placements.
- Pre-Mask ECO Changes from an ECO File  
Allows you to use an ECO file to make changes to the netlist.
- Post-Mask ECO Changes from a New Verilog Netlist (Using Spare Cells Flow)  
Allows you to make late logic changes after the masks are made. This flow uses pre-existing spare cells, so no poly/diffusion changes are allowed and only the routing is modified. You can direct the software to make routing changes only on specific layers.
- Post-Mask ECO Changes from a New Netlist (Using Gate Array Cells Flow)  
Allows you to make late logic changes after the masks are made. This flow uses pre-existing Gate Array Style Filler Cells (GACORE Site), so no poly/diffusion changes are allowed, and only the routing is modified. You can direct the software to make routing changes only on specific layers.
- Post-Mask ECO Changes from a New Verilog Netlist (Using Gate Array Filler Cells Flow)  
Allows you to make late logic changes after the masks are made. This flow uses pre-existing gate array style filler cells (CORE Site) to create new cells. No poly/diffusion changes are allowed, and only the routing is modified.

## Pre-Mask ECO Changes from a New Verilog File

In this flow, your design is placed and possibly routed, and you want to make a few changes. The changes are done before the masks are made, so it is a pre-mask ECO flow and there is no need to keep the original poly/diffusion patterns.

## Preparation

Before starting the flow steps, you should have the following files available:

- oldchip.fp

oldchip.fp.spr

Create these files (floorplan, special routing, placement, and routing) by using the `saveFPlan` command.

*or*

- oldchip.def

Create this file (DEF formats for floorplan, placement, special routing, optionally routing) by using the `defOut` command.

- newchip.v

The new Verilog file is typically created by manually editing the old Verilog netlist.

**Note:** If you want to preserve routing, your existing design must contain the antenna diode cells that were added during the previous routing.

## Flow

1. Read the new netlist
2. Load old floorplan/placement/routing data
3. Add level shifter or isolation cell for low power design (optional).
4. Remove filler cells and notches (optional)
5. Perform incremental placement
6. Add filler cells (optional)
7. Perform incremental or final route

Or,

Use the `ecoDesign` command to perform the pre-mask ECO operations.

## Steps

### 1. Read the new netlist.

```
source newchip.globals # same as oldchip.globals except use newchip.v
init_design
or
source oldchip.globals
set init_verilog "newchip.v"
init_design
```

The netlist includes old libraries, global power connections, and so forth. It typically uses old timing constraints. However, new timing constraints can be used.

### 2. Read the old floorplan, special routes, placements and routing from the old floorplan and def files.

```
loadFPlan oldchip.fp
ecoDefIn -reportFile ecoDefIn.rpt oldchip.def
applyGlobalNets
```

During this step,

- Matching instances receive old placements.  
Soft matching happens when a DEF net name does not have an equivalent name and another net is found in memory, that has the same connections as described in the DEF. The most common case for soft matching is when nets have multiple aliases in a hierarchical design "net1" = "inst1/net2" = "inst1/inst2/net3" and so on. Any of these net names can be used in the DEF and can have the same connections. Without soft matching, net `a', for example, is removed and net `b' is created in its place, resulting in ripping of the wire.
- Instances existing only in the oldchip.def file are ignored, so they are not added to the current netlist.
- Changed instances (new cell) are assigned a new cell and are left unplaced.
- Physical-only cells in the old netlist (marked with +SOURCE DIST in the DEF) get added (for example, well taps, end caps, and filler cells).
- Instances that are only in the new netlist are left unplaced.

- Routing for existing or modified nets is restored (possibly with opens or shorts).
  - Routing for deleted nets is removed.
3. (Optional) Add level shifter or isolation cell for low power design.
- ```
loadCPF test.cpf
commitCPF -keepRows
```
- During this step, level shifters or isolation cells will be added to new ECO nets that cross the power domain. Retain the old design row definitions. The newly added cells will be unplaced. The `ecoPlace` command will place them in their respective power domain boundaries.
4. Remove filler cells or notch fill (if present).
- ```
deleteFiller -prefix FILL
deleteNotchFill
```
5. Perform incremental placement.
- ```
ecoPlace
```
- Unplaced instances are placed, previously placed cells are not moved, and routing is unaffected. You can manually preplace critical cells before using the `ecoPlace` command by placing the cell in the bottom, left corner, selecting it, and then moving it graphically.
- ```
placeInstance i1/i2/i3 0 0
selectInst i1/i2/i3
```
6. Add filler cells back into the rows.
- ```
addFiller -cell {FILL4 FILL 2 FILL 1} -prefix FILL
```
- Global power connections are performed automatically based on rules loaded from the globals file or floorplan file earlier.
7. Perform incremental or final route.
- ```
ecoRoute
```

NanoRoute automatically detects modified and new nets and incrementally routes any nets that are incomplete or have violations.

Or,

Use the `ecoDesign` command to perform ECO operations. For example, the following command performs a pre-mask ECO:

```
ecoDesign original.enc.dat top_cell newchip.v
```

Use the `ecoDesign -noecoPlace` or `ecoDesign -noecoRoute` command to perform ECO operations. The command interrupts before `ecoPlace` or `ecoRoute`. This provides user more flexibility to control separate steps or perform some interactive ECO operations.

## Pre-Mask ECO Changes from a New DEF File

In this flow, your design is placed and possibly routed, and you can make a few changes with known cell placements from a new DEF file.

Examples of this flow include:

- Bringing in an external clock tree after placement
- Bringing in external optimizations such as new buffers or cell resizing
- Bringing in external postRoute fixes such as new buffers or cell resizing

## Preparation

Before starting the flow steps, you should have the following files available:

- `oldchip.enc`
- `oldchip.enc.dat/`

Create these files by using the `saveDesign` command after the previous placement, optimization, and routing steps.

or

- oldchip.globals
- oldchip.v
- oldchip.def

Create these files by using the `saveNetlist` and `defOut` commands after the previous placement, optimization and routing steps.

- newchip.def

The new DEF file is typically created by an external tool, or possibly done manually to fix a few critical postRoute violations with specific placements required. Any necessary physical cells (+SOURCE DIST) are expected to also be in the new DEF.

You must start with the old Verilog and update the Verilog modules with new ports and nets as required to match the new DEF netlist. You need to make sure the DEF instance names match the expected Verilog names (for example, a new buffer added to the output of instance /i1/i2/i3 should have a name such as /i1/i2/mynewbuf\_i100), otherwise spurious Verilog ports will be created.

## Flow

1. Read the old floorplan/netlist
2. Compare the old netlist to DEF
3. Load the ECO file
4. Write the modified netlist (optional)
5. Read the new placement data
6. Perform incremental or final routing

## Steps

1. Read the old Verilog netlist, floorplan, and placement information into Innovus by doing one of

the following:

`restoreDesign oldchip.enc`

*or*

`source oldchip.globals`

`init_design`

`defIn oldchip.def`

This step reads in the following information:

- Old libraries and global power connections
- Old timing constraints (could be new constraints if necessary)
- Special routing, placements and optionally old routing
- Old filler cells, end caps, well taps, and other cell information

2. Compare the current old netlist to the new DEF netlist to create an ECO file.

`ecoCompareNetlist -def newchip.def -outFile oldchip.eco`

The ECO file has all the changes required to make the current netlist match the new netlist. Physical-only cells are ignored (for example, `+SOURCE DIST` cells such as filler, end caps, and well taps). Examine the ECO file to ensure it is correct.

3. Load the ECO file to incrementally update the current netlist to match the new netlist.

`loadECO oldchip.eco`

During this step,

- Instances and nets that are only in the old Verilog are deleted (for example, an old clock tree). Some Verilog ports may now be unconnected due to deleted nets.
- New instances are still unplaced.
- New ports and nets are created in Verilog modules as needed to connect instances in different Verilog modules.
- If any nets are deleted, then any routing attached to the net is also deleted.
- If any nets are modified, then any routing on those nets is left unchanged for later repair.
- Global power connections are done automatically based on the rules from the globals file or floorplan file loaded earlier.

4. (Optional) Write out the modified Verilog netlist.

`saveNetlist oldchip_after_eco.v`

The `oldchip_after_eco.v` file should be the same netlist as `newchip.def`, although it is possible for the net names to be different (any new DEF net names that connect across multiple Verilog modules may be renamed). If you need a DEF file that has exactly the same net names, you can use the `defOut` command.

5. Read in the new placements.

`defIn newchip.def`

During this step,

- All instance placements are updated, including unplaced instances. Typically any existing old instances are not moved, but nothing prevents them from moving if the new DEF moved them.
- Remove the `deleteFiller` command before using `defIn` if the new DEF contains different fill, end cap, or well tap cells (+SOURCE DIST). If the filler cells are not changed, the `deleteFiller` command is not necessary. If the new DEF does not have any filler cells, the filler cells (if any) from the old DEF are left in place.
- If any instances are still unplaced, the `ecoPlace` command can be used to place them after removing any notch-fill or metal-fill wiring using the `editDelete` command.
- If only legalization of the placement is needed, the `refinePlace` command can be used.
- If routing is in the new DEF file (typically from the routing done on the old netlist), the routing will also be read in, and it will replace the routing on existing nets.

6. Perform incremental or final routing.

`ecoRoute`

`ecoRoute` automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.

7. Continue with the normal post-routing flow (analysis, repair, notch-fill, metal-fill, verify, sign-off, and so forth).

## Pre-Mask ECO Changes from an ECO File

In this flow, your design is placed and possibly routed, and you can make a small number of changes using an ECO file methodology. The changes are done before the masks are made so it is a pre-mask ECO flow, and there is no need to keep the original poly/diffusion patterns.

For example, you might want to apply a small number of late logical changes, but you want to keep as much of the previous placement, optimization, clock tree, and routing to avoid disturbing previous timing/SI optimization and repair.

## Preparation

Before starting the flow steps, you should have the following files available:

- oldchip.enc

oldchip.enc.dat/

Create these files by using the `saveDesign` command after the previous placement, optimization, and routing steps.

or

- oldchip.globals

oldchip.v

oldchip.def

Create the first three files by using the `saveNetlist` and `defOut` commands after the previous placement, optimization and routing steps. The new Verilog file (`newchip.v`) is typically created by manually editing the old Verilog netlist.

**Note:** If you want to preserve routing, your existing design must contain the antenna diode cells that were added during the previous routing.

or

- oldchip.eco

This file contains the list of changes to be applied to the old netlist. The changes required (see the `loadECO` command syntax) are typically created manually. You might be able to create an ECO file more easily by using `ADDINST` and `DELETEINST` rather than creating a new Verilog file.

## Flow

1. Read the old netlist
2. Load the ECO file
3. Write the new netlist (optional)
4. Remove filler cells or notch fill (if present)
5. Perform incremental placement
6. Add filler cells
7. Perform incremental or final routing

## Steps

1. Read the old Verilog netlist, floorplan, and placement information into Innovus.

`restoreDesign oldchip.enc`

or

`source oldchip.globals`

`init_design`

`defIn oldchip.def`

This step reads in the following information:

- Old libraries and global power connections
- Old timing constraints or new constraints, if necessary
- Special routing, placement, and optionally, old routing information
- Old filler cells, end caps, well taps, and other cell information

2. Load the ECO file to incrementally update the current netlist to match the new netlist.

`loadECO oldchip.eco`

During this step,

- Instances and nets that are only in the old Verilog are deleted (for example, an old clock tree). Some Verilog ports may now be unconnected due to deleted nets.
- New instances are still unplaced.

- New ports and nets are created in Verilog modules as needed to connect instances in different Verilog modules.
- If any nets are deleted, the routing attached to the net is also deleted.
- If any nets are modified, the routing on those nets is left unchanged for later repair.
- Global power connections are done automatically based on the rules from the globals file or floorplan file loaded earlier.

3. (Optional) Write out new Verilog netlist.

```
saveNetlist oldchip_after_eco.v
```

The `oldchip_after_eco.v` and `newchip.v` netlists should be identical, with one exception: the newly created Verilog module ports and nets might have different names because they are automatically generated whenever a new connection is made between separate Verilog modules.

4. Remove filler cells or notch fill (if present).

```
deleteFiller -prefix FILL
```

```
deleteNotchFill
```

5. Perform incremental placement.

```
ecoPlace
```

Unplaced instances are placed; however, previously placed cells are not moved and routing is unaffected.

**Note:** You can manually preplace critical cells before using the `ecoPlace` command by placing the cell in the bottom, left corner, selecting it, and then moving it graphically. For example:

```
placeInstance i1/i2/i3 0 0
```

```
selectInst i1/i2/i3
```

6. Add filler cells back into the rows.

```
addFiller -cell FILL4 -prefix FILL
```

Global power connections are done automatically based on rules loaded from the globals file

or floorplan file earlier.

7. Incremental or final route.

```
setNanoRouteMode -routeWithEco true      # set for incremental routing
globalDetailRoute
```

NanoRoute automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.

Or,

You can instead use the `ecoRoute` command to perform incremental or final routing.

8. Continue with the normal post-routing flow (analysis, repair, add metal fill, notch fill, verify, sign-off, and so forth).

**Note:** ECO file contains ECO directive commands. The syntax of these commands is different from that of ECO tcl commands. For detailed information, refer the ECO Directives section of the Innovus Text Command Reference.

## Post-Mask ECO Changes from a New Verilog Netlist (Using Spare Cells Flow)

Use this flow when:

- The design is taped out but has errors, or you need to add new features to the taped-out design.
- There is a new Verilog file that has only a few logical changes from the old Verilog file.
- You want to use the pre-existing spare cells so the poly/diffusion and lower layers are not changed, and only the metal and via layer masks need to be modified.

To save mask costs, you can direct the software to perform routing changes only on specific layers.

For this flow, you need the following files:

- `oldchip.enc*` (or `oldchip.globals`, `oldchip.fp*`, `oldchip.def`)
- `newchip.v` (this can be output from Conformal ECO Designer)

The original Verilog file already has spare cells because they are typically added by creating spare cells at the top-level or inside a Verilog module(s) just to hold the spare cells. The placer spreads

the spare cells evenly throughout the design. If the design is hierarchical, you can add more spare cells inside modules that are likely to change.

## Steps

### 1. Read the new netlist.

```
source newchip.globals # same as oldchip.globals except use newchip.v
init_design
or
source oldchip.globals
set init_verilog "newchip.v"
init_design
```

The netlist includes old libraries, global power connections, and so forth. It typically uses old timing constraints, but new timing constraints can be used.

### 2. Load old floorplan/placement/routing data.

```
loadFPlan oldchip.fp #(Optional: To be done if the DEF file does not include the
floorplan information)
ecoDefIn -postMask -reportFile ecoDefIn.rpt G1.pr.def
applyGlobalNets
```

During this step, the following happens:

- The `-postMask` option ensures that deleted items are also restored.
- Matching instances get old placements.
- When a DEF net name does not have a matching name in memory, soft matching happens. The tool matches the DEF net name to another net that has the same connections, as described in the DEF. The most common case for soft matching is when nets have multiple aliases in a hierarchical design. For example, `"net1" = "inst1/net2" = "inst1/inst2/net3"`. Any of these net names can be used in the DEF and can have the same connections. Without soft matching, net "a", for example, is

removed and net "b" is created in its place, resulting in ripping of the wire.

- Any instance that exists only in the oldchip.def file (deleted cells) is kept in the design, and its name is appended with the string specified by -suffix.
- For example, if you specify -suffix \_spare, instance i1/i2/i3 is changed to i1/i2/i3\_spare.
- Changed instances (new cell) are assigned a new cell and are left unplaced.
- Physical-only cells in the oldchip.def file (marked with +SOURCE DIST in the DEF) are added; for example, well taps, end caps, and filler cells.
- Instances that are only in the new netlist are left unplaced.
- Routing for existing or modified nets is restored (possibly with opens or shorts).
- Routing for deleted nets is also restored. The `ecoRoute` command removes the nets according to the -modifyOnlyLayer option.
- All unplaced cells are mapped to spare cells during ECO placement in a later step.

### 3. (Optional) Low power related changes.

```
loadCPF test.cpf  
commitCPF -keepRows
```

During this step, the following happens:

- Level shifters or isolation cells will be added to new ECO nets that cross the power domain.
- Old design row definitions are retained.
- Newly added cells are unplaced. The `ecoPlace` command will map them to spare cells.

### 4. Specify the spare cell list.

```
specifySpareGate -inst SPARE*
```

### 5. (Optional) Remove notch fill (if present).

```
deleteNotchFill
```

**Note:** Do not perform this step if you plan to freeze metal layers, because this command modifies all layers.

### 6. Perform incremental placement.

```
ecoPlace -useSpareCells true
```

In the post-mask flow, you must specify `-useSpareCells` to ensure that `ecoPlace` is switched to mapping mode. In this mode, `ecoPlace` maps all unplaced cells to spare cells with the same cell type.

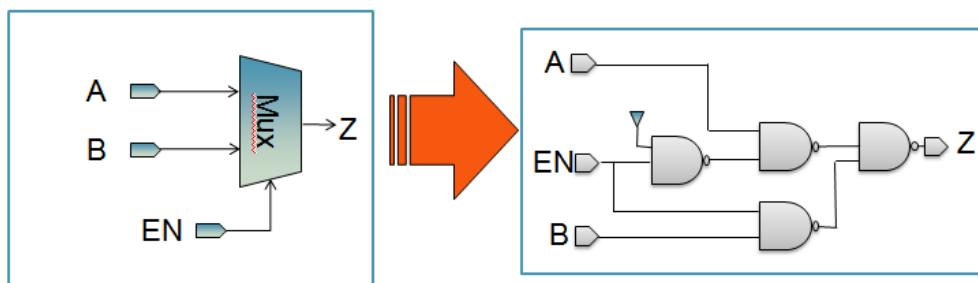
The `ecoPlace` command automatically chooses a spare cell that is identical to the cell being mapped.

```
ecoRemap [-allowConstantTie number] [-useGACells {techSiteName} | -  
useGAFillerCells {cellName ...}] [-useGAStdCells { cellName ...}] [-rcSpareMapFile  
fileName]
```

The `ecoRemap` command might map the unplaced cell to a combination of other cell types to achieve a similar function and yield better DRC and timing results. It needs a timing library so that it can compute timing slack.

The following is an example of `ecoRemap`:

```
ecoRemap -allowConstantTie 1
```



By connecting one input pin to tieHi, Nand logic cones combine together to realize the mux function.

## 7. (Optional) Swap spare cells.

```
ecoSwapSpareCell i_9649 spare1
```

At this step of the flow, all the newly added cells should be mapped. In this step, you can use the `ecoSwapSpareCell` command to change the mapping manually.

`i_9649` is the instance name of the placed cell that `ecoSwapSpareCell` swaps with spare cell `spare1` must be a spare cell specified in the previous step. Use

the `specifySpareGate` command if necessary.

8. (Optional) Make tie connections.

```
addTieHiLo -postMask [-cell "tieHighCellName tieLowCellName"] [-createHierPort  
{true | false}]
```

During this step, the software reuses the existing tie cells to tie off a newly created spare instance in the design, instead of adding or deleting tie cells.

9. Perform incremental or final routing.

```
ecoRoute -modifyOnlyLayers 2:3
```

You can use the `-modifyOnlyLayers` option to restrict the modifications to a specified range of metal layers. If the `-modifyOnlyLayers` range begins with layer 2, and the spare cell pins are only available from metal 1, then the `ecoRoute` command automatically drops a VIA12 via. This behavior is not available if the `-modifyOnlyLayers` range does not begin with 2.

The `ecoRoute` command might not be successful if the specified layer range is not sufficient to meet the changes required. You must restore the design from the previous step, then use a different range, such as 2:4, 1:3, and so on.

The unused routing segments of deleted and modified nets will appear in the SPECIALNETS section of the DEF file.

OR,

Use the `ecoDesign` command to perform post-mask ECO operations. The following command and options are used to implement a post-mask ECO:

```
ecoDesign -postMask -modifyOnlyLayers 2:3 -spareCells *spare* original.enc.dat  
top_cell newchip.v
```

## Post-Mask ECO Changes from a New Netlist (Using Gate Array Cells Flow)

Use this flow when:

- The design is taped out but has some errors, or you need to add new features to the taped-out design.
- There is a new Verilog netlist that has a few logical changes from the old Verilog netlist.
- You want to use pre-existing gate array style filler cells that can be programmed with metal layers so the poly/diffusion and lower layers are not changed, and only the metal and via layer masks need to be modified.
- The new netlist can be output from Conformal ECO or created through manual changes. The new instances can be a combination of standard cells and GACells (with GACORE site).
- Before running the flow, the GACORE library should be ready and GACORE filler cells must be inserted into the design. The prepared files include old design database and new netlist (this may be the output from Conformal ECO).
- The GACORE library has the following features:
  - All cells have a common transistor pattern.
  - The cells are a fixed number of GACORE sites wide. For example, the width of a GACORE site might be four times the width of a CORE site.
  - The logical cells are programmed by metal1 for various AND and OR type gates.
  - Filler cells use the same transistor pattern (for example, GAfiller).

## Steps

1. Read the new netlist.

```
source newchip.globals # same as oldchip.globals except use newchip.v
init_design
or
source oldchip.globals
set init_verilog "newchip.v"
init_design
```

The netlist includes old libraries, global power connections, and so on. It typically uses old

timing constraints, but new timing constraints can also be used.

2. Read the old floorplan, special routes, placements, and routing from the old netlist files.

```
loadFPlan oldchip.fp # Optional: To be done if DEF file does not include the floorplan information:
```

```
ecoDefIn -useGACells GACORE -suffix _spare -reportFile ecoDefIn.rpt G1.pr.def  
applyGlobalNets
```

During this step:

- GACORE cells that are only in the old DEF are deleted (it will leave a hole in the layout and this space can be reused through placing a new instance of GACORE). There are some GACORE function cells that do not exist in new netlist, if you do not use the option -useGACells for ecoDefIn, these cells will become spare cells like regular standard cells.
- This procedure reads in the following information:
  - Special routing, placements, and old routing
  - Old filler cells, end caps, well taps, and other cell information
- Regular standard cells that are only in the old DEF file are implicitly deleted by leaving them in place and changing the name from i1/i2/i3 to i1/i2/i3\_SPARE. The input pins of these new spare cells are tied to the ground net or tie-low cell.
- New instances are left unplaced.
- Global power connections are made automatically based on the rules from the globals file or floorplan file loaded earlier.

Any GACORE rows in the old design are restored; normal CORE rows are also restored. GACORE rows could optionally come from a separate DEF file if they are not saved with the old design.

3. (Optional) Specify spare gates

```
specifySpareGate -inst *SPARE*
```

4. (Optional) Remove notch fill

```
deleteNotchFill
```

**Note:** Do not do this step if you plan to freeze metal layers, because this command modifies all layers.

5. Perform incremental placement.

```
ecoPlace -useSpareCells true (#optional)  
deleteFiller -prefix GAFILL
```

#### Fixed all cells in the design

```
ecoPlace -useGACells GACORE  
addFiller -cell GAFiller -prefix GAFILL
```

This step does the following:

- If there are standard cells (such as site CORE) and GAcells (such as site GACORE), both need to be placed. You need to use the `ecoPlace` command twice. First, using the spare cell for unplaced standard cells, and then using the GA filler cells for unplaced GA function cell.
- Removes GACORE filler cells to leave gaps for the `ecoPlace`. The `ecoPlace` command snaps GACORE cells to the GACORE row sites. Routing is unaffected.
- Puts back the GACORE filler cells in any leftover gaps.
- `ecoPlace` maps unplaced std cell to the same function spare cell. `ecoPlace` places the GACORE cells in a legal placement location.

#### 6. (Optional) Swap spare cells.

```
ecoSwapSpareCell i_9649 spare1
```

During this step, all the newly added cells should be mapped.

In this step, you can use the `ecoSwapSpareCell` command to manually change the mapping. `i_9649` is the instance name of the placed cell that `ecoSwapSpareCell` swaps with spare cell `spare1`. `spare1` must be a spare cell specified in the previous step. Use the `specifySpareGate` command if necessary.

#### 7. (Optional) Make tie connections.

```
addTieHiLo -postMask [-cell "tieHighCellName tieLowCellName"] [-createHierPort  
{true | false}]
```

During this step, the software reuses the existing tie cells to tie off a newly created spare instance in the design, instead of adding or deleting tie cells.

8. Perform incremental or final route.

```
ecoRoute -modifyOnlyLayers 2:3
```

During this step:

- NanoRoute automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.
- The insertion of antenna diode cells is disabled. The poly/diffusion layers cannot be modified, so only layer-hopping can be used to avoid process antenna violations.
- You can use the `-modifyOnlyLayers` option to restrict the modifications to a specified range of metal layers.
- The `ecoRoute` command might not be successful if the specified layer range is not sufficient to meet the changes required. You must restore the design from the previous step, then use a different range, such as 2:4, 1:3, and so on.
- The unused routing segments of deleted and modified nets will appear in the SPECIALNETS section of the DEF file.

## Post-Mask ECO Changes from a New Verilog Netlist (Using Gate Array Filler Cells Flow)

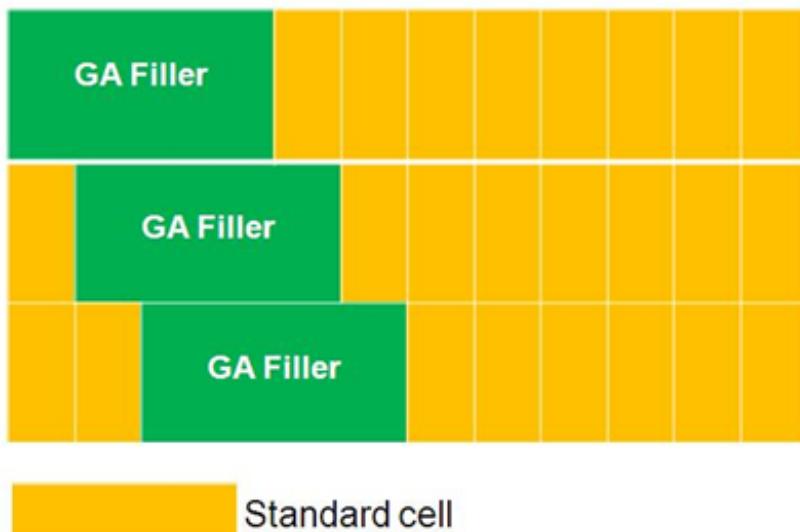
Currently, `ecoPlace` supports GACells and GAFillerCells flows. GACells flow is GACORE site based filler insertion while GAFillerCells is CORE site based filler insertion. Using the GAFillerCells flow, the tool can insert a GA Filler at any arbitrary grid rather than at the GACORE site grid to provide more available locations for post-mask ECOs.

The following diagrams illustrate the insertion of GA Fillers for GACells flow and GAFillerCells flow.

**Figure 2 GA Fillers Insertion for GACells Flow**



**Figure 3**



### GA Fillers Insertion for GAFillerCells Flow

To perform incremental placement:

```
ecoPlace -useSpareCells true (#optional)
```

Unfix all GAFillers in the design

```
ecoPlace -useGAFillerCells {List of GAFillerCells }
```

This procedure does the following:

- If you need to place standard cells (such as site CORE) and GACells (such as site GACORE),

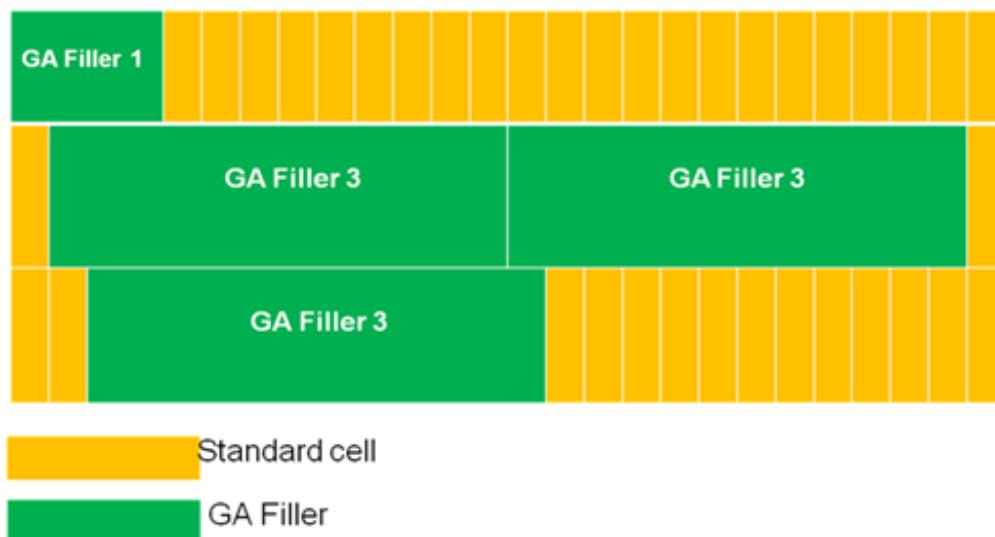
you need to use the `ecoPlace` command twice. First, using the spare cell for unplaced standard cells, and then using the GA filler cells for unplaced GA function cell. Since the tool cannot distinguish standard cells from gate-array cells, you must first specify the GAFillerCell list. Change the GAFillers status from fixed to placed status.

- `ecoPlace` maps unplaced std cells to the same function spare cell (optional).
- `ecoPlace` placed GACells overlap with existing GA Fillers based on the connectivity, deleting the overlapping original GA Fillers and filling in the gap using new smaller GA Fillers.

Figure 3 and 4 illustrate the replacement of GA Fillers:

**Figure 4**

**Before ecoPlace -useGAFillerCells {GAFILL1 GAFILL2 GAFILL3}**



**Figure 5**

**After ecoPlace -useGAFillerCells {GAFILL1 GAFILL2 GAFILL3}**



# ECO Directives

This appendix describes the directives that you specify in an ECO directives file. After you complete the file, you can then read it into the Innovus™ Implementation System (Innovus) by using the `loadECO` command on the Innovus command line. The following command loads `myDirectivesFile`:

```
loadECO myDirectivesFile
```

 These are ECO directives, not Innovus Tcl commands.

- You can use these directives only in an ECO directives file.
- You cannot use these directives on the Innovus command line.
- You cannot source this file to run the directives.
- You must use the `loadECO` command to read the file.

The names of the directives appear in this appendix in uppercase characters to distinguish them from interactive Innovus commands with the same names; however, the software is case-insensitive.

The ECO File directives do not support Verilog® escape name syntax. For directives that modify or delete existing objects, you cannot specify the name of the object using Verilog escape name syntax.

File format requirements are shown in the section [Example ECO File](#).

The directives are presented in alphabetical order.

- [ADDHIERINST](#)
- [ADDINST](#)
- [ADDMODULEPORT](#)
- [ADDNET](#)
- [ATTACHMODULEPORT](#)
- [ATTACHTERM](#)
- [CHANGECELL](#)
- [CHANGEINST](#)
- [CHANGEINSTNAME](#)

- [DELETEBUFFER](#)
- [DELETEINST](#)
- [DELETEMODULEPORT](#)
- [DELETENET](#)
- [DETACHMODULEPORT](#)
- [DETACHTERM](#)
- [INSERTBUFFER](#)
- [Example ECO File](#)

## ADDHIERINST

ADDHIERINST

*instName*

*moduleName*

Creates an instance of a new hierarchical module. You can later use the `ADDINST` directive to add spare cells inside the new hierarchical instance. The software automatically creates new ports for the hierarchical module when you run the `ATTACHTERM` directive. Currently, there are no directives available that allow you to manually create new ports for the new hierarchical module.

## Parameters

<i>instName</i>	Specifies the name of the new hierarchical instance. If the instance already exists, or if the module containing the instance does not exist, the directive stops and the software displays an error message.
<i>moduleName</i>	Specifies the name of the new hierarchical module. If the module already exists, the software uses the module definition and creates a new hierarchy.

## Example

- The following directive creates a new hierarchical cell, sparecell, and an instance of sparecell named i1/i2/i3/spare1:

```
ADDHIERINST i1/i2/i3/spare1 sparecell
```

If the instance `i1/i2/i3` does not exist, or instance `i1/i2/i3/spare1` already exists, the directive stops and the software displays an error message.

## ADDINST

```
ADDINST
[-moduleBased moduleName
]
cellName instName
```

Adds an instance.

When `instName` is specified, the new instance is bound with the correct cell in the power domain.

**Note:** If `nrTerm` is greater than zero, then you specify `nrTerm` lines of `INSTTERM...` after the line `ADDINST...nrTerm`.

If ADDINST is module based, the syntax is:

```
ADDINST -moduleBased moduleName cellName instName
```

If ADDINST is not module-based, the syntax is:

```
ADDINST instName cellName nrTerm
INSTTERM termName netName
```

## Parameters

<i>cellName</i>	Specifies the master of the instance.
<i>instName</i>	Specifies the name of the instance to add and place.
-moduleBased <i>moduleName</i>	Adds the new instance to the specified verilog module.
<i>netName</i>	Specifies the net name.
<i>termName</i>	Specifies the terminal name.
<i>nrTerm</i>	Number of terminals of an instance.

## Example

- The following directive adds an instance `BUF1` having two terminals `A` and `Y` connecting to nets `net_a` and `net_b`:

```
ADDINST BUF1 BUF 2
```

```
INSTTERM A net_a
```

```
INSTTERM Y net_b
```

## ADDMODULEPORT

```
ADDMODULEPORT
```

```
moduleName
```

```
| '-'
```

```
portName
```

```
{input | output | bidi}
```

```
[-bus n1 :n2]
```

Adds a port or a bussed port to a module.

## Parameters

-bus <i>n1:n2</i>	Adds a bussed port to the module. Specify the bus range (the beginning and end of the bus). Use integers to specify the range.
<i>moduleName</i>   '-'	Specifies the module to which you want to attach the port. To specify the top module, enter '-'.
<i>portName</i>	<p>Specifies the name of the port to be added.</p> <p>The specified <i>portName</i> can be a new port name or any of the existing net names to which you want to add the port.</p> <p>The new port name can be the same as the existing net name. This port is attached directly to the existing net name if you specify the port direction (scalar port).</p>
input   output   bidi	Specifies whether the port is input, output, or bidirectional.

## Examples

- The following directive creates an input port *p1* on instance *i1/i2/i3*. The port *p1* must be a new port name in instance *i1/i2/i3*. Instance *i1/i2/i3* contains no nets name *p1*:

```
ADDMODULEPORT i1/i2/i3 p1 input
```

- The following set of directives adds a hierarchical block and then adds a bussed port to the block.

```
ADDHIERINST i_block1 i_block
```

```
ADDMODULEPORT i_block1 data output -bus 31:0
```

## ADDNET

```
ADDNET
[-moduleBased verilogModule
]
netName
[-physical]
```

**[-bus *startID*:*endID*]**

Adds a net to the design. The net can be logical or physical.

## Parameters

-bus <i>startID</i> : <i>endID</i>	Creates a bussed Verilog net. The <i>startID</i> and <i>endID</i> indicate the first and last bits on the bus. You must separate the start and end IDs with a colon (:).
-moduleBased <i>verilogModule</i>	Adds the new net to the specified verilog module.
<i>netName</i>	Specifies the name of the net to add. If the module containing the net does not exist, or if the net already exists, the software displays an error message and the directive stops.
-physical	Adds a physical net.

## Example

- The following directive adds net i1/i2/net26 to the netlist:

```
ADDNET i1/i2/net26
```

If the module i1/i2 does not exist, or if the net i1/i2/net26 already exists, the software displays an error message and the directive stops.

## ATTACHMODULEPORT

```
ATTACHMODULEPORT
{ moduleName
| '-'}
portName
netName
```

Attaches a port in the specified instance (or top level) to a net.

## Parameters

<i>moduleName   '-'</i>	Specifies the module to which you want to attach the port. To specify the top module, enter '-'.
<i>netName</i>	Specifies the net to which you want to create to attach to the port.
<i>portName</i>	Specifies the port you want to create on the module.

## Examples

- The following directives create a port *p1* on instance *i1/i2/i3* and a net *i1/i2/n1*. The ATTACHMODULEPORT directive then connects the created port *p1* on instance *i1/ i2/i3* to the net *i1/i2/n1*:

```
ADDMODULEPORT i1/i2/i3 p1 input
ADDNET i1/i2/n1
ATTACHMODULEPORT i1/i2/i3 p1 i1/i2/n1
```

- The following directive attaches port *in* on the top module to *net123*:

```
ATTACHMODULEPORT - in net123
```

## ATTACHTERM

```
ATTACHTERM
[-moduleBased verilogModule
]
[-noNewPort]
instName
termName
netName
[-port portName | -pin refInstName refPinName ]
```

Attaches a terminal to a net. If the terminal already connects to the net, the software first detaches the terminal from the current net, then attaches it to the new net.

## Parameters

<i>instName</i>	Specifies the instance containing the terminal. If the instance name does not exist, the software displays an error message and the directive stops.
-moduleBased <i>verilogModule</i>	Attaches the terminal to the specified verilog module.
<i>netName</i>	Specifies the name of the net to attach to the terminal. If the net name does not exist, Innovus displays an error message and the directive stops.
-noNewPort	Prohibits Innovus from creating hierarchical ports when it attaches a terminal. If the terminal cannot connect to the net through existing ports, Innovus displays an error message and the directive stops. <i>Default:</i> If you do not specify this parameter, Innovus creates hierarchical ports as needed.
-pin <i>refInstName</i> <i>refPinName</i>	Specifies the pin <i>refPinName</i> on instance <i>refInstName</i> connected to the net that Innovus connects to the terminal. You cannot specify the -port parameter if you use this parameter.
-port <i>portName</i>	Specifies the hierarchical port used to connect the terminal with the net. If you specify this parameter, the hierarchical port must exist in the module that contains the instance. The hierarchical port must connect to the net. This parameter lets you use a specific port to maintain the same netlist topology, which simplifies equivalence checking later in the design flow. You cannot specify the -pin parameter if you use this parameter. <i>Default:</i> If you do not specify this parameter, Innovus uses existing ports or creates new hierarchical ports as necessary to connect the terminal to the net.
<i>termName</i>	Specifies the name of the terminal that Innovus connects to the specified net. If the terminal name does not exist, Innovus displays an error message and the directive stops.

## Examples

- The following directive attaches terminal `in1` of instance `i1/i2/i3` to net `i1/i2/net26`:

```
ATTACHTERM i1/i2/i3 in1 i1/i2/net26
```

If `i1/i2/i3` does not exist, or if `in1` is not a terminal of `i1/i2/i3`, or if `i1/i2/net26` does not exist, the software displays an error message and the directive stops.

- The following directive attaches terminal `in2` of instance `i1/i2/i3` to net `net27`, using hierarchical port `myPort`:

```
ATTACHTERM i1/i2/i3 in2 net27 myPort
```

The `myPort` port must exist in the module definition for `i1/i2`, and `myPort` must already connect to `net27`. If this is not the case, the software displays an error message and the directive stops.

- The following directive attaches terminal `in3` on instance `i1/i2/i3` through existing ports to `net28`:

```
ATTACHTERM -noNewPorts i1/i2/i3 in3 net28
```

If ports do not exist, the software displays an error message and the directive stops.

- The following directive attaches terminal `Y` of instance `testInst` to the verilog module `hier_t3` using the port `testPort`:

```
ATTACHTERM -moduleBased hier_t3 testInst Y testPort
```

## CHANGECELL

```
CHANGECELL
instName
newCellName
[oldCellName]
```

Changes the cell type of an instance to a new cell type. You can only change cells that have the same footprint and functionality. If you provide the existing cell type (`oldCellName`), this directive verifies that the specified instance (`instName`) is a current instance of the existing cell type before

changing the cell type to a new type (`newCellName`).

If the current instance is already placed, the modified instance keeps the existing placement. Keeping the old placement might cause overlap violations between the modified instance and neighboring instances if the new cell is larger than the existing cell.

## Parameters

<code>instName</code>	Specifies the name of the instance whose cell type you want to change.
<code>newCellName</code>	Specifies the new cell type. The <code>newCellName</code> must exist in the library; otherwise, the software displays an error message and the directive stops.
<code>oldCellName</code>	Specifies the existing cell type. If the current cell is not <code>oldCellName</code> , the software displays an error message and the directive stops.

## Example

The following directive changes the instance `i1/i2/i3` of type `and2_1x` to cell type `and2_2x`:

```
CHANGECELL i1/i2/i3 and2_2x and2_1x
```

If `i1/i2/i3` does not exist, the software displays an error message and the directive stops. If `and2_2x` does not exist in the library, or if all terminals of `and2_2x` do not exactly match the terminals of the current cell, or if the current cell is not `and2_1x`, the software displays an error message and the directive stops.

## CHANGEINST

```
CHANGEINST
instName
newCellName
oldCellName
nrNewCellFTerm
TERM newTermName oldTermName
```

Changes the cell type (master) for an instance. Unlike the `CHANGECELL` directive, you can exchange the cell type for one with a different footprint but identical functionality. If you provide the existing cell type (`oldCellName`), the `CHANGEINST` directive verifies that the specified instance (`instName`) is a

current instance of the existing cell type before changing the cell type to a new type (`newCellName`).

If the current instance is already placed, the modified instance keeps the existing placement. Keeping the old placement might cause overlap violations between the modified instance and neighboring instances if the new cell is larger than the existing cell.

## Parameters

<code>instName</code>	Specifies the name of the instance you want to change.
<code>newCellName</code>	Specifies the new cell type. The <code>newCellName</code> must exist in the library and all terminals of <code>newCellName</code> must match the terminals of the current cell type exactly; otherwise, the software displays an error message and the directive stops.
<code>nrNewCellFTerm</code>	Specifies the number of terminals belonging to the new cell type.
<code>oldCellName</code>	Specifies the existing cell type. If the current cell is not <code>oldCellName</code> , the software displays an error message and the directive stops.
<code>TERM</code>	<p>Specifies the old and new terminal names for the instance. You must specify a TERM directive for every terminal name that must change.</p> <ul style="list-style-type: none"> <li>● <code>newTermName</code>: Specifies the name of the terminal on the changed instance.</li> <li>● <code>oldTermName</code>: Specifies the name of the terminal on the existing instance.</li> </ul>

## Example

The following directives exchange a three-terminal flip-flop master (FF3) for a four-terminal master (FF4) for instance `i1/i2`.

```
CHANGEINST i1/i2 FF4 FF3 4
TERM D D
TERM Q Q
TERM CK CLK
```

Suppose that `FF3` has input `D` connected to net `n1`, clock terminal `CK` connected to net `clock`, and output `Q` connected to `n2`. `FF4` has input `D` connected to `n1`, clock terminal `CLK` connected to net `clock`, output `Q` connected to `n2`. Flip-flop `FF4` also has output `QN`. When the master for `i2/i2` is changed to

FF3, QN is not connected to a net.

## CHANGEINSTNAME

CHANGEINSTNAME  
*instName*

*baseName*

Changes the base name of the specified instance to the given base name.

**Note:** The `CHANGEINSTNAME` directive does not change the path of hierarchy.

### Parameters

<i>instName</i>	Specifies the name of the instance.
<i>baseName</i>	Specifies the new base name to use for the instance.

## DELETEBUFFER

DELETEBUFFER  
*instName*

*keepNetName*  
[*deleteNetName*]

Deletes a buffer instance after merging the nets on both sides of the buffer into one net.

 The `ecoDeleteRepeater` command provides the equivalent functionality.

## Parameters

<i>deleteNetName</i>	Specifies the name of the net connected to the terminal on <i>instName</i> opposite to the terminal that connects to <i>keepNetName</i> . If <i>deleteNetName</i> is not connected to the terminal on <i>instName</i> opposite to the terminal that connects to <i>netName</i> , the software displays an error message and the directive stops. For example, if <i>keepNetName</i> connects to the input of <i>instName</i> , <i>deleteNetName</i> specifies the name of the net connecting to the output. The software uses the <i>deleteNetName</i> for error checking only. When the <b>DELETEBUFFER</b> directive merges the nets, it might detach connections to hierarchical ports, but does not change the direction of hierarchical ports.
<i>instName</i>	Specifies the name of the buffer instance that the <b>DELETEBUFFER</b> directive deletes. The instance must have exactly one input terminal and one output terminal, and one of the terminals must connect to <i>keepNetName</i> .
<i>keepNetName</i>	Specifies the name of the existing net into which the nets from both of the instance's terminals are merged. The <i>keepNetName</i> net must connect to one of the instance's terminals.

## Example

- The following directive deletes buffer *i1/i2/i3* and merges the nets from the buffer's two terminals into net *net26*:

```
DELETEBUFFER i1/i2/i3 net26 i1/net25
```

Net *net26* already connects to one of the instance's terminals. Net *i1/net25* connects to the terminal opposite the terminal that connects to net *net26*. If buffer *i1/i2/i3* does not exist, or if net *net26* and net *i1/net25* are not already attached to two terminals of buffer *i1/i2/i3*, the software displays an error message and the directive stops.

## DELETEINST

```
DELETEINST
[-moduleBased verilogModule]
```

]  
*instName*

Deletes an instance after deleting all the instance terminal connections to nets.

## Parameters

<i>instName</i>	Specifies the name of the instance to delete. If the specified instance does not exist, the software displays an error message and the directive stops.
-moduleBased <i>verilogModule</i>	Deletes the instance from the specified verilog module.

## Example

- The following directive deletes instance `i1/i2/i3`:

```
DELETEINST i1/i2/i3
```

If instance `i1/i2/i3` does not exist, the software displays an error message and the directive stops.

- The following directive deletes the instance `insta` from the verilog module `HIER_2`:

```
DELETEINST -moduleBased HIER_2 insta
```

## DELETEMODULEPORT

DELETEMODULEPORT

*moduleName*

| '-'

*portName*

*netName*

Disconnects the specified port from its net and deletes the port.

You can use wildcards (`*?`) to specify the nets you want Innovus to delete.

## Parameters

<i>moduleName</i>   '-'	Specifies the module from which you want to delete the port. To specify the top module, enter '-'.
<i>netName</i>	Specifies the name of the net from which you want to delete the port.
<i>portName</i>	Specifies the name of the port to be deleted

## Example

- The following directive deletes *port1* from the top-level module:

```
DELETEMODULEPORT - port1
```

## DELETENET

```
DELETENET
[-moduleBased verilogModule
]
netName
```

Deletes a net after deleting all the instance terminal connections to the net. If routing is connected to the net, the routing is deleted.

You can use wildcards (\*?) to specify the nets you want Innovus to delete.

## Parameters

-moduleBased <i>verilogModule</i>	Deletes the net from the specified verilog module.
<i>netName</i>	Specifies the name of the net to delete.

## Example

- The following directive deletes net i1/i2/net26:

```
DELETENET i1/i2/net26
```

If net i1/i2/net26 does not exist, the software displays an error message and the directive stops.

## DETACHMODULEPORT

DETACHMODULEPORT

*moduleName*

*portName*

Detaches the net connected to the specified port on the specified instance.

## Parameters

<i>moduleName</i>	Specifies the module from which you want to detach the net. To specify the top module, enter '-'.
<i>portName</i>	Specifies the port on the module.

## Example

- The following directive detaches port p1 from moduleA:

```
DETACHMODULEPORT moduleA p1
```

## DETACHTERM

DETACHTERM

[ -moduleBased *verilogModule*

]

*instName*

*termName*

[*netName* ]

Disconnects a terminal from a net.

**Note:** Detaching a terminal that drives an output terminal of a module produces a Verilog violation at the output terminal if you use DETACHTERM. Instead, use ATTACHTERM to attach the terminal to a new net. The ATTACHTERM directive automatically detaches the terminal from the net connecting to the output terminal, then attaches the terminal to the net you specify.

## Parameters

<i>instName</i>	Specifies the instance that contains the terminal you want to detach.
-moduleBased <i>verilogModule</i>	Detaches the terminal from the verilog module.
<i>netName</i>	Specifies the net that is already connected to the terminal. If the terminal does not connect to the specified net, or if the net does not exist, the software displays an error message and the directive stops. The software uses this parameter for error checking only.
<i>termName</i>	Specifies the terminal to disconnect. If the terminal does not exist on the specified instance, the software displays an error message and the directive stops.

## Example

- The following directive disconnects terminal *in1* of instance *i1/i2/i3*:

```
DETACHTERM i1/i2/i3 in1 i1/i2/net26
```

If instance *i1/i2/i3* does not exist, or terminal *in1* is not a terminal of *i1/i2/i3*, the software displays an error message and the directive stops. The net *i1/i2/net26* is specified, so if net *i1/i2/net26* does not exist, or if the terminal is not already connected to net *i1/i2/net26*, the software displays an error message and the directive stops.

## INSERTBUFFER

INSERTBUFFER

[ -noNewPorts]

*netName*

*nrNetTerm*

*nrBuffer*

INST *instName cellName nrInstTerm*

INSTTERM *termName netName [portName ]*

...

NETTERM *instName termName netName*

...

Inserts a buffer on a net.

 This directive has been replaced by [insertRepeater](#) command.

**Note:** You must specify the INST, INSTTERM, and NETTERM directives in the order given in the syntax. You must enter each of these directives on its own line, at the beginning of that line. You can add these directives only after you have specified the [-noNewPorts] *netName nrNetTerm nrBuffer* parameters.

## Parameters

<i>netName</i>	Specifies the name of the net on which to insert the buffer. You must specify this parameter.	
- <i>noNewPorts</i>	Specifies that Innovus must not add new ports when inserting the buffer. If you try to insert a buffer that needs a new port, Innovus issues an error message and does not insert the buffer.	
<i>nrBuffer</i>	Specifies the number of buffers attached to the net.	
<i>nrNetTerm</i>	Specifies the original number of terminals attached to the net.	
<b>INST</b>	Specifies a buffer instance. You must specify the <b>INST</b> directive for each buffer you want to add.	
	<i>instName</i>	Specifies the name of the buffer instance to insert.
	<i>cellName</i>	Specifies the cell master for the buffer instance.
	<i>nrInstTerm</i>	Specifies the number of instance terminals contained in the buffer. Buffers have one input and one output terminal, so specify 2.
<b>INSTTERM</b>	Specifies a terminal on a buffer instance. The <i>termName</i> and <i>netName</i> parameters are required. you must specify the <b>INSTTERM</b> directive for each buffer you want to add.	
	<i>termName</i>	Specifies the name of the terminal on the buffer.
	<i>netName</i>	Specifies the net to connect with the terminal.
	<i>portName</i>	Specifies the physical port corresponding to the terminal.
<b>NETTERM</b>	Specifies the net connection between a driver terminal and the added buffer.	
	<i>instName</i>	Specifies the instance containing the terminal.
	<i>netName</i>	Specifies the net connected to the terminal.
	<i>termName</i>	Specifies the terminal connected to the net.

## Example

- The following directives insert three buffers, b1, b2, and b3, on net0, which originally connects terminal out on instance i0 to receivers i1, i2, and i3. After the three buffers are added, terminal out drives one terminal: b1/in.

```
INSERTBUFFER net0 4 3

INST b1 buffer 2
INSTTERM in net0

INSTTERM out net1
INST b2 buffer 2
INSTTERM in net1
INSTTERM out net2
INST b3 buffer 2
INSTTERM in net1
INSTTERM out net3
NETTERM i0 out net0
NETTERM i1 in net2
NETTERM i2 in net1
NETTERM i3 in net3
```

- Buffer b1 has terminal `in`, connected to `net0` and `out`, connected to `net1`. Buffer b1 drives buffers b2 and b3, and connects to receiver i2.
- Buffer b2 has terminal `in`, connected to b1 through `net1`, and `out`, connected to receiver i1 through `net2`.
- Buffer b3 has terminal `in`, connected to b1 through `net1`, and `out`, connected to receiver i3 through `net3`.

## Example ECO File

The file format consists of directives, each ending with a newline. The keywords are case insensitive.

Comments must begin with a pound symbol (#) as the leading, non-white space character, and end with a newline.

**i** The first directive in the file must be FORMATVERSION 2.

```
#  
# FORMATVERSION  
#  
FORMATVERSION 2  
#  
# ADDINST: Add at top level, no connectivity  
#  
ADDINST eco_inst_19 BUFX1  
  
#  
# ADDINST: Add at block level, no connectivity  
#  
ADDINST DTMF_INST/TDSP_CORE_INST/eco_inst_1 BUFX1  
  
#  
# ADDINST: Add at top level, with connectivity  
#  
ADDINST eco_inst_2341 BUFX1 2  
INSTTERM A test_mode  
INSTTERM Y reset  
  
#  
# ADDINST: Add at block level, with connectivity  
#  
ADDINST DTMF_INST/TDSP_CORE_INST/eco_inst_3 BUFX1 2
```

```
INSTTERM A scan_en
INSTTERM Y reset

#
# DELETEINST: Delete block level instance
#
DELETEINST DTMF_INST/m_clk_L6_I6

#
# ADDNET: Add new top level net
#
ADDNET eco_new_top_net

#
# ADDNET: Add new block level net
#
ADDNET DTMF_INST/eco_new_block_net

#
# DELETENET: Delete top level net
#
DELETENET n_7875

#
# DELETENET: Delete block level net
#
DELETENET DTMF_INST/TDSP_CORE_INST/ALU_32_INST/n_1496
```

```
#  
# ATTACHTERM: Attach block level inst term to existing net  
#  
ATTACHTERM DTMF_INST/TDSP_CORE_INST/ACCUM_STAT_INST/i_9529 A  
DTMF_INST/TDSP_CORE_INST/ACCUM_STAT_INST/n_73  
  
#  
# DETACHTERM: Detach block level inst term  
#  
DETACHTERM DTMF_INST/TDSP_CORE_INST/ACCUM_STAT_INST/i_9529 A  
  
#  
# ADDHIERINST: create a new module + inst  
#  
ADDHIERINST DTMF_INST/ECO_NEW_HIER_INST ECO_NEW_HIER
```

## HECO Directives

The HECO directives are used to edit a netlist hierarchically similar to what is done in case of a logic-synthesis tool, such as a RTL compiler - specify a hierarchical instance to work on and edit ports, supports, and pins (referred to as "term" below), and their connectivity.

The HECO directives must reside inside a START\_HECO/END\_HECO pair; ECO directives, such as ADDINST or ADDNET must be outside such a pair. Otherwise, ECO and HECO directives can be freely mixed in the same ECO file. HECO directives currently do not work on a hierarchical instances with a sibling instantiated from the same Verilog module. Only HECO directives are used in this section.

## HECO Syntax

CURRENT_INST hinst	Specifies the hierarchical instance to work on. Omit hinst to work on the top instance/cell
--------------------	---

CREATE_INST inst cell	Creates a leaf instance with the specified cell master.
REMOVE_INST inst	Removes net connections on the specified leaf instance and delete it
REMOVE_BUFFER inst termin termout	Removes the specified leaf instance and short the nets connected to the specified input term and output term (both must belong to the leaf instance).
CONNECT term term	Connects two terms together, merging their old nets into one. Neither term can be 1'b1/1'b0.
CONNECT_NET net term	Connects a term to a net.
CREATE_NET net	Creates the specified net.
DISCONNECT term	Makes term a singleton. The term will no longer be connected to any other term or 1'b1/1'b0.
DISCONNECT_INST inst	Disconnects all terms on the specified leaf instance.
TIE_1 term/ TIE_0 term	Connects term directly to 1'b1/1'b0.
TIE_1_NET net/ TIE_0_NET net	Makes the net a 1'b1/1'b0 net ("assign net = 1'b1/1'b0;" in the Verilog).
UN_TIE_NET net	Removes the 1'b1/1'b0 property from "net." Note that this applies to the current scope only. The (flat) DEF net that "net" belongs to may still be a 1'b1/1'b0 net because of the 1'b0/1'b1 property on another "local net" that is part of the DEF net.
port1 INPUT/OUTPUT/INOUT ... portn INPUT/OUTPUT/INOUT	Creates the specified scalar ports on the hierarchical instance  CREATE_PORT number_of_lines.
port1 ... portn	Removes the specified scalar ports on the hierarchical instance  REMOVE_PORT number_of_lines.

## Examples

```
#####
# Example Verilog
module sub (
in,
in2,
out);
input in;
input in2;
output out;

BUFX4 u1 ();
BUFX4 u2 ();
endmodule

module top (
x,
y,
z);
input x;
output y;
output z;

// Internal wires
wire net;

assign z = 1'b1 ;

BUFX4 i (.A(z));
BUFX4 i0 (.Y(net),
.A(x));
BUFX4 i1 (.A(net));
BUFX4 i2 ();
sub i3 ();
endmodule

#####
# Example ECO file. Comments start with "#"
FORMATVERSION 2

START_HECO

# create two ports for i3
CURRENT_INST i3
CREATE_PORT 2
```

```
fin INPUT
fout OUTPUT
# and add a feedthrouh
CONNECT fin fout

# switch context to the top instance
CURRENT_INST
# and make i1 drive i2 through i3's feedthrough
CONNECT i1/Y i3/fin
CONNECT i3/fout i2/A

# z is no longer a 1'b1 port but i/A is still a 1'b1 term
DISCONNECT z

# remove buffer i0 so that x drives i1/A directly
REMOVE_BUFFER i0 A Y

# connect i3's in port to 1'b1
TIE_1 i3/in

# can't use "TIE_0" on a port or subport; tie the net 1'b0 instead
TIE_0_NET y

# make i3/u1/A a 1'b1 term (through i3's in port)
CURRENT_INST i3
CONNECT in u1/A

END_HECO

#####
# Result after the above ECO file is loaded with "loadECO"
module sub (
    in,
    in2,
    out,
    fin,
    fout);
    input in;
    input in2;
    output out;
    input fin;
    output fout;

    assign fout = fin ;

    BUFX4 u1 (.A(in));
    BUFX4 u2 ();
```

```
endmodule

module top (
x,
Y,
z);
input x;
output y;
output z;

// Internal wires
wire n2;
wire n1;
wire n;

assign n2 = 1'b1 ;
assign y = 1'b0 ;

BUFX4 i (.A(n2));
BUFX4 i1 (.Y(n),
.A(x));
BUFX4 i2 (.A(n1));
sub i3 (.in(1'b1),
.fin(n),
fout(n1));
endmodule
```

Specifies the existing cell type. If the current cell is not oldCellName, the software displays an error message and the directive stops.

# Interactive ECO

- [Overview](#)
- [Before You Begin](#)
- [Results](#)
- [Adding Buffers](#)
- [Changing the Cell](#)
- [Deleting Buffers](#)
- [Displaying Buffer Trees](#)
- [Running ECO Placement](#)
- [Naming Conventions for Interactive ECO](#)
- [Comparing Physical Design Data](#)

## Overview

The Interactive ECO feature enables you to run manual incremental updates to the design to repair timing or transition time violations. You can run Interactive ECO after running placement, timing optimization, or signal integrity analysis (CeltIC NDC).

If you performed trial route and RC extraction on the design, and the timing graph was built before running an ECO, then the trial route data, RC extraction data, and timing graph are incrementally updated.

## Before You Begin

Before you can perform interactive ECO, the following conditions must be met:

- You must place and route the design,
- You must load the design into the current session.

## Results

The following output files are generated:

- Updated netlist
- Updated placement

## Adding Buffers

You can add a single buffer or a pair of inverters at a time on a net.

1. To open the Interactive ECO form, select *ECO - Interactive ECO* from the Innovus menu. This opens the Interactive ECO form. The *Add Repeater* page is selected.



2. Enter the net name in the *Net* field.

Type the net name, or click on a displayed net in the design display window and click *get selected*.

3. To select the terminals, choose one of the following:

- To connect the added buffer to drive all the receivers, specify *All Terminals*. Use this to reduce the delay and output transition time of a weak driver driving a large capacitive load.
- To connect the added buffer to drive the listed receivers, specify *Listed Terminals*. This provides full flexibility for building an arbitrary buffer connection.
- *Draw Terminal* button - Allows you to draw an area covering the terminals to which you

want to add the buffer.

4. In the *New Cell* field, enter the cell type name of the repeater to add, or click on the arrow to right of the field and select a buffer from the list.

5. In the *Place Modepane*, you can choose one among several options:

- *Default*

The software automatically determines a location and places the new cell.

- *Don't Place Cells*

Specifies that the inserted cells should not be placed. Only the logical change in connectivity will be made.

- *Location*

Enter the location for the buffer using one of the following methods:

- You can use the automatically assigned locations, enter the locations, or click on an area in the design display window and click *get coord*.

- *Relative Distance to the Sink*

Specifies the location of the buffer based on its distance from the sink or the driver pin. The value is a number between 0 and 1. A low value (0.1) places the buffer near the sink; a high value (0.9) places the buffer near the driver. The fraction is based on the length of the wire.

This option works when one term is provided; it does not work if no term or multiple terms are specified.

- *Offload*

- a. To connect the added buffer to drive only noncritical receivers, select *By Slack*. This checks the timing graph for noncritical receivers and offloads those from the critical path, and could improve critical path timing by penalizing noncritical path delays.

- b. To add a buffer at a specific location, select *By Location* and enter the x, y coordinates.

- 6. Specify a radius.

- Specify the radius in which the added instances are free to move. If no legal location can be found in the specified radius, the cells would be placed at the specified location resulting in an overlap with other cells. In that case, you should perform legalization.

- 7. (Optional) To legalize placement of the ECO changes, click *Do Refine Placement*.

- 8. Click *Apply*.

- 9. (Optional) Click the *Eval* button to evaluate the effect on timing if you add a new cell. The

values are not applied in the database.

- 10. (Optional) Click the *Eval All* button to evaluate the effect on timing for all the cell types available for the new cell. The timing report shows the effects of all the cell types, enabling you to select the best cell for your design. The values are not applied in the database.

**Note:** You can add a buffer around the I/O pin of a block using the [attachIOBuffer](#) command.

**Note:** By clicking the Mode button, you can open the Set Interactive Mode form to select different modes that control the behavior of ECO commands.

The following text command and parameters provide equivalent or additional functionality:

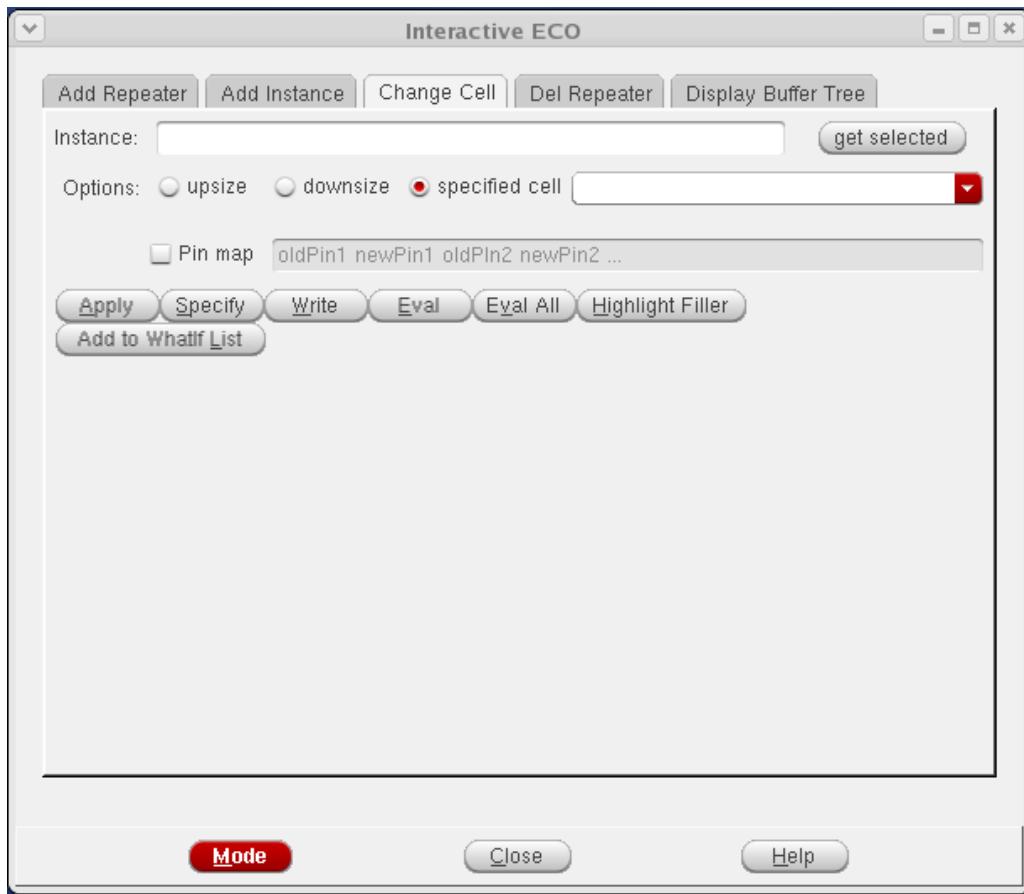
- [ecoAddRepeater](#)

For more information, see "[Interactive ECO Commands](#)" in the *Innovus Text Command Reference*.

## Changing the Cell

You can upsize or downsize instances. Upsizing an instance that drives a large load can improve the driver delay and the transition time at the receivers. You can also downsize an instance on the noncritical path to reduce the loading of its driver on the critical path.

1. To open the Interactive ECO form, select *ECO - Interactive ECO* from the Innovus menu, and click the *Change Cell* tab. The Change Cell page is displayed.



2. In the Instance field, enter the hierarchical instance name to be changed. Type the instance name, or click an instance in the design display window and click *get selected*. Select either upsize, downsize, or specified cell. If you select specified cell, enter the replacement cell name in the adjacent field.
3. Type the cell name, or use the pull-down menu to select a cell.
4. (Optional) Specify the pin mapping for the new cell based on the old cell.
5. (Optional) Click the *Eval* button to evaluate the effect on timing if you add a new cell. The values are not applied in the database.
6. (Optional) Click the *Eval All* button to evaluate the effect on timing for all the cell types available for the new cell. The timing report shows the effects of all the cell types, enabling you to select the best cell for your design. The values are not applied in the database.
7. (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.
8. Click *Apply*.

**Note:** By clicking the Mode button, you can open the Set Interactive Mode form to select different modes that control the behavior of ECO commands.

The following text command and parameters provide equivalent or additional functionality:

- [ecoChangeCell](#)

For more information, see "[Interactive ECO Commands](#)" in the *Innovus Text Command Reference*.

## Deleting Buffers

You can delete redundant buffers that cause extra delay. Buffers are typically over-added by synthesis tools based on wireload models.

1. To open the Interactive ECO form, select *ECO - Interactive ECO* from the Innovus menu, and click the *Del Repeater* tab. The *Delete Repeater* page is displayed.



2. Enter the buffer instance name to be removed in the *Instance* field. Type the instance name, or click an instance in the design display window and click *get selected*.
3. Select a deletion option: *Only This Instance* or *Whole Buffer Tree*.
4. (Optional) Click the *Eval* button to evaluate the effect on timing if you delete the cell. The values are not applied in the database.
5. (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.
6. Click *Apply*.

**Note:** By clicking the Mode button, you can open the Set Interactive Mode form to select different modes that control the behavior of ECO commands.

The following text command and parameters provide equivalent or additional functionality:

- [ecoDeleteRepeater](#)

For more information, see "[Interactive ECO Commands](#)" in the *Innovus Text Command Reference*.

## Displaying Buffer Trees

You can inspect the routing topology of the buffer tree after it is created. If the buffer tree requires correction, you can rebuild or modify it through the other three pages in the Interactive ECO form.

1. To open the Interactive ECO form, select *ECO - Interactive ECO* from the Innovus menu, and click the *Display Buffer Tree* tab. The Display Buffer Tree page is displayed.



2. To select a buffer tree, enter the net name in the *Net* field. You can type the net name, or click a net in the design display window and click *get selected*.
3. (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.
4. Click *Apply*.

**Note:** By clicking the Mode button, you can launch the Set Interactive Mode form to select different modes that control the behavior of ECO commands.

The following text command provides equivalent or additional functionality:

- [displayBufTree](#)

For more information, see "[Interactive ECO Commands](#)" in the *Innovus Text Command Reference*.

## Running ECO Placement

ECO placement updates the placement from a prior Innovus session to reflect the netlist changes, merging the new netlist changes into the prior netlist's placement. The modified netlist can then be imported into an Innovus session so that the result is a new placement that reflects the changes made in the modified netlist.

You can run either an incremental timing or logic change to the design. You can run ECO after running placement, although ECO is usually run after analyzing speed or RC data.

To update the placement with the ECO netlist, complete the following steps:

1. Save the pre-ECO netlist placement data.
2. Start a new Innovus session.
3. Import the (ECO) design.
4. Load the floorplan.
5. Run ECO Placement.

This references the pre-ECO netlist placement data. The changes reflected in the new netlist are ECO'd into the pre-ECO placement. All designs are placed in the resulting placement.

After running ECO successfully, you can run Trial Route to view the routing congestion and analyze the design for timing.

## Naming Conventions for Interactive ECO

After running interactive ECO, you can use the Design Browser to view the newly added instance names, prefixed with `FE_`. The interactive ECO operation naming conventions are described in the following table:

Name Prefix	Description
<code>FE_ECON</code>	A net added by interactive ECO
<code>FE_ECOC</code>	An instance added by interactive ECO

## Comparing Physical Design Data

After making changes to a DEF file, you can compare the new file to the information stored in the Innovus database. You can perform this comparison after you perform ECO.

Use the following command to compare physical design data:

```
defComp defFile -reportFile fileName
```

The default filename is `defPhyDiff.rpt`

The report file includes the following information:

- VERSION statement

VERSION *num*

<i>num</i>	Specifies the file version number.
------------	------------------------------------

- UNITS statement

UNITS *num*

<i>num</i>	Specifies the unit for the values such as coordinates, width, and so on.
------------	--

- ADDINST statement

ADDINST *instName* *cellName* *x* *y* *orientation*

<i>instName</i>	Specifies the instance name.
<i>cellName</i>	Specifies the master cell name of the instance.
<i>x</i> <i>y</i>	Specifies the coordinates of the instance.
<i>orientation</i>	Specifies the orientation of the instance. The orientation can be one of the following: N, FN, S, FS, W, FW, E, FE, as used by DEF file.

**Note:** The report provides the connectivity of added instances in the NETS section described below.

- DELINST statement

DELINST *instName* *cellName* *x* *y* *orientation*

The arguments have the same meaning as described in ADDINST statement.

- CHANGECELL statement

CHANGECELL *instName* *newCellName* *oldCellName*

The master cell of the instance *instName* is changed from *oldCellName* to *newCellName*. The coordinate, orientation, and connectivity of the instance are not changed. If a master cell is changed along with any of the coordinates, orientation, or connectivity of the instance, Innovus considers this change as a deletion and addition of the instance. Rather than including a CHANGECELL statement, the file contains one pair of DELINST and ADDINST statements.

- MOVEINST statement

MOVEINST *instName* [COORD *oldX* *oldY* *newX* *newY*] [ORIENT *oldOrientation* *newOrientation* ]

The statement contains the COORD phrase if the instance *instName* has moved, and the ORIENT phrase if the instance has changed orientation.

- ADDNET statement

ADDNET *netName*

<i>netName</i>	Specifies the name of a added net.
----------------	------------------------------------

- DELNET statement

DELNET *netName*

<i>netName</i>	Specifies the name of the deleted net.
----------------	--

- ADDPIN statement

ADDPIN *netName* *instName* *pinName*

<i>netName</i>	Specifies the net from which the pin is added.
<i>instName</i>	Specifies the instance name of the added pin.
<i>pinName</i>	Specifies the name of the added pin.

- DELPIN statement

DELPIN *netName instName pinName*

<i>netName</i>	Specifies the net from which the pin is deleted.
<i>instName</i>	Specifies the instance name of the deleted pin.
<i>pinName</i>	Specifies the name of the deleted pin.

- CHANGEROUTE and ENDCHANGEROUTE statements

CHANGEROUTE *netName*

ENDCHANGEROUTE

These statements mark the beginning and end of the CHANGEROUTE section that contains changes on the routing segment on net *netName*. The wire change statements are included between the CHANGEROUTE and ENDCHANGEROUTE statements.

- POWERROUTE and ENDPOWERROUTE statements

POWERROUTE *netName*

ENDPOWERROUTE

These two statements mark the beginning and the end of the POWERROUTE section that contains the power routing differences between two DEF files. The wire change statements are included between the POWERROUTE and ENDPOWERROUTE statements.

- Wire change statements

The ADDWIRE, DELWIRE, ADDVIA, and DELVIA statements appear between the CHANGEROUTE and ENDCHANGEROUTE statements, or POWERROUTE and ENDPOWERROUTE statements.

- ADDWIRE statement

ADDWIRE *layerName width x1 y1 x2 y2*

<i>layerName</i>	Specifies the layer name of wire segment added to the net.
<i>width</i>	Specifies the width of the wire segment added to the net.

<i>x1 y1</i>	Specifies the left or bottom coordinate of wire segment added to the net.
<i>x2 y2</i>	Specifies the right or top coordinate of wire segment added to the net.

- **DELWIRE statement**

**DELWIRE** *layerName width x1 y1 x2 y2*

<i>layerName</i>	Specifies the layer name of wire segment deleted to the net.
<i>width</i>	Specifies the width of the wire segment deleted to the net.
<i>x1 y1</i>	Specifies the right or top coordinate of wire segment deleted from the net.
<i>x2 y2</i>	Specifies the left or bottom coordinate of wire segment deleted from the net.

- **ADDVIA statement**

**ADDVIA** [ *botLayerName bx1 by1 bx2 by2*] *topLayerName tx1 ty1 tx2 ty2*

<i>botLayerName</i>	Specifies the bottom layer name of the via added to the net.
<i>bx1 by1</i>	Specifies the lower-left coordinate of bottom layer of the via added to the net.
<i>bx2 by2</i>	Specifies the top-right coordinate of bottom layer of the via added to the net.
<i>topLayerName</i>	Specifies the top layer name of via added to the net.
<i>tx1 ty1</i>	Specifies the lower-left coordinate of top layer of the via added to the net.
<i>tx2 ty2</i>	Specifies the top-right coordinate of top layer of the via added to the net.

**Note:** For turnvias, Innovus reports *topLayerName* only.

- **DELVIA statement**

**DELVIA** [ *botLayerName bx1 by1 bx2 by2*] *topLayerName tx1 ty1 tx2 ty2*

<i>botLayerName</i>	Specifies the bottom layer name of via deleted from the net.
<i>bx1 by1</i>	Specifies the lower-left coordinate of bottom layer of the via deleted from the net.
<i>bx2 by2</i>	Specifies the top-right coordinate of bottom layer of the via deleted from the net.

<i>topLayerName</i>	Specifies the top layer name of via deleted from the net.
<i>tx1 ty1</i>	Specifies the lower-left coordinate of top layer of the via deleted from the net.
<i>tx2 ty2</i>	Specifies the top-right coordinate of top layer of the via deleted from the net.

**Note:** For turnvias, Innovus reports *topLayerName* only.

- ADDOBS statement

ADDOBS *layerName x1 y1 x2 y2*

<i>layerName</i>	Specifies the layer name for the obstruction added.
<i>x1 y1</i>	Specifies the lower-left coordinate of the obstruction.
<i>x2 y2</i>	Specifies the top-right coordinate of the obstruction.

- DELOBS statement

DELOBS *layerName x1 y1 x2 y2*

<i>layerName</i>	Specifies the layer name of the deleted obstruction.
<i>x1 y1</i>	Specifies the lower-left coordinates of the obstruction.
<i>x2 y2</i>	Specifies the upper-right coordinates of the obstruction.

# Editing Wires

- [Overview](#)
- [Before You Begin](#)
- [Using Keyboard Shortcuts](#)
- [Selecting Wires](#)
- [Deleting Wires](#)
- [Moving Wires](#)
- [Copying Wires](#)
- [Adding Wires](#)
- [Cutting Wires](#)
- [Trimming Antennas on Selected Stripes](#)
- [Changing Special Wire Width](#)
- [Repairing Maximum Wire Width Violations](#)
- [Duplicating Special Wires](#)
- [Stretching Wires](#)
- [Changing Wire Layers](#)
- [Splitting and Merging Special Wires](#)
- [Adding Vias](#)
- [Changing Vias](#)
- [Moving Vias](#)
- [Reshaping Routes](#)
- [Controlling Cell Blockage Visibility](#)

## Overview

You can edit the wires and vias in your design manually by using the Edit Route form, the wire editing commands, and a combination of keyboard shortcuts (bindkeys) and tool widgets.

For signal wires, you can perform the following actions:

- Add wires
- Cut wires
- Move wires
- Change the wire to another layer
- Change wire width
- Change vias

- Delete wires
- Merge selected wires
- Force wires to use specified widths
- Add vias
- Trim selected wires
- Copy/paste wires

For power wires, you can perform all of the actions available for signal wires, as well as the following additional actions:

- Split selected wires
- Duplicate selected wires
- Fix wires wider than the maximum width
- Force wires associated with special nets to be created as signal wires

For description of tabs and fields on the Edit Route form, see the [Edit Menu](#) chapter in the *Innovus Menu Reference*.

## Before You Begin

Before you can use the wire editing features, load the design into the current Innovus session.

Before running Wire Editor commands on a Double Patterning Technology (DPT) design, you must run the `colorizeGeometry` command to mark the colors on all existing objects and wire tracks.

In releases prior to 14.1, the Wire Editor color scheme always followed the track color. However, with recent advancements in the DPT and Multi-Mask Patterning (MMP) process, most library cells, pins, and pre-route are pre-colored and do not follow the track color due to color conflict. From the 14.1 release, Wire Editor has been enhanced to comply with the latest MMP spec. It dynamically colorizes the wire segment and via being edited by automatically assigning and flipping the color on the active wire segment to avoid color conflict with neighboring objects on the same DPT layer. If multiple color conflicts cannot be resolved by flipping the *maskNum/color*, a color violation marker is flagged immediately. You then have a choice to override the default mask color as needed.

You can use the `setEdit -assign_multi_pattern_color {Auto | Mask1 | Mask2 | Mask3}` parameter to specify how to assign the mask color on the DPT layer of the wire segment to be created. The default value is `Auto`. Alternatively, you can use the *Assign Multi-Pattern Color* drop-down in the *Advanced* tab of the Edit Route form to assign a mask color manually.

**Note:** After you use the wire editing features, the Innovus software saves the new and modified wires and vias in the database.

## Using Keyboard Shortcuts

The Innovus software includes keyboard shortcuts for use with the wire editing features. Type the keyboard shortcuts while the main Innovus window is active and the cursor is in the design display area. Some of the keyboard shortcuts provide functionality that is not available through the Edit Route form or the wire editing commands.

### Keyboard Shortcuts That Open Forms

Click in the design display area, then use one of the following shortcuts:

D	Opens or closes the Select/Delete/Deselect Route form
E	Opens or closes the Edit Route form

### Keyboard Shortcuts That Are Equivalent to Tool Widgets

A		Select
Shift+A		Edit Wire
M		Move Wire
O		Add Via
Shift+R		Move/Resize/Reshape (non-connectivity-based move/resize/stretch)
S		Stretch Wire
Shift+X		Cut Wire

For more information, see "[Tool Widgets](#)" in the *Innovus Menu Reference*.

## Keyboard Shortcuts Used in Auto Query Mode

N	Toggles to next object under cursor.
P	Toggles to previous object under cursor.
Shift+S	<p>Populates the Edit Route form with net name, width, layers, and shape of highlighted (queried) wire or pin. The <i>Nets</i> field of the Select/Delete Routes form is also populated.</p> <p>If the queried object is a pin, the layer and width information is set for both horizontal and vertical directions. If the queried object is a wire, the width information is set for both horizontal and vertical directions, but only one of the layers is set. That is, only the horizontal layer is set for a horizontal wire and only the vertical layer is set for vertical wires. This keyboard shortcut does not populate the form with spacing information.</p>
Ctrl+W	Deletes the queried segment or via.

These keyboard shortcuts work only while you are in *auto query* mode--they do not work while you are drawing a wire. For more information, see "[Auto Query](#)" in the *Innovus Menu Reference*.

## Keyboard Shortcuts Used in Edit Wire Mode

D	Changes the added wire to the layer below the current layer.
U	Changes the added wire to the layer above the current layer.
Backspace	Deletes the last segment created in the design area. This allows you to remove one segment of the route at a time.
Esc	Removes the entire route.
Number keys	Change the added wire to a specific layer number. If you want the wire to be added to metal layer 1, use the <sup>1</sup> keyboard shortcut, use the <sup>2</sup> keyboard shortcut for metal layer 2, and so forth.
Single-click	Ends the segment, allowing you to continue the route in either the same direction or the orthogonal direction.
Double-click	Ends the route.

## Keyboard Shortcuts Used in Stretch Wire Mode

<sup>1</sup>	Stretches or reduces horizontal wires from the left and vertical wires from the bottom, using the Shift key and the arrow keys.
<sup>2</sup>	Stretches or reduces horizontal wires from the right and vertical wires from the top, using the Shift key and the arrow keys.

For more information, see [Stretching Wires](#).

## Keyboard Shortcuts Used to Change Vias

Shift+N	Changes the selected via to the next available via.
Shift+P	Changes the selected via to the previous available via.

For more information, see [Changing Vias](#).

## Selecting Wires

1. Click the *Move Wires* widget  in the Tool Widgets area of the Innovus main window or press the `M` keyboard shortcut while the cursor is in the design display area.
2. Click a wire.

Tip

If multiple objects exist at the location of the cursor, press the space bar to toggle the selection among them. To select multiple objects, press the `Shift` key while clicking.

## Deleting Wires

To delete a wire without deleting the vias connected to it, complete the following steps:

1. Turn on *Auto Query*. 
2. Move the cursor over the wire to delete.
3. Use the `N` (next) or `P` (previous) keyboard shortcut to select the correct wire.
4. Press `Ctrl+W` or the Delete key.

**Note:** To delete a wire and the vias connected to it, use the `editDelete` command. To delete a wire without deleting the vias connected to it, you can use the `editDelete` command with the `-wires_only` option.

During post-mask ECO, you can freeze wires and vias by changing their status to `COVER`. The

Innovus software does not delete wires or vias with status COVER. Type the following commands to freeze the wires and vias on metal layers 1 and 2:

```
deselectAll  
editSelect -layer { M1 M2}  
editSelectVia -cut_layers V12  
editChangeStatus -to COVER
```

## Moving Wires

You can move wires in the orthogonal direction by using the `editMove` command, mouse, or the keyboard arrow keys (in conjunction with the `Shift` key).

### Using the Mouse to Move Wires

1. Click the *Move Wires* icon  in the Tool Widgets area of the Innovus main window.  
The equivalent keyboard shortcut is `M`.
  2. Click the wire to be moved.  
The selected wire is highlighted.
  3. Move the cursor slightly within the selected wire.  
The cursor changes to a circle shape.
  4. Click and release the mouse.  
The wire moves with the cursor in the orthogonal direction (up or down for a horizontal wire, left or right for a vertical wire). Wires connected to the moved wires stretch to maintain connectivity.
  5. Click the mouse again to place the wire in the new location.
- Note:** To cancel the move before you click the mouse, press the `Esc` key. The wire returns to its original location.

**Note:** If you select the *Snap to Track* option, the wire automatically snaps to the appropriate routing track.

## Using Arrow Keys to Move Wires

1. Choose *Edit - Wire - Edit* from the menu.

The Edit Route form opens.

The equivalent keyboard shortcut is **E**.

2. Click the *Misc* tab.

3. Specify a value, in microns, in the *Arrow Increment* field.

This value defines the distance that the wire is to move each time you press an arrow key while holding the **Shift** key. You can specify either a positive or negative number.

4. Click the *Move Wire* icon  in the Tool Widgets area of the Innovus main window.

The equivalent keyboard shortcut is **M**.

5. Click the wire to be moved.

The selected wire is highlighted.

6. Hold the **Shift** key, then press the up or down arrow key for a horizontal wire or the left or right key for a vertical wire.

The selected wire moves in the direction of the arrow.

## Copying Wires

You can copy wires and vias by using the following methods:

- Using the **editCopy** command (all wires and vias)
- Using the mouse (all wires and vias)
- Duplicating and moving (only special wires and vias)

## Using the **editCopy** Command

To copy a wire or via to a specific location, use the **editCopy** command. You can either pre-select the wire object to be copied or specify the object pointer with **editCopy**. You can also specify whether or not the copied object should retain the net name of the original object.

## Using the Mouse to Copy Wires or Vias

To copy wires or vias using the mouse, complete the following steps:

1. Select the wire or via to be copied.
2. Use the `C` bindkey or click the *Copy* icon () to switch to *Copy* mode.
3. (Optional) Press `F3` to open the Copy form. Select the direction in which you want to move the copied object from the *Move Direction* drop-down. You also choose to retain net name of the original object in the copied object.
4. Move the mouse over any of the selected objects. The cursor changes to a black dot.
5. Click once to start copying the selected object. A ghost image of the original object will move along with the cursor.
6. Click again to place the copy at the desired location.

## Copying & Moving Special Wires or Vias

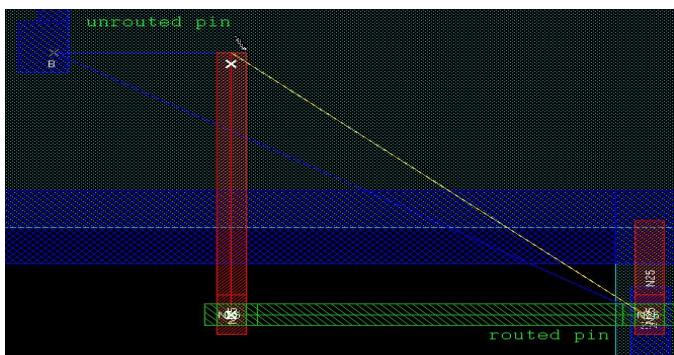
To copy/paste and then move selected special wires or vias, complete the following steps:

1. Select wires or vias.
2. Type the `editDuplicate` command (or use the `C` keyboard shortcut) to copy the objects. The duplicate object is created directly on top of the original object.
3. Use the `Shift+R` keyboard shortcut or click the *Move/Resize/Reshape* icon () to switch to *non-connectivity-based move* mode.
4. Move the mouse over any of the selected objects. A black dot appears.
5. Click once to start moving the selected objects.
6. Click again to place the objects in the desired location.

**Note:** To cut & paste, and move selected special wires or vias, skip step 2.

## Adding Wires

You can add one or more wires interactively to single or multiple nets. When you add wires, the flight lines to routed pins are displayed in the pin color (by default, yellow) and flight lines to unrouted pins are displayed in the wire color (by default, blue).



By default, the routing status for newly added signal wires is **FIXED**. A **FIXED** routing status means that the automated routers do not rip up and reroute preroutes. Signal wires that are moved, cut, or otherwise changed by wire editing commands maintain the routing status that was set before the wire editing commands were issued.

## Adding a Wire for a Single Net

1. Click the *Edit Wire* widget  (or press **Shift+A**).

This places the Innovus software in the *Edit Wire* mode and the mouse cursor changes to a pencil . In addition, Innovus is automatically placed in the *Auto Query* mode, even if the *Q* widget below the design display area is not enabled.

2. If pins are not visible, select *Pin Shapes*.  in the *Cell* group on the Layer Control bar.
3. Place the cursor over the pin or wire at the starting point for the wire to be drawn, and then type **Shift+S** while the cursor is in the design display window.

This populates the *Edit Route* form with the net name, layer, and width information that is used in creating new wires.

**Note:** If multiple objects exist at a particular point, use the **N** or **P** keyboard shortcut to cycle through the objects.

4. (Optional) Choose *Edit - Wire - Edit* from the menu or use the **E** keyboard shortcut.  
The *Edit Route* form opens, and has been automatically populated with the net name, layers, and widths. The form is not populated with spacing information, which only applies while editing multiple nets.
5. Click the *Basic* tab on the *Edit Route* form and adjust the values in the *Layer* and *Width* fields.
6. (For special wires only) Select a shape from the *Shape* drop-down menu on the *Basic* tab.

**Note:** Shapes are only defined for power/special wires. This value is ignored for signal wires.

7. Click the start point for the wire you want to add, then move the mouse to a new point.  
The wire is drawn interactively as you move the mouse.
8. Click a new location to change the direction of the wire or continue in the same direction with a different segment.

**Note:** If there is a layer change, a via is automatically created.

#### Tips

- Press a number key to change the layer of the wire being added.  
When the software is in the *Edit Wire* mode, number keys can be used as keyboard shortcuts, with the number indicating the layer number of the wire being drawn. For example, if you press the number 2, the segment is added to metal layer 2. Alternatively, you can use the `U` or `D` keyboard shortcuts to change the layer of the segment. The `U` keyboard shortcut changes the segment to the next higher layer, and the `D` keyboard shortcut changes the segment to the next lower layer.
- Press the `Backspace` key to erase the most recently drawn segment.  
You can do this for as many segments as needed.

9. Double-click the mouse.

The wire ends at the location of the cursor.

**Note:** After double-clicking, you cannot use the `Backspace` key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the *Edit Delete* form.

**Note:** Widths for signal wires depend on the applicable LEF rule, no matter what value is populated in the GUI. To specify a wire width that is different from the default wire width value, select a non-default rule other than *Default* from the *Rule* drop-down menu on the *Basic* tab of the *Edit Route* form.

## Adding Parallel Wires for Multiple Nets

To add parallel wires for multiple nets at the same time, complete the following steps:

1. Click the *Edit Wire* widget (or press Shift+A).

This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil.

2. Choose *Edit - Wire - Edit* from the menu or use keyboard shortcut E.

The Edit Route form opens.

3. Select the appropriate *Route Action* - *Create Regular Wire* or *Create Special Wire*.

4. Click the *Basic* tab on the Edit Route form and enter the net names in the *Nets* field.

**Note:** You can also specify a file that contains a list of nets. See [Adding Wires that Automatically Extend to a Target](#) for more information.

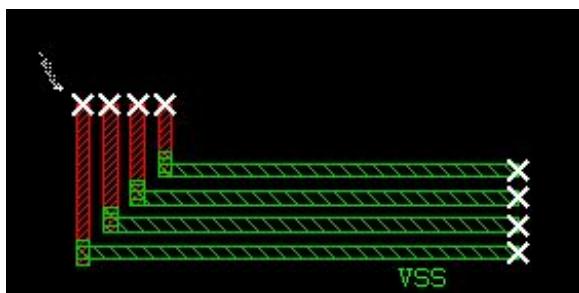
5. (Optional) Select horizontal and vertical layer names and specify width and spacing values.

**Note:** To use different width or spacing values for different nets, use the *Multi-Net* tab. See [Using Override to Add Wire Groups with Multiple Widths and Spacing](#) for more information.

6. (Optional) Specify a value in the *Drawing Wire* field on the *Multi-Net* tab.

This specifies which of the nets (specified in the *Nets* field) corresponds to the mouse pointer location. By default, this value is 1, meaning the mouse position corresponds to the position of the left-most or bottom-most net of the group.

For example, if the *Nets* field contains vss VDD VDDA VSSA, the vss net is the bottom-most net for horizontal segments, and the left-most net for vertical segments. If the value in the *Drawing Wire* field is set to 1, the mouse location corresponds to wires on the VSS net.



7. (For special wires only) Click the *Basic* tab and select a shape from the drop-down menu.

**Note:** Shapes are only defined for power wires. This value is ignored for signal wires.

8. Click the start point for the wires you want to add, then move the mouse to a new point.

The wires are drawn interactively as you move the mouse.

9. (Optional) Click a new location to change the direction of the wires or to continue in the same direction with a different segment.

**Note:** If there is a layer change, a via is automatically created.

Tip

Press the `Backspace` key to erase the most recently drawn set of segments. You can do this for as many sets of segments as needed.

10. Double-click the mouse.

The wires end at the location of the cursor.

**Note:** After double-clicking, you cannot use the `Backspace` key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the *Edit Delete* form.

## Adding Wires that Automatically Extend to a Target

To create a wire group for multiple nets that automatically extend to targets, complete the following steps:

1. Click the *Edit Wire* widget .

This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil.

2. Choose *Edit - Wire - Edit Route* from the menu.

The *Edit Route* form opens.

3. Select the appropriate *Route Action* - *Create Regular Wire* or *Create Special Wire*.

4. Create a text file that contains the names of multiple nets.

Make sure that each line in the file contains the name of one net, and that the nets are listed in the order in which you want to create the wire group.

5. Click the *Load* button on the *Basic* tab of the *Edit Route* form.

This opens the *Open* form.

6. Select the file you created in step 4, then click *OK*.

The *Nets* field now contains the net names in the file.

7. On the *Basic* tab, select horizontal and vertical layer names and specify width and spacing values.

**Note:** To use different widths or spacing values for different nets, use the *Multi-Net* tab. See [Using Override to Add Wire Groups with Multiple Widths and Spacing](#) for more information.

8. Click the *Misc* tab, and select the *Extend Start Wires* and *Extend End Wires* options. These options extend both ends of the wires until they connect to a target.
9. Click the point in the design display area where the left-most or bottom-most wire should start.  
**Note:** The start point does not have to be at a target.
10. Move the mouse horizontally or vertically.  
The wires are drawn interactively.
11. Double-click the mouse.  
The start point and end point of the wire extend until they connected to a target. If no target is present, the wire does not extend.

## Using Override to Add Wire Groups with Multiple Widths and Spacing

To add pairs of power and ground wires, where wires have different widths and spacing, complete the following steps:

1. Click the *Edit Wire* widget .

This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil.

2. Choose *Edit - Wire - Edit* from the menu. The *Edit Route* form opens.
3. Click the *Create Special Wire* option button.
4. Click the *Basic* tab on the *Edit Route* form and enter the net names into the *Nets* field. For example, "vdd vss vdd" for illustration purpose.

**Note:** You can also specify a file that contains a list of nets. See [Adding Wires that Automatically Extend to a Target](#) for more information.

5. On the *Basic* tab, select the horizontal and vertical layer names and specify the width and spacing values. These settings will be used for non-overridden nets.
6. Click the *Multi-Net* tab and then enter a set of width and spacing values for the nets that do not have default width and spacing values.

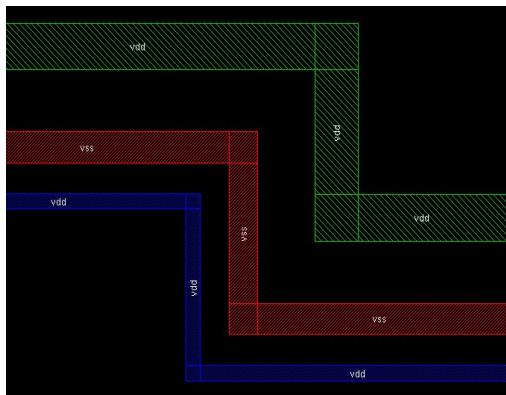
For example, if you specify the following values:

```
1 0.1 0.2 1
2 0.2 0.4 2
3 0.3 0.6 3
```

The first line indicates that the first net (`vdd`) has a width of 0.1 microns; the spacing value between the first and second net is 0.2 microns; and the first net is routed on M1 only. The

second line indicates that the second net (`vss`) has a width of `0.2` microns; the spacing value between the second and third net is `0.4` microns; and the second net is routed on M2 layer only. To specify the same constraints for multiple nets, use `,` to separate the net numbers. For example, if you specify `"netNum1,netNum2,netNum3 width spacing layer"`, all three nets will have the same width, spacing, and layer number.

**Note:** To specify a value of less than `1`, you must include a `0` before the decimal point. For example, a value of `.6` is not valid, and must be expressed as `0.6`.



7. Close the Edit Route form.
8. Click the point in the design display area where the leftmost stripe should start.  
**Note:** The start point does not have to be at a target.
9. Double-click the mouse.

The wires end at the location of the cursor.

## Cutting Wires

1. Click the *Cut Wire* widget .

The cursor changes to the shape of a scissors, indicating that the Innovus software is in the *Cut Wire* mode.

2. Click the location at which you want to start cutting the shields.
3. Move the mouse so that the drawn line is touching or overlaps the wire orthogonally.
4. Click to complete the cut.
5. Use the `A` keyboard shortcut to enter the *Select* mode.

The cursor changes to an arrow shape.

6. Click the piece of wire to be deleted.  
The selected piece of wire is highlighted.

**Tip**

If multiple objects exist at the location of the cursor, press the space bar to toggle the selection among them. To select multiple objects, press the `Shift` key while clicking.

7. Use the `D` keyboard shortcut to delete the selected objects.

**Note:** Wires can only be cut in the orthogonal direction. If you cut multiple wires, including wires in the same direction as the cut, the cut only affects wires in the orthogonal direction to the cut. Once cut, signal wire pieces maintain a 1/2 wire width extension, but power wires are not extended.

## Trimming Antennas on Selected Stripes

If your completed power structure contains stripes in a mesh configuration, physical antennas might remain on some stripes.

1. Use the `D` keyboard shortcut to display the Select/Delete Routes form.
2. Choose *Select* from the *Action* drop-down menu.
3. (Optional) Click *Nets*, then specify one or more nets for the wires to be trimmed.
4. (Optional) Click *Direction*, then click `H` to trim horizontal wires or `V` to trim vertical wires.
5. Click *Shapes*, then select *STRIPE*.
6. Click *Apply*.

The selected wires are highlighted in the design display area.

7. Use the `Shift+T` keyboard shortcut or click the  widget at bottom of the Edit Route form to

trim the selected wires.

The selected power wires are trimmed back to the last connection point and deselected.

## Changing Special Wire Width

After running power analysis, you might need to increase the width of some power stripes to alleviate any IR drop or EM issues.

1. Make sure the software is in *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to be widened.
2. Use the `E` keyboard shortcut.

This displays the Edit Route form without placing the software in the *Edit Wire* mode.

3. Click the *Create Special Wire* option.
4. Click the *Basic* tab on the Edit Route form and enter values in the *Width* fields.  
Specify a width value in the *Horizontal* section for horizontal wires and a width value in the *Vertical* section for vertical wires.

5. Use the `Shift+W` keyboard shortcut or click the  widget at bottom of the Edit Route form.

This changes the width of the selected wire. Any via connected to that the wire is also updated based upon the new width.

## Repairing Maximum Wire Width Violations

Violations occur if you specify wires widths greater than the maximum width defined by the `maxWidth` rule in the LEF file.

1. Use the `E` keyboard shortcut.

This displays the Edit Route form without placing the software in the *Edit Wire* mode.

2. Click the *Fix Wires Wider than Max Width* widget  at the bottom of the Edit Route form.  
This executes the `editFixWideWires` command, which finds any wires violating the `maxWidth` rule and splits up both the wires and the associated vias as minimally as possible while maintaining the same footprint.

## Duplicating Special Wires

After running power analysis, you might need to add some power stripes to alleviate any IR drop or EM issues. Instead of creating new wires interactively, you can duplicate existing special wires.

1. Make sure the software is in the *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to duplicate.
2. Click the *Duplicate Selected Wires* widget  or use the `c` keyboard shortcut.

The duplicated wire is automatically selected and placed directly on top of the original wire.

**Note:** The width and layer of the duplicated wire are always the same as the original wire. To duplicate a wire and change the layer, use the `editDuplicate` command and specify the layer for the duplicate wire. For more information, see `editDuplicate` in the "Wire Edit Commands" chapter of the *Innovus Text Command Reference*.

3. Use the `M` keyboard shortcut.

This places the software in the *Move* mode, allowing you to use the mouse or the arrow keys (while holding down the `Shift` key) to move the newly created wire segment to the desired location.

## Stretching Wires

1. Click the *Select* widget  in the Tool Widgets area of the Innovus main window.  
The cursor shape is an arrow, indicating that Innovus software is in the *Select* mode. The equivalent keyboard shortcut is `A`.
2. Click the wire to stretch.  
The selected wire is highlighted.
3. Click the *Stretch Wire* widget  in the Tool Widgets area of the Innovus main window.  
The equivalent keyboard shortcut is `S`.
4. Move the cursor to the end point of the wire to be stretched. The cursor changes to a T shape .
5. Click the end point, then release the mouse button and move the cursor to a new location and click again. The wire stretches to the new location.

Alternatively, you can use the `Shift` key in conjunction with the arrow keys to stretch or shrink the wire. When the software is in the *Stretch Wire* mode, you can use `1` and `2` as keyboard

shortcuts to set the edge of the wire to be stretched. By default, the wire is stretched from the top or the right using the arrow keys regardless of the cursor location. To stretch the wire from the bottom or the left, use the **1** keyboard shortcut. The *Stretch Wire* widget reverses  so that the outer arrow points to the left. To return to stretching wires from the top or right, use the **2** keyboard shortcut. The *Stretch Wire* wedge change back to the original picture and the software is in the default stretch mode.

From the 14.1 release, the Floorplan Move/Size/Reshape command (`Shift + R` bindkey) can be used to resize and stretch wires, in addition to moving, without checking the DRC. Use the `editStretch -no_conn` parameter to specify whether the tool should honor wire connectivity and drop vias between wires in different layers. If you set `-no_conn` to:

- **1:** No via will be created or removed.
- **0:** Tool will create or remove via, if necessary.

Currently, resizing or stretching using `Shift+R` is supported only for horizontal and vertical wires and not for 45-degree wires. Only special wires support resizing.

## Changing Wire Layers

You may need to change sections of wires to different layers in order to relieve congestion on a specific layer or to fix process antenna violations.

1. Make sure the software is in the *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to be updated.
2. Use the `E` keyboard shortcut.

This displays the Edit Route form without placing the software in the *Edit Wire* mode.

3. Click the *Basic* tab on the Edit Route form and enter values in the *Layer* fields.  
Specify a layer value in the *Horizontal* section for horizontal wires and a layer value in the *Vertical* section for vertical wires.

4. Use the `Shift+L` keyboard shortcut or click the  widget at bottom of Edit Route form.

This changes the layer of the selected wire. Any via connected to that wire is also updated automatically based on the new layer.

## Splitting and Merging Special Wires

Stripes that spread over the entire die may need to be altered only in specific locations. In this case, a stripe that is represented as a single piece of wire segment must be split into multiple segments before any local editing. You can split a single stripe into multiple cut stripes at each crossover automatically:

1. Make sure the software is in the *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to be split.
2. Use the `Ctrl+S` keyboard shortcut.

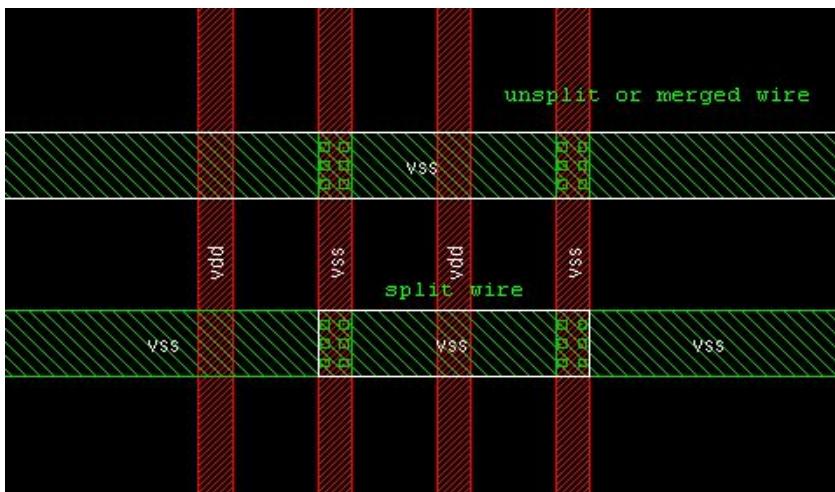
This automatically splits the single wire segment into multiple segments at points connected to other orthogonal wires.

After splitting a wire, you can merge those wire segments that align back into a single segment.

1. Select a single segment.
2. Use the `Shift+M` keyboard shortcut.

This merges the wire segments into a single segment.

In the following picture, the second horizontal stripe (vss) is split into 3 wire segments due to two crossover points.



## Adding Vias

1. Select the *Add Via* widget . The equivalent keyboard shortcut is **O**.
2. Press the **F3** key.  
This displays the Edit Via form.
3. Select *Geometry* in the *Create Via by* field.
4. Fill all of the fields in the form. For more information, see "Edit Via" in [Edit Menu](#) chapter of the *Innovus Menu Reference*.
5. Move the cursor to the location to which the via is to be added, then click the mouse.

A via with the exact configuration specified in the Edit Via form is added at that location.

## Changing Vias

- Using the `editChangeVia` command

You can change one or more vias using this command. For example, to change all `VIA_XX` vias of a specified net located within a specified region to `VIA YY` vias, type the following command:

```
editChangeVia -net netName -area { x1 y1 x2 y2 } -from VIA_XX -to VIA YY
```

For more information, see [editChangeVia](#) in the *Innovus Text Command Reference*.

- Using keyboard shortcuts

You can change one via at a time using keyboard shortcuts.

- a. Place the cursor on the via to be changed in *Auto Query* mode.
- b. Use the **N** (next) or **P** (previous) keyboard shortcuts to select the correct via if multiple vias exist in the same location on different layers.
- c. Without moving the mouse, use the **Shift+N** (next) or **Shift+P**(previous) keyboard shortcut to display a via that has the same LEF rule as the selected via.
  - If a via is available, the display is updated with the new via when you press the keyboard shortcut.
  - If another via is not available, you will hear a warning beep when you press the

keyboard shortcut. This can occur when only one via is defined in the LEF file, when the currently queried object is not a via, or when no object is currently queried.

**Note:** By default, if the net is routed with a Non Default Rule (NDR), the Wire Editor circles through the list of NDR vias defined in the LEF for that NDR rule *plus* the default vias, each time the *Shift + N* bindkey is used. If you want the Wire Editor to consider only the NDR vias when *Shift + N* is pressed, set the new `setEdit - circle_NDR_vias_only` parameter to 1. This restricts circle list to NDR vias only.

- Using the Edit Power Vias form

For information, see [Edit Power Vias](#) in the *Innovus Menu Reference*.

**Note:** You cannot change vias using the Edit Route form.

## Moving Vias

Select vias. Use the Shift+R keyboard shortcut or click the *Move/Resize/Reshape* icon  to switch to *non-connectivity-based move* mode then move the vias. The Innovus software moves vias without considering connectivity.

## Reshaping Routes

You can reshape routes by specifying that wires at the corner of a route are to be trimmed after adding wires within an area that makes the existing corner wires obsolete. In addition, if you add a wire that circumvents an existing path, the existing route is trimmed after the new wires are added.

1. Click the *Edit Wire* widget .

This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil. In addition, it places the software in the *Auto Query* mode.

The equivalent keyboard shortcut is *Shift+A*.

2. Choose *Edit - Wire - Edit* from the menu. The *Edit Route* form opens.

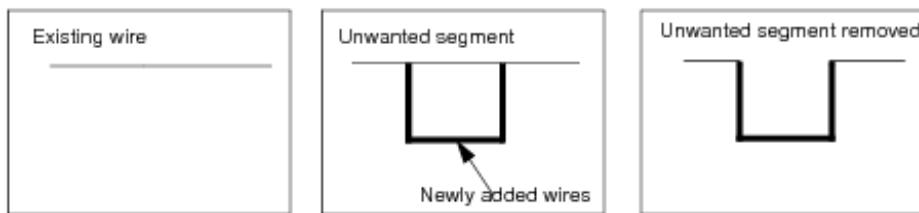
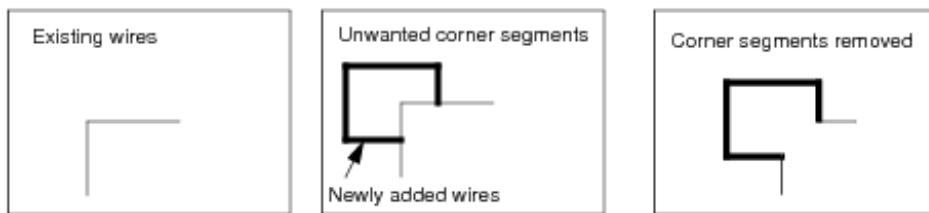
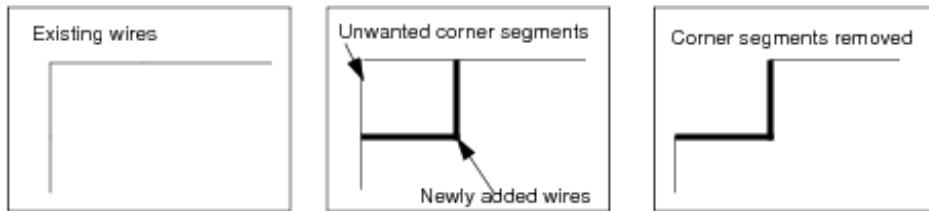
The equivalent keyboard shortcut is *E*.

3. Click the *Misc* tab.

4. Select the *Reshape* option on the form.

5. Add wires to the design, as described in [Adding Wires](#).

The following illustrations show the results of using the *Reshape* option:



## Controlling Cell Blockage Visibility

If you see a spacing violation when adding or editing a via or wire, it might be caused by a cell blockage that is not currently visible.

To see cell blockages, select the *Cell Blkg* option on the *Routing color control* display (click the sidebar to display this option). Alternatively, you can click the *All Colors* button to display the Color Preferences form, then select the *Cell Blockage* visibility option. In addition, depending on whether the blockage is outside or inside a cell, you must do one of the following:

- Cell blockages outside a cell are visible by default. To control the visibility of these blockages for particular layers, click the *Wire Layers* tab of the Color Preferences form. Use the buttons in the fifth column, *Blkg*, to deselect the visibility of blockages for particular layers. By default, all layers are selected.
- Cell blockages within a cell are not visible by default. To see these cell blockages, click the *Wire Layers* tab of the Color Preferences form, then use the buttons in the sixth column, *V*

*B/kg*, to select the visibility of blockages for each cut layer that you want to see. By default, all layers are deselected.

---

# Design Methodology for 3D IC with Through Silicon Via

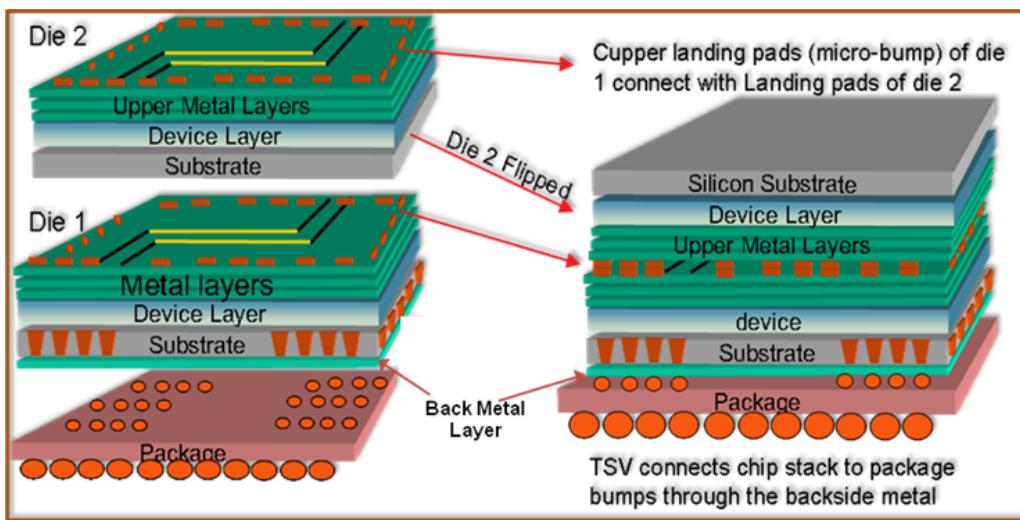
---

- Overview
- TSV/Bump/Back Side Metal Modeling in Innovus
- Defining Keep Out Area in Hard Macros
- 3D IC Flow in Innovus
- Design Import
- Stacked IC Verilog Input
- Stack Configuration Input
- Power Connectivity Input
- Interface Synchronization and Information Exchange between Dies
- TSV and Bump Manipulation
- TSV and Bump Routing
- Cross Die Connectivity Verification

## Overview

A 3D IC system usually contains several dies connected in three dimensions. In conventional IC, the IO pins are implemented by either bumps or bonding pads on one side of the chip. To enable the 3D interconnection, several additional components are created on the chip. Firstly, several Re Distributed Layers (RDL) are formed on the back side of the chip. Therefore, bumps can be placed on both front side and back side. Secondly, the Through Silicon Via (TSV) is dropped on the silicon substrate between the first metal and the back side RDL. Finally, when the dies are stacked, the aligned bumps between them constitute the data path from one die to the other.

**Figure 1 Scheme of 3D IC Profile**



Innovus supports TSV designs. In Innovus, all the dies of the 3D ICs are divided into several tiers. Each tier contains several dies, and each die can be flipped, rotated and with offset related to the package. With Innovus, the designers are able to specify the multi-die system configuration (the interconnection between dies, the related position of each die), manipulate TSVs and bumps, perform co-design, and sync-up the interface among all the dies.

### Related Topics

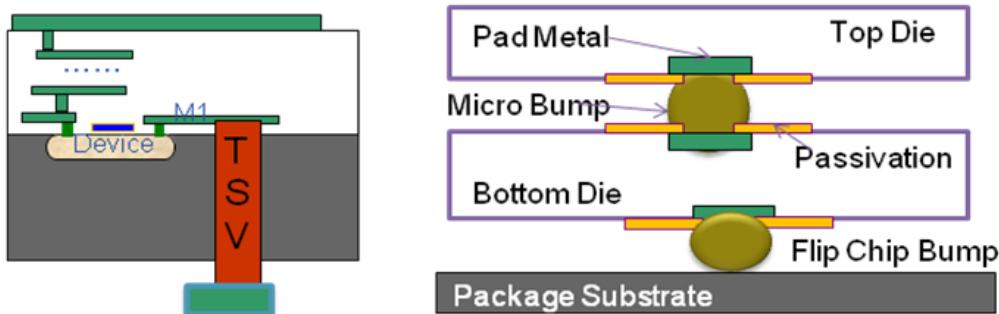
- "Through Silicon Via Design Commands" chapter of the *Innovus Text Command Reference*.
- "TSV Tool Box" section of the Tools Menu chapter in the *Innovus Menu Reference*.

## TSV/Bump/Back Side Metal Modeling in Innovus

Back Side Metal (MB), TSV, and Micro bump are introduced to establish 3D stacked interconnection between dies. MB is the Redistribution Layer on the back side of the substrate. TSV is the via which penetrates the silicon substrate. The top cap layer of TSV is the first normal routing layer (M1) and the bottom cap layer of TSV is MB, therefore, the back side metal and the 1st metal layer is connected through TSV.

To connect the die with the others, some solder balls (or pillars) are placed on the top metal layer or back side metal layer. These solder balls and the underneath metal pad are called bumps in Innovus. The aligned bump between dies is called micro bump or landing pad. The cross die signal or power goes to/comes from the adjacent die through the micro bump. The bump between the die and the package substrate is called flip chip bump.

**Figure 2 TSV and Bump Cross-Section**



## TSV and Bump Cross-Section

All the physical information for back side metal, TSV, and bump is defined in the LEF file (the version for the LEF file should be LEF 57 or later). MB is modeled as ROUTING layer before the first normal metal, and a LEF property "BACKSIDE" is assigned to MB. TSV is model as a CUT layer with a LEF property "TYPE TSV". Below is an example of the additional information in LEF file for TSV and back side metal definition.

The pad metal of the bump is also described in the LEF file. In the LEF file, the bump is modeled as a cell. The bump cell has a pin which has the same shape and layer with the pad metal of the bump. The solder ball information is not described in the LEF. The other way to categorize the bump is based on the bump pad layer. If a bump is on top metal layer, it could be called as front bump, and if a bump is on back side metal layer it is back bump. Micro Bump could be either front bump or back bump depending how the die is stacked.

## Example

The statement in LEF to describe the TSV is given below:

```
PROPERTYDEFINITIONS
  LAYER LEF58_BACKSIDE STRING ;
  LAYER LEF58_TYPE STRING ;
  ...
END PROPERTYDEFINITIONS
  LAYER MB      # Back side metal layer
  TYPE ROUTING ;
  PROPERTY LEF58_BACKSIDE "BACKSIDE ; " ;
  ...
END MB
```

```
LAYER TSV      # TSV cut layer between M1 & MB
TYPE CUT ;
PROPERTY LEF58_TYPE "TYPE TSV ; " ;
SPACING 20.0 LAYER OVERLAP ;

...
END TSV

LAYER M1      # M1
TYPE ROUTING ;
...
END M1

...
VIA VIAB1
RESISTANCE 0.01 ;
LAYER MB ;
RECT -11.00 -11.00 11.00 11.00 ;
LAYER TSV ;
RECT -5.00 -5.00 5.00 5.00 ;
LAYER M1 ;
RECT -7.0 -7.0 7.0 7.0 ;
END VIAB1
```

## Defining Keep Out Area in Hard Macros

You can define keep-out area constraints inside macros to avoid overlap between bumps/adjacent die edges and these areas. Innovus will report warning if creating bumps inside the keep-out area, and this keep-out area information can be passed to the adjacent die.

Blockage area can be specified in hard macros by creating the following layers of OBS in the macro LEF file:

- Passivation layer in front side
- Passivation layer in back side
- Master slice layer for top die
- Master slice layer for bottom die

Use the passivation layers in the LEF file for defining bump keep-out area, and Master Slice layers in LEF for defining die edge checking. The statement in the LEF file to define blockage for die edge checking and bump keep-out area is given below:

### **Layer for back bump keep out region**

```
LAYER BACKPASSIV
  TYPE CUT ;
  PROPERTY LEF58_TYPE "TYPE PASSIVATION ; " ;
  PROPERTY LEF58_BACKSIDE "BACKSIDE ; " ;
END BACKPASSIV
```

### **Layer for front bump keep out region**

```
LAYER TOPPASSIV
  TYPE CUT ;
  PROPERTY LEF58_TYPE "TYPE PASSIVATION ; " ;
END TOPPASSIV
```

### **A new layer for top die edge**

```
LAYER TOPDIE
  TYPE MASTERSLICE ;
  PROPERTY LEF58_TYPE "TYPE ABOVEDIEEDGE ; " ;
END TOPDIE
```

### **A new layer for bottom die edge**

```
LAYER BOTTOMDIE
  TYPE MASTERSLICE ;
  PROPERTY LEF58_TYPE "TYPE BELOWDIEEDGE ; " ;
END BOTTOMDIE
```

Then, define the OBS layers in the macro for these constraints. The sample syntax for OBS layer definition is given below:

.....

OBS

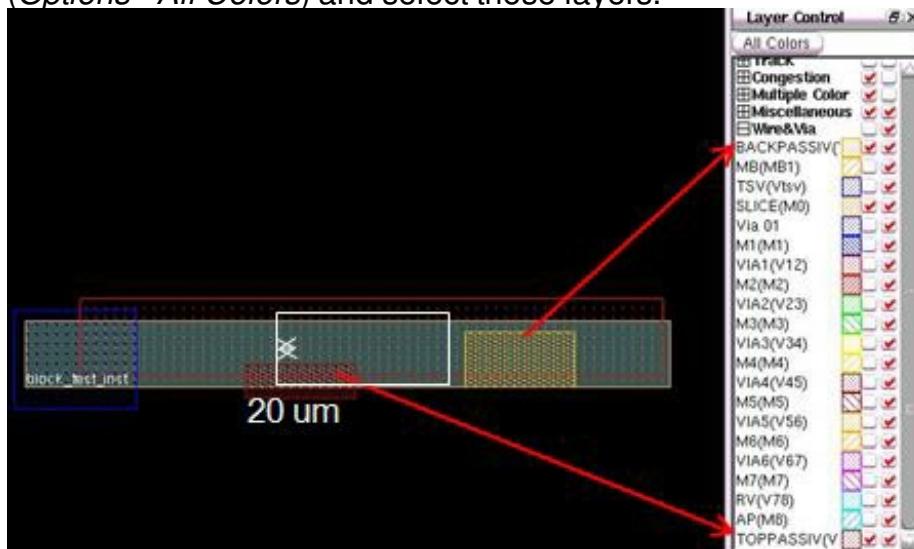
```
LAYER TOPDIE
SPACING 10 ;
```

```

RECT -10 -20 100.940 70.885 ;
LAYER BOTTOMDIE
SPACING 5 ;
RECT 50 10 580 80.885 ;
LAYER TOPPASSIV
SPACING 20 ;
RECT 200 -10 300 20 ;
LAYER BACKPASSIV ;
RECT 400 0 500 50 ;

```

To control the visibility of the OBS layers, click the *Wire/Via* tab of the *Color Preferences* form (*Options - All Colors*) and select these layers.



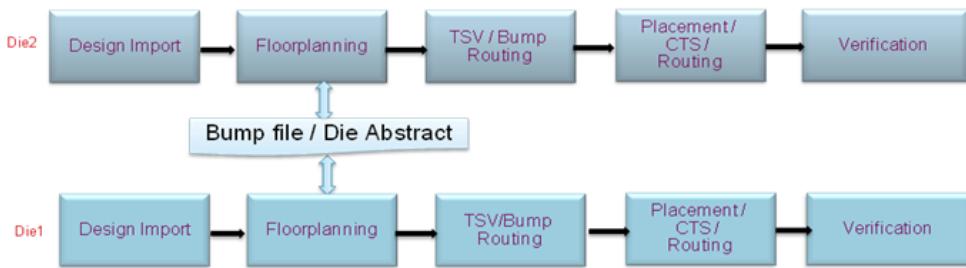
## Check Bump Keep Out Area Violation

Use the `verify_stacked_die -check_type {bump_keepout}` command parameter to check the overlap between the OBS layer and bumps (both front and back bumps). You can use the `verify_stacked_die -check_type {die_edge}` command parameter for die edge violation checking between adjacent dies. Once you define an OBS layer inside a macro, the adjacent die edge should not overlap this OBS area. If there is any violation between macro OBS and adjacent die edge, it will be marked in the GUI window. The `verify_stacked_die` command must be used after `readDieAbstract` to input adjacent die's die abstract file. Both top and bottom die edge violation is checked at the same time.

# 3D IC Flow in Innovus

The following figure shows the general Innovus 3D IC design flow. Only 2 dies are included in this flow chart, and it can be extended to multiple dies.

**Figure 3** TSV Design Flow in Innovus

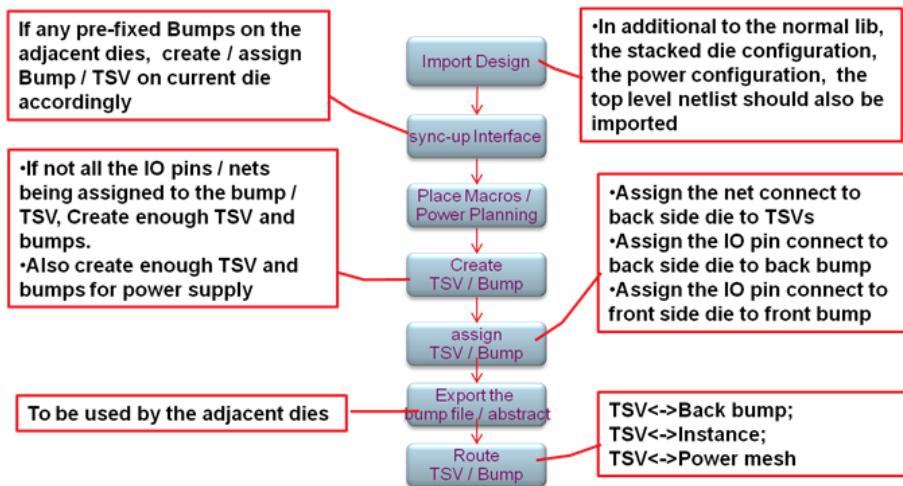


Innovus designs one die at a time. When designing one die, the micro bump and the instance on the adjacent die are honored and the micro bump on the current die is synchronized and optimized accordingly. The design flow for each die is quite similar to the normal design, except the following steps:

1. In the design import stage, additional configuration for stacked die should be imported.
2. In the floorplan stage, interface between each dies is ensured to be synchronized, and TSV/Bumps is created and assigned.
3. After floorplanning, the TSV and bump should be routed.
4. In the verification stage, the connectivity between dies should be verified.

Figure 4 shows the detailed floorplan flow for a 3D IC design.

**Figure 4** Detailed Floorplan Flow for One of the Dies



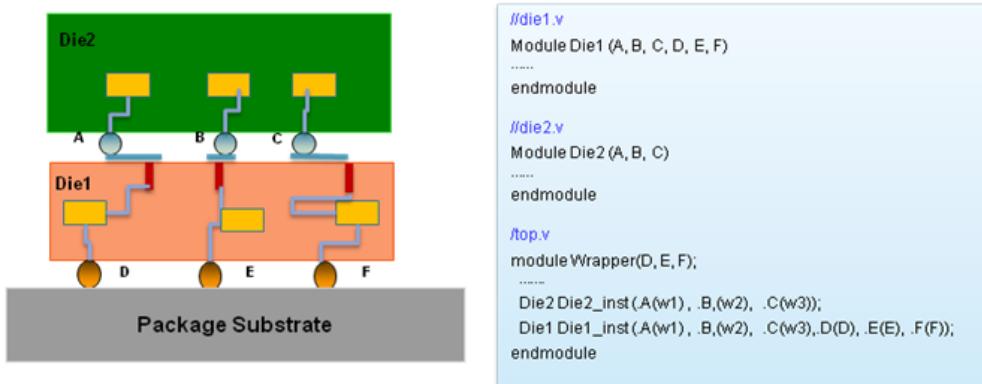
## Design Import

Innovus takes the conventional and the additional information for stacked dies as inputs for each die design.

## Stacked IC Verilog Input

Innovus inputs an additional top-level net list to describe the die-to-die interconnection and die to package interconnection. In the top-level netlist, package is set as the top module; each die is instantiated and the connectivity among dies is described. Figure 5 shows an example of the top-level netlist.

**Figure 5** Top-Level Netlist Example



The design .globals file needs to be modified to import the top-level netlist. Except for the module of

the die to be designed, the top-level netlist should also be included in the design .globals file. The init\_top\_cell should be set as the module name of the die to be designed. Considering Figure 5 as an example, the following setting in the design configuration is for Die1 design.

```
set init_top_cell {Die1}  
set init_verilog { Die1.v top.v }
```

## Stack Configuration Input

Innovus inputs an xml that describes the position and the orientation for each die. The xml file could be generated and edited through the *Stacked Config Editor* form.

For more information, see the "*Stacked Config Editor*" form in the Tools Menu chapter of the *Innovus Menu Reference Guide*.

## Power Connectivity Input

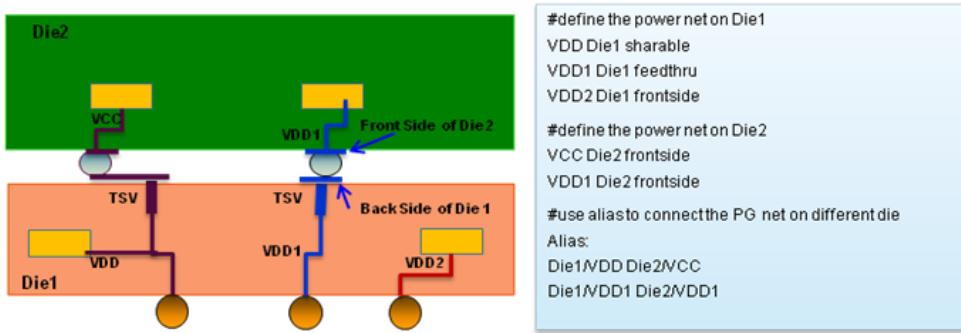
A text file is used to set up P/G net configuration on each die.

There are four types of P/G nets: frontside, backside, sharable, feedthru.

- frontside: the P/G net is connected to the front side bump only
- backside: the P/G net is connected to the back side bump only
- sharable: the P/G net is connected to both front and back side bumps and other instance in the chip
- feedthru: the P/G net is connected to front and back side bumps directly, without the connection to any instance of current chip; use alias to define the same P/G net on different dies

Figure 6 shows an example of a power connectivity file.

### Figure 6 Example of Power Configuration File



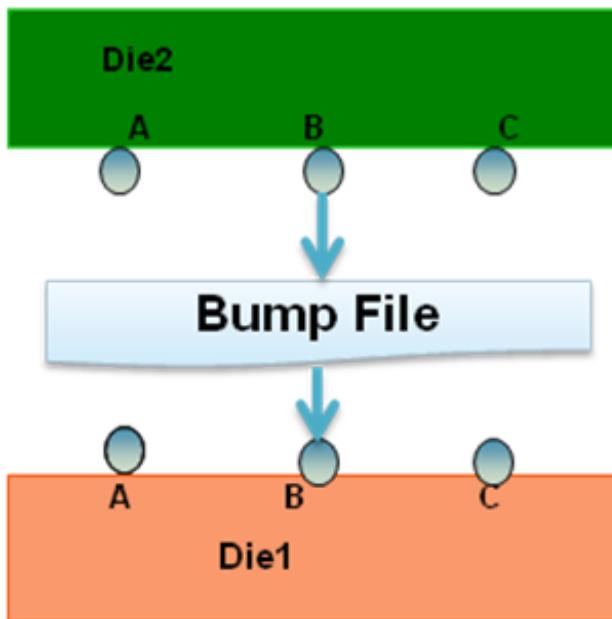
## Related Information

- "readTSVConfig" command in the "Through Silicon Via Design Commands" chapter of the *Innovus Text Command Reference*.
- "Load Stacked Die Config" form in "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference*.

# Interface Synchronization and Information Exchange between Dies

In 3D ICs, the micro bump is the data/power path between dies. As shown in Figure 2, the pad of micro bump should be exactly aligned to build the connection between the dies. Innovus uses the bump file to transfer the bump information from one die to the adjacent die. The bump file contains all the bump information of a die. Firstly, the bump file is dumped out by `writeBumpLocation` command from one die. Then `readBumpLocation` command should be called on the adjacent die. `readBumpLocation` command creates or assigns micro bumps on the current die according to the bumps on the adjacent die, as shown in the following picture.

**Figure 7** Interface Sync-Up Flow



The write/read bump information process could be iterated between the two adjacent dies. If the micro bump on one of the dies is changed, the bump information should be written out and read by the other die.

When designing one die, the adjacent die information should be honored and displayed. This step is not essential, but helpful for optimization across the dies and for manual debug. Firstly, the die abstract file is dumped out by the `writeDieAbstract` command from one die. Then, the `readDieAbstract` command should be called on the adjacent die. `readDieAbstract` command imports the adjacent die information. This information is displayed and honored by the die being designed.

## Related Information

- "writeBumpLocation", "readBumpLocation", "writeDieAbstract" and "readDieAbstract" commands in the "Through Silicon Via Design Commands" chapter of the *Innovus Text Command Reference*.
- "Exchange Information Between Dies" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference Guide*.

# TSV and Bump Manipulation

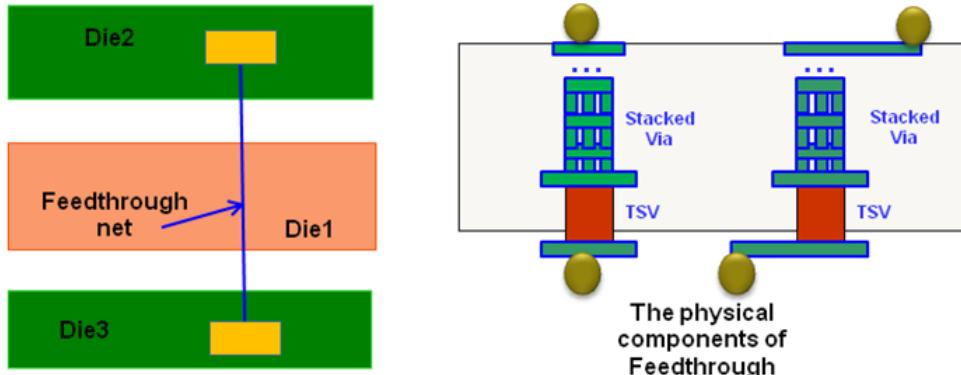
Innovus provides the capability to create/delete and assign/unassign TSV and bumps. Run the `addTSV` command to create the TSV and bumps within the user-specified box, or with user specified number. Since TSVs and bumps are the path between the dies, place TSVs and back bumps close to the IOs/Blocks whose pin connects to the adjacent dies on the back side. Keep TSVs outside the core area, unless a TSV has to be connected with a block inside the core area because the TSVs will break the follow pin.

After the TSVs and bumps are created, run the `assignTSV` command to assign nets to the TSVs and bumps.

## Feedthru Handling

In a TSV design, there is a special kind of net called feedthrough net. The feedthrough net has only two pins: one is the IO pin on the front side and the other is the IO pin on the back side. It does not connect to any instance on the die. The feedthrough net on one die is defined for the connection between the adjacent dies on the front side and back side, as shown in Figure 8.

**Figure 8 Logic Concept and Physical Components of Feedthrough**



You can define the feedthrough net in the Verilog netlist by assigning the two IO pins on the front side and back side together. To create a feedthrough, specify the `-tsvFeedthru` parameter of the `addTSV` command. `assignTSV` command assigns the feedthrough net and the normal net simultaneously or separately by turning on and off the `-feedthru` and `-nonFeedthru` parameters. The embedded TSV, which is modeled as a pin on the back side metal can also be supported by `assignTSV`. `assignTSV` will not assign this back side pin to TSV.

## Related Information

- "addTSV", "deleteTSV", "assignTSV" and "unassignTSV" commands in the " Through Silicon

"Via Design Commands" chapter of the *Innovus Text Command Reference*.

- "TSV/Bump Manipulation" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference*.

## TSV and Bump Routing

In 3DIC design, User could use fcroute for bump, TSV, IO pad and power stripe routings:

1. Route TSV to IO Pad. Use fcroute with aio mode to support TSV to IO pad routing. For example:

```
fcroute -type signal -designStyle aio -connectTsvToPad -routeWidth 3 -  
layerChangeTopLayer AP -layerChangeBotLayer M1
```

2. Route TSV to bump. fcroute can not route front side bump and back side bump at the same time, which needs to route separately. For example, to route TSV and back side bump:

```
fcroute -type signal -designStyle aio -routeStyle 45DegreeRoute -routeWidth 8 -  
layerChangeTopLayer MB -layerChangeBotLayer MB -connectPowerCellToBump -  
connectTsvToBump -extraConfig ./conf/backside.extraConf
```

In the backside.extraConf, user need to define "srouteExcludeBumpType FRONT\_BUMP", "FRONT\_BUMP" is the cell name of the front side bump.

To route TSV and front side bump:

```
fcroute -type signal -designStyle aio -routeStyle manhattan -routeWidth 8 -  
layerChangeTopLayer AP -layerChangeBotLayer M1 -connectPowerCellToBump -  
connectTsvToBump -extraConfig ./conf/frontside.extraConf
```

In the frontside.extraConf, user need to define "srouteExcludeBumpType BACK\_BUMP", "BACK\_BUMP" is the cell name of the back side bump.

3. Route TSV to power stripe. Use `fcroute` with type power to route TSV and power stripes, for example:

```
fcroute -type power -connectTsvToRingStripe -routeWidth 6 -layerChangeTopLayer M2  
-layerChangeBotLayer M1
```

The embedded TSV is modeled as a pin on back side metal. It is supported by `fcroute`, which will honor the pin on back side metal.

## Related Information

- "`fcroute`" and "`routePointToPoint`" commands in the "Flip Chip Commands" chapter of the *Innovus Text Command Reference*.
- "*Route TSV/Bump*" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference*.
- "*Flip Chip Methodology*" chapter of the *Innovus User Guide*.

## Cross Die Connectivity Verification

The `verifyConnectivity` command checks whether the connectivity between the dies is correctly implemented, that is, the micro bumps on the adjacent die with the same signal are aligned.

To check the micro bump alignment on one die, you need to dump the die abstract of all the adjacent dies, and then run the command `verifyConnectivity -tsv die_abstract_list`. The violation will be shown on the violation browser, and the violation marker will be displayed on the layout window.

`readBumpLocation -checkAlignment` can also be used for checking bump alignment.

## Related Information

- "`verifyConnectivity`" command in the "Verify Commands" chapter of the *Innovus Text Command Reference*.
- "`readBumpLocation`" command in the "TSV Design Commands" chapter of the *Innovus Text Command Reference*.
- "*Verify Connectivity*" form in the "Verify Menu" section of the Tools Menu chapter in the *Innovus Menu Reference*.

- "Identifying and Viewing Violations" chapter in the *Innovus User Guide*.

## **Innovus User Guide**

Design Methodology for 3D IC with Through Silicon Via--Cross Die Connectivity Verification

---

## Syntax and Scripts

---

- [CCOpt Properties](#)
- [Creating the ICT File](#)
- [Clock Mesh Specification File](#)
- [Supported CPF 1.0 Commands](#)
- [CPF 1.0 Script Example](#)
- [Supported CPF 1.0e Commands](#)
- [CPF 1.0e Script Example](#)
- [Supported CPF 1.1 Commands](#)
- [CPF 1.1 Script Example](#)
- [Supported SAI Commands](#)
- [Cadence-Specific Liberty Extensions](#)

# CCOpt Properties

- add\_driver\_cell
- add\_exclusion\_drivers
- adjacent\_rows\_legal
- allow\_non\_standard\_inputs\_clock\_gate
- allow\_resize\_of\_dont\_touch\_cells
- associated\_row
- auto\_limit\_insertion\_delay\_factor
- auto\_limit\_insertion\_delay\_factor\_skew\_group
- auto\_sinks
- auto\_sinks\_filter
- balance\_mode
- bounds
- buffer\_cells
- call\_cong\_repair\_during\_final\_implementation
- cannot\_clone\_reason
- capacitance\_margin\_absolute
- capacitance\_margin\_percentage
- capacitance\_override
- case\_analysis
- ccopt\_auto\_limit\_insertion\_delay
- ccopt\_check\_db\_every\_tcl\_command
- ccopt\_merge\_clock\_gates
- ccopt\_merge\_clock\_logic
- ccopt\_worst\_chain\_report\_timing\_too
- cell\_density
- cell\_halo\_x
- cell\_halo\_y
- center\_line
- check\_route\_follows\_guide
- check\_route\_follows\_guide\_max\_deviation
- check\_route\_follows\_guide\_min\_length
- clock\_gate\_buffering\_location
- clock\_gating\_cells
- clock\_gating\_depth
- clock\_gating\_depth\_top\_down
- clock\_gating\_only\_optimize\_above\_flops
- clock\_tree
- clock\_tree\_generator\_sink\_is\_leaf
- clock\_tree\_source\_group
- clock\_trees
- clone\_clock\_gates
- clone\_clock\_logic

- compatibility\_warning
- consider\_em\_constraints
- consider\_power\_management
- constrains
- cts\_edge\_sensitivity
- cts\_isolated
- cts\_merge\_clock\_gates
- cts\_merge\_clock\_logic
- debug\_stage\_delays\_for\_nodes
- def\_lock\_clock\_sinks\_after\_routing
- delay\_cells
- detailed\_cell\_warnings
- effective\_clock\_period
- effective\_clock\_period\_sources
- effective\_sink\_type
- effort
- enable\_all\_views\_for\_io\_latency\_update
- enable\_datapoints
- enable\_initial\_clustering
- enable\_locked\_node\_check\_failure
- engine\_implemented
- error\_on\_problematic\_slew\_violating\_nets
- error\_on\_problematic\_slew\_violating\_nets\_max\_printout
- exclusive\_sinks\_rank
- expand\_multi\_child\_regions
- extract\_balance\_multi\_source\_clocks
- extract\_check\_timing\_slacks
- extract\_clock\_generator\_skew\_group\_name\_prefix
- extract\_clock\_generator\_skew\_groups
- extract\_clock\_node\_timing\_endpoints
- extract\_create\_explicit\_ignores\_for\_implicit\_excludes\_beyond\_sta\_clock\_network
- extract\_cts\_case\_analysis
- extract\_faster\_sdc\_clocks\_as\_clock\_trees
- extract\_merge\_identical\_skew\_groups
- extract\_network\_latency
- extract\_no\_exclude\_pins
- extract\_skew\_group\_sinks\_at\_clock\_node\_timing\_endpoints
- extract\_source\_latency
- extract\_through\_multi\_output\_cells\_with\_single\_clock\_output
- extract\_through\_to
- extracted\_from\_clock\_name
- extracted\_from\_constraint\_mode\_name
- extracted\_from\_delay\_corners
- extract\_pin\_insertion\_delays

- fastest\_buffer
- fastest\_buffer\_max\_trans
- fastest\_inverter
- fastest\_inverter\_max\_trans
- final\_cell
- flexible\_htree\_max\_synthesis\_grid\_points
- force\_all\_module\_boundaries\_dont\_touch
- force\_all\_virtual\_delay\_updates
- force\_clock\_objects\_to\_propagated
- force\_clock\_topology\_changed\_after\_adjustment
- force\_clock\_tree
- force\_cluster\_only
- force\_update\_io\_latency
- generated\_by\_sinks
- grid\_step
- hard
- htree\_sinks
- hv\_balance
- ignore\_pins
- implicit\_sink\_type
- include\_source\_latency
- insertion\_delay
- insertion\_delay\_wire
- inverter\_cells
- is\_user\_skew\_group
- is\_vertical
- last\_target\_insertion\_delay
- last\_target\_insertion\_delay\_wire
- last\_target\_max\_trans
- last\_target\_skew
- last\_target\_skew\_wire
- leaf\_buffer\_cells
- leaf\_inverter\_cells
- legalized\_on\_clock\_spine
- lock\_on\_clock\_spine
- locked\_originally
- log\_precision
- log\_special\_case\_cell\_selections
- logic\_cells
- low\_power\_clustering
- manage\_power\_management\_illegalities
- max\_buffer\_depth
- max\_clock\_cell\_count
- max\_delta\_band\_factor

- max\_delta\_slack\_factor
- max\_fanout
- maximum\_insertion\_delay
- max\_skew\_passes\_with\_insufficient\_progress
- max\_source\_to\_sink\_net\_length
- max\_source\_to\_sink\_net\_resistance
- min\_delta\_band\_factor
- move\_clock\_gates
- move\_clock\_nodes\_for\_slack
- move\_logic
- move\_middle\_cell\_first\_when\_adding\_wire\_delay
- name\_prefix
- net\_unbufferable
- no\_symmetry\_buffers
- opt\_ignore
- optimize\_ir\_drop\_skew\_amount
- override\_minimum\_max\_trans\_target
- override\_minimum\_skew\_target
- override\_vias
- override\_zero\_placeable\_area
- parents
- pin
- pin\_target\_max\_trans
- post\_conditioning
- post\_conditioning\_enable\_drv\_fixing
- post\_conditioning\_enable\_drv\_fixing\_by\_rebuffering
- post\_conditioning\_enable\_drv\_fixing\_by\_rebuffering\_ccopt
- post\_conditioning\_enable\_skew\_fixing
- post\_conditioning\_enable\_skew\_fixing\_by\_rebuffering
- post\_conditioning\_enable\_skew\_fixing\_by\_rebuffering\_ccopt
- primary\_delay\_corner
- print\_clock\_tree\_modifications
- put\_driver\_in\_centre\_of\_fanout\_bounding\_box
- recluster\_to\_reduce\_power
- remove\_bufferlike\_clock\_logic
- rename\_clock\_tree\_nets
- report\_only\_skew\_group\_with\_target
- report\_only\_timing\_corners\_associated\_with\_skew\_groups
- resistance\_margin\_absolute
- resistance\_margin\_percentage
- restricted\_cells\_buffers
- restricted\_cells\_inverters
- route\_balancing\_buffers\_with\_default\_rule
- route\_type

- route\_type\_autotrim
- routing\_override
- routing\_top\_fanout\_count
- routing\_top\_min\_fanout
- routing\_top\_transitive\_fanout
- save\_old\_viz\_on\_cluster
- schedule
- sink\_grid
- sink\_grid\_box
- sink\_grid\_exclusion\_zones
- sink\_grid\_prefix
- sink\_grid\_sink\_area
- sink\_type
- sinks
- sinks\_active
- size\_clock\_gates
- size\_logic
- skew\_band\_size
- skew\_group\_insertion\_delay
- skew\_group\_insertion\_delay\_wire
- skew\_group\_report\_columns
- skew\_group\_report\_histogram\_bin\_size
- skew\_groups\_active
- skew\_groups\_active\_sink
- skew\_groups\_ignore
- skew\_groups\_sink
- skew\_groups\_source\_pin
- skew\_pass\_sufficient\_progress\_band\_size\_factor
- skew\_passes\_per\_cluster
- skip\_move\_gates\_in\_chains
- source\_driver
- source\_group\_clock\_trees
- source\_input\_max\_trans
- source\_latency
- source\_max\_capacitance
- source\_output\_max\_trans
- source\_pin
- sources
- stop\_at\_sdc\_clock\_roots
- stripes\_bbox
- stripes\_bottom\_edge
- stripes\_cells
- stripes\_height
- stripes\_left\_edge

- stripes\_pitch
- stripes\_repeat
- stripes\_width
- target\_insertion\_delay
- target\_insertion\_delay\_wire
- target\_max\_capacitance
- target\_max\_trans
- target\_max\_trans\_sdc
- target\_multi\_corner\_allowed\_insertion\_delay\_increase
- target\_skew
- target\_skew\_wire
- timing\_delta
- top\_buffer\_cells
- top\_inverter\_cells
- trace\_bidi\_as\_input
- transitive\_fanout
- trunk\_cell
- trunk\_override
- update\_io\_latency
- update\_slacks\_before\_skewing\_adjustors
- use\_estimated\_routes\_during\_final\_implementation
- use\_inverters
- use\_macro\_model\_pin\_cap\_only
- use\_skew\_implementation\_cache\_hold\_slacks
- useful\_skew\_max\_delta
- useful\_skew\_min\_delta
- useful\_skew\_post\_implement\_db
- useful\_skew\_pre\_implement\_db
- user\_skew\_group
- virtual\_delay
- visualization\_clock\_trees\_initially\_collapsed
- weak\_driver\_check\_bounding\_box\_vs\_drive\_distance

To set the value of any CCOpt property, use the `set_ccopt_property` command:

```
set_ccopt_property use_inverters -clock_tree ct1 true
```

To get the current value of any CCOpt property, use the `get_ccopt_property` command:

```
get_ccopt_property property_name
```

Example:

```
get_ccopt_property cell_density
```

The software returns the following information:

1

To get a description of any CCOpt property, use the following command:

```
get-ccopt_property -help property_name
```

For more information, see "CCOpt Property System" in the [Clock Tree Synthesis](#) chapter of the *Innovus User Guide*.

## **add\_driver\_cell**

Specify a driver cell to add for clock tree after root.

*Default:* {}

Optional applicable arguments: "-clock\_tree <name>".

## **add\_exclusion\_drivers**

Setting this property true causes CCOpt to add extra drivers above exclude pins to remove them from the clock tree.

*Default:* false

This global property does not use additional arguments.

## **adjacent\_rows\_legal**

Specifies whether CCOpt is allowed to place clock tree cells in a row adjacent to another clock tree cell. This is a similar control to the cell\_density property, but specifies whether a single row "exclusion zone" applies in the Y-axis. If set to false, clock tree cells cannot be placed in adjacent rows when the X-axis overlaps. Setting this property to false can help with IR drop problems in the clock tree, but may cause a degradation in clock tree QoR, particularly on highly utilized designs.

Valid values: true false

See also: cell\_density

*Default:* false

This global property does not use additional arguments.

## **allow\_non\_standard\_inputs\_clock\_gate**

If this property is set, CTS will allow the use of clock gates with non-standard pins. CCOpt considers the following pin types to be standard: clock pins, enable pins, test enable pins, retention pins and power gating pins. Before starting CTS CCOpt will emit a warning, indicating which pin(s) it considers non-standard.

Type: boolean

*Default:* false

Current Value: false

This global property does not use additional arguments.

## **allow\_resize\_of\_dont\_touch\_cells**

Allows CTS to perform resizes on dont\_touch clock instances. When `true`, instances marked as being `dont_touch` are resized during CTS. When `false`, instances marked as being `dont_touch` are not resized during CTS. In this case, instance sizing can be permitted on a per instance basis by setting the `ResizeOnly` property like this:

```
dbSetPropValue [dbGetOrCreatePropByName [dbGetInstByName <instName>] ResizeOnly] 1
```

*Valid values:* `true` `false`

*Default:* `true`

This global property does not use additional arguments.

## **associated\_row**

For horizontal clock spines, this property returns the name of the cell row on which the `center_line` of the clock spine is deemed to lie. It will be the row that gets clock spine cell clock pins nearest to the `center_line` of the clock spine. This property is undefined for vertical clock spines and will return an empty string in that case.

*Valid values:* `string`

*Read-only:* This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

*Optional applicable arguments:* `"-clock_spine <name>"`.

## **auto\_limit\_insertion\_delay\_factor**

CCOpt attempts to keep the insertion delays of each clock tree a fixed multiple of the longest insertion delay that would result from a global skew approach. This multiple can be modified by design timing and the presence of other clock trees, but will start at a fixed fraction above the global skew insertion delay. This property specifies that fixed fraction.

*Valid values:* `real`

*See also:* [auto\\_limit\\_insertion\\_delay\\_factor\\_skew\\_group](#)

*Default:* `1.5`

This global property does not use additional arguments.

## **auto\_limit\_insertion\_delay\_factor\_skew\_group**

CCOpt uses this property to generate a maximum insertion delay for any skew groups on which it is set. This is done by multiplying the insertion delay for that skew group that would result from a global skew approach by the specified factor. For any skew groups where this property is not set, a limit based on the longest global skew insertion delay across all skew groups is used; see the `auto_limit_insertion_delay_factor` property.

*Valid values:* `real`

*See also:* [auto\\_limit\\_insertion\\_delay\\_factor](#)

*Default:* `auto`

Optional applicable arguments: "-skew\_group <name>".

## auto\_sinks

Specifies whether CCOpt automatically determines the target sinks for the skew group based on the input pins reachable from the source.

Valid values: true false

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-skew\_group <name>".

## auto\_sinks\_filter

A list of allowed sinks for this skew group. Only sinks in this list will be eligible as `auto_sinks` for the skew group.

Valid values: list cell

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-skew\_group <name>".

## balance\_mode

Replace CCOpt mode setting `cts_opt_type {full | cluster | trial}`. If not full, causes CCOpt and CCOpt CTS to halt before final completion of the clock tree to facilitate clock tree inspection.

The possible values for this property are as follows:

- full - default value, a full CTS is performed.
- cluster - a cluster-only CTS is performed. The clock tree has no balancing delay applied.
- trial - The clock has only virtual (numeric annotation) balancing delays applied.

Type: string

Default: full

Current Value: full

This global property does not use additional arguments.

## bounds

Describes the physical space that the spine occupies in the form of a tcl list of four values: {xmin ymin xmax ymax} All (x/y) (min/max) values are of type double expressed in microns.

Valid values: list double

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-clock\_spine <name>".

## buffer\_cells

A Tcl list of symmetric buffer cells. All buffers created in the clock tree under a power domain will be instances of these cells. Set the global property to specify buffer cells for all clock trees and all power domains:

```
set_ccopt_property buffer_cells {bufA bufB bufC}
```

Set the per-clock tree/power domain property to specify buffer cells for a particular clock tree and ALL power domains:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk
```

Set the per-clock tree /power domain property to specify buffer cells for a particular clock tree and power domain:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk -power_domain pd
```

By default CCOpt automatically selects appropriate cells.

Valid values: list lib\_cell

Default: {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## call\_cong\_repair\_during\_final\_implementation

If set, congRepair is called during the final implementation clock routing phase. This can reduce congestion and ease later routing issues.

Valid values: true or false

Default: true

This global property does not use additional arguments.

## cannot\_clone\_reason

A list of reasons to explain why the cell could not be cloned during clustering. This property will not be set if properties `clone_clock_gates`/`clone_clock_logic` are `false`. The property returns a string list, each string being a reason why the cell cannot be cloned.

There are three categories of reason as to why a cell can not be cloned: the cell is marked as dont touch, reasons specific to the clock node and general reasons.

The possible `dont_touch` reasons are:

<code>dont_touch.add_driver_cell</code>	Set if the cell was added below the root via property <code>add_driver_cell</code>
---	--

dont_touch.clock_root	Set during clock tree extraction if identified as having the root pin
dont_touch.clock_sink	Set on cell if any pin is a clock sink in a clock tree
dont_touch.clock_tree_generator_path	Set if nodes and wires are in clock tree generator paths
dont_touch.clock_wire	Set on cell if clock input wire is user dont_touch
dont_touch.clockgate_no_power_domain	Set if clockgate/clocklogic is a clocknode in a no clockgate/clocklogic power domain
dont_touch.dft_observability_test_mode_pin	Set if a cell has the <code>dft_observability_test_mode_pin</code>
dont_touch.drives_multi_driver_net	Set if the cell drives wires with other drivers
dont_touch.external_skew_group_pin	Set if a pin on the cell is a skew group sink, source or ignore pin for a skew group created by the user
dont_touch.flexible_htree	Set if the cell was added to a flexible H-tree
dont_touch.internal_skew_group_pin	Set if a pin on the cell is a skew group sink, source or ignore pin for a skew group created by CCOpt
dont_touch.neg_edge_clock_gate	Set if a clock gate gates the falling edge ( <code>extract_enable_neg_edge_clock_gate_handling</code> is true)
dont_touch.non_flop_clock_gating	Set by SetDontTouchBlackBoxGating
dont_touch.non_standard_inputs_clock_gate	Set if a cell is a clock gate with 'non-standard' inputs, to avoid disconnecting them, or if property <code>consider_all_clock_gates_to_have_non_standard_inputs</code> is true
dont_touch.observability_clock_gate	Set if a cell is a clock gate with an observability output pin
dont_touch.placer.lock	Set if a cell is user locked or locked by DEF
dont_touch.power_management	Set if a cell is a power management cell
dont_touch.sdc	Set if constrained by SDC timing
dont_touch.sdc_path_group	Set if there is an SDC path group start/endpoint on one of the pins of this cell
dont_touch.sub_block	Set if cell is in dont_touch module
dont_touch.unmergeable_composite_clock_gate	Set if some of the clock gate is dont_touch
dont_touch.user	Set by user
Contact Cadence Support if any of the following reasons are listed:	

dont_touch.cannot_understand_clock_gate	Set if clock gate not recognized, extract_enable_neg_edge_clock_gate_handling <b>is</b> false
dont_touch.composite_clock_gate_enable_test_or_gate	
dont_touch.initial_netlist	
dont_touch.output_wire	
dont_touch.port_preservation_prevents_cg_opt	
dont_touch.prevent_assign	

The possible reasons specific to the clock node are:

PowerManagement	Set if cell is power management cell and property clone_power_management_cells <b>is</b> false
PowerManagementInconsistency	Set if a disconnection of the cell from the netlist would cause a power management inconsistency
PreservedUMPB	Set if cell is a User Module Port Bit which is preserved

Contact Cadence Support if any of the following reasons are listed:

ClockGatesAlways	Set if cell is clock gate, considered always dont_touch and property clock_tree_clock_gate_logic_always_dont_touch <b>is</b> true
ClockLogicAlways	Set if cell is clock logic, considered always dont_touch and property clock_tree_clock_gate_logic_always_dont_touch <b>is</b> true
NodesRoot	Set if the cell is the clock root (cannot be cloned)
NoRoot	Set if the clock node has no root
RootIgnored	Set if the clock root is ignored
SpineCell	Set if cell is a clock spine cell and is marked as dont_touch

The possible General Reasons are:

DrivesAcrossPowerDomains	The cell drives across power domain boundaries
NodeHasSGConstraints	The cell has user mode skew group constraints in default mode and property clone_clock_cells_with_skew_group_constraints <b>is</b> false
InvertingClockGate	The cell is an inverting clock gate and cloning of these is not supported
ClockDriverCannotCloneInverter	
ClockDriverCreatedByBuffLongNets	Need to keep cell as created by 'Buffering long nets' and property clustering_leave_cmf_drivers_alone <b>is</b> true

ClockDriverInverterCloningDisabled	The cell is an inverter and cloning inverters is disabled (property <code>clone_inverters</code> is <code>false</code> )
Contact Cadence Support if any of the following reasons are listed:	
ClockLogicMultiOutputCell	
IgnoredTree	
IsUnclonable	
NonIntegratedClockGate	
NotTreeViewNode	
OutputPinCellPinNull	
ClockDriverNeedToKeep	

Valid values: list string

*Default:* {}

Applicable arguments: "-inst <name>". Required: "-inst <name>".

## **capacitance\_margin\_absolute**

Capacitance absolute margin used in CTS clustering. Specified in femtoFarads.

Valid values: double, or auto

In auto, CCOpt automatically determines an appropriate margin

*Default:* auto

This global property does not use additional arguments.

## **capacitance\_margin\_percentage**

Capacitance percentage margin used in CTS clustering.

Valid values: double or auto

In auto, CCOpt automatically determines an appropriate margin

*Default:* auto

This global property does not use additional arguments.

## **capacitance\_override**

Specifies an overridden capacitance value for this pin. Can be specified in pF, fF, F (default pF). Can set per delay corner (delay\_corner) and with reference to an event (rise|fall). If set to `auto`, CCOpt will calculate the capacitance value using the library information.

**Valid values:** Library dependent

**Default:** `auto`

**Applicable arguments:** `"-delay_corner <name>"`, `"-pin <name>"`, `"-rise"` and `"-fall"`. **Required:** `"-pin <name>"`.

## **case\_analysis**

Specifies a constant value to be used as pin signal when analyzing timing within CTS. Only applies to input pins of cells in the clock tree. If set to `'none'` (the default), CTS will consider the input as non-constant.

**Valid values:** 0 1 none

**Default:** `auto`

**Applicable arguments:** `"-pin <name>"`. **Required:** `"-pin <name>"`.

## **ccopt\_auto\_limit\_insertion\_delay**

If set to `true`, CCOpt will automatically limit the insertion delay of clock trees. The limits are determined by looking at the value of the `ccopt_auto_limit_insertion_delay_factor` property, and also taking into considering the timing of the design and the insertion delay of other clock trees in the design.

**Type:** boolean

**Default:** `true`

**Current Value:** `true`

This global property does not use additional arguments.

## **ccopt\_check\_db\_every\_tcl\_command**

Specifies whether CCOpt runs the database checks invoked by `ccopt_check_db` after every Tcl command. This feature is currently turned off by default. To activate, set the following property:

```
set_ccopt_property ccopt_check_db_every_tcl_command true
```

**Valid values:** `true` `false`

**See also:**

- `ccopt_check_db_continue_after_error`
- `ccopt_check_db_release_error_object`

**Default:** `false`

This global property does not use additional arguments.

## **ccopt\_merge\_clock\_gates**

If set to `true`, clock gate merging is enabled. If this is `false`, merging of all clock gates is disabled, including clock gates which may have been cloned by CTS.

Note that this property has no impact on '`ccopt_design -cts`'. See also property `cts_merge_clock_gates`.

Valid values: `true false`

*Default:* `true`

Optional applicable arguments: "`-clock_tree <name>`".

## **ccopt\_merge\_clock\_logic**

If set to `true`, clock logic merging is enabled. If this is `false`, merging of all clock logics is disabled, including clock logics which may have been cloned by CTS.

Note that this property has no impact on '`ccopt_design -cts`'. See also property `cts_merge_clock_logic`.

Valid values: `true false`

*Default:* `true`

Optional applicable arguments: "`-clock_tree <name>`".

## **ccopt\_worst\_chain\_report\_timing\_too**

When CCOpt runs the worst chain report, additionally run [report\\_timing](#) .

Valid values: `true false`

*Default:* `false`

This global property does not use additional arguments.

## **cell\_density**

If set to less than 1.0, clock tree synthesis reduces the density of clock tree gates to this limit. For example, if set to 0.5, each clock tree buffer is surrounded by an equal area of either empty space or cells from the datapath to the left and the right of the clock tree cell. This property is useful when double-spacing the clock tree would result in routing violations - you can push the clock tree cells further apart and hence ease the routing problem.

Set the global property to specify the cell density for all clock trees:

```
set_ccopt_property cell_density 0.8
```

Set the per-clock tree property to specify the balance clock edge for a particular clock tree:

```
set_ccopt_property cell_density 0.9 -clock_tree clk
```

**Valid values:** 0.01 to 1

**See also:**

- . adjacent\_rows\_legal

**Default:** 0.75

Optional applicable arguments: "-clock\_tree <name>".

## cell\_halo\_x

Specifies the cell halo distance in the x direction. The default value of this property is auto. By default, CCOpt can automatically compute a `cell_halo_x` in terms of `cell_density` property.

**See also:**

- . `cell_density`

**Default:** auto

Optional applicable arguments: "-clock\_tree <name>" and "-cell <name>".

## cell\_halo\_y

Specifies the cell halo distance in the y direction. The default value of this property is auto. By default, CCOpt can automatically compute a `cell_halo_y` in terms of `cell_density` property.

**See also:**

- . adjacent\_rows\_legal

**Default:** auto

Optional applicable arguments: "-clock\_tree <name>" and "-cell <name>".

## center\_line

The center line of the clock spine, expressed as a pair of points, in microns. This is the line onto which the clock pins of cells assigned to the clock spine will be lined up with as closely as possible subject to cell site, row legalization and other design constraints. The value will be a tcl list of lists.

For example for a horizontal clock spine at a Y position of 2000um extending from a 200um X position to a 1800um X position the value returned would be {{200.0 2000.0} {1800.0 2000.0}}.

**Valid values:** list list double

**Read-only:** This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "-clock\_spine <name>".

## check\_route\_follows\_guide

Specifies whether to run diagnostic checks to ensure that clock nets routed by NanoRoute follow route guides. The diagnostic checks compare lengths of estimated routes to the final detailed wiring.

The `check_route_follows_guide_min_length` property lets you specify a minimum route length for diagnostic checks. Only nets longer than this minimum are checked.

The `check_route_follows_guide_max_deviation` property lets you specify the max allowed percentage deviation before diagnostics are triggered.

**Valid values:** true false

**See also:**

- `check_route_follows_guide_min_length`
- `check_route_follows_guide_max_deviation`

**Default:** true

This global property does not use additional arguments.

## check\_route\_follows\_guide\_max\_deviation

Maximum allowed percentage deviation before diagnostics are triggered when `check_route_follows_guide` is set to true.

**Valid values:** double

**See also:**

- `check_route_follows_guide`
- `check_route_follows_guide_min_length`

**Default:** 0.5

This global property does not use additional arguments.

## check\_route\_follows\_guide\_min\_length

Minimum net length to perform guide route diagnostics when `check_route_follows_guide` is set to true.

**Valid values:** double (in microns) | auto

Auto: computed automatically based on design

**See also:**

- `check_route_follows_guide`
- `check_route_follows_guide_max_deviation`

**Default:** auto

This global property does not use additional arguments.

## clock\_gate\_buffering\_location

Specifies where CCOpt should place delay buffers relative to clock gates during clock tree synthesis.  
If set to `below`, delay buffers will be put underneath clock gates. This is best for power, but may be detrimental to timing.  
If set to `above`, delay buffers will be put above clock gates; this is best for timing.  
If set to `auto`, CCOpt will automatically select where to put the buffers to best improve power without sacrificing timing.

Valid values: `auto` `above` `below`

*Default:* `above`

Optional applicable arguments: "`-clock_tree <name>`".

## clock\_gating\_cells

A Tcl list of clock gating cells. CCOpt makes all full cycle clock gates in the clock tree and power domain instances of these cells.

Set the global property to specify clock gating cells for all clock trees and all power domains:

```
set_ccopt_property clock_gating_cells {cgA cgB cgC}
```

Set the per-clock tree/power domain property to specify clock gating cells for a particular clock tree and ALL power domains:

```
set_ccopt_property clock_gating_cells {cgX cgY} -clock_tree clk
```

Set the per-clock tree /power domain property to specify clock gating cells for a particular clock tree and power domain:

```
set_ccopt_property clock_gating_cells {cgX cgY} -clock_tree clk -power_domain pd
```

By default CCOpt automatically selects appropriate cells.

Valid values: `list lib_cell`

*Default:* {}

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

## clock\_gating\_depth

The depth of gating at and below a clock gate, counted either top-down or bottom-up as determined by the value of the `clock_gating_depth_top_down` property. This property has a value of 0 for any cell that is not a clock gate in a defined clock tree.

Valid values: `int`

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

## **clock\_gating\_depth\_top\_down**

Determines whether the `clock_gating_property` counts gates top-down or bottom-up. In the former case, all gates immediately below a non-generated clock root are assigned the same level. In the latter, all gates immediately above sinks are assigned the same level. In both cases, lower numbers are closer to sinks, and higher numbers closer to roots.

Type: boolean

*Default:* true

Current Value: true

This global property does not use additional arguments.

## **clock\_gating\_only\_optimize\_above\_flops**

Allows CCOpt to work around spurious warnings and failures in downstream tools. If set to true, CCOpt does not optimize clock gating above RAMs, latches, black boxes, or any other clock tree sink that is not a flop. Power optimizations are achieved by transferring flops to a clone of the affected clock gating path. (Note that if CCOpt is run with `-cg_opt off`, flops are not transferred to a cloned clock gating path.)

Setting to false increases power saving opportunities, but may cause spurious formal equivalence issues with some third-party tools.

Type: boolean

*Default:* true

Current Value: true

This global property does not use additional arguments.

## **clock\_tree**

The clock tree to which this object belongs. Flops do not belong to a clock tree, but their clock pins do.

Valid values: `clock_tree`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

## **clock\_tree\_generator\_sink\_is\_leaf**

Controls whether the input pin of generator sinks is considered to be a leaf for clock tree routing rules and slew constraints.

Type: boolean

*Default:* false

Current Value: false

This global property does not use additional arguments.

## **clock\_tree\_source\_group**

Specifies the clock tree source group to which this clock tree belongs.

**Valid values:** list clock\_tree\_source\_group

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Optional applicable arguments:** "-clock\_tree <name>".

## **clock\_trees**

A list of clock trees the pin is contained within. This includes parents of generated clock trees and all relevant parents when clock trees overlap.

**Valid values:** list clock\_tree

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## **clone\_clock\_gates**

Specifies whether CCOpt should clone clock gates in an attempt to improve timing. This is bad for power, congestion and utilization, but may in some cases improve clock gate enable timing. The `clock_gate_buffering_location` property controls whether CCOpt always clones (if it is "above"), never clones (if it is "below"), or clones as appropriate to automatically trade power for timing (if it is "auto").

Set the global property to disable clock gate cloning for all clock trees:

```
set_ccopt_property clone_clock_gates false
```

Set the per-clock tree property to disable clock gate cloning for a particular clock tree:

```
set_ccopt_property clone_clock_gates false -clock_tree clk
```

**Valid values:** true false

**See also:**

- `clock_gate_buffering_location`
- `clone_clock_logic`

**Default:** false

**Optional applicable arguments:** "-clock\_tree <name>".

## clone\_clock\_logic

Specifies whether CCOpt should clone clock logic in an attempt to improve timing. This is bad for power, congestion and utilization, but may in some cases improve clock gate enable timing. The `clock_gate_buffering_location` property controls whether CCOpt always clones (if it is "above"), never clones (if it is "below"), or clones as appropriate to automatically trade power for timing (if it is "auto").

Set the global property to disable clock logic cloning for all clock trees:

```
set_ccopt_property clone_clock_logic false
```

Set the per-clock tree property to disable clock logic cloning for a particular clock tree:

```
set_ccopt_property clone_clock_logic false -clock_tree clk
```

Valid values: true false

See also:

- `clock_gate_buffering_location`
- `clone_clock_gates`

Default: false

Optional applicable arguments: "-clock\_tree <name>".

## compatibility\_warning

Compatibility warning.

Valid values: true false

Default: true

This global property does not use additional arguments.

## consider\_em\_constraints

This property specifies the current waveform calculation methods when considering EM constraints. One or more of the three methods rms, peak, and avg can be specified with a space between them:

1. rms means to check root-mean-square current limit.
2. peak means to check peak current limit.
3. avg means to check average current limit.

No or incorrect method means NOT considering EM constraints.

If EM constraints is considering, CCOpt will try to find the max load capacitance without violating EM constraints for all userable buffer/inverter/clockgate cells specified by corresponding properties and then set `target_max_capacitance` property for these cells.

Valid values: rms peak avg

Default: {}

This global property does not use additional arguments.

## **consider\_power\_management**

Consider power management.

**Valid values:** true false

**Default:** true

This global property does not use additional arguments.

## **constraints**

A list of CCOpt components which should be constrained by this skew group. If this list includes "icts" or "ccopt\_initial", this skew group will constrain sinks during "ccopt\_design -cts" and during the initial global balancing step of a regular "ccopt\_design" run. "icts" and "ccopt\_initial" are synonymous. If this list includes ccopt or all, this skew group will constrain both "ccopt\_design -cts" and the whole of "ccopt\_design" - not just the initial solution. "ccopt" and "all" are synonymous.

The special value "none" is a synonym for an empty list, and specifies that the skew group will only be used for reporting purposes.

**Valid values:** icts ccopt\_initial ccopt all none

**Default:** {ccopt\_initial icts}

Optional applicable arguments: "-skew\_group <name>".

## **cts\_edge\_sensitivity**

The clock edge the pin should be considered sensitive to. This may be set to 'rise', 'fall', 'both', or 'auto' being determined from the timing models for the pin. This property has no effect unless active-edge balancing is being used (that is, unless balance\_edge is set to 'active').

**Valid values:** rise fall both auto

**See also:**

. balance\_edge

**Default:** auto

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **cts\_isolated**

Indicates whether this is an isolated pin. CTS does not cluster isolated pins with any other pins.

**Valid values:** true false

**Default:** false

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **cts\_merge\_clock\_gates**

If set to true, clock gate merging is enabled. If this is false, merging of all clock gates is disabled, including clock gates which may have been cloned by CTS.

Note that this property only impacts 'ccopt\_design -cts'. See also property

`ccopt_merge_clock_gates`.

**Valid values:** true false

**Default:** false

Optional applicable arguments: "-clock\_tree <name>".

## **cts\_merge\_clock\_logic**

If set to true, clock logic merging is enabled. If this is false, merging of all clock logics is disabled, including clock logics which may have been cloned by CTS.

Note that this property only impacts 'ccopt\_design -cts'. See also property `ccopt_merge_clock_logic`.

**Valid values:** true false

**Default:** false

Optional applicable arguments: "-clock\_tree <name>".

## **debug\_stage\_delays\_for\_nodes**

Internal property to temporarily increase clocktree/timing/stagedelays.cpp debug when an update for any of the given clock node uids is performed.

**Default:** {}

This global property does not use additional arguments.

## **def\_lock\_clock\_sinks\_after\_routing**

If set to `true`, we will DEF lock clock tree sinks after routing in addition to any clock node locking (fixed).

If set to `soft`, we will DEF lock clock tree sinks after routing in addition to any clock node soft\_locking (softFixed).

**Valid values:** true false soft

**Default:** false

This global property does not use additional arguments.

## delay\_cells

A Tcl list of symmetric delay cells. Clock tree synthesis uses instances of library cells from this list during processing. If list is empty, clock tree synthesis will use any delay cells it detects in the library.

Set the global property to specify delay cells for all clock trees and power domains:

```
set_ccopt_property delay_cells {delayA delayB delayC}
```

Set the per-clock tree property to specify delay cells for a particular clock tree and ALL power domains:

```
set_ccopt_property delay_cells {delayX delayY} -clock_tree clk
```

Set the per-clock tree /power domain property to specify delay cells for a particular clock tree and power domain:

```
set_ccopt_property delay_cells {delayX delayY} -clock_tree clk -power_domain pd
```

Valid values: list lib\_cell

Default: {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## detailed\_cell\_warnings

If set to true, CCOpt outputs detailed cell warning diagnostics when it encounters issues with library cell selection, power domains and/or signal levels.

Valid values: true false

Default: false

This global property does not use additional arguments.

## effective\_clock\_period

Clock period annotations are used to calculate the `effective_clock_period` property for a pin. The `effective_clock_period` is the minimum clock period present at that pin (corresponding to the highest frequency clock).

The `effective_clock_period` property is read-only, and is calculated from the `clock_period` property on pins. To change the `effective_clock_period` for a pin set the `clock_period` property (either for the pin itself, or the clock root above the pin). A value of 'auto' for this property indicates either that the pin is not a clock pin, or that no `clock_period` annotations were found for the pin.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## effective\_clock\_period\_sources

This property lists the set of pins that define the `effective_clock_period` at a given pin (i.e. which `clock_period` property settings defined this pin's `effective_clock_period`).

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **effective\_sink\_type**

Indicates how CCOpt will treat a given pin, taking into account both its `implicit_sink_type`, and any `sink_type` settings. Setting a non-default value for the `sink_type` property will override the `implicit_sink_type` property.

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **effort**

Amount of effort to use in the H-tree synthesis. By default, medium effort is used.

Valid values: `high`, `medium`, `low`

*Default:* `medium`

Optional applicable arguments: "-flexible\_htree <name>".

## **enable\_all\_views\_for\_io\_latency\_update**

If set, enables all views before updating IO latencies, and restores the set of enabled views after.

Valid values: `true` `false`

*Default:* `true`

This global property does not use additional arguments.

## **enable\_datapoints**

Enable datapoints.

Type: `boolean`

*Default:* `false`

Current Value: `false`

This global property does not use additional arguments.

## **enable\_initial\_clustering**

Controls whether `ccopt_design` should do initial clustering.

**Valid values:** true false

**Default:** true

This global property does not use additional arguments.

## **enable\_locked\_node\_check\_failure**

Before running CCOpt, a check is made to see how many cells are DEF or user locked. If the number of clock tree cells locked in this manner is close to the total number of clock tree cells in the design then the check is considered to have failed and an error message is emitted. If the `enable_locked_node_check_failure` property is set to `true` then CCOpt will not continue running.

If the property is set to `false` then the error message is still emitted but CCOpt will continue to run.

**Valid values:** true false

**Default:** true

This global property does not use additional arguments.

## **engine\_implemented**

Indicates whether CCOpt has done implementation.

**Valid values:** true false

**Default:** false

This global property does not use additional arguments.

## **error\_on\_problematic\_slew\_violating\_nets**

Post slew-fixing, CCOpt will identify nets with slew violations that probably can not be fixed and return an error. Set the property to `false` to bypass this error. Common problems include setting a net that requires buffering as constant. Review your design carefully before setting this property, as it may lead to poor results.

**Type:** boolean

**Default:** false

**Current Value:** false

This global property does not use additional arguments.

## **error\_on\_problematic\_slew\_violating\_nets\_max\_printout**

Early on on the CCOpt flow nets that are marked as e.g. don't touch but have violations are identified. Set this property to control how many nets with each kind of violation will be printed. Setting the property to -1 will print all violating nets.

**Type:** integer

*Default:* 5

Current Value: 5

This global property does not use additional arguments.

## **exclusive\_sinks\_rank**

The rank of this skew\_group. Sinks will be considered as a sink of all skew groups with a rank equal to the greatest rank of all skew groups of which they are defined as sinks.

Valid values: integer

*Default:* 0

Optional applicable arguments: "-skew\_group <name>".

## **expand\_multi\_child\_regions**

When set to true, this property tells the clustering phase of CCOpt to expand regions of clock nodes, even if they drive more than a single item of fanout. This can help to reduce insertion delay.

Valid values: true false

*Default:* false

Optional applicable arguments: "-clock\_tree <name>".

## **extract\_balance\_multi\_source\_clocks**

Causes [create\\_ccopt\\_clock\\_tree\\_spec](#) to set up a clock tree source group for SDC clocks with multiple source pins. If this property is set to true, [create\\_ccopt\\_clock\\_tree\\_spec](#) defines one clock tree for each source pin and then uses the [create\\_ccopt\\_clock\\_tree\\_source\\_group](#) command to collect those clock trees together, so that CTS can distribute sinks between the clock trees.

Type: boolean

*Default:* false

Current Value: false

This global property does not use additional arguments.

## **extract\_check\_timing\_slacks**

When set, this property causes [create\\_ccopt\\_clock\\_tree\\_spec](#) to only consider pins with a valid timing slack to be contained within the clock trees. Any pin without valid timing slack will be identified as an ignore pin.

Type: boolean

*Default:* false

**Current Value:** false

This global property does not use additional arguments.

## **extract\_clock\_generator\_skew\_group\_name\_prefix**

This property controls the skew group name prefix used for skew groups generated to balance generator flops with their adjacent flops. Default is "\_clock\_gen". The default has a start underscore at the beginning to cause listings of skew groups ordered by name to collect such skew groups together at the end of a list.

**Type:** string

*Default:* \_clock\_gen

**Current Value:** \_clock\_gen

This global property does not use additional arguments.

## **extract\_clock\_generator\_skew\_groups**

This property will cause the `create_ccopt_clock_tree_spec` command to create skew groups for sequential generators and their adjacent registers. Such skew groups will be specified with the same highest rank so that they can be balanced from the other normal skew groups that share some sinks of them. The adjacent registers of a generator are registers that have a datapath timing path to talk with the generator directly. When this property is set to `true`, one skew group will be created per sequential generator instance, master clock and generated clock tree triple. The resulting skew groups will by default be named in the pattern:

`_clock_gen_<master_clock_name>_<generator_local_name><_optional_number>/<constraint_mode_name>.`

For example, for a pair of generators, with the same local name "reg\_clkgen", CCOpt creates generated clock trees from the same master clock named "fclk" in a constraint mode named "func" the skew groups emitted into the clock tree specification file would be named:

`_clock_gen_fclk_reg_clkgen_1/func`  
`_clock_gen_fclk_reg_clkgen_2/func`

The prefix for the names of such skew groups is controlled by the `extract_clock_generator_skew_group_name_prefix` CCOpt property and defaults to "`_clock_gen`" (the start underscore is used to group such skew groups at the end of any skew group listing ordered by name).

**Type:** boolean

*Default:* true

**Current Value:** true

This global property does not use additional arguments.

## **extract\_clock\_node\_timing\_endpoints**

A list of the pins on internal clock nodes that are timing endpoints in the clock and timing\_config the skew group corresponds to. This list is generated when clock trees are extracted, and is not used directly by CCOpt.

**Valid values:** list pin

**Default:** {}

Optional applicable arguments: "-skew\_group <name>".

## **extract\_create\_explicit\_ignores\_for\_implicit\_excludes\_beyond\_sta\_clock\_network**

If true, then clock tree extraction will create explicit ignore pins for pins which have implicit\_sink\_type exclude and which are at the boundary of the STA clock network. If false, then clock tree extraction will create explicit exclude pins for these pins.

**Default:** true

This global property does not use additional arguments.

## **extract\_cts\_case\_analysis**

During `create_ccopt_clock_tree_spec`, set case\_analysis properties for any clock tree cell inputs determined to have constant values. CTS considers timing through cells according to the case\_analysis properties of the cell's input pins. If this property is set to false, `create_ccopt_clock_tree_spec` will not set any case\_analysis properties and CTS will assume worst-case timing.

**Type:** boolean

**Default:** true

**Current Value:** true

This global property does not use additional arguments.

## **extract\_faster\_sdc\_clocks\_as\_clock\_trees**

If set to true, this property causes `create_ccopt_clock_tree_spec` to create a separate clock tree for SDC clocks that are faster than the parent clock.

**Type:** boolean

**Default:** true

**Current Value:** true

This global property does not use additional arguments.

## **extract\_merge\_identical\_skew\_groups**

Causes `create_ccopt_clock_tree_spec` to merge `skew_groups` from different `timing_configs` that are otherwise identical.

**Type:** boolean

**Default:** false

**Current Value:** false

This global property does not use additional arguments.

## **extract\_network\_latency**

When `true`, this property causes `create_ccopt_clock_tree_spec` to use the clock network latency as the insertion delay target for the representative clock tree. An SDC command of the form `set_clock_latency <clock> <delay>`; will result in the `create_ccopt_clock_tree_spec` command setting the `-target_insertion_delay` on the corresponding `create_ccopt_skew_group` command.

**Type:** boolean

**Default:** true

**Current Value:** true

This global property does not use additional arguments.

## **extract\_no\_exclude\_pins**

When the clock tree extraction algorithm finds a pin on the clock tree that has no outgoing timing arcs (for example a design output, black box input, or flop D input), it will normally exclude that pin from the clock tree. The pin will still be clocked, but the clock tree will include as a sink a point higher up in the clocking structure that drives this excluded pin, rather than the pin itself. If this property is `true`, the clock tree will instead include as sinks all pins with no outgoing timing arcs, which may lead to the extraction algorithm following convoluted paths in the clock tree with the subsequent effect that CCOpt will attempt to meet the clock tree's slew target on these paths.

**Type:** boolean

**Default:** false

**Current Value:** false

This global property does not use additional arguments.

## **extract\_skew\_group\_sinks\_at\_clock\_node\_timing\_endpoints**

If set, `create_ccopt_clock_tree_spec` will emit `skew_group` sinks at clock node inputs which are timing endpoints for a SDC clock. If not set, then `create_ccopt_clock_tree_spec` will emit `skew_group` ignores at such inputs and they will not be considered for balancing. Typical examples of clock node timing endpoints include flops that are used as generators for one clock (typically the functional clock) but are strictly sinks for another clock (typically a scan clock). Usually such endpoints are fine to be balanced as skew group ignore pins since there are normally relatively few of them and they have loose timing constraints. In addition relatively few designs have the correct logic structures above such timing endpoints to balance them correctly as sinks, hence the default value of `false`.

Type: boolean

*Default:* false

Current Value: false

This global property does not use additional arguments.

## **extract\_source\_latency**

When `true`, this property causes `create_ccopt_clock_tree_spec` to use the clock source latency as the source delay for the representative clock tree. An SDC command of the form `set_clock_latency -source <clock> <delay>` will result in the `create_ccopt_clock_tree_spec` command setting the `source_latency` property.

Type: boolean

*Default:* true

Current Value: true

This global property does not use additional arguments.

## **extract\_through\_multi\_output\_cells\_with\_single\_clock\_output**

When `true`, this property causes `create_ccopt_clock_tree_spec` to extract through cells that have multiple outputs but where only one of those outputs has a clock emanating from it. In such cases, the input is marked as a through pin to the relevant output that carries the clock signal. When `false`, `create_ccopt_clock_tree_spec` stops at multi output cells.

Type: boolean

*Default:* false

Current Value: false

This global property does not use additional arguments.

## **extract\_through\_to**

Clock tree definition will, by default, not continue through certain types of cell arc (for instance, the clock to Q arc in a DFF). This property allows you to override the default behavior by specifying the output pin to which the clock tree should propagate, when it arrives at a given input pin.

**Valid values:** pin

**Default:** {}

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **extracted\_from\_clock\_name**

This contains the name of the SDC clock that this skew group has been created to represent the balancing constraints in CCOpt.

**Valid values:** string

**Default:** {}

Optional applicable arguments: "-skew\_group <name>".

## **extracted\_from\_constraint\_mode\_name**

This contains the name of the constraint mode that this skew\_group has been created to represent the balancing constraints in CCOpt.

**Valid values:** string

**Default:** {}

Optional applicable arguments: "-skew\_group <name>".

## **extracted\_from\_delay\_corners**

This contains the delay corners associated with this skew\_group that has been created to represent the balancing constraints in CTS.

**Valid values:** string

**Default:** {}

Optional applicable arguments: "-skew\_group <name>".

## **extract\_pin\_insertion\_delays**

If set to `true`, `create_ccopt_clock_tree_spec` will extract pin insertion delay settings from SDC `set_clock_latency` assertions on clock sinks, if the clock network latency specified for the sink differs from the network latency specified for the corresponding clock.

**Type:** boolean

**Default:** true

**Current Value:** true

This global property does not use additional arguments.

## **fastest\_buffer**

This property tells the clustering phase of CCOpt which library buffer cell it should consider to be the fastest, when prioritizing insertion delay. If not set, CCOpt will automatically calculate this value. Any clock tree and power context combination that does not have this buffer available for use will revert to the standard calculation of fastest buffer cell.

**Valid values:** lib\_cell

**Default:** {}

This global property does not use additional arguments.

## **fastest\_buffer\_max\_trans**

This property tells the clustering phase of CCOpt what slew value will provide fastest insertion delay, when prioritizing insertion delay. If not set, CCOpt will automatically calculate this value.

**Valid values:** double

**Default:** auto

This global property does not use additional arguments.

## **fastest\_inverter**

This property tells the clustering phase of CCOpt which library inverter cell it should consider to be the fastest, when prioritizing insertion delay. If not set, CCOpt will automatically calculate this value.

Any clock tree and power context combination that does not have this inverter available for use will revert to the standard calculation of fastest inverter cell.

**Valid values:** lib\_cell

**Default:** {}

This global property does not use additional arguments.

## **fastest\_inverter\_max\_trans**

This property tells the clustering phase of CCOpt what slew value will provide fastest insertion delay, when prioritizing insertion delay. If not set, CCOpt will automatically calculate this value.

*Valid values:* double

*Default:* auto

This global property does not use additional arguments.

## **final\_cell**

The library cell to use for the H-tree sinks.

*Valid values:* string

*Default:* {}

Optional applicable arguments: "-flexible\_htree <name>".

## **flexible\_htree\_max\_synthesis\_grid\_points**

The maximum number of grid points allowed in a synthesis grid. Grid will be coarsened until this limit is satisfied.

*Default:* 50000

This global property does not use additional arguments.

## **force\_all\_module\_boundaries\_dont\_touch**

When this property is true, the terminals of all hierarchical modules in the design are treated as if they are marked don't touch, and so preserved throughout CTS. This has no effect on non-CCOpt optimizations.

*Valid values:* true false

*Default:* false

This global property does not use additional arguments.

## **force\_all\_virtual\_delay\_updates**

If this is set to true, any call to set or reset a virtual delay, even if that call doesn't change the stored value, will push through into the timing engine. Normally we'll avoid calling into CTE if the annotation is left unchanged - saves runtime.

*Valid values:* true false

*Default:* false

This global property does not use additional arguments.

## **force\_clock\_objects\_to\_propagated**

Determine whether to put the propagated\_clock assertion on all clocks within ccopt\_design.

Valid values: true false

*Default:* true

This global property does not use additional arguments.

## **force\_clock\_topology\_changed\_after\_adjustment**

Force a full timing update after each batch of adjustments.

Valid values: true false

*Default:* false

This global property does not use additional arguments.

## **force\_clock\_tree**

Can be used to make sure a clock tree sink pin gets driven by a particular clock tree. The clock tree can alternatively be specified as the tree's root pin, in case extraction renames the clock trees defined from a single SDC clock. By default CCOpt automatically selects appropriate pins.

Valid values: clock\_tree | list of pins

*Default:* {}

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **force\_cluster\_only**

Forces the CTS algorithm to avoid doing any fine-grained balancing of the clock tree, and instead makes it terminate after the clustering code has finished. This is not useful unless you are debugging the CTS clustering code.

Type: boolean

Default: false

Current Value: false

This global property does not use additional arguments.

## **force\_update\_io\_latency**

If false (default), I/O latencies are updated if and only if all SDC clocks are in ideal mode. If true, run I/O latency updates even when clocks are in propagated mode.

Valid values: true false

**Default:** false

This global property does not use additional arguments.

## generated\_by\_sinks

The list of parent clock tree sinks for which timing to this clock tree should be considered, if any.

**Valid values:** list pin

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-clock\_tree <name>".

## grid\_step

The grid step used in flexible H-tree synthesis.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-flexible\_htree <name>".

## hard

A hard clock spine moves other cells out of its defined rectangle, by creating a fence placement region for that rectangle. It does this in order to make space for cells that are assigned to the clock spine in question by CTS. If the clock spine is not marked as hard then CTS will not move other cells out of the spine regions and clock spine assignment is more likely to run out of space when assigning cells to clock spines.

Property value is 1 if the clock spine is hard and 0 if not.

**Valid values:** true false.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-clock\_spine <name>".

## htree\_sinks

Specifies H-tree sinks as approximate rectangular areas for locations of final cells (given by -final\_cell) or pins to wire to.

**Valid values:** list {pin | {xmin ymin xmax ymax}}

**Default:** {}

Optional applicable arguments: "-flexible\_htree <name>".

## hv\_balance

Specifies whether horizontal and vertical wires can only be balanced against other wires of the same orientation (true), or whether any wire can be balanced against any other wire (false).

**Valid values:** true false

**Default:** true

Optional applicable arguments: "-flexible\_htree <name>".

## ignore\_pins

A list of ignore pins for this skew group.

**Valid values:** list pin

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-skew\_group <name>".

## implicit\_sink\_type

Indicates the type of sink that CCOpt classified this pin as. Note that the sink\_type property can override these internal classifications.

Possible values are:

`exclude` Indicates that this pin is a sink which represents a non-clock pin.

`ignore` Indicates that CCOpt has determined not to search for more clock tree through this pin. Additionally, this pin will not be balanced.

`stop` Indicates that CCOpt has determined not to search for more clock tree through this pin.

An empty value for this property indicates either that this pin is either not a sink, or that it is a sink that is not implicitly exclude or ignore or stop.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## include\_source\_latency

Specifies whether clock tree source latency should be included when timing the skew group.

**Valid values:** true false

**Default:** false

Optional applicable arguments: "-skew\_group <name>".

## insertion\_delay

The insertion delay under this pin.

**Valid values:** double | auto

**Default:** auto

**Applicable arguments:** "-delay\_corner <name>", "-pin <name>", "-early", "-late", "-rise", "-fall", "-max" and "-min".

**Required:** "-pin <name>".

## insertion\_delay\_wire

The wire component of the insertion delay under this pin.

**Valid values:** double | auto

**Default:** auto

**Applicable arguments:** "-delay\_corner <name>", "-pin <name>", "-early", "-late", "-rise", "-fall", "-max" and "-min".

**Required:** "-pin <name>".

## inverter\_cells

A Tcl list of symmetric inverter cells. All inverters created in the clock tree and power domain will be instances of these cells.

Set the global property to specify inverter cells for all clock trees and all power domains:

```
set_ccopt_property inverter_cells {invA invB invC}
```

Set the per-clock tree property to specify inverter cells for a particular clock tree and all power domains:

```
set_ccopt_property inverter_cells {invX invY} -clock_tree clk
```

Set the per-clock tree/power domain property to specify inverter cells for a particular clock tree and power domain:

```
set_ccopt_property inverter_cells {invX invY} -clock_tree clk -power_domain pd
```

By default CCOpt automatically selects appropriate cells.

**Valid values:** list lib\_cell

**Default:** {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## is\_user\_skew\_group

If set this skew group was created by the user.

**Valid values:** true false

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-skew\_group <name>".

## **is\_vertical**

Indicates whether a clock spine is defined as being vertical. Property value is 1 if the clock spine is vertical and 0 if not.

Valid values: true false.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-clock\_spine <name>".

## **last\_target\_insertion\_delay**

The previous insertion delay target used for clock tree synthesis, for this skew group.

Valid values: double | auto

Default: auto

Optional applicable arguments: "-skew\_group <name>".

## **last\_target\_insertion\_delay\_wire**

The previous wire insertion delay target used for clock tree synthesis, for this skew group.

Valid values: double | auto

Default: auto

Optional applicable arguments: "-skew\_group <name>".

## **last\_target\_max\_trans**

Indicates the slew time target that was used the last time this clock tree was balanced.

Valid values: default | auto | ignore | double

Default: default

Optional applicable arguments: "-delay\_corner <name>", "-clock\_tree <name>", "-net\_type <name>", "-early" and "-late".

## **last\_target\_skew**

Indicates the skew target that was used the last time this `skew_group` was balanced.

Valid values: double

**Default:** default

Optional applicable arguments: "-skew\_group <name>", "-delay\_corner <name>", "-early" and "-late".

## **last\_target\_skew\_wire**

Indicates the wire skew target that was used the last time this skew\_group was balanced.

**Valid values:** double

**Default:** default

Optional applicable arguments: "-skew\_group <name>", "-delay\_corner <name>", "-early" and "-late".

## **leaf\_buffer\_cells**

A Tcl list of buffer cells. All leaf routed clock tree nets will be driven by instances of these cells.

**Valid values:** list cell

**Default:** {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## **leaf\_inverter\_cells**

A Tcl list of inverter cells. All leaf routed clock tree nets will be driven by instances of these cells.

**Valid values:** list cell

**Default:** {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## **legalized\_on\_clock\_spine**

The name of the clock spine to which the specified pin has been assigned by CCOpt, if any. Pins which have been aligned with a clock spine report the name of that spine in this property. Clock spine cell placement inside CCOpt CTS will record the clock spine that it chose to place the pin on (or near, in the case of placement failure).

When setting the property if a clock spine name is specified which does not correspond to an existing clock spine then the command will fail.

When setting the property if a clock spine name is specified which is not in the corresponding list in the lock\_on\_clock\_spine property then the clock spine name will be added to that property as well.

**Valid values:** string

**Default:** {}

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## lock\_on\_clock\_spine

A set of clock spine names restricting where the pin should be located. If the set contains more than one item, the pin may be placed on any of the specified clock spines. Clock tree nets above pins locked on a clock spine pins will automatically be routed as top nets.

When setting the property if a clock spine name is specified which does not correspond to an existing clock spine then the command will fail.

When setting the property if the list of clock spine names no longer contains the clock spine referred to by the property `legalized_on_clock_spine` for this pin then that property will be unset.

**Valid values:** list string

**Default:** {}

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## locked\_originally

CCOpt will not touch insts that were locked previously. This property is set on insts detected as originally locked.

**Valid values:** true false soft

**Default:** false

**Applicable arguments:** "-inst <name>". **Required:** "-inst <name>".

## log\_precision

The number of significant figures printed in floating point numbers written to the log

**Type:** integer

**Default:** 3

**Current Value:** 3

This global property does not use additional arguments.

## log\_special\_case\_cell\_selections

When specified, CCOpt will provide additional log information on the lists of cells that it will consider using when determining if a cell can be resized. This information is only provided for the more "specialized" cases.

Specialized cases include logic cells and any cells marked (either by the user or by an internal constraint) as dont touch.

**Valid values:** true false

**Default:** false

This global property does not use additional arguments.

## **logic\_cells**

A Tcl list of clock logic cells. Clock tree synthesis makes all clock logic cells in the clock tree/power domain that are members of the relevant families instances of these cells. Families with no specified cells still have their cells selected automatically. If not set, clock tree synthesis picks clock logic cells from the library automatically, respecting the don't use settings. If given explicitly, this property overrides the don't use settings.

By default CCOpt automatically selects appropriate cells.

**Valid values:** list lib\_cell

**Default:** {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## **low\_power\_clustering**

When set, this property enables routines in the clustering phase of CTS which are designed to reduce clock tree power.

**Valid values:** true false

**Default:** false

Optional applicable arguments: "-clock\_tree <name>".

## **manage\_power\_management\_illegalities**

If this property is set, the CTS algorithm will work around power management illegalities in the clock tree, as opposed to failing with an error when it encounters them. This allows the clock tree to be synthesized, but any power management illegalities will remain in the exported design.

**Valid values:** true false

**Default:** true

This global property does not use additional arguments.

## **max\_buffer\_depth**

Constrains CTS to ensure that at most this many buffers are present along each path from source to sink in the skew group.

**Default:** auto

Optional applicable arguments: "-skew\_group <name>".

## **max\_clock\_cell\_count**

The maximum number of clock cells taken from a library before a warning is printed. Parts of CCOpt can exhibit poor runtime performance when the list of possible library cells to consider becomes too large. A warning is printed when the list of cells exceeds the limit set in this property. It may indicate that the number of cells specified should be reduced. If done properly, this will have little effect on results.

Valid values: Any positive number or 0 to disable.

Type: integer

*Default:* 15

Current Value: 15

This global property does not use additional arguments.

## **max\_delta\_band\_factor**

Determines the maximum allowed adjustment as a multiple of band size.

Valid values: double

*Default:* 2

This global property does not use additional arguments.

## **max\_delta\_slack\_factor**

Determines the maximum allowed adjustment as a multiple of negative slack.

Valid values: double

*Default:* 1

This global property does not use additional arguments.

## **max\_fanout**

The maximum fanout at any point in the clock tree.

Valid values: integer ranged between 2 and 1000 inclusive

*Default:* 100

This global property does not use additional arguments.

## **maximum\_insertion\_delay**

For instances in the clock tree, specifies a maximum desired insertion delay beneath that instance. For instances not in the clock tree, this property has no effect.

Because it relies on the existence of the clock tree, this property can only be set after clock trees have been created.

**Valid values:** double

**Default:** -1.79769e+308

Applicable arguments: "-inst <name>". Required: "-inst <name>".

## **max\_skew\_passes\_with\_insufficient\_progress**

Determines the number of skew adjustment passes allowed whilst not making (enough) forward progress on WNS.

**Valid values:** double

**Default:** 3

This global property does not use additional arguments.

## **max\_source\_to\_sink\_net\_length**

The maximum routing length in microns between driving source pin and driven sink pin on each net that clock tree synthesis should observe. This constraint can be applied to either a pin, a clock tree, or a net type. By default (if this property is not set) no explicit clock tree net length constraint is enforced. However, other clock tree constraints such as maximum slew (transition) and maximum capacitance will indirectly limit the maximum net length.

**Valid values:** double

**Default:** auto

Optional applicable arguments: "-clock\_tree <name>", "-lib\_pin <name>" and "-net\_type <name>".

## **max\_source\_to\_sink\_net\_resistance**

The maximum routing resistance (in resistance library units) between source and sink that clock tree synthesis should observe.

**Valid values:** double

**Default:** auto

This global property does not use additional arguments.

## **min\_delta\_band\_factor**

Determines the minimum allowed adjustment as a multiple of band size.

**Valid values:** double

**Default:** 0

This global property does not use additional arguments.

## **move\_clock\_gates**

If this property is set, the CTS algorithm will move clock gates that appear in the clock tree. 'Logic' does not include clock gates, buffers, and inverters in a clock tree, which are always moved unless they are locked, or clock generators that are above the root of the clock tree. Usually, this will affect multiplexers used for selecting one of a number of clocks, or for switching between a test clock and the main clock. Setting this property may cause the clock tree to have a lower insertion delay, but might break datapath timing. During optimization, this isn't a problem, because the timing will be automatically recovered during the optimization process.

Type: boolean

*Default:* true

Current Value: true

This global property does not use additional arguments.

## **move\_clock\_nodes\_for\_slack**

When set, CCOpt will consider moving nodes in the clock tree with poor timing in an attempt to reduce negative slack. This property can be set globally or per-clock tree.

Valid values: true false

*Default:* false

Optional applicable arguments: "-clock\_tree <name>".

## **move\_logic**

If this property is set, the CTS algorithm will move logic that appears in the clock tree. "Logic" does not include clock gates, buffers, and inverters in a clock tree, which are always moved unless they are locked, or clock generators that are above the root of the clock tree. Usually, this will affect multiplexers used for selecting one of a number of clocks, or for switching between a test clock and the main clock. Setting this property may cause the clock tree to have a lower insertion delay, but might break datapath timing. During optimization, this isn't a problem, because the timing will be automatically recovered during the optimization process.

Valid values: true false

*Default:* true

This global property does not use additional arguments.

## **move\_middle\_cell\_first\_when\_adding\_wire\_delay**

When moving three cells to add wire delay, move middle cell first, followed by the other added cell and then lastly the parent or child of the middle cell.

*Default:* false

This global property does not use additional arguments.

## **name\_prefix**

The name prefix of instances/nets created by CTS. The default value is "CTS". The default names of nets are CTS, CTS\_1, CTS\_2... ...The default names of instances are CTS\_\*.

**Valid values:** string

**Default:** CTS

This global property does not use additional arguments.

## **net\_unbufferable**

This property contains a list of reasons why CCOpt was not able to buffer the clock net attached to the specified pin.

**Valid values:** string

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## **no\_symmetry\_buffers**

If specified, do not add symmetry buffers to balance the pin load of nets in the H-tree.

**Default:** false

**Optional applicable arguments:** "-flexible\_htree <name>".

## **opt\_ignore**

Specifies whether CCOpt will balance this clock tree. If set to true, CCOpt will not balance or optimize this clock tree. The default is false.

**Valid values:** true false

**Default:** false

**Optional applicable arguments:** "-clock\_tree <name>".

## **optimize\_ir\_drop\_skew\_amount**

Save the skewing amount of clock tree sink pin during the ir drop optimization.

**Valid values:** double

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## **override\_minimum\_max\_trans\_target**

Specifies that CCOpt should allow any slew target to be used, even if it is likely to lead to runtime problems. By default, CCOpt computes a minimum slew target which should not lead to runtime problems, and does not allow any slew target to be set lower than this minimum. This property overrides that check, and allows any slew target to be used.

*Valid values:* true false

*Default:* false

This global property does not use additional arguments.

## **override\_minimum\_skew\_target**

Specifies that CCOpt should allow any skew target to be used, even if it is likely to lead to poor results. By default, CCOpt computes a minimum skew target which should not lead to excessive buffering, and does not allow any skew target to be set lower than this minimum. This property overrides that check, and allows any skew target to be used.

*Valid values:* true false

*Default:* false

This global property does not use additional arguments.

## **override\_vias**

When specified, this property defines the vias to be used in RC extraction from routing estimates. The listed vias will be used in place of those configured on the routing rules for the clock network. The order of the list is irrelevant. The vias may also be overridden for each non-default rule (NDR). In this case, the name of the NDR is given as the first element in the list with subsequent entries being the via names. Multiple NDRs can be specified, e.g. {via1d {NDR1 via2d} {NDR2 via2d}} will override via1d in the default rule, and both via1d and via2d in NDR1 and NDR2.

*Valid values:* list via\_call

*Default:* {}

This global property does not use additional arguments.

## **override\_zero\_placeable\_area**

If set CCOpt will allow CTS to run on a design which has zero placeable area. Setting this property may be useful to temporarily work-around problems with row definition and/or blockages causing placeable area to be zero.

*Valid values:* true false

*Default:* false

This global property does not use additional arguments.

## parents

The list of parent clock trees from which this clock tree is generated, if any.

**Valid values:** `list clock_tree`

**Read-only:** This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

**Optional applicable arguments:** `"-clock_tree <name>"`.

## pin

The pin under which the H-tree is created. This pin must be part of a clock tree at the time of synthesis.

**Valid values:** `pin`

**Default:** `{}`

**Optional applicable arguments:** `"-flexible_htree <name>"`.

## pin\_target\_max\_trans

The per-pin target slew used for clock tree synthesis. This overrides any target set by `target_max_trans`. If set to auto (the default) the transition target used for optimization falls back the `clock_tree`, global target, `liberty max_transition` or the value of `target_max_trans_sdc`.

**Valid values:** `double` or `auto`

**Default:** `auto`

**Applicable arguments:** `"-delay_corner <name>"`, `"-pin <name>"`, `"-early"` and `"-late"`. Required: `"-pin <name>"`.

## post\_conditioning

Enable post-conditioning optimization after clock nets are routed.

**Valid values:** `true` `false`

**Default:** `true`

This global property does not use additional arguments.

## post\_conditioning\_enable\_drv\_fixing

If set to `false`, post-conditioning will skip its DRV-fixing step.

**Valid values:** `true` `false`

**Default:** `true`

This global property does not use additional arguments.

## **post\_conditioning\_enable\_drv\_fixing\_by\_rebuffering**

If set, post-conditioning will fix DRVs by adding buffers.

Valid values: true false

*Default:* false

This global property does not use additional arguments.

## **post\_conditioning\_enable\_drv\_fixing\_by\_rebuffering\_ccopt**

If set, post-conditioning will fix DRVs by adding buffers during CCOpt.

Valid values: true false

*Default:* false

This global property does not use additional arguments.

## **post\_conditioning\_enable\_skew\_fixing**

If set to `false`, post-conditioning will skip its skew-fixing step.

Valid values: true false

*Default:* false

This global property does not use additional arguments.

## **post\_conditioning\_enable\_skew\_fixing\_by\_rebuffering**

If set to `true`, post-conditioning will attempt skew-fixing using rebuffering.

Valid values: true false

*Default:* false

This global property does not use additional arguments.

## **post\_conditioning\_enable\_skew\_fixing\_by\_rebuffering\_ccopt**

If set to `true`, post-conditioning will attempt skew-fixing using rebuffering during CCOpt.

Valid values: true false

*Default:* false

This global property does not use additional arguments.

## **primary\_delay\_corner**

This specifies the delay corner in which clock tree balancing applies the slew and insertion delay targets. If more than one timing corner is defined, this must be set before running CCOpt. By default, this is set to the first defined delay corner.

**Valid values:** corner name, or empty

**Default:** {}

This global property does not use additional arguments.

## **print\_clock\_tree\_modifications**

Print clock tree modifications.

**Valid values:** internal

**Default:** false

This global property does not use additional arguments.

## **put\_driver\_in\_centre\_of\_fanout\_bounding\_box**

When set, this property tells the CTS clustering code to expand regions of clock nodes, even if they drive more than a single item of fanout.

**Type:** string

**Default:** false

**Current Value:** false

This global property does not use additional arguments.

## **recluster\_to\_reduce\_power**

Enable additional clustering passes in order to reduce clock tree power. This incurs a runtime cost but should improve QoR.

**Valid values:** true false

**Default:** false

Optional applicable arguments: "-clock\_tree <name>".

## **remove\_bufferlike\_clock\_logic**

When set to true, CCOpt will identify and remove clock tree logic that is logically equivalent to buffering.

**Valid values:** true false

*Default:* false

Optional applicable arguments: "-clock\_tree <name>".

## rename\_clock\_tree\_nets

Tells CCOpt to rename all clock tree nets for easy identification later in the flow.

Valid values: true false

*Default:* true

This global property does not use additional arguments.

## report\_only\_skew\_group\_with\_target

The skew groups report (run using the `report_skew_groups` command) displays insertion delay, skew, and min/max path information for different combinations of skew group, timing corner, and early/late path. Set the `report_only_skew_group_with_target` property to `false` (the default) to report on all skew group/timing corner/path combinations regardless of whether a skew target has been set. Set the `report_only_skew_group_with_target` property to `true` to report only on skew group/timing corner/path combinations where a skew target has been set (either explicitly or using the 'auto' setting).

Type: boolean

*Default:* false

Current Value: false

This global property does not use additional arguments.

## report\_only\_timing\_corners\_associated\_with\_skew\_groups

Specifies whether the skew groups report displays all skew groups, in all corners or only corners associated with the timing configs from which the skew group was extracted.

Type: boolean

*Default:* false

Current Value: false

This global property does not use additional arguments.

## resistance\_margin\_absolute

Resistance absolute margin used in CTS clustering. Specified in Ohms.

Valid values: double

*Default:* auto

This global property does not use additional arguments.

## **resistance\_margin\_percentage**

Resistance percentage margin used in CTS clustering.

**Valid values:** double

**Default:** auto

This global property does not use additional arguments.

## **restricted\_cells\_buffers**

This property allows the set of buffer and delay cells that are to be used by CCOpt to be restricted to only allow those specified for the skew group. The special value of "disallow" can be used to disallow the use of buffers and delay cells entirely for the given skew group. If CCOpt needs to insert either a buffer or delay cell that is associated with multiple skew groups, the list of buffer and delay cells available to use is restricted to the intersection of this property across all skew groups that apply.

When using this property and clustering with buffers, it is mandatory to always have the strongest buffer included in the list. If no delay cells are included in this list then no delay cells will get used. If this property is not set or the special value "no\_restriction" is used, it means "allow all buffers and delay cells".

**Valid values:** no\_restriction | list lib\_cell

**Default:** {}

Optional applicable arguments: "-skew\_group <name>".

## **restricted\_cells\_inverters**

This property allows the set of inverters that are to be used by CCOpt to be restricted to only allow those specified for the skew group. The special value of "disallow" can be used to disallow the use of inverters entirely for the given skew group. If CCOpt needs to insert an inverter that is associated with multiple skew groups, the list of inverters available to use is restricted to the intersection of this property across all skew groups that apply.

When using this property and clustering with inverters, it is mandatory to always have the strongest inverter included in the list. If this property is not set or the special value "no\_restriction" is used, it means "allow all inverters".

**Valid values:** no\_restriction | list lib\_cell

**Default:** {}

Optional applicable arguments: "-skew\_group <name>".

## **route\_balancing\_buffers\_with\_default\_rule**

If set CCOpt will always use the default routing rule for balancing buffers.

**Valid values:** true false

**Default:** false

This global property does not use additional arguments.

## route\_type

Route type.

**Valid values:** internal

**Default:** default

Optional applicable arguments: "-clock\_tree <name>" and "-net\_type <name>".

## route\_type\_autotrim

Enable/disable autotrimming for all route types. If set, the allowed range of layers to use for routing may be restricted to a subset of the user-defined range in order to improve performance.

**Valid values:** true false

**Default:** true

This global property does not use additional arguments.

## routing\_override

The value auto means the pin has no routing override (either the user has not set an override, or the pin does not support routing overrides).

The value top means the pin has a routing override in place, forcing the net attached to the pin to be treated as a top net.

The value trunk means the pin has a routing override in place, forcing the net attached to the pin to be treated as a trunk net.

The value leaf means the pin has a routing override in place, forcing the net attached to the pin to be treated as a leaf net.

**Valid values:** auto top trunk leaf

**Default:** auto

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## routing\_top\_fanout\_count

The number of clock sinks this sink counts for when applying the top routing rules. Note that this property is only valid for sink pins, and it returns auto for non-sink pins. For a sink pin, a non-auto value means that this sink is counted as though it were multiple sinks, for the purposes of determining which nets should have top routing. An auto value for a sink pin means that the sink counts as a single sink.

**Valid values:** integer > 0

*Default:* auto

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **routing\_top\_min\_fanout**

Minimum number of transitive fanout in the clock tree for a net to be routed as a top net. Nets with at least this many sinks in their transitive fanout in the clock tree will have the special routing rules applied to them.

Valid values: integer

*Default:* unset

Optional applicable arguments: "-clock\_tree <name>".

## **routing\_top\_transitive\_fanout**

The number of clock sinks in the transitive fanout of the pin as counted for applying the top routing rules. This property is very similar to the `transitive_fanout` property but counts sink fanout using the `routing_top_fanout_count` property instead of always counting sinks as a single item of fanout.

Requesting this property for a pin not in the clock tree will result in an error.

Valid values: integer

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **save\_old\_viz\_on\_cluster**

Save old visualization on cluster.

Valid values: true false

*Default:* false

This global property does not use additional arguments.

## **schedule**

Controls what CCOpt can do with the schedule for this pin.

Allowed values are:

`auto`: Move the sink to make timing better during CCOpt optimization, and to make the clock tree QoR better where possible.

`timing_only`: Move the sink for timing if necessary, but otherwise leave it near the original schedule.

`off`: Leave the sink as close as possible to the original schedule, even if this breaks timing.

**Valid values:** auto timing\_only off

**Default:** auto

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## **sink\_grid**

Specifies the columns and rows of a grid of H-tree sinks.

**Valid values:** {columns rows}

**Default:** {}

**Optional applicable arguments:** "-flexible\_htree <name>".

## **sink\_grid\_box**

The box describing the area that the grid of H-tree sinks should cover. This property only has an effect if the sink\_grid property of the flexible H-tree is set.

**Valid values:** {xmin ymin xmax ymax}

**Default:** {}

**Optional applicable arguments:** "-flexible\_htree <name>".

## **sink\_grid\_exclusion\_zones**

Boxes describing zones that should not be covered by the grid of H-tree sinks. This property only has an effect if the sink\_grid property of the flexible H-tree is set.

**Valid values:** list {xmin ymin xmax ymax}

**Default:** {}

**Optional applicable arguments:** "-flexible\_htree <name>".

## **sink\_grid\_prefix**

Prefix used for inst names of final cells (given by -final\_cell) of H-tree sinks in the grid. The name of the cell will be <prefix>\_<htree\_name>\_<id>, where id is a running index. This property only has an effect if the sink\_grid property of the flexible H-tree is set.

**Valid values:** string

**Default:** {}

**Optional applicable arguments:** "-flexible\_htree <name>".

## **sink\_grid\_sink\_area**

The approximate size of the rectangle describing valid locations for final cells (given by `-final_cell`) per H-tree sink in the grid. This property only has an effect if the `sink_grid` property of the flexible H-tree is set.

**Valid values:** {width height}

**Default:** {}

Optional applicable arguments: "-flexible\_htree <name>".

## **sink\_type**

The type of sink this pin represents.

Valid values are as follows:

`auto`: The pin type will be automatically determined by CCOpt through Through pin. Trace the clock tree through this pin.

`stop`: Stop pin. When defining clock trees, CCOpt stops searching for parts of the clock tree at stop pins.

`ignore`: Ignore pin. CCOpt stops searching for parts of the clock tree at ignore pins and it does not attempt to balance the insertion delay of ignore pins.

`min`: Min pin. Keep the pin at minimal insertion delay.

`exclude`: Exclude pin. Exclude this pin from the clock tree.

**Valid values:** auto through stop ignore min exclude

**Default:** auto

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **sinks**

A list of sinks for this skew group.

**Valid values:** list pin

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-skew\_group <name>".

## **sinks\_active**

Returns the list of active sinks for this skew group. All sink pins that have this skew group in the `skew_groups_active_sink` property are included.

Note that only endpoint sink pins are included. Non-sink pins through which this skew group passes are not included.

**Valid values:** list pin

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-skew\_group <name>".

## **size\_clock\_gates**

When set to true (the default), the CTS algorithm sizes clock gates that appear in the clock tree. 'Logic' does not include clock gates, buffers, and inverters in a clock tree, which are always sized unless they are locked, or clock generators that are above the root of the clock tree. Usually, this affects multiplexers used for selecting one of a number of clocks, or for switching between a test clock and the main clock. Setting this property may cause the clock tree to have a lower insertion delay, but might change the cell types of logic gates in the clock tree, which in turn may require them to be moved slightly to find a legal location for the new cell.

Type: boolean

*Default:* true

*Current Value:* true

This global property does not use additional arguments.

## **size\_logic**

When set to true (the default), the CTS algorithm sizes logic that appears in the clock tree. "Logic" does not include clock gates, buffers, and inverters in a clock tree, which are always sized unless they are locked, or clock generators that are above the root of the clock tree. Usually, this affects multiplexers used for selecting one of a number of clocks, or for switching between a test clock and the main clock. Setting this property may cause the clock tree to have a lower insertion delay, but might change the cell types of logic gates in the clock tree, which in turn may require them to be moved slightly to find a legal location for the new cell.

*Valid values:* true false

*Default:* true

This global property does not use additional arguments.

## **skew\_band\_size**

Internal property to control ccopt\_design flow.

*Valid values:* internal

*Default:* 0.1

This global property does not use additional arguments.

## **skew\_group\_insertion\_delay**

The amount of insertion delay under this pin for a specific skew group. Clock tree synthesis will attempt to make the insertion delay of this pin less than that of other pins in the skew group by this amount. A negative value should be used if you would like the insertion delay of this pin to be greater than that of other pins. The value 'auto' instructs CCOpt to use the value from target\_insertion\_delay. The value 'ignore' instructs CCOpt to put the pin at the insertion delay which is most convenient (attempting to minimize the insertion delay of the pin).

**Valid values:** double | auto

**Default:** auto

**Applicable arguments:** "-skew\_group <name>", "-delay\_corner <name>", "-pin <name>", "-early", "-late", "-rise", "-fall", "-max" and "-min". **Required:** "-pin <name>".

## **skew\_group\_insertion\_delay\_wire**

The wire component of the insertion delay under this pin for a specific skew group.

**Valid values:** double | auto

**Default:** auto

**Applicable arguments:** "-skew\_group <name>", "-delay\_corner <name>", "-pin <name>", "-early", "-late", "-rise", "-fall", "-max" and "-min". **Required:** "-pin <name>".

## **skew\_group\_report\_columns**

A Tcl list of columns to include in skew group reports produced by the skew group report. You can use this property to specify the columns you would like to include in the skew group report, and the order in which the columns should appear. Most of the legal values are straightforward. However, the set of legal values of the following form deserve further explanation:

summaryType\_summaryLocation[\_event]

These values let you report the delay value for other paths that go through the pin.

summaryType is one of: max (show the longest delay), min (show the shortest delay), or skew (show the skew, that is the difference between the longest and the shortest delay).

summaryLocation is one of: above (show the delay/skew above this pin), below (show the delay/skew below this pin), or through (show the delay/skew for paths through this pin).

event is one of: rise (show the delay/skew for the rise event at this pin), fall (show the delay/skew for the fall event at this pin), or both (show the delay/skew for both events at this pin).

**Valid values:**  
capacitance  
distance  
event  
fanout  
increment

```
length
lib_cell
load_capacitance
location
max_above
max_above_fall
max_above_rise
max_below
max_below_fall
max_below_rise
max_through
max_through_fall
max_through_rise
min_above
min_above_fall
min_above_rise
min_below
min_below_fall
min_below_rise
min_through
min_through_fall
min_through_rise
name
net
pin
resistance
skew_above
skew_above_fall
skew_above_rise
skew_below
skew_below_fall
skew_below_rise
skew_through
skew_through_fall
skew_through_rise
slew
status
time
wire_capacitance
```

**Default:** {name lib\_cell event increment time slew capacitance location distance fanout status}

This global property does not use additional arguments.

## **skew\_group\_report\_histogram\_bin\_size**

When set to a numeric value, that numeric value will be used as the histogram range size (in library units). For example, if the library time units are set to 1 nanosecond, a value of 0.010 for report\_skew\_groups\_histogram\_bin\_size will result in histogram ranges of 10 picoseconds.

When set to auto, the size of the histogram ranges are dependent on the skew targets that are set. If a skew target is set for

a given half corner and skew group combination, then the histogram range size will be 10% of the skew target for that half corner and skew target combination. If no skew target is set for a half corner and skew group combination but a skew target is set for the primary half corner and skew group combination, then the histogram range size will be 10% of the skew target for the primary half corner and skew group combination.

In the event that no skew targets are set and report\_skew\_groups\_histogram\_bin\_size is set to auto, a default value of 10 picoseconds will be used for the histogram range size.

**Valid values:** auto | string

**Default:** auto

This global property does not use additional arguments.

## **skew\_groups\_active**

Returns the list of active skew groups for this pin. For sink pins, this property lists both skew groups that pass through this pin and skew groups for which this sink is an endpoint. For non-sink pins, shows skew groups that pass through this pin.

**Valid values:** list skew\_groups

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## **skew\_groups\_active\_sink**

Returns the list of skew groups for which this pin is an active sink. For sink pins, this property lists the skew groups for which this sink is an endpoint. Skew groups that pass through this pin are not included. For non-sink pins, this property always returns null.

**Valid values:** list skew\_groups

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## **skew\_groups\_ignore**

The list of skew groups for which paths through this pin are ignored.

**Valid values:** list skew\_groups

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Applicable arguments:** "-pin <name>". **Required:** "-pin <name>".

## **skew\_groups\_sink**

The list of skew groups for which this pin is a sink.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: `"-pin <name>"`. Required: `"-pin <name>"`.

## **skew\_groups\_source\_pin**

The list of skew groups for which this clock tree or pin is specified as a source.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: `"-pin <name>"`. Required: `"-pin <name>"`.

## **skew\_pass\_sufficient\_progress\_band\_size\_factor**

Determines what constitutes forward progress on WNS during skew adjustment as a multiple of band size.

Valid values: `double`

*Default:* `0.1`

This global property does not use additional arguments.

## **skew\_passes\_per\_cluster**

Internal property to control ccopt\_design flow.

Valid values: `internal`

*Default:* `25`

This global property does not use additional arguments.

## **skip\_move\_gates\_in\_chains**

Force CTS to skip the move gates in chains step. Force CTS to skip the move gates in chains step. This could seriously damage CTS QoR.

*Default:* `true`

This global property does not use additional arguments.

## **source\_driver**

Specifies the library pin which is assumed to drive this clock tree. It is either a single lib\_pin (in which case all arcs to that lib\_pin shall be used when timing the clock tree root) or a pair of lib\_pins (in which case only arcs from the first specified

pin to the second will be considered). By default this is generated from clock tree extraction.

**Valid values:** lib\_pin | {lib\_pin lib\_pin}

**Default:** {}

Optional applicable arguments: "-clock\_tree <name>".

## **source\_group\_clock\_trees**

A list of the clock trees relevant to this source group.

**Valid values:** list clock\_trees

**Default:** {}

Optional applicable arguments: "-clock\_tree\_source\_group <name>".

## **source\_input\_max\_trans**

The slew which will be assumed at the input of the root driver.

**Valid values:** double

**Default:** 0

Optional applicable arguments: "-delay\_corner <name>", "-clock\_tree <name>", "-early" and "-late".

## **source\_latency**

Specifies a delay value between the global clock source and this clock tree. This additional delay will be included in all timing analysis involving skew groups for which this clock tree is a source. The default is 0.

**Valid values:** double

**Default:** 0

Optional applicable arguments: "-delay\_corner <name>", "-clock\_tree <name>", "-early", "-late", "-rise" and "-fall".

## **source\_max\_capacitance**

The maximum capacitive load which this clock tree is permitted to drive.

**Valid values:** double | auto

Auto: from clock tree extraction

**Default:** auto

Optional applicable arguments: "-clock\_tree <name>".

## **source\_output\_max\_trans**

If non-zero, the slew which will be assumed at the output of the root driver. This overrides the value from SDC.

Valid values: double

*Default:* 0

Optional applicable arguments: "-delay\_corner <name>", "-clock\_tree <name>", "-early" and "-late".

## **source\_pin**

The source pin for this clock\_tree.

Valid values: pin

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-clock\_tree <name>".

## **sources**

A list of sources for this skew group.

Valid values: list pin

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Optional applicable arguments: "-skew\_group <name>".

## **stop\_at\_sdc\_clock\_roots**

If specified, stop searching for parts of the clock tree through SDC clock roots when defining generated clock trees for H-tree sinks.

*Default:* false

Optional applicable arguments: "-flexible\_htree <name>".

## **stripes\_bbox**

The bounding box associated with a preferred cell stripes instance. Cell instances within this box will be preferentially placed on the stripes, if their library cell appears in the cell list associated with the stripes instance.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-preferred\_cell\_stripes <name>". Required: "-preferred\_cell\_stripes <name>".

## **stripes\_bottom\_edge**

The y coordinate (in microns) of the bottom edge of the each stripe in a preferred cell stripes instance.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-preferred\_cell\_stripes <name>". Required: "-preferred\_cell\_stripes <name>".

## **stripes\_cells**

The list of library cells which should be preferentially placed within a stripe of a preferred stripes instance.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-preferred\_cell\_stripes <name>". Required: "-preferred\_cell\_stripes <name>".

## **stripes\_height**

The height (in microns) of each stripe in a preferred cell stripes instance.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-preferred\_cell\_stripes <name>". Required: "-preferred\_cell\_stripes <name>".

## **stripes\_left\_edge**

The x coordinate (in microns) of the left edge of the first stripe in a preferred cell stripes instance.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-preferred\_cell\_stripes <name>". Required: "-preferred\_cell\_stripes <name>".

## **stripes\_pitch**

The distance in microns between the left edges of two adjacent stripes.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-preferred\_cell\_stripes <name>". Required: "-preferred\_cell\_stripes <name>".

## **stripes\_repeat**

The number of stripes in a preferred cell stripes instance.

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-preferred\_cell\_stripes <name>". Required: "-preferred\_cell\_stripes <name>".

## **stripes\_width**

The width (in microns) of each stripe in a preferred cell stripes instance.

**Read-only:** This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

**Applicable arguments:** "-preferred\_cell\_stripes <name>". **Required:** "-preferred\_cell\_stripes <name>".

## **target\_insertion\_delay**

The target insertion delay used for clock tree synthesis. This may be set to the following values:

**auto** - Allow the minimum clustered insertion delay to be pushed up a little (around 5%) to facilitate clock tree power reduction.

**A numeric value** - Attempt to balance the clock tree to the specified insertion delay (specified in library units). CTS will attempt to have a longest clock path delay of no more than this value plus half of the skew target, and a shortest path delay of no less than this value minus half the skew target.

**Valid values:** auto | double

**Default:** auto

**Optional applicable arguments:** "-skew\_group <name>".

## **target\_insertion\_delay\_wire**

The target wire insertion delay used for clock tree synthesis.

**Valid values:** auto | double

**Default:** auto

**Optional applicable arguments:** "-skew\_group <name>".

## **target\_max\_capacitance**

The target maximum capacitive load to allow during clock tree synthesis. This property specifies a maximum (combined pin and wire) capacitance that the clock tree synthesis algorithm will allow any given library pin to drive in a given clock tree when driving a given net\_type. It is specified in library units. It currently only constrains the primary delay corner capacitance values - other delay corners can be specified but will not be constrained. This property is applied in addition to the max\_capacitance constraints read from the liberty library data - the tightest (lowest) of the constraint specified by this property and the constraint present in the liberty data will be used. It also doesn't apply at the root pins of clock trees - to constrain those nets the source\_max\_capacitance CCOpt property should be used instead.

**Valid values:** auto | double

**Default:** auto

**Optional applicable arguments:** "-delay\_corner <name>", "-clock\_tree <name>", "-lib\_pin <name>", "-net\_type <name>",

"-early" and "-late".

## **target\_max\_trans**

The target slew used for clock tree synthesis. This property specifies a maximum slew time that the clock tree synthesis algorithm will allow in this clock tree, in library units. 'default' means 'auto' in primary half corner and 'ignore' in other half corners. If set to 'auto', CTS picks an appropriate value based on the collection of allowed buffer sizes and library parameters, although this may not give optimal quality of results. If set to 'ignore', CTS does not constrain the corner.

**Valid values:** default | auto | ignore | double

**Default:** default

**Optional applicable arguments:** "-delay\_corner <name>", "-clock\_tree <name>", "-net\_type <name>", "-early" and "-late".

## **target\_max\_trans\_sdc**

If non-zero, the target slew used for clock tree synthesis, overriding the SDC. This property specifies a maximum slew time that the clock tree synthesis algorithm will allow, in library units obtained from SDC.

**Valid values:** double

**Default:** 0

**Optional applicable arguments:** "-delay\_corner <name>", "-clock\_tree <name>", "-net\_type <name>", "-early" and "-late".

## **target\_multi\_corner\_allowed\_insertion\_delay\_increase**

This slack is the factor by which we will permit the insertion delay target of this skew group to increase for the purposes of multi corner balancing in general, and wire/cell delay balancing in particular. If a wire/cell delay balance is not possible even after increasing insertion delays by the allowed amount, we will not permit further insertion delay increases.

Values less than 1.0 are not permitted.

A value of 1.0 means "do not allow insertion delay to increase at all".

A value of 2.0 means "allow the insertion delay to at most double."

A value of infinity means "allow any amount of insertion delay increase".

An undefined value ("auto") is treated as infinite.

**Valid values:** auto | double

**Default:** auto

**Optional applicable arguments:** "-skew\_group <name>".

## **target\_skew**

This specifies the target skew for clock tree balancing. This may be set to a numeric value, or one of 'auto', 'ignore' or 'default'.

If set to 'auto' this indicates that an appropriate skew target should be computed.

If set to 'ignore' this indicates that skew should not be balanced for this corner/path combination.

If unspecified then the value of this property is 'default'.

If the value of the property is 'default' the target skew for late delays in the primary delay corner is interpreted as 'auto' and as 'ignore' otherwise.

**Valid values:** default | auto | ignore | double

**Default:** default

**Optional applicable arguments:** "-skew\_group <name>", "-delay\_corner <name>", "-early" and "-late".

## **target\_skew\_wire**

This specifies the target wire skew for clock tree balancing. This may be set numeric value, or one of 'auto', 'ignore' or 'default'.

If set to 'auto' this indicates that an appropriate skew target should be computed.

If set to 'ignore' this indicates that skew should not be balanced for this corner/path combination.

If unspecified then the value of this property is 'default'.

If the value of the property is 'default' the target skew for late delays in the primary delay corner is interpreted as 'auto' and as 'ignore' otherwise.

**Valid values:** default | auto | ignore | double

**Default:** default

**Optional applicable arguments:** "-skew\_group <name>", "-delay\_corner <name>", "-early" and "-late".

## **timing\_delta**

This is the minimum difference in timing values (e.g. slews) that CTTE and CTS will recognize. Values that are closer together than the timing delta will be treated as identical. A smaller delta produces a more accurate timing measurement, but at the expense of runtime (e.g. slews must be propagated further through the timing graph). When set to auto CCOpt automatically determines an appropriate value.

**Valid values:** double | auto

**Default:** auto

This global property does not use additional arguments.

## **top\_buffer\_cells**

A Tcl list of buffer cells. All top routed clock tree nets will be driven by instances of these cells.

Valid values: list cell

*Default:* {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## **top\_inverter\_cells**

A Tcl list of inverter cells. All top routed clock tree nets will be driven by instances of these cells.

Valid values: list cell

*Default:* {}

Optional applicable arguments: "-clock\_tree <name>" and "-power\_domain <name>".

## **trace\_bidi\_as\_input**

Trace bidi pins as input pins during `create_clock_tree_spec`.

Valid values: true false

*Default:* true

This global property does not use additional arguments.

## **transitive\_fanout**

The number of clock sinks in the transitive fanout of the pin, within the clock tree. Requesting this property for a pin not in the clock tree will result in an error.

Valid values: int

Read-only: This property cannot be modified by [set\\_ccopt\\_property](#) or [unset\\_ccopt\\_property](#).

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **trunk\_cell**

The library cell to use inside the H-tree.

Valid values: string

*Default:* {}

Optional applicable arguments: "-flexible\_htree <name>".

## **trunk\_override**

Prefer trunk routing rules for this pin. Only applies to clock tree sinks.

**Valid values:** true false

**Default:** false

Applicable arguments: "-pin <name>". Required: "-pin <name>".

## **update\_io\_latency**

Determine whether to update IO latencies within ccopt\_design.

**Valid values:** true false

**Default:** true

This global property does not use additional arguments.

## **update\_slacks\_before\_skewing\_adjustors**

Forces a full timing update prior to each round of useful skew.

**Valid values:** true false

**Default:** false

This global property does not use additional arguments.

## **use\_estimated\_routes\_during\_final\_implementation**

Use route estimates, rather than NanoRoute or trialRoute, during the final implementation clock routing phase.

**Valid values:** true OR false

**Default:** false

This global property does not use additional arguments.

## **use\_inverters**

Specifies whether clock tree synthesis should prefer to use inverters rather than buffers when balancing the clock tree. If set to true, CTS will use inverters for the clock tree balancing process. If set to false, CTS will use the minimum number of levels of inverters required to maintain logical correctness. If set to auto (the default) CTS will use what it considers to be the best combination of buffers and inverters to get optimal quality of results.

**Valid values:** auto true false

**Default:** auto

Optional applicable arguments: "-clock\_tree <name>".

## **use\_macro\_model\_pin\_cap\_only**

This property is for translating the Macro Model capacitance constraint from FE-CTS spec format Macro Model constraints to CCOpt. The default is false, implying that the Macro Model pin capacitance will be added to the library pin capacitance. If the property is set to true only the Macro Model pin capacitance will be used.

Type: boolean

Default: false

Current Value: false

This global property does not use additional arguments.

## **use\_skew\_implementation\_cache\_hold\_slacks**

If set, CCOpt will gather hold slacks for constructing implementation slack windows. Setting this to true will temporarily switch to the hold analysis mode and gather hold slacks for the purposes of building implementation slack windows that factor in a contribution from the hold views. This incurs extra timing updates.

Valid values: true false

Default: true

This global property does not use additional arguments.

## **useful\_skew\_max\_delta**

The maximum delta for useful skew.

Type: real

Default: 1000

Current Value: 1000

This global property does not use additional arguments.

## **useful\_skew\_min\_delta**

The minimum non-zero delta which useful skew will apply.

Valid values: double | auto

Default: auto

This global property does not use additional arguments.

## **useful\_skew\_post\_implement\_db**

Filename to save design state to after useful skew implementation.

Type: file

*Default:* {}

Current Value:

This global property does not use additional arguments.

## **useful\_skew\_pre\_implement\_db**

Filename to save design state to prior to useful skew implementation.

Type: file

*Default:* {}

Current Value:

This global property does not use additional arguments.

## **user\_skew\_group**

Determines whether a skew group is a user-defined (rather than an internally defined) skew group.

Valid values: true false

*Default:* false

Optional applicable arguments: "-skew\_group <name>".

## **virtual\_delay**

The amount of virtual delay that has been applied under this pin.

Valid values: double

*Default:* 0

Applicable arguments: "-delay\_corner <name>", "-pin <name>", "-category <name>", "-early", "-late", "-rise" and "-fall". Required: "-pin <name>".

## **visualization\_clock\_trees\_initially\_collapsed**

If true, clock trees in the skew and logical clock tree visualization will initially be shown collapsed.

Type: boolean

*Default:* false

Current Value: `false`

This global property does not use additional arguments.

### **`weak_driver_check_bounding_box_vs_drive_distance`**

The ratio between the half perimeter of the sink bounding box and the maximum distance of strongest buffers/inverters to allow clustering with weak drivers.

Valid values: `double`

*Default:* 100

This global property does not use additional arguments.

## **Creating the ICT File**

- [Overview](#)
- [Format](#)
- [Data](#)
- [Comments](#)
- [Case Sensitivity](#)
- [Warnings and Errors](#)
- [Invalid Layer Names](#)
- [Commands](#)
- [Sample ICT File](#)

### **Overview**

The first step involved in modeling the parasitic interconnect capacitance and resistance of your design is to specify the fabrication process information in an Interconnect technology (ICT) file by using the syntax defined in this section. You can use any text editor to enter this information.

**Note:** Although there are no file-naming restrictions for ICT files, you should name your ICT file by using the process name with the `.ict` file extension, as follows:

`process_name.ict` (ICT file)

Fabrication process information consists of the following requirements:

- Minimum spacing and minimum width of the conductors as specified in the design rules for the conductor layers
- Thicknesses of the conductor layers
- Heights of the conductor layers above the substrate (measuring height from the field) or as a delta from a previously defined lower-level conductor layer

- Resistivities of the conductor layers
- Interlayer planar dielectric constant, its height above the substrate (measuring height above the field), and its thickness
- Names of the top conductor layer of a via, the bottom conductor or diffusion layer of the via, and the contact resistance of the via
- Names of the wells

The rest of this section describes the syntax and format of the ICT file containing the process information for your design.

For more information on generating the ICT files, see the *EXT TechGen Reference* manual.

## Format

Lines in the ICT file are in the following general format:

command name {argument\_list}

where argument\_list is a list of field-value pairs. The fields in this syntax are separated by white space. ViewICT, IceCaps, and RCgen ignore blank lines.

**Note:** A backslash (\) is generally required for line continuation, but it is not required if you are using braces ({} ) to define a list.

## Data

All data entered into the ICT file should be the actual physical fabrication process information, not the drawn data.

## Comments

A pound-sign character (#) at the beginning of a line indicates text that ViewICT, IceCaps, and RCgen are treated as comments.

## Case Sensitivity

All keywords in the ICT file are case-insensitive. However, the arguments are case-sensitive. Keywords consist of all command and field names.

## Warnings and Errors

The ViewICT utility displays all errors, warnings, and informational messages on screen and writes them in a log file. Warnings and errors include the corresponding line number.

## Invalid Layer Names

The "NX" string is an invalid layer name.

## Commands

This section describes the commands available in the ICT file.

All command fields are enclosed in braces ({}).

## Process

The `process` command specifies the background dielectric constant. Use it only once in the ICT file.

### Syntax

```
process name {background_dielectric_constant value}
```

or

```
process name {  
    background_dielectric_constant value  
}
```

This syntax contains the following parameters:

- *name*  
Specifies the name of the process.
- `background_dielectric_constant value`  
Specifies the dielectric constant for the region above the top passivation layer or last dielectric layer. This field is required.

### Example

```
process "Process_Example" {  
    background_dielectric_constant 1.0  
}
```

## Well

The `well` command which defines the well layers is an optional command that you can use to differentiate capacitance to a well from capacitance to the substrate.

## Syntax

```
well name { }
```

*name* specifies the name of the well layer.

Anything placed in the brackets is ignored.

## Example

```
well nwell { }
```

```
well pwell { }
```

## Conductor

The `conductor` command defines conductor layers.

You can specify the height of a conductor layer in three ways:

- Height (absolute)
- Delta height (relative)
- Upto (maximum top down)

You can use more than one of these methods per conductor definition, as long as the numbers are valid.

All measurements are in microns, unless otherwise specified.

## Syntax

```
conductor name {field1 value1 ... fieldN valueN}
```

or

```
conductor name {
```

```
    field1 value1
```

```
    ...
```

```
    fieldN valueN
```

```
}
```

You can specify the field-value pairs in any order.

This syntax contains the following parameters:

- `name`  
Specifies the name of the conductor layer.
- `min_spacing value`  
Specifies the minimum spacing permitted by the technology between two conductors (wires) on a layer.
- `min_width value`  
Specifies the minimum width of a conductor.
- `height value`  
Specifies the layer's height above the substrate.
- `delta_height value`  
Specifies the layer's height relative to the top of another layer. This parameter must be used with `delta_layer`.
- `delta_layer layer_name`  
Specifies the reference layer for `delta_height`. It must be a layer that has already been defined. The reference layer must be a conducting layer, a dielectric layer, or a passivation layer. This parameter must be used with `delta_height`.
- `thickness value`  
Specifies the layer's thickness.
- `upto value`  
Specifies the layer's top surface height above the substrate. This value is equal to the height plus the thickness. You only need to specify two of the three or four parameters (`height, {delta_height, delta_layer}, thickness, upto`) to complete the geometrical definition of a conductor layer.
- `resistivity value| [value width] +`  
Specifies the layer's sheet resistance, in ohms per square. You can enter the resistivity value as a constant, or you can enter value-width pairs as a piecewise linear function. You may want to use the value-width pairs to account for width-dependent resistivity.

If you enter value-width pairs, the syntax is as follows:

`resistivity value1 width1 value2 width2 ... valuen widthn`

If the width of the wire is less than the minimum width, `width1`, use the minimum value, `value1`. If the width of the wire is greater than the maximum width, `widthn`, use the maximum value, `valuen`. For widths between value-width pairs, the resistivity value through linear interpolation.

**Note:** The width in the value-width pair refers to the silicon width of the wire.

- `rho`
  - `rho_widths W1 ... Wn`
  - `rho_spacings S1 ... Sm`

- `rho_values R11 .... R1n`

....

`Rm1 ... Rmn`

This parameter is for specifying resistivity as a function of both width and spacing. For values that fall between specified points, linear interpolation is applied. When values are outside of the boundary values, it uses the boundary values.

- `gate_forming_layer [true|false]`

Specifies that this layer forms the gate. The polycide or polysilicon layer is a typical gate-forming conducting layer.

- `field_poly_diffusion_spacing value`

Specifies the lateral spacing between field polycide and diffusion for transistor-level parasitic extraction. There is no lateral separation between gate polycide and diffusion.

- `PnR_widths [value]+, PnR_spacings [value]+`

Allows you to provide design widths and spacings used in the layout to the technology file generation program. These are not necessary for accurate extraction of parasitics if the design widths and spacings are within small perturbations of the minimum process widths and spacings. However, if the design widths and spacings are routinely different from the specified process parameters, it is recommended that you provide these values to the technology file generator.

- `capacitor_only_layer_to layer_name`

Specifies that the current layer be used solely to create high capacitance values in the design and that it is located a few angstroms above or below the `layer_name` layer. Layers having the `capacitor_only_layer_to` keyword set are not extracted.

- `wire_top_enlargement_c Etop`

`wire_top_enlargement_r Etop`

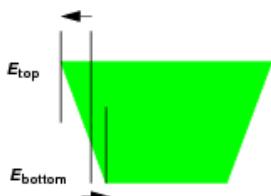
Specifies the enlargement, either positive or negative, of the top edge of the wire. Specify values for both R and C to account for different bias values for R and C. See Figure 1 below.

- `wire_bottom_enlargement_c Ebottom`

`wire_bottom_enlargement_r Ebottom`

Specifies the enlargement, either positive or negative, of the bottom edge of the wire. Specify values for both R and C to account for different bias values for R and C. See the figure below.

**Figure 1: Trapezoidal Wire Shape Resulting from Manufacturing Processes**



- `wire_edge_enlargement | wire_edge_enlargement_[r|c]`

`wee_widths W1 ... Wn`

`wee_spacings S1 ... Sm`

`wee_adjustments E11 ... E1n`

.

.

`Em1 ... Emn`

Models the effect of wire-edge enlargement, if the `wire_edge_enlargement`, `wee_width`, `wee_spacings`, and `wee_adjustments` keywords are specified. The `wee_adjustments` table describes the amount of enlargement applied when certain spacings and widths are observed. For example, the wire is enlarged by  $E_{ij}$  for spacing  $s_i$  and width  $w_j$ . Positive enlargements oversize and negative enlargement undersize the wire. A piecewise constant interpolation is used to obtain enlargements for intermediate spacings and widths. For width/spacings outside of the boundary width/spacing points, the boundary values are used.

`Wire_edge_enlargement_r` and `wire_edge_enlargement_c` can be used if one wants to specify different values for resistance and capacitance.

- `wee_widths W1 ... Wn`

Specifies the widths of the wires in the design. Typically, variation is only seen for widths less than 1.5 microns.

- `wee_spacings S1 ... Sm`

Specifies the spacings of the wires in the design. Typically, variation is only seen for spacings less than 1.5 microns.

- `wee_adjustments E11...E1n ... Em1...Emn`

Specifies the enlargement, either positive or negative, of the wire edge.

See the "Wire-Width Values" section for information on the wire-width values to use.

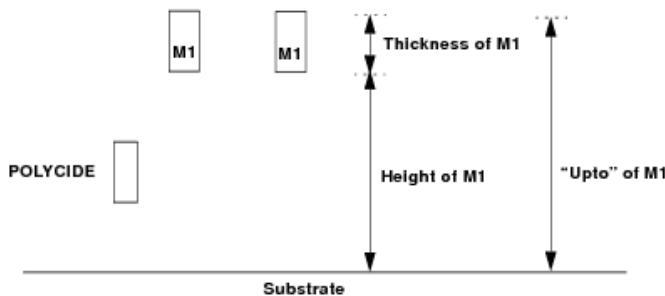
### **Required Conductor Command Fields**

The required fields in this syntax are `min_spacing`, `min_width`, `resistivity`, `gate_forming_layer`, `min_net_fill_spacing`, `x_fill_fill_spacing`, `y_fill_fill_spacing`, `unit_fill_region`, and two of the following three parameters:

- height (or delta\_height and delta\_layer)
- thickness
- upto

The figure below illustrates these parameters.

**Figure 2: Geometric Fields in a Conducting Layer**



### Wire-Width Values

You can use the `wire_edge_enlargement` statement with the `wire_top_enlargement` statement or the `wire_bottom_enlargement` statement, or both in the ICT file. If you use the `wire_edge_enlargement` statement with either or both of these statements, the width of the wires defined by `wee_widths` must be biased as follows:

`drawn_width + ((top + bottom) / 2)`

When calculating resistivity as a function of width, you must use the `wire_top_enlargement` and `wire_bottom_enlargement` values to correct the resistance-width pairs. If a table of wire-edge enlargement values is available, the RC extractor uses the wire widths in the table, which always include biasing and wire-edge enlargement. If this table is not available, the resistance is calculated as follows:

`rho * L / (drawn_width + (top + bottom) / 2 + (top + bottom) / 2)`

where `rho` is the sheet resistivity.

Wire-width values are used in the following order:

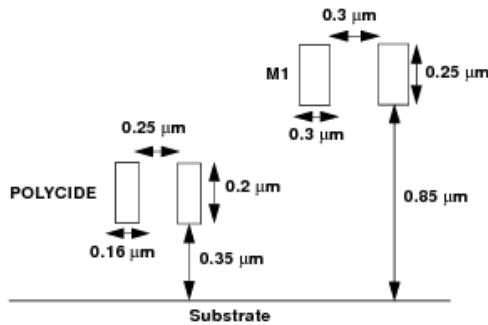
1. Drawn width
2. Biased width
3. Edge-enlarged width
4. Resistivity as a function of width

The figure below illustrates the defining of the conductor layer.

**Figure 3: Example Conductor Definition**

```

;
conductor "POLYCIDE" {
    min_spacing      0.25
    min_width        0.16
    height           0.35
    thickness         0.20
    resistivity       8.6
    gate_forming_layer true
}
conductor "M1" {
    min_spacing      0.30
    min_width        0.30
    height           0.85
    thickness         0.25
    resistivity       8.0
    gate_forming_layer false
}
;
```



### ***Example File for Conductor Definition***

```

conductor "POLYCIDE" {
    min_spacing      0.25
    min_width        0.16
    height           0.35
    upto             0.55
    resistivity       8.6
    gate_forming_layer true
}

conductor "M1" {
    min_spacing      0.30
    min_width        0.30
    delta_layer      POLYCIDE
    delta_height     0.30
    thickness         0.25
    resistivity       8.0
    gate_forming_layer false
    wire_top_enlargement 0.01
    wire_bottom_enlargement -0.01
    wire_edge_enlargement {
        wee_widths      0.18  0.00   0.26   0.30   0.34
        wee_spacings    0.18  0.00   0.26   0.30   0.34   0.38
        wee_adjustments 0.00  0.00  -0.10  -0.10  -0.20
                                         0.00  0.00   0.00  -0.10  -0.20
                                         0.10  0.00   0.00   0.00  -0.10
    }
}
```

```
    0.10  0.10  0.00  0.00  0.00
    0.20  0.20  0.10  0.00  0.00
    0.30  0.20  0.20  0.10  0.00
}
}
```

## Dielectric

The `dielectric` command defines dielectric layers.

All measurements are in microns unless otherwise specified.

## Syntax

```
dielectric name {conformal value field1 value1 ... fieldN valueN}
```

or

```
dielectricname {
    conformalvalue
    field1 value1
    ...
    fieldN valueN
}
```

You can specify the *field-value* pairs in any order.

The syntax for planar dielectrics contains the following parameters:

- `name`  
Specifies the name of the dielectric layer.
- `conformal false`  
Specifies that the dielectric is planar. This field is required.
- `height value`  
Specifies the layer's height above the substrate.
- `thickness value`  
Specifies the layer's thickness.

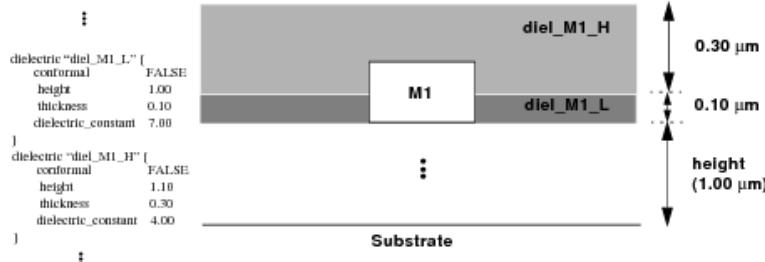
- `dielectric_constant value`  
 Specifies the dielectric constant for this material.
- `delta_height value`  
 Specifies the layer's height relative to the top of another layer.
- `delta_layer layer_name`  
 Specifies the reference layer for `delta_height`. It must be a layer that has already been defined. A reference layer can be a conducting layer or a dielectric layer.
- `upto value`  
 Specifies the layer's top surface height above the substrate. This value is equal to the height plus the thickness. You only need to specify two of the three parameters (`height` (or `{delta_height, delta_layer}`), `thickness`, `upto`) to complete the geometrical definition of a dielectric layer.

The required fields in the specification for planar dielectrics are `conformal`, `dielectric_constant`, and two of the following three parameters:

- `height` (or `{delta_height and delta_layer}`)
- `thickness`
- `upto`

The figure below illustrates the planar dielectric syntax.

**Figure 4: Planar Dielectric Syntax**



## Passivation

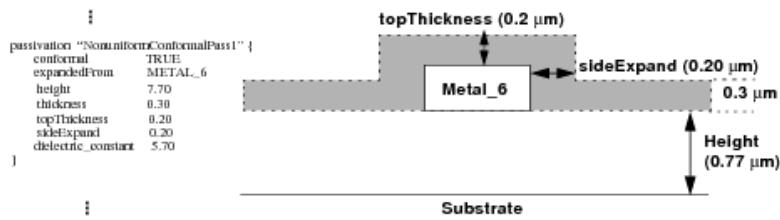
The `passivation` command defines passivation layers. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

## Syntax

The syntax of this command is the same as that of the `dielectric` command, except that `name` specifies the name of the passivation layer. See the "Dielectric" section for information on this syntax. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

The figure below illustrates the defining of the passivation layer.

**Figure 5: Passivation Syntax**



### **Example**

```
passivation "NonuniformConformalPass1" {
    conformal      TRUE
    expandedFrom   METAL_6
    height         7.70
    thickness      0.30
    topThickness   0.20
    sideExpand     0.20
    dielectric_constant 5.70
}
```

### **Via**

The `via` command defines vias or contacts.

### **Syntax**

```
via name {top_layer value bottom_layer value contact_resistance value}
```

This syntax contains the following parameters:

- `top_layer value`  
Specifies the name of the top conductor.
- `bottom_layer value`

Specifies the name of the bottom conductor or diffusion layer.

- contact\_resistance value

Specifies the contact resistance of the via, in ohms.

## Example

```
via "V A1" {  
    top_layer METAL_2  
    bottom_layer METAL_1  
    contact_resistance 7.9  
}
```

Following is a sample specification of a local interconnect via layer. The name of the conductor and the name of the via are the same.

```
conductor "LI" {  
    min_spacing      0.3  
    min_width        0.35  
    height           0.55  
    thickness         0.60  
    resistivity       0.40  
    gate_forming_layer FALSE  
}  
  
via "LI" {
```

```
    top_layer LI  
    bottom_layer POLYCIDE  
    contact_resistance 2.000  
}
```

**Note:** Local interconnect is a layer, usually thicker than the polysilicon layer, that can be deposited after polysilicon and can connect to source-drain regions on the polysilicon layer.

## Sample ICT File

```
#  
# Copyright (c) 2003 Cadence Design Systems, Inc.  
#
```

```
#####
# Process declaration.

#####
process "DIFFERENT_KINDS_OF_DIELECTRIC" {
    background_dielectric_constant 1.0
}

#####

# Well declarations.

#####

well nwell {}
well pwell {}

#####

# Diffusion declarations.

#####

diffusion "N_SOURCE_DRAIN" {
    # Tox is (height of POLYCIDE - thickness of diffusion) = (0.35 - 0.3455) = 0.0045um
    thickness 0.3455
    resistivity 7.7
}
diffusion "P_SOURCE_DRAIN" {
    thickness 0.3455
    resistivity 8.3
}

#####

# Conducting layer declarations.

#####
```

```
conductor "POLYCIDE" {
    min_spacing 0.25
    min_width 0.16
    height 0.35
    thickness 0.20
    resistivity 8.6
    gate_forming_layer true
}

conductor "METAL_1" {
    min_spacing 0.23
    min_width 0.23
    height 1.05
    thickness 0.53
    resistivity 0.086
    gate_forming_layer false
}

conductor "METAL_2" {
    min_spacing 0.28
    min_width 0.28
    height 2.38
    thickness 0.53
    resistivity 0.086
    # The key words TRUE and FALSE are not case sensitive.
    gate_forming_layer FALSE
}

conductor "METAL_3" {
    min_spacing 0.28
    min_width 0.28
    height 3.71
    thickness 0.53
    resistivity 0.086
    gate_forming_layer false
}

conductor "METAL_4" {
```

```
min_spacing 0.28
min_width 0.28
# delta_height + delta_layer is an alternative to height.
delta_height 0.80
delta_layer METAL_3
# "height" is then redundant but it's okay to specify.
#     height 5.04
thickness 0.53
resistivity 0.086
gate_forming_layer false
}

conductor "METAL_5" {
min_spacing 0.28
min_width 0.28
height 6.37
thickness 0.53
resistivity 0.086
gate_forming_layer false
}

conductor "METAL_6" {
min_spacing 0.46
min_width 0.44
height 7.70
thickness 0.99
resistivity 0.035
gate_forming_layer false
}

#####
# Dielectric and passivation layer declarations.
#####

#####
# Base dielectric from substrate...
```

```
#####
#####
```

```
dielectric "First_dielectric" {
# Starts at height zero.

    conformal FALSE
    height 0.00
    thickness 0.35
    dielectric_constant 3.90
}
```

```
# Simple planar dielectric starts at the bottom of POLYCIDE
# and ends at 1.08um which is 0.03um above the bottom of M1.
```

```
dielectric "SimplePlanar1" {
# Starts at height of Poly

    conformal FALSE
    height 0.35
    thickness 0.73
    dielectric_constant 4.00
}
```

```
#####
#####
```

```
# M1 level...
```

```
#####
#####
```

```
# Now a planar intra-metal (M1) dielectric starts 0.03um above from the
# bottom of M1.
```

```
dielectric "PlanarIntraMetal1" {
    conformal FALSE
#
# Starts at height of M1
    height 1.08
# Laterally intersect with M1
```

```
thickness 0.03
dielectric_constant 7.00
}

# The second intra-metal dielectric across M1
# and on top of "PlanarIntraMetal1".

dielectric "PlanarIntraMetal2" {
# Yet another intra-metal planar dielectric layer.

    conformal FALSE
    height 1.11
    upto 1.15
# OR
#     thickness 0.04
    dielectric_constant 3.00
}

# A conformal dielectric.

# When specifying a conformal dielectric (whether it is uniform or
# non-uniform, we must use "conformal TRUE", "expandedFrom", "sideExpand",
# and "topThickness" together.

#
# 1. "conformal" must be set to TRUE.
# 2. "expandedFrom" can be a metal layer or a dielectric/passivation layer.
# The conformal dielectric layer must be expanded from its immediate
# lower (metal/dielectric/passivation) layer. It cannot be expanded
# from a planar dielectric layer.
# 3. "thickness" is the bottom dielectric thickness.
# 4. "sideExpand" specifies the side thickness.
# 5. "topThickness" is the thickness of the dielectric above the
# top of the "expandedFrom" layer.

dielectric "conformalAtTopOFM1" {
# Conformal above M1
```

```
conformal TRUE
expandedFrom METAL_1

# and starts from the top of "PlanarIntraMetal2"
height 1.15

# Base/Bottom thickness of the conformal dielectric.
thickness 0.43

# The thickness of the dielectric above the "expandedFrom" object, i.e. M1.
topThickness 0.43

# This is the side thickness of the dielectric.
sideExpand 0.43
dielectric_constant 4.10
}

dielectric "SimplePlanar2" {
# From top of M1 to bottom of M2
conformal FALSE
height 1.58
thickness 0.80
dielectric_constant 4.00
}

#####
# M2 level...
#####

# An uniform conformal dielectric starting from the bottom of M2.

dielectric "UniformConformal1" {
conformal TRUE
expandedFrom METAL_2
```

```
# Height of M2
    delta_height 0.00
        delta_layer SimplePlanar2
#    height 2.38
    thickness 0.50
    topThickness 0.50
    sideExpand 0.50
    dielectric_constant 3.00
}

# A nonuniform conformal dielectric is one when any one of "thickness",
# "sideExpand", and "topThickness" are different.

dielectric "NonuniformConformal1" {
    conformal TRUE
    height 2.88
    thickness 0.10
    expandedFrom UniformConformal1
    sideExpand 0.03
    topThickness 0.05
    dielectric_constant 7.00
}

dielectric "SimplePlanar3" {
    conformal FALSE
    height 2.98
    thickness 0.73
    dielectric_constant 4.10
}

#####
# M3 level...
#####
```

```
# A special case of conformal dielectric.

dielectric "NonuniformConformal2" {

# Humps over M3 with side and top thicknesses equal to 0.17 um and 0.50 um, respectively.

    conformal TRUE
    expandedFrom METAL_3
    height 3.71

# Note that the bottom thickness is thicker than M3!

    thickness 0.90
    topThickness 0.50
    sideExpand 0.17
    dielectric_constant 4.10
}

dielectric "SimplePlanar5" {

    conformal FALSE
    height 4.61

# Upto the bottom of M4.

    upto 5.04
    dielectric_constant 3.00
}

#####
# M4 level...
#####

dielectric "NonuniformConformal3" {

    conformal TRUE
    expandedFrom METAL_4
    # Height of M4

    height 5.04
    thickness 0.30
    topThickness 0.30
    sideExpand 0.10
}
```

```
# Special case. See SimplePlanar6.

    dielectric_constant 4.10

}

dielectric "PlanarIntraMetal3" {

# Planar intrametal dielectric.

    conformal FALSE

    height 5.34

    upto 5.44

    dielectric_constant 3.10

}

dielectric "PlanarIntraMetal4" {

# Top off the top of NonuniformConformal3.

    height 5.44

    upto 5.87

    dielectric_constant 3.00

}

dielectric "SimplePlanar6" {

    conformal FALSE

    height 5.87

# Upto the bottom of M5.

    upto 6.37

# NOTE that it has the same dielectric constant as NonuniformConformal3.

# This makes "NonuniformConformal3" a special case.

    dielectric_constant 4.10

}

#####
# M5 level...
#####

dielectric "UniformConformal3" {

    conformal TRUE

    expandedFrom METAL_5

    height 6.37
```

```
thickness 0.10
topThickness 0.10
sideExpand 0.10
dielectric_constant 7.20
}

dielectric "PlanarIntraMetal5" {
# Special planar dielectric which intersects "UniformConformal3"
conformal FALSE
height 6.47
thickness 0.40
dielectric_constant 3.00
}

dielectric "PlanarIntraMetal6" {
# Another special planar dielectric which intersects "UniformConformal3"
conformal FALSE
height 6.87
thickness 0.10
dielectric_constant 4.00
}

dielectric "PlanarIntraMetal7" {
# Yet another special planar dielectric which intersects "UniformConformal3"
conformal FALSE
height 6.97
thickness 0.03
dielectric_constant 7.00
}

#####
#passivation "PlanarPass1" {
# From top of M5 to bottom of M6.
#conformal FALSE
#height 7.00
#thickness 0.70
```

```
dielectric_constant 4.00
}

#####
# M6 level...
#####

passivation "NonuniformConformalPass1" {
    conformal TRUE
    expandedFrom METAL_6
    height 7.70
    thickness 0.30
    topThickness 0.20
    sideExpand 0.20
    dielectric_constant 5.70
}
passivation "PlanarPass2" {
    conformal FALSE
    height 8.00
    upto 8.89
    dielectric_constant 4.30
}
#####

passivation "PlanarPass3" {
    conformal FALSE
    height 8.89
    thickness 1.00
    dielectric_constant 3.00
}

#####
# Contacts and Via declarations.
#####
```

```
via "CONT" {
    top_layer METAL_1
    bottom_layer POLYCIDE
    contact_resistance 7.8
}

via "CONT" {
    top_layer METAL_1
    bottom_layer N_SOURCE_DRAIN
    contact_resistance 11
}

via "CONT" {
    top_layer METAL_1
    bottom_layer P_SOURCE_DRAIN
    contact_resistance 10
}

via "VA1" {
    top_layer METAL_2
    bottom_layer METAL_1
    contact_resistance 7.9
}

via "VA2" {
    top_layer METAL_3
    bottom_layer METAL_2
    contact_resistance 8.2
}

via "VA3" {
    top_layer METAL_4
    bottom_layer METAL_3
    contact_resistance 8.1
}

via "VA4" {
    top_layer METAL_5
    bottom_layer METAL_4
}
```

```
contact_resistance 8.0
}
via "VA5" {
    top_layer METAL_6
    bottom_layer METAL_5
    contact_resistance 4.0
}
```

# Clock Mesh Specification File

- [Overview](#)
- [Routing Type Definitions](#)
- [Cutout Definitions](#)
- [Clock Mesh Definitions](#)
  - [Timing and Power Constraints Section](#)
  - [Tracing and Analysis Scope Section](#)
  - [Mesh Structure Section](#)
  - [Global Mesh Section](#)
  - [Multispine Clock Mesh](#)
  - [Analysis Section](#)
  - [Top Chain Section](#)
  - [Local Tree Section](#)
- [Clock Mesh Specification File Example](#)

## Overview

Before running clock mesh synthesis, you must create a clock mesh specification file. The clock mesh specification file defines the clock meshes to be created for the design.

A clock mesh specification file contains the following main sections:

- [Routing Type Definitions](#)
- [Cutout Definitions](#)
- [Clock Mesh Definitions](#)

You can specify the following units in a clock mesh specification file:

- Distance unit: um
- Time unit: ps, ns
- Voltage unit: v, mv
- Power unit: w, mw

If you do not specify units in the file, the clock mesh tool generates warning messages.

## Routing Type Definitions

The routing type definition is a set of routing properties to be used during clock mesh implementation. You must define at least two routing types: one for vertical and one for horizontal mesh trunks or branches. The routing types are then referenced in the clock mesh definition.

The following table describes the entries for the routing type definition:

RouteTypeDef <i>typeName</i>	Specifies the routing type for which you are defining routing attributes. This attribute denotes the beginning of the <code>RouteTypeDef</code> section.
Layer <i>layerName</i>	Specifies the metal layer to be used. The name you specify must be a layer name defined in the LEF file.
Width <i>distance</i>	Specifies the width of the metal layer to be used.
Spacing <i>distance</i>	Specifies the spacing to all other wires in the design. <i>Default:</i> Uses the minimum spacing values specified in the LEF file
ShieldWidth <i>distance</i>	Specifies the width of the shield wire. <i>Default:</i> The tool gets the width requirement from a LEF file.
ShieldSpacing <i>distance</i>	Specifies the spacing between the shield wire and the neighboring wire. <i>Default:</i> The tool gets the spacing requirement from a LEF file.
End	Denotes the end of the <code>RouteTypeDef</code> section.

**Note:** `ShieldWidth` and `ShieldSpacing` are optional settings.

## Cutout Definitions

The cutout definition section specifies areas that should not be covered by the clock mesh. When you specify a cutout area, the clock mesh tool must stop mesh routing at the cut out boundary. If you want to completely avoid an area, use a placement or routing blockage.

The `Cutout` definition section is optional.

You can define a cutout area in one of the following ways:

- X Y coordinates

You can specify a set of upper-right and lower-left X and Y coordinates to define the cutout area. For example:

```
Cutout
```

```
+ 0um 0um 400um 280um
```

- Instance names

You can specify an instance name on which to base the cutout area. The cutout area then covers the area of the instance. For example:

```
Cutout
```

```
+ INST1/C1
```

+ INST2/C1

- Instance names with instance halos

You can define a cutout area by specifying the distance in microns for which to increase the size of instance-based cutouts from the instance-cell boundary. For example:

Cutout

+ INST1/C1 HALO 50um

**Note:** You also can use the [createClockMeshCutout](#) command to create cutout areas.

## Clock Mesh Definitions

Each clock mesh definition defines a single clock mesh to be synthesized. A clock mesh specification file can contain multiple clock mesh definitions.

ClockMesh <i>meshName</i>	Specifies the name of the clock mesh to be synthesized. Each clock mesh definition in the clock mesh specification file must start with a <code>ClockMesh</code> statement.
End	Marks the end of a clock mesh definition. Each clock mesh definition in the clock mesh specification file must close with an <code>End</code> statement.

A clock mesh definition can contain the following information sections:

- [Timing and Power Constraints Section](#) (required)
- [Tracing and Analysis Scope Section](#) (required)
- [Mesh Structure Section](#) (required)
- [Global Mesh Section](#)(required)
- [Top Chain Section](#) (optional)
- [Local Tree Section](#) (optional)

## Timing and Power Constraints Section

This section defines timing and power constraints for the clock mesh.

The following table describes the entries for the timing and power constraints section:

Period <i>timeValue</i>	Specifies the clock period (in picoseconds) for the mesh. <i>Default:</i> 0 (frequency also will default to 0)
-------------------------	---

MaxPower <i>timeValue</i>	Specifies the maximum power value (in milliwatts) for the mesh.  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0
RootTrans <i>timeValue</i>	Specifies the transition time (in picoseconds) at the root pin.  <i>Default:</i> 0
MinDelay <i>timeValue</i>	Specifies the minimum phase delay (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0
MaxDelay <i>timeValue</i>	Specifies the maximum phase delay (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> Largest possible number, based on the machine
MaxSkew <i>timeValue</i>	Specifies the maximum skew between sink pins (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0
MaxBufferTrans <i>timeValue</i>	Specifies the maximum input transition time for buffers (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0
MaxLeafTrans <i>timeValue</i>	Specifies the maximum input transition time for leaf pins (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0

## Tracing and Analysis Scope Section

This section specifies the logical extent of the clock domain, from the root through any mesh drivers to the leaves. Scope is determined by tracing the netlist from the clock root.

The following table describes the entries for the tracing and analysis scope section:

<pre>RootPin instanceName/pinName</pre>	Specifies the complete name of the clock root pin from which the tracing starts.
<pre>LeafPin + instanceName/pinName {rising   falling}</pre>	<p>Defines the specified input pin as a leaf pin for non-clock type instances. The clock mesh tool will stop tracing at this pin, and skew is analyzed only to this pin. This statement also defines the rising or falling trigger edge for the inputs.</p> <p>For example:</p> <pre>LeafPin + corem/sync1/reg_3/D rising + corem/sync1/reg_4/D rising</pre>
<pre>LeafCellPin + cellType/pinName {rising   falling}</pre>	<p>Defines the specified input pin as a leaf pin for non-clock type cells. The clock mesh tool will stop tracing at this pin, and skew is analyzed only to this pin. This statement also defines the rising or falling trigger edge for the inputs.</p> <p>For example:</p> <pre>LeafCellPin + AND2X/A rising</pre>
<pre>DefaultTrigger {rising   falling}</pre>	Specifies the default trigger edge value for leaves that do not have a specified trigger edge value.
<pre>AllowGating {true   false}</pre>	<p>Specifies whether the clock mesh tool traces through gates. If you specify <code>true</code>, the tool traces until it finds leaf pins. If you specify <code>false</code>, the tool stops at gate inputs.</p> <p><i>Default:</i> <code>false</code></p>

## Mesh Structure Section

This section defines the overall logical attributes of the clock mesh structure.

The following table describes the entries for the mesh structure section:

<pre>UseMeshModule [true   false]</pre>	<p>Specifies whether the clock mesh tool should create a separate module for the mesh buffers. If you specify <code>true</code>, the tool creates a module at the level of the root net. If you specify <code>false</code>, the tool inserts buffers (flat) at the level of the root net.</p> <p><i>Default:</i> <code>true</code></p>
<pre>MeshModule moduleName</pre>	<p>Specifies the module name to be used when synthesizing the clock mesh.</p> <p><b>Note:</b> The <code>UseMeshModule</code> statement must be set to <code>true</code> in order for this statement to apply.</p> <p><i>Default:</i> The clock mesh tool creates a name based on the mesh name.</p>

<pre>LoadCell + cellName1 [+ cellName2] ...</pre>	<p>Defines the specified cell as a loading cell that can be inserted into the final stage global mesh net, to minimize skew for the global mesh. You can specify buffers, inverters, or single input cells that do not have an output pin.</p> <p>For example:</p> <pre>LoadCell  + T2BUFCLXR  + CLKBUFX16</pre>
<pre>MeshArea llx lly urx ury</pre>	<p>Specifies a set of upper-right and lower-left X and Y coordinates to define the initial choice of mesh area for the tool to consider when synthesizing global clock mesh.</p>

## Global Mesh Section

This section defines physical attributes of the global clock mesh structure.

The following table describes the entries for the global mesh section:

<pre>GlobalMesh</pre>	<p>Denotes the beginning of the global mesh definition. The global mesh definition must begin with a <code>GlobalMesh</code> statement, and close with an <code>End</code> statement.</p>	
<pre>MeshDrivePoint {Center   Root   x, y}</pre>	<p>Specifies the position of the first-stage pre-drivers.</p> <p>If you do not specify this statement, the software positions the first-stage pre-drivers as follows:</p> <ul style="list-style-type: none"> <li>• For HTreeMesh and Fishbone meshes, positions the pre-drivers near the center of the clock mesh structure (<code>Center</code>)</li> <li>• For CTS-generated pre-drive structures, positions the pre-drivers based on whether there is a <code>TopChain</code> section specified in the clock mesh specification file. If there is a <code>TopChain</code> section in the spec file, the drive point will be the center of the mesh structure (<code>Center</code>). If there is no <code>TopChain</code> section defined, the drive point will be the clock root pin (<code>Root</code>).</li> </ul>	
	<pre>Center</pre>	<p>Positions the pre-drivers near the center of the clock mesh structure.</p>
	<pre>Root</pre>	<p>Positions the pre-drivers near the clock root pin.</p>
	<pre>x , y</pre>	<p>Positions the pre-drivers at the specified location.</p>

MeshType {Fishbone   HTreeMesh   MultiSpine}	<p>Specifies the type of mesh structure to be synthesized.</p> <p><b>Note:</b> You must specify the <code>MeshType</code> statement if you want to synthesize a clock mesh. If you only want to generate a report, the statement is optional.</p> <p>For more information, see <a href="#">Multispine Clock Mesh</a>.</p>	
PatternTrunkClusterTargetSize <i>n</i>	<p>Selects the routing pattern for clock nets between the final level mesh drivers and receivers (either flops or instance).</p> <ul style="list-style-type: none"> <li>● If <i>n</i> is 0, the pattern is Steiner.</li> <li>● If <i>n</i> is 1, the pattern is Trunk.</li> <li>● If <i>n</i> is greater than 1, the trunk pattern has a target cluster size.</li> </ul> <p><i>Default:</i> 1</p>	
TrunkOrientation [Horizontal   Vertical]	<p>Specifies the trunk orientation.</p> <p><b>Note:</b> You must specify the <code>TrunkOrientation</code> statement if you want to synthesize a clock mesh. If you only want to generate a report, the statement is optional.</p>	
TrunkPlacement {UniformPitch   LoadWeighted}	<p>Specifies how trunks are placed if pre-defined target locations are not specified using the <code>TargetTrunkLocs</code> statement in the mesh stage section.</p>	
	UniformPitch	Places the trunks uniformly according to trunk pitch.
	LoadWeighted	Places each trunk so that it drives a similar load.
HTreePattern <i>pattern</i>	<p>Specifies the order for building horizontal (<code>H</code>) and vertical (<code>V</code>) structures for an HTree + Mesh clock mesh.</p> <p>You can use <code>H</code> and <code>V</code> in any pattern to specify the order. Specifying an asterisk (*) causes the tool to repeat alternating structures. For example, specifying <code>H*</code> is equivalent to specifying <code>HVHV</code>; it is <i>not</i> equivalent to specifying <code>HHHH</code>.</p> <p><b>Note:</b> You only can use this statement when you specify <code>MeshType HTreeMesh</code>.</p>	
TrunkDriveDist {StrictAttach   Uniform   LoadWeighted   LoadWeightedMatch}	<p>Specifies how buffers are placed along driving trunks in the final global mesh stage.</p> <p><i>Default:</i> StrictAttach</p>	
	StrictAttach	Places the buffers according to the specified <code>BranchAttachFrequency</code> value.
	Uniform	<p>Places the buffers in an even, uniform distribution along the trunk.</p> <p><b>Note:</b> If you specify <code>Uniform</code>, the clock mesh tool ignores the <code>BranchAttachFrequency</code> value.</p>

	LoadWeighted	Places the buffers according to load distribution. The <code>BranchAttachFrequency</code> value limits how closely the buffers can be placed to each other.
	LoadWeightedMatch	Places the drivers according to load distribution, and applies the same pattern to all trunks. The <code>BranchAttachFrequency</code> value limits how closely the buffers can be placed to each other.
PreDriveCTS		Denotes the beginning of the CTS-generated pre-drive structure definition. The pre-drive structure definition must begin with a <code>PreDriveCTS</code> statement, and close with an <code>End</code> statement.
Enabled [true   false]		<p>Enables the implementation of the pre-drive structure.</p> <p>When you specify <code>true</code>, the clock mesh tool calls CTS to synthesize the clock tree using the last stage of the global mesh drivers as the leaf pin.</p> <p>The position of the first-level driver in the CTS pre-drive structure is determined by the <code>MeshDrivePoint</code> statement in the clock mesh spec file. If the <code>MeshDrivePoint</code> statement is not defined, the position depends on whether a <code>TopChain</code> section is specified. If there is a <code>TopChain</code> section in the spec file, the drive point will be the center of the mesh structure; if there is no <code>TopChain</code> section defined, the drive point will be the clock root pin.</p> <p>By default, the clock mesh tool decides the number of levels for the pre-drive structure.</p> <p><i>Default:</i> <code>false</code></p>
DriveCells + <i>cellName1</i> + <i>cellName2</i> ...		<p>Specifies the types of drive cells to use for the pre-drive structure. You must specify at least one cell choice. Multiple drive cells can be specified.</p> <p>For example:</p> <pre>DriveCells + + CLKBUFX20 + + CLKBUFX16</pre>
NonDefaultRule <i>ruleName</i>		<p>Specifies the LEF <code>NONDEFAULTRULE</code> statement for the router to use when routing the nets in the pre-drive structure.</p> <p><i>Default:</i> The router uses the default routing rule.</p>
TopPreferlayer <i>layerName</i>		Specifies the top preferred metal layer to use for routing the pre-drive structure. The name you specify must be a layer defined in the LEF file.
BottomPrefLayer <i>layerName</i>		Specifies the bottom preferred metal layer to use for routing the pre-drive structure. The name you specify must be a layer defined in the LEF file.

DummyBuffer + <i>cellName1</i> + <i>cellName2</i> ....	Specifies the cells for dummy load insertion while performing pre-drive CTS.
optAddBuffer {true   false}	Specifies whether the buffer insertion is permitted or not while performing pre-drive CTS.  <i>Default:</i> true
optAddBufferLimit <i>n</i>	Specifies the maximum number of buffers to be inserted while performing pre-drive CTS.  By default, it uses a dynamic number (1/2 the number of mesh drivers).
TargetInputSkewSlewRatio <i>ratio</i>	Fixes the maximum transition time of the mesh driver input to meet the skew to slew ratio. Here, the skew and slew is the maximum skew or slew of all the mesh drivers.  If ratio is zero, do not fix the transition time. If ratio is not specified in the clock mesh specification file, then the default value is 0.
PreferredExtraSpace [0-3]	Specifies the extra space to add around clock wires when routing the pre-drive structure.  <i>Default:</i> 1
End	Denotes the end the pre-drive structure definition. The pre-drive structure definition must begin with a PreDriveCTS statement, and close with an End statement.

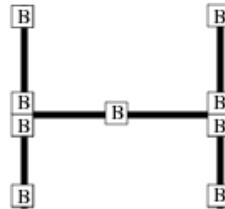
```
HTreeSplitDrive [true | false]
```

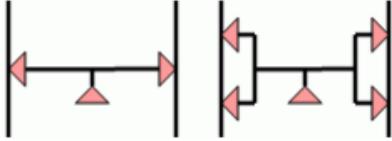
Splits branches at the driving point. Splitting a branch can increase the drive strength without creating a multi-drive net. To split a branch, you must have an even number of drivers at the branching point (for example, 2, 4, 6).

For example, if you define the following in the clock mesh spec file:

```
ClockMesh Clk  
...  
GlobalMesh  
HTreeSplitDrive true  
Stage  
    NumDriver 1  
End  
Stage  
    NumDriver 4  
X2  
End  
Stage  
    NumDriver 4  
End  
...
```

The software splits the branches as follows:



Stage	<p>Denotes the beginning of a particular stage definition. This section defines stage-specific parameters for the global mesh, including driver cells, route types, and trunk and branch information.</p> <p>Each stage definition must begin with a <code>Stage</code> statement, and close with an <code>End</code> statement.</p> <p><b>Note:</b> You must define an even number of stage definitions if you use an inverter for the drive cell (<code>DriveCell</code>).</p>
NumDriver <i>number</i>	<p>Specifies the number of drivers that should be inserted at the current stage.</p>
X <i>number</i>	<p>Specifies a grouping factor that controls the number of drivers at each node or endpoint at the current stage of an HTree clock mesh structure. If you use the X grouping factor, you can increase the drive at any given level of the tree.</p> <p><b>Note:</b> This statement only can be used when you are defining an HTree + Mesh structure.</p> <p><i>Default:</i> Uses a single driver at the end of each node.</p> <p>In the following illustration, the diagram on the left shows two drivers in stage 2, where X = 1, and the diagram on the right shows four drivers in stage 2, where X=2.</p> <div style="text-align: center;">  </div>
DriveCell <i>cellName</i>	<p>Specifies the type of driver to be used for the stage being defined.</p>
RouteTypePair <i>type1 type2</i>	<p>Specifies the routing types to be used for the stage. Routing types are defined using the <code>RouteTypeDef</code> statement.</p>
NumTrunk <i>number</i>	<p>Specifies the number of trunks to create for a particular stage. You can specify this statement for any stage in the clock mesh structure, but it only is useful for the final construction stage.</p> <p>A Fishbone mesh structure can have one or two trunks (that is, you can specify single or double Fishbone mesh structures). The number of trunks for the second-to-last stage will be 1 or 3, depending on whether the mesh structure is a single or double Fishbone. For all other stages, the number of trunks is 1.</p> <p>For an HTree + Mesh structure, this statement is relevant only to the last stage. If you do not specify this statement for an HTree + Mesh structure, the clock mesh tool computes a suitable value automatically. If you specify a value that is less than the minimum number of trunks required to implement the structure, the clock mesh tool displays an error message.</p>

<i>TrunkPitch distance</i>	<p>Specifies the pitch for the trunk for a particular stage.</p> <p><b>Note:</b> The clock mesh tool considers the <code>TrunkPitch</code> statement to be a soft constraint that applies to the final mesh stage. The tool might need to adjust the specified pitch in order to meet physical constraints.</p>
<i>TrunkAttachFrequency number</i>	<p>Specifies the frequency with which buffers are attached to the trunks. You must specify a number that is greater than or equal to 1.</p> <p><i>Default:</i> 1</p> <p>In the following illustration, assume the design has eight drivers, and the trunk orientation is horizontal.</p> <p>If <code>TrunkAttachmentFrequency</code> is 1, the minimum number of horizontal trunks is 2, as shown in the diagram on the left (each trunk must have a driver attached to it).</p> <p>If <code>TrunkAttachmentFrequency</code> is 2, the minimum number of trunks is 3, as shown in the diagram on the right (the middle trunk must not have a driver attached to it). The clock mesh tool generates an error if the specified number of branches does not meet the minimum number of branches.</p> 
<i>NumBranch number</i>	<p>Specifies the number of branches.</p> <p><i>Default:</i> The clock mesh tools computes a suitable value automatically.</p>
<i>BranchPitch distanceValue</i>	<p>Specifies the pitch for the branches for a particular stage.</p> <p><b>Note:</b> The clock mesh tool considers the <code>BranchPitch</code> statement to be a soft constraint that applies to the final mesh stage. The tool might need to make adjustments to the specified pitch in order to meet physical constraints.</p>

<p><code>TargetBranchOrigin coordinate_in_um</code></p>	<p>Specifies the absolute location, in microns, of the first branch for a clock mesh.</p> <p><b>Note:</b> The <code>TargetBranchOrigin</code> statement changes the branch origin only. It does not affect the number of branches or the branch pitch.</p> <p>For a horizontal trunk, <code>coordinate_in_um</code> is the x-axis coordinate for the left-most branch, as shown in the following illustration:</p> <div style="text-align: center; margin-bottom: 20px;"> <p>Horizontal Trunk</p> <p>TargetBranchOrigin 100μm</p> <p>BranchPitch 400μm</p> </div> <p>For a vertical trunk, <code>coordinate_in_um</code> is the y-axis coordinate for the bottom branch, as shown in the following illustration:</p> <div style="text-align: center;"> <p>Vertical Trunk</p> <p>BranchPitch 400μm</p> <p>TargetBranchOrigin 50μm</p> </div>
<p><code>BranchAttachFrequency number</code></p>	<p>Controls the frequency with which buffers are attached to branches. You can specify a positive or negative whole number for <code>number</code>.</p> <p><b>Note:</b> The clock mesh tool honors the <code>BranchAttachFrequency</code> statement depending on the setting of the <code>TrunkDriveDist</code> statement. If you specify <code>TrunkDriveDist Uniform</code>, the clock mesh tool ignores the <code>BranchAttachFrequency</code> value.</p> <p>In the following illustration, if <code>BranchAttachFrequency</code> is 1, each branch must have a driver attached to it, as shown in the diagram on the left. If <code>BranchAttachFrequency</code> is 2, there must be one branch between each driver that does not have a driver attached to it, as shown in the middle diagram. If <code>BranchAttachFrequency</code> is -2, there are drivers between the branches that are not attached to branches, as shown in the diagram on the right.</p> <div style="text-align: center;"> <p>Branch</p> </div>

<pre>TargetTrunkLocs + trunkLocation...</pre>	<p>Positions the trunks at the specified locations.</p> <p>For example:</p> <pre>TargetTrunkLocs  + 100um  + 200um  + 300um  + 550um</pre> <p>You are not required to specify target trunk locations for every stage; however, if you specify a stage, you must list all trunks for that stage.</p> <p><b>Note:</b> The clock mesh tool considers the <code>TargetTrunkLocs</code> statement a soft constraint; it attempts to position the trunks at the specified locations, but might deviate a small distance to find a solution.</p>
<pre>NonDefaultRule ruleName</pre>	<p>Specifies the LEF <code>NONDEFAULTRULE</code> statement for the router to use when routing the nets for a particular stage of the global mesh structure.</p> <p><i>Default:</i> The router uses the default routing rule.</p>
<pre>TopPreferLayer layerName</pre>	<p>Specifies the top preferred metal layer to use for routing for a particular stage of the global mesh structure. The name you specify must be a layer defined in the LEF file.</p>
<pre>BottomPreferLayer layerName</pre>	<p>Specifies the bottom preferred metal layer to use for routing for a particular stage of the global mesh structure. The name you specify must be a layer defined in the LEF file.</p>
<pre>PreferredExtraSpace [0-3]</pre>	<p>Specifies the extra space to add around clock wires, when routing the nets in the clockmesh stage.</p> <p><i>Default:</i> 0</p>
<pre>End</pre>	<p>Denotes the end of a particular stage definition. Each stage definition must begin with a <code>Stage</code> statement, and close with an <code>End</code> statement.</p>
<pre>End</pre>	<p>Denotes the end of the global mesh definition. The global mesh definition must begin with a <code>GlobalMesh</code> statement, and close with an <code>End</code> statement.</p>
<pre>ShieldNet netname</pre>	<p>Specifies the shielding net name for the global mesh.</p> <p><i>Default:</i> No shielding</p> <p><b>Note:</b> Shield nets using special wires for shielding global mesh nets are created during <code>synthesizeClockMesh</code>.</p> <p><b>Note:</b> Shield nets using regular wires for shielding top chain and local tree nets are created during <code>routeClockMesh</code>.</p>

## Multispine Clock Mesh

The multispine clock mesh is a structure that provides a good QoR for non-rectangular floorplans.

The following table describes the parameters that are pertaining to multispine clock mesh structure only:

### The MainDrive Section

This section defines the physical attributes of the main mesh drive structure. The `MainDrive` section contains the `SpineTemplate` and `Spine` sections. The following table describes the entries for the `MainDrive` section.

<code>MainDrive</code>	Denotes the beginning of the <code>MainDrive</code> section. The <code>MainDrive</code> section must begin with a <code>MainDrive</code> and close with an <code>End</code> statement. The tool supports only one stage of spine and each spine must have same type of clock driver cell and routing type.
<code>SpineTemplate</code>	This section is for specification of common settings for all spines such as the routing type and the clock driver used.
<code>Spine</code>	This section is for specification of settings for each spine.
<code>Stage</code>	This section contains same type of clock driver cell and routing type.
<code>End</code>	Denotes the end of the <code>Stage</code> section.
<code>End</code>	Denotes the end of the <code>Spine</code> definition. The spine definition must begin with a <code>SpineTemplate</code> statement, and close with an <code>End</code> statement
<code>End</code>	Denotes the end of the <code>SpineTemplate</code> definition. The <code>SpineTemplate</code> definition must begin with a <code>SpineTemplate</code> statement, and close with an <code>End</code> statement.
<code>End</code>	Denotes the end of the <code>MainDrive</code> definition. The <code>MainDrive</code> definition must begin with a <code>MainDrive</code> statement, and close with an <code>End</code> statement.

### The SpineTemplate Section

This section is for specification of common settings for all spines such as the routing type and the clock driver used. The following table describes the entries for the `SpineTemplate` section

<code>SpineTemplate</code>	Denotes the beginning of a <code>SpineTemplate</code> section.  The <code>SpineTemplate</code> section must begin with a <code>SpineTemplate</code> and close with an <code>End</code> statement.
<code>Stage</code>	Denotes the beginning of a stage definition. The <code>Stage</code> section must begin with a <code>Stage</code> and close with an <code>End</code> statement.  The stage section of a spine template is for specification of parameters such as the driver cells, route types and routing attributes.

End	Denotes the end of a stage definition. Each stage definition must begin with a <code>Stage</code> statement and close with an <code>End</code> statement.
End	Denotes the end of the <code>SpineTemplate</code> definition. The <code>SpineTemplate</code> definition must begin with a <code>SpineTemplate</code> statement, and close with an <code>End</code> statement.

## The Spine Section

This section is for specification of settings for each spine. The following table describes the entries for the `spine` section.

Spine	Denotes the beginning of a spine definition. The <code>Stage</code> section must begin with a <code>spine</code> and close with an <code>End</code> statement. This section is for specification of the target location of the spine.
<u>Stage</u>	Denotes the beginning of a stage definition. The <code>Stage</code> section must begin with a <code>Stage</code> and close with an <code>End</code> statement.  The stage section of a spine section is for specification of parameters such as the number of driver for each spine.
TargetLoc	Specifies the target location.
End	Denotes the end of a stage definition. Each stage definition must begin with a <code>Stage</code> statement, and close with an <code>End</code> statement.
End	Denotes the end of the Spine definition. The spine definition must begin with a <code>SpineTemplate</code> statement, and close with an <code>End</code> statement.

## ***The Stage Definition for the MainDrive Section***

The following table describes the `Stage` definition for the `MainDrive` section

Stage	Denotes the beginning of a stage definition. The <code>Stage</code> section must begin with a <code>Stage</code> and close with an <code>End</code> statement.
DriveCell	Specifies the clock driver cell.
RouteType	Specifies the routing type.
End	Denotes the end of a stage definition. Each stage definition must begin with a <code>Stage</code> statement, and close with an <code>End</code> statement

## ***The Stage Definition for the SpineTemplate Section***

The following table describes the `Stage` definition for the `SpineTemplate` section.

Stage	<p>Denotes the beginning of a stage definition. The <code>Stage</code> section must begin with a <code>Stage</code> and close with an <code>End</code> statement.</p> <p>The stage section of a spine template is for specification of parameters such as the driver cells, route types and routing attributes.</p>
RouteType	Specifies the routing types to be used.
TopPreferLayer <i>layerName</i>	Specifies the top preferred metal layer to use for routing the nets of the global mesh structure. The name that you specify must be a layer defined in the LEF file.
BottomPreferLayer <i>layerName</i>	Specifies the bottom preferred metal layer to use for routing the nets of the global mesh structure. The name you specify must be a layer defined in the LEF file.
PreferredExtraSpace [0-3]	Specifies the extra space to add around clock wires, when routing the nets in the clock mesh stage.  <i>Default:</i> 0
End	Denotes the end of a stage definition. Each stage definition must begin with a <code>Stage</code> statement and close with an <code>End</code> statement.

### ***The Stage Definition for the Spine Section***

The following table describes the `Stage` definition for the `Spine` section.

Stage	<p>Denotes the beginning of a stage definition. The <code>Stage</code> section must begin with a <code>Stage</code> and close with an <code>End</code> statement.</p> <p>The stage section of a spine section is for specification of parameters such as the number of driver for each spine.</p>
NumDriver	Specifies the number of mesh drivers to be place along the spine.
End	Denotes the end of a stage definition. Each stage definition must begin with a <code>Stage</code> statement, and close with an <code>End</code> statement.

### ***The Mesh Section***

This section is for specification of final mesh wires grid. The following table describes the entries for the mesh section.

Mesh	Denotes the beginning of a mesh definition. The <code>Stage</code> section must begin with a <code>Mesh</code> and close with an <code>End</code> statement. This section is for specifying the number of branches and the route types.
NumBranch <i>number</i>	Specifies the number of branches.  <i>Default:</i> The clock mesh tools computes a suitable value automatically.

<b>BranchPitch</b> <i>distanceValue</i>	<p>Specifies the pitch for the branches for a particular stage.</p> <p><b>Note:</b> The clock mesh tool considers the <code>BranchPitch</code> statement to be a soft constraint that applies to the final mesh stage. The tool might need to make adjustments to the specified pitch in order to meet physical constraints.</p>
<b>RouteType</b>	<p>Specifies the routing types to be used for the final mesh grid. Routing types are defined using the <code>RouteTypeDef</code> statement. There must be two <code>RouteType</code> specifications for the vertical and horizontal wires of the mesh grid.</p>
<b>End</b>	<p>Denotes the end of a <code>Mesh</code> definition. The mesh definition must begin with a <code>Mesh</code> statement, and close with an <code>End</code> statement.</p>

## Defining Attributes of MultiSpine Mesh

To describe the physical attributes of a multi spine mesh style, the basic specification should have the the following parameters or sections defined:

- `MeshType` (parameter, must set to `MultiSpine`)
- `TrunkOrientation` (parameter)
- PreDrive CTS
- MainDrive
- Mesh

The following parameters are optional to the `GlobalMesh` section:

- TrunkPlacement
- TrunkDriveDist

The following parameters are not applicable to `MainDrive` section and `Mesh` section:

- TrunkAttachment
- TrunkPitch
- NumTrunk
- Branch Attachment Frequency
- Trunk Attachment Frequency

## Analysis Section

This section defines how to generate spice run deck.

The following table describes entries for the Analysis section:

Analysis	Denotes the beginning of generating spice run deck specification section.
MultiPartSpice {true   false}	Enables the implementation of multiple spice run deck generation. <i>Default:</i> false
MultiPartSpicePartitionLevel num   Global [+ - num]	Specifies the partition level between global and local portions of the clock network. The partition levels can be relative to the root or to the final global mesh drive(lowest level with a single multi-drive net).
End	Denotes the end of generating spice run deck specification section.
SpiceLib	Denotes the beginning of SpiceLib section.
Library <i>libFile</i> section	Specifies the list of library sections to be loaded with .lib cards.
Include <i>incFile</i>	Specifies the list of files to be included in the Spice netlist with .INCLUDE cards. This parameter is used to include the Spice model files, library files and the .temp statement.
Global [Power   Ground] node	Specifies the list of supply nodes to be created in the sub-circuits rely on global nets.
SubcktPortSeq <i>file</i>	Specifies the list of sub-circuit port order file.
End	Denotes the end of SpiceLib section.

## Top Chain Section

This section defines how the top chain is to be synthesized. A top-level chain is a cascaded driver chain from the mesh root to the first level of mesh pre-driver drivers.

The following table describes the entries for the top chain section:

TopChain	Denotes the beginning of the top chain definition. The top chain definition must begin with a TopChain statement, and close with an End statement.
Enabled [true   false]	Enables the implementation of the top chain. <i>Default:</i> false

DriveCell <i>cellName</i>	Specifies the type of buffer to be used for constructing the top-level chain. Only one cell type can be used for the chain.
NumLevel <i>number</i>	Specifies the number of levels in the chain structure (which is equal to the number of buffers).  You must specify an even number of levels if you use an inverter for the drive cell (DriveCell).
TargetLocs + <i>driverLocationX</i> <i>driverLocationY</i> ...	Positions the top chain drivers at the specified locations.  <b>For example:</b> TargetLocs  + 250um 100um  + 500um 500um  The number of locations you specify must match the number of levels. If they do not match, the clock mesh tool ignores them.  <b>Note:</b> The clock mesh tool considers the TargetLocs statement to be a soft constraint; it attempts to place the drivers at the specified locations, but might deviate a small distance in order to find a solution.
NonDefaultRule <i>ruleName</i>	Specifies the LEF NONDEFAULTRULE statement for the router to use when routing the nets in the top-level chain.  <i>Default:</i> The router uses the default routing rule.
TopPreferLayer <i>layerName</i>	Specifies the top preferred metal layer to use for routing the top-level chain buffers. The name you specify must be a layer defined in the LEF file.
BottomPreferLayer <i>layerName</i>	Specifies the bottom preferred metal layer to use for routing the top-level chain buffers. The name you specify must be a layer defined in the LEF file.
PreferredExtraSpace [0 - 3]	Specifies the extra space to add around clock wires, when routing the nets in the top-level chain.  <i>Default:</i> 0
End	Denotes the end of the top chain definition. The top chain definition must begin with a TopChain statement, and close with an End statement.

<pre>ShieldNet netname</pre>	<p>Specifies the shielding net name for top chain.</p> <p><i>Default:</i> No shielding</p> <p><b>Note:</b> Shield nets using special wires for shielding global mesh nets are created during <code>synthesizeClockMesh</code>.</p> <p><b>Note:</b> Shield nets using regular wires for shielding top chain and local tree nets are created during <code>routeClockMesh</code>.</p>
------------------------------	--

## Local Tree Section

This section defines how the local tree is to be synthesized.

The following table describes the entries for the local tree section:

<pre>LocalTree</pre>	<p>Denotes the beginning of the local tree definition. The local tree definition must begin with a <code>LocalTree</code> statement, and close with an <code>End</code> statement.</p>	
<pre>Enabled [true   false]</pre>	<p>Enables the implementation of the local tree.</p> <p><i>Default:</i> false</p>	
<pre>RootPos {ClusterCenter   OnMesh   NearMeshInCluster}</pre>	<p>Specifies how to place the buffers.</p> <p><i>Default:</i> ClusterCenter</p>	
	<pre>ClusterCenter</pre>	<p>Places buffers at the cluster center of gravity.</p>
	<pre>OnMesh</pre>	<p>Places buffers on or along the nearest mesh trunk or branch. Buffer placement can extend beyond the bounding box of the cluster.</p>
	<pre>NearMeshInCluster</pre>	<p>Places the buffers as close as possible to a mesh trunk or branch without going outside the bounding box of leaves driven by the local root.</p>
<pre>DriveCells + cellName1 + cellName2...</pre>	<p>Specifies the types of drive cells to use in the local tree. Multiple drive cells can be specified. You must specify at least one buffer choice to drive non-gated leaves; the clock mesh tool does not automatically choose a buffer if none is specified.</p> <p>For gated domains, the clock mesh tool chooses an appropriate cell from the list (that is, an LEQ cell to the original gating element). If no LEQ cells are specified, the tool attempts to find choices from the footprint of the original gating cell. If no footprint exists, the tool uses the original gating cell.</p>	
<pre>NonDefaultRule ruleName</pre>	<p>Specifies the LEF <code>NONDEFAULTRULE</code> statement for the router to use when routing the local tree.</p> <p><i>Default:</i> The router uses the default routing rule.</p>	
<pre>TopPreferLayer layerName</pre>	<p>Specifies the top preferred metal layer to use for routing the local tree. The name you specify must be a layer defined in the LEF file.</p>	

BottomPreferLayer <i>layerName</i>	Specifies the bottom preferred metal layer to use for routing the local tree. The name you specify must be a layer defined in the LEF file.
PreferredExtraSpace [0 - 3]	Specifies the extra space to add around clock wires when routing the local tree.  <i>Default:</i> 0
Cluster	Denotes the beginning of a cluster subsection within the local tree definition.  By default, the clock mesh tool performs automatic clustering during the local tree synthesis. You can use a cluster subsection to manually specify how a certain group of leaves can be driven by a driver or a clock gating cell. You can specify multiple clusters, as needed.  The cluster subsection must begin with a <code>Cluster</code> statement, and close with an <code>End</code> statement.
DriveCell <i>cellName</i>	Specifies the type of cell to use to drive the cluster. If you do not specify a cell type, the clock mesh tool automatically chooses one. If you specify a cell type, the cell must be consistent with the leaves of the cluster. For example, you cannot change the logic by specifying an AND gate for leaves that are originally ungated.
TargetLoc <i>driverLocationX</i> <i>driverLocationY</i> ...	Positions the drivers at the specified locations.  <b>Note:</b> The clock mesh tool considers the <code>TargetLoc</code> statement to be a soft constraint; it attempts to place the drivers at the specified locations, but might deviate a small distance in order to find a solution.  <i>Default:</i> The clock mesh tool automatically chooses the local root location.
LeafPin + <i>instanceName/pinName</i>	Specifies the instance pins to use to form a cluster. The set of leaf pins must be consistent (that is, they must all be from the same original domain). Additionally, the same leaf cannot appear in multiple clusters.
End	Denotes the end of the cluster subsection. The cluster subsection must begin with a <code>Cluster</code> statement, and close with an <code>End</code> statement.
End	Denotes the end of the local tree definition. The local tree definition must begin with a <code>LocalTree</code> statement, and close with an <code>End</code> statement.
ShieldNet <i>netname</i>	Specifies the shielding net name for local tree.  <i>Default:</i> No shielding  <b>Note:</b> Shield nets using special wires for shielding global mesh nets are created during <code>synthesizeClockMesh</code> .  <b>Note:</b> Shield nets using regular wires for shielding top chain and local tree nets are created during <code>routeClockMesh</code> .

## Clock Mesh Specification File Example

#Comments in the clock mesh spec file should be preceded with the # character.

```
#Routing Type definition  
#At least two routing types must be specified.
```

```
RouteTypeDef RT1  
  
Layer M5  
  
Width 2  
  
Spacing 0.5  
  
ShieldWidth 0.4  
  
ShieldSpacing 0.4  
  
End
```

```
RouteTypeDef RT2  
  
Layer M6  
  
Width 2  
  
Spacing 0.5  
  
ShieldWidth 0.4  
  
ShieldSpacing 0.4  
  
End  
  
MeshArea 0um 0um 500um 500um  
  
#Cutout definition  
  
#All target locations are absolute values.
```

```
Cutout  
+ 480um 400um 860um 560um  
+ Inst/RAM1/ HALO 30um
```

```
#Clock Mesh definition  
  
ClockMesh clk  
  
#Timing and Power Constraints definition
```

```
Period 1000

SupplyVoltage 1.0

MaxPower 0

RootTrans 400

MinDelay 1000

MaxDelay 1020

MaxSkew 10

MaxBufTrans 400

MaxLeafTrans 400

#Analysis Section

    Analysis

        MultiPartSpice true

        MultiPartSpicePartitionLevel 6

    End

    SpiceLib

        Library spice.lib slow

        Include slow.spicemodel

        Global Power VDD

        Global Ground VSS

        SubcktPortSeq my.spo

    End

#Tracing and Analysis Scope definition

RootPin clk

#LeafPins can be rising or falling.

LeafPin

+ U1/A rising

#LeafCellPins can be rising or falling.
```

```
LeafCellPin  
+ NAND/A rising  
  
#DefaultTrigger can be rising or falling.  
DefaultTrigger rising  
  
#Set AllowGating to true to handle gated clock.  
AllowGating true  
  
#Mesh Structure definition  
  
#UseMeshModule can be true or false  
UseMeshModule true  
  
MeshModule mesh_module  
#GlobalMesh definition  
#####For MultiSpine#####  
#GlobalMesh  
#MeshType MultiSpine  
#MeshDrivePoint Center  
#TrunkOrientation Vertical  
#TrunkDrivePlace LoadWeighted  
#TrunkDriveDist Uniform  
  
#PreDrive CTS  
#Enabled True  
#DriveCells  
#+ BUFX20  
#+ BUFX16  
#NonDefaultRule myrule  
#TopPreferLayer M4  
#BottomPreferLayer M3
```

```
#PreferredExtraSpace 0  
#End  
  
#MainDrive  
#SpineTemplate  
#Stage  
    #RouteType RT6  
    #DriveCell BUFX16  
#End  
#End  
  
#Spine  
#TargetLoc 800um  
#Stage  
    #NumDriver 10  
#End  
#End  
#Spine  
#TargetLoc 1050um  
#Stage  
    #NumDriver 14  
#End  
#End  
#Mesh  
#RouteType RT6  
#RouteType RT7  
#NumBranch 6  
#End  
GlobalMesh  
  
#MeshDrivePoint can be Center, Root, or X,Y.  
  
MeshDrivePoint Center
```

#MeshType can be Fishbone or HTreeMesh or MultiSpine

MeshType HTreeMesh

#TrunkOrientation can be Horizontal or Vertical

TrunkOrientation Horizontal

#Use H and V for HTreePattern. H\* means HVHVHV (alternating pattern).

HTreePattern H\*

#TrunkPlacement can be UniformPitch or LoadWeighted

TrunkPlacement UniformPitch

#TrunkDriveDist can be StrictAttach, Uniform, LoadWeighted, or

#LoadWeightedMatch.

TrunkDriveDist StrictAttach

PatternTrunkClusterTargetSize 1

#CTS pre-drive structure definition

PreDriveCTS

#Set Enabled to true to implement top chain.

Enabled true

DriveCells

+ CLKBUFX20

+ CLKBUFX16

NonDefaultRule NDR1

TargetInputSkewSlewRatio 0.5

```
optAddBuffer true  
optAddBufferLimit 100  
DummyBuffer + cell1 + cell2  
TopPreferlayer M5  
BottomPreferLayer M4  
PreferredExtraSpace 1  
  
#Marks the end of the CTS pre-drive structure definition  
End  
  
#Definition for mesh Stage 1  
Stage  
NumDriver 1  
X 4  
DriveCell CLKBUFX20  
RouteTypePair RT1 RT2  
ShieldNet VSS  
End  
  
#Definition for mesh Stage 2  
Stage  
NumDriver 4  
X 2  
DriveCell CLKBUFX20  
RouteTypePair RT1 RT2
```

ShieldNet VSS

End

#Definition for mesh Stage 3 (final stage)

Stage

NumDriver 8

X 2

DriveCell CLKBUFX20

RouteTypePair RT1 RT2

NumTrunk 2

NumBranch 4

#TrunkPitch 50

#BranchPitch 20

#TrunkAttachFrequency 2

#BranchAttachFrequency 2

#All target locations in clock mesh spec file are absolute values.

#TargetTrunkLocs

#+ 500um

#+ 1200um

ShieldNet VSS

End

End

#Top Chain definition (this section is optional)

TopChain

#Set Enabled to true to implement top chain.

Enabled true

```
DriveCell CLKBUFX20

NumLevel 2

#NonDefaultRule NDR1

#TargetLocs

TopPreferlayer M5

BottomPreferLayer M4

PreferredExtraSpace 1

ShieldNet VSS

#Marks the end of the Top Chain definition.

End

#Local Tree definition (this section is optional)

LocalTree

#Set Enabled to true to implement local tree synthesis

Enabled true

#RootPos can be ClusterCenter, OnMesh, or NearMeshInCluster.

RootPos ClusterCenter

DriveCells

+ CLKBUFX20

+ CLKBUFX16

+ CLKBUFX12

#NonDefaultRule NDR2

TopPreferLayer M4

BottomPreferlayer M3

PreferredExtraSpace 1

#Manual Cluster definition (this subsection is optional)
```

Cluster

#DriveCell can be buffer or gated cell.

DriveCell BUFX16

#TargetLoc 300um 600um

LeafPin

+ FF1/CK

+ FF2/CK

+ FF9/CK

#Marks the end of the cluster definition

End

#Marks the end of the local tree definition

End

#Marks the end of the clock mesh defintion

End

=====

## Supported CPF 1.0 Commands

**Note:** The following commands are supported unless otherwise noted.

Command Name	Option	Notes
		N/A = not available in this release
create_analysis_view		
	-name	
	-mode	

	-domain_corners	
create_bias_net		
	-net	
	-driver	N/A
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_ground_nets		
	-nets	
	-voltage	N/A
	-internal	N/A
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_isolation_rule		
	-name	
	-isolation_condition	
	-pins	
	-from	
	-to	
	-isolation_target	N/A
	-isolation_output	
	-exclude	

create_level_shifter_rule		
	-name	
	-pins	
	-from	
	-to	
	-exclude	
create_mode_transition		N/A
	-start_condition	
create_nominal_condition		
	-name	
	-voltage	
	-pmos_bias_voltage	N/A
	-nmos_bias_voltage	N/A
create_operating_corner		
	-name	
	-voltage	
	-process	
	-temperature	
	-library_set	
create_power_domain		
	-name	
	-default	
	-instances	
	-boundary_ports	
	-shutoff_condition	
	-default_restore_edge	
	-default_save_edge	
	-power_up_states	N/A

create_power_mode		
	-name	
	-domain_conditions	
	-default	
create_power_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-internal	
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_power_switch_rule		
	-name	
	-domain	
	-external_power_net	
	-external_ground_net	
create_state_retention_rule		
	-name	
	-domain	
	-instances	
	-restore_edge	
	-save_edge	
define_always_on_cell		
	-cells	
	-power_switchable	
	-power	
	-ground_switchable	
	-ground	
define_isolation_cell		

	-cells	
	-library_set	
	-always_on_pin	
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
	-valid_location	
	-non_dedicated	N/A
	-enable	
define_level_shifter_cell		
	-cells	
	-library_set	
	-always_on_pin	N/A
	-input_voltage_range	
	-output_voltage_range	
	-direction	
	-output_voltage_input_pin	N/A
	-input_power_pin	
	-output_power_pin	
	-ground	
	-valid_location	
define_open_source_input_pin		
	-cells	
	-pin	
	-library_set	
define_power_clamp_cell		N/A
	-cells	

	-data	
	-ground	
	library_set	
	-power	
define_power_switch_cell		
	-cells	
	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-power_switchable	
	-power	
	-ground	
	-ground_switchable	
	-on_resistance	Accessible by ::CPF::getCpfPsoCell
	-stage_1_saturation_current	Accessible by ::CPF::getCpfPsoCell
	-stage_2_saturation_current	Accessible by ::CPF::getCpfPsoCell
	-leakage_current	Accessible by ::CPF::getCpfPsoCell
define_state_retention_cell		
	-cells	
	-library_set	
	-always_on_pin	N/A
	-clock_pin	N/A
	-restore_function	

	-restore_check	N/A
	-save_function	
	-save_check	N/A
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_library_set		
	-name	
	-libraries	
end_design		
identify_always_on_driver		N/A
identify_power_logic		
	-type	
	-instances	
set_array_naming_style		
set_cpf_version		
set_design		
	-ports	
	module	
set_hierarchy_separator		
set_instance		
	-port_mapping	
	-merge_default_domains	
	hier_instance	
set_power_target		N/A
set_power_unit		N/A
set_register_naming_style		

set_switching_activity		
	-all	
	-pins	
	-instances	
	-hierarchical	
	-probability	
	-toggle_rate	
	-clock_pins	N/A
	-toggle_percentage	N/A
	-mode	N/A
set_time_unit		
update_isolation_rules		
	-names	'
	-location	
	-cells	
	-library_set	
	-prefix	
	-combine_level_shifting	N/A
	-open_source_pins_only	
-update_level_shifter_rules		
	-names	
	-location	
	-cells	
	-library_set	
	-prefix	
update_nominal_condition		
	-name	
	-library_set	

update_power_domain		
	-name	
	-internal_power_net	
	-internal_ground_net	
	-min_power_up_time	N/A
	-max_power_up_time	N/A
	-pmos_bias_net	N/A
	-nmos_bias_net	N/A
	-user_attributes	Accessible by ::CPF::getCpfUserAttr
	-rail_mapping	N/A
	-library_set	
update_power_mode		
	-name	
	-activity_file	N/A
	-activity_file_weight	N/A
	-sdc_files	
	-peak_ir_drop_limit	N/A
	-average_ir_dropt_limit	N/A
	-leakage_power_limit	N/A
	-dynamic_power_limit	N/A
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-library_set	
	-prefix	

	-peak_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
	-average_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
update_state_retention_rules		
	-name	
	-cell_type	
	-cell	
	-library_set	

## CPF 1.0 Script Example

The following section contains an example of the CPF 1.0 file using a sample design and library.

For list of supported CPF commands and options within Innovus product family, see "[Supported CPF 1.0 Commands](#)".

```
set_cpf_version 1.0

#####
# #
# Technology portion of the CPF: #
# Defining the special cells for low-power designs #
# #

#####

##### High-to-Low level shifters #####
#####



define_level_shifter_cell -cells LVLH2L* \

```

```
-output_power_pin VDD \
-ground VSS \
-valid_location to

#####
#####[ Always-on High-to-low level shifters
#####

define_level_shifter_cell -cells AOLVLH2L* \
-input_voltage_range 0.8:1.0:0.1 \
-output_voltage_range 0.8:1.0:0.1 \
-direction down \
-output_power_pin TVDD \
-ground VSS \
-valid_location to

#####
#####[ Low-to-High Level Shifters
#####

define_level_shifter_cell -cells LVLL2H* \
-input_voltage_range 0.8:1.0:0.1 \
-output_voltage_range 0.8:1.0:0.1 \
-input_power_pin VDDI \
-output_power_pin VDD \
-direction up \
-ground VSS \
-valid_location to

#####
#####[ Low-to-High level shifting plus isolation combo cells
#####

define_level_shifter_cell -cells LVLCIL2H* \
-input_voltage_range 0.8:1.0:0.1 \
```

```
-output_voltage_range 0.8:1.0:0.1 \
-output_voltage_input_pin ISO \
-input_power_pin VDDI \
-output_power_pin VDD \
-direction up \
-ground VSS \
-valid_location to

#####
##### Isolation cells
#####

define_isolation_cell -cells LVLCIL2H* \
-power VDD \
-ground VSS \
-enable ISO \
-valid_location to

#####
##### Power switch cells: headers
#####

define_power_switch_cell -cells {HEADERHVT HEADERAOPHVT} \
-power_switchable VDD -power TVDD \
-stage_1_enable !ISOIN1 \
-stage_1_output ISOOUT1 \
-stage_2_enable !ISOIN2 \
-stage_2_output ISOOUT2 \
-type header

#####
##### SRPG cells
#####
```

```
define_state_retention_cell -cells { SRPG2Y } \
-clock_pin CLK \
-power TVDD \
-power_switchable VDD \
-ground VSS \
-save_function "SAVE" \
-restore_function "!NRESTORE"

#####
##### Always-on cells: buffers and level shifters
#####

define_always_on_cell -cells {AOBUFF2Y AOLVLH2L*} \
-power_switchable VDD -power TVDD -ground VSS
#####

# #
# Design part of the CPF #
# #

set_design top

set_hierarchy_separator "/"

set constraintDir ../CONSTRAINTS

set libdir ../LIBS

#####
##### create the power and ground nets in this design
#####

### VDD will connect the power follow-pin of the instances in the always-on
#power domain
```

```
### VDD_core_SW will connect the power follow-pin of the instances in the
#switchable power domain and is the power net that can be shut-off
### VDD_core_AO is the always-on power net for the switchable power domain

create_power_nets -nets VDD -voltage {0.8:1.0:0.1}
create_power_nets -nets VDD_core_AO -voltage 0.8
create_power_nets -nets VDD_core_SW -internal -voltage 0.8
create_power_nets -nets AVDD -voltage 1.0

create_ground_nets -nets VSS
create_ground_nets -nets AVSS

#####
##### Creating three power domains:
##### AO is the default always-on power domain
##### CORE is the switchable power domain
##### PLL is another always-on power domain
##### Also specifying the power net-pin connection in each power domain
#####

#####
### For power domain "AO"
#####

create_power_domain -name AO -default
update_power_domain -name AO -internal_power_net VDD

create_global_connection -domain AO -net VDD -pins VDD
create_global_connection -domain AO -net VSS -pins VSS
create_global_connection -domain AO -net VDD_core_SW -pins VDDI

#####
### For power domain "CORE"
#####


```

```
create_power_domain -name core -instances CORE_INST \
-shutoff_condition {PWR_CONTROL/power_switch_enable}
update_power_domain -name core -internal_power_net VDD_core_SW

create_global_connection -domain CORE -net VSS -pins VSS
create_global_connection -domain CORE -net VDD_core_AO -pins TVDD
create_global_connection -domain CORE -net VDD_core_SW -pins VDD

#####
### For power domain "PLL"
#####

### PLL contains a single PLL macro and five top-level boundary ports which
#connect to the PLL macro directly

create_power_domain -name PLL -instances PLLCLK_INST -boundary_ports \
{refclk vcom vcop ibias pllrst}
update_power_domain -name PLL -internal_power_net AVDD

create_global_connection -domain PLL -net AVDD -pins avdd!
create_global_connection -domain PLL -net AVSS -pins agnd!
create_global_connection -domain PLL -net VDD -pins VDDI
create_global_connection -domain PLL -net AVDD -pins VDD
create_global_connection -domain PLL -net AVSS -pins VSS

#####
####

set lib_1p1_wc "$libdir/technology45_std_1p1.lib"
set lib_1p3_bc "$libdir/technology45_std_1p3.lib"
set lib_1p0_bc "$libdir/technology45_std_1p0.lib"
set lib_0p8_wc "$libdir/technology45_std_0p8.lib"
```

```
set lib_ao_wc_extra "\$libdir/technology45_lv112h_1p1.lib "
"
set lib_ao_bc_extra "\$libdir/technology45_lv112h_1p3.lib "
"
set lib_core_wc_extra "\$libdir/technology45_lv1h21_0p8.lib \
\$libdir/technology45_headers_0p8.lib \
\$libdir/technology45_sprg_ao_0p8.lib "
"
set lib_core_bc_extra "\$libdir/technology45_lv1h21_1p0.lib \
\$libdir/technology45_headers_1p0.lib \
\$libdir/technology45_sprg_ao_1p0.lib "
"

set lib_pll_wc "\$libdir/pll_slow.lib \
\$libdir/ram_256x16_slow.lib \
\$libdir/rom_512x16_slow.lib "
"
set lib_pll_bc "\$libdir/pll_fast.lib \
\$libdir/ram_256x16_fast.lib \
\$libdir/rom_512x16_fast.lib "
"
#####
#### Define library sets
#####
define_library_set -name ao_wc_0p8 -libraries "$lib_0p8_wc \$lib_ao_wc_extra"
define_library_set -name ao_bc_1p0 -libraries "$lib_1p0_bc \$lib_ao_bc_extra"

define_library_set -name ao_wc_1p1 -libraries "$lib_1p1_wc_base \$lib_ao_wc_extra"
```

```
define_library_set -name ao_bc_1p3 -libraries "$lib_1p3_bc_base $lib_ao_bc_extra"

define_library_set -name core_wc_0p8 -libraries "$lib_0p8_wc $lib_core_wc_extra"
define_library_set -name core_bc_1p0 -libraries "$lib_1p0_bc $lib_core_bc_extra"

define_library_set -name pll_wc_1p1 -libraries "$lib_pll_wc"
define_library_set -name pll_bc_1p3 -libraries "$lib_pll_bc"

#####
##### Create operating corners
#####

create_operating_corner -name BC_PVT_AO_L \
    -process 1 -temperature 0 -voltage 1.0 \
    -library_set ao_bc_1p0

create_operating_corner -name WC_PVT_AO_L \
    -process 1 -temperature 125 -voltage 0.8 \
    -library_set ao_wc_0p8

create_operating_corner -name BC_PVT_AO_H \
    -process 1 -temperature 0 -voltage 1.3 \
    -library_set ao_bc_1p3

create_operating_corner -name WC_PVT_AO_H \
    -process 1 -temperature 125 -voltage 1.1 \
    -library_set ao_wc_1p1

create_operating_corner -name BC_PVT_CORE \
    -process 1 -temperature 0 -voltage 1.0 \
    -library_set core_bc_1p0

create_operating_corner -name WC_PVT_CORE \
    -process 1 -temperature 125 -voltage 0.8 \
```

```
-library_set tdsp_wc_0p8

create_operating_corner -name BC_PVT_PLL \
-process 1 -temperature 0 -voltage 1.3 \
-library_set core_bc_1p3

create_operating_corner -name WC_PVT_PLL \
-process 1 -temperature 125 -voltage 1.1 \
-library_set tdsp_wc_1p1

#####
#### Create and update nominal conditions
#####

create_nominal_condition -name high_ao -voltage 1.1
update_nominal_condition -name high_ao -library_set ao_wc_1p1

create_nominal_condition -name low_ao -voltage 0.8
update_nominal_condition -name low_ao -library_set ao_wc_0p8

create_nominal_condition -name low_core -voltage 0.8
update_nominal_condition -name low_core -library_set core_wc_0p8

create_nominal_condition -name high_pll -voltage 1.1
update_nominal_condition -name high_pll -library_set pll_wc_1p1

create_nominal_condition -name off -voltage 0

#####
#### Create and upDate four power modes
#####


```

```
create_power_mode -name PM_HL_FUNC \
-domain_conditions {AO@high_ao CORE@low_core PLL@high_pll} \
-default
update_power_mode -name PM_HL_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_HL_TEST \
-domain_conditions {AO@high_ao CORE@low_core PLL@high_pll}
update_power_mode -name PM_HL_TEST -sdc_files ${constraintDir}/top_test.sdc

create_power_mode -name PM_HO_FUNC \
-domain_conditions {AO@high_ao CORE@off PLL@high_pll}
update_power_mode -name PM_HO_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_LO_FUNC \
-domain_conditions {AO@low_ao CORE@off PLL@high_pll}
update_power_mode -name PM_LO_FUNC -sdc_files ${constraintDir}/top_slow.sdc

#####
##### Creating ten analysis views
#####

create_analysis_view -name AV_HL_FUNC_MIN_RC1 -mode PM_HL_FUNC \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MIN_RC2 -mode PM_HL_FUNC \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}

create_analysis_view -name AV_HL_FUNC_MAX_RC1 -mode PM_HL_FUNC \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MAX_RC2 -mode PM_HL_FUNC \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_HL_SCAN_MIN_RC1 -mode PM_HL_TEST \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_SCAN_MAX_RC1 -mode PM_HL_TEST \
```

```
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_HO_FUNC_MIN_RC1 -mode PM_HO_FUNC \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HO_FUNC_MAX_RC1 -mode PM_HO_FUNC \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_LO_FUNC_MIN_RC1 -mode PM_LO_FUNC \
-domain_corners {AO@BC_PVT_AO_L CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_LO_FUNC_MAX_RC1 -mode PM_LO_FUNC \
-domain_corners {AO@WC_PVT_AO_L CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

#####
#### Creating and updating the rules for the insertion
#### of power switch, level shifter, isolation cell
#####

#####
#### One power switch rule
#####

create_power_switch_rule -name PWRSW_CORE -domain CORE \
-external_power_net VDD_core_AO

update_power_switch_rule -name PWRSW_CORE \
-cells HEADERHVT \
-prefix CDN_SW_ \
-acknowledge_receiver SIWTCH_ENOUT
#####

#### One isolation rule using level-shifting and isolation combo cells
#####

create_isolation_rule -name ISORULE -from CORE \
-isolation_condition "!PWR_CONTROL/isolation_enable" \
```

```
-isolation_output high

update_isolation_rules -names ISORULE -location to -cells LVLCIL2H2Y

#####
##### Three level shifting rules
#####

##### For signals from AO to CORE

create_level_shifter_rule -name LSRULE_H2L -from AO -to CORE \
-exclude {PWR_CONTROL/power_switch_enable PWR_CONTROL \
/state_retention_enable PWR_CONTROL/state_retention_restore}
update_level_shifter_rules -names LSRULE_H2L -cells LVLH2L2Y -location to

##### Only for the control signals from AO to CORE

create_level_shifter_rule -name LSRULE_H2L_AO -from AO -to CORE \
-pins {PWR_CONTROL/power_switch_enable PWR_CONTROL/state_retention_enable\
PWR_CONTROL/state_retention_restore}
update_level_shifter_rules -names LSRULE_H2L_AO -cells AOLVLH2L2Y -location to

##### For signals from PLL to AO

create_level_shifter_rule -name LSRULE_H2L_PLL -from PLL -to AO
update_level_shifter_rules -names LSRULE_H2L_PLL -cells LVLH2L2Y -location to

#####
##### One SRPG rule
#####

create_state_retention_rule -name SRPG_CORE \
-domain CORE \
-restore_edge {!PWR_CONTROL/state_retention_restore} \
```

```
-save_edge {PWR_CONTROL/state_retention_enable}
```

```
update_state_retention_rules -names SRPG_CORE \
```

```
-cell SRPG2Y \
```

```
-library_set tdsp_wc_0v792
```

```
end_design
```

## Supported CPF 1.0e Commands

**Note:** The following commands and options are supported unless otherwise noted.

Command Name	Option	Notes
create_analysis_view		
	-name	
	-mode	
	-domain_corners	
	-group_views	
	-user_attributes	
create_assertion_control		
	-name	Unsupported
	-assertions	Unsupported
	-domains	Unsupported
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-net	
	-driver	

	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_power_domain		
	-name	
	-instances	
	-boundary_ports	
	-default	
	-shutoff_condition	
	-external_controlled_shutoff	
	-default_isolation_condition	
	-default_restore_edge	
	-default_save_edge	
	-default_restore_level	Supported
	-default_save_level	Supported
	-power_up_states	Unsupported
	-active_state_condition	Unsupported
	-secondary_domains	
create_ground_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	

	-average_ir_drop_limit	
create_isolation_rule		
	-name	
	-isolation_condition	
	-no_condition	Unsupported
	-pins	
	-from	
	-to	
	-exclude	
	-isolation_target	
	-isolation_output	
	-secondary_domain	
create_level_shifter_rule		
	-name	
	-pins	
	-from	
	-to	
	-exclude	
create_mode_transition		
	-name	
	-from	
	-to	
	-start_condition	
	-end_condition	
	-cycles	
	-clock_pin	
	-latency	
create_nominal_condition		

	-name	
	-voltage	
	-ground_voltage	
	-state	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
create_operating_corner		
	-name	
	-voltage	
	-ground_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
	-process	
	-temperature	
	-library_set	
create_power_mode		
	-name	
	-default	
	-group_modes	
	-domain_conditions	
create_power_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_power_switch_rule		

	-name	
	-domain	
	-external_power_net	
	-external_ground_net	
create_state_retention_rule		
	-name	
	-domain	
	-instances	
	-exclude	
	-restore_edge	
	-save_edge	
	-restore_precondition	
	-save_precondition	
	-target_type	
	-secondary_domain	
define_always_on_cell		
	-cells	
	-library_set	
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_isolation_cell		
	-cells	
	-library_set	
	-always_on_pins	
	-power_switchable	
	-ground_switchable	

	-power	
	-ground	
	-valid_location	
	-enable	
	-no_enable	Unsupported
	-non_dedicated	
define_level_shifter_cell		
	-cells	
	-library_set	
	-always_on_pins	
	-input_voltage_range	
	-output_voltage_range	
	-ground_output_voltage_range	Unsupported
	-groung_output_voltage_range	
	-direction	
	-input_power_pin	
	-output_power_pin	
	-input_ground_pin	Unsupported
	-output_ground_pin	Unsupported
	-ground	
	-power	Unsupported
	-enable	
	-valid_location	
define_library_set		
	-name	
	-libraries	
	-user_attributes	cdb: specify cdb libraries for the library set aocv: specify aocv libraries for the library set
define_power_clamp_cell		

	-location	Unsupported
	-within_hierarchy	Unsupported
	-cells	Unsupported
	-prefix	Unsupported
define_power_switch_cell		
	-cells	
	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-power_switchable	
	-power	
	-ground_switchable	
	-ground	
	-on_resistance	Supported (for use with addPowerSwitch)
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-leakage_current	Supported (for use with addPowerSwitch)
define_state_retention_cell		
	-cells	
	-library_set	
	-cell_type	
	-always_on_pins	
	-clock_pin	
	-restore_function	

	-save_function	
	-restore_check	
	-save_check	
	-always_on_components	Unsupported
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
end_macro_model		
end_power_mode_control_group		
get_parameter		
include		
identify_power_logic		
	-type	Only "isolation" is supported for the -type
	-instances	Supported
	-module	Supported
set_array_naming_style		
set_cpf_version		
set_hierarchy_separator		
set_design		
	-ports	
	-honor_boundary_port_domain	
	-parameters	
set_equivalent_control_pins		
	-master	Unsupported
	-pins	Unsupported
	-domain	Unsupported
	-rules	Unsupported

set_floating_ports		
set_input_voltage_tolerance		
	-ports	Unsupported
	-bias	Unsupported
set_instance		
	-design	
	-model	
	-port_mapping	
	-domain_mapping	
	-parameter_mapping	
set_macro_model		
set_power_mode_control_group		
	-name	
	-domains	
	-groups	Unsupported
set_power_target		
	-leakage	Unsupported
	-dynamic	Unsupported
set_power_unit		
set_register_naming_style		
set_switching_activity		
	-all	Supported
	-pins	Supported
	-instances	Supported
	-hierarchical	Supported
	-probability	Supported
	-toggle_rate	Supported
	-clock_pins	Unsupported

	-toggle_percentage	Unsupported
	-mode	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
	-open_source_pins_only	Supported
update_level_shifter_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-name	
	-primary_power_net	
	-primary_ground_net	
	-pmos_bias_net	Unsupported
	-nmos_bias_net	Unsupported
	-user_attributes	Supported: query getCPFUserAttributes
	-transition_slope	Unsupported

**Innovus User Guide**  
**Syntax and Scripts--CPF 1.0e Script Example**

---

	-transition_latency	Unsupported
	-transition_cycles	Unsupported
update_power_mode		
	-name	
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-sdc_files	
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
	-leakage_power_limit	Unsupported
	-dynamic_power_limit	Unsupported
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-gate_bias_net	Unsupported
	-prefix	
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
update_state_retention_rules		
	-names	
	-cell_type	
	-cells	
	-set_rest_control	Unsupported

## CPF 1.0e Script Example

The following section contains an example of the CPF 1.0e file using a sample design and library.

For list of supported CPF commands and options within Innovus product family, see "[Supported CPF 1.0e Commands](#)".

```
#-----
# setting
#-----
set_cpf_version 1.0e
set_hierarchy_separator /



#-----
# define library_set/cells
#-----
define_library_set -name wc_0v81 -libraries { \
..../LIBS/timing/library_wc_0v81.lib }

define_library_set -name bc_0v81 -libraries { \
..../LIBS/timing/library_bc_0v81.lib }

define_library_set -name wc_0v72 -libraries { \
..../LIBS/timing/library_wc_0v72.lib }

define_library_set -name bc_0v72 -libraries { \
..../LIBS/timing/library_bc_0v72.lib }



define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
VDD -power TVDD -ground VSS

define_isolation_cell -cells { LVLLH* } -power VDD -ground VSS -enable \
NSLEEP -valid_location to

define_isolation_cell -cells { ISOHID* ISOLOD* } -power VDD -ground VSS \
-enable ISO -valid_location to


define_level_shifter_cell -cells { LVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin VDD -ground VSS -valid_location to
```

```
define_level_shifter_cell -cells { PTLVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHCD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \

```

```
create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-----
# create power domains
#-----
```

```
create_power_domain -name PDdefault -default

create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifsip \
IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
-external_controlled_shutoff -shutoff_condition io_shutoff_ack

create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
IOPADS_INST/Pbiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip \
IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } -boundary_ports { ibias reset \
refclk vcom vcop pllrst }

create_power_domain -name PDram_virtual

create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \
-shutoff_condition !INST/PM_INST/power_switch_enable \
-secondary_domains { PDram_virtual }

create_power_domain -name PDtdsp -instances { INST/RAM_128x16_TEST_INST1 \
INST/DSP_CORE_INST0 INST/DSP_CORE_INST1 } -shutoff_condition \
!INST/PM_INST/power_switch_enable -secondary_domains { PDdefault }

#-----
# set instances
#-----

set_instance INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -domain_mapping \
{ {RAM_DEFAULT PDtdsp} }

set_macro_model ram_256x16A

create_power_domain -name RAM_DEFAULT -boundary_ports { A* D* CLK CEN WEN Q* } \
-default -external_controlled_shutoff

create_state_retention_rule -name RAM_ret -instances { mem* } -save_edge !CLK

update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
-primary_ground_net VSS

end_macro_model
```

```
#-----
# create power modes
#-----

create_power_mode -name PMdvfs1 -default -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDTdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs1_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDshutoff_io@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs1_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2 -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDTdsp@nom_0v72 PDram@nom_0v72 PDshutoff_io@nom_0v72 \
PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDshutoff_io@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMscan -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDTdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }

create_analysis_view -name AV_dvfs1_BC -mode PMdvfs1 -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDTdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_WC -mode PMdvfs1 -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDTdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_off_BC -mode PMdvfs1_off -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_off_WC -mode PMdvfs1_off -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_shutoffio_off_BC -mode \
PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
PDdefault@PMdvfs1_bc }
```

```
create_analysis_view -name AV_dvfs1_shutoffio_off_WC -mode \
PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
PDdefault@PMdvfs1_wc }

create_analysis_view -name AV_dvfs2_BC -mode PMdvfs2 -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDtdsp@PMdvfs2_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs2_bc }

create_analysis_view -name AV_dvfs2_WC -mode PMdvfs2 -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDtdsp@PMdvfs2_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs2_wc }

create_analysis_view -name AV_PMdvfs2_off_BC -mode PMdvfs2_off -domain_corners \
{ PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDshutoff_io@PMdvfs2_bc }

create_analysis_view -name AV_PMdvfs2_off_WC -mode PMdvfs2_off -domain_corners \
{ PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDshutoff_io@PMdvfs2_wc }

create_analysis_view -name AV_dvfs2_shutoffio_off_BC -mode \
PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
PDdefault@PMdvfs2_bc }

create_analysis_view -name AV_dvfs2_shutoffio_off_WC -mode \
PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
PDdefault@PMdvfs2_wc }

create_analysis_view -name AV_scan_BC -mode PMscan -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDtdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_scan_WC -mode PMscan -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDtdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs1_wc }

#-----
# create rules
#-----

create_power_switch_rule -name PDram_SW -domain PDram -external_power_net VDDL
create_power_switch_rule -name PDtdsp_SW -domain PDtdsp -external_power_net \
VDD

create_isolation_rule -name ISORULE1 -isolation_condition { \
```

```
!INST/PM_INST/isolation_enable } -from { PDtdsp } -to { PDdefault } \
-isolation_target from -isolation_output high
create_isolation_rule -name ISORULE3 -isolation_condition { \
!INST/PM_INST/isolation_enable } -from { PDram } -to { PDdefault } \
-isolation_target from -isolation_output high
create_isolation_rule -name ISORULE4 -isolation_condition { \
!INST/PM_INST/spi_ip_isolate } -from { PDshutoff_io } \
-isolation_target from -isolation_output low

create_level_shifter_rule -name LSRULE_H2L3 -from { PDdefault } -to { PDram } \
-exclude { INST/PM_INST/power_switch_enable }
create_level_shifter_rule -name LSRULE_H2L_PLL -from { PDpll }
create_level_shifter_rule -name LSRULE_L2H2 -from { PDram } -to { PDdefault }

create_state_retention_rule -name \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -instances { \
INST/DSP_CORE_INST0 } -save_edge !INST/DSP_CORE_INST0/clk
create_state_retention_rule -name \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -instances { \
INST/DSP_CORE_INST1 } -restore_edge \
!INST/PM_INST/state_retention_restore -save_edge \
INST/PM_INST/state_retention_save

#-----
# update domains/modes
#-----
update_nominal_condition -name nom_0v81 -library_set wc_0v81
update_nominal_condition -name nom_0v72 -library_set wc_0v72

update_power_domain -name PDdefault -primary_power_net VDD -primary_ground_net \
VSS
update_power_domain -name PDshutoff_io -primary_power_net VDD_IO \
-primary_ground_net VSS
update_power_domain -name PDpll -primary_power_net Avdd -primary_ground_net \
```

Avss

```
update_power_domain -name PDram_virtual -primary_power_net VDDL \
-primary_ground_net VSS
update_power_domain -name PDram -primary_power_net VDDL_sw -primary_ground_net \
VSS
update_power_domain -name PDtdsp -primary_power_net VDD_sw -primary_ground_net \
VSS
```

```
update_power_mode -name PMdvfs1 -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_off -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs2 -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_off -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMscan -sdc_files ../RELEASE/mmmc/scan.sdc
```

```
#-----
```

```
# update rules
#
#-----
```

```
update_power_switch_rule -name PDram_SW -cells { HEADERHVT1 } -prefix \
CDN_SW_RAM -peak_ir_drop_limit 0 -average_ir_drop_limit 0
update_power_switch_rule -name PDtdsp_SW -cells { HEADERHVT2 } -prefix \
CDN_SW_TDSP -peak_ir_drop_limit 0 -average_ir_drop_limit 0
```

```
update_isolation_rules -names ISORULE1 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE3 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE4 -location to -prefix CPF_ISO_
```

```
update_level_shifter_rules -names LSRULE_H2L3 -location to -cells { LVLHLD* \
} -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_H2L_PLL -location to -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_L2H2 -location to -prefix CPF_LS_
```

```
update_state_retention_rules -names \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -cell_type master_slave
update_state_retention_rules -names \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -cell_type balloon_latch

#-----
# end
-----end_design
```

## Supported CPF 1.1 Commands

**Note:** The following commands and options are supported unless otherwise noted.

Command Name	Option	Notes
asset_illegal_domain_configurations		
	-domain_conditions	
	-group_modes	
	-name	
create_analysis_view		
	-domain_corners	
	-group_views	
	-mode	
	-name	
	-user_attributes	
create_assertion_control		
	-assertions	Unsupported
	-domains	Unsupported

	-exclude	
	-name	Unsupported
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-average_ir_drop_limit	
	-driver	
	-net	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
create_global_connection		
	-domain	
	-instances	
	-net	
	-pins	
create_ground_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_isolation_rule		
	-exclude	
	-from	
	-isolation_condition	
	-isolation_output	

	-isolation_target	
	-name	
	-no_condition	
	-pins	
	-secondary_domain	
	-to	
	-force	
create_level_shifter_rule		
	-exclude	
	-from	
	-name	
	-pins	
	-to	
	-force	
create_mode_transition		
	-clock_pin	
	-cycles	
	-end_condition	
	-from	
	-latency	
	-name	
	-to	
	-start_condition	
create_nominal_condition		
	-ground_voltage	
	-name	
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported

	-state	Unsupported
	-voltage	
create_operating_corner		
	-ground_voltage	Unsupported
	-library_set	
	-name	
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-process	
	-temperature	
	-voltage	
create_power_domain		
	-active_state_condition	Unsupported
	-base_domains	
	-boundary_ports	
	-default	
	-default_isolation_condition	
	-default_restore_edge	
	-default_restore_level	Supported
	-default_save_edge	
	-default_save_level	Supported
	-external_controlled_shutoff	
	-instances	
	-name	
	-power_up_states	Unsupported
	-shutoff_condition	
create_power_mode		
	-default	

	-domain_conditions	
	-group_modes	
	-name	
create_power_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_power_switch_rule		
	-domain	
	-external_ground_net	
	-external_power_net	
	-name	
create_state_retention_rule		
	-domain	
	-exclude	
	-instances	
	-name	
	-restore_edge	
	-restore_precondition	
	-save_edge	
	-save_precondition	
	-secondary_domain	
	-target_type	
define_always_on_cell		

	-cells	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
define_isolation_cell		
	-always_on_pins	
	-cells	
	-enable	
	-ground	
	-ground_switchable	
	-library_set	
	-no_enable	
	-non_dedicated	
	-power	
	-power_switchable	
	-valid_location	
define_level_shifter_cell		
	-always_on_pins	
	-cells	
	-direction	
	-enable	
	-ground	
	-ground_input_voltage_range	
	-ground_output_voltage_range	
	-input_ground_pin	
	-input_power_pin	
	-input_voltage_range	

	-library_set	
	-output_ground_pin	
	-output_power_pin	
	-output_voltage_range	
	-power	
	-valid_location	
define_library_set		
	-libraries	
	-name	
	-user_attributes	cdb: specify cdb libraries for the library set aocv: specify aocv libraries for the library set
define_power_clamp_cell		
	-cells	Unsupported
	-data	Unsupported
	-power pin	Unsupported
	-ground	Unsupported
	-library_set	
define_power_switch_cell		
	-cells	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-ground	
	-ground_switchable	
	-leakage_current	Supported (for use with addPowerSwitch)
	-library_set	
	-power	
	-power_switchable	

	-stage_1_on_resistance	
	-stage_2_on_resistance	
	-stage_1_enable	
	-stage_1_output	
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_enable	
	-stage_2_output	
	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-type	
define_state_retention_cell		
	-always_on_components	Unsupported
	-always_on_pins	
	-cell_type	
	-cells	
	-clock_pin	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
	-restore_check	
	-restore_function	
	-save_check	
	-save_function	
end_design		
end_macro_model		
end_power_mode_control_group		

get_parameter		
include		
identify_power_logic		
	-instances	Supported
	-module	Supported
	-type	Only "isolation" is supported for the -type
set_array_naming_style		
set_cpf_version		
set_hierarchy_separator		
set_design		
	module	
	-ports	
	-honor_boundary_port_domain	
	-parameters	
set_equivalent_control_pins		
	-domain	Unsupported
	-master	Unsupported
	-pins	Unsupported
	-rules	
set_floating_ports		
set_input_voltage_tolerance		
	-bias	Unsupported
	-ports	Unsupported
set_instance		
	-design	
	-domain_mapping	
	-model	
	-parameter_mapping	

	-port_mapping	
set_macro_model		
set_power_mode_control_group		
	-domains	
	-groups	Unsupported
	-name	
set_power_target		
	-dynamic	Unsupported
	-leakage	Unsupported
set_power_unit		
set_register_naming_style		
set_switching_activity		
	-all	Supported
	-clock_pins	Unsupported
	-hierarchical	Supported
	-instances	Supported
	-mode	Supported
	-pins	Supported
	-probability	Supported
	-toggle_percentage	Unsupported
	-toggle_rate	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-cells	
	-location	
	-names	
	-open_source_pins_only	Supported
	-prefix	

	-within_hierarchy	
update_level_shifter_rules		
	-cells	
	-location	
	-names	
	-prefix	
	-within_hierarchy	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-equivalent_ground_nets	
	-equivalent_power_nets	
	-name	
	-nmos_bias_net	Unsupported
	-pmos_bias_net	Unsupported
	-primary_ground_net	
	-primary_power_net	
	-transition_cycles	Unsupported
	-transition_latency	Unsupported
	-transition_slope	Unsupported
	-user_attributes {{disable_secondary_domains {list of domains}}}	Supported: query getCPFUserAttributes
update_power_mode		
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-average_ir_drop_limit	
	-dynamic_power_limit	Unsupported

	-leakage_power_limit	Unsupported
	-name	
	-peak_ir_drop_limit	
	-sdc_files	
update_power_switch_rule		
	-acknowledge_reciever_1	
	-acknowledge_reciever_2	
	-average_ir_drop_limit	
	-cells	
	-enable_condition_1	
	-enable_condition_2	
	-gate_bias_net	Unsupported
	-name	
	-peak_ir_drop_limit	
	-prefix	
update_state_retention_rules		
	-cell_type	
	-cells	
	-names	
	-set_rest_control	Unsupported

## CPF 1.1 Script Example

The following section contains an example of the CPF 1.1 file using a sample design and library.

For list of supported CPF commands and options within Innovus product family, see "[Supported CPF 1.1 Commands](#)".

```
#-----
# setting
#-----
```

```
set_cpf_version 1.1
set_hierarchy_separator /

#-----
# define library_set/cells
#-----

define_library_set -name wc_0v81 -libraries { \
..../LIBS/timing/library_wc_0v81.lib }

define_library_set -name bc_0v81 -libraries { \
..../LIBS/timing/library_bc_0v81.lib }

define_library_set -name wc_0v72 -libraries { \
..../LIBS/timing/library_wc_0v72.lib }

define_library_set -name bc_0v72 -libraries { \
..../LIBS/timing/library_bc_0v72.lib }

define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
VDD -power TVDD -ground VSS

define_isolation_cell -cells {LVLLH*} -power VDD -ground VSS -enable \
NSLEEP -valid_location to
define_isolation_cell -cells {ISOHID* ISOLOD*} -power VDD -ground VSS \
-enable ISO -valid_location to

define_level_shifter_cell -cells {LVLHLD*} -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells {PTLVLHLD*} -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells {LVLLHCD*} -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells {LVLLHD*} -input_voltage_range \
```

```
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \
-input_power_pin VDDL -output_power_pin VDD -ground VSS -valid_location to

define_power_switch_cell -cells { HEADERHVT1 HEADERHVT2 } \
-stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 -stage_2_enable \
NSLEEPIN2 -stage_2_output NSLEEPOUT2 -type header -power_switchable VDD \
-power TVDD -stage_1_on_resistance 10 - stage_2_on_resistance 10

define_state_retention_cell -cells { MSSRPG* } -cell_type \
master_slave -clock_pin CP -restore_check !CP -save_function !CP \
-always_on_components { DFF_inst } -power_switchable VDD -power TVDD \
-ground VSS

define_state_retention_cell -cells { BLSRPG* } -cell_type ballon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS

#-----
# macro models
#-----

#-----
# top design
#-----  
set_design top

create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72
```

```
create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_EQ -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-----
# create power domains
#-----
create_power_domain -name PDdefault -default
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifisp \
IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
-external_controlled_shutoff -shutoff_condition io_shutoff_ack
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
IOPADS_INST/Pbiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip }
```

```
IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } -boundary_ports { ibias reset \
refclk vcom vcop pllrst }

create_power_domain -name PDram_virtual

create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \
-shutoff_condition !INST/PM_INST/power_switch_enable \
-base_domains { PDram_virtual }

create_power_domain -name PDtdsp -instances { INST/RAM_128x16_TEST_INST1 \
INST/DSP_CORE_INST0 INST/DSP_CORE_INST1 } -shutoff_condition \
!INST/PM_INST/power_switch_enable -base_domains { PDdefault }

#-----
# set instances
#-----

set_instance INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -domain_mapping \
{ {RAM_DEFAULT PDtdsp} }

set_macro_model ram_256x16A

create_power_domain -name RAM_DEFAULT -boundary_ports { A* D* CLK CEN WEN Q* } \
-default -external_controlled_shutoff

create_state_retention_rule -name RAM_ret -instances { mem* } -save_edge !CLK

update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
-primary_ground_net VSS

end_macro_model ram 256x16A

#-----
# create power modes
#-----

create_power_mode -name PMdvfs1 -default -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDtdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }
```

```
create_power_mode -name PMdvfs1_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDshutoff_io@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs1_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2 -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDtdsp@nom_0v72 PDram@nom_0v72 PDshutoff_io@nom_0v72 \
PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDshutoff_io@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMscan -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDtdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }

create_analysis_view -name AV_dvfs1_BC -mode PMdvfs1 -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDtdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_WC -mode PMdvfs1 -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDtdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_off_BC -mode PMdvfs1_off -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_off_WC -mode PMdvfs1_off -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_shutoffio_off_BC -mode \
PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
PDdefault@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_shutoffio_off_WC -mode \
PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
PDdefault@PMdvfs1_wc }

create_analysis_view -name AV_dvfs2_BC -mode PMdvfs2 -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDtdsp@PMdvfs2_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs2_bc }
```

```
create_analysis_view -name AV_dvfs2_WC -mode PMdfs2 -domain_corners { \
PDpll@PMdfs1_wc PDdefault@PMdfs2_wc PDtdsp@PMdfs2_wc PDram@PMdfs2_wc \
PDshutoff_io@PMdfs2_wc }

create_analysis_view -name AV_PMdfs2_off_BC -mode PMdfs2_off -domain_corners \
{ PDpll@PMdfs1_bc PDdefault@PMdfs2_bc PDshutoff_io@PMdfs2_bc }

create_analysis_view -name AV_PMdfs2_off_WC -mode PMdfs2_off -domain_corners \
{ PDpll@PMdfs1_wc PDdefault@PMdfs2_wc PDshutoff_io@PMdfs2_wc }

create_analysis_view -name AV_dvfs2_shutoffio_off_BC -mode \
PMdfs2_shutoffio_off -domain_corners { PDpll@PMdfs1_bc \
PDdefault@PMdfs2_bc }

create_analysis_view -name AV_dvfs2_shutoffio_off_WC -mode \
PMdfs2_shutoffio_off -domain_corners { PDpll@PMdfs1_wc \
PDdefault@PMdfs2_wc }

create_analysis_view -name AV_scan_BC -mode PMscan -domain_corners { \
PDpll@PMdfs1_bc PDdefault@PMdfs1_bc PDtdsp@PMdfs1_bc PDram@PMdfs2_bc \
PDshutoff_io@PMdfs1_bc }

create_analysis_view -name AV_scan_WC -mode PMscan -domain_corners { \
PDpll@PMdfs1_wc PDdefault@PMdfs1_wc PDtdsp@PMdfs1_wc PDram@PMdfs2_wc \
PDshutoff_io@PMdfs1_wc }

#-----
# create rules
#-----

create_power_switch_rule -name PDram_SW -domain PDram -external_power_net VDDL
create_power_switch_rule -name PDtdsp_SW -domain PDtdsp -external_power_net \
VDD

create_isolation_rule -name ISORULE1 -isolation_condition { \
!INST/PM_INST/isolation_enable } -from { PDtdsp } -to { PDdefault } \
-isolation_target from -isolation_output high

create_isolation_rule -name ISORULE3 -isolation_condition { \
!INST/PM_INST/isolation_enable } -from { PDram } -to { PDdefault } \
-isolation_target from -isolation_output high

create_isolation_rule -name ISORULE4 -isolation_condition { \
```

```
!INST/PM_INST/spi_ip_isolate } -from { PDshutoff_io } \
-isolation_target from -isolation_output low

create_level_shifter_rule -name LSRULE_H2L3 -from { PDdefault } -to { PDram } \
-exclude { INST/PM_INST/power_switch_enable }

create_level_shifter_rule -name LSRULE_H2L_PLL -from { PDpll }

create_level_shifter_rule -name LSRULE_L2H2 -from { PDram } -to { PDdefault }

create_state_retention_rule -name \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -instances { \
INST/DSP_CORE_INST0 } -save_edge !INST/DSP_CORE_INST0/clk

create_state_retention_rule -name \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -instances { \
INST/DSP_CORE_INST1 } -restore_edge \
!INST/PM_INST/state_retention_restore -save_edge \
INST/PM_INST/state_retention_save

#-----
# update domains/modes
#-----

update_nominal_condition -name nom_0v81 -library_set wc_0v81
update_nominal_condition -name nom_0v72 -library_set wc_0v72

update_power_domain -name PDdefault -primary_power_net VDD -primary_ground_net \
VSS -equivalent_power_nets VDD_EQ
update_power_domain -name PDshutoff_io -primary_power_net VDD_IO \
-primary_ground_net VSS
update_power_domain -name PDpll -primary_power_net Avdd -primary_ground_net \
Avss
update_power_domain -name PDram_virtual -primary_power_net VDDL \
-primary_ground_net VSS
update_power_domain -name PDram -primary_power_net VDDL_sw -primary_ground_net \
VSS
update_power_domain -name PDtdsp -primary_power_net VDD_sw -primary_ground_net \
```

VSS

```
update_power_mode -name PMdvfs1 -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_off -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_shutoffio_off -sdc_files \
..../RELEASE/mmmc/dvfs1.sdc

update_power_mode -name PMdvfs2 -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_off -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_shutoffio_off -sdc_files \
..../RELEASE/mmmc/dvfs2.sdc

update_power_mode -name PMscan -sdc_files ../RELEASE/mmmc/scan.sdc

#-----
# update rules
#-----

update_power_switch_rule -name PDram_SW -cells { HEADERHVT1 } -prefix \
CDN_SW_RAM -peak_ir_drop_limit 0 -average_ir_drop_limit 0
update_power_switch_rule -name PDtdsp_SW -cells { HEADERHVT2 } -prefix \
CDN_SW_TDSP -peak_ir_drop_limit 0 -average_ir_drop_limit 0

update_isolation_rules -names ISORULE1 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE3 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE4 -location to -prefix CPF_ISO_

update_level_shifter_rules -names LSRULE_H2L3 -location to -cells { LVLHLD* \
} -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_H2L_PLL -location to -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_L2H2 -location to -prefix CPF_LS_

update_state_retention_rules -names \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -cell_type master_slave
update_state_retention_rules -names \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -cell_type balloon_latch
```

```
#-----  
# end  
#-----end_design
```

## Supported SAI Commands

The following are the supported SAI Commands:

- [add\\_clock](#)
- [add\\_macro](#)
- [add\\_module](#)
- [connect](#)
- [delete\\_macro](#)
- [delete\\_module](#)
- [insert\\_boundary\\_flops](#)
- [set\\_floorplan](#)
- [set\\_ref\\_flop](#)
- [set\\_ref\\_gate](#)
- [set\\_ref\\_memory](#)
- [set\\_sai\\_version](#)

### **add\_clock**

```
add_clock  
[-help]  
clockName  
-buffer string  
-period string  
[-waveform string]
```

Allows the software to add new clocks. The `add_clock` command creates clock root in netlist and adds the `create_clock` constraint in generated SDC along with connecting the clock root to clock port of hierarchical instances. If SDC is provided before SAI, then the `add_clock` command only connects clock root to clock port of hierarchical instances.

Use this command in SAI interactive mode or write in SAI file.

## Parameters

<i>clockName</i>	Specifies the clock name.
<i>-buffer string</i>	Specifies the buffer cell name. You can also specify “-” for clock port. <ul style="list-style-type: none"><li>● If buffer cell name is specified, instance of the cell is inserted as clock root in top level.</li><li>● If “-” is specified, clock root is created for the external port on top level.</li></ul>
<i>-help</i>	Outputs a brief description that includes type and default information for each <code>add_clock</code> parameter.
<i>-period string</i>	Specifies the clock period.
<i>-waveform string</i>	Specifies the clock waveform.

## **add\_macro**

```
add_macro
[-help]
[ref_counts]
-cell targetModule
[-memory string]
[-update_memory_bit_count string]
```

Allows the software to add macro instances to the netlist.

Use this command in SAI interactive mode or write in SAI file.

## Parameters

-cell <i>targetModule</i>	Specifies target module to add macros.
-help	Outputs a brief description that includes type and default information for each add_macro parameter.
-memory <i>string</i>	<p>Specifies the reference memory name created using the <code>set_ref_memory</code> command.</p> <p><b>For example,</b>  <code>set_ref_memory mem1 -bit_size 1024 -area_per_bit 1.1 -aspect_ratio 0.75</code>  <code>add_module iBlock -cell mBlock -memory_bit_count 4096</code>  <code>add_macro -cell mBlock -memory mem1</code></p> <p>then, <math>4096 / 1024 = 4</math> “mem1” macros are added in mBlock module.</p>
<i>ref_counts</i>	<p>Specifies a list of macro cell name and count pair.</p> <p><b>For example,</b>  <code>{RAM1 12 RAM2 24 ...}</code></p>
-update_memory_bit_count <i>string</i>	Updates memory bit count for adding reference memory further.

## Example

```
add_macro -cell ryon_mod1 {ram_128x16A 2} ... (A)
add_macro -cell ryon_mod1 {ram_128x16A 4} ... (B)
```

**add\_macro (A) creates**  
`XM_PORT0/ram_128x16A_0`  
`XM_PORT0/ram_128x16A_1`

**add\_macro (B) creates**  
`XM_PORT0/ram_128x16A_3`  
`XM_PORT0/ram_128x16A_4`  
`XM_PORT0/ram_128x16A_5`  
`XM_PORT0/ram_128x16A_6`

## **add\_module**

```
add_module
[-help]
name
[-aspect_ratio_range { min max }]
[-cell moduleName]
[-flop_count string]
[-flop_ref_clock clkName]
[-gate_count string]
[-memory_bit_count string]
[-util float]
```

Generates hierarchical instance and adds the associated module if it does not exist yet. It automatically creates intermediate level instances and modules if missing.

Use this command in SAI interactive mode or write in SAI file.

## Parameters

-aspect_ratio_range { min max}	Specifies the aspect ratio range for <code>planDesign</code> . These aspect ratios will appear in the <code>sai_planDesign.seed</code> file.
-cell <i>moduleName</i>	Specifies the module name. A module is created if the specified module name does not exist. If the module name is not specified, then the local hierarchical instance name is used as the module name.  For example, <code>add_macro A/B/C/D</code> creates module D and its hierarchical instance A/B/C/D.
-flop_count <i>string</i>	Specifies the number of flops to insert as intermediate flops. It uses the cell set for <code>set_ref_flop</code> .  To add flops with multiple clock domains, specify <code>{numFlops1 clkName1 numFlops2 clkName2 ... }</code> .  <b>Note:</b> You can also specify the flop count using an integer value. However, if you specify an integer, then you must specify the <code>-flop_ref_clock clkName</code> option. In this case, all added flops' clock pins are connected to the <code>clkName</code> port.
-flop_ref_clock <i>clkName</i>	Specifies the flop's reference clock
-gate_count <i>string</i>	Specifies the number of instance to insert. It uses the cell set for <code>set_ref_gate</code> .
-help	Outputs a brief description that includes type and default information for each <code>add_module</code> parameter.
-memory_bit_count <i>string</i>	Specifies the number of memory bits in the module.
<i>name</i>	Specifies hierarchical instance name to generate.
-util <i>float</i>	Specifies the target utilization value for <code>planDesign</code> . This target utilization appears in the <code>sai_planDesign.seed</code> file.

## connect

```
connect
[-help]
src
[-bus_width integer]
-clock string
[-insert_driver_flop {auto | none | force}]
[-insert_receiver_flop {auto | none | force}]
[-pipeline_stages integer]
-to string
[-to_clock string]
```

Creates a net connection by connecting the source and destination clocks.

## Parameters

<code>-bus_width integer</code>	Specifies the bus width of connection. If not specified, single connection is created.
<code>-clock string</code>	Specifies clock name of source and destination flops.
<code>-help</code>	Outputs a brief description that includes type and default information for each <code>connect</code> parameter.
<code>-insert_driver_flop {auto   none   force}</code>	<p>Inserts source boundary flops.</p> <ul style="list-style-type: none"> <li>• <code>auto</code>: If the specified source module is created by SAI, then inserts source boundary flops automatically.</li> <li>• <code>none</code>: Does not insert source boundary flops.</li> <li>• <code>force</code>: Enforces the insertion of source boundary flops.</li> </ul> <p><i>Default:</i> <code>auto</code></p>
<code>-insert_receiver_flop {auto   none   force}</code>	<p>Inserts destination boundary flops.</p> <p>Inserts source boundary flops.</p> <ul style="list-style-type: none"> <li>• <code>auto</code>: If the specified destination module is created by SAI, then inserts destination boundary flops automatically.</li> <li>• <code>none</code>: Does not insert destination boundary flops.</li> <li>• <code>force</code>: Enforces the insertion of destination boundary flops.</li> </ul> <p><i>Default:</i> <code>auto</code></p>
<code>-pipeline_stages integer</code>	Specifies the number of pipeline flops between source flops and destination flops. If not specified, source flops and destination flops are connected directly.
<code>src</code>	Specifies the source hierarchical port name.
<code>-to string</code>	Specifies the destination hierarchical port name.
<code>-to_clock string</code>	Specifies clock name of destination flops. If not specified, destination flops are connected to <code>clock_name1</code> .

## delete\_macro

```
delete_macro
[-help]
[macroInstanceOrCellName]
[-cell moduleName]
```

Delete macro instances. It allows the software to delete macros created using the `add_macro` command.

Use this command in SAI interactive mode or write in SAI file.

## Parameters

<code>-cell moduleName</code>	Specify the cell name that has the macro inside.  Deletes all the macros that the <code>macroInstanceOrCellName</code> in module specified by <code>moduleName</code> .
<code>-help</code>	Outputs a brief description that includes type and default information for each <code>delete_macro</code> parameter.
<code>macroInstanceOrCellName</code>	Deletes the specified macro instance name or the macro's cell name if <code>-cell</code> is specified.

## delete\_module

```
delete_module
[-help]
hinstName
[-cell]
```

Deletes hierarchical instance and module. It allows the software to delete modules inserted using the `add_module` command.

Use this command in SAI interactive mode or write in SAI file.

## Parameters

<code>-cell</code>	If specified, Verilog module of <code>hinstName</code> is also deleted.
<code>-help</code>	Outputs a brief description that includes type and default information for each <code>delete_module</code> parameter.
<code>hinstName</code>	Specifies hierarchical instance name to delete.

## insert\_boundary\_flops

```
insert_boundary_flops
[-help]
-clock string
-module string
-terms string
```

Allows insertion of boundary flops for empty modules/blocks (instantiated in the top level) through SAI. The command options specify the clock net name, the module name and the module pin name.

## Parameters

<code>-clock string</code>	Specifies clock net name.
<code>-help</code>	Outputs a brief description that includes type and default information for each <code>insert_boundary_flops</code> parameter.
<code>-module string</code>	Specifies the module name.
<code>-terms string</code>	Specifies module pin name.

## set\_floorplan

```
set_floorplan
[-help]
[-aspect_ratio float]
[-height float]
[-side_spacing float]
[-util float]
[-width float]
```

Defines specifications for the `floorplan` command before netlist import. The aspect ratio and utilization values will be passed to the `floorplan` command.

Use this command in SAI interactive mode or write in SAI file.

## Parameters

- aspect_ratio <i>float</i>	Specifies the core dimensions of the chip as the ratio of the height divided by the width. If a value of 1.0 is used, a square chip is defined. A value of 2.0 will define a rectangular chip with height dimensions that are twice the width dimension.
-height <i>float</i>	Specifies the height of the die.
-help	Outputs a brief description that includes type and default information for each <code>set_floorplan</code> parameter.
- side_spacing <i>float</i>	Specifies the distance from all the sides to the core.
-util <i>float</i>	Specifies the target utilization for the core.
-width <i>float</i>	Specifies the width of the die.

## set\_ref\_flop

```
set_ref_flop
[-help]
reference_flop_cell_name
```

Defines the reference flop cell. The reference flop cell is used for boundary flop (inserted by `connect` command) and dummy flops (inserted by `add_module` command).

Use this command in SAI interactive mode or write in SAI file.

## Parameters

-help	Outputs a brief description that includes type and default information for each <code>set_ref_flop</code> parameter.
reference_flop_cell_name	Specifies the cell name of the reference flop.

## set\_ref\_gate

```
set_ref_gate
[-help]
reference_unit_gate_cell
```

Defines reference gate cell. The reference gate cell is used for dummy gates (inserted by `add_module` command).

Use this command in SAI interactive mode or write in SAI file.

## Parameters

<code>-help</code>	Outputs a brief description that includes type and default information for each <code>set_ref_gate</code> parameter.
<code>reference_unit_gate_cell</code>	Specifies the cell name of the reference gate.

## set\_ref\_memory

```
set_ref_memory
[-help]
name
-area_per_bit float
[-aspect_ratio float]
[-bit_size float]
```

Allows the software to set reference memory cells in SAI mode. This command creates LEF MACRO. Reference memory is used for macro insertion by `add_module` and `add_macro` command.

Use this command in SAI interactive mode or write in SAI file.

## Parameters

<code>-area_per_bit float</code>	Specifies the area per bit.
<code>-aspect_ratio float</code>	Specifies the aspect ratio of the memory cell. <i>Default:</i> 1.0
<code>-bit_size float</code>	Specifies the bit size. This size is used with <code>add_module</code> and <code>add_macro</code> command.  For example, <code>set_ref_memory mem1 -bit_size 1024 -area_per_bit 1.1 -aspect_ratio 0.75</code> <code>add_module iBlock -cell mBlock -memory_bit_count 4096</code> <code>add_macro -cell mBlock -memory mem1</code>  then, $4096 / 1024 = 4$ "mem1" macro are added in mBlock module.
<code>-help</code>	Outputs a brief description that includes type and default information for each <code>set_ref_memory</code> parameter.
<code>name</code>	Specifies the reference memory cell name. It generates <code>name.lef</code> on the Innovus working directory.

## set\_sai\_version

```
set_sai_version
[-help]
version
```

Sets the version of the SoC Architecture Information (SAI) mode.

Use this command in SAI interactive mode or write in SAI file.

### Parameters

-help	Outputs a brief description that includes type and default information for each <code>set_sai_version</code> parameter.
version	Specifies the SAI format version.

# Cadence-Specific Liberty Extensions

- [Overview](#)
- [Guidelines For Adding ECSM Extensions](#)
- [Representing ECSM Information in a Library](#)
- [Defining ECSM Extensions in a Library](#)
  - [ecsm\\_waveform Group](#)
  - [ecsm\\_capacitance Group](#)
- [Example](#)

This chapter describes Cadence-specific extensions to the Liberty (.lib) library file. These extensions allow the storage of the data required by the delay calculator's effective current source model (ECSM) voltage and current profiles.

## Overview

The delay calculator uses an advanced cell driver model to represent the effect of non-linear switching waveforms on cell-based interconnect delay calculation and signal integrity. This ECSM model provides an efficient mechanism for storing output current or voltage profiles during active transitions on the circuit.

ECSM models are typically generated by the Encounter Library Characterizer, and stored directly in the delay calculator's database. However, you can also derive ECSM information from a generic characterization flow and store it in a traditional library file using the extensions described in this appendix. These extensions allow the contents of ECSM models to be stored in library files in a way that is compatible with existing data and with other new signal integrity constructs.

**Note:** The presence of ECSM data does not interfere with other uses of the library file.

## Guidelines For Adding ECSM Extensions

- Extensions must be compatible with existing or new library model constructs. They cannot be stored as comments, which could potentially be stripped out by internal or commercial tools that read the library models.
- Extensions must resemble similar library constructs.
- Extensions must use the same context as the equivalent library construct. For example, units must be consistent. If an extension uses a current value and the current unit is defined as 1 milliampere, the values placed in the attribute must be in milliamperes.
- Syntactically correct extensions added to the libraries models must pass through the Library Compiler, and should not cause any change in behavior in any tool that uses the compiled models.
- Extensions should be easily extracted by tools that correctly parse library models.
- Storage of the waveforms must be efficient. Appropriate reduction must be performed on the waveforms to a user-specified level of accuracy. The controls provided to adjust the accuracy should enable you to achieve a compromise between the accuracy of the analysis and the size of the model.

## Representing ECSM Information in a Library

ECSM information is stored in the form of output voltage waveforms, which enables the library to include more information about the transition, and to improve the driver model accuracy. Given that the library transition table is based on two types of indexes (input slew rate and output loading capacitance), and the delay model uses a lumped capacitance to represent the observed loading, the following equation computes the output current ( $I_{out}$ ):

$$ECSM(t_1, t_2) = \frac{\int_{t_1}^{t_2} I_{out}(t) dt}{t_2 - t_1} = C_{load} \times \frac{(V_{out}(t_2) - V_{out}(t_1))}{t_2 - t_1}$$

You can use this equation to convert the output voltage waveform to ECSM. To support multiple supply voltage corners, normalize the output voltage numbers from 0.0 to 1.0 of the supply voltage. The effective current should also be normalized to the supply voltage.

Using the output voltage waveform approach, an ECSM extension becomes a table containing voltage over transition times for every input slew and output loading capacitance index combination. Add this table to the output transition table group using the following format:

```
rise_transition ( template_of_slew_load ) { // fall_transition (template_of_slew_load) {
index_1 : // redefine of slew rate index

index_2 : // redefine of loading index

ecsm_waveform( name0 ) {

index_1 : "..."; // output voltage sample points

values : "..."; // output voltage sample times

}
```

```
ecsm_waveform(name1) {
    index_1 : "..."; // output voltage sample points
    values : "..." ; // output voltage sample times
    ...
}
```

## Defining ECSM Extensions in a Library

Two user-defined groups can be added to a library to define an ECSM extension:

- `ecsm_waveform`
- `ecsm_capacitance`

The `ecsm_waveform` describes the output rise or fall voltage waveform during a transition. This group allows the creation of output voltage waveforms as a function of time. The `ecsm_capacitance` group describes the input capacitance during the rise or fall transition. The input capacitance can be specified as a function of input transition and output load.

**Note:** In accordance with ECSM specification version 1.2, the `ecsm_capacitance` group also allows you to represent the input pin capacitance as a function of input transition and output load.

The two groups are defined within the `rise_transition` or `fall_transition` group within a `timing` group for regular delay arcs, and within the `retain_rise_slew` or `retain_fall_slew` group within a `timing` group for retain delay arcs. The `rise_transition`, `fall_transition`, `retain_rise_slew`, and `retain_fall_slew` groups specify the output slew or transition time as a function of the input net transition and total output capacitance.

- To include these groups in a library, add the following `define_group` statements to the library.

```
define_group( ecmp_waveform, rise_transition);
define_group( ecmwaveform, fall_transition);
define_group( ecm_capacitance, rise_transition);
define_group( ecm_capacitance, fall_transition);
define_group( ecm_capacitance, pin);
```

or:

```
define_group( ecmwaveform, retain_rise_slew);
define_group( ecmwaveform, retain_fall_slew);
define_group( ecm_capacitance, retain_rise_slew);
```

```
define_group( ecsm_capacitance, retain_fall_slew);
define_group( ecsm_capacitance, pin);
```

- Add the following `define` statements to the library to define attributes for the groups:

```
define( index_1, ecsm_waveform, string );
define( values, ecsm_waveform, string );
define( values, ecsm_capacitance, string );
```

You must also include a `version` statement to enable the exact processing of ECSM constructs. Add the following `version` statement in the library:

```
define( ecsm_version, library, float );
```

The current version is 1.2.

## **ecsm\_waveform Group**

The `ecsm_waveform` group enables the output voltage waveforms to be specified separately for each index permutation suggested by the slew and load indexes. It also enables each waveform to be sampled at different points so that waveforms that are predominantly linear, can be represented with fewer sample points. The number of `ecsm_waveform` groups must match the number of entries in the `values` attribute of the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

The name of the `ecsm_waveform` group must be an integer enclosed within quotation marks. The minimum value of the integer must be 0, and the maximum value must be a number that is one less than the number of entries in the `values` attribute of the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

Including the `ecsm_waveform` group in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group specifies that all information related to the transition arc also applies to the `ecsm_waveform` group. If not explicitly specified, the lookup template name and the index overrides are inherited from the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. Attributes associated with the `timing` group in which the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group resides are also associated with the `ecsm_waveform` group, including `related_pin`, `timing_sense`, and `timing_type`.

The Figure 1 below shows four sampled output voltage waveforms that begin a transition at 1 nanosecond. The following example represents an `ecsm_waveform` group (for a regular delay arc) that specifies the points on each waveform shown in the figure:

```
rise_transition( temp_1x4 ) {
index_1 : "0.1n"; index_2 : "0.1p 0.2p 0.3p 0.4p"; values ( "0.01n, 0.02n, 0.026n, 0.45n" );
ecsm_waveform( "0" ) {
index_1 : "0.1, .3, .7,.9";

values : "1.005n, 1.012n, 1.018n, 1.02n";
} ecm_waveform( "1" ) {
index_1 : "0.1, .48, .7,.9";
```

```

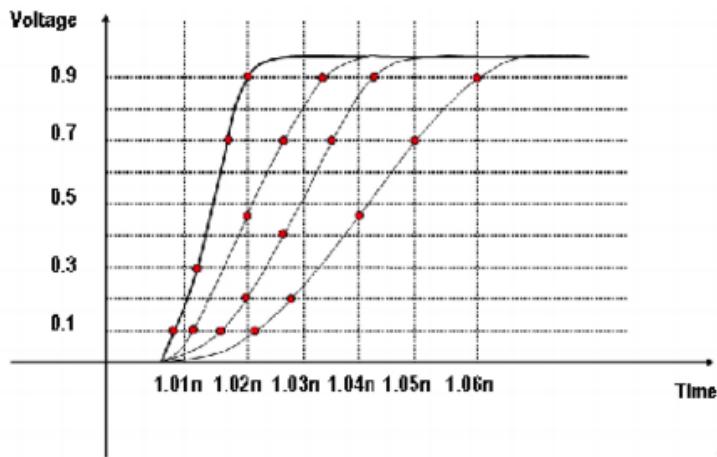
values : "1.011n, 1.02n, 1.027n, 1.032n";
} ecsm_waveform( "2" ) {
index_1 : "0.1, .2, .4, .7,.9";

values : "1.015n, 1.02n, 1.028n, 1.035n, 1.07n";
} ecsm_waveform( "3" ) {
index_1 : "0.1, .2, .45, .7,.9";

values : "1.021n, 1.029n, 1.04n, 1.06n, 1.07n";
}
}

```

**Figure 1: Sampled Waveform**



The following example represents an `ecsm_waveform` group for a retain delay arc:

```

retain_rise_slew(delay_template_6x6) {

index_1 ("0.001, 0.0105, 0.02, 0.039, 0.077, 0.152");

index_2 ("0.012007, 0.09354, 0.189187, 0.373938, 0.757224, 1.50616");

values ( \
"0.026141, 0.027748, 0.031751, 0.038769, 0.051329, 0.070025", \
"0.136231, 0.135178, 0.137198, 0.137161, 0.145185, 0.160992", \
"0.247332, 0.247016, 0.244592, 0.244943, 0.251245, 0.260942", \
"0.469122, 0.469234, 0.463448, 0.467459, 0.464259, 0.478051", \
"0.912834, 0.903097, 0.907181, 0.907186, 0.907485, 0.902419", \
"1.78894, 1.77071, 1.78538, 1.78471, 1.76567, 1.781");

ecsm_waveform("0") {

index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;
}
}

```

```
values : "0.100926, 0.104628, 0.108441, 0.112042, 0.11568, 0.119465, 0.12354, 0.128418, 0.134582,  
0.144439, 0.160979" ;  
}  
  
ecsm_waveform("1") {  
  
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;  
  
    values : "0.101614, 0.108072, 0.113352, 0.117633, 0.121708, 0.125782, 0.129998, 0.13488, 0.141099,  
0.15062, 0.164307" ;  
}  
  
ecsm_waveform("2") {  
  
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;  
  
    values : "0.1019, 0.109498, 0.116861, 0.122593, 0.127486, 0.132252, 0.136939, 0.142209, 0.148612,  
0.158116, 0.178427" ;  
}  
  
ecsm_waveform("3") {  
  
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;  
  
    values : "0.103866, 0.119331, 0.129292, 0.137018, 0.143679, 0.149411, 0.155144, 0.161135, 0.168061,  
0.178948, 0.198529" ;  
}  
  
...  
  
ecsm_waveform("35") {  
  
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;  
  
    values : "0.148963, 0.344816, 0.57925, 0.813683, 1.0576, 1.3144, 1.59557, 1.93059, 2.36025, 2.98918,  
4.13212" ;  
}  
}
```

## **ecsm\_waveform Attributes**

Two simple attributes are allowed in an `ecsm_waveform` group:

- `index_1`

The `index_1` attribute is a comma-separated list of floating-point numbers representing normalized output voltage sample points. These values are normalized and must be between 0.0 and 1.0.

- `values`

The `values` attribute is a comma-separated list of floating-point numbers representing the times at which the output voltages are sampled. The number of floating-point numbers must be equal to the number of entries in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` table multiplied by the number of entries in the `index_1` attribute. The time entries in this attribute must monotonically increase, but it is not necessary to start from a time reference of 0.

Each value represents the time at which the corresponding sampled point in the `index_1` list is crossed for the first time. The `index_1` and `values` attributes must have the same number of floating-point entries. Each `ecsm_waveform` group can have a different number of entries for this attribute; however, the number of entries for a group must match the number of points in the corresponding `index_1` attribute. The time units for this attribute use the value of the library-level `time_unit` attribute.

## Waveform Order and Size

It is not necessary for `ecsm_waveform` groups to appear in order. However, the integer number of `ecsm_waveform` groups must correspond exactly to the number of entries in the `values` attribute of the `rise_transition` or `fall_transition` group. The parser uses the integer number to determine the value of `index_1` and `index_2`.

The `ecsm_waveform` group provides more accurate voltage waveforms using a smaller number of sampling points. You can use any kind of waveform reduction method to obtain the minimum number of points, and linear interpolation to eliminate those points between two voltage points. Set the interpolation error to less than 0.001.

## `ecsm_capacitance` Group

The `ecsm_capacitance` group describes the input capacitance during the rise or fall transition. It specifies the input capacitance for each input transition and capacitive load defined by the specified `lu_table_template` or `index_1` and `index_2` overrides. The indexes are inherited from the specified `lu_table_template` or `index_1` and `index_2` overrides within the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group, and cannot be overridden. The `ecsm_capacitance` group requires the name `rise` or `fall`, and must match the transition direction of the parent group.

Including the `ecsm_capacitance` group in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group specifies that the capacitance values are processed in the order specified by the rules used to analyze the values within a `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

The following example represents an `ecsm_capacitance` group in a library for a regular delay arc:

```
rise_transition( temp_1x4 ) {
    index_1 : "0.1n";
}
```

```
index_2 : "0.1p 0.2p 0.3p 0.4p";  
  
values ( "0.01n, 0.02n, 0.026n, 0.45n" );  
  
ecsm_capacitance( rise ) {  
  
values : "0.01, 0.02, 0.03, 0.04";  
  
}  
}
```

The following example represents an `ecsm_capacitance` group in a library for a retain delay arc:

```
retain_rise_slew(delay_template_6x6) {  
  
index_1 ("0.001, 0.0105, 0.02, 0.039, 0.077, 0.152");  
  
index_2 ("0.012007, 0.09354, 0.189187, 0.373938, 0.757224, 1.50616");  
  
values ( \  
"0.026141, 0.027748, 0.031751, 0.038769, 0.051329, 0.070025", \  
"0.136231, 0.135178, 0.137198, 0.137161, 0.145185, 0.160992", \  
"0.247332, 0.247016, 0.244592, 0.244943, 0.251245, 0.260942", \  
"0.469122, 0.469234, 0.463448, 0.467459, 0.464259, 0.478051", \  
"0.912834, 0.903097, 0.907181, 0.907186, 0.907485, 0.902419", \  
"1.78894, 1.77071, 1.78538, 1.78471, 1.76567, 1.781");  
  
ecsm_capacitance(rise) {  
  
values : "0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \  
0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \  
0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \  
0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \  
0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176" ;  
  
}  
}
```

Including the `ecsm_capacitance` group in the `pin` group allows you to represent the input pin capacitance as a function of

input transition and output load. For example:

```
cell (cellname) {  
    pin (pinname) {  
        ecssm_capacitance(rise) {  
            index_1 : "0.1n 0.2n";  
            values : "0.01, 0.02";  
        }  
        ecssm_capacitance(fall) {  
            index_1 : "0.1n 0.2n";  
            values : "0.01, 0.02 ";  
        }  
    }  
}
```

Attributes associated with the `timing` group in which the `pin` group resides are also associated with the `ecsm_capacitance` group, including `timing_sense`, and `timing_type`.

## **ecsm\_capacitance Attributes**

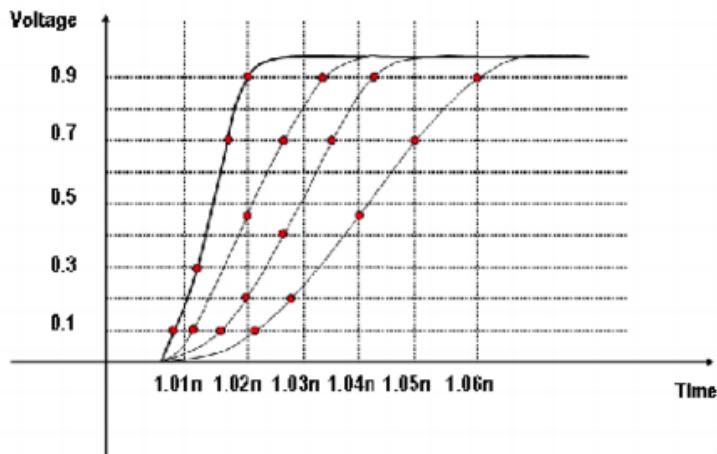
One simple attribute is allowed in an `ecsm_capacitance` group:

- `values`

The `values` attribute is a comma-separated list of floating-point numbers representing the input capacitance at each specified index point. The values for `index_1` and `index_2` cannot be overridden and are inherited from the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. The number of floating point numbers must be equal to the number of entries in the `values` attribute in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. The capacitive units for this attribute use the same value as the library-level `capacitive_load_unit` attribute.

The figure below shows four sampled output voltage waveforms that begin a transition at 1 nanosecond.

**Figure 2: Sampled Waveform**



## Example

The following example of a Liberty library includes ECSM extensions:

```
library (typ) {  
    delay_model : table_lookup ;  
    date : "Tue Mar 25 14:54:48 CST 2003" ;  
    time_unit : 1ns ;  
    voltage_unit : 1V ;  
    current_unit : 1mA ;  
    capacitive_load_unit(1, pf);  
    pulling_resistance_unit : 1kohm ;  
    leakage_power_unit : 1mW ;  
    input_threshold_pct_fall : 50.0 ;  
    input_threshold_pct_rise : 50.0 ;  
    output_threshold_pct_fall : 50.0 ;  
    output_threshold_pct_rise : 50.0 ;  
    slew_lower_threshold_pct_fall : 10.0 ;  
    slew_lower_threshold_pct_rise : 10.0 ;  
    slew_upper_threshold_pct_fall : 90.0 ;  
    slew_upper_threshold_pct_rise : 90.0 ;  
    nom_process : 1.0 ;  
    nom_temperature : 25 ;  
    nom_voltage : 1.5 ;
```

```
default_cell_leakage_power : 0.0 ;
default_fanout_load : 1.0 ;
default inout_pin_cap : 1.0 ;
default input_pin_cap : 1.0 ;

default_leakage_power_density : 0.0 ;
default_output_pin_cap : 0.0 ;

define_group( ecssm_waveform, rise_transition);
define_group( ecssm_waveform, fall_transition);
define_group( ecssm_capacitance, rise_transition);
define_group( ecssm_capacitance, fall_transition);

define( ecssm_version, library, float)
define( index_1, ecssm_waveform, string );
define( values, ecssm_waveform, string );
define( index_1, ecssm_capacitance, string );

ecssm_version : 1.2

input_voltage(default) {
vil : 0.0 ; vih : 1.5 ; vimin : 0.0 ; vimax : 1.5 ;
}

operating_conditions(typ) {
process : 1.0 ; temperature : 25 ; voltage : 1.5 ;
}

output_voltage(default) {
vol : 0.0 ; voh : 1.5 ; vomin : 0.0 ; vomax : 1.5 ;
}

lu_table_template(tmg_ntin_oload_5x5) {
variable_1 : input_net_transition ; variable_2 : total_output_net_capacitance ; index_1("1.0, 2.0, 3.0,
4.0, 5.0"); index_2("1.0, 2.0, 3.0, 4.0, 5.0");
}

power_lut_template(pwr_tin_oload_3x3) {
variable_1 : input_transition_time ; variable_2 : total_output_net_capacitance ; index_1("1.0, 2.0, 3.0");
index_2("1.0, 2.0, 3.0");
}

cell(INV1) {
area : 1.0 ; cell_leakage_power : 4.467483e-06 ;
```

```
pin(A) {   capacitance : 0.00230301522 ;   direction : input ; }

pin(Z) {   direction : output ;   function : "!A" ;   internal_power()
related_pin : "A" ;

power(pwr_tin_oload_3x3) {

index_1("0.074822, 0.249940, 0.828130");
index_2("0.000010, 0.051277, 0.412740");
values("1.673300, 1.478500, 1.323100", \
"1.746800, 1.541300, 1.372000", \
"2.187000, 1.934600, 1.669400");

}

      timing() {
related_pin : "A" ;

timing_sense : negative_unate ;
timing_type : combinational ;

cell_fall(tmg_ntin_oload_5x5) {

index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
index_2("0.01, 0.030465, 0.1023016, 0.239484, 0.4532074");
values("0.0176577, 0.0450122, 0.1403913, 0.3230411, 0.6076288", \
"0.0273645, 0.055584, 0.1511069, 0.3335839, 0.6181163", \
"0.0310345, 0.0807605, 0.1886131, 0.3696548, 0.6535805", \
"0.0133436, 0.0864669, 0.2438067, 0.4415566, 0.7228604", \
"0.000844, 0.0666426, 0.2756392, 0.5369333, 0.8356729");

}

cell_rise(tmg_ntin_oload_5x5) {

index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
```

```
index_2("0.01, 0.01745396, 0.043619, 0.09358491, 0.1714293");  
  
values("0.0403989, 0.0630895, 0.1434606, 0.2971998, 0.5368189", \  
      "0.0520428, 0.0741513, 0.1547805, 0.3085111, 0.5480488", \  
      "0.0903201, 0.1192166, 0.2002351, 0.3506313, 0.5883372", \  
      "0.1425557, 0.1785485, 0.2836617, 0.4403199, 0.6725487", \  
      "0.2117557, 0.2558067, 0.379942, 0.5725538, 0.8146103");  
  
}  
  
fall_transition(tmg_ntin_oload_5x5) {  
  
    index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");  
  
    index_2("0.01, 0.030465, 0.1023016, 0.239484, 0.4532074");  
  
    values("0.0331679, 0.0861214, 0.2739402, 0.6353564, 1.198358", \  
          "0.0484462, 0.095128, 0.274244, 0.6354434, 1.198372", \  
          "0.1031723, 0.1519032, 0.3070776, 0.6442821, 1.198351", \  
          "0.1955602, 0.2593558, 0.4179065, 0.7180122, 1.235083", \  
          "0.3278351, 0.4087576, 0.6007222, 0.8868572, 1.358997");  
  
ecsm_waveform("0") {  
  
    index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;  
  
    values : "0.0, 0.02, 0.025, 0.45, 0.85" ;  
  
}  
  
ecsm_waveform("1") {  
  
    index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;  
  
    values : "0.0, 0.02, 0.025, 0.45, 0.85" ;
```

}

```
ecsm_capacitance(rise) {  
  
    values : "0.00, 0.01, 0.02, 0.03, 0.04, \  
              0.05, 0.06, 0.07, 0.08, 0.09, \  
              0.10, 0.11, 0.12, 0.13, 0.14, \  
              0.15, 0.16, 0.17, 0.18, 0.19, \  
              0.20, 0.21, 0.22, 0.23, 0.24" ;  
  
}
```

}

```
rise_transition(tmg_ntin_oload_5x5) {  
  
    index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2") ;  
  
    index_2("0.01, 0.01745396, 0.043619, 0.09358491, 0.1714293") ;  
  
    values("0.0802415, 0.132298, 0.3133739, 0.6588676, 1.197433", \  
           "0.0894187, 0.1360771, 0.3130822, 0.6588872, 1.197413", \  
           "0.1337757, 0.1777775, 0.3401834, 0.6683502, 1.197639", \  
           "0.2099556, 0.261308, 0.4215237, 0.7312425, 1.236211", \  
           "0.3291244, 0.3794321, 0.5553791, 0.8574848, 1.341265") ;  
  
}
```

```
ecsm_waveform("1") {  
  
    index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;  
  
    values : "0.0, 0.02, 0.025, 0.45, 0.85" ;  
  
}
```

```
ecsm_waveform("0") {
```

```
index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;  
values : "0.0, 0.02, 0.025, 0.45, 0.85" ;  
}  
  
ecsm_capacitance(fall) {  
values : "0.00, 0.01, 0.02, 0.03, 0.04, \  
0.05, 0.06, 0.07, 0.08, 0.09, \  
0.10, 0.11, 0.12, 0.13, 0.14, \  
0.15, 0.16, 0.17, 0.18, 0.19, \  
0.20, 0.21, 0.22, 0.23, 0.24" ;  
}  
}  
}  
}
```