



# **EDI System User Guide**

**Product Version 13.1**  
**April 2013**

© 2013-2014 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

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

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

All other trademarks are the property of their respective holders.

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

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

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

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

# Contents

---

1.	46
<b>About This Manual</b>	<b>46</b>
Audience	46
How This Manual Is Organized	46
Conventions Used in This Manual	46
Related Documents	48
EDI System Product Documentation	48
2.	50
<b>Product and Licensing Information</b>	<b>50</b>
Overview	50
About EDI System Products and Product Options	50
EDI System	50
First Encounter Hierarchical Prototyping Solution	53
EDI System Product Options	53
About EDI System Licenses	56
Licensing Terminology	56
Checking Out Licenses for Product Options	56
Checking Out Multi-CPU Licenses	58
Additional Considerations	59
Advanced Node License Required for 32/28/20 nm and Smaller Nodes	59
3.	62
<b>Getting Started</b>	<b>62</b>
Product and Installation Information	62
Setting the Run-Time Environment	62
Supported and Compatible Platforms	62
Specifying the 64-Bit or 32-Bit Version of EDI System Applications	63
Using the CDS_AUTO_64BIT Environment Variable	63
Configuring OpenAccess	64
Launching the Console	64
Completing Command Names	64
Command-Line Editing	65
Control (^) Characters	65
Escape Sequences	66
Setting Preferences	67
Initialization Files	67
Starting the Software	68

encounter	68
Parameters	69
Using Generic Parameters to Specify 32- or 64-Bit Version	72
Interrupting the Software	73
Interrupt Behavior for Long-running Commands	73
NanoRoute Router	73
Timing Optimization (optDesign)	73
Verification	73
Interrupting the Execution of Batch Files	74
Suspending the Execution of a Script	74
Stopping the Software	74
Using the Log File Viewer	74
Integrated Log File Viewer	74
Standalone Log File Viewer	75
Accessing Documentation and Help	75
Launching Cadence Help From the Command Prompt	75
Accessing Documentation and Help From the Encounter GUI	76
Select Help on the Main Encounter Menu	76
Select Help on an Encounter Form	76
Using the Encounter man and help Commands on the Text Command Line	77
Using the help Command to View the Command Syntax	77
Using the man Command to View the Command Description	78
Using the help Command to View Message Summary	78
Using the man Command to View Message Detail	79
Using the Integrated Log File Viewer	79
Other Sources of Information	80
Accessing EDI System Tutorials	80
4.	81
Customizing the User Interface	81
Overview	81
Creating a New Menu	82
Modifying an Existing Menu	83
Adding a Menu Element to an Existing Menu	83
Replacing an Existing Menu Element	83
Adding a New Toolbar and Toolbutton	84
Supported Image Formats for Icons	85
Querying and Configuring Interface Elements	85
Iterating, Querying, and Configuring a Menu	86
Updating the Message on the Status Bar	86
Setting the Main Window's Size and Title	87
Migrating Obsolete Internal Menu APIs	87

5.		89
Accelerating the Design Process By Using Multiple-CPU Processing		89
Overview		89
Running Distributed Processing		91
Running Multi-Threading		92
Running Superthreading		92
Memory and Run Time Control		93
Checking the Distributed Computing Environment		95
Setting and Changing the License Check-Out Order		95
Limiting the Multi-CPU License Search to Specific Products		95
Releasing Licenses Before the Session Ends		96
Controlling the Level of Usage Information in the Log File		96
Where to Find More Information on Multi-CPU Licensing		96
6.		97
Data Preparation		97
Generating a Technology File		97
Creating Technology Information Using LEF		97
Creating Technology Information Using OpenAccess		97
Preparing Physical Libraries		98
Using LEF to Create Physical Libraries		98
Creating OpenAccess Physical Libraries		98
Unsupported LEF and DEF Syntax		98
Unsupported LEF 5.7 Syntax		98
Unsupported DEF 5.7 Syntax		99
Generating the I/O Assignment File		102
Creating an I/O Assignment File		102
Specifying Area I/O Information		115
Creating a Rule-Based I/O Assignment File		116
I/O Pad and Pin Assignment Examples		117
Assigning Pads for Multiple Rows		118
Assigning Module Pins		119
Recognizing Multiple Corner Cells		120
Performing Area I/O Placement		121
Defining the Connection between a Bump and P/G Pin Shape		122
Defining BUMP CELL in LEF		123
Defining BUMP CELL Placement Status		124
Importing LEF Files		124
Preparing Timing Libraries		125
Encrypting Libraries		126
Parameters		126
Preparing Timing Constraints		126

Preparing Capacitance Tables	127
Preparing Data for Delay Calculation	127
Preparing Data for Crosstalk Analysis	127
Checking Designs	127
Preparing Data in the Timing Closure Design Flow	128
Converting iPRT Format to LEF	128
<b>7.</b>	<b>129</b>
<b>Importing and Exporting Designs</b>	<b>129</b>
Overview	130
The New Design Import Use Model in EDI System 11	131
Verifying Data before Importing a Design	132
Preparing the Design Netlist	133
The <code>init_design</code> Import Flow	133
Legacy Configuration File Data Flow	133
<code>init_design</code> Simple Data Flow	134
Configuration File to MMMC Object Mapping Example	137
Supported <code>init_design</code> Invocation Methods	141
Using a Pointer to an MMMC Configuration File	141
Using a Pointer to a CPF File	142
Using <code>init_design</code> with an Inline MMMC Script	142
Using Physical-Only Flow	143
Importing Designs Saved in Previous Versions	143
<code>restoreDesign</code> of a 10.x or Earlier MMMC database	144
<code>loadConfig</code> of a 10.x or Earlier MMMC Configuration	145
<code>restoreDesign</code> of 10.x or Earlier min/max Database	146
<code>loadConfig</code> of 10.x or Earlier min/max Configuration	147
MMMC Objects Created During Design Import	147
Timing and SI Libraries	148
Extraction Data	149
Timing Constraints	150
Delay Corners	151
Multi-Supply/Multi-Voltage Domains	154
Analysis Views	156
Reporting the MMMC Configuration	157
Adapting Command Scripts for MMMC Compatibility	159
Summary of Command Changes in EDI System 11	159
MMMC Command Mapping Matrix	160
Importing Designs using the GUI	163
Importing an OpenAccess Design	163
Importing a Design with LEF and Verilog	164
Loading a Previously Saved Global Variables File	165
Handling Verilog Assigns	165

Configuring the Setup for Multi-Mode Multi-Corner Analysis	166
Creating Library Sets	166
Editing a Library Set	167
Creating Virtual Operating Conditions	168
Editing a Virtual Operating Condition	168
Creating RC Corner Objects	169
Editing an RC Corner Object	169
Creating Delay Calculation Corner Objects	170
Editing a Delay Corner Object	171
Adding a Power Domain Definition to a Delay Calculation Corner	172
Editing a Power Domain Definition	172
Creating Constraint Mode Objects	173
Editing a Constraint Mode Object	174
Entering Constraints Interactively	174
Constraint Support in Multi-Mode and Multi-Mode Multi-Corner Analysis	175
Creating Analysis Views	177
Editing an Analysis View Object	177
Setting Active Analysis Views	178
Guidelines for Setting Active Analysis Views	179
Changing the Default Active Analysis View	179
Checking the Multi-Mode Multi-Corner Configuration	179
Changing How the MMMC Browser Displays Configuration Information	179
Saving Multi-Mode Multi-Corner Configurations	180
Saving Designs	180
Saving an OpenAccess Design	181
Transferring OpenAccess Data between EDI System and Virtuoso Chip Editor for ECO	181
Loading and Saving Design Data	181
Loading a Partition	181
Loading Floorplan Data	182
Placement File Requirement	183
Loading an I/O Assignment File	183
Loading an FSDB File	183
Saving a Partition	183
Saving Floorplan Data	183
Converting an EDI System Database to GDSII Stream or OASIS Format	184
Related Topics	185
Creating Cells and Instances	185
Renaming LEF Vias	186
Merging GDSII Stream or OASIS Files	186
Merging Files Using the Command Line	186
Merge Examples	187
Case 1	187

Example 1	187
Example 2	188
Example 3	188
Example 4	188
Results	188
Case 2	188
Example 5	189
Example 6	189
Example 7	189
Example 8	189
Results	189
Merging GDS/OASIS Files Using the GUI	190
About the GDSII Stream or OASIS Map File	190
Map File Format	190
Map File Columns	191
Specifying Object Subtypes	195
Fill Subtype	195
Net Name Subtype	196
Net Name Subtype	197
Voltage Subtype	198
SIZE Subtype	198
MASK Subtype	199
Using Multiple Layers and Data Types	200
Updating Files During an EDI System Session	201
SKILL to TCL Mapping	202
8.	205
<b>Flip Chip Methodologies</b>	205
Overview	205
Related Packaging Tools	206
Before You Begin	206
Using this Chapter	206
Related Flip Chip Information	206
Flip Chip Flow in EDI System	207
Flip Chip Flow Steps	209
SiP Bump Flow	212
Reducing Data Size for SiP Import (Bypass Flow)	212
Splitting Wires in Metal Layers	212
Testing the Package Routing Feasibility	213
Area I/O Flow	213
Area I/O (AIO) Command Flow	214
Routing Bumps to I/O Driver Cells (Hierarchical Area I/O Flow)	215
Flip Chip Routing on Shielded Nets in AIO	216

Example	216
Constraint File CFG/aio.constr: Shield Net Description	216
DEF Syntax	217
Peripheral I/O Flow	217
Data Preparation	218
Peripheral I/O Flow Steps	219
Peripheral I/O (PIO) Command Flow	219
RDL Planning and Routing	221
Place peripheral I/O pads	223
Optimize peripheral I/O placement	225
Reassign bumps	226
Route bumps	226
Splitting wires	227
Adding power stripes	228
Routing the power bumps	228
Peripheral I/O Extraction	228
SI and Timing Analysis	229
Differentiating Area I/O and Peripheral I/O	230
LEF MACRO CLASS PAD and PAD AREAIO	231
Two-Layer RDL Routing	231
Handling Flip Chip Designs with Complex Floorplans	233
Pillar Bump Support	233
Viewing Flip Chip Flightlines	234
Automatic Redraw Feature	234
Selection-Based Highlighting	236
Colored Flightlines	236
Object-Specific Flightlines	237
DIFFPAIR-Based Highlighting	238
viewBumpConnection Display Rules	239
Example 1: Design has only one bump, bump_vdd, for VDD	240
Example 2: Design has multiple bumps for VDD	240
Point-To-Point Routing	243
Virtual Connection Area for RDL Routing	244
Port Numbering Feature for Power Nets	246
Multi-PG Pads to Multi Bumps Assignment with a Controlled Ratio	248
Assigning Multi-PG Pads to Bumps Using a Single Ratio	250
Example	251
Assigning Multi-PG Pads to Bumps Using Multiple Ratios	252
Example	253
Distributed Co-design	254
Swapping Signals	255
Creating Constraints for Flip Chip Routing	257

Specify Routing Nets	257
Syntax	257
Example	258
Define Differential Pairs	258
Syntax	258
Example	259
Define Nets to Match Tolerance	259
Syntax	259
Example	259
Define Splitting	259
Syntax	260
Example	260
Define a Shield Net	260
Syntax	260
Example	261
Route Multiple Nets with Different Widths	262
Example Constraints File	262
Route Nets with Tapering Pin Widths	262
Syntax	263
Example	264
Change Pin Access Direction	264
Examples and Report Files	264
Routing and Placement Constraints	265
IO_FILE Example	269
Format Definitions	270
Flip Chip Router Report	271
Format Definitions	272
ECO Routing	275
9.	278
<b>Using ART in Hierarchical Designs</b>	<b>278</b>
Overview	278
Types of Active Logic Views	278
Flat Top	279
Critical	279
Creating an Active Logic View	279
Example of Active Logic View Creation	280
The FlexView Flow	280
Overview	280
FlexView Flow Optimization Modes	280
Top-Only	280
Top-and-Partition Interface Only	281
Top and Full Partition	281

Hierarchical FlexView Flow	281
Assemble the Top Data and the Block Data	281
Verify Data Accuracy	282
Define the FlexView Mode	282
Optimize Data	282
Save the Hierarchical Data	283
Restore the Hierarchical Data	283
Verify the Optimized Data	283
Sample Summary Report	283
Sample timeDesign Summary	283
Sample optDesign Summary	284
The flexILM PreCTS Closure Flow	285
Preliminary Results on Large Design: Floorplan	285
Preliminary Results on Large Design: Memory and TAT	286
FlexILM model generation	286
Specifying and committing FlexILM	287
Distributed ECO process	287
Debugging Timing Inside and Across Partitions	287
<b>10.</b>	<b>288</b>
<b>Using Interface Logic Models in Hierarchical Designs</b>	<b>288</b>
Overview	288
Creating ILMs	289
Example ILM Creation	290
Sample Summary Report	290
Preserving Selected Instances in ILMs	291
Creating ILMs for Shared Modules	291
Creating ILMs Without Using Encounter Database	292
Specifying ILM Directories at the Top Level	293
Example Top-Level Implementation Flow with ILMs	293
ILMs Supported in MMMC Analysis	295
ILMs Supported in SI	295
SI Model Generation	296
Merged ILM Model	297
Interactive Use of ILMs	297
Handling Interactive Constraints	298
<b>11.</b>	<b>299</b>
<b>What-If Timing Analysis</b>	<b>299</b>
Performing What-If Timing Analysis	299
Prerequisite	300
Timing Models Supported for What-If Timing Analysis	300
Using the What-If Timing Commands	303

12.		306
<b>Bus Planning</b>		306
Overview		306
Bus Planning Flow in Encounter		306
Creating a Bus Guide		307
Using the Edit Bus Guide GUI		307
Drawing a Bus Guide		307
Using Text Commands		309
Example		310
Moving and Stretching a Bus Guide		311
Customizing the Bus Guide Display		311
Highlighting and Dehighlighting the Bus Guide		311
Saving and Restoring Bus Guide Information		312
Verifying Bus Guide		312
Limitations of Bus Planning		312
13.		313
<b>Partitioning the Design</b>		313
Overview		313
Flow Methodologies		313
Top-down Methodology		314
Chip Planning		314
Implementation		316
Chip Assembly		316
Bottom-up Methodology		318
Implementation		318
Block Implementation		318
Top-level Implementation		320
Chip Assembly		320
Specifying Partitions and Blackboxes		320
Defining Partitions		320
Defining Partitions as Power Domains		322
Defining Blackboxes		323
Blackbox Flow		324
Saving Blackboxes		325
Reshaping Blackboxes		325
Deleting Blackboxes		325
Handling of Blackboxes with Non-R0 Orientation		326
Automatic Conversion of Orientation		326
Performing R0 Transformation		327
Specifying Multiple Instantiated Partitions and Blackboxes		328
Changing Partition Clone Orientation		329

Specifying Rectilinear Partitions and Blackboxes	329
Specifying Core-to-I/O Distance for Partition Cuts	330
Working with Nested Partitions	331
Specifying Second-level Partitions	331
Using the Multi-level Hierarchical Flow	331
Defining Nested Partitions	333
Pin Assignment Across Nested Partitions	333
Pin Checking and Legalization Across Nested Partitions	333
Handling Pin Objects Across Nested Partitions	334
Committing Nested Partitions	334
Assembling Nested Partitions	334
Assigning Pins	335
Assigning Partition and Blackbox Pins	336
Setting Pin Constraints	336
Pin Group	336
Net Group	337
Pin Guides	338
Pin Size (Width and Height)	340
Pin Spacing	340
Pin Layers	342
Pin-to-corner distance	343
Pin Blockage	343
Performing Pin Pre-Assignment	344
Setting Constraints on a Specific Pin	344
Assigning Pins	345
Placement-based Pin Assignment	346
Route-based Pin Assignment	346
Tips for Assigning Partition Pins	347
Validating Pin Placement Results	348
Checking the Pin Legality	349
Reporting QoR of Pin Assignment	350
Refining Pin Assignment and Fixing Pin Violations	353
Adjusting Pins	353
Aligning Partition Pins	353
Running Incremental Pin Assignment	354
Adjusting Floorplan or Floorplanning the Design Again	355
Performing Pin Assignment Again	355
ECO Pin Assignment	355
General Flow	355
Saving the Partition Pins	356
Restore Partition Pin Information	356
Assigning I/O Pins	357

Setting Pin Constraints	357
Performing Initial Pin Assignment	358
Refining Pin Placement	358
Using the assignIOPins Command to Optimize I/O Placement	358
Validating Pin Placement	359
Performing Congestion-aware Pin Assignment for Channel-based Designs	360
Salient Points About Congestion-aware Pin Assignment	362
Assigning Pins on Rectilinear Edges	363
Swapping Partition Pins	363
Pin Alignment	363
Snapping Pins to the Grid	364
Assigning Pins for Bus Guides	364
Pin Assignment Limitations	365
Inserting Feedthroughs	365
Inserting Feedthrough Buffers	367
Inserting Feedback Buffers	368
Limitations	368
Procedure	369
Using a Topology File to Insert Feedthrough Buffers	369
Replicating Feedthrough Insertions Across ECO Netlists	374
Reducing the Number of Buffers and Ports Added for Route-based Feedthrough Insertions	
Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels	375
Net Connecting Through Adjoining Partition	376
Abbreviating Lengthy Feedthrough Net Names	376
Highlighting the Nets for which Feedthrough Buffers Have been Inserted	377
Utilizing Pre-defined Feedthrough Pins in Custom Macros	377
Use Flow	378
How the connectMacroFeedthrough Command Connects Feedthroughs	378
Feedthrough Connection for Abutted Macros	379
Mapping File For Describing Feedthrough Connectivity	380
Limitations	382
Inserting Routing Feedthroughs	382
Generating the Wire Crossing Report	384
Interpreting the Wire Crossing Report	385
Estimating the Routing Channel Width	387
Running the Partition Program	388
Creating a Top-Level Partition	389
Block-Level Partition	389
Pushing Down Signal Routes	390
How Top-level Stripes Are Pushed Down	390
The Default Behavior	390
Behavior with the -stripStayOnTop Option	391

How Bumps, Routes, and Area I/O Cells Are Affected	392
Area I/O Cells are Part of the Top-level Netlist	393
Area I/O Cells are Part of the Partition Netlist	394
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	394
Bumps and Routing are on Top Routing Layer--Default Behavior	395
Bumps and Routing are on Reserved Routing Layer--Default Behavior	396
Limitations	397
Case 1: All Routing Layers Reserved for the Partition	397
Case 2: Top Layer Not Reserved for Routing	399
Restoring the Top-Level Floorplan with Partition Data	400
Concatenating Netlist Files of a Partitioned Design	401
Saving Partitions	402
Loading Partitions	402
Unpartitioning with Routing Data	402
Working with OpenAccess Database	403
Parallel Job Processing	404
Setting Pin Constraints	405
<b>14.</b>	<b>406</b>
<b>Floorplanning the Design</b>	<b>406</b>
Overview	406
Common Floorplanning Sequence	407
Viewing the Floorplan	408
Module Constraint Types	409
Target Utilization Display	411
Effective Utilization Display	412
Calculating Density	413
Standard Row Spacing	413
Grouping Instances	414
Defining the Bounding Box	415
setObjFPlanBox	415
setObjFPlanBoxList	415
Adding Logical Hierarchy Without Creating Additional Hierarchy	416
Logical Hierarchy Manipulation	416
Moving Instances to a New Top Module	417
Moving Instances to an Existing Module	417
Moving Instances to the Top Root Level	418
Creating and Editing Rows	419
Using Vertical Rows	420
Limitations	420
Using Multiple-height Rows	421

Using Integer Multiple-height Rows	421
Using Non-Integer Multiple-height Rows	423
Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows	425
Merging Hierarchical Floorplans from Partitions	428
Performing I/O Row Based Pad Placement	431
Prerequisites	431
Enabling the I/O Row Flow in EDI System	432
Use Models	433
Starting a new design	433
Reading an old design	435
Resizing Rectilinear Blocks	435
Use Models	436
Assumptions	437
Results	437
Editing Pins	437
Pin Snapping on Resized Boundaries	437
Moving Pins	438
Swapping Pins	438
Using the Pin Editor	439
Using the Pin-Spreading Feature	439
Basic Concepts for Pin Spreading	439
Using the Center of a Side or Edge as the Starting Point for Spreading Pins	441
Spacing Pins Evenly Along an Edge or Side	442
Spacing Pins Evenly Using Explicit Starting and Ending Points	443
Running Relative Floorplanning	445
Orientation Key	445
Instance Place Example	446
Pre-Route Examples	446
Saving and Restoring Relative Floorplan	448
Saving and Loading Floorplan Data	448
Resizing the Floorplan	448
Resize Floorplan Options	449
Proportional Spacing	449
Shift-based Spacing	449
Congestion-based Spacing	449
Setting Resize Lines	450
Specifying Resize Directions	450
Snapping Resize Values	450
Viewing Resize Lines using Color Preferences	450
Distributing I/O's using Resize Floorplan	453
15.	455

<b>Prototyping Foundation Flow</b>	<b>455</b>
Overview	455
The Prototyping Foundation Flow Stages	456
Generate Models	457
The Model Generation Flow	459
Examples	460
Debug Constraints and Prototype Design	462
Closing the Flat Floorplan	463
Running timeDesign	463
Examples	464
Running proto_design	465
Analyze Floorplan and Adjust	465
Defining Partitions	466
Finish and Save Partition	467
Finishing the Floorplan	469
Setting the Effort	469
Analyzing the Floorplan	469
Calculating Delay Using Timing-Driven trialRoute	471
Feedthrough Insertion	472
Pin Assignment	473
Running timeDesign	473
Budget Timing	474
Save Design	474
Commit Partition	475
Save Partition	475
Restore Design	475
Advantages of Using Prototyping Foundation Flow	476
<b>16.</b>	<b>477</b>
<b>Using Structured Data Paths</b>	<b>477</b>
Overview	477
Benefits of Using SDP	477
General SDP Flow	478
Support for High-Speed Flip Flop Columns	478
SDP Placement Flow	479
placeDesign with SDP Placement	479
Sample Use Model	480
Placement with Flop Clustering	480
Overview	480
Flop Clustering Flow	480
SDP Alignment	481
Sample Use Model	481

Implementing SDP Capability	482
SDP Relative Placement File	482
SDP File Examples	482
SDP File Format	483
Reusing SDP Instantiations	485
Aligning SDPs by Pins	487
Setting SDP Options	487
Optimizing a Design with SDPs	488
Checking SDP Placement	489
<b>17.</b>	<b>490</b>
<b>Design Methodology for 3D IC with Through Silicon Via</b>	<b>490</b>
Overview	490
TSV/Bump/Back Side Metal Modeling in EDI	491
Example	492
Defining Keep Out Area in Hard Macros	493
Check Bump Keep Out Area Violation	495
3D IC Flow in EDI system	495
Design Import	497
Stacked IC Verilog Input	497
Stack Configuration Input	498
Power Connectivity Input	498
Interface Synchronization and Information Exchange between Dies	499
TSV and Bump Manipulation	500
Feedthru Handling	501
TSV and Bump Routing	501
Cross Die Connectivity Verification	502
<b>18.</b>	<b>503</b>
<b>Power Planning and Routing</b>	<b>503</b>
Overview	503
Before You Begin	504
Results	504
Loading, Saving, and Updating Special Route	505
Global Net Connections	505
globalNetConnect Command and Connections for Signal Pins and Power/Ground Pins	506
Creating a Ring with User Defined Coordinates	506
Fixing LEF Minimum Spacing Violations	507
Adding Stripes to Power Domains	508
Adding Stripe in Multi-CPU mode	509
Creating Differential Routing to Signal Bumps	510
<b>19.</b>	<b>512</b>

<b>Low Power Design</b>	<b>512</b>
Overview	513
Power Domain Shutdown and Scaling	514
Support for the Common Power Format (CPF)	515
CPF Version Support	515
EDI System Commands Supporting CPF	515
Loading and Committing a CPF File	515
Saving a CPF Database	516
CPF Documentation	516
Multiple Supply Voltage Flat Flow	517
Preparing Data	518
Preparing the Netlist	518
Preparing Libraries	518
Defining Level Shifter Cells	519
Load the design (init_design)	520
Floorplanning the Design	521
Loading and Committing the CPF File	522
Setting the Power Domain Size	523
Setting the Power Domain mingap	523
Adding Power Switches	523
Verify Power Domains	523
Adding Well Tap Cells	523
Planning Power	523
Placing Standard Cells and Macros	524
Highlight Power Domains (Optional)	526
Adding Tie High/Low cells	526
Routing Power	526
Trial Routing	527
Optimizing Timing	528
Synthesizing Clock Trees	530
Optimizing Timing (Post CTS)	530
Routing the Design	530
Analyzing Timing	531
Analyzing Power	531
Optimizing Timing (Post-Route)	531
Multiple Supply Voltage Top-Down Hierarchical Flow	531
Overview	532
Always-On Feedthrough Handling	532
Chip Partitioning	533
Block-level CPF Generation	534
Top-Level CPF Generation	535
Block-Level Implementation	535

Top-Level Implementation	535
Chip Assembly	535
Example of Block-Level CPF Generated by EDI System	536
Example of Top-Level CPF Generated by EDI System	539
Multiple Supply Voltage Bottom-Up Hierarchical Flow	543
Block-Level Implementation	544
Top-Level Implementation	544
Chip Assembly	545
Leakage Power Optimization Techniques	545
Multi-Vth Optimization	546
Optimizing Leakage Power While Running optDesign (Recommended)	547
Substrate Biasing	547
Power Shutdown Techniques	549
Data Preparation	549
Buffer Styles	550
Adding Column Switches	551
Attaching the Acknowledge Receiver Pin	552
Example	553
Enable Chaining	553
Controlling the Maximum Enable Chain Depth	556
Synthesizing Acknowledge Trees	556
Adding Power Switch Rings	557
Creating Patterns	559
Ring Conventions	559
Specifying Sides in a Switch Ring	560
Starting the Enable Chain at a Different Corner	560
Counter-Clockwise	561
Left Sides	561
Right Sides	562
Horizontal Sides	562
Vertical Sides	562
Top Sides	563
Bottom Sides	563
Using Pitch Control and Offsets	563
Forcing Offsets	565
Setting the Global Offset	565
Setting Different Offsets for Different Sides	566
Specifying Switch Location	567
Example of Offsets	568
Power Switch Optimization	570
Power Switch Reduction	570
Power Switch ECO	571

Power Switch Prototyping	571
Power Domain Parameters and Specification	572
Attribute for the domain power:	573
Attributes for the switch cells:	573
Options Summary - Switch and Power Domain	573
Switch Cell Characterization	573
Power Domain Specification	573
Options Summary - Prototyping Features	574
Chain Style Impacts on Ramp Up Time and Rush Current	575
More simultaneous chain:	575
Longer Chain Depth:	575
Ideal:	575
Prototyping Results	575
Optimal Switch Results	575
Switch Number Enumeration Results	576
Ramp Up Switch Enumeration Results	576
Number of Switches Given Current Maximum Ramp Up	577
Switch Delay Given Current Maximum Ramp Up Current	577
Ramp Up Time	578
20.	579
Placing the Design	579
Overview	580
Loading a Design	580
Preparing for Placement	580
Guiding Placement With Blockages	581
Placement Treatment of Preroutes	582
Adding Well-Tap Cells	583
Controlling the Distance Between Well-Tap Cells	583
Adding Well-Tap Cells to MSV Designs	584
Deleting Well-Tap Cells	584
Adding End-Cap Cells	584
Adding End Cap Cells to MSV Designs	585
Deleting End-Cap Cells	585
Placing Spare Cells and Spare Modules	585
Placing Spare Cells That Are Included in the Netlist	585
Related Topics	586
Placing Spare Cells That Are Not Included in the Netlist	586
Spare Cell Placement Behavior	587
Running Hierarchy-Aware Spare Cell Placement	588
Adding Padding	592
Adding Instance or Module Padding	593
Adding Instance Padding	593

Adding Module Padding	594
Adding Cell Padding	594
Placing Standard Cells	595
Related Topics	596
Running Placement in Multi-CPU Mode	596
Multi-Threading Placement Steps	597
Calculating Multi-Thread Speed-Up	599
Related Topics	599
Checking Placement	599
Using the Amoeba View	600
Using the Density Map	601
Adding Filler Cells	601
Adding Fillers to MSV Designs	602
Deleting Filler Cells	602
Placing Gate Array Style Filler Cells for Post-Mask ECO	602
Adding Decoupling Capacitance	604
Deleting Decoupling Capacitance	604
Adding Logical Tie-Off Cells	604
Saving Placement Data	605
Specifying and Placing JTAG and Other Cells Close to the I/Os	605
Related Topics	606
Optimizing and Reordering Scan Chains	606
Related Topics	606
Specifying Scan Cells	606
About Scan Chains	607
Reordering Scan Chains	607
Native Scan Reordering Approach	608
Valid Design Types	610
Scan Chains with Two-Pin Logic Cells	612
scanDEF-Based Reordering Approach	613
Using the scanReorder Command	613
Netlist-to-scanDEF Mismatch	614
scanDEF File Format	615
Valid Design Types	615
Saving Scan Files	618
Loading Scan Files	618
21.	619
Synthesizing Clock Trees	619
Before You Begin	620
Results	621
Understanding the CTS Operation Modes	621

Manual CTS Mode	621
Automatic CTS Mode	622
How CTS Calculates Skew Values	623
Improving PostRoute Correlation	624
Method 1	624
Method 2	625
Specifying Macro Model Delays	625
Macro Model Support for MMMC Views	626
Example1	626
Example2	626
Dynamic Macro Model	627
Example	627
Grouping Clocks	628
Analyzing Hierarchical Clock Trees	628
Module Placement Utilization	630
Clock Designs with Tight Area	630
Balancing Pins for Macro Models	630
Timing Model Requirement for Cells	630
Delay Variation and OCV	630
Understanding Post-CTS Clock Tree Optimization	631
Using the ckECO Command for Post-CTS Clock Tree Optimization	631
Support for Local Skew Optimization	632
Command Modes for the ckECO Command	632
Using a SPEF File with the ckECO Command for RC Estimation	632
Running Post-CTS Optimization with the ckECO Command	633
Example 1	633
Example 2	633
Guidelines for Using the ckECO Command	634
Creating a Clock Tree Specification File	634
Using the Automatic Clock Tree Specification File Generator	635
Example of a Clock Tree Specification File	636
Naming Attributes Section	641
NanoRoute Attribute Section	641
Macro Model Data Section	643
Clock Grouping Data Section	647
Clock-Tree Topology Section	647
Automatic Gated CTS Section	647
Log File Headings	665
CTS Report Descriptions	665
General Information	665
Macro Model Information	668
Power Information	668

AC Current Density Violations	669
Supported SDC Constraints	669
Clock Concurrent Optimization	670
Overview	670
The CCOpt Flow in the EDI System	671
The Scripted Integration Mode	672
The Scripted CCOpt Flow	672
Native Integration Mode	675
The Native CCOpt Flow	675
CCOpt Properties and Configuration	677
Setting CCOpt Properties	678
Specifying the property name, value, object type, and object pattern	678
The property name, value, object type, and object pattern need to be specified while setting CCOpt properties. For example:	678
Using Wildcards	679
Using Properties with Switches	679
Getting Properties from Tcl	680
Getting a List of Properties and their Detailed Descriptions	681
Key-Value Pairs	683
Defaults for Key Value Pairs	683
CCOpt Clock Tree Specification	684
Clock Trees and Skew Groups	684
Example1: Balancing Shared Sinks	685
Example2: Balancing Independent Clock Trees	685
Example3: Balancing Flops with Clock Trees	686
Automatic Extraction of Clock Trees	687
Defining Clock Trees	688
Configuring Stop and Ignore Pins	688
Defining Clock Trees	688
Defining Generated Clock Trees	689
Setting Pin Insertion Delays	691
Defining Skew Groups	693
Skew Group Rank and Active Sinks	695
How Automatic Clock Tree Extraction Works	696
Backtracking During Automatic Extraction	696
Automatic Extraction and Multiple Outputs	697
Keeping All SDC Clocks	698
Reporting on Clock Trees	699
Clock Tree Summary	700
Restricting the Clock Trees Report to One Corner	700
Reporting on Skew Groups	701
CCOpt Clock Tree Debugger	703

22.		706
Working with Clock Mesh Structures		706
Overview		706
Clock Meshes Versus Clock Trees		706
Creating Clock Meshes		709
Determining the Mesh Structure		709
Supported Mesh Styles		709
Clock Mesh Structure Characteristics		711
Multilevel Structure of a Mesh		712
Implementing the Clock Mesh		713
Analyzing the Clock Mesh		714
Pre-Route Wire Estimation		714
RC Extraction		714
Computing Mesh Delays		715
Generating Multiple Spice Run Deck For Big Clock-Mesh Networks		716
Steps to Generate Multi-Part Spice		716
Sample Scripts to Run Spice Simulation		717
Simulate Automatically With UltraSim		717
Simulate with User-Defined Procedure		717
Simulate Manually		718
MultiSpine Clock Mesh		718
23.		720
Editing Wires		720
Overview		720
Before You Begin		721
Results		721
Using Keyboard Shortcuts		721
Keyboard Shortcuts That Open Forms		721
Keyboard Shortcuts That Are Equivalent to Tool Widgets		722
Keyboard Shortcuts Used in Auto Query Mode		722
Keyboard Shortcuts Used in Edit Wire Mode		723
Keyboard Shortcuts Used in Stretch Wire Mode		723
Keyboard Shortcuts Used to Change Vias		724
Selecting Wires		724
Deleting Wires		724
Moving Wires		725
Using the Mouse to Move Wires		725
Using Arrow Keys to Move Wires		725
Moving Selected Wires or Vias		726
Adding Wires		726
Adding a Wire for a Single Net		726

Adding Wires for Multiple Nets	728
Adding Wires that Automatically Extend to a Target	729
Using Override to Add Wire Groups with Multiple Widths and Spacing	729
Cutting Shielding Wires	730
Trimming Antennas on Selected Stripes	731
Changing Wire Width	731
Repairing Maximum Wire Width Violations	732
Duplicating Wires	732
Stretching Wires	733
Changing Wire Layers	733
Splitting and Merging Wires	734
Adding Vias	734
Changing Vias	734
Moving Vias	735
Reshaping Routes	735
Controlling Cell Blockage Visibility	736
<b>24.</b>	<b>738</b>
<b>Using Trial Route for Congestion and Timing Analysis</b>	<b>738</b>
Overview	738
Data Preparation	739
Routing A Flat Design	739
Routing a Partitioned Design	740
Routing Two-Metal Layer Designs	742
Routing Using the NanoRoute Global Router	742
Loading and Saving Route Data	743
Analyzing Route Data	743
Congestion Markers in the Display	744
Congestion Marker Color Boxes	745
Congestion Distribution Report	746
Default Congestion Distribution Report	746
Usage and Routing Overflow	746
Gcell Overflow	747
Detailed Congestion Distribution Report	749
Virtual (global) wire length	749
Range of tracks in a gcell	750
Number of gcells with remaining tracks, including blocked gcells	750
Number of gcells with remaining tracks, excluding blocked gcells	752
Track usage in gcells	753
Real wire length	753
Improving Route Congestion	754
Using Bus Guides	755
Additional Information	755

Wire Overlap	756
<b>25.</b>	<b>757</b>
<b>Using the NanoRoute Router</b>	<b>757</b>
About NanoRoute Routing Technology	758
Routing Phases	758
Global Routing	759
Related Topics	759
Detailed Routing	759
NanoRoute Router in the EDI System Flow	759
Before You Begin	760
Checking Your LEF Files	760
Related Topics	761
Checking for Problems with Cells, Pins, and Vias	761
Generating Tracks	761
Related Topics	761
Specifying Routing Layers	761
Specifying Hard Layer Limits	762
Specifying Soft Layer Limits	762
Interrupting Routing	762
Using the routeDesign Supercommand	763
Related Topics	763
Results	764
Use Models	764
Running the NanoRoute Router with EDI System Menu Commands and Forms	764
Running the NanoRoute Router with EDI System Text Commands	765
Running the NanoRoute Router in Standalone Mode	765
Using NanoRoute Parameters	765
Using Attributes and Options Together	767
Accelerating Routing with Multi-Threading and Superthreading	767
Related Topics	767
When to Accelerate Routing	768
Usage Notes	768
Superthreading Log File Excerpts	768
Following a Basic Routing Strategy	769
Using the EDI SystemText Commands	770
Using the EDI System GUI	770
Run Global Routing	771
Run Initial Detailed Routing	771
Run Search and Repair	771
Run Postroute Optimization	772
Checking Congestion	772

Using the Congestion Analysis Table	772
Interpreting the Table	774
Using the Congestion Map	774
Interpreting the Congestion Map	774
Resolving Open Nets	776
Log File Examples	777
Diagnosing Problems Using verifyTracks	777
Resolving Additional Open Net Problems	778
Running Timing-Driven Routing	779
Input Files	779
Using the CTE and the NanoRoute Router in Native Mode	779
Using the CTE and Standalone NanoRoute	779
Routing Clocks	780
Setting Attributes for Clock Nets	781
Routing Clock Nets Using the GUI Forms	781
Running Postroute Optimization	782
Related Topics	782
Preventing and Repairing Crosstalk Problems	782
Related Topics	783
Crosstalk Prevention Options	783
Running ECO Routing	784
ECO Limitations	785
ECO Flow	785
Specifying Nets for ECO Routing	785
ECO Routing After Multiple-Cut Via Insertion	785
Evaluating Violations	785
Violations on Upper Metal Layers	788
Violations in Timing-Driven Routing	790
Deleting Violated Nets	791
Related Topics	791
Using Additional Strategies to Repair Violations	791
Process Antenna Violations	791
Core Congestion	791
Concurrent Routing and Multi-Cut Via Insertion	791
Postroute Via Optimization	792
Related Topics	792
Optimizing Vias in Selected Nets	792
Via Optimization Options	793
Performing Shielded Routing	793
Shielding Option	794
Performing Shielded Routing Using the GUI	794
Performing Shielded Routing Using Text Commands	795

Interpreting the Shielding Report	795
Routing Wide Wires	796
Using Non-Default Rules	796
Repairing Process Antenna Violations	797
Repairing Violations on Multiple-Pin Nets	798
Changing Layers	798
Using Diodes	798
Deleting and Rerouting Nets with Violations	798
Repairing Violations on Cut Layers	798
Process Antenna Options	799
Examples	799
Using a Design Flow that Includes Astro or Apollo	800
Troubleshooting	801
<b>26.</b>	<b>802</b>
<b>Optimizing Metal Density</b>	<b>802</b>
Overview	802
Before You Begin	803
Adding Metal Fill in the Multiple-CPU Processing Mode	803
After You Complete Adding Via and Metal Fills	804
Metal Fill Features	804
Staggered Metal Fill Pattern	804
Connected and Floating Metal Fill	805
Timing-Aware Metal Fill	809
Specifying Metal Fill Parameters	811
Recommendations for Adding Timing-Aware Metal Fill	812
Timing-Aware Examples	813
Specifying the Active Spacing Value	814
Adding Metal Fill Over Macros	815
Estimating Density of Blockage	816
Estimating Density of BLOCK Cell	816
Recommendations for Power Strapping Mode	817
Adding Via Fill	818
Recommendations for Metal/Via Fill Flow	819
Recommendations for In-design Sign-off Metal Fill Flow	822
Achieving Gradient Density with Preferred Density Setting	824
Specifying Metal Fill Spacing Table	826
Trimming Metal Fill	829
Trimming Metal Fill for Timing Closure	831
Verifying Metal Density	834
Adding Metal Fill Using the GUI	835
Adding Metal Fill with Iteration	835

27.		838
Timing Budgeting		838
Overview		839
Is My Design Ready for Budgeting?		840
Deriving Timing Budgets		841
Budgeting Using the GUI		841
Budgeting Using Text Commands		841
Top-Level Budgets Derived by Using Active Logic View		842
Deriving Preliminary Budgets in Early Design Phase		843
Budgeting Output Files for MMMC Designs		844
Corner Cloning		845
Mode Cloning		846
Setup and View Handling for MMMC Designs		846
Master Clone Budgeting		847
Constraints Adjustment		850
Analyzing Timing Budgets		851
Resolving Conflicts with Path-Based Exceptions		852
Examples		852
Case 1		852
Case 2		853
Case 3		854
Case 4		854
Budgeting Clock Latency in Propagated Mode		855
Budgeting Libraries		856
Resolving Conflicts with Path-based Exceptions		856
Case 1: A single cycle path and a multicycle path through the partition port		856
Case 2: A single cycle path and two different multicycle paths through the partition port		857
Defining Clocks Inside the Partition		858
Power Pin Support in Budgeted Timing Models for Low Power Designs		859
Calculating Timing Budgets		860
Customizing Budget Generation		862
Verifying Timing Budgets		863
Modifying Budgets		864
Reading the Justify Budget Report		865
Design Example		868
SDC Constraints for Design Example		869
Generated Report for Design Example		869
Generate Summarized Report of Budget Data		871
Support for Distributed Processing in Budgeting		873
Example		874
Constraints Support in Budgeting		874
Warning Report		878

Pin Constraint Values Greater than Available Time	878
Warning Report Example	878
<b>28.</b>	<b>881</b>
<b>RC Extraction</b>	<b>881</b>
Overview	881
Before You Begin	882
Results	883
Specifying Temporary File Locations	883
Extraction Flow in EDI System	883
PreRoute Extraction	884
PostRoute Extraction	884
Native Detailed	884
TQRC and IQRC	885
Examples of Incremental Extraction Support	886
Sign-Off Extraction Using QRC	887
Inputs for QRC Sign-Off Extraction	887
Scale Factor Setting	887
Generating a Capacitance Table	888
Inputs for Generating a Capacitance Table	888
Capacitance Table Generation Flow	889
Capacitance Table Examples	890
Generating Capacitance Table with Specified Scale Factors	894
Reading a Capacitance Table	894
Reading a QRC Techfile	895
PreRoute Extraction Flow without Capacitance Table Data	895
For designs above 32nm	896
Correlating Native Extraction With Sign-Off Extraction	896
Correlating SPEF Files Using the Ostrich Utility	897
Comparing SPEF Files Using a Perl Script	899
Report File Example	900
Defining the Scale Factor	902
Example:	903
Distributed Processing	903
Setting-up Distributed Processing	903
Generating a Capacitance Table in Multi-CPU Mode	904
TCL Script to Run the generateCapTbl Command in the Distributed Mode	904
TCL Script to Run the generateCapTbl Command in the Local Mode	904
Performing IQRC, TQRC, and Standalone QRC Extraction in Multi-CPU Mode	904
TCL Script for IQRC, TQRC, and Standalone QRC Extraction Invoked in the Distributed Mode	904
TCL Script for IQRC, TQRC, and Standalone QRC Extraction Invoked in the Local Mode	904
Setting Scale Factors in Capacitance Table File for PreRoute Extraction	905904

Overview	905
Key Characteristics of the Feature	905
<b>29.</b>	<b>908</b>
<b>Calculating Delay</b>	<b>908</b>
Overview	908
Data Preparation	908
Operating Conditions	909
ECSM/CCS Libraries	909
Delay Calculation Modes and Related Controls	909
Choosing A Delay Calculation Engine	911
Running Delay Calculation	911
Calculating Delay in Multi-Thread Mode	911
Example:	912
<b>30.</b>	<b>913</b>
<b>Timing Analysis</b>	<b>913</b>
Overview	913
Timing Analysis Features	913
Static Timing Analyzer (STA)	913
What-If Timing Analysis	914
Wireload Model Generation in Hierarchical and Flat Format	914
Minimum and Maximum Timing Analysis	914
Timing Analysis Ideal and Propagated Modes	914
MMMC-On By Default Functionality	915
Before You Begin	915
Calculating Clock Latency	916
Specifying Timing Analysis Modes	917
Definition of Early and Late Paths	917
Single Timing Analysis Mode	918
Setup Check in Single Timing Analysis Mode	918
Hold Check in Single Timing Analysis Mode	918
Performing Timing Analysis in Single Analysis Mode	921
Best-Case Worst-Case (BC-WC) Timing Analysis Mode	922
Setup Check in BC-WC Mode	922
HOLD Check in BC-WC Mode	923
Performing Timing Analysis in BC-WC Analysis Mode	925
On-Chip Variation (OCV) Timing Analysis Mode	927
Setup Check	927
Hold Check	928
Performing Timing Analysis in OCV Mode with Two Libraries And Operating Conditions	930
Using set_timing_derate with OCV Analysis Mode	931
Clock Path Pessimism Removal	932

CPPR and Reconvergent Logic	934
CPPR Flow	935
Timing Analysis Results Before and After CPPR	935
Analyzing Timing Problems	939
Resolving Buffer-Related Problems	941
<b>31.</b>	<b>942</b>
<b>    Debugging Timing Results</b>	<b>942</b>
Overview	942
Timing Debug Flow	942
Generating Timing Debug Report	943
Displaying Violation Report	943
Analyzing Timing Results	944
Viewing Power Domain Information	947
Creating Path Categories	948
Creating Predefined Categories	948
Creating New Categories	948
Creating Sub-Categories through the GUI	949
Creating Sub-Categories through Command Line	950
Viewing Sub-Categories	951
Hiding path categories	951
Reporting Path Categories	951
Using Categories to Analyze Timing Results	952
Analyzing MMMC Categories	953
Manual Slack Correction of Categories	955
Editing Table Columns	956
Cell Coloring	957
Viewing Schematics	957
Running Timing Debug with Interface Logic Models	958
<b>32.</b>	<b>960</b>
<b>    Statistical Static Timing Analysis</b>	<b>960</b>
SSTA Overview	960
SSTA Inputs	962
Libraries with sensitivities	963
Statistical Parameter Distribution Format (SPDF) File	964
Specifying Global or Die-to-Die Variations in SPDF File	964
Specifying Random Variations in SPDF File	964
Specifying Spatial Variations in SPDF File	965
Sensitivity-Based SPEF (S-SPEF) File	966
Loading the S-SPEF File	966
SSTA Flows	966
Running Block-Based SSTA	967

Running Path-Based SSTA	967
Running WorstRC Corner SSTA	968
SSTA Outputs	968
SSTA Reporting Options	968
JPDF Report	969
Block-Based SSTA Report	970
Path-Based SSTA Report	970
Path-Based SSTA Report	970
WorstRC SSTA Report	972
WorstRC SSTA Report	972
SSTA Correlation With Monte-Carlo Analysis	973
<b>33.</b>	<b>975</b>
<b>Extracting Timing Models</b>	<b>975</b>
ETM Overview	975
Using ETMs in Different Timing Analysis Modes	976
Limitation of Timing Models	977
ETM Inputs	978
Guidelines for Generating ETMs	978
ETM Generation Flow	981
Validating the Generated Model	983
Reducing the Size of GreyBox Models	984
ETM Outputs	985
Timing Library File	986
Boundary Nets	986
Internal Nets	986
Timing Paths	987
Input to Register Paths	987
Register to Output Paths	987
Input to Output Paths	988
Minimum Pulse Width and Minimum Period	988
Path Exceptions	988
Constants	989
Gating Checks	989
Simple Clock Gating with AND Gate	989
No Clock Gating Logic	989
Annotated Delays and Slews	990
Design Rules	990
Generated Clocks	991
Handling Multiple Clocks on Same Pin	992
Handling Generated Clock on Multiple Pins	993
Timing Constraints Files	994
set false path and set multicycle path constraints	994

set_disable_timing and set_case_analysis	995
create_clock and create_generated_clock	995
set_input_delay and set_output_delay	995
Design Rules	995
set_load, set_resistance and set_annotated_transition	995
set_annotated_delay and set_annotated_check	996
set_input_transition and set_driving_cell	996
<b>34.</b>	<b>997</b>
<b>Optimizing Timing</b>	<b>997</b>
Overview	998
Before You Begin	998
Results	999
Interrupting Timing Optimization	1001
Performing Optimization Before Clock Tree Synthesis	1002
Correcting Violations in Pre-CTS Mode for the First Time	1002
Performing Rapid Timing Optimization for Design Prototyping	1003
Using Additional Pre-CTS Timing Optimization Parameters	1003
Performing Incremental Pre-CTS Optimization	1004
Changing Default Settings in Pre-CTS Mode	1004
Performing Post-CTS Optimization	1005
Correcting Violations in Post-CTS Mode	1005
Skipping Path Groups or Clock Domains During Hold Fixing	1006
Using Additional Post-CTS Timing Optimization Parameters	1006
Performing Incremental Post-CTS Optimization	1007
Changing Default Settings in Post-CTS Mode	1008
Performing Postroute Optimization	1009
About Postroute Optimization	1010
Correcting Violations & Signal Integrity Issues using GigaOpt Technology in Postroute Mode	1011
GigaOpt in PostRoute Setup Timing Flow	1011
GigaOpt in PostRoute Hold Timing Flow	1011
GigaOpt in PostRoute Use Model	1012
Changing Default Settings in Postroute Mode	1015
Optimizing Power During optDesign	1015
Leakage Power Optimization	1015
Dynamic Power Optimization	1016
Using Useful Skew	1016
Using Useful Skew in Pre-CTS Mode	1016
Using Useful Skew in Post-CTS Mode	1017
Controlling Useful Skew Optimization	1017
Distributed Timing Analysis for Hold Fixing	1019
Using Active Logic View for Chip-Level Interface Circuit Timing Closure	1019

Optimizing Timing in On-Chip Variation Analysis Mode	1020
Specifying the MMMC Environment	1021
Optimizing Timing in OCV Mode Using the Default Delay Calculator	1024
Optimizing Timing in OCV Mode Using the Sign-Off Delay Calculator	1024
Using Conformal Constraint Designer During Timing Optimization	1024
Post-Processing Approach	1025
Integrated Approach	1025
Optimizing Timing Using a Rule File	1028
Optimizing Timing When the Constraint File Includes the <code>set_case_analysis</code> Constraint	1028
Using the Footprintless Flow	1028
Using Cell Footprints	1030
Viewing Added Buffers, Instances, and Nets	1030
Default Naming Conventions	1030
<b>35.</b>	<b>1033</b>
<b>Interactive ECO</b>	<b>1033</b>
Overview	1033
Before You Begin	1033
Results	1033
Adding Buffers	1034
Changing the Cell	1036
Deleting Buffers	1038
Displaying Buffer Trees	1040
Running ECO Placement	1042
Naming Conventions for Interactive ECO	1042
Comparing Physical Design Data	1042
<b>36.</b>	<b>1048</b>
<b>Integration with LPA and CCP</b>	<b>1048</b>
Overview	1048
Before You Begin	1048
Results	1048
Running LPA from Encounter	1049
Routing Layers Only Mode	1049
Running LPA in Routing Layers Only Mode	1049
Viewing Hotspots	1053
Fixing Hotspots	1055
Sign-Off Mode	1056
Running LPA in Sign-Off Mode	1056
Viewing Hotspots	1062
Fixing Hotspots	1064
Running CCP from Encounter	1064
CCP Flow in Encounter	1065

Running CCP in Cadence Model Flow	1065
Running CMP Analysis in TSMC Model Flow	1069
Viewing Hotspots	1070
<b>37.</b>	<b>1072</b>
<b>Analyzing and Repairing Crosstalk</b>	<b>1072</b>
Overview	1072
Inputs and Outputs for SI Analysis	1073
Setting Up Encounter for SI Analysis	1074
RC Extraction Settings	1074
Extraction Engine	1074
Extraction Filters	1075
Effect of RC Extraction Settings on SI Analysis	1075
Guidelines for RC Extraction Settings	1075
Noise Analysis Settings	1076
Noise Models	1076
Timing Windows	1076
Internally Generated Timing Windows	1077
Externally Generated Timing Windows	1077
Noise Analysis Engine	1078
Delta Delay Threshold	1078
Virtual Attacker Mode	1079
Guidelines for Noise Analysis Settings	1079
Static Timing Analysis (STA) Settings	1079
Input Timing Library	1079
STA Engine	1079
Analysis Conditions	1080
Advanced Settings for SI Analysis	1081
Multi-CPU Processing Settings	1081
Setting Up Multi-Threading for Noise Analysis	1081
Setting Up Distributed Processing for Noise Analysis	1082
Setting Up Super-Threaded Noise Analysis for MMMC Designs	1082
Examples of Setting Up Multi-CPU Processing	1083
Path Group Settings for SI Optimization	1084
Example of Setting Up Encounter for SI Analysis	1085
Preventing Crosstalk Violations	1086
Fixing Crosstalk Violations	1087
Data Preparation	1087
Using optDesign to Fix Setup Violations with Crosstalk Effects	1088
Using RC Data Generated by an External Tool for SI Fixing	1088
Using SDF Data Generated by an External Tool for SI Fixing	1088
Fixing Setup Violations Using External SDF for Non-MMMC Designs	1089
Fixing Setup Violations Using External SDF for MMMC Designs	1089

Using optDesign to Fix Hold Violations with Crosstalk Effects	1089
Using RC Data Generated by an External Tool for SI Hold Fixing	1090
Using SDF Data Generated by an External Tool for SI Hold Fixing	1090
Fixing Hold Violations Using External SDF for Non-MMMC Designs	1090
Fixing Hold Violations Using External SDF for MMC Designs	1091
Using optDesign to Fix Transition Time Violations with Crosstalk Effects	1091
SI Transition Violation Fixing	1091
Transition Violation Fixing Using Transition File Generated During Noise Analysis Without Glitch or Noise-On-Delay Fixing	1092
Transition Violation Fixing Using External Transition File(s)	1092
External Transition and SDF Files Used for Transition Violation and Delay Fixing	1094
Performing XILM-Based SI Analysis and Fixing	1095
<b>38.</b>	<b>1096</b>
<b>Power and Rail Analysis</b>	<b>1096</b>
Overview	1096
Early Rail Analysis	1096
Early Rail Analysis Key Features	1098
Prior to Running Early Rail Analysis	1099
Setting up and Running Early Rail Analysis	1100
CPF Support for ERA	1113
Running Early Rail Analysis in Unplaced Mode	1113
Viewing Early Rail Analysis Results	1114
Signoff-Rail Analysis	1118
Electrostatic Discharge Analysis	1118
EDI System and EPS menu differences	1119
<b>39.</b>	<b>1121</b>
<b>Identifying and Viewing Violations</b>	<b>1121</b>
Overview	1121
Interrupting Verification	1123
Verifying Connectivity	1124
Before You Begin	1124
Types of Connectivity Violations Reported	1124
Results	1125
Verifying Metal Density	1125
Before You Begin	1125
Metal Density Statements in the LEF File	1126
Results	1126
Verifying Metal Density in Multi-Thread Mode	1126
Related Topics	1126
Verifying Geometry	1126
Before You Begin	1127

Verify Geometry Statements in the LEF File	1127
Verifying Geometry in Multi-Thread Mode	1128
Related Topics	1129
Spacing Violation Checks	1129
Types of Antenna Violations Reported	1130
Support for Via Rules	1130
Results	1131
Verifying Process Antennas	1131
Before You Begin	1131
Verifying PAE	1132
Results	1132
Sample Process Antenna Report	1132
Verifying Well-Process-Antenna Violations	1134
Sample verifyWellAntenna Report	1134
Verifying End Cap Violations	1136
Results	1137
Sample Verify End Cap Report	1138
Verifying Maximum Floating Area Violations	1140
Verifying AC Limit	1140
Overview	1140
Calculating Irms Waveform	1141
Calculating Ipeak Waveform	1144
Calculating Iavg Waveform	1146
Calculating Effective Frequency	1148
Computing Irms /Ipeak /Iavg for each Routing Segment	1150
Checking the AC Current Limits	1151
RMS/Peak/Avg Current Limit Violations	1152
Before You Begin	1158
Results	1158
Verifying Isolated Cuts	1158
Viewing Violations With the Violation Browser	1158
Viewing Geometry or Metal Density Violations	1158
Viewing Connectivity, Process Antenna, or AC Limit Violations	1159
Viewing Violation Markers From Assura, Calibre, PVS, or Other Software Applications	1159
Violation Browser Features	1159
Clearing Violations	1161
40.	1162
Power Analysis and Reports	1162
Static Power Analysis Overview	1162
Type of power (internal, leakage, switching)	1162
Definition of activity, duty cycle, and transition density	1162
How PM calculates internal, leakage and switching power including state dependency	1163

State Dependent Internal Power (input pins)	1163
State Dependent Arc-based Internal Power (output pins)	1164
State Dependent Leakage Power	1166
Switching Power	1168
Vector-based Average Power Calculation	1169
Propagation-based average power calculation	1169
Activity Propagation in the power engine	1169
Activity propagation through combinational cells	1169
Activity propagation through sequential Cells	1169
Activity propagation through macros	1170
Activity propagation through clock network and clock gates	1170
Recommended methodology for activity propagation	1170
Static Power Analysis Flow	1172
Set Power Analysis Mode	1172
Run Power Analysis	1172
TCL Command:	1175
MMMC mode default views	1176
Static Power Reports	1177
TCL Command:	1178
Static Power Analysis Plots	1178
TCL Command:	1180
Viewing and Debugging Static Plots	1180
Interactive Queries of Power Data	1181
Static Power Histograms and Pie-charts	1181
41.	1183
<b>Creating An Initial Floorplan Using Automatic Floorplan Synthesis</b>	<b>1183</b>
Overview	1183
Automatic Floorplan Synthesis Flow	1184
Data Preparation	1185
Selecting Seeds	1186
Automatic Seed Selection	1186
User-Specified Seed Selection	1187
Creating a Seed Section In the Constraint File	1189
Importing the Design	1193
Setting Automatic Floorplan Synthesis Global Parameters	1193
Creating an Initial Floorplan	1194
Creating Floorplan for Hierarchical Design	1195
Macro placement	1196
Full-chip Floorplan	1197
Power-Domain Aware Floorplan	1197
Creating Multiple Alternative Floorplans	1198

Analyzing the Floorplan	1199
Adjusting Macro Placement	1200
Manual Macro Adjustment	1201
Automatic Floorplan Synthesis Macro Adjustment	1201
Adjusting the Placement of a Specific Macro Pack	1201
Adjusting Macro Placement Within a Specified Area	1202
Marking Refinement Steps	1205
Restoring Refinement Steps	1206
Saving the Floorplan	1206
42.	1207
<b>Creating the ICT File</b>	<b>1207</b>
Overview	1207
Format	1208
Data	1208
Comments	1208
Case Sensitivity	1208
Warnings and Errors	1208
Invalid Layer Names	1209
Commands	1209
Process	1209
Syntax	1209
Example	1209
Well	1210
Syntax	1210
Example	1210
Conductor	1210
Syntax	1210
Required Conductor Command Fields	1214
Wire-Width Values	1215
Example File for Conductor Definition	1216
Dielectric	1217
Syntax	1218
Passivation	1219
Syntax	1219
Example	1220
Via	1220
Syntax	1220
Example	1221
Sample ICT File	1222
43.	1241
<b>ECO Flows</b>	<b>1241</b>

Overview	1241
Assumptions	1241
Flows	1242
Pre-Mask ECO Changes from a New Verilog File	1242
Preparation	1242
Steps	1243
Read the new netlist.	1243
Pre-Mask ECO Changes from a New DEF File	1245
Preparation	1246
Flow	1246
Steps	1247
Pre-Mask ECO Changes from an ECO File	1249
Preparation	1249
Flow	1249
Steps	1250
Post-Mask ECO Changes from a New Verilog Netlist (Using Spare Cells Flow)	1252
Post-Mask ECO Changes from a New Netlist (Using Gate Array Cells Flow)	1256
Post-Mask ECO Changes from a New Verilog Netlist (Using Gate Array Filler Cells Flow)	1259
44.	1262
<b>ECO Directives</b>	<b>1262</b>
<b>ADDHIERINST</b>	<b>1263</b>
Parameters	1263
Example	1264
<b>ADDINST</b>	<b>1264</b>
Parameters	1264
Example	1265
<b>ADDMODULEPORT</b>	<b>1265</b>
Parameters	1266
Examples	1266
<b>ADDNET</b>	<b>1267</b>
Parameters	1267
Example	1267
<b>ATTACHMODULEPORT</b>	<b>1268</b>
Parameters	1268
Examples	1268
<b>ATTACHTERM</b>	<b>1268</b>
Parameters	1269
Examples	1270
<b>CHANGEINSTNAME</b>	<b>1270</b>
Parameters	1271
<b>DELETEBUFFER</b>	<b>1271</b>
Parameters	1271

Example	1272
<b>DELETEINST</b>	1272
Parameters	1272
Example	1272
<b>DELETEMODULEPORT</b>	1273
Parameters	1273
Example	1273
<b>DELETENET</b>	1273
Parameters	1274
Example	1274
<b>DETACHMODULEPORT</b>	1274
Parameters	1274
Example	1275
<b>DETACHTERM</b>	1275
Parameters	1275
Example	1276
<b>INSERTBUFFER</b>	1276
Parameters	1277
Example	1278
Example ECO File	1279
<b>HECO Directives</b>	1282
HECO Syntax	1282
Examples	1283
<b>45.</b>	1287
<b>Verifying Well Pins and Bias Pins</b>	1287
Overview	1287
Flow	1287
Adding Information to the Technology and Cell LEF Files	1288
LEF File Example (Row-Based Checking)	1288
LEF File Example (Block-Based Checking)	1292
Specifying Connections of Pins to Wells	1293
Validating Connections	1293
Validating Width, Spacing, and Shorts	1293
Exporting the Verilog Netlist	1293
Important Considerations for Usage	1294
<b>46.</b>	1295
<b>Clock Mesh Specification File</b>	1295
Overview	1295
Routing Type Definitions	1296
Cutout Definitions	1296
Clock Mesh Definitions	1297

Timing and Power Constraints Section	1298
Tracing and Analysis Scope Section	1300
Mesh Structure Section	1301
Global Mesh Section	1302
Multispine Clock Mesh	1314
The MainDrive Section	1315
The SpineTemplate Section	1315
The Spine Section	1316
The Stage Definition for the MainDrive Section	1317
The Stage Definition for the SpineTemplate Section	1317
The Stage Definition for the Spine Section	1318
The Mesh Section	1318
Defining Attributes of MultiSpine Mesh	1319
Analysis Section	1320
Top Chain Section	1321
Local Tree Section	1323
Clock Mesh Specification File Example	1326
47.	1339
Supported CPF 1.0 Commands	1339
48.	1352
CPF 1.0 Script Example	1352
49.	1370
Supported CPF 1.0e Commands	1370
50.	1386
CPF 1.0e Script Example	1386
51.	1399
Supported CPF 1.1 Commands	1399
52.	1415
CPF 1.1 Script Example	1415
53.	1428
Cadence-Specific Liberty Extensions	1428
Overview	1428
Guidelines For Adding ECSM Extensions	1429
Representing ECSM Information in a Library	1429
Defining ECSM Extensions in a Library	1430
ecsm_waveform Group	1432
ecsm_waveform Attributes	1435
Waveform Order and Size	1436

ecsm_capacitance Group	1436
ecsm_capacitance Attributes	1438
Example	1439

---

# About This Manual

---

The Cadence® Encounter® Digital 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 Encounter Digital Implementation System (EDI System) to implement digital integrated circuits.

## Audience

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

## How This Manual Is Organized

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

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

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

- Chapter 2, "[Getting Started](#)"  
Describes how to install, set up, and run the EDI System software, and use the online Help system.
- Chapter 6, "[Data Preparation](#)"  
Describes how to prepare data for import into the EDI System software.

## Conventions Used in This Manual

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

text

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

	<code>analyze_connectivity -analyze all</code>
<i>text</i>	<p>Indicates information for which you must substitute a name or value.</p> <p>In the following example, you must substitute the name of a specific file for <i>configfile</i>:</p> <pre>wroute -filename configfile</pre>
<i>text</i>	<p>Indicates the following:</p> <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>
[ ]	<p>Indicates optional arguments.</p> <p>In the following example, you can specify none, one, or both of the bracketed arguments:</p> <pre>command [-arg1] [arg2 value]</pre>
[   ]	<p>Indicates an optional choice from a mutually exclusive list.</p> <p>In the following example, you can specify any of the arguments or none of the arguments, but you cannot specify more than one:</p> <pre>command [arg1   arg2   arg3   arg4]</pre>
{   }	<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</p>

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

## Related Documents

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

### EDI System Product Documentation

- [What's New in EDI System](#)  
Provides information about new and changed features in this release of the EDI System family of products.
- [EDI System Known Problems and Solutions](#)  
Describes important Cadence Change Requests (CCRs) for the EDI System family of products, including solutions for working around known problems.
- [EDI System Text Command Reference](#)

Describes the EDI System text commands, including syntax and examples.

- [EDI System Menu Reference](#)

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

- [EDI System Database Access Command Reference](#)

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

- [EDI System Foundation Flows User Guide](#)

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

- [EDI System Library Development Guide](#)

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

- [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).

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

---

# Product and Licensing Information

---

- [Overview](#)
- [About EDI System Products and Product Options](#)
  - [EDI System](#)
  - [First Encounter Hierarchical Prototyping Solution](#)
  - [EDI System Product Options](#)
- [About EDI System Licenses](#)
  - [Licensing Terminology](#)
  - [Checking Out Licenses for Product Options](#)
  - [Checking Out Multi-CPU Licenses](#)
    - [Additional Considerations](#)
  - [Advanced Node License Required for 32/28/20 nm and Smaller Nodes](#)

## Overview

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

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

## About EDI System Products and Product Options

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

- EDI System
- First Encounter Hierarchical Prototyping Solution
- EDI System Product Options

### EDI System

This package includes the products listed in Table 1-1. To start any of these products, type the

following UNIX/Linux command:

`encounter`

For information on using this command, see the [Getting Started](#) chapter.

**Table 1-1 EDI System Products**

Name	Abbreviation	Prod. Num.	Description
Encounter® Digital Implementation System L	EDS-L	EDS100	Provides block-level floorplanning, physical synthesis, clock tree synthesis, routing, optimization and design closure (~300000 instances per license), dual layout-abstract views for prototyping analog blocks in EDI System, OpenAccess interoperability including shared cds.lib, cross-domain data and shared look-and-feel with Virtuoso, hierarchical model creation, automatic floorplan synthesis, automatic macro placement, wire editing, multi-Vth optimization, clock gating for low power, early rail analysis using signoff power analysis engine, SMART routing, metal fill, verify DRC/LVS, ECOs, GigaOpt engine for post-route optimization, Turbo-QRC extraction for post-route RC extraction, IQRC incremental extraction available with QRC-XL, metal fill timing optimization, signoff timing and delay calculation, Turbo QRC extraction, Global Timing Debug with links to Conformal Constraints Designer (CCD), GDSII, Oasis, and OpenAccess support and interoperability.
Encounter® Digital Implementation System XL	EDS-XL	EDS200	Has all the features of EDS-L without the 300,000 instance limitation. In addition, this product provides multi-core support, netlist-to-netlist optimization and advanced netlist restructuring, hierarchical top-level assembly, automatic floorplan synthesis and ranking, optimization, end-to-end MMMC support, incremental extraction (requires QRC license), SI analysis and fixing using AAE, timing-aware ECO/remapping,

			signoff SI, Global Clock Debug, Global Power Debug, complete flip-chip support, and 45 degree routing (area/peripheral IO support).
NanoRoute ® Ultra SoC Routing Solution	NRU	FE150	Provides optimized routing and routing verification system with utmost in speed and capacity for signal integrity, timing, and interconnect optimization for manufacturability, SMART routing, metal fill, verify DRC/LVS, ECOs, concurrent litho hotspot prevention, native multi-thread, multi-CPU, and superthreaded routing, GDSII, Oasis, and OpenAccess support and interoperability.
Virtuoso ® Digital Implementation	VDI	3002	Has all the features of EDS-L and adds RTL Compiler functionality and netlist-to-netlist optimization for logic synthesis, block-level floorplanning, wire editing, clock tree synthesis, routing, optimization and design closure, hierarchical model creation, automatic floorplan synthesis, automatic macro placement, and wire editing, SMART routing, metal fill, verify DRC/LVS, ECOs, Multi-Vth optimization, clock gating for low power, early rail analysis using signoff power engine, signoff timing and delay calculation, global timing debug with links to Conformal Constraint Designer (CCD), GDSII, Oasis and OpenAccess support and interoperability. Limited to 50,000 instances in the netlist. However, multiple VDI licenses can be stacked upto 100000 instances.
Virtuoso ® Digital Implementation XL	VDI-XL	3003	<p>VDI-XL has all the functionalities of the VDI package with the same 50,000 instance capacity being stackable up to 100,000 instances with two licenses.</p> <p>In addition, VDI-XL provides the support for SI analysis and MMMC optimization to address process variation challenges.</p> <p>VDI-XL also supports the complete advanced</p>

		low power design methodology driven by unified power intent in CPF, including low power synthesis of RC Low Power.
		VDI-XL is ideal for digital logic design in the context of Cadence analog-driven mixed-signal and low power design methodology with seamless OpenAccess interoperability.

## First Encounter Hierarchical Prototyping Solution

In addition to the products listed in Table 1-1, the `encounter` command can be specified to start the products that belong to the First Encounter Hierarchical Prototyping Solution, which are listed in Table 1-2. To start any of these products, type the following UNIX/Linux command:

```
encounter
```

**Table 1-2 First Encounter Hierarchical Prototyping Solution**

Name	Abbreviation	Prod. Num.	Description
First Encounter® L	FE-L	FE80	An automatic silicon virtual prototyping and hierarchical partitioning solution with built-in power planning and floorplanning.
First Encounter® XL	FE-XL	FE100 GPS	Has all the capabilities of First Encounter L. In addition, this product provides MMMC support, implementation-quality placement and optimization, basic clock tree synthesis, and flipchip capability.

## EDI System 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 both EDI System and First Encounter Hierarchical Prototyping Solution product packages.

**Note:** The EDI System product options can be checked out only if you are using the EDI System products (EDI-L, EDI-XL, NRU, VDI). The product options are not allowed to be checked out for the old

FE products, such as FE-L and FE-XL. These old FE products can check out the Encounter GigaScale GXL option.

The EDI System includes the following product options:

**Table 1-3 EDI System Product Options**

Name	Abbreviation	Prod. Num.	Description
Encounter Low Power GXL Option	ENC-LP Opt.	EDS10	Adds advanced low-power functionality by automating multiple power domain and power-switch-aware floorplan synthesis, implementation, and routing, enabled by full common power format (CPF) support, provides end-to-end multi-supply voltage (MSV) support, power domain creation, power domain-aware automatic floorplan synthesis, power domain-aware routing, dynamic voltage frequency scaling support, power shut-off and power switch prototyping, state retention power gating support, always-on buffer support, hierarchical macro model support, dual-flop support (limited-access).
Encounter Mixed Signal GXL Option	ENC-MS Opt.	EDS20	<p>Adds mixed-signal functionality such as EDI System's mixed-signal routing (diff-pair, and so on). It allows access to non-digital custom Virtuoso data such as pcells, fig-groups, RODs, and custom design constraints from a common OpenAccess database. It also provides constraints-driven mixed-signal routing, VSR integration for mixed-signal routing, full digital timing analysis of AMS block without characterization for analog macros/ IP, digital and analog concurrent floorplanning.</p> <p><b>Note:</b> Virtuoso IC6.1 or later version is required for OpenAccess-based interoperability flow.</p>
Encounter Advanced Node GXL Option	ENC-AN Opt.	EDS30	Provides new 32 nm and 28 nm rules support, concurrent design for yield/design for manufacturing capability by preventing lithography hotspots and analyzing and optimizing them if they occur, support for on-chip/ off-chip variation mode and statistical

			<p>static timing analysis (SSTA), context-driven placement, structured datapath support, DFM/ DFY optimization for wires and vias, 1-D routing support, clock mesh and hybrid implementation, SOI support, interface to signoff litho and CMP simulation.</p> <p>For more information on 32 nm rules licensing, see Advanced Node License Required for 32/28/20 nm and Smaller Nodes.</p>
Encounter Clock Concurrent Opt GXL option	ENC-CCO Opt	EDS210	Provides Clock Concurrent Optimization, which is a single step process that optimizes both the clock tree and datapath to meet global timing constraints.
Encounter GigaScale GXL Option	ENC-GS Opt.	EDS70	Enables flex model abstraction, GigaFlex flow, FlexILM flow, and partition-in-partition capability.
Encounter Stacked Die GXL Option	ENC-SD Opt	EDS60	Adds advanced features such as multi-die rail analysis and multi-die die model creation.
Encounter T20 GXL Option	ENC-T20 Opt.	EDST20	Provides all features of TSMC 20 nm rules support, Double Patterning (DPT) support, FlexColor flexible colorization engine, placement, optimization and routing, in-design and analysis.
Encounter S20 GXL Option	ENC-S20 Opt.	EDSS20	Provides all features of ISDA 20 nm rules support, Double Patterning (DPT) support, FlexColor flexible colorization engine, placement, optimization and routing, and in-design and analysis.
EDI CPU Accelerator GXL Option	ENC-MCPU Opt.	EDS03	Provides multi-CPU acceleration. It allows three additional CPUs to be used with any applicable base license, for multi-CPU enabled steps.

**Note:** The EDS30 and EDS50 product options provide an interface to standalone LPA and CCP. However, to run these standalone applications you require LPA and CCP licenses.

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

## About EDI System Licenses

When you run a command to invoke a product or product option, a license is checked out. Each product and product option has a unique license string (also called a license key).

### 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. For more information on dynamic licenses, see "Checking Out Licenses for Product Options".

- **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 [EDI System Packaging and Licensing](#).

## Checking Out Licenses for Product Options

- The following command specifies the product options to be checked out when you invoke the software. The product options specified with the `-checkoutList` parameter are checked out immediately and the product options specified with the `-optionList` parameter are checked out dynamically.

```
encounter -checkoutList "option1 option2 ..." -optionList "option1 option2 ..." "
```

**Note:** If you do not want any product options to be checked out dynamically, use empty

quotes with the `-optionList` parameter, as follows:

```
encounter -checkoutList " option1 option2 ... " -optionList " "
```

- The following command can be used to check out product options after you have invoked the software. The product option specified with the `-checkoutList` parameter is checked out immediately and the product options specified with the `-optionList` parameter are checked out dynamically.

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

With the `-checkout` parameter, you can specify only one product option.

**Note:** If you do not want any product options to be checked out dynamically, use empty quotes with the `-optionList` parameter, as follows:

```
setLicenseCheck -checkout option -optionList " "
```

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

The following table shows the product options that each base product can check out dynamically. In the table:

- 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.

For example, a base license for the EDS-L product allows you to check out the following product options: ENC-LP Opt., ENC-MS Opt., ENC-AN Opt., EPS-XL, EPS-L, ETS-XL, and ETS-L. It does not allow you to check out NRU or CelIIC product options.

**Table 1-4 Dynamic Licensing Table**

	<b>Product Name</b>	<b>ENC-LP Opt</b>	<b>ENC-MS Opt</b>	<b>ENC-AN Opt</b>	<b>ENC-DFM Opt</b>	<b>ENC-SD Opt</b>	<b>ENC-GS Opt</b>	<b>ENC-T20 Opt</b>	<b>ENC-S20 Opt</b>	<b>ENC-CC0 Opt</b>	<b>EPS-XL</b>	<b>EPS-L</b>	<b>EPS-AA Opt</b>	<b>EPS-HA Opt</b>	<b>ETS-XL</b>	<b>ETS-L</b>	<b>ETS-AA Opt</b>	<b>NRU</b>	<b>ENC-NG</b>	<b>ETS-NG</b>	<b>EPS-NG</b>
<b>Base</b>	<b>Product</b>	<b>EDS10</b>	<b>EDS20</b>	<b>EDS30</b>	<b>EDS50</b>	<b>EDS60</b>	<b>EDS70</b>	<b>EDST20</b>	<b>EDSS20</b>	<b>EDS210</b>	<b>EPS200</b>	<b>EPS100</b>	<b>EPS201</b>	<b>EPS301</b>	<b>FE725</b>	<b>FE625</b>	<b>FE830</b>	<b>FE150</b>	<b>ENG100</b>	<b>ETSNG100</b>	<b>EPSNG100</b>
<b>EDS-L</b>	<b>EDS100</b>	✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓		✓			✓		
<b>EDS-XL</b>	<b>EDS200</b>	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓		
<b>FE-L</b>	<b>FE80</b>			✓				✓	✓	✓		✓	✓					✓	✓		
<b>FE-XL</b>	<b>FE100GPS</b>			✓				✓	✓	✓		✓	✓					✓	✓		
<b>FE-GXL</b>	<b>FE800</b>						✓				✓	✓	✓					✓	✓		
<b>ETS-L</b>	<b>FE625</b>					✓					✓	✓						✓	✓	✓	
<b>ETS-XL</b>	<b>FE125</b>						✓				✓	✓						✓	✓	✓	
<b>EPS-L</b>	<b>EPS100</b>																		✓		✓
<b>EPS-XL</b>	<b>EPS200</b>																		✓		✓
<b>VPS-XL</b>	<b>VPS200</b>																		✓		✓
<b>VDI</b>	<b>3002</b>	✓	✓	✓	✓	✓		✓	✓			✓	✓					✓		✓	
<b>VDI-XL</b>	<b>3003</b>	✓	✓	✓	✓	✓		✓	✓			✓	✓					✓		✓	
<b>NRU</b>	<b>FE150</b>	✓	✓	✓	✓	✓		✓	✓			✓	✓					✓		✓	

## Checking Out Multi-CPU Licenses

You must check out a base license before checking out a multi-CPU license. The following table shows the number of CPUs are enabled by each base license and the number of additional CPUs enabled by each multi-CPU license.

In the table:

- The first column lists the base product names in abbreviated format.
- The second column lists the base product number.
- 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.
- The third column shows the number of CPUs enabled by the base product in that row.
- 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.

For example, a base license for EDS-XL enables two CPUs. An additional license (a multi-CPU license) for EDS-L enables one more CPU; an additional license for EDS-XL enables three more CPUs, and an additional license for NRU enables three more CPUs.

**Table 1-5 Multi-CPU Licensing Table**

	Product Name	Base	ENC-CPU	EDS-L	EDS-XL	YDI	YDI-XL	ETS-L	ETS-XL	EPS-L	EPS-XL	VPS-XL	NRU	FE-L	FE-XL	FE-GXL
Base License	Product Number	Base Count	EDS 03	EDS 100	EDS 200	3002	3003	FE625	FE725	EPS100	EPS200	VPS200	FE150	FE80	FE100GPS	FE800
<b>EDS-L</b>	<b>EDS 100</b>	1	3	1	3									3		
<b>EDS-XL</b>	<b>EDS 200</b>	2	3	1	3									3		
<b>YDI</b>	<b>3002</b>	1	3													
<b>YDI-XL</b>	<b>3003</b>	1	3													
<b>ETS-L</b>	<b>FE625</b>	1						1	4							
<b>ETS-XL</b>	<b>FE725</b>	4						1	4							
<b>EPS-L</b>	<b>EPS100</b>	1		1	3					1	3					
<b>EPS-XL</b>	<b>EPS200</b>	2		1	3					1	3					
<b>VPS-XL</b>	<b>VPS200</b>	1														
<b>FE-L</b>	<b>FE80</b>	1		1	3									1	3	3
<b>FE-XL</b>	<b>FE100GPS</b>	2		1	3									1	3	3
<b>FE-GXL</b>	<b>FE800</b>	2		1	3									1	3	3
<b>NRU</b>	<b>FE150</b>	2	3	1	3								3			

## Additional Considerations

The following additional rules apply to multi-CPU licensing:

- The software uses the same license model for multi-threading, distributed processing, and Superthreading.
- EDS-L is limited to 300,000 logic instances for both the base license and the multi-CPU license.
- Product option licenses cannot be used as base licenses or multi-CPU licenses.
- You can stack multi-CPU licenses; that is, you can check out more than one multiple-CPU license for a base license.
- You can check out multi-CPU licenses for different products for the same base product.
- 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.

## Advanced Node License Required for 32/28/20 nm and Smaller Nodes

Newly added DRC rules for 32/28/20 nm and smaller process nodes require the Advanced Node license.

The following EDI system product options provide licenses for LEF property rules:

- **EDSS20** (Encounter S20 Option)
- **EDST20** (Encounter T20 Option)
- **EDS30** (Encounter Advanced Node GXL Option)

The LEF property keywords used for these rules that require the above licenses are highlighted below in **bold** text.

- For specific 20nm rules, you require EDSS20license option (and its prerequisite license EDS30) when a design is loaded:

- VIACLUSTER

```
PROPERTY LEF58_VIACLUSTER " VIACLUSTER . . . " ;
```

- ENCLOSURETOJOINT

```
PROPERTY LEF58_ENCLOSURETOJOINT " ENCLOSURETOJOINT . . . " ;
```

- For specific 20nm rules, you require EDST20license option when a design is loaded:

- ENCLOSURETABLE

```
PROPERTY LEF58_ENCLOSURETABLE " ENCLOSURETABLE . . . " ;
```

- For generic 20nm rules, either EDSS20 or EDST20 (and the prerequisite license EDS30) are required when the design is loaded:

- SAMEMASK

This enables double-patterning support, and is in a SPACING rule:

```
PROPERTY LEF58_SPACING " SPACING . . . SAMEMASK . . . " ;
```

- MINWIDTH (or WIDTH if no MINWIDTH given) is less than or equal to 0.04

Any routing layer that has a min-width rule that is less than or equal to 0.04.

- The 32/28nm rules require EDS30 license options to run NanoRoute:

- CUTCLASS

```
PROPERTY LEF58_CUTCLASS " CUTCLASS className . . . " ;
```

- ENCLOSUREWIDTH

```
PROPERTY LEF58_ENCLOSUREWIDTH " ENCLOSUREWIDTH . . . " ;
```

- OPPOSITEEOLSPACING

```
PROPERTY LEF58_OPPOSITEEOLSPACING " OPPOSITEEOLSPACING . . . " ;
```

- JOGTOJOGSPACING

```
PROPERTY LEF58_SPACINGTABLE " SPACINGTABLE JOGTOJOGSPACING . . . " ;
```

- MINWIDTH (or WIDTH if no MINWIDTH given) that is less than or equal to 0.05

Any routing layer that has a min-width rule that is less than or equal to 0.05.

**Note:** The 20nm rules are checked first, so they will automatically check-out the appropriate 20nm license. If no EDSS20 rules are found, then EDST20 license is automatically checked-out for the generic 20nm rules (SAMEMASK and min-width less than 0.04). If you prefer to use EDSS20 for generic 20nm rules, you are required to use `encounter -checkoutList encs20` command during startup to force the check-out of EDSS20 instead.

**Note:** Any 20nm option (ENC-T20 or ENC-S20) would need the additional prerequisite ENC-AN option. If ENC-AN is not available, ENC-T20 or ENC-S20 options are also not checked out.

## Getting Started

---

- [Product and Installation Information](#)
- [Setting the Run-Time Environment](#)
- [Configuring OpenAccess](#)
- [Launching the Console](#)
- [Completing Command Names](#)
- [Command-Line Editing](#)
- [Setting Preferences](#)
- [Starting the Software](#)
- [Interrupting the Software](#)
- [Using the Log File Viewer](#)
- [Accessing Documentation and Help](#)
- [Accessing EDI System Tutorials](#)

### Product and Installation Information

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

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

For information about EDI System licenses, see "About EDI System 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 Encounter 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/fe/man
```

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

Example:

```
setenv MANPATH  
${MANPATH}:/icd/flow/EDI/EDI113/latest.USR3.lnx86/lnx86/share/fe/man
```

### Supported and Compatible Platforms

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

## Specifying the 64-Bit or 32-Bit Version of EDI System Applications

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

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

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

For more information, see the README file.

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

- Set the CDS\_AUTO\_64BIT environment variable before starting the software. For more information, see Using the CDS\_AUTO\_64BIT Environment Variable.
- Use a command parameter when you start the software. For information, see [Using Generic Parameters to Specify 32- or 64-Bit Version](#).

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

By default, the EDI System software runs in the 64-bit mode. You can use the CDS\_AUTO\_64BIT environment variable to control which applications are run in the 64-bit mode and which ones are run in 32-bit. To run 64-bit versions of all or some applications, complete the following steps before starting the software:

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

For example,

- To run all applications in the 64-bit mode, type the following command:  

```
setenv CDS_AUTO_64BIT ALL
```
- To run just a few applications, such as NanoRoute® and Celtic®, in 64-bit mode and all other applications in the 32-bit mode, type one of the following commands:  

```
setenv CDS_AUTO_64BIT nanoroute:celtic
setenv CDS_AUTO_64BIT nanoroute,celtic
setenv CDS_AUTO_64BIT 'nanoroute;celtic'
```
- To run all applications except EDI System in the 64-bit mode, type the following command:  

```
setenv CDS_AUTO_64BIT exclude:encounter
```

This will set 32-bit as default for EDI System.
- To run all applications in the 32-bit mode, type the following command:  

```
setenv CDS_AUTO_64BIT none
```

This will select 32-bit as default for all applications.

## Specifying Temporary File Locations

In 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.

Example:

```
setenv FE_TMPDIR /project/tmpdir
```

```
encounter ...
```

## Configuring OpenAccess

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

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

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

**Important:** Cadence recommends that you use the OpenAccess kit that comes with the EDI System software for almost all uses.

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

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

To set the variable, type the following command:

```
setenv OA_HOME oa_install_dir
```

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

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

## Launching the Console

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

```
encounter>
```

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

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

## Completing Command Names

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

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

command name that completes or matches the text you typed (if the string is unique to one text command) or all the commands that match the text you typed.

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

```
setPlace
```

The software displays the following command:

```
setPlaceMode
```

If you type the following text and press the Tab key

```
setPl1
```

The software displays the following commands:

```
setPlaceMode      setPlanDesignMode
```

## Command-Line Editing

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

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

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

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

### Control (^) Characters

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

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

## Escape Sequences

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

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

## Setting Preferences

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

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

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

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

## Initialization Files

The EDI System software uses the following initialization files for setting preferences:

.encrc	Used for setting Tcl parameters or adding user-defined Tcl commands. If different versions of this file exist in the installation, home, or working directories, the file in the working directory takes precedence.  <b>Note:</b> Usage of this file is no longer recommended, but is allowed for backward compatibility. Use enc.tcl instead. This file is processed before the GUI is created, so it cannot be used to customize the GUI.
enc.tcl	Used for setting Tcl parameters, customizing the GUI, or adding user-defined Tcl commands or global variables. If different versions of this file exist in the installation, home, or working directories, the file in the working directory takes precedence.  <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 ( <i>encounter_installation_path/tools/fe/etc</i> ), home directory, or working directory.
enc.pref.tcl	Contains design preferences set using the Design, Display, Floorplan, and

	<p>Selection tabs in the Preferences form in the GUI (see <i>Set Preference</i> in the <a href="#">Options Menu</a> chapter of the <i>EDI System Menu Reference</i> ).</p> <p><b>Note:</b> By default, EDI System saves changes that you make to your preferences to the <code>enc.pref.tcl</code> file in the working directory.</p>
<code>.enc</code>	Contains design preferences set using the Windows tab in the Preferences form in the GUI (see <i>Set Preference</i> in the <a href="#">Options Menu</a> chapter of the <i>EDI System Menu Reference</i> ).
<code>cds.lib</code>	Contains library path information. The <code>cds.lib</code> entries are available from Tcl in <code>cds_lib vars</code> .

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

- **encounter**

Starts one of the following products:

- Encounter Digital Implementation System L
- Encounter Digital Implementation System XL
- First Encounter® L
- First Encounter XL
- First Encounter GXL
- NanoRoute Ultra
- Virtuoso® Digital Implementation

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

```
encounter
encounter
[-edsxl] [-edsl] [ -fegxl] [-fexl] [ -nru] [ -vdixl] [ -vdi] [ -fel]
[-N2]
[-checkoutList "lic1 lic2 ..."]
[-cmd file.cmd ]
[-exitOnError]
[-help]
[-init file.tcl ]
```

```

[-libDefFile file.defs ]
[-log file.log ]
[-nowin | -win]
[-nologv]
[-optionList "lic1 lic2 ..."]
[-multiCpuLicenseList "lic1 lic2 ..."]
[-overwrite]
[-version]
[-wait t ]

```

#### Parameters

<pre>[-edsx1] [-edsL] [ -fegx1] [-fex1] [ -nru] [-vdix] [-vdi] [-fel]</pre>
---

Checks out the base license for the specified product. If more than one license is given, EDI System searches for each license in the order given from left to right on the command line. If none of them is available, the software generates an error message and quits. If no license is given, the software tries each one in the order given above.

Specify one or more of the following:

- edsx1 : Encounter Digital Implementation System XL
- edsL : Encounter Digital Implementation System L
- fegx1 : First Encounter® GXL
- fex1 : First Encounter XL
- nru : NanoRoute® Ultra
- vdix1 : 3003 Virtuoso® Digital Implementation XL.
- vdi : 3002 Virtuoso® Digital Implementation.
- fel : First Encounter L

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

-N2	<p>This option only works with -vdi or -vdix1. By default, only one copy of vdi or vdix1 is checked out, and it allows a maximum of 50,000 instances. -N2 checks out two copies of vdi or vdix1 and allows up to 100,000 instances.</p>
-----	---

**-checkoutList "lic1 lic2 ..."**

	<p>Checks out licenses for the specified product options at startup and holds the licenses for the remainder of the session. The product options provide additional features to your base license.</p>
--	--

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

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

Specify one or more of the following parameters:

	<p>enclp: Encounter Low Power GXL option</p> <p>encms: Encounter Mixed Signal GXL option</p> <p>encan: Encounter Advanced Node GXL option</p> <p>encsd: Encounter Stacked Die GXL option</p> <p>encgs: Encounter GigaScale GXL option</p> <p>enct20: Encounter T20 GXL option</p> <p>encs20: Encounter S20 GXL option</p> <p>enccco: Encounter Clock Concurrent Opt GXL option</p> <p>epsxl: Encounter Power System XL</p> <p>eps1: Encounter Power System L</p> <p>etsxl: Encounter Timing System XL</p> <p>etsl: Encounter Timing System L</p> <p>nru: NanoRoute Ultra Routing Solution</p> <p>encng: Encounter Next Generation (This license is used to enable beta features.)</p> <p><b>Note:</b> For information on these parameters, see <a href="#">Product and Licensing Information</a>.</p>
<b>-cmd</b> <i>file.cmd</i>	Changes the default cmd file.
<b>-help</b>	Prints a brief description for each encounter parameter.
<b>-exitOnError</b>	Specifies that the EDI System software should exit if an error occurs while running a script.
<b>-init</b> <i>file.tcl</i>	
	Specifies the Tcl file to read in at the start of the session and starts the software in non-GUI mode. When the command finishes executing the Tcl file, the software switches to GUI mode.
<b>-libDefFile</b> <i>file.defs</i>	
	Specifies the path and file name of the library definition file for OpenAccess-based flows. If you do not specify this parameter, all OpenAccess-based operations refer to the <i>lib.defs</i> file in the current working directory by default.
<b>-log</b> <i>file.log</i>	Specifies a name for the log file. By default, the software saves log files in the run directory and increments them, for example, <i>encounter.log</i> , <i>encounter.log1</i> , <i>encounter.log2</i> , <i>encounter.log3</i> , and so on. <i>Default:</i> <i>encounter.log</i>
<b>-nologv</b>	Specifies that log file will not be verbose.
<b>-nowin</b>   <b>-win</b>	Specifies whether the software runs in a GUI environment. <i>Default:</i> <i>-win</i>

`-optionList " option1 option2 ... "`

	<p>Specifies an ordered list of product options for automatic or dynamic license checkout. Checkout is dynamic, which means that even though you specify options with this parameter, the licenses for the options are checked out only when they are needed. The licenses are held for the duration of the session.</p> <p>If you specify a product option that is not allowed with your base product, or an option without an available, the software does not check out that license and instead issues a warning message.</p> <p>If you need to specify more than one product option, begin and end the list with double quotation marks or braces.</p> <p><b>Note:</b> For information on these parameters, see <a href="#">Product and Licensing Information</a>.</p> <p>Specify one or more of the following parameters:</p> <ul style="list-style-type: none"> <li>enclp : Encounter Low Power GXL option</li> <li>encms : Encounter Mixed Signal GXL option</li> <li>encan: Encounter Advanced Node GXL option</li> <li>encsd: Encounter Stacked Die GXL option</li> <li>encgs : Encounter GigaScale GXL option</li> <li>enct20: Encounter T20 GXL option</li> <li>encs20: Encounter S20 GXL option</li> <li>enccco: Encounter Clock Concurrent Opt GXL option</li> <li>epsx1: Encounter Power System XL</li> <li>eps1: Encounter Power System L</li> <li>etsx1: Encounter Timing System XL</li> <li>ets1: Encounter Timing System L</li> <li>nru: NanoRoute Ultra Routing Solution</li> <li>encng: Encounter Next Generation (This license is used to enable beta features.)</li> </ul>
--	---

`-multiCpuLicenseList " lic1 lic2 ... "`

	<p>Specifies an ordered list for automatic multi-CPU license checkout. The allowed multi-CPU licenses are different for various base licenses as shown in the "Multi-CPU Licensing Table" in the <a href="#">Product and Licensing Information</a> chapter.</p>
<code>-overwrite</code>	Overwrites the existing log file.
<code>- version</code>	Displays the version of EDI System software installed on the host machine without checking out a license or starting the software.

<code>-wait t</code>	Specifies the amount of time the system waits for a license to become available. If the license is available in less than the specified wait time, the system checks out the next needed license without waiting. <i>Default:</i> 0 (no wait time) <i>Value range:</i> 0 to 10,000
----------------------	--

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

When you start the software, complete the following steps:

1. Specify one of the following parameters:

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

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

**Note:** If the `CDS_AUTO_64BIT` environment variable is not set and one of the following parameters is specified, the wrapper sets `CDS_AUTO_64BIT` to NONE:

- - 32
- - 32only
- - 3264

If the `CDS_AUTO_64BIT` environment variable is not set and one of the following parameters is specified, the wrapper sets `CDS_AUTO_64BIT` to ALL :

- - 64
- - 64only
- - 6432

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

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

-debug3264	Prints the environment, updated by the wrapper and the command launched.
-plat <i>platform</i>	Allows you to override the default platform selection when you launch the tool from the following directory: <i>install_root</i> /bin
-quiet3264	Suppresses warning, error, and info messages generated by the -32, -32only, -3264, -64, -64only, or -6432 parameters.
-v3264	Prints the wrapper's version string.

## Interrupting the Software

You can interrupt an EDI System session by using the **ctrl - c** key combination. For most commands, **ctrl - c** exits the session and causes the software to issue the following message:

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

- If you do not press **ctrl - c** again, the software proceeds as normal.
- If you do press **ctrl - c** again, the software stops and the session ends.

### Interrupt Behavior for Long-running Commands

The behavior of the software when you use **ctrl - c** differs for the following long-running commands:

#### NanoRoute Router

- `routeDesign`
- `globalDetailRoute`

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

#### Timing Optimization (optDesign)

- `optDesign`
- For information, see "Interrupting Timing Optimization" in the [Optimizing Timing](#) chapter.

#### Verification

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

For information, see "Interrupting Verification" in the [Identifying and Viewing Violations](#) chapter.

## Interrupting the Execution of Batch Files

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

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

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

## 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 EDI System prompt. You can then type any command required for debugging. Whenever you want to resume your script, just type `resume` at the EDI System prompt.

## Stopping the Software

Use one of the following methods to stop the software:

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

## Using the Log File Viewer

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

- [Integrated Log File Viewer](#)
- [Standalone 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 *Encounter Menu Reference* .
- Access to the documentation in the *EDI System 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 *EDI System 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.

## Standalone Log File Viewer

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

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

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

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

## Accessing Documentation and Help

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

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

### Launching Cadence Help From the Command Prompt

1. Change to the following directory:

```
installation_dir /tools/bin
```

2. Enter the following command:

```
./cdnshelp
```

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

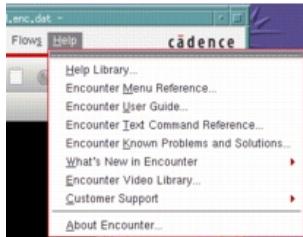
For more information see the [Cadence Help](#) manual.

## Accessing Documentation and Help From the Encounter GUI

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

- Select Help on the Main Encounter Menu
- Select Help on an Encounter Form

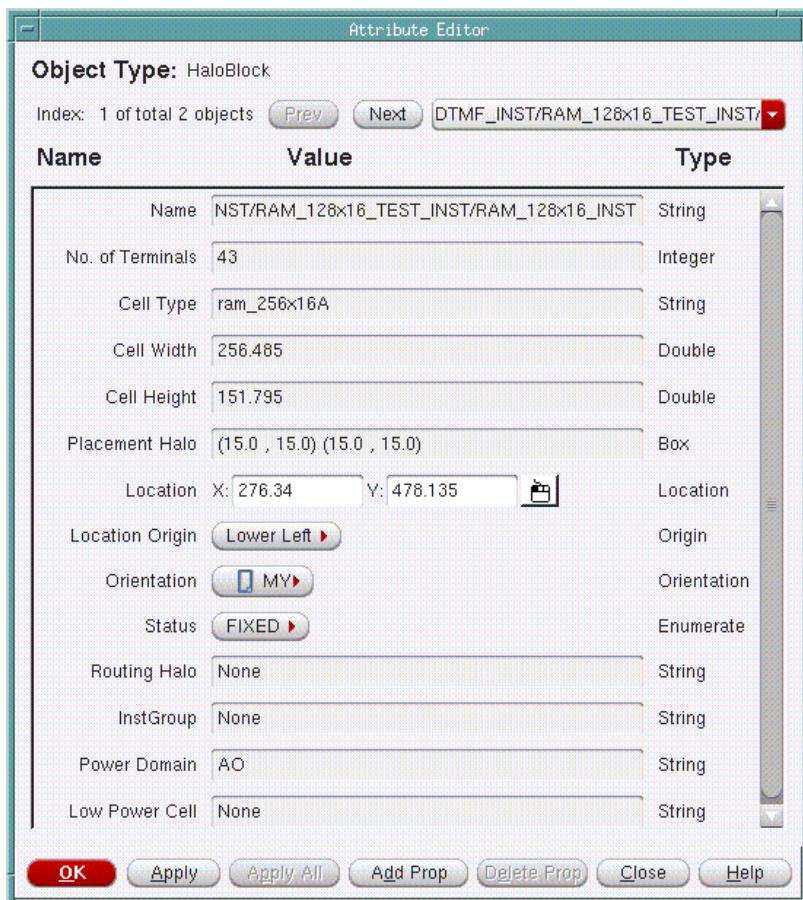
### Select Help on the Main Encounter Menu



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

### Select Help on an Encounter Form

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



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

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

### Using the help Command to 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 Encounter 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 Encounter command, type the following command in the software console:

```
man command_name
```

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

```
man getVersion
```

The software displays the following text:

```
Product Version 11.1 Cadence Design Systems, Inc.
```

```
getVersion(1) getVersion(1)
```

NAME

```
getVersion
```

SYNTAX

```
getVersion [-help]
```

DESCRIPTION

```
Returns the EDI System software version number.
```

Parameters

```
-help Prints a brief description of the command. For a detailed description of the command, use the man command:
```

```
man getVersion
Tue Oct 11 13:09:27 2011 getVersion(1)
(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 ENCSYC-3160 message ID, type the following command:

```
help ENCSYC-3160
```

The software displays the following text:

```
Ignoring the -keepEmptyModule setting in the configuration file. In the non-physical mode, the software keeps  
the empty modules and converts them into leaf cells. Remove the -keepEmptyModule setting from the  
configuration file.
```

#### 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 ENCSYC-3160 message ID, type the following command:

```
man ENCSYC-3160
```

The software displays the following text :

NAME

ENCSYC-3160

SYNOPSIS

```
Ignoring the -keepEmptyModule setting in the configuration file. In the non-physical mode, the software keeps  
the empty modules and converts them into leaf cells. Remove the -keepEmptyModule setting from the  
configuration file.
```

DESCRIPTION

```
{This warning is displayed when you use the -keepEmptyModule setting in the configuration file. The software  
does not honor this setting and keeps the empty modules (in non-physical mode) by default. In the non-  
physical mode, the empty modules are converted into leaf cells. To avoid this warning, remove the -  
keepEmptyModule setting from the configuration file.}
```

 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](#) or [viewLog](#) in the "General Commands" chapter of the *EDI System Text Command Reference*.

## Other Sources of Information

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

- Cadence Online Support  
Opens Cadence Online Support in your browser.
- Web Collaboration  
Opens SpaceCruiser in your browser.
- Education Services  
Opens the Education Services Web site in your browser.

## Accessing EDI System Tutorials

If you are new to EDI System, it might be a good idea to first go through a relevant tutorial to master the basics of EDI System. Copy and paste the following link in your browser to view available tutorials:

- `http://support.cadence.com/wps/mypoc/cos?`  
`uri=deeplinkmin:DocumentViewer;src=wp;q=ProductInformation/Digital_IC_Design/EDI_Tutorial.html`

These tutorials cover the basics of flat implementation (including virtual prototyping), hierarchical implementation, and flip chip designs.

---

# 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](#)
- [Migrating Obsolete Internal Menu APIs](#)

## Overview

EDI 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 *EDI System 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

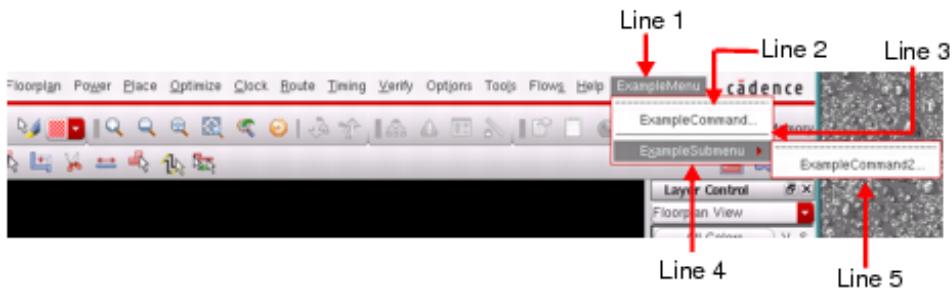
uiAdd expSep -type separator -in expMenu

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 EDI System *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]

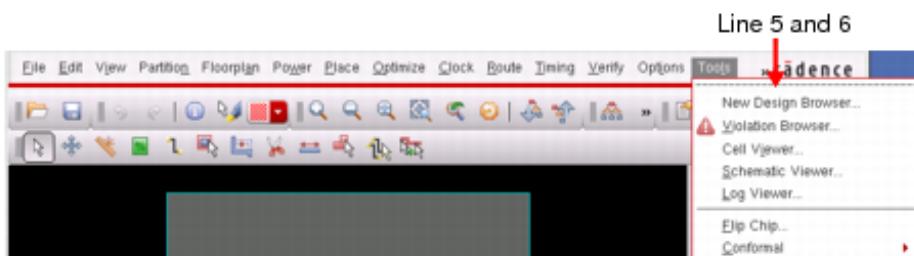
uiDelete $oldMenu

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

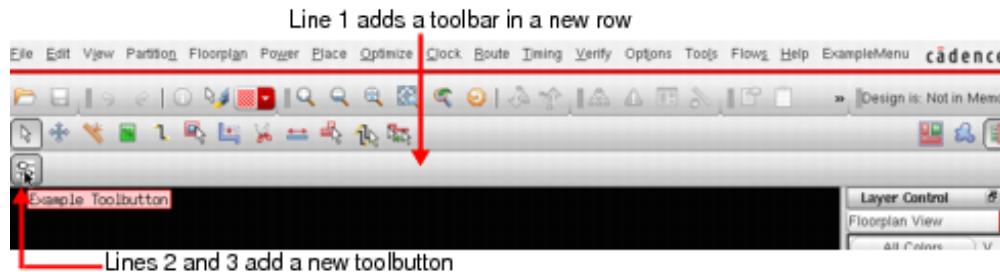
Using the [uiAdd](#) command, you can add a new toolbar and toolbuttons as shown in the following script:

```
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 Toolbar" -icon [file join $ICON_DIR example.xbm]
```

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 an `.xbm` file as its icon.



## 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

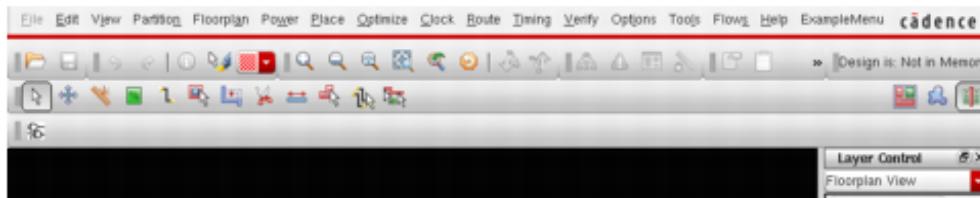
Using the `uiGet`, `uiFind` and `uiSet` commands in the GUI development kit, you can query and configure various interface elements, including menus, status bar, and the main window.

## 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 edi_statusbar [uiGet main -statusbar]
uiSet $edi_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

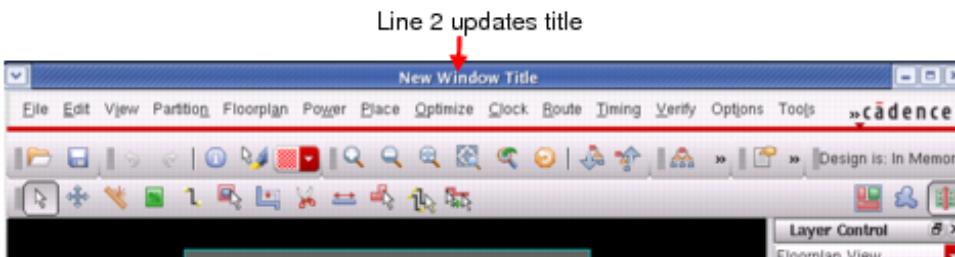
You can use the [uiSet](#) command to set the size of the main window as desired. For instance, you can set the main window size to 800x600 as follows:

```
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* .



## Migrating Obsolete Internal Menu APIs

The following tk-based internal menu APIs have been made obsolete in EDI System 9.1 release:

- `uiAddChildMenuSeparator`
- `uiAddMenuItem`
- `uiAddMenuSeparator`
- `uiAddChildMenu`

- uiAddChildMenuItem
- uiDeleteMenu
- uiAddMenu

These internal menu APIs can be migrated using the new public APIs. Let's look at an example:

The script below uses the old internal menu APIs to add a menu labeled Flow to the menu bar in the main window.

```
### tk-based menu APIs

uiAddMenu dac Flow

uiAddMenuItem dac CCD {ccd -d ./dofile &}

uiAddMenuItem dac NewSDC {read_sdc -reset leon.sdc}
```

To do the same using the new public APIs, you can use the following script:

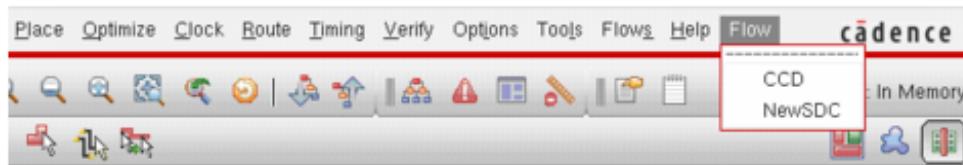
```
### new public APIs

uiAdd dac -type menu -label Flow -in main

uiAdd dac.ccd -type command -in dac -label CCD -command {ccd -d ./dofile &}

uiAdd dac.newSdc -type command -in dac -label NewSDC -command {read_sdc -reset
leon.sdc}
```

Line 1 of the script adds a menu labeled Flow. Lines 2 and 3 add menu items CCD and NewSDC to the Flow menu.



---

# 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 Encounter Digital Implementation System (EDI System) 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 *EDI System Text Command Reference* or the "Multiple CPU Processing" form in the [Options Menu](#).

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

**Table 4-1 EDI 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> <a href="#">addFiller</a>	<ul style="list-style-type: none"> <li>▪ Supported in default mode only (- modulePlan is true)</li> </ul> <p>For more information, see <a href="#">Running Placement in Multi-CPU Mode</a> in the "Placing the Design" chapter.</p>
Optimization	<a href="#">optDesign</a> {-preCTS   -postCTS   -postRoute}	For more information, see <a href="#">Distributed Timing Analysis for Hold Fixing</a> in the "Optimizing Timing" chapter.
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> <a href="#">globalDetailRoute</a> <a href="#">detailRoute</a> <a href="#">routeDesign</a> <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> <p>For more information, see <a href="#">Accelerating Routing with Multi-Threading and Superthreading</a> in the "Using the NanoRoute Router" chapter.</p>
TQRC, IQRC, and Standalone extraction	<a href="#">setExtractRCMode</a> <a href="#">extractRC</a>	For more information, see <a href="#">Distributed Processing</a> in the "RC Extraction" chapter.

Signal integrity analysis	<p><a href="#"><u>optDesign</u></a> - postRoute -si</p> <p><a href="#"><u>timeDesign</u></a> - si</p> <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="#"><u>Multi-CPU Processing Settings</u></a> in the "Analyzing and Repairing Crosstalk" chapter.</p>
Verify connectivity	<a href="#"><u>verifyConnectivity</u></a>	For more information, see <a href="#"><u>Verifying Connectivity</u></a> in the "Identifying and Viewing Violations" chapter.
Verify geometry	<a href="#"><u>verifyGeometry</u></a>	For more information, see <a href="#"><u>Verifying Geometry in Multi-Thread Mode</u></a> in the "Identifying and Viewing Violations" chapter.
Verify metal density	<a href="#"><u>verifyMetalDensity</u></a>	For more information, see <a href="#"><u>Verifying Metal Density in Multi-Thread Mode</u></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="#"><u>Calculating Delay in Multi-Thread Mode</u></a> in the "Calculating Delay" chapter.
Timing Budgeting	<a href="#"><u>deriveTimingBudget</u></a> <a href="#"><u>saveTimingBudget</u></a>	For more information, see <a href="#"><u>Support for Distributed Processing in Budgeting</u></a> in the "Timing Budgeting" 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, or load-sharing facility queue to use for distributed processing. If you request more machines than are available, most applications wait until all requested machines are available.

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

- [setMultiCpuUsage](#)

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

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

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

```
setDistributeHost -lsf -queue mem4G
setMultiCpuUsage -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 EDI System software in the Superthreading 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 slave (setDistributeHost -local) command, the memory information will include the memory consumed by master and slaves. Otherwise, the master and slave 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	Private	Virtual	Resident
---------------	---------	---------	----------

	Set(DRS)	Dirty(DRT)	Size(VIRT)	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
Slave peak:	748.8M	368.7M	1219.4M	456.4M

Slave peak reports peak value of each item from all slaves, therefore, it is possible that eight values come from eight different slaves.

## 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 { nru vdi eds1 edsxl fexl fegxl }
```

For information on the default check-out order, see [EDI 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]encounter 1> getMultiCpuUsage

Total CPU(s) Enabled: 2
Current License(s): 1 Encounter_Digital_Impl_Sys_XL
keepLicense: true
licenseList: nru 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 EDI System 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 [EDI 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 Encounter software, you must create physical libraries (cells and macros).

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

### Using LEF to Create Physical Libraries

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

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

### Creating OpenAccess Physical Libraries

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

### Unsupported LEF and DEF Syntax

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

#### Unsupported LEF 5.7 Syntax

The Encounter software parses but ignores the following LEF 5.7 syntax:

LEF Statement	Unsupported Syntax
Layer (Routing)	[DIAGWIDTH <i>diagWidth</i> ;] [DIAGSPACING <i>diagSpacing</i> ;] [DIAGMINEDGELENGTH <i>diagLength</i> ;] [SLOTWIREWIDTH <i>minWidth</i> ;]

	<pre>[SLOTWIRELENGTH <i>minLength</i> ;]  [SLOTWIDTH <i>minWidth</i> ;]  [SLOTLLENGTH <i>minLength</i> ;]  [MAXADJACENTSLOTSPACING <i>spacing</i> ;]  [MAXCOAXIALSLOTSPACING <i>spacing</i> ;]  [MAXEDGESLOTSPACING <i>spacing</i> ;]  [SPLITWIREWIDTH <i>minWidth</i> ;]  [HEIGHT <i>distance</i> ;]  [SHRINKAGE <i>distance</i> ;]  [CAPMULTIPLIER <i>value</i> ;]</pre>
Macro Pin	<pre>[TAPERRULE <i>ruleName</i> ;]  [NETEXPR "netExprPropName defaultNetName " ;]</pre>
Nondefault Rule	<pre>[DIAGWIDTH <i>diagwidth</i> ;]  [HARDSPACING ;]  [USEVIARULE <i>viaRuleName</i> ;]</pre>
Via Rule Generate	[DEFAULT]

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

LEF Statement	Unsupported Syntax
Layer (Routing)	DIRECTION {DIAG45   DIAG135} ;

## Unsupported DEF 5.7 Syntax

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

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

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

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

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

## Generating the I/O Assignment File

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

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

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

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

This section discusses the following topics:

- Creating an I/O Assignment File
- 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

io_order = clockwise

total_edge = 10

space = 1.06

)

(row_margin

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

    (io_row ring_number = 1 margin = 0.0000)

    (io_row ring_number = 2 margin = 94.0000)

    (io_row ring_number = 3 margin = 181.0000)

  )

( ...

)

(iopad

  (top | north | left | west | right | east | bottom | south | row

    (locals

      row_name = name_of_row

      space = 1.2

      ring_number = 1

      io_order = counterclockwise

    )


```

```
(inst
  name = ioinst_1
  skip = 2.2
  space = 1.2
  offset = 10.2
  indent = 10.2
  orientation = r180
  place_status = fixed
)
(keepclear begin = 10.0 end = 12.0)

(inst
  name = ioinst_2
  orientation = r180
  skip = 2.2
  cell = mymaster
)
(endspace gap = 1.2)
)

# corner io cell

  (topright | northeast | topleft | northwest | bottomright | southeast | bottomleft
  | southwest | row
  (locals
```

```
row_name = name_of_row
ring_number = 1
)
(inst
name = corner_1
orientation = r180
cell = corner_master
)
(inst
name = ioinst_2
x = 100.0
y = 200.0
orientation = r180
place_status = fixed
)
)
(iopin
(top | north | edge num = 0
(locals
space = 1.2
io_order = counterclockwise
```

```
)  
  
(pin name = "din [0]"  
  
    layer = 3  
  
    width = 0.28  
  
    depth = 0.28  
  
    skip = 2.2  
  
    space = 1.2  
  
    offset = 10.2  
  
    place_status = fixed  
  
)  
  
)  
  
(left | west | edge #  
  
)  
  
(right | east | edge #  
  
)  
  
(bottom | south | edge #  
  
)  
  
(up  
  
    (pin name="address[2]"  
  
        x=158.0700  
  
        y=180.6400  
  
        layer=4
```

```
width=0.2800
depth=0.2800
)
(pin name="rcc_clk"
x=159.3400
y=180.5600
layer=6
width=0.6000
depth=0.8000
)
)
(bump
(array
  name = array_1
  cell = bc1
  llx = 100
  lly = 100
  urx = 100
  ury = 100
  x = 100.0
  y = 200.0
```

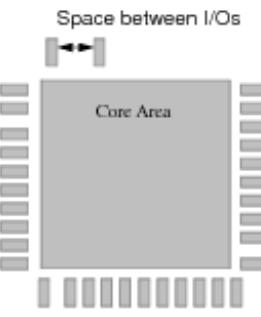
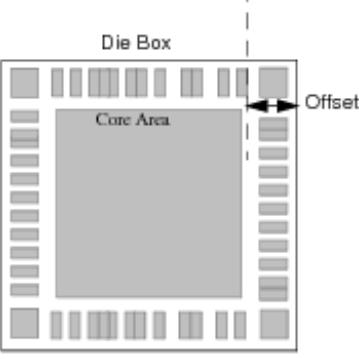
```
xpitch = 20
ypitch = 20
xspace = 10
yspace = 10
row = 6
column = 6
out_rings = 3
style = stagger | full
center_column = 4
center_row = 4
center_style = stagger | full
)
(bump
name = lvdsov33v12_ca_sref_i42_r1c1
cell = bc1
x = 100.0
y = 200.0
signal = vdd12
type = power | ground
assignment = fixed
array = array_1
```

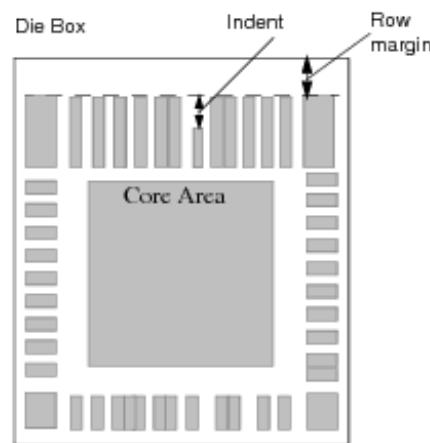
```
orientation = r180
)
)
```

The following entries are included in the template:

globals		
	<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.</p> <p>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>iopad locals</code>		
	<code>space</code>	Specifies the global I/O pin spacing, in meters.
	<code>space</code>	Specifies the local I/O pad spacing, in meters.

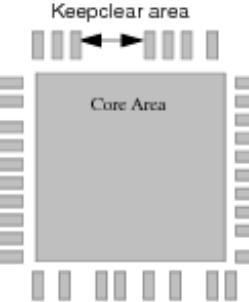
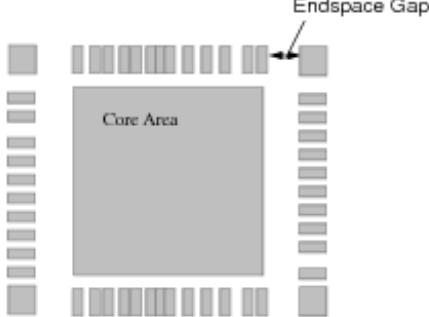
		<b>Note:</b> This space setting is honored by the first cell on one edge, when xy or offset is not specified.
	<i>ring_number</i>	Specifies the ring number in which the I/O pad is placed.
	<i>row_name</i>	Specifies the I/O <i>row</i> name.
<b>iopad instance</b>		
	<i>name</i>	Specifies the name of the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

		
	<i>offset</i>	<p>Specifies the offset in meters. The offset of a pad is the offset from the die boundary, based on the order direction.</p> <p>The value that you specify here is valid only for this cell.</p> 
	<b>Note:</b> For one I/O pad, you can specify only one of the following parameters:	<ul style="list-style-type: none"> <li>▪ skip</li> <li>▪ space</li> <li>▪ offset</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>
	<i>indent</i>	<p>Specifies the offset, in meters, from the row margin.</p>

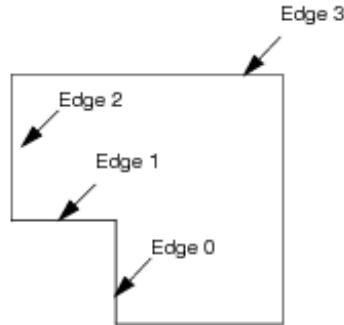


However, for designs with single I/O ring, row margin is 0. Hence, indent is the offset of the I/O pad from the die boundary.

	<i>orientation</i>	Specifies the orientation of the I/O.
	<i>place_status</i>	<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>
	<i>keepclear</i>	<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 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>

		 A diagram showing a central gray rectangle labeled "Core Area". This core is surrounded by a double-line border consisting of two vertical columns of small gray rectangles. The space between these two columns is labeled "Keepclear area".
	<code>cell</code>	Specifies the physical I/O cell.
	<code>endspace gap</code>	Specifies the space, in meters, between the corner pad and the last I/O pad for the specified side of the design.   A diagram showing a central gray rectangle labeled "Core Area". This core is surrounded by a double-line border of I/O pads. On the right side, there is a gap between the last I/O pad and the corner pad, which is labeled "Endspace Gap".
<code>iopin locals</code>		
	<code>side</code>	<p>Specifies the side of the I/O pin. This can be:</p> <ul style="list-style-type: none"> <li>▪ top   north</li> <li>▪ left   west</li> <li>▪ right   east</li> <li>▪ bottom   south</li> </ul>
	<code>edge num = 0</code>	Specifies the edge number of the I/O pin, with <code>edge num = 0</code> starting from the left side of the lowest y coordinate and the left most corner, in

the clockwise direction.



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

## iopin

	<i>pin name</i>	Specifies the name of a pin. Specify I/Os as pins for block designs.
	<i>layer</i>	Specifies the metal layer on which the pin must be placed.
	<i>width</i>	Specifies the width of the pin in 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 meters.
	<i>up</i>	Specifies the details of internal I/O pins.
	<i>x, y</i>	<p>Specifies the absolute x,y location of the internal I/O pin.</p> <p><b>Note:</b> The I/O file supports specifying xy location for internal I/O pins only.</p>

#	Specifies the incremented I/O pin edge number.
---	--

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

Row Margin	
side	Specifies the side of the I/O row margin. This can be: <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 [setItoRowMargin](#) 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.

## Specifying Area I/O Information

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

- **Bump**

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

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

*Bump: bump\_name bumpCell\_name x y signal type assignment array orientation*

For example,

*Bump: A3 BUMP 300 100 vdd12 power fixed array\_1 r180*

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

*Bump: bump\_name bumpCell 1lx 1ly urx ury x y xpitch ypitch xspace yspace row column out\_rings style center\_column center\_row center\_style*

For example,

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

- **IOInst**

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

*IOInst: inst\_name [x y [orientation] ] [place\_status]*

For example,

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

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

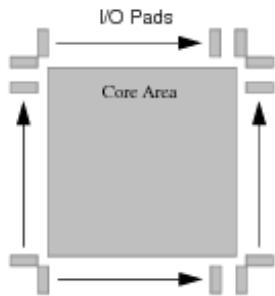
To create a rule-based I/O assignment file:

1. Create an I/O assignment file with I/O pads in the proper sequence. This file can include VDD and VSS filler pads.
2. Import the design.
3. After reviewing the I/O pads, choose *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

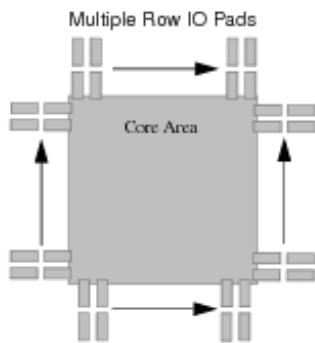
)
(inst

    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:

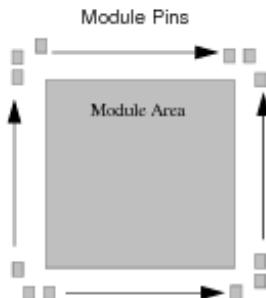


```
version = 3  
  
io_order = clockwise  
  
total_edge = 4  
  
space = 1.06  
  
  
iopad  
(topright  
(locals  
    ring_number = 1  
  
)  
  
(instname = IOPADS_INST/pad1 W
```

```
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 = R0
    cell = ZMGACS101N
)
( instname = CNR@0002
    orientation = R0
    cell = ZCGLSNEIS1A
)
)
)
```

## Performing Area I/O Placement

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

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

```
SITE IO CLASS PAD ; SIZE 210 BY 100.8 ; END IO
MACRO INBUF
CLASS PAD AREAIO ;
FOREIGN INBUF 0.00 0.00 ;
ORIGIN 0 0 ;
SIZE 210 BY 100.8 ;
SYMMETRY X Y R90 ;
```

```
SITE 10 ;  
  
PIN PAD  
  
DIRECTION INPUT ;  
  
USE SIGNAL ;  
  
PORT ;  
  
LAYER M6 ;  
  
RECT 95.0 40.0 115.0 60.0 ;  
  
END  
  
END PAD
```

**Note:** The CLASS PAD AREAIO saves bump status defined in the DEF file only if the bump status is FIXED or COVER. See Defining BUMP CELL Placement Status.

#### 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 I01 ;
```

```
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*.

#### **Defining BUMP CELL in LEF**

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

```
MACRO BUMPCELL
  CLASS COVER BUMP ;
  ORIGIN 0 0 ;
  SIZE 80.0 BY 80.0 ;
  SYMMETRY X Y ;
PIN PAD
  DIRECTION INPUT ;
  USE SIGNAL ;
PORT
```

```
LAYER M6 ;  
  
RECT 0.0 0.0 80.0 80.0 ;  
  
#POLYGON 23.0 0.057.0 0.0 80.0 2  
  
END  
  
END PAD  
  
END BUMPCELL
```

### Defining BUMP CELL Placement Status

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

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

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

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

### Importing LEF Files

To import the LEF files, use the following procedure:

1. Select *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 Encounter 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 Encounter software reads timing library format files (.tlf) or Synopsys Technology Library format files (.lib). You do not need to translate timing library files before reading them into the software.

## Encrypting Libraries

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

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

### Parameters

<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, PrimeTime, or Physical Compiler. These commands eliminate any variable substitution confusion, making them easier for the user and the software to read.

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

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

Or, you can use the following command:

```
write_sdc
```

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

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

## Preparing Capacitance Tables

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

For more information on preparing capacitance tables, see chapter [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 Encounter at various stages of the design process, you can check for missing or inconsistent library and design data.

To perform these checks, use the following command:

[`checkDesign`](#)

You can check for the following data:

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

Cadence recommends that you check libraries and data as follows:

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

## Preparing Data in the Timing Closure Design Flow

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

## 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](#)
- [The New Design Import Use Model in EDI System 11](#)
- [Verifying Data before Importing a Design](#)
- [Preparing the Design Netlist](#)
- [The init\\_design Import Flow](#)
  - [Legacy Configuration File Data Flow](#)
  - [init\\_design Simple Data Flow](#)
  - [Configuration File to MMMC Object Mapping Example](#)
  - [Supported init\\_design Invocation Methods](#)
- [Importing Designs Saved in Previous Versions](#)
  - [restoreDesign of a 10.x or Earlier MMMC database](#)
  - [loadConfig of a 10.x or Earlier MMMC Configuration](#)
  - [restoreDesign of 10.x or Earlier min/max Database](#)
  - [loadConfig of 10.x or Earlier min/max Configuration](#)
  - [MMMC Objects Created During Design Import](#)
  - [Timing and SI Libraries](#)
  - [Extraction Data](#)
  - [Timing Constraints](#)
  - [Delay Corners](#)
  - [Multi-Supply/Multi-Voltage Domains](#)
  - [Analysis Views](#)
  - [Reporting the MMMC Configuration](#)
- [Adapting Command Scripts for MMMC Compatibility](#)
  - [Summary of Command Changes in EDI System 11](#)
  - [MMMC Command Mapping Matrix](#)
- [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 EDI System and Virtuoso Chip Editor 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 EDI System 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 EDI System Session](#)
- [SKILL to TCL Mapping](#)

## Overview

The Encounter® Digital Implementation System (EDI System) 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 EDI System 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 EDI System environment.

## The New Design Import Use Model in EDI System 11

In the EDI System 11 release, significant changes have been made to the design import use model or the way design data, such as netlist files, library data, constraints, and so on, are specified and organized within the system. The new import model has been streamlined, eliminating redundant syntax and providing a cleaner, better documented system. The most notable changes in the new system include:

- The configuration (.conf) file and the related `loadConfig` and `commitConfig` commands have been retired. The old variables of the config file have been replaced by a much smaller number of better named and documented globals, and a new [`init\_design`](#) command.
- The software no longer has separate ways of configuring two-corner and multi-mode/multi-corner (MMMC) analyses. All analyses are configured the same way using the MMMC style of configuration, and the configurations are used directly for initialization.

To learn more about the changes in the design import process from previous releases to EDI System 11, see the following sections:

- [The init\\_design Import Flow](#)

This section introduces the basics of the `init_design` flow, and how it compares to the old configuration file based flow. It provides a detailed mapping of 10.x configuration items to EDI System 11. MMMC is a key part of the `init_design` model. Therefore, this section also introduces a simple example of how information in the config file maps to MMMC.

- [Importing Designs Saved in Previous Versions](#)

This section describes how to import designs from 10.x to EDI System 11. Generally, designs that are already MMMC in 10.x should load or restore straightforwardly in EDI System 11. These subsections provide more detail:

- [restoreDesign of a 10.x or Earlier MMMC database](#)

- [loadConfig of a 10.x or Earlier MMMC Configuration](#)

10.x min/max designs need to be moved to an MMMC configuration for EDI System 11.

The following subsections outline the necessary steps for these types of designs:

- [restoreDesign of 10.x or Earlier min/max Database](#)

- [loadConfig of 10.x or Earlier min/max Configuration](#)

- [MMMC Objects Created During Design Import](#)

- [Adapting Command Scripts for MMMC Compatibility](#)

This section provides information for adapting existing scripts to be MMMC compatible. It also highlights the design import-related 10.x features that have officially reached end-of-life (EOL) status with the EDI System 11 release:

- [Summary of Command Changes in EDI System 11](#)

- [MMMC Command Mapping Matrix](#)

## Verifying Data before Importing a Design

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

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

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

getCheckMode

## Preparing the Design Netlist

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

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

### [uniquifyNetlist](#)

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

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

## The `init_design` Import Flow

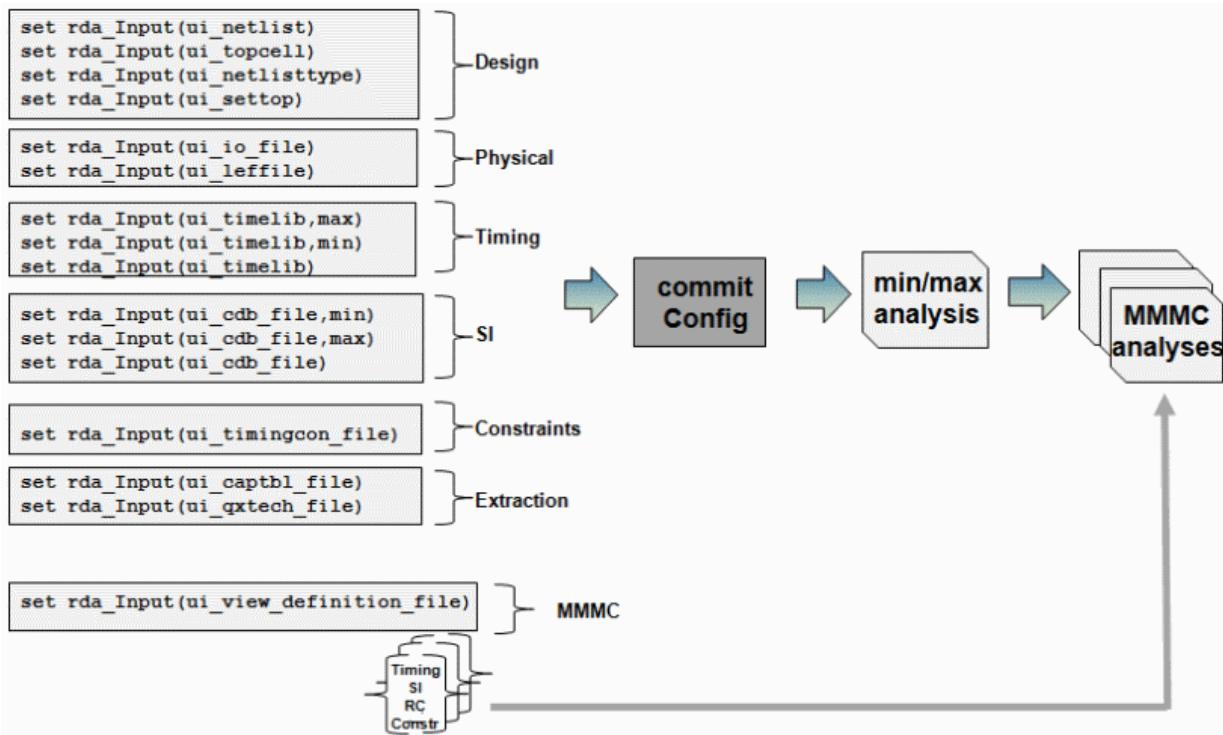
This section briefly covers the legacy configuration file-based data flow and then explains how the new `init_design`-based data flow introduced in EDI System 11 relates to it. All designs, whether they originate as physical-only, 10.x style min/max or MMMC, are saved in EDI System 11 using the new `init_design` style structure.

This section has the following subsections:

- [Legacy Configuration File Data Flow](#)
- [init\\_design Simple Data Flow](#)
- [Configuration File to MMMC Object Mapping Example](#)
- [Supported init\\_design Invocation Methods](#)

### Legacy Configuration File Data Flow

The configuration file used in EDI System 10.x was essentially a single global array, `rda_Input(...)`, which was populated by a myriad of different data ranging from required design initialization data, such as netlist and library information, default value customization settings, and persistence settings from various commands. The `loadConfig` command was used to parse the config file, and the `commitConfig` command was used to apply or invoke it. From a design import perspective, a simplified view of the data flow looks as follows:



### 10.x commitConfig Two-Corner, Single Mode Data Flow

As shown in the diagram, the configuration file-based flow allowed you to initially configure a two-corner, single-mode based flow. `commitConfig` initialized the design into min/max mode. Then, additional (sometimes redundant) timing, SI, extraction, and constraint data was processed from the MMMC view definition file to put the system into MMMC mode, if desired.

### init\_design Simple Data Flow

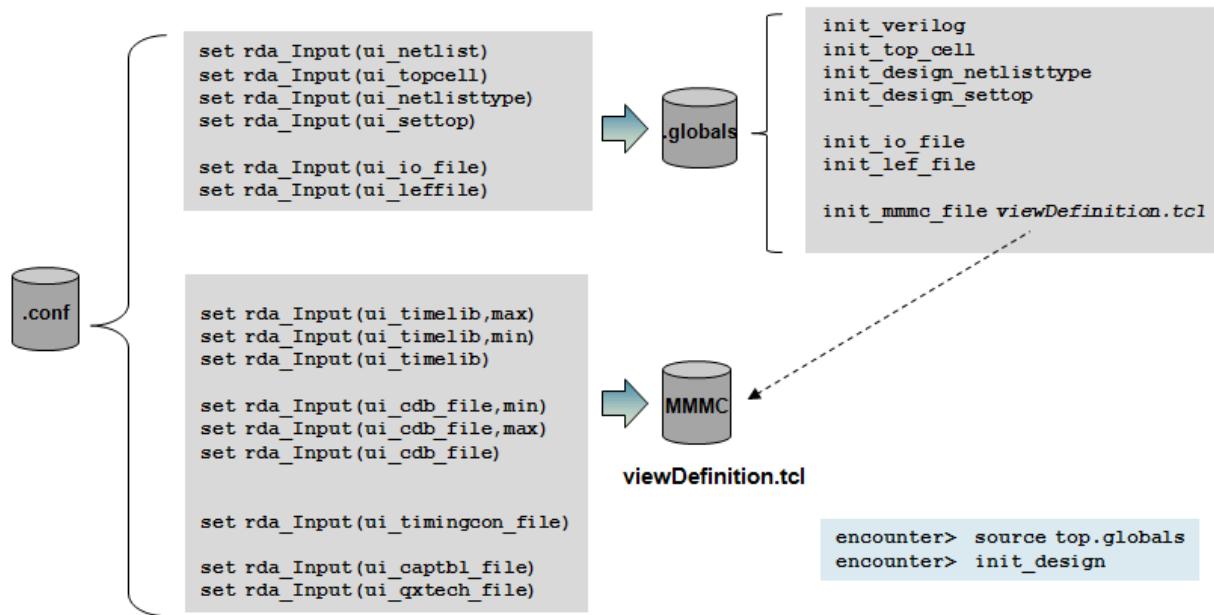
The re-architecture of the design import phase, based on `init_design`, has several key differentiating characteristics from the legacy configuration file-based flow:

- The [`saveDesign`](#) process does not save a separate configuration file (`.conf`) any longer.
- The `rda_Input` global array is no longer used as the catch-all for storing all data. Storage has been selectively broken up into sets of global variables. Variables that store data explicitly required for the initialization process are prefixed with `init_`.
- All of the new global variables that are replacing the `rda_Input` array entries 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

- The EDI System 11 `init_design` style configuration cannot be restored by previous releases of the software directly.

The diagram below shows how data based on a configuration file methodology in previous releases of the software maps to a globals file and MMMC view definition configuration in the EDI System 11 release.



As shown in the diagram, new `init_*` style variables have been introduced for the equivalent design-level and physical data. The [`init\_mmmc\_file`](#) variable has replaced `rda_Input(ui_view_definition_file)` as the pointer to the file containing the MMMC configuration. In addition, the [`init\_cpf\_file`](#) has been added to provide 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. Off-loading all the timing, SI, and extraction requirements to the MMMC configuration has considerably simplified what used to be in the configuration file.

The following table illustrates the mapping of 10.x configuration file `rda_Input` array variables that have been migrated to EDI System 11 as Tcl global variables used by `init_design`.

<b>10.x rda_Input Array Variables</b>	<b>EDI System 11 Variables</b>
<b>Used by</b> <code>loadConfig</code>	<b>Used by</b> <code>init_design</code>
<code>rda_Input(ui_netlist)</code>	<a href="#"><u>init_verilog</u></a>
<code>rda_Input(ui_topcell)</code>	<a href="#"><u>init_top_cell</u></a>
<code>rda_Input(ui_settop)</code>	<a href="#"><u>init_design_settop</u></a>
<code>rda_Input(ui_netlisttype)</code>	<a href="#"><u>init_design_netlisttype</u></a>
<code>rda_Input(ui_leffile)</code>	<a href="#"><u>init_lef_file</u></a>
<code>rda_Input(ui_view_definition_file)</code>	<a href="#"><u>init_mmmc_file</u></a>
N/A	<a href="#"><u>init_cpf_file</u></a>
<code>rda_Input(ui_io_file)</code>	<a href="#"><u>init_io_file</u></a>
<code>rda_Input(ui_import_mode)</code>	<a href="#"><u>init_import_mode</u></a>
<code>rda_Input(ui_assign_buffer)</code>	<a href="#"><u>init_assign_buffer</u></a>
<code>rda_Input(ui_pwrnet)</code>	<a href="#"><u>init_pwr_net</u></a>
<code>rda_Input(ui_gndnet)</code>	<a href="#"><u>init_gnd_net</u></a>
<code>rda_Input(ui_oa_designLib)</code>	<a href="#"><u>init_oa_design_lib</u></a>
<code>rda_Input(ui_oa_designCell)</code>	<a href="#"><u>init_oa_design_cell</u></a>
<code>rda_Input(ui_oa_designView)</code>	<a href="#"><u>init_oa_design_view</u></a>
<code>rda_Input(ui_oa_reflib)</code>	<a href="#"><u>init_oa_ref_lib</u></a>
<code>rda_Input(ui_oa_abstractname)</code>	<a href="#"><u>init_abstract_view</u></a>
<code>rda_Input(ui_oa_layoutname)</code>	<a href="#"><u>init_layout_view</u></a>

N/A	<a href="#">init oa search lib</a>
N/A	<a href="#">init oa special rule</a>

For the complete mapping of rda\_Input() variables to EDI System 11 equivalents, see "[Old Configuration File Variables](#)" chapter in the EDI System Text Command Reference.

Several possible init\_design scenarios are discussed in a later section of this chapter. The next subsection covers how the remaining part of the old configuration file, dealing mostly with library information, has been translated into the equivalent two-corner, single-mode MMMC configuration.

## Configuration File to MMMC Object Mapping Example

Much of the library information in the config file is already naturally grouped into max and min corners, typically reflecting a best-case/worst-case style EDI configuration. It is natural just to map the two corners into two equivalent corners using MMMC syntax. This section illustrates a simple mapping of the information in the .conf file to an MMMC configuration file. The actual mapping used in the 10.x to 11.1 transition is covered the [MMMC Objects Created During Design Import](#) section.

You can name your library sets anything you like, including 'max' or 'min'. However, it is generally recommend to include some reference to the object type. You do not have to specify all of the information at once. Timing library data is the only piece critical to the initialization process. The library set binds similar data together under a common name.

The table below depicts a simple mapping of library file setting from config file to MMMC.

<b>10.x</b>	set rda_Input(ui_timelib,max) "/icd/libs/syn/stdcell/slow/slow.lib"
<b>Configuration</b>	
<b>File</b>	
<b>Syntax</b>	set rda_Input(ui_timelib,min) "/icd/libs/syn/stdcell/fast/fast.lib"
	set rda_Input(ui_cdb_file,min) "/icd/libs/si/stdcell.cdb"
	set rda_Input(ui_cdb_file,max) "/icd/libs/si/stdcell.cdb"

	<pre>set rda_Input(ui_aocvlib)           "/icd/libs/aocv/test.aocv"</pre>
<b>Equivalent</b>	
<b>EDI System 11</b>	<pre>create_library_set -name my_max_library_set -timing [list /icd/libs/syn/stdcell/slow/slow.lib] -si [list /icd/libs/si/stdcell.cdb] -aocv [list /icd/libs/aocv/test.aocv]</pre>
<b>MMMC</b>	
<b>Syntax</b>	<pre>create_library_set -name my_min_library_set -timing [list /icd/libs/syn/stdcell/fast/fast.lib] -si [list /icd/libs/si/stdcell.cdb] -aocv [list /icd/libs/aocv/test.aocv]</pre>

Extraction data in the configuration file, including cap tables, QX tech files, and various scale factors, have been mapped one-for-one to similar constructs in the MMCRC corner object. The old config file could define up to three different RC corners-typical, best, and worst. For each RC corner present in the configuration file, you create one unique MMCRC corner object. The naming of the MMCRC corner objects does not matter; you choose names that are easy for you to reference later.

The table below depicts a simple mapping of extraction settings from the config file to MMCRC.

<b>10.x Configuration File Syntax</b>	<pre>set rda_Input(ui_captbl_file)           /icd/libs/tech/6mlv- typ.capTbl</pre>
	<pre>set rda_Input(ui_qxtech_file)          /icd/libs/tech/6mlv-typ.qx</pre>
	<pre>set rda_Input(ui_preRoute_cap)         {1}</pre>
	<pre>set rda_Input(ui_postRoute_cap)        {1}</pre>
	<pre>set rda_Input(ui_postRoute_xcap)       {1}</pre>
	<pre>set rda_Input(ui_preRoute_res)         {1}</pre>
	<pre>set rda_Input(ui_postRoute_res)        {1}</pre>
	<pre>set rda_Input(ui_preRoute_clkcap)      {0.0}</pre>
	<pre>set rda_Input(ui_preRoute_clkres)      {0.0}</pre>

	<pre>set rda_Input(ui_postRoute_clkcap) {0.0}  set rda_Input(ui_postRoute_clkres) {0.0}</pre>
<b>Equivalent</b> <b>EDI System 11</b> <b>MMMC</b> <b>Syntax</b>	<pre>create_rc_corner -name my_rc_corner_worst -cap_table          /icd/libs/tech/6mlv-typ.capTbl -qx_tech_file       /icd/libs/tech/6mlv-typ.qx -preRoute_res        1 -postRoute_res       1 -preRoute_cap        1 -postRoute_cap       1 -postRoute_xcap      1 -preRoute_clkres     0 -preRoute_clkcap     0 -postRoute_clkres    0 -postRoute_clkcap    0</pre>

All constraint file (SDC) information is mapped into MMC constraint mode objects. The standard EDI flow is single-mode (with a variant for running in ILM mode) so only one mapping needs to be done.

The table below depicts a simple mapping of constraint settings from the config file to MMC.

<b>10.x</b> <b>Configuration</b> <b>File</b> <b>Syntax</b>	<pre>set rda_Input(ui_timingcon_file) ./constraints/design.sdc</pre>
<b>Equivalent</b> <b>EDI System 11</b> <b>MMMC</b> <b>Syntax</b>	<pre>set rda_Input(ui_timingcon_file,full) "foo.enc.dat/top.ilm_data/design3.sdc"</pre>
<b>Equivalent</b> <b>EDI System 11</b> <b>MMMC</b> <b>Syntax</b>	<pre>create_constraint_mode -name my_constraint_mode -sdc_files           [list ./constraints/design.sdc] -ilm_sdc_files [list foo.enc.dat/top.ilm_data/design3.sdc]</pre>

The examples so far simply relocate the data from the configuration file into new MMC object syntax. In the remainder of the MMC configuration, we mix and match the library sets, RC corners, and constraints mode objects that we have already configured to create higher-level delay corners

and analysis views. However, not much of this additional assembly is required to mimic the behavior of the configuration file.,

We require two top-level corners:

```
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
```

For an MMMC analysis environment that is consistent with the previous min/max environment, you need to create one view for Setup that analyzes the constraints at the max corner and one view for Hold that analyzes the same constraints at the min corner. The constraint\_mode and delay\_corner object names are previously defined above.

```
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
```

Finally, it is the [set\\_analysis\\_view](#) command that selects which views are made active for Setup analysis and which views are made active for Hold. When this command is executed, this necessary library information is actually loaded into the system to prepare for the required analyses.

```
set_analysis_view -setup [list my_analysis_view_setup]
                  -hold [list my_analysis_view_hold]
```

Whether the flow is running in BcWc or OnChipVariation mode, the set\_analysis\_view settings do not need to change. With a proper MMMC configuration, the timing libraries will be automatically handled properly in both analyses modes.

A small amount of additional configuration is involved with the MMMC syntax, but once you complete the basic template, configuring additional types of analyses is very straightforward. For example, if you want to configure sign-off Setup and Hold analyses at both the corners created in the above example, you only need to do the following:

```
set_analysis_view -setup [list my_analysis_view_setup
                           my_analysis_view_hold]
-hold [list my_analysis_view_hold
        my_analysis_view_setup]
```

For more information about MMMC configurations, you can refer to the [Configuring the Setup for Multi-Mode Multi-Corner Analysis](#) section.

## Supported init\_design Invocation Methods

You have seen how to get all the data required to bring up an EDI 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. This would be a one-for-one replacement for the 10.x

loadConfig use model:

<b>10.x loadConfig Use Model</b>	<b>Corresponding init_design Use Model in EDI System 11</b>
loadConfig test.conf 0	source test.globals
commitConfig	init_design
or	
loadConfig test.conf	

**Note:** Here, it is assumed that `test.conf` and `test.globals` are 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 Saved in Previous Versions

As you have seen in the [The init\\_design Import Flow](#) section:

- With the exception of physical-only mode, all designs must have an MMMC style configuration to initialize into EDI System 11.
- All designs saved in EDI System 11 will be MMMC and will use the new `init_design` style (.conf-less) on disk format.

- There is a straightforward mapping between most of the old `rda_Input` array config variables and the new `init_*` globals and the MMMC design objects

This section outlines the current capabilities and options for importing 10.x or earlier designs into EDI System 11. There are four primary import scenarios:

- [restoreDesign of a 10.x or Earlier MMMC database](#)
- [loadConfig of a 10.x or Earlier MMMC Configuration](#)
- [restoreDesign of 10.x or Earlier min/max Database](#)
- [loadConfig of 10.x or Earlier min/max Configuration](#)

## restoreDesign of a 10.x or Earlier MMMC database

Designs that were saved by the `saveDesign` command in EDI System version 10.x are generally expected to be restored cleanly with `restoreDesign` into version 11.x of the software. Since the design was already MMMC, users scripts are also expected to continue to run smoothly. There are a few known areas where problems may occur:

- If your `viewDefinition.tc1` file references any of the 10.x default MMMC design objects (names like `default_*`) that are created automatically when the `.conf` file is read, then the `restoreDesign` operation will fail. In 11.x, the default MMMC objects are no longer created and are not available to be used. There are warnings in 10.x on many of these objects to that effect. If your MMMC configuration is relying on this default MMMC objects being created for you, you will need to manually add definitions for these to your `viewDefinition.tc1` file. You should also rename these objects and references to them.

In the following example, the user may have created a custom MMMC delay corner but referenced one of the default library sets, `default_libs_max`, which was derived previously from the configuration file. When the software tries to build the delay corner in EDI System 11, the default library set can no longer be found. An error message like the following is given in such a condition to indicate the initial failure. In this case, the situation has also resulted in the software being unable to initialize properly into MMMC mode. As a result, a second error message is also reported:

```
**ERROR: (TCLCMD-994): Can not find 'library set' object of name  
'default_libset_max'
```

\*\*ERROR : The software requires that designs with timing library information be properly initialized into a multi-mode/multi-corner (MMMC) environment. An error has occurred that has prevented proper initialization. You should review your design import configuration to make sure all file references are valid and MMMC configuration is correct and complete.

- When running a successful design import case of this type, you should expect to see a warning like the following. This warning is merely a reminder that on the next saveDesign, the on-disk representation of the design will change to the new format:

\*\*WARN: (ENCSYT-40500): The design you are restoring is a configuration file based multi-mode/multi-corner (MMMC) database. In this release of the software, configuration file support is being discontinued. When you saveDesign in this release, the design will be saved in the new init\_design based format. You should refer to the 'Importing Designs saved in previous versions' section in the 'Importing and Exporting Designs' chapter in the EDI11 user guide for more detailed information.

## loadConfig of a 10.x or Earlier MMMC Configuration

loadConfig is the second entry point to initializing a 10.x MMMC design in the 11.1 release. An MMMC configuration is defined as:

- The .conf has an rda\_Input(ui\_view\_definition\_file) entry pointing to a valid, error-free MMMC configuration capable of bringing up a Setup and Hold view with a complete, valid set of timing libraries

The loadConfig MMMC flow faces the same potential problems as the restoreDesign MMMC-based flow:

- It will fail if the MMMC specification references 10.x default MMMC object names.
- It may potentially give different results if the MMMC library specification is incomplete. All MMMC library\_sets are expected to provide a library cell representation for each cell used in the design.

The loadConfig plus MMMC flow will issue different warning messages to indicate that the loadConfig command itself is on an end-of-life track, and a flow such as this needs to be moved to init\_design:

\*\*WARN: (ENCSYT-40502): Support for the loadConfig command and the configuration file based design import methodology is being discontinued. This flow will continue to work in this release for designs that are already using a multi-mode/multi-corner (MMMC) configuration or are pure physical-only designs. For compatibility with future releases, you should transition your design to use the init\_design based methodology for design import. You should refer to the 'Importing Designs saved in previous versions' section in the 'Importing and Exporting Designs' chapter in the EDI11 user guide for more detailed information.

## **restoreDesign of 10.x or Earlier min/max Database**

When restoreDesign in EDI System 11 attempts to bring up a design database that was not originally MMMC, it will try to map the underlying data in the .conf and .mode files and synthesize an equivalent MMMC configuration on-the-fly. If the restore is successful, the system will return initialized in MMMC mode. You should expect to see the following warning message regardless of the outcome of the import:

\*\*WARN: (ENCSYT-40501): The design you are restoring is not based on a multi-mode/multi-corner (MMMC) configuration. In this release of the software, an MMMC configuration is required for most design flows. The software will support your current database in this release by automatically update the design to an MMMC configuration for you. When the design has been restored, you can then saveDesign and it will be saved in the new init\_design based format. Once the design has been updated, you may only utilize commands that are consistent with the MMMC environment. You should refer to the 'Importing Designs saved in previous versions' section in the 'Importing and Exporting Designs' chapter in the EDI11 user guide for more detailed information.

Additional information on the MMMC configuration created during the 10.x to 11.x import is covered in the [MMMC Objects Created During Design Import](#) section.

There are some known support limitations based primarily on support for older formats that has not been brought forward to MMMC. The limitations include:

- Import of designs using STAMP timing models is not supported due to lack of MMMC support.
- Import of designs using native SignalStorm IPDB database is not supported due to lack of MMMC support.
- Designs using wild-carded library file syntax in the .conf file are supported but the resulting MMMC specification will have the expanded, flat file description as wildcards are not supported

in MMMC syntax.

- The `setOpCond -forceLibrary` option is not supported during import.

Unlike the previous scenarios where the 10.x saved design was originally MMMC, scripting associated with these original min/max designs is likely not to be MMMC-aware, and will likely need to be updated. The transition to MMMC described in this section is strictly limited to importing of the database and will not provide any emulation for subsequent min/max style commands. Subsequent sections in this document will provide guidance for adapting min/max command usage to related commands in the MMMC environment.

## loadConfig of 10.x or Earlier min/max Configuration

EDI 11.x does not provide a direct import path for design flows which use `loadConfig` for min/max designs. By itself, `loadConfig` does not fully configure the system sufficiently for most applications. The recommended resolution for most users is to perform a `loadConfig` in 10.x, along with whatever other commands are used in conjunction to fully initialize the system. When this is accomplished, you should issue a `saveDesign` in 10.x. You can subsequently use `restoreDesign` in 11.x to restore the design and update it automatically to MMMC.

Attempting to load a min/max configuration in EDI System 11 will result in the following error condition:

\*\*ERROR: (ENCSYT-40503): Support for the `loadConfig` command and the configuration file based design import methodology for designs which are not configured as multi-mode/multi-corner (MMMC) has been discontinued in this release.

## MMMC Objects Created During Design Import

This section describes the MMMC objects that are created when a 10.x min/max design is restored by the 11.1 `restoreDesign` command and transitioned to an MMMC design representation. As illustrated in the [Configuration File to MMMC Object Mapping Example](#) section which showed a simple example of mapping from a configuration file to MMMC objects, the overall concept is quite straight forward. In this section, we will discuss the details of what is actually created from the configuration file during design import, and how that configuration is further refined through subsequent steps of the restore sequence.

The MMMC objects created during design import are:

- [Timing and SI Libraries](#)
- [Extraction Data](#)

- [Timing Constraints](#)
- [Delay Corners](#)
- [Multi-Supply/Multi-Voltage Domains](#)
- [Analysis Views](#)

## Timing and SI Libraries

Liberty (.lib) timing library, Celtic SI (.cdb), and AOCV (.aocv) derating library information are all combined under the MMMC library set object via the `create_library_set` command. Max library set information (`rda_Input(*,max)`) is grouped under the `library_set default_libset_max`, and min library set information (`rda_Input(*,min)`) is grouped as `default_libset_min`. Any common library information, `rda_Input(ui_timelib)`, `rda_Input(ui_cdb_file)`, is appended to both the min and max library lists. The AOCV derating library information is by default common and is therefore always included in both library set objects.

### 10.x Min/Max to EDI 11 MMMC Library Mappings

<b>10.x Configuration File</b>	<b>11.x MMMC Object Mapping</b>	<b>Notes</b>
<code>rda_Input(ui_timelib,max)</code>	<code>create_library_set</code>	No import is done for non-standard library corners such as <code>rda_Input(ui_timelib,typ)</code> or other such variations.
<code>rda_Input(ui_timelib)</code>	<code>-name default_libset_max</code>	
<code>rda_Input(ui_cdb_file,max)</code>	<code>-timing ...</code>	
<code>rda_Input(ui_cdb_file)</code>	<code>-si ...</code>	
<code>rda_Input(ui_aocvlib)</code>	<code>-aocv ...</code>	
		MMMC library_set does not support library search paths. Any search paths will be flattened, and the enumerated library list will be used in the new MMMC specification
<code>rda_Input(ui_timelib,min)</code>	<code>create_library_set</code>	
<code>rda_Input(ui_timelib)</code>	<code>-name default_libset_min</code>	
<code>rda_Input(ui_cdb_file,min)</code>	<code>-timing ...</code>	STAMP and IPDB support are not provided in MMMC.
<code>rda_Input(ui_cdb_file)</code>	<code>-si ...</code>	

rda_Input(ui_aocvlib)	-aocv ...
-----------------------	-----------

## Extraction Data

The MMMC RC corner object contains the extraction capacitance table, RC scale factor, QX tech file, and extraction temperature data that are represented in the configuration file. The configuration file can define up to three RC corners, although in the minimum/maximum environment only two RC corners can be active at a time. For the transition to MMMC, the software uses up to three RC corners to store the RC data in the config as necessary, and the `defineRcCorner` command is emulated to decide which RC corners are active at any given time.

The MMMC RC corners that get created are named as follows, depending on the number of corners in the configuration file:

- 1 default\_rc\_corner\_worst
- 2 default\_rc\_corner\_best
- 3 default\_rc\_corner\_typical

The name of the RC corner objects is significant. When you import SPEF files in MMMC mode with the `spefIn` command, you will need to specify the `spefIn -rc_corner` option and include the name of the RC corner. Similarly, when exporting SPEF files using the `rcout` command, you will need to specify the `-rc_corner` option and include the RC corner name.

## 10.x Min/Max To EDI 11 MMMC Extraction Setting Mappings

10.x Configuration File	11.x MMMC Object Mapping	Notes
<code>rda_Input(ui_captbl_file)</code>	<code>create_rc_corner</code>	
<code>rda_Input(ui_qxtech_file)</code>	<code>-name</code> <b><code>default_rc_corner_worst</code></b>	The .conf entry for the captbl may include up to three cap tables along with temperature values.
<code>rda_Input(ui_preRoute_cap )</code>	<code>-cap_table ...</code>	
<code>rda_Input(ui_preRoute_res \</code>	<code>-T temperatureValue</code>	Each cap table will result in an explicit MMMC RC corner object. The RC scale factors are copied to each of the RC

		corners that is created.
rda_Input(ui_preRoute_clkcap)	-qx_tech_file ...  -preRoute_cap ...	
rda_Input(ui_preRoute_clkres)	-preRoute_res ...	
rda_Input(ui_postRoute_cap)	-preRoute_clkcap ..  -preRoute_clkres ...	
rda_Input(ui_postRoute_res)	-postRoute_cap ...  -postRoute_res ...	
rda_Input(ui_postRoute_clkcap)	-postRoute_clkcap ...	
rda_Input(ui_postRoute_clkres)	-postRoute_clkres ...	

## Timing Constraints

The EDI flow in non-MMMC mode supports only a single set of design timing constraints (SDC files). Different versions of these constraints are supported depending on whether the system is in ILM mode or not. Dual mode is not supported by the current import flow. However, after the standard transition to MMC has been achieved, extending the configuration to any mix of multi-mode or multi-corner analysis is a very straightforward exercise.

The following table shows the mapping between the two configuration file entries for constraints and the MMC constraint mode object:

### 10.x Min/Max To EDI 11 MMC Constraint Setting Mappings

10.x Configuration File	11.x MMC Object Mapping
rda_Input(ui_timingcon_file)	create_constraint_mode
rda_Input(ui_timingcon_file,full)	-name <b>default_constraint_mode</b>  -sdc_files ...

```
-ilm_sdc_files ...
```

In MMMC mode, you can select which constraint modes should be active while specifying interactive constraints at the command line. The term active implies that the mode is associated with an MMMC analysis view that is currently enabled.

You select the active modes by using the following command:

```
set_interactive_constraint_modes {list_of_constraint_modes}
```

This puts the software into interactive constraint entry mode for the specified MMMC constraint modes. Any timing constraints that you specify after this command take effect immediately on all active analysis views that are associated with the specified constraint modes.

By default, no constraint modes are normally considered interactive but as part of the import, the default\_constraint\_mode is made immediately interactive for convenience. After you perform a save/restore sequence in EDI System 11, you must turn on interactive constraint mode if you need to enter additional constraints interactively.

For more information on using and controlling interactive constraint mode, refer to the following commands in the "Timing Analysis" chapter of the *EDI System Text Command Reference Guide*:

- [all\\_constraint\\_modes](#)
- [set\\_interactive\\_constraint\\_modes](#)
- [get\\_interactive\\_constraint\\_modes](#)

Some of the most common examples are for setting all active Setup, all active Hold, or all active Setup and Hold modes available for interactive commands.

- To make all active Setup modes interactive:

```
set_interactive_constraint_modes [all_constraint_modes -active_setup]
```

- To make all active Hold modes interactive:

```
set_interactive_constraint_modes [all_constraint_modes -active_hold]
```

- To make all active Setup and Hold modes interactive:

```
set_interactive_constraint_modes [all_constraint_modes -active]
```

## Delay Corners

EDI configuration files are most often setup for best-case / worst-case (BCWC) style analysis with

Setup analysis done at WC and Hold at BC in a typical two-corner scenario. The RC extraction side can pull in three corners of extraction data, but only two can be used at any given time by the system.

The configuration file is sometimes configured in a single-corner, early/late configuration for signoff style, on-chip variation (OCV) analysis using early and late libraries. The [setAnalysisMode](#) - analysisType option is used to select the type of analysis done by the system-single, bcwc, or onChipVariation.

During the migration process (from older versions to EDI System 11), three MMMC delay corners are created-one to handle each of the BC, WC, and OCV cases:

```

create_delay_corner
-name default_delay_corner_max
-library_set      default_libset_max

create_delay_corner
-name           default_delay_corner_min
-library_set      default_libset_min

create_delay_corner
-name           default_delay_corner_ocv
-early_library_set default_libset_min
-late_library_set   default_libset_max

```

Normally, you would never use the same library sets for both BCWC and OCV styles of analysis. In this case, however, as it is known beforehand which type of analysis the libraries are intended for so the system is prepared to handle both. Ultimately, only one will be configured to be used. BCWC style analysis is always done between MMMC delay corners, whereas OCV analysis needs to be always configured within a single MMMC delay corner using early and late options.

By incorporating the library set into the delay corner, you take care of the device portion of the delay effects. To account for interconnect effects, you now need to include an RC corner. The RC corner that gets attached by default to the MMMC delay corners during import depends on the number of RC corners that are defined.

## 10.x Min/Max to 11.1 MMMC RC Corner Assignment

# of RCs In Config	default_delay_corner_min	default_delay_corner_max
1	-rc_corner <i>default_rc_corner_worst</i>	-rc_corner <i>default_rc_corner_worst</i>

2	<code>-rc_corner <i>default_rc_corner_best</i></code>	<code>-rc_corner <i>default_rc_corner_worst</i></code>
3	<code>-rc_corner <i>default_rc_corner_best</i></code>	<code>-rc_corner <i>default_rc_corner_worst</i></code>

The delay corner is further configured as the restoreDesign process continues and any operating condition settings in the .mode file are processed. The setOpCond command (similar to the SDC set\_operating\_condition constraint) sets a named operating condition for the design. The condition reflects a specific process, voltage, and temperature (PVT) operating point. The setOpCond command can either define BC and WC operating points for bcWc style analysis, or it can define early and late sub-corner operating points for OCV style analysis. When performing the import, the setOpCond command is used to update all three delay corners according to the following table:

## 10.x Min/Max to 11.1 MMMC setOpCond Command

<code>setOpCond</code>	<code>default_delay_corner_min</code>	<code>default_delay_corner_max</code>	<code>default_delay_corner_ocv</code>
<code>name</code>	<code>-opcond name</code>	<code>-opcond name</code>	<code>-late_opcond name</code> <code>-early_opcond name</code>
<code>-max maxName</code>		<code>-opcond maxName</code>	<code>-late_opcond maxName</code>
<code>-min minName</code>	<code>-opcond minName</code>		<code>-early_opcond minName</code>
<code>-max maxName</code> <code>-min minName</code>	<code>-opcond minName</code>	<code>-opcond maxName</code>	<code>-late_opcond maxName</code> <code>-early_opcond minName</code>

The mappings for the library options to the setOpCond follow similarly to those of the operating condition names shown above.

In MMMC mode, the [set\\_timing\\_derate](#) command now requires a -delay\_corner option instead of -min or -max so you need to be familiar with the names of the delay corners that you create.

The delay corner is configured to supply a uniform delay calculation description for the entire design. For many designs, the chip may be partitioned into different power domains, which can be run at different voltages from other domains on the chip. For these domains, different operating conditions

may need to be specified. In addition, these domains may need to have a specific set of libraries characterized for the voltages assigned to those power domain regions. To support this capability, power domain subsets can be added to each MMMC delay corner, and each of these subsets can have explicit library set and operating condition settings provided. For more information, see section [Multi-Supply/Multi-Voltage Domains](#).

## Multi-Supply/Multi-Voltage Domains

Most low power design techniques involve setting up different power domains on the chip, which operates at various voltage levels. Supporting analysis of designs with power domains requires the ability to identify (bind) some instances with a certain set of libraries and voltage, and other with a different set.

- In EDI System, both in 10.x min/max and 11.x MMMC, the definition of power domains and the identification of the membership of power domains is handled by a CPF specification.
- CPF is usually the origination of the binding information of library files to power domains
- In a min/max flow, the operating condition PVT used for timing analysis is set by the `setOpCond -powerDomain` option and not by the CPF.

In the 10.x min/max style flow, the power domain information in the CPF is implicitly applied on the min and max delay corners defined by the configuration file. As part of the transition to MMMC, this same power domain information is added explicitly to the default delay corners by successive calls to the `update_delay_corner` command.

The EDI System 11 restoreDesign based MMMC import flow adds the power domain information from the CPF database to the MMMC configuration so that you get consistent results with the 10.x release of the software. The CPF format itself is capable of supporting a rich set of MMMC syntax natively.

**Note:** Since you will be working in an MMMC environment in EDI System 11, Cadence recommends that you upgrade your CPF file to a native CPF with internal MMMC based definitions. This is the preferred low-power/MSV methodology in EDI system. The 10.x to EDI System 11 import flow will not update the CPF syntax directly for you.

The following example illustrates a simplified 10.x min/max CPF file showing only the timing related information associated with the default power mode. (Only this part is relevant for design import.)

```
create_power_domain -name pd_default -default
create_power_domain -name pd_volt1V -instances {BLK1V}
create_power_domain -name pd_volt3V -instances {BLK3V}
```

```
create_power_domain -name pd_volt5V -instances {BLK5V}

define_library_set -name libs-1volt
    -libraries {libs/delayvolt_1V.lib libs/slow.lib}

define_library_set -name libs-2volt
    -libraries {libs/delayvolt_2V.lib libs/slow.lib}

define_library_set -name libs-3volt
    -libraries {libs/delayvolt_3V.lib libs/slow.lib}

define_library_set -name libs-5volt
    -libraries {libs/delayvolt_5V.lib libs/slow.lib}

update_nominal_condition -name nom1V -library_set libs-1volt

update_nominal_condition -name nom3V -library_set libs-3volt

update_nominal_condition -name nom5V -library_set libs-5volt

update_nominal_condition -name nom2V -library_set libs-2volt

create_power_mode -name pmStandard
    -default
    -domain_conditions {pd_volt1V@nom1V pd_volt3V@nom3V
        pd_volt5V@nom5V pd_default@nom2V}
```

Here is the EDI System 11 MMMC mapping of the power domain data. It shows the resulting set of MMMC object commands from committing the CPF and the subsequent translation to MMMC.

```
create_library_set -name libs-1volt
    -timing [list libs/delayvolt_1V.lib libs/slow.lib]

create_library_set -name libs-2volt
    -timing [list libs/delayvolt_2V.lib libs/slow.lib]
```

```
create_library_set -name libs-3volt
    -timing [list libs/delayvolt_3V.lib libs/slow.lib]
```

```
create_library_set -name libs-5volt
    -timing [list libs/delayvolt_5V.lib libs/slow.lib]
```

```
create_delay_corner -name default_delay_corner_max
    -library_set      libs-2volt
    -rc_corner        default_rc_corner_worst
```

```
update_delay_corner -name default_delay_corner_max
    -power_domain    pd_volt1V
    -library_set     libs-1volt
```

```
update_delay_corner -name default_delay_corner_max
    -power_domain    pd_volt3V
    -library_set     libs-3volt
```

```
update_delay_corner -name default_delay_corner_max
    -power_domain    pd_volt5V
    -library_set     libs-5volt
```

```
update_delay_corner -name default_delay_corner_max
    -power_domain    pd_default
    -library_set     libs-2volt
```

## Analysis Views

The MMMC analysis view object is used to associate a specific set of constraints (SDCs) with a delay calculation corner. Normally, an EDI configuration file is set up with one mode and two corners (BC and WC). The design import process creates the following views to create an MMMC environment consistent with this:

```
create_analysis_view
    -name          default_analysis_view_setup
    -delay_corner default_delay_corner_max
    -constraint_mode default_constraint_mode
```

```
create_analysis_view
  -name          default_analysis_view_hold
  -delay_corner  default_delay_corner_min
  -constraint_mode default_constraint_mode
```

In min/max mode, Setup analysis is always associated with the WC corner and Hold analysis with the BC corner. In MMMC, which type of analysis is done at which corner is not predefined. You need to specify this explicitly with the `set_analysis_view` command:

```
set_analysis_view
  -setup        default_analysis_view_setup
  -hold         default_analysis_view_hold
```

Once the `set_analysis_view` command has been issued, the system is ready to perform analyses for you. Many commands in the system now take a `-view` option to enable you to explicitly specify the data in a specific view. For example:

```
write_sdf -view default_analysis_view_setup ...
read_sdf -view default_analysis_view_setup ...
```

To report the current set of active setup and hold views, use the following commands:

```
all_setup_analysis_views
all_hold_analysis_views
```

## Reporting the MMMC Configuration

You can use the `report_analysis_views` command to output a formatted report of the current MMMC configuration. Here is an sample report:

```
+ Analysis View: default_analysis_view_setup
|
+ Delay Calc Corner: default_delay_corner_max
|
| + library_set: libs-2volt
```

```
| | |
| | + timing: libs/delayvolt_2V.lib libs/slow.lib
| |
| + rc_corner: default_rc_corner_worst
| |
| + pd_volt1V Power Domain
| |
| | +
| | + library_set: libs-1volt
| |
| | +
| | + timing: libs/delayvolt_1V.lib libs/slow.lib
| |
| + pd_volt3V Power Domain
| |
| | +
| | + library_set: libs-3volt
| |
| | +
| | + timing: libs/delayvolt_3V.lib libs/slow.lib
| |
| + pd_volt5V Power Domain
| |
| | +
| | + library_set: libs-5volt
| |
| | +
| | + timing: libs/delayvolt_5V.lib libs/slow.lib
| |
```

```
| + pd_default Power Domain
| |
| | + library_set: libs-2volt
| |
| | + timing: libs/delayvolt_2V.lib libs/slow.lib
|
+ Constraint Mode: default_constraint_mode
|
| + SDC Constraint Files: test.sdc
```

You can also use the EDI System GUI to review the design import configuration:

- Select *File* -> *Import Design* -> *Create Analysis Configuration*
- or
- Select *Timing* -> *MMMC Browser*.

For more information, see the [File Menu](#) chapter in the EDI System Menu Reference Guide.

## Adapting Command Scripts for MMMC Compatibility

This section provides important information required for adapting scripts to be MMMC compatible. The commands that are discontinued and the command options that are now required in EDI System 11 MMMC are largely related to RC extraction, delay calculation, and timing analysis. These commands have functionality, such as library loading and option adjustments, which is superseded by the MMMC architecture.

### Summary of Command Changes in EDI System 11

The following commands have reached end-of-life (EOL) in the EDI System 11 release. Using them will generate error messages from the software. These commands have had EOL warnings issued in the 10.x release of the software:

- `defineRCCorner`
- `loadStampModel`

- `loadTimingCon`
- `readCapTable`
- `setOpCond`
- `setQRCTechfile`
- `setRCFactor`
- `setTimingLibrary`
- `unloadTimingCon`

In addition, the following commands require additional arguments in order to perform properly in an MMMC environment:

- `rcOut`
- `read_sdf`
- `reset_timing_derate`
- `set_timing_derate`
- `spfIn`
- `write_sdf`

## MMMC Command Mapping Matrix

10.x Command	11.1 MMMC Command Mapping
<code>loadTimingCon</code> <code>unloadTimingCon</code>	<p>All loading and unloading of timing constraints is handled via the MMMC constraint mode objects. To change the constraints that the current analysis is using, you can use one of several options:</p> <ul style="list-style-type: none"><li>■ Use the <code>update_constraint_mode</code> command <code>-sdc_files</code> option to update the set of SDCs on the current constraint mode object to a new set.</li><li>■ Use the <code>create_constraint_mode</code> command to create a new</li></ul>

	<p>constraint mode using the new SDCs, then use the <code>update_analysis_view -constraint_mode</code> to associate the active view with the constraints.</p> <p>There is no formal way to simply unload the SDCs without reloading a new set of constraints.</p>
<code>readCapTable</code>	Cap table data, whether loaded from the configuration file or loaded via command, is now handled by the MMMC RC corner object. You can create and control RC corners with the <code>create_rc_corner</code> and <code>update_rc_corner</code> commands. For each <code>-best</code> , <code>-worst</code> , <code>- typical</code> capturable defined by the command, you should create separate MMMC RC corner objects.
<code>setQRCTechfile</code>	The QRC techfile should be now be configured in the MMMC RC corner objects. These are create and controlled by the <code>create_rc_corner</code> and <code>update_rc_corner</code> commands.
<code>defineRCCorner</code>	<p>The <code>defineRCCorner</code> command is used to specify the two corners of extraction data (out of the possible choices-best, worst, typical) to be used in two corner analysis. Assuming that you have created an MMMC RC corner for each of RC-best, RC-worst, and RC-typical, you should use the <code>update_delay_corner</code> command to assign the appropriate RC corner to the <code>-rc_corner</code> attribute of your BC and WC MMMC delay corners. During import, this is handled automatically.</p> <p>You must use <code>spfIn -rc_corner</code> in place of the <code>-latespif</code> and <code>-earlyspef</code>.</p>
<code>setRCFactor</code>	All RC scaling factors are handled directly by similarly named attributes in the MMMC RC corner objects and should be set using the <code>create_rc_corner</code> or <code>update_rc_corner</code> commands.
<code>loadStampModel</code>	STAMP model support has reached EOL and is not supported in the MMMC use model.
<code>setTimingLibrary</code>	The <code>setTimingLibrary</code> command was primarily used in 10.x to rebind min and max library sets so that different types of analyses could be performed; that is you could perform both Setup and Hold analysis on the max corner. In a proper MMMC configuration, there is no ambiguity with

how libraries are interpreted for bcWc vs. OCV style analysis so library binding intervention is not required. The mapping in MMMC is really a change in how you would configure the `set_analysis_view` command to achieve the desired result.

Examples: Assume you have configured two views sharing the same mode: `view-BC` and `view-WC`.

1. Setup at the max/WC corner; Hold at the min/BC corner

```
10.X: setTimingLibrary -max Max -min Min
```

```
11.1: set_analysis_view -setup view-WC -hold view-BC
```

2. Setup and Hold at the max/WC corner

```
10.X: setTimingLibrary -max Max -min Max
```

```
11.1 set_analysis_view -setup view-WC -hold view-WC
```

3. Setup and Hold at the min/BC corner

```
10.X: setTimingLibrary -max Min -min Min
```

```
11.1 set_analysis_view -setup view-BC -hold view-BC
```

<code>set_timing_derate</code>  <code>reset_timing_derate</code>	<p>The <code>-min</code> and <code>-max</code> corner selection options of the derating commands are replaced by the <code>-delay_corner</code> option in MMMC mode. Use the <code>-delay_corner</code> option plus the name of the MMMC delay corner you want to derate. All views using the delay corner will receive the specified derating factors.</p>
--	---

<code>read_sdf</code>  <code>write_sdf</code>	<p>Delay calculation has both a corner and mode dependency and so is done on an analysis view-specific basis in MMMC mode. In MMMC mode, you need to use the <code>-view</code> option to specify which view you want to annotate or generated the SDF to or from.</p>
---	--

Generally, you will normally generate a single SDF for each view where the min and max delay slots of the SDF will represent the early and late delays for the specific view. This is the most accurate representation of the delays for that view. There are also other ways of combining delays from multiple views into a single SDF. You should consult the EDI System Text Command Reference for more information on the various `write_sdf` options.

You will normally be reading an explicit SDF file per view. However, the

	<p>The <code>read_sdf</code> command allows the <code>-view</code> option to take a list of views if you which to annotate the same SDF file to multiple views. Consult the command reference for more information on the <code>read_sdf</code> options.</p>
<code>spefIn</code> <code>rcOut</code>	<p>When working in MMMC mode, both the <code>spefIn</code> and <code>rcOut</code> commands required the <code>-rc_corner</code> option along with the name of an active MMMC RC corner in order to properly import or export SPEF data.</p> <p>It is important to note that the extraction system requires all active RC corners to be in the same state. Either all RC corners should be subject to extraction or all should be subject to annotation. You cannot have a mix. The system will issue errors if this condition is not met.</p>

## 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 *EDI System User Guide*.

### Importing an OpenAccess Design

To import an OpenAccess design, complete the following steps:

1. Select *File - Import Design*.
2. Select OA .
3. Specify the following information:
  - *Library*  
Specifies the OpenAccess database library.
  - *Cell*  
Specifies the OpenAccess database cell.
  - *View*  
Specifies the OpenAccess database view.
4. Specify the following OpenAccess technology and physical library information:
  - *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.

5. Click Save or OK .

- Save saves your settings to a configuration file. The design is not imported.

- OK uses the current settings to import the design. The configuration file is not updated.

**Note:** In version 8.1 and earlier of the software, lib.defs was used by default. However, in EDI System 9.1 and later releases, the cds.lib plugin is used by default to improve interoperability between EDI System and Virtuoso.

If you still want EDI System to interoperate with 8.1 and earlier versions, use the lib.defs plugin.

To use lib.defs plugin, set the following:

```
setenv OA_PLUGIN_PATH install_hierarchy_root /share/oa/data/altplugins
```

## Importing a Design with LEF and Verilog

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

1. Select *File - Import Design*.

2. Specify the gate-level Verilog netlist files to import in the *Files* text field.

3. 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.

4. 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, EDI System reads the geometry information only from the first definition. For subsequent definitions, EDI System 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.

5. Click *Save* or *OK*.

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

## Loading a Previously Saved Global Variables File

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

1. Select *File - Import Design*.
2. Click *Load*. The Load Global Variables form is displayed.
3. Select the directory of the file you want to load.
4. Select *Global Variable files (\*.globals)* in the *Files of type* field.
5. Specify a file name or click on the filename in the window. The filename suffix is *.global*.
6. Click *Open*. The Load Global Variables form closes. The global variable file is loaded.
7. In the Design Import form, continue to specify data you want to import into the design.
8. 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 EDI System main window.

## Handling Verilog Assigns

Verilog assign statements may be added, removed, or replaced with buffers automatically by EDI System. However, if IPO cannot resolve design-rule violations (DRVs) of nets with an assign statement, you may need to replace the assign statement with a buffer by specifying the following command before loading a design:

```
setDoAssign on -buffer buffer_name
```

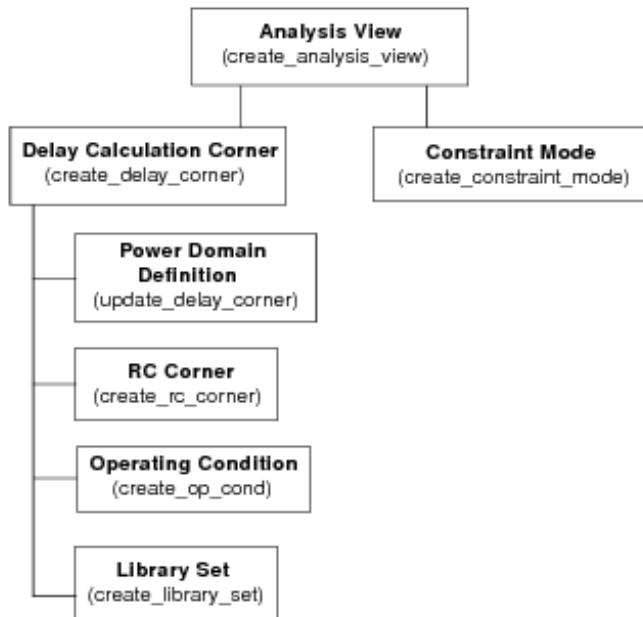
The above command replaces each assign statement with a buffer, including ones that are not involved with a DRV problem. It also does not affect an assign statement driven by 1'b1/1'b0 unless the following command is also specified:

```
setImportMode -bufferTieAssign 1
```

## 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.

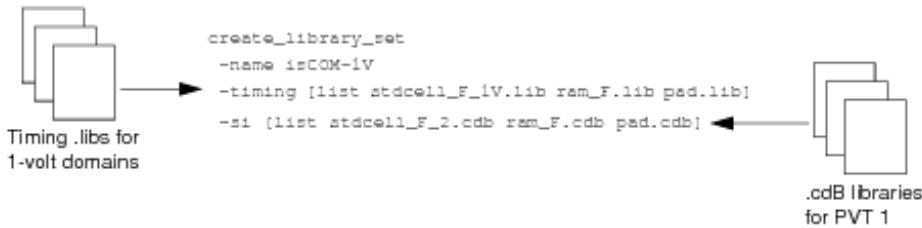
Library sets allow a group of library files to be treated as a single entity so that higher-level descriptions (delay calculation corners) can simply refer to the library configuration by name. A library set can consist of just timing libraries, or it also can include cdB libraries to keep timing and 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.

- 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

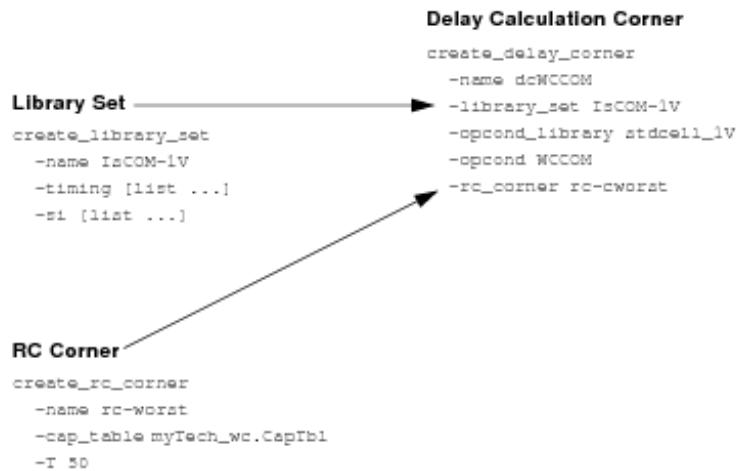
A delay calculation corner provides all of the information necessary to control delay calculation for a specific analysis view. Each corner contains information on the libraries to use, the operating conditions with which the libraries should be accessed, and the RC extraction parameters to use for calculating 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.

- 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 Encounter software isolates the following SDC constraints from conflicting with each other, in both multi-mode and multi-mode multi-corner analysis modes:

- `create_clock`

- `create_generated_clock`
- `set_annotated_check`
- `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.

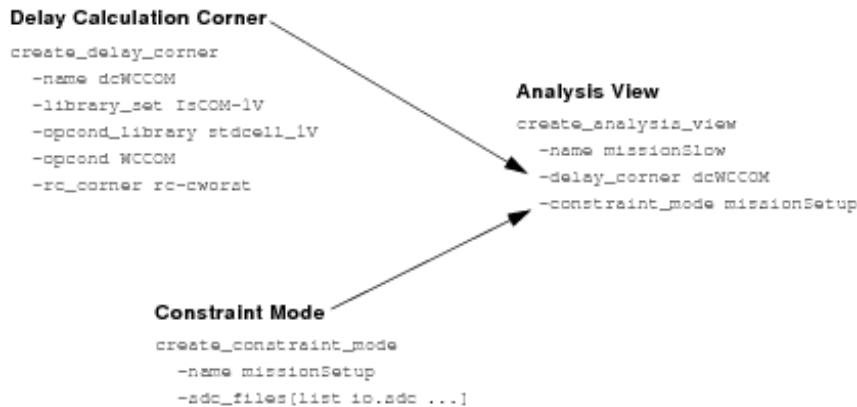
## Creating Analysis Views

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. Encounter applications can handle the views concurrently or sequentially, depending on their individual capabilities. Libraries and data are loaded into the system, as required to support the selected set of active views.

→ To set active analysis views, use the following command:

[set\\_analysis\\_view](#)

**Note:** You must define at least one setup and one hold analysis view.

For example, the following command sets `missionSlow` and `mission2Slow` as the active views for setup analysis, and `missionFast` and `testFast` as the active views for hold analysis:

```
set_analysis_view
  -setup {missionSlow mission2Slow}
  -hold {missionFast testFast}
```

## Guidelines for Setting Active Analysis Views

- The order in which you specify views using the `set_analysis_view` command is important. By default, the first views defined in the `-setup` and `-hold` lists are the default views. Certain Encounter applications that do not support multi-mode multi-corner can only process the data defined for a single view. These applications use the information defined for the default view.
- Concurrent analysis of views for timing optimization costs memory and CPU.

## Changing the Default Active Analysis View

Some Encounter applications can function only on a single analysis view at a time. By default, these single-view applications use the default analysis view. If an application or flow step does not provide a native option for specifying which view to use, you can use the `set_default_view` command to temporarily change the default view to a different active view that is better suited to that flow step.

For example, if the analysis view `missionSlow` is currently the default active setup view in the design, the following command temporarily changes the default view to `mission2Slow`:

```
set_default_view -setup mission2Slow
```

**Note:** Using the `set_default_view` command does not affect software performance because it only uses views that are already active in the design. If you use the `set_analysis_view` command to change the default views, the existing timing, delay calculation, and RC data is reset.

## Checking the Multi-Mode Multi-Corner Configuration

→ Use the following command to generate a hierarchical report of your current multi-mode multi-corner configuration:

```
report\_analysis\_views
```

You can customize the report to show only the active setup or hold analysis views, all of the active views, or all of the defined views in the design, including those that are currently inactive.

## Changing How the MMMC Browser Displays Configuration Information

By default, the MMMC Browser displays configuration information in two columns: one displays the different existing analysis views, and the other displays the different existing multi-mode multi-corner objects.

You can change how the MMMC Browser displays configuration information using the MMMC Preferences form. You can use this form to change the number of columns displayed, and rename

the titles of the columns.

You also can use this form (and the Add Object form) to add and delete objects from columns, and rearrange the order in which the objects are listed, by clicking and holding the left mouse button on an object name, and dragging it to a different position in the list.

→ Choose *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 EDI System GUI, click the Save Design command on the [File Menu](#) and then click the *Encounter* option button in the Save Design form.

The design files you save depend on the work completed during an EDI System 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 EDI System session. If you make no changes, EDI System references the original netlist when it saves the design. Do *not* use the Save Design form to save a partition.

## 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 EDI System and Virtuoso Chip Editor for ECO

1. From an EDI System session, save the OpenAccess design.

```
saveDesign sessionName -cellview { lib cell view }
```

2. Exit the EDI System session.

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

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

4. Save the design.

5. Exit the VCE tool.

6. Start an EDI System session.

7. 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 EDI System environment and exporting data out of the EDI System 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:

1. Import the design
2. Load the full chip (flat) floorplan, including partition specifications
3. Commit the partition without pin assignment or a timing budget

#### 4. Place and route each of the partitions

When you load a partition design, the EDI System 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 EDI System software treats the following items as floorplan data:

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

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

### Placement File Requirement

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

### Loading an I/O Assignment File

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

### Loading an FSDB File

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

### Saving a Partition

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

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

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

### Saving Floorplan Data

When you save a floorplan, the EDI System 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 EDI System 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 EDI System Database to GDSII Stream or OASIS Format

To convert an EDI System 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:

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

## Related Topics

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 *EDI System Text Command Reference* .

## Merging GDSII Stream or OASIS Files

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

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

### Merging Files Using the Command Line

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

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

If you specify the `-uniquifyCellNames` parameter, you must list the top-level file first, as the software uses the first name in the search path when renaming cells. For more information, see "Merge Examples".

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

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

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

- Top structure (the design data from the EDI System software)
- Via structures (output from the EDI System 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 unification, reference from x to Y changed to Y\_GDS1.

***Results***

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

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

For more information, see the following commands:

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

### Merging GDS/OASIS Files Using the GUI

Use the GDS/OASIS Export form.

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

For more information, see *Save - GDS/OASIS* in the [File Menu](#) chapter of the *EDI System Menu Reference* .

## About the GDSII Stream or OASIS Map File

When the software converts an EDI System database to GDSII Stream or OASIS format, it creates a file for mapping the layers in the EDI System 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>	Specifies a LEF layer from the LAYER statement in the LEF technology file.  If the <i>layerObjName</i> is a LEF layer name, the <i>layerObjType</i> must specify the DEF object type.
	COMP	Specifies component outlines.  If the <i>layerObjName</i> is COMP, the <i>layerObjType</i> must specify ALL.
	DIEAREA	Specifies the chip boundary.  If <i>layerObjName</i> is DIEAREA, the <i>layerObjType</i> must specify ALL.
	NAME	Specifies a text label for the layer name and

	<p>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 EDI System 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. (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>layerObjType</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,</li> </ul>

		VIAFILL, and VIAFILLOPC.
	BLOCKAGE	Equivalent to DEF BLOCKAGES without + FILLS.
	BLOCKAGEFILL	Equivalent to DEF BLOCKAGES with + FILLS.
	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 .  <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.

- i** Layer names or object types that exist in the EDI System 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

VIAFILL0PC 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 VIAFILL0PC: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  
  
...  
...
```

## 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 METAL1 NET 32 1	Two layers, each with a different data type	31:0, 32:1

## Updating Files During an EDI System Session

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

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

\* The EDI System 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 EDI System TCL functions while using the [setOaxMode](#) -bindkeyFile parameter.

<b>Virtuoso Key (Default)</b>	<b>SKILL Function</b>	<b>EDI System Key (Default)</b>	<b>EDI System Command</b>
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 <DrawThru3>	hiZoomOut()	Z	zoomOut
a	geSingleSelectPoint()	a	selectMode
c	leHiCopy()	c	copySpecialWire
e	leHiEditDisplayOptions()	e	popUpEdit
f	hiZoomAbsoluteScale (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 (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 (\\\"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 EDI System for all of the equivalent commands in the mapping.

---

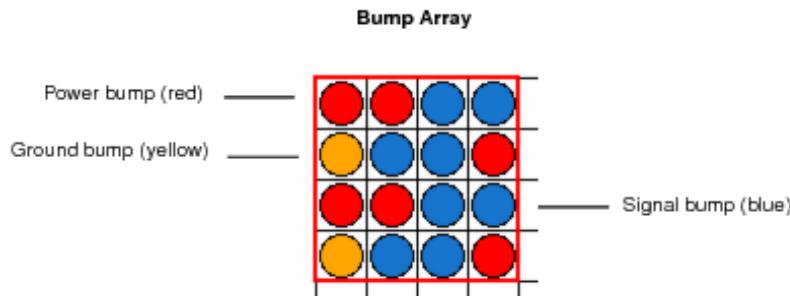
# Flip Chip Methodologies

---

- [Overview](#)
- [Flip Chip Flow in EDI System](#)
- [SiP Bump Flow](#)
- [Area I/O Flow](#)
- [Peripheral I/O Flow](#)
- [Differentiating Area I/O and Peripheral I/O](#)
- [Two-Layer RDL Routing](#)
- [Handling Flip Chip Designs with Complex Floorplans](#)
- [Pillar Bump Support](#)
- [Viewing Flip Chip Flightlines](#)
- [Point-To-Point Routing](#)
- [Virtual Connection Area for RDL Routing](#)
- [Port Numbering Feature for Power Nets](#)
- [Multi-PG Pads to Multi Bumps Assignment with a Controlled Ratio](#)
- [Distributed Co-design](#)
- [Swapping Signals](#)
- [Creating Constraints for Flip Chip Routing](#)
- [Examples and Report Files](#)
- [ECO Routing](#)

## 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 EDI 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 EDI System 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 EDI System".
- 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

## Related Flip Chip Information

- Text commands

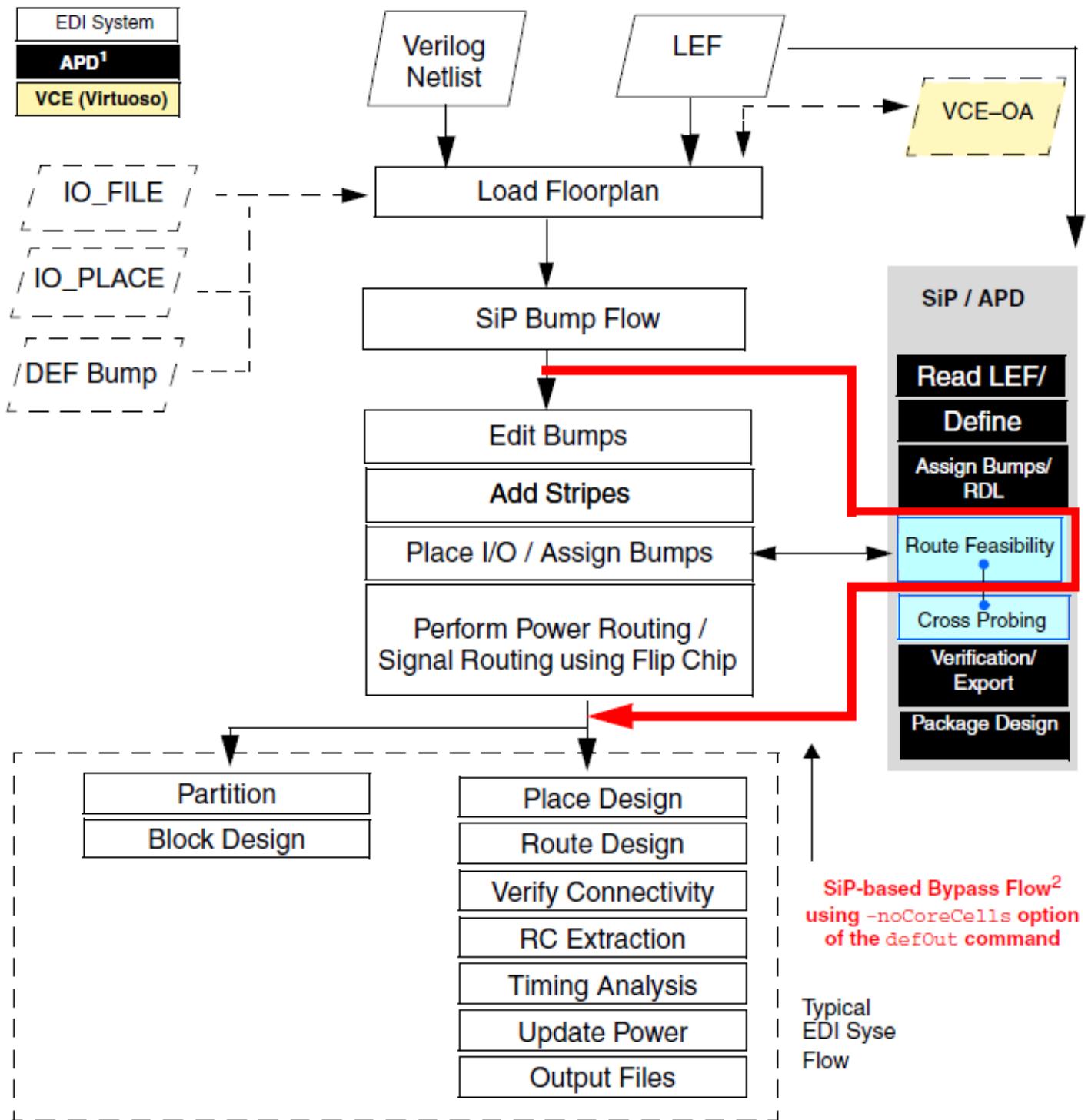
For information on the flip chip commands, see the "[Flip Chip Commands](#)" chapter of the *EDI System Text Command Reference*.

- Flip Chip Toolbox Menu

For information on the flip chip forms, see the "[Flip Chip](#)" section of the *Tools Menu* chapter in the *EDI System Menu Reference*.

## Flip Chip Flow in EDI System

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



<sup>1</sup> License Required

<sup>2</sup> Bypasses *Flip Chip Toolbox* menu (see Reducing Data Size for SiP Import (Bypass Flow))

## Flip Chip Flow Steps

### 1. Load the floorplan.

Load the floorplan as in a typical EDI System flow.

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

The following files are imported during this step:

- Verilog netlist

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

- LEF File

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

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

For more information, see Differentiating Area I/O and Peripheral I/O .

*Text Command : [loadLefFile](#)*

- OA database via Virtuoso (VCE)

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

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

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

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

- The IO\_FILE contains bumps, I/O rows (optional), and I/O instances (optional).  
For an example IO\_FILE, see the "IO\_FILE Example" section.

*Text Command : [loadIoFile](#)*

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

*Text Command : [loadIoFile](#)*

- The DEF file can be used to import bumps.

*Text Command : [defIn](#)*

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

- Bump Flow-- See "SiP Bump Flow".

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

- Bump Matrix Generation

Use the bump matrix generator. These bumps will be assigned later in step 5.

- **IO\_FILE Generation**

Generate an IO\_FILE that contains the x and y locations of the bumps along with the x and y locations of the I/O rows. The I/O rows are the rows or sites into which the I/O driver cells are placed. These bumps may or may not be assigned to signals at this time.

- **APD Bump Generation**

Use APD to generate the bump matrix or other DEF input file, and pass the bumps via a DEF instance.

**3. Edit the bumps.**

Use the following flip chip forms to edit bumps. For more information on these forms, see the [Tools Menu](#) chapter of the EDI System Menu Reference:

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

**4. Add stripes.**

Generate the power stripes on the chip using the [addStripe](#) text command.

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

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

Use the [placePIO](#) command to perform initial peripheral I/O pad placement. After you run the [assignBump](#) command to assign the signal and power/ground bumps, use the [placePIO](#) - assignBump - noRandomPlacement command and options to optimize the initial bump assignment.

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

Use the [fcroute](#) - type power command and option to connect the power and ground bumps to stripes. Use the fcroute - type signal command and option to connect the signal bumps to the I/O driver cell specified in the netlist.

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

- i** Before running the `placePIO` and `fcroute` commands, you must specify the flip chip constraints using the [setFlipChipMode](#) command which loads data for `placePIO` and `fcroute` commands.

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

The remainder of the flow is similar to the typical EDI System flow.

7. Partition the design.  
Bumps, bump routing, power routing, and I/O driver cells can be pushed down as blockages into the partition. See [specifyPartition](#) and [handlePtnArealo](#) commands for more information.
8. Place the design.  
Place the design using the `placeAIO` command.
9. Route the design.  
`NanoRoute` ([globalDetailRoute](#) command) can be used to connect the regular nets in the design.
10. Verify the connectivity.  
Verify the bump (physical cells) connections to the logical cells using the [verifyConnectivity](#) command.
11. Run extraction.  
Extract the RC data using the signoff extraction mode `setExtractRCMode -engine postRoute -effortLevel signoff` command and then generate a SPEF file. The signoff extraction run input is the DEF output file which contains the bumps that are not present in the original Verilog file. You can create a Verilog output file containing bumps to match the signoff extraction run SPEF.  
**Note:** You can also create a defout file and convert the bumps to pins so you do not have to create a physical verilog.
12. Do a timing analysis.  
The timing analysis report is the same as in the normal EDI System flow.  
See [report\\_timing](#) command.
13. Update power.  
Update power using the [report\\_power](#) and [analyze\\_early\\_rail](#) commands. A flip chip design can have multiple power sources.
14. Output the files.  
Write out the DEF, Verilog, OpenAccess, SPEF, and GDSII files. The [defOut](#) command contains the `-noCoreCells` option to reduce the data sent to APD. For more information, see the ["Reducing Data Size for SiP Import \(Bypass Flow\)"](#).

## 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 EDI System).

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

## Splitting Wires in Metal Layers

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

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

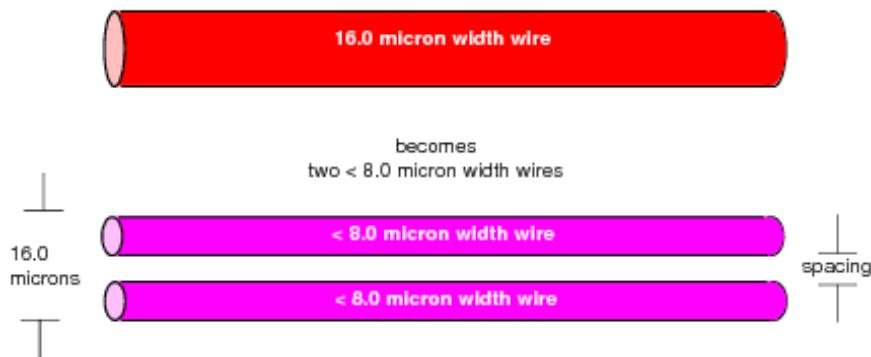
```
LAYER M5 TYPE ROUTING ; DIRECTION VERTICAL ;
WIDTH 0.70 ; SPACING 0.70 ; PITCH 1.4 ;
CAPACITANCE CPERSQDIST 0.0001000 ; RESISTANCE RPERSQ 0.010000 ;
MAXWIDTH 8.0 ;
END M5
```

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

The following figure shows how a 16.0 micron wire is split using this LEF layer code and the

editFixWideWires command. The resulting split wires will be slightly less than 8.0 microns each. There will be a split spacing between the wires such that the total width is 16.0 microns.

The split spacing is automatically determined by considering the MINSPACING, PARALLEL RUNLENGTH SPACING, and DENSITY constraints. The split spacing will be greater than or equal to the MINSPACING constraint. There is no manual control for the split spacing parameter.



## Testing the Package Routing Feasibility

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

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

## Area I/O Flow

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

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

1. Load the floorplan.  
Use the [Load FPlan File](#) form to load the floorplan file.
2. Define the bumps.  
Use the [Create Bump Array](#) form to set up the bump array.
3. Create area I/O driver rows.  
Use the [Create Area IO Row](#) form to set up the area I/O rows.
4. Place Area I/O pads and standard cells.  
Use the [Place Area I/O](#) form to place I/O driver cells.

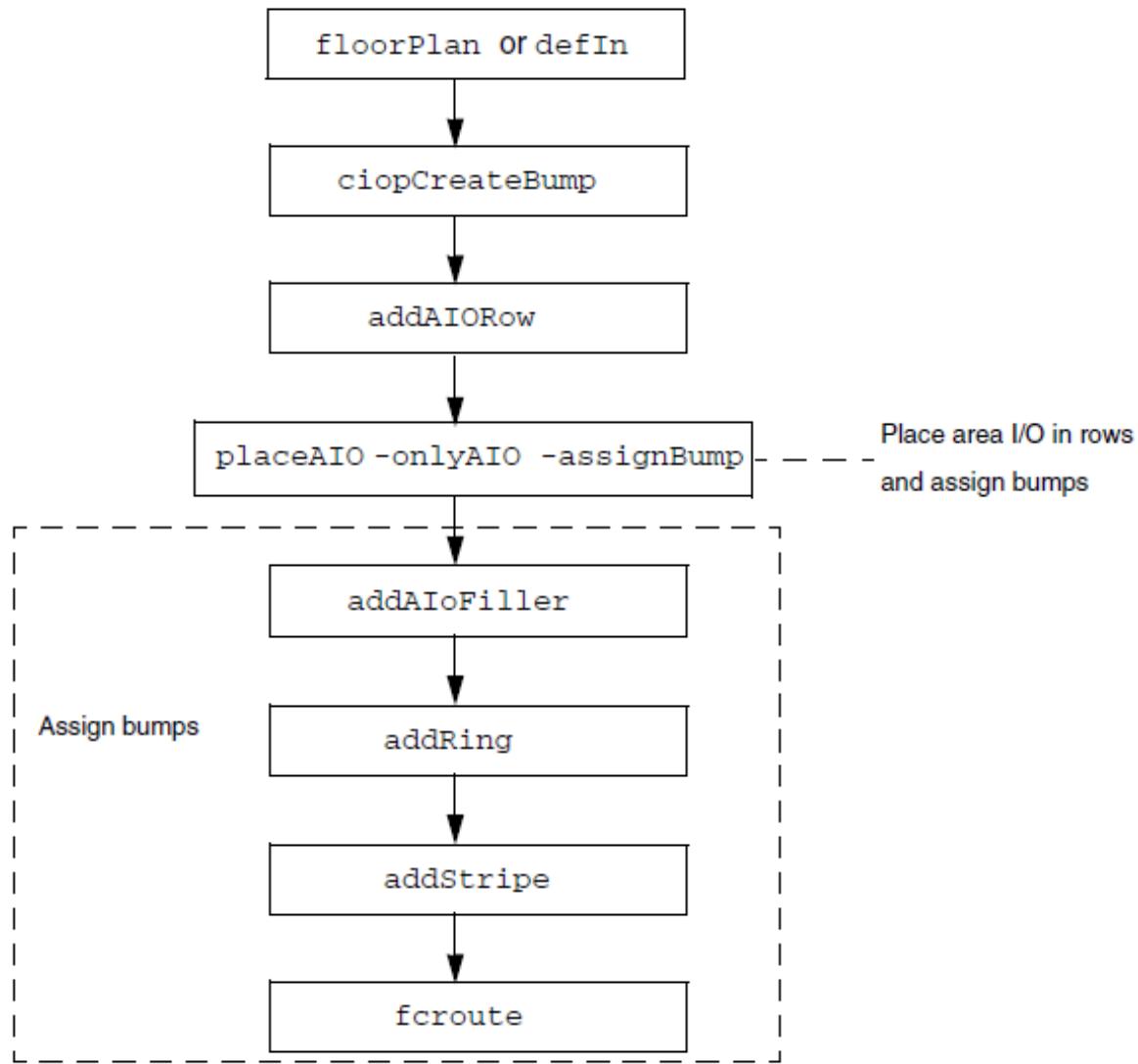
5. Assign signals, power, and ground to the bumps.
  - Use the [Assign Signals](#) form to assign the signals to the bumps. Signal bumps are blue-filled squares.
  - Use the [Assign Power/Ground Bumps](#) form to assign power and ground to bumps. Power bumps are red-filled squares. Ground bumps are yellow-filled squares.
6. Add area I/O filler cells in the blank sites of the specified area I/O row clusters using the [addAloFiller](#) command.
7. Create power rings and stripes.
  - Use the [Add Rings](#) form to create rings around the core area and around the power and ground bumps.
  - Use the [Add Stripes](#) form to create stripes that connect to the power and ground bumps.
8. Connect power, from bumps to I/O cells or from bumps to rings/stripes.  
Use the [Route Flip Chip - Advanced - Routing Style](#) form to establish the power connections.

**Note:** The remainder of this flow is similar to the typical EDI System flow.

## Area I/O (AIO) Command Flow

The area I/O command flow is described as follows:

## Area I/O Flow using Text Commands



## Routing Bumps to I/O Driver Cells (Hierarchical Area I/O Flow)

The hierarchical area I/O flow allows you to route the bumps, using the [fcroute](#) command, to I/O driver cells and then push down (partition) this data into a lower level.

The text command is:

[handlePtnAreaIo buffer\\_name](#)

This command pushes down data in the partition as follows:

- Bumps become routing blockages
- I/O cells become placement and routing blockages
- An internal pin is created over the I/O cell pin
- A boundary pin is created
- A buffer is created between the internal pin and the boundary pin

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

## Flip Chip Routing on Shielded Nets in AIO

When using the fcroute shielding option in the AIO mode with manhattan (90 degree) routing style, the defOut marks the shielded nets as 'SHIELD', while displaying the SHAPE and ROUTED status of the metal shield wire.

### 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

The defOut contains the SHIELD syntax as follows:

```
-scan_out_2 ( Bump_27_6_2 PAD ) (IOPADS_INST/Pscanout2op PAD)
+ ROUTED METAL8 16000 + SHAPE IOWIRE (1255310 541920 ) ( 1369310 *)
NEW METAL8 16000 + SHAPE IOWIRE (1263310 533920 ) ( * 695760 )
+ PROPERTY BUMP_ASSIGNMENT "ASSIGNED"
;
-VSS ( * VSS )

+ SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275310 554320 ) ( 1315310 *)
NET METAL7 16000 + SHAPE IOWIRE ( 1255310 541920 ) ( 1315310 *)
NET METAL8 800 + SHAPE IOWIRE ( 1250510 529520 ) ( 1315310 *)
+ ROUTED METAL6 16000 + SHAPE STRIPE ( 1553200 109600 ) ( * 186800 )
NET METAL6 16000 + SHAPE STRIPE ( 1753200 109600 ) ( * 186800 )
+ SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275710 553920 ) ( * 675760 )
METAL7 16000 + SHAPE IOWIRE ( 1263310 533920 ) ( * 695760 )
METAL8 800 + SHAPE IOWIRE ( 1250910 529120 ) ( * 675760 )
```

## Peripheral I/O Flow

The peripheral I/O approach to flip chip methodology places I/O driver cells at the edges of the core area of the design. This means that the bump cells require routing to the I/O driver cells using the top-most layer with or without one extra layer below. This layer is called the redistribution layer (RDL), and is used to connect the bumps to the I/O pads. The procedures and examples in this section use the two-layer approach.

The peripheral I/O flow is similar to area I/O, wherein you can use I/O rows (regions/sites) to place the I/O driver cells since they remain on the boundary. The peripheral I/O flow also includes non-orthogonal (45-degree) RDL routing, I/O cell optimization, and bump reassignment for better single layer routing.

Since the top two layers are used for RDL routes, and RDL routes are wider than regular routes, coupling effects from the RDL routes to regular routes can be significant. To avoid huge coupling effects, avoid regular routing in one layer below the RDL.

From the 10.1 release, multiple layer routing has been enabled in the PIO mode. To turn on multiple layer routing in PIO mode and set layer change preferred location, set the PIOLAYERCHANGE variable in the `fcroute` constraint file as follows:

```
PIOLAYERCHANGE {mode_number | mode_name}
```

The following table describes the possible modes:

Mode	Mode Name
0	OFF
1	PAD
2	BUMP
3	PADBUMP

Examples of usage:

```
PIOLAYERCHANGE 2
```

or

```
PIOLAYERCHANGE BUMP
```

There are three major aspects of the peripheral I/O flow:

- RDL planning and routing
  - The automatic placement of the I/O cells on the edge of the design
  - The optimization of the I/O cells and the reassignment of bumps to enhance single layer routing.
  - Non-orthogonal routing on the redistribution layer (RDL).
- RC extraction
- Signal integrity and timing analysis

## Data Preparation

The LEF CLASS statements for I/O pad cells and bump cells must contain the following classes for the peripheral I/O flow to work.

I/O cell: CLASS PAD AREAIO

Bump cell: CLASS COVER BUMP

These are the LEF properties used for connecting power/signal bumps to power/signal I/O cells.

Normally, the bump to I/O pad connection is defined in the Verilog file. The signal names are specified in the Verilog top module port list, and the I/O cells are connected to these ports.

For I/O power pads which are not defined in the Verilog file, you can define the connection of the I/O pads to bumps using the command:

- [setFlipChipMode](#) -connectPowerCellToBump

The MACRO PIN statement added in LEF 5.7 tells which power/ground pin shape on the I/O driver cell must be connected to a bump. See [Defining the Connection between a Bump and P/G Pin Shape](#) in the "Data Preparation" chapter of the *EDI System User Guide*.

## Peripheral I/O Flow Steps

The peripheral I/O implementation flow is similar to the traditional physical implementation flow, except for the handling of bump cells and RDL routing.

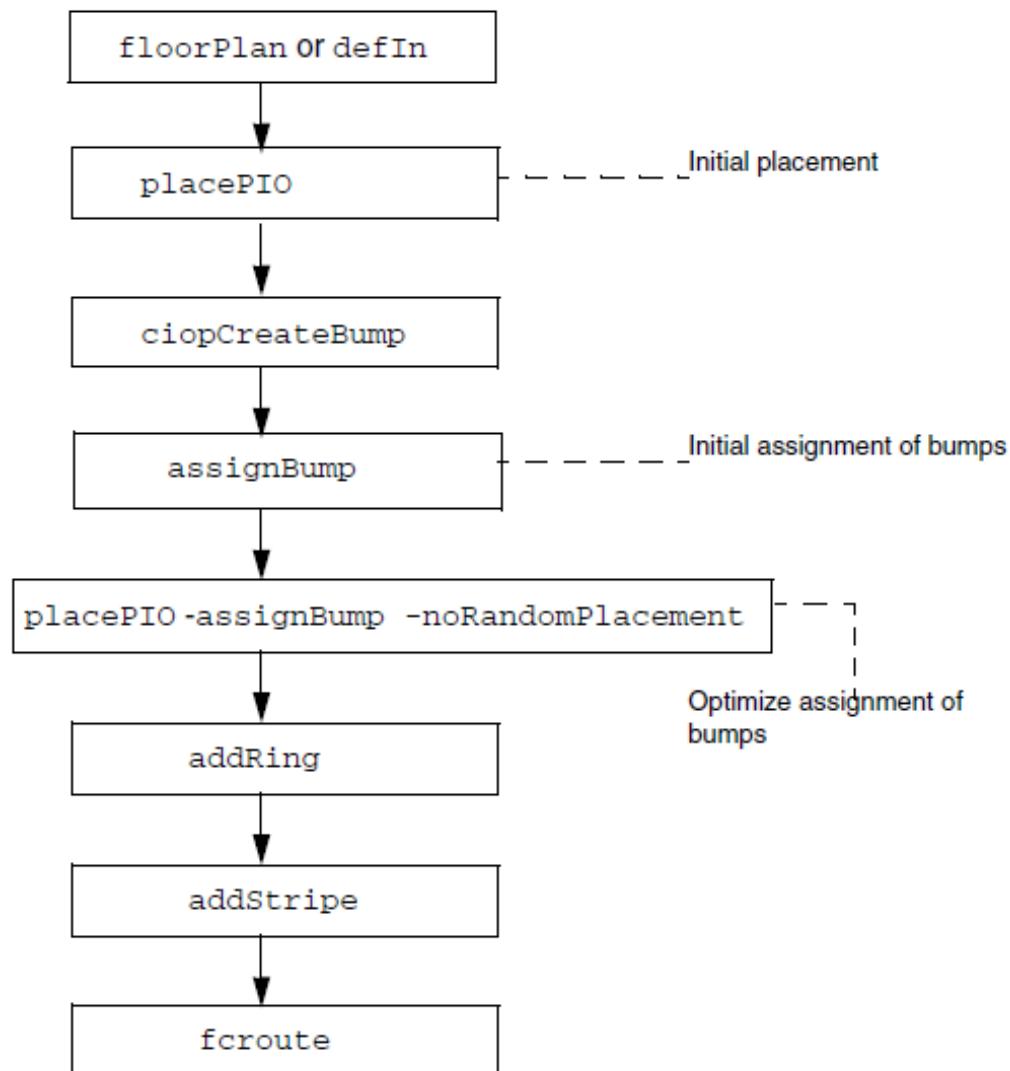
There are four major elements of the flow:

- After the initial floorplanning stage (set die and area and place I/O driver cells), the RDL implementation flow includes bump placement and assignment, optimization of I/O driver cell placement, and RDL routing.
- The bump placement and assignment is passed to APD (Allegro® Package Designer) for package design. You can determine the route feasibility by using APD to check the bump routability to the package. This can be invoked from the EDI System user interface.
- The RDL-routed design is then ready for power planning / QRC / other placement and routing operations.
- Initial parasitics can be extracted in EDI System using the extractRC command. If more accurate parasitics are required, the signal-routed design can be streamed out in GDSII format and sent to Assura™ RCX for extracting RC parasitics, which can be used for timing and SI analysis with the RDL effects.

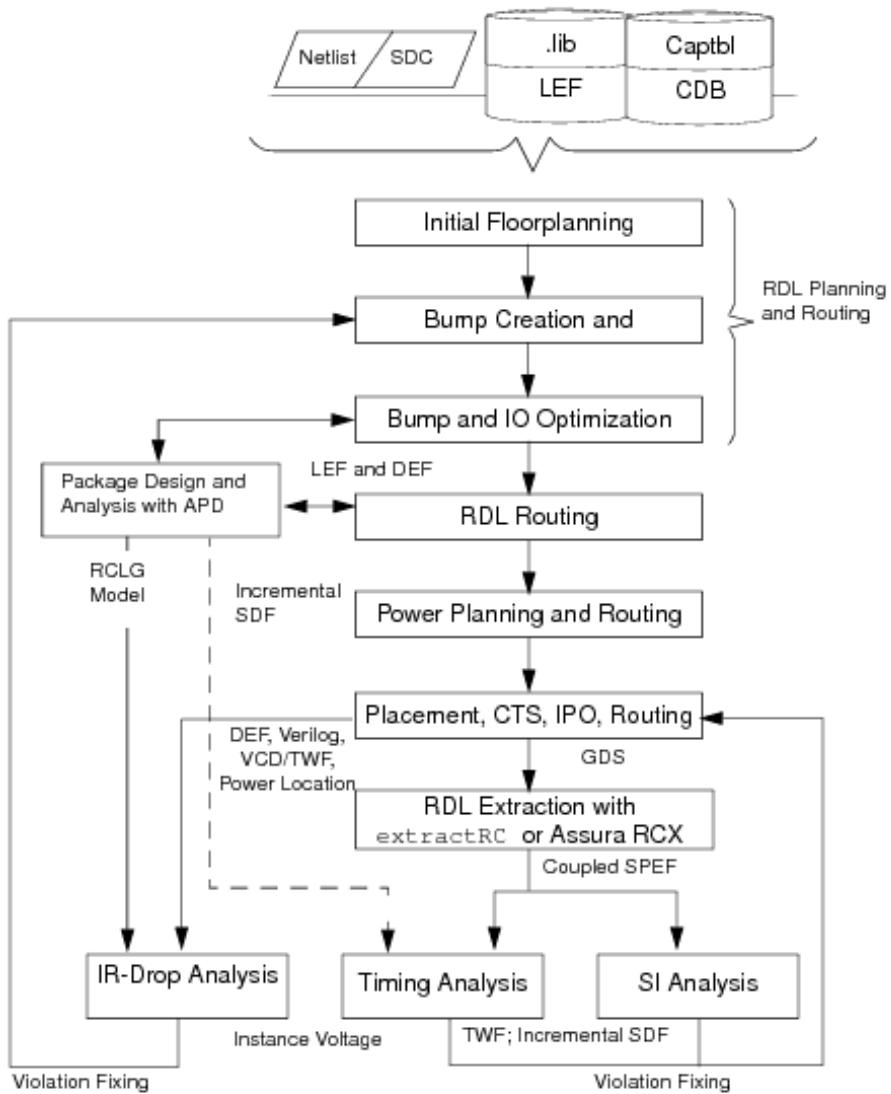
## Peripheral I/O (PIO) Command Flow

The peripheral I/O command flow is described as follows:

## Peripheral I/O Flow using Text Commands



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



## RDL Planning and Routing

The following are the basic steps for planning and routing in a peripheral I/O flow.

- ## 1. Load the floorplan.

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

- ## 2. Define the bumps.

Create a bump matrix based on bump pitch and other parameters by using the EDI System bump selection and assignment user interface or the [ciopCreateBump](#) text command.

From the EDI System user interface, select *Tools -> Flip Chip -> Create Bump Array*.

3. Place the peripheral I/Os. Refer to the "Place peripheral I/O pads" section for details.
4. Assign the power and ground bumps either by loading a predefined I/O File using the [loadIoFile](#) command or by using the EDI System bump selection and assignment user interface or the text command, [assignPGBumps](#).

From the EDI System user interface, select *Tools -> Flip Chip -> Assign Power/ Ground*.

5. Assign the signal bumps by either loading a predefined I/O File using the [loadIoFile](#) command or by using the EDI System bump selection and assignment user interface or the text command, [assignBump](#).

From the EDI System user interface, select *Tools -> Flip Chip -> Assign Signal*. The `assignBump` command uses the signal names (ports) in the Verilog top module list and assigns them to the closest I/O cell. The `assignBump` command assumes the I/O cells have been preplaced.

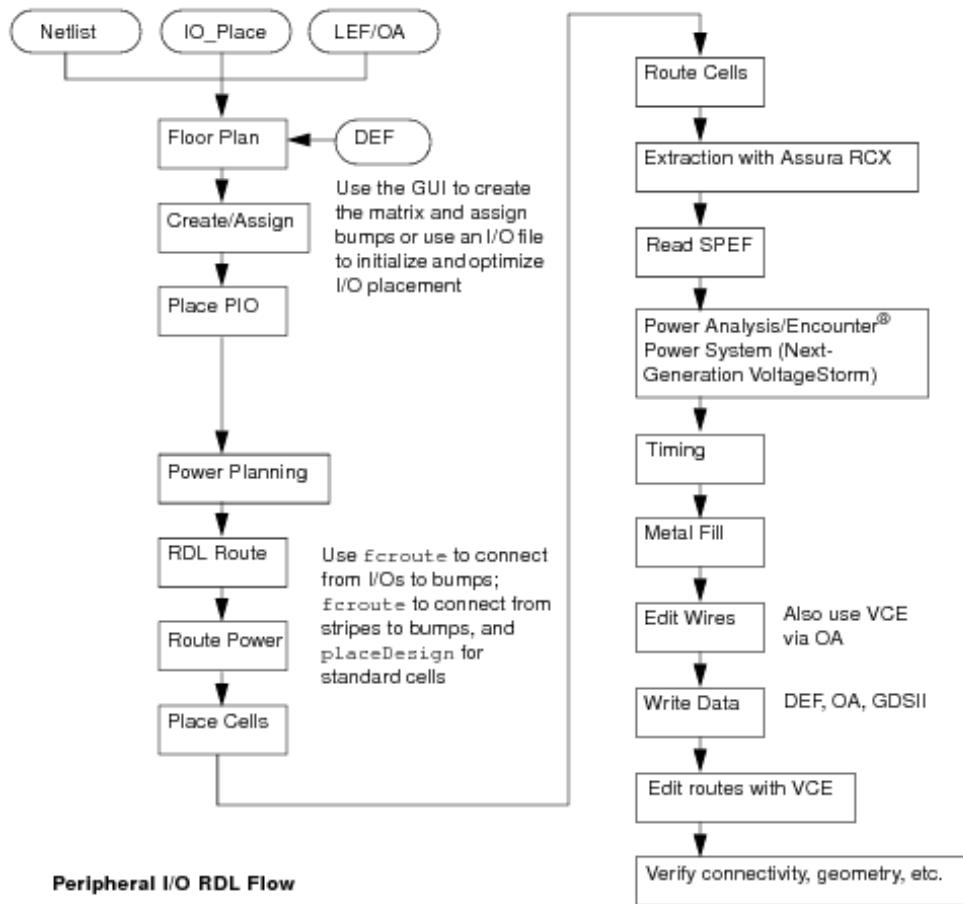
6. Route the signal and power/ground bumps to the I/O driver cells or power/ground stripes using the `fcroute` command. Refer to the "Route bumps" section for details.
7. If the routing is not optimal, either reassign the bumps or change the I/O cell placement using the [placePIO](#) command. Refer to "Reassign bumps" section for details.
8. Snap or split route.
9. Create power rings and stripes.
  - Use the [Add Rings](#) form to create rings around the core area and around the power and ground bumps.
  - Use the [Add Stripes](#) form to create stripes that connect to the power and ground bumps.
10. Connect power, from bumps to I/O cells or from bumps to rings/stripes.

Use the [Route Flip Chip - Advanced - Routing Style](#) form to establish the power connections.

**Note:** The remainder of this flow is similar to the typical EDI System flow.

The EDI System log file displays the routing status of each bump-pad pair in the `fcroute` PIO mode.

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



## Place peripheral I/O pads

Since CLASS PAD AREA I/O cells are not automatically placed, you must invoke the [placePIO](#) command to randomly place the I/Os on the periphery.

Alternatively, from the EDI System user interface, select *Tools -> Flip Chip -> Place & Route -> Place Flip Chip I/O -> Peripheral I/O*.

**Note:** The [placePIO](#) command also reads flip chip options from `setFlipChipMode` command.

Another method for creating the initial I/O placement is to read in an I/O file specifying the Pad: or `IOInst:` syntax with the instance name and side of the design or XY location.

You can refine the initial placement or reassign bumps by using various command options. The initial placement can be modified in two ways:

- Fixed Bumps

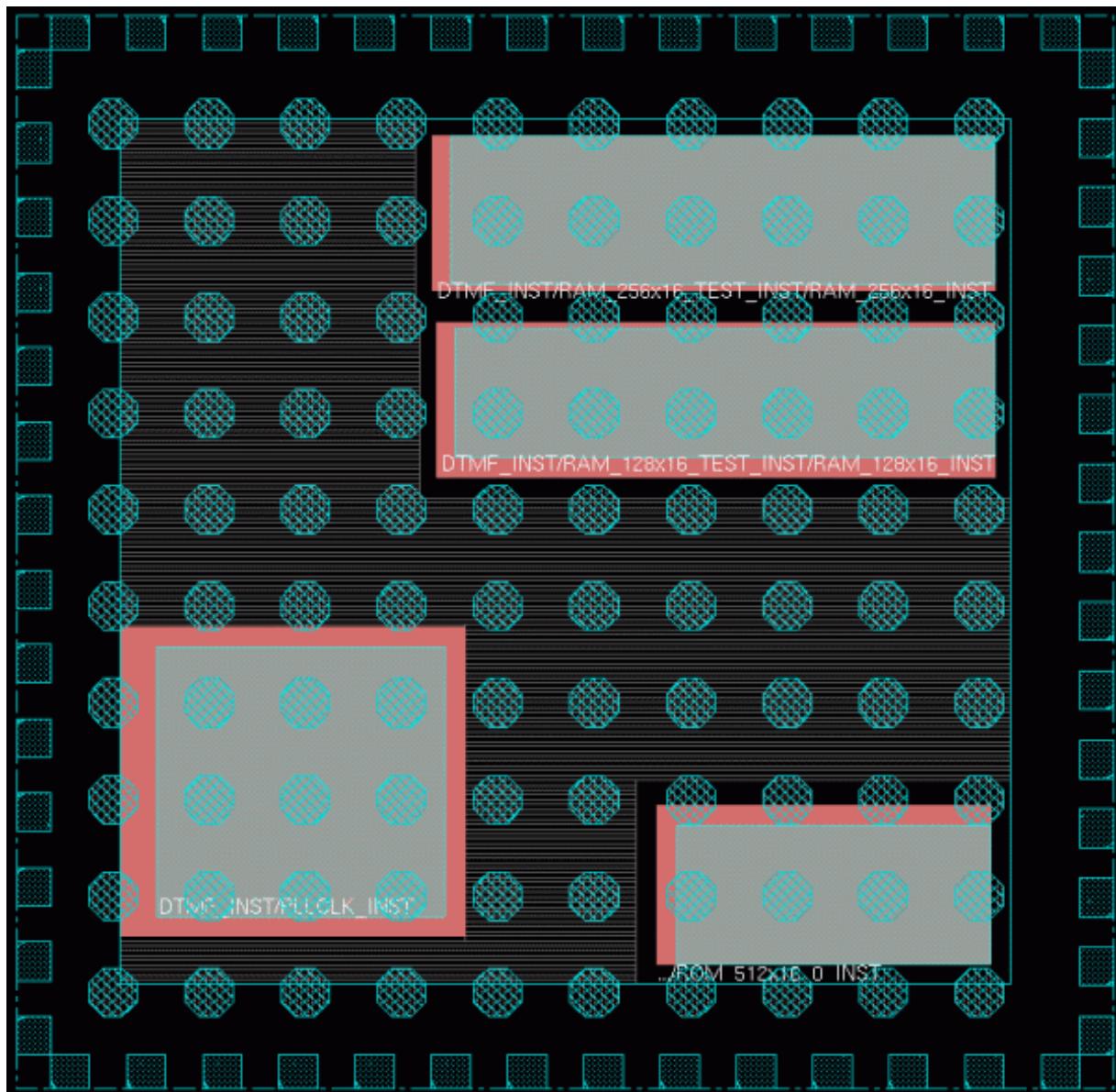
If the bumps have been assigned, the I/O cells can be moved using the `placePIO` command to help ensure a one-layer route.

- Fixed I/Os

The bumps can be reassigned to improve the routing if the I/O cells have been fixed.

Once the placement is finished, the data can be stored in the floor plan file and restored. The I/O cells can also be moved manually with the `move` command since there are no specified I/O rows.

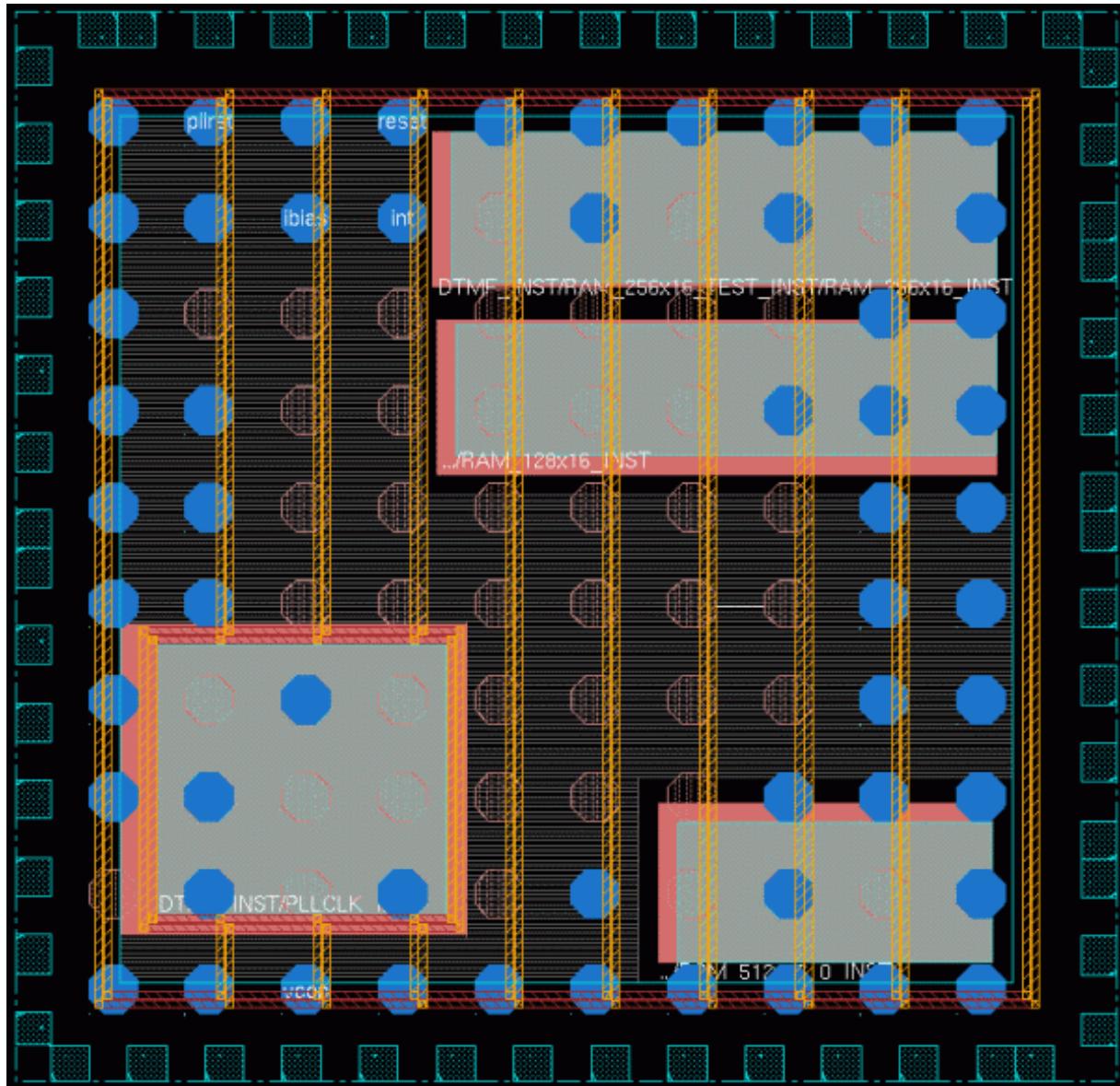
The following figure shows the results of the `placePIO` command.



## Optimize peripheral I/O placement

During peripheral I/O placement, you can specify a constraint file which controls certain features of the optimization. For an example of a constraint file, see [Routing and Placement Constraints](#).

The following figure shows the results of the `placePIO` command.



**Note:** `placePIO` has two features. The first is to randomly place the I/O cells on the periphery, and the second is to optimize the I/O pad cells. If you specify an I/O file with the `-ioFile` option or use the `-noRandomPlacement` option, `placePIO` does not do a random placement.

## Reassign bumps

To optimize the bump assignment, use the `placePIO -assignBump -noRandomPlacement` command.

## Route bumps

This command routes the bumps to I/O cells using 45-degree routing.

1. In EDI System, select signal routing type, using `fcroute -type signal`  
By default, 45-degree routing style is selected.
2. Select the peripheral I/O routing style using the command `fcroute -designStyle pio`.  
This option calls both the detail and global routers to route the bumps.
3. Set the minimum escape distance from the bump to where the route can proceed at an angle.  
Specify either the Minimum Escape Distance constraint on the form or use the command  
`fcroute -minEscapeDistance unit` to specify the distance.
4. To specify the minimum distance before the route can turn, you must create a configuration file  
(`fcroute.config`) and include the following command:

```
srouteMinlength value
```

Use the command `-fcroute -extraConfig fcroute.config` to specify the file or specify the file from the *Route Flip Chip Advanced* form.

5. You can specify the routing constraints by using the `-constraintFile` option.

```
fcroute -constraintFile file_name
```

For example:

```
fcroute -constraintFile fcroute.constr
```

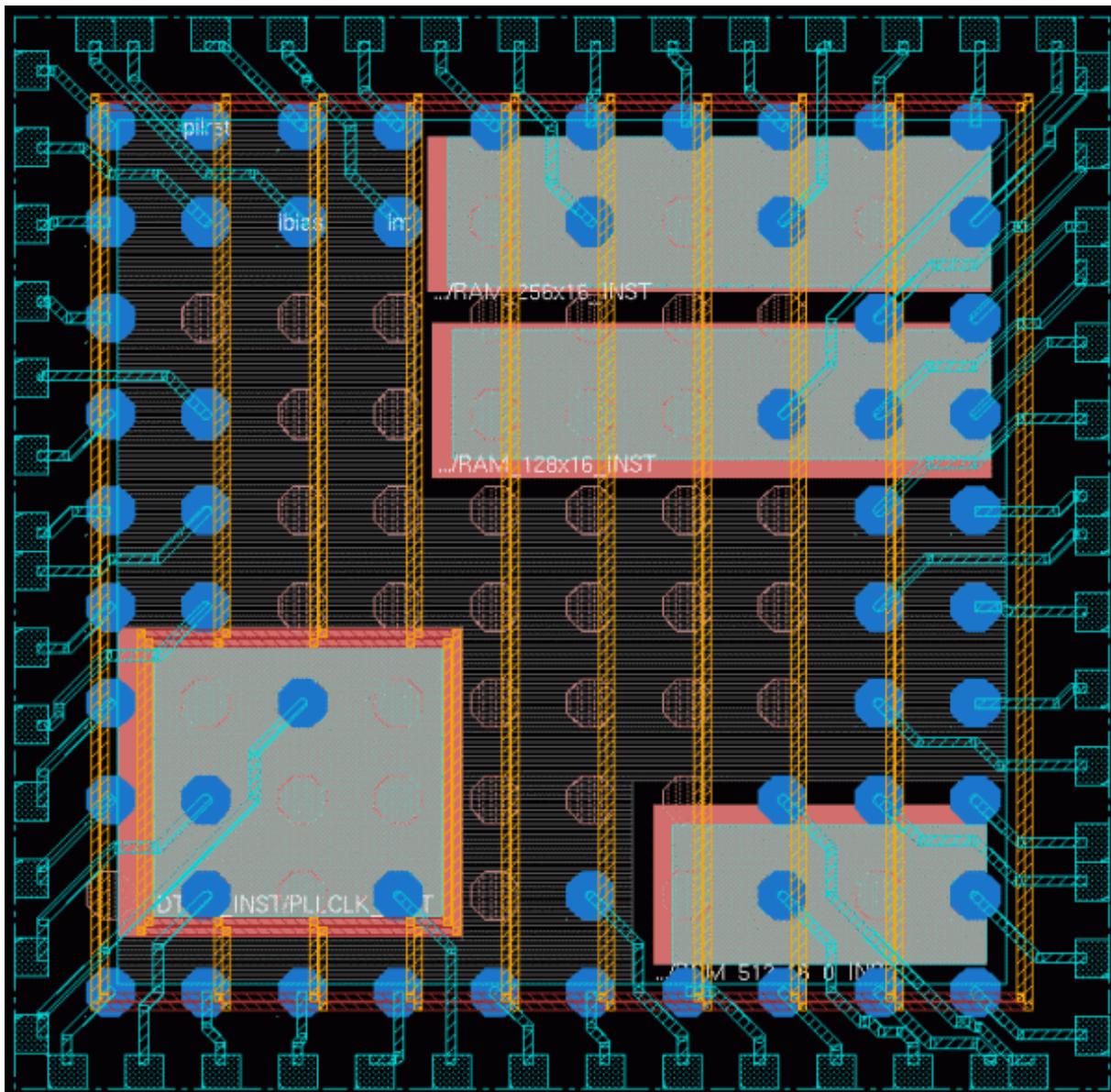
For an example of an `fcroute` constraint file, see [Routing and Placement Constraints](#).

Alternatively, you can specify basic and advanced routing and placement constraints using the [\*Flip Chip Route\*](#) form in the EDI System GUI.

For more information, see the following topics in the *Tools Menu* chapter of the *EDI System Menu Reference* :

- [Flip Chip Route- Basic](#)
- [Flip Chip Route - Advanced](#)

The following figure shows the results of the `fcroute` command.



### Splitting wires

The `fcroute` command splits the route during the routing process. You can invoke the wire splitting function during `fcroute` by specifying the `MAXWIDTH` value in the LEF layers section.

LAYER METAL7

....

```
MAXWIDTH 10.0 ;
```

### Adding power stripes

You can use the [addStripe](#) command to add a power stripe over or between power bumps without specifying the exact xy locations. If the stripe is on a different layer than the bump layer, addStripe will automatically drop a via array.

From the EDI System user interface, select *Power -> Power Planning -> Edit Power Planning Option -> Stripe*

You can also open the Edit Power Planning Option form by clicking the + icon next to the *Use option set* field on the *Basic* tab of the Add Stripes form.

### Routing the power bumps

→ Route the power routes to the stripes by using the EDI System user interface or the [fcroute](#) - type power command.

From the EDI System user interface, select *Tools -> Flip Chip -> Place & Route -> Route Flip Chip -> Advanced -> Routing Style -> Connect Power*

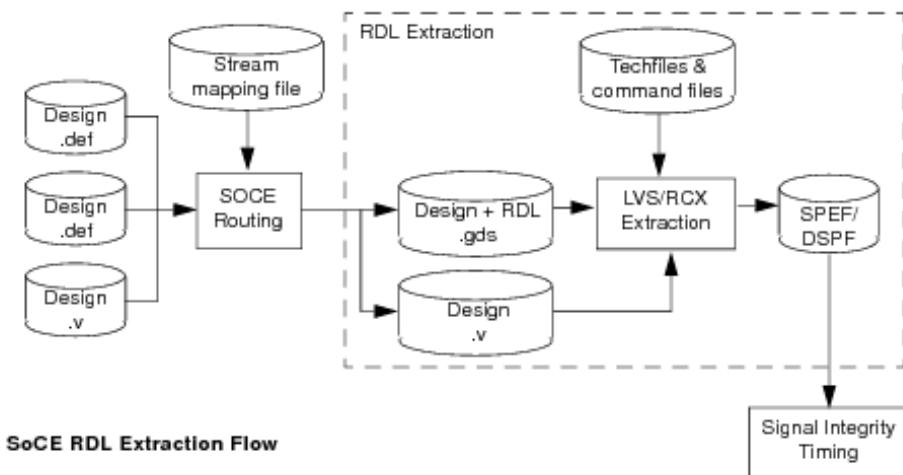
## Peripheral I/O Extraction

In the RDL extraction flow for designs using peripheral I/O methodology, EDI System 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 EDI System. If all the nets are back annotated, EDI System displays the following message:

0 nets are missing in SPEF file.

3. Perform timing analysis in EDI System.

- 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 EDI System. 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 EDI System.

```
timeDesign -postRoute -reportOnly -si
```

## Differentiating Area I/O and Peripheral I/O

The LEF I/O Driver cells must contain CLASS PAD (for peripheral I/O) or CLASS PAD AREAIO (for area I/O).

**Note:** Depending on your design style, you may need to modify the LEF macro CLASS statement.

- Area I/O  
CLASS PAD AREAIO = I/O cell without bump.

CLASS PAD AREAIO is used by the [assignBump](#) and [placeAIO](#) commands.

Additionally, the SITE must be defined and referenced in the LEF macro. See [Performing Area I/O Placement in the Data Preparation chapter](#) for more information and example.

- Peripheral I/O

CLASS PAD = I/O cell with bound pad.

CLASS PAD is used by the io\_placer to place the pads along the boundary.

By default, the CLASS PAD macro is automatically placed along the boundary when the configuration file is read. You can also load a file with the [loadIoFile](#) command.

The normal wire bound I/O cells are CLASS PAD. However, to use the [assignBump](#) and [placePIO](#) commands, they must be CLASS PAD AREAIO even on the periphery.

#### LEF MACRO CLASS PAD and PAD AREAIO

To support a peripheral I/O-driver with flip-chip bumps flow, PAD AREAIO cells are allowed outside the core box.

- LEF MACRO CLASS PAD has the bonding pad built into the cell.
- LEF MACRO PAD AREAIO has no bonding pad built-in, so it requires routing to the bump.

## Two-Layer RDL Routing

As flip chip design become more and more complex, one layer may not be sufficient for completing RDL routing. EDI System 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 "l1x1 l1y1 ur1 ur1 l1x2 l1y2 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 ur1 ur1 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 ur1 ur1 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_1l_x rect_1l_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 honor routing blockages. If you use a routing blockage with other routing constraints, `fcroute` honors the mixed usage as well.

## 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.

## 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:

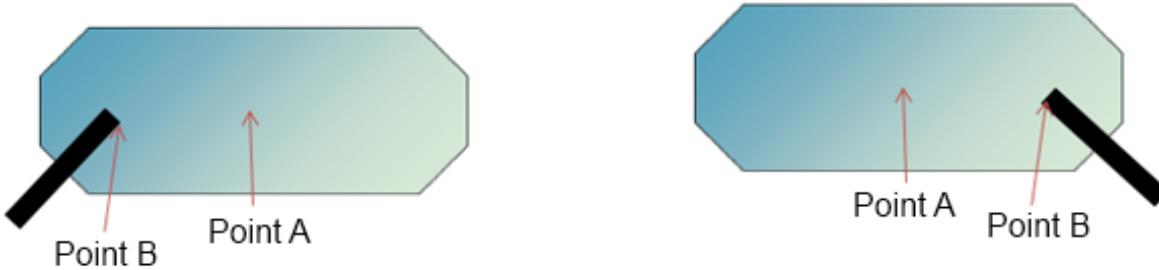
Standard Bump



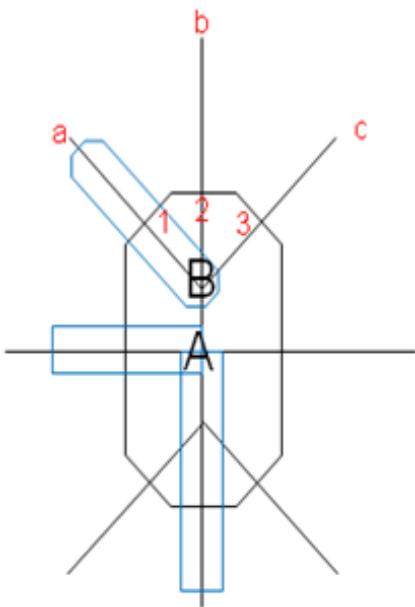
Pillar Bump



`fcroute` supports horizontal and vertical pillar bumps and 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.

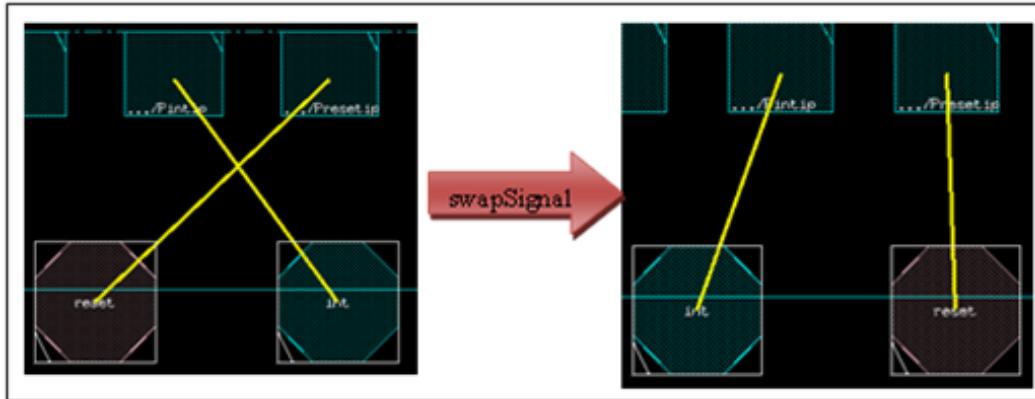
## Viewing Flip Chip Flightlines

In flip chip designs, flightlines are used extensively to interact with the design. Flip chip flightlines are different from the normal blue flightlines in EDI System and can be displayed using [`viewBumpConnection`](#). `viewBumpConnection` provides the following features to make it easy for you to use flightlines:

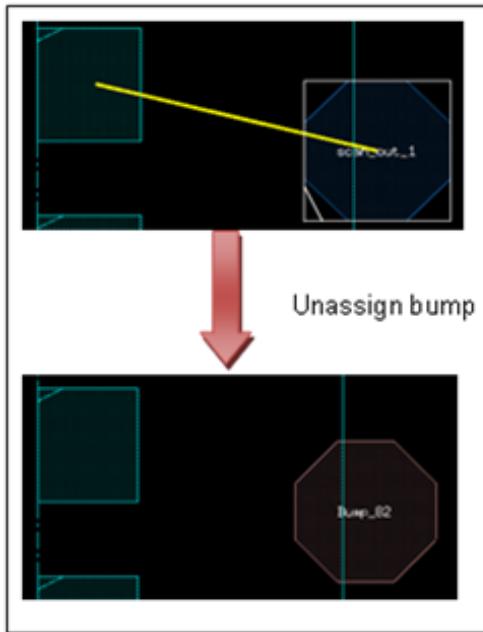
### Automatic Redraw Feature

`viewBumpConnection` redraws flightlines automatically after the following bump manipulation actions:

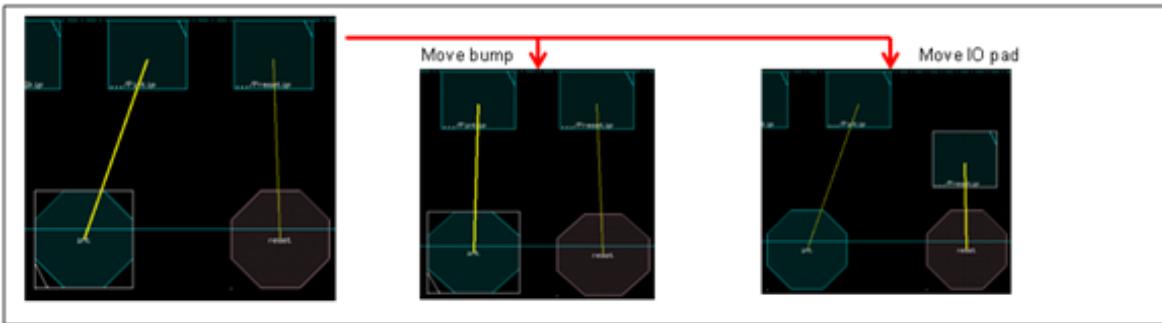
- Bump assignments are swapped using [swapSignal](#): Flightlines of the selected bumps are swapped to reflect the manipulation.



- 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.

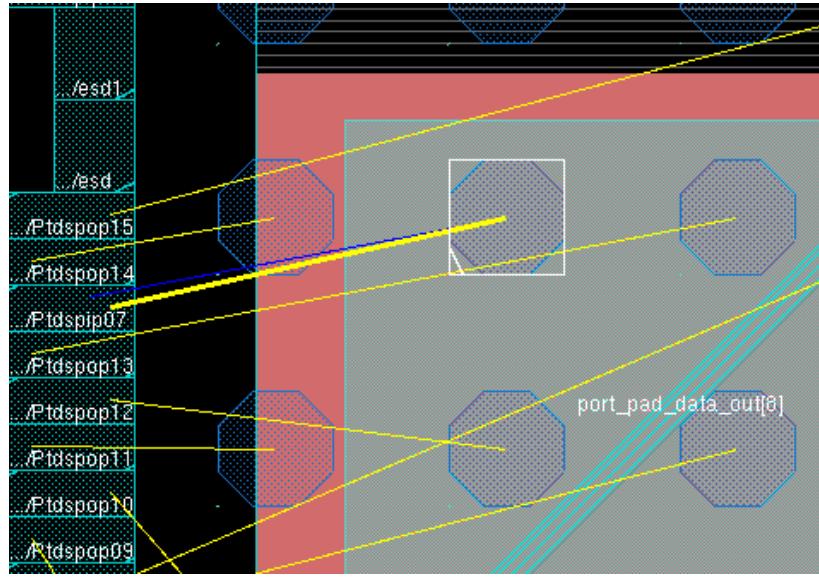


## Selection-Based Highlighting

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.

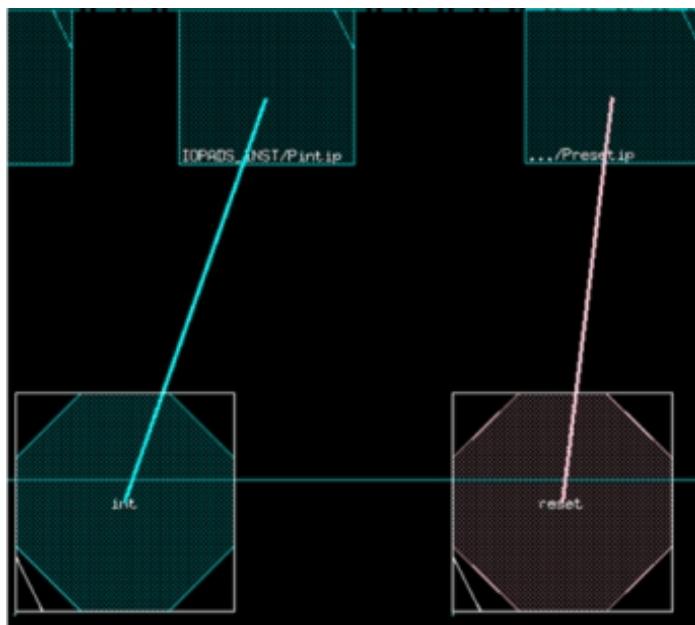


## Colored Flightlines

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:
- Define bump color settings in a bump color map file using the following format :  
*net\_name color\_name*
- Example:
- ```
int cyan
reset pink
```
- Load the bump color file using the [`ciopLoadBumpColorMapFile`](#) command.
  - 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.

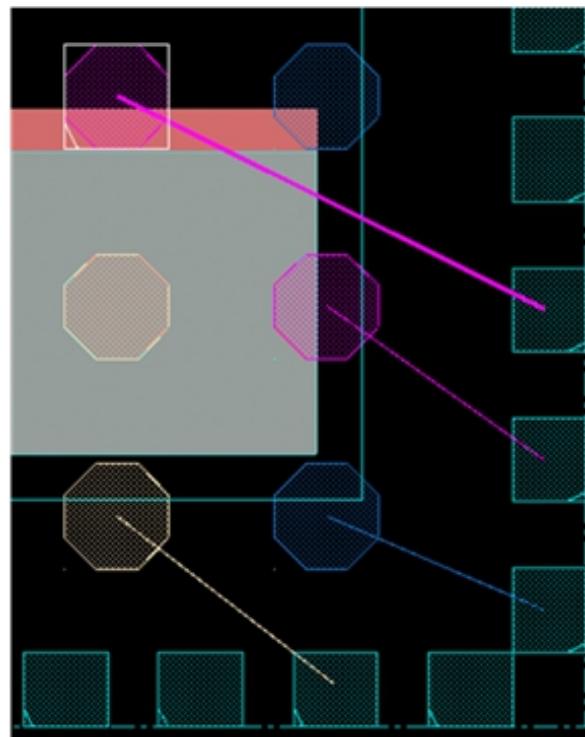
## Object-Specific Flightlines

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:

**DIFFPAIR**

```
port_pad_data_in[15]  
port_pad_data_in[13]
```

**END 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 PAIR constraint and display the connection between bump and IO pad for power/ground net.

If you want flip chip flightlines to honor the PAIR constraint:

1. Specify the PAIR constraint in the flip chip constraint file using the following syntax:  
`PAIR`

```
net_name pad_name_list bump_name_list
END 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 PAIR constraint, `viewBumpConnection` displays the connection specified by the 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 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 the PAIR constraint, `viewBumpConnection` auto-pairs them and displays the connections for VDD .
- If all of the bumps have only the PAIR constraint, `viewBumpConnection` displays the connections specified by the 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 PAIR constraint as in the following example:
  - a. Bump\_1: Does not have either the port number property or the PAIR constraint
  - b. Bump\_2: Only has a PAIR constraint:

```
PAIR
VDD pad_2 Bump_2
END PAIR
```
  - c. Bump\_3: Only has the port number property

d. Bump\_4: Has both the PAIR constraint and the port number property:

```
PAIR
VDD pad_4 Bump_4
END PAIR
```

e. Bump\_5 and Bump\_6: Only have a PAIR constraint:

```
PAIR
VDD pad_1 pad_5 pad_6 Bump_5 Bump_6
END PAIR
```

f. Bump\_7 and Bump\_8: Have a PAIR constraint as below and Bump\_8 also has the port number property:

```
PAIR
VDD pad_7 pad_8 Bump_7 Bump_8
END 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 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 PAIR constraint and does not display the connection of Bump\_7. This means if one or more bumps in a PAIR constraint have the port number property, the other bumps without port number property in this PAIR constraint are ignored and their connections are not displayed.

Note that the PAIR constraint does not support the following scenarios and will treat the last pairing as the available one.

#### **Scenario #1**

PAIR

```
VDD pad_1 Bump_1
```

END PAIR

PAIR

VDD pad\_1 Bump\_2 # This constraint works

END PAIR

If you want to pair pad\_1 to Bump\_1 and Bump\_2, use the following syntax:

PAIR

VDD pad\_1 Bump\_1 Bump\_2

END PAIR

### **Scenario #2**

PAIR

VDD pad\_1 Bump\_1

END PAIR

PAIR

VDD pad\_2 Bump\_1 # This constraint works

END PAIR

If you want to pair pad\_1 and pad\_2 to Bump\_1, use the following syntax:

PAIR

VDD pad\_1 pad\_2 Bump\_1

END PAIR

## Point-To-Point Routing

The Point-To-Point routing in flip chip enables routing between any two DEF SPECIALNET objects such as a bump and an I/O pad pin, a wire and a bump, or a bump and a stripe. The point-to-point router can point any location in the chip area.

Use the point-to-point router any time on special nets, especially after you run [fcroute](#) and you find an area where routing is not complete or an area which contains a problem route. In such cases, delete the problem route and reroute using the point-to-point router.

To perform point-to-point routing:

1. From the EDI System user interface, select *Tools -> Flip Chip -> Place & Route -> Route Point to Point*.

or

In the tools area, click the *routePoint2Point*  icon, and press the *F3* key.

2. In the [Point-To-Point](#) form, specify the minimum *width* .
3. Select 2 points in the design, an I/O pad pin and a bump (or a wire). View the routing that occurs between the 2 selected points.

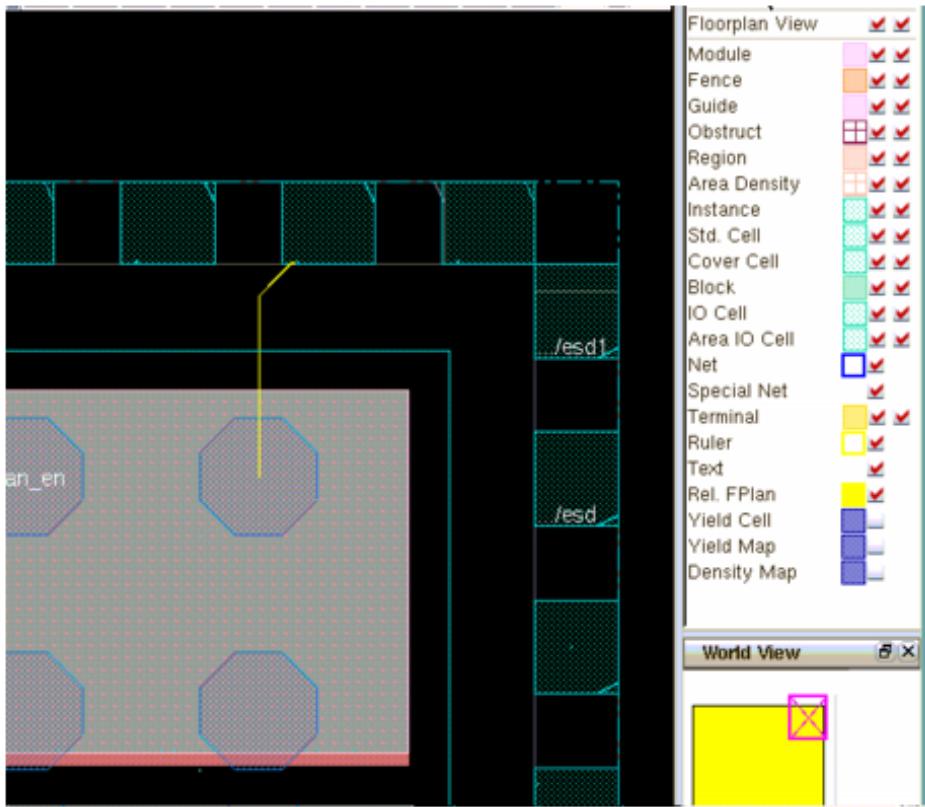
To view the point-to-point routing, ensure that you are in the physical view in EDI System. If the point-to-point route is not complete, check the encounter.log file or check for any error message on the screen.

The point-to-point router connects any two objects defined in the *Point-To-Point* form only. The router automatically selects the net name when you point the two objects -- bump and I/O pad.

Alternatively, you can run the [routePointToPoint](#) command to perform point-to-point routing.

The following example displays the results of the `routePointToPoint` command:

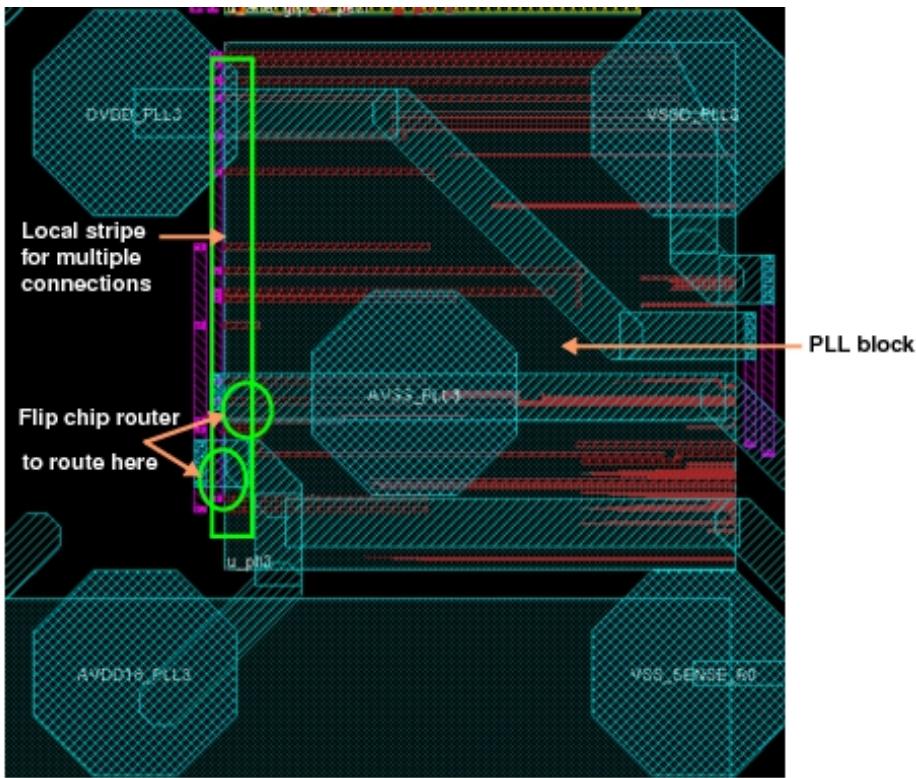
```
routePointToPoint -routeLayer M8:M8 -width 0.44 -spacing 0.46 -routeStyle
doubleBend -pin [IOPADS_INST/Ptdspip01 PAD (944.4765 1013.278)] -pin
(Bump_81_B_8 port_pad_data_in[1] (933.832 917.4755))
```



## Virtual Connection Area for RDL Routing

If your design has some IO pins on macros (PLL and ESD cells) that are not typical pins for flip chip routing, the flip chip router might have difficulty in handling these pins. However, it is very challenging to enhance the flip chip router to make connections automatically on to these pins because the pin shapes may vary or simply not be suitable for redistribution layer (RDL) routing. In such cases, you have to handle these pins manually.

As shown in the figure, the power and ground pins of the Phase Lock Loop (PLL) need to be routed to bumps. However, because of the special structure of the pin shapes with multiple ports, the flip chip router has difficulty in handling this case. In such a case, you can create some local stripes for the block and make connections to the ports one by one. The flip chip router can then route to the specified virtual areas on the stripe.



You can define virtual connection areas on an instance using the [createVirtualConnArea](#) command. This command adds the `virtual_conn_prop` property on to an instance with all necessary information. The `virtual_conn_prop` property has the following format:

```
io_pin_name :loc_x :loc_y :size :metal_layer
```

Here, `io_pin_name` is the pin name on the instance that the virtual connection area represents. The virtual area's location is defined by the `loc_x` and `loc_y` values in the property. These values specify the relative location of the area with respect to the lower left point of the master instance. The virtual area is a square shape so that the direction from which the flip chip router accesses the area does not matter. The virtual area appears only on the metal layer defined by `metal_layer`. The area is considered as an obstruction on the metal layer during routing of other nets as a normal pin geometry.

For an instance, one pin can only have one virtual connection area. Once a virtual connection area is defined for a pin, the real geometry of the pin is not routed any more. This means the virtual connection area is the new target that replaces the real pin. The flip chip router routes to the virtual connection area and drops vias accordingly if the area is not on the same metal layer as the routing layer.

You can use the [editVirtualConnArea](#) command to modify a virtual connection area on an instance.

All related information of the property can be edited including the relative locations, size and metal layer of the virtual connection area.

Use the [deleteVirtualConnArea](#) command to delete a virtual connection area on a specific instance.

Use the [writeFlipChipProperty](#) command to save the virtual connection area properties in the bump and IO pad property file, which is a manually editable text. Then, you can use the [readFlipChipProperty](#) command to load the bump and IO pad property file. After the file is loaded, the virtual connection area property is added to IO pads and can be displayed and used for flip chip flow. The [readFlipChipProperty](#) command prints error message if the following errors occur:

- Type error of property name
- Instance/pin not found
- Metal layer not found
- Size is zero or minus

The [writeFlipChipProperty](#) and [readFlipChipProperty](#) commands enable you to use the virtual connection area properties for instance pins in the flip chip flow.

Virtual connection areas are visible in the main window as blank boxes with pin names. The areas are displayed according to their relative location to the instance and metal layer. To view virtual connection areas in the main window, open the *View-Only* tab of the Color Preferences form and ensure that the *Virtual Connection Area* check box is selected.

**Note:** This release supports only creation, modification, and deletion of virtual connection areas. Other flip chip functions, such as bump assignment and RDL routing, do not currently support the virtual connection area feature.

## Port Numbering Feature for Power Nets

EDI System includes a port numbering feature to support multiple bump to multiple pad routing for power nets. It provides an easy way to determine which port is available for a bump. In addition, it provides mechanisms for adding, editing, and deleting the properties easily.

EDI System provides the following commands to implement port numbering:

- [findPinPortNumber](#) : This command enables you to determine easily which port is available for a bump. The command has the following functions:
  - Check if the IO pad has a standard LEF definition.
  - Suggest the number of a port that is suitable for flip chip routing on the specified instance based on the pin

name or nets in a specified area.

- Filter ports by layer and geometry. Only ports that match the specified layer or geometry range are returned by `findPinPortNumber`.
- Highlight bumps on the nets so that you can easily assess which bump is usable for property manipulation.

Standard LEF definition for port numbering can be illustrated as follows:

```

MACRO PVDD1DGZ
    #CLASS PAD POWER ;
    CLASS PAD AREAIO ;
    FOREIGN PVDD1DGZ 0.000 0.000 ;
    ORIGIN 0.000 0.000 ;
    #SIZE 40.000 BY 235.000 ;
    SIZE 40.000 BY 35.280 ;
    SYMMETRY x y r90 ;
    #SITE pad ;
    SITE IO1 ;
    PIN VDD
        DIRECTION OUTPUT ;
        USE POWER ;
    PORT
        CLASS BUMP ;
        LAYER METALS ;
        RECT 5.0 15.0 15.0 35.0 ;
    END
    PORT
        CLASS BUMP ;
        LAYER METALS ;
        RECT 25.0 0.0 35.0 10.0 ;
    END
    PORT
        CLASS BUMP ;
        LAYER METAL7 ;
        RECT 25 15 35 35 ;
    END
    END VDD
END PVDD1DGZ

```

Each geometry for RDL routing is separated with CLASS BUMP defined for the port

If the LEF is not defined as the standard, the `findPinPortNumber` command prints an error message such as:

```
**ERROR: (ENCSIP-6052): Cell pllclk has port defined as CLASS BUMP but pin gnd! has no CLASS BUMP ports
```

Pin gnd! of instance DTMF\_INST/PLLCLK\_INST is connected to net VSS To use the port numbering feature in flipchip flow, please separate the geometries to be used for RDL routing and add keyword CLASS BUMP to explicitly claim that the geometry is for RDL routing.

- [addBumpConnectTargetConstraint](#) : This command enables you to add a string property in one or multiple bumps.
- [editBumpConnectTargetConstraint](#) : This command enables you to edit the property values on bumps.

- [deleteBumpConnectTargetConstraint](#) : This command enables you to delete existing properties on specified, selected, or all bumps.

You can save properties on bumps using the `writeFlipChipProperty` command. The `bump_connect_target` property has the following format:

*instance\_name:pin\_name:port\_num*

You can restore the properties on bumps by using the `readFlipChipProperty` command to load the bump and IO pad property file. When the file is loaded, the bump connection target property is added to bumps and can be displayed and used for flip chip flow.

## Multi-PG Pads to Multi Bumps Assignment with a Controlled Ratio

By default, the tool assigns one power and ground (PG) pad to one bump in the same way as it assigns one signal pad to one bump. However, unlike signal pads, PG pads usually share the same PG nets with each other. EDI System now provides a way to assign multiple PG (multi-PG) pads on the same net to multiple bumps as per a ratio that you define. Different PG nets can have different ratios. This enhancement not only saves a lot of bump resource, it also requires less manual effort for bump assignment.

If you want multiple PG (multi-PG) pads on the same net to be assigned automatically to multiple bumps, you can use the `-ratio` parameter of [assignBump](#) to define the ratio in which ports should be assigned to bumps. You can define the ratio based on the design and your requirements.

You can specify more than one ratio of PG ports on the same net to bumps using the following format:

`-ratio {port_num_1:bump_num_1 port_num_2:bump_num_2 ...}`

Here, the variables `port_num_#` and `bump_num_#` are positive integers.

Keep the following things in mind when using the `-ratio` parameter:

- The `port_num : bump_num` ratio should be larger than 1. If not, the tool issues a WARNING message and assigns one pad to one bump.
- The `port_num : bump_num` ratio should be in the simplest form. If not , the tool reduces the ratio to the simplest form and issues a WARNING message. For example, if the specified ratio is 4:2, the tool simplifies it to 2:1 for PG net to bump assignment.
- The `-ratio` parameter affects only PG net to bump assignment. Therefore, it must always be used along with either the `-pgnet` or `-exclude_pgnet` parameter.
  - When used with `-pgnet`, the `-ratio` parameter supports multiple ratios for different

nets.

- If `-ratio` specifies only one ratio but more than one PG net is specified using `-pgnet`, the given ratio is used for all specified nets and the tool issues a WARNING message.
- If the number of ratios specified is equal to the number of PG nets specified, the ratios are used in the order in which the PG nets are specified. This means the first net specified uses the first ratio, the second net uses the second ratio, and so on.

For example, suppose that the PG nets specified are VDD and VSS and the ratios specified are 2:1 and 3:1. As the number of ratios is same as the number of PG nets, the ratios are mapped as follows:

- VDD - 2:1 (First ratio specified)
- VSS - 3:1 (Second ratio specified)
- If the number of ratios specified is less than the number of PG nets, the last specified ratio is used for the rest of the PG nets and the tool issues a WARNING message.
  - VDD1 - 2:1 (First ratio specified)
  - VDD2 - 3:1 (Second ratio specified)
  - For the remaining nets, VSS1 and VSS2, the ratio 3:1 is used as it is the last ratio specified.
- If the number of ratios specified is more than the number of PG nets, the tool uses ratios in the order of specification. Extra ratios are ignored and the tool issues a WARNING message.
  - VDD - 2:1 (First ratio specified)
  - VSS - 3:1 (Second ratio specified)
  - The extra ratio 4:1 is ignored.
- When used with `-exclude_pgnet`, the `-ratio` parameter supports only one ratio. This is because the order of assigned nets for `-exclude_pgnet` is controlled by the tool so it is not reasonable to support multiple ratios with this usage.
- The `-ratio` parameter must be used with the `-useTargetProp` parameter. With this

usage, `assignBump` will automatically add bump connect target onto bumps to do pairing and `fcroute` also will honor bump connect target to finish routing.

- The `-ratio` parameter is compatible with all other `assignBump` parameters, except `-pginst`.
- When you specify the `-ratio` parameter, the tool calculates the number of pad/port groups and the maximum number of ports in each group as follows:

Assume the total number of ports with a net is `total_port`, and the ratio is `port_num:bump_num`.

- If `total_port` is an integer multiple of the ratio, then:

$$\text{Number of groups} = \text{total\_port} \div (\text{port\_num}/\text{bump\_num})$$

If `total_port` is not an integer multiple of the ratio, the number of groups is obtained by rounding off `total_port ÷(port_num/bump_num)` and then adding 1:

$$\text{Number of groups} = [\text{total\_port} \div (\text{port\_num}/\text{bump\_num})] + 1$$

- If `port_num` is an integer multiple of `bump_num`, then:

$$\text{Maximum number of ports in each group} = \text{port\_num} \div \text{bump\_num}$$

If `port_num` is not an integer multiple of `bump_num`, the maximum number of ports in each group is obtained by rounding off `port_num ÷ bump_num` and then adding 1 :

$$\text{Maximum number of ports in each group} = [\text{port\_num} \div \text{bump\_num}] + 1$$

For example, suppose that there are 5 pads with VDD and that the VDD pin of each pad has only one port. Now if you specify the ratio as 3:2, then:

$$\text{Number of groups} = [5 \div (3/2)] + 1 = [3.33] + 1 = 3 + 1 = 4$$

$$\text{Maximum number of ports in each group} = [3/2] + 1 = [1.5] + 1 = 1 + 1 = 2$$

The tool tries to provide the best assignment subject to the following conditions:

- The number of groups remains unchanged.
- In each group, the number of ports must not be more than the maximum number of ports.

## Assigning Multi-PG Pads to Bumps Using a Single Ratio

The syntax for assigning multi-PG pads to bumps using a single ratio is as follows:

```
assignBump -pgnet {net1 net2} -ratio x:y
```

where  $x$  is *port\_num* and  $y$  is *bump\_num*

When this command is run:

1. The tool first checks whether the  $x : y$  ratio is in the simplest form. If not, it reduces the ratio to the simplest form.
2. The tool calculates the number of pad/port groups and the maximum number of ports in each group. It then uses these values to control how bump assignment is done.  
**Note:** If the number of bumps is less than that needed by the ports as per the calculations, the tool issues an ERROR message.

### Example

Consider a design that has:

- Three pads with VDD. One of the pads has three ports of pin VDD while the other two have one port of pin VDD each.
- Two pads with VSS. One of the pads has two ports of pin VSS. The other pad has a single port of pin VSS.

The total number of ports with VDD in this design is 5 and the number of ports with VSS is 3.

Let's see how the tool processes the following command for this design:

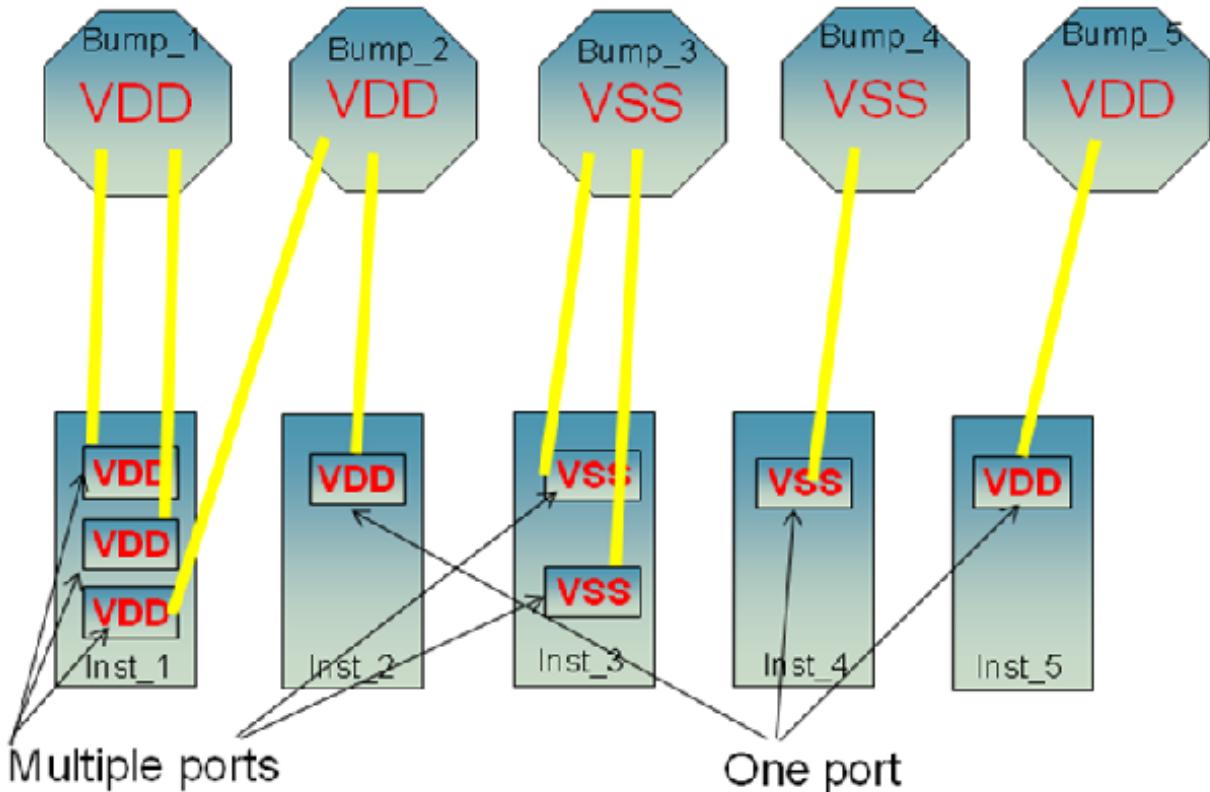
```
assignBump -pgnet VDD VSS -ratio {4:2}
```

1. The tool first simplifies the ratio to 2:1.
2. 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:  
$$\text{Number of groups} = [\text{total\_port} \div (\text{port\_num}/\text{bump\_num})] + 1 = [5 \div (2/1)] + 1 = [2.5] + 1 = 2 + 1 = 3$$
  - For VSS, as the total number of ports with VSS (3) is not an integer multiple of *port\_num* (2), therefore:  
$$\text{Number of groups} = [\text{total\_port} \div (\text{port\_num}/\text{bump\_num})] + 1 = [3 \div (2/1)] + 1 = [1.5] + 1 = 1 + 1 = 2$$
  - 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:

Maximum number of ports in each group for VDD and VSS =  $port\_num \div bump\_num = 2/1 = 2$

The multi-PG pad to bump assignment is depicted below.



## Assigning Multi-PG Pads to Bumps Using Multiple Ratios

The syntax for assigning multi-PG pads to bumps using multiple ratios is as follows:

```
assignBump -pgnet {net1 net2 net3} -ratio {x1:y1 x2:y2}
```

where *x* is *port\_num* and *y* is *bump\_num*

When this command is run:

1. The tool first checks whether the ratios *x1:y1* and *x2:y2* ratio are in the simplest form. If not, it reduces the ratios to their simplest forms.
2. The tool maps ratios to nets as follows:
  - *net1* - *x1:y1*.

- *net2 - x2:y2.*
  - *net3 - x2:y2.*
3. The tool calculates the number of pad/port groups for each net and the maximum number of ports in each group. It then uses these values to control how bump assignment is done.
- Note:** If the number of bumps is less than that needed by the ports as per the calculations, the tool issues an ERROR message.

### Example

Consider a design that has:

- Two pads with VDD. One of the pads has two ports of pin VDD while the other has a single port of pin VDD.
- Two pads with VSS. One of the pads has two ports of pin VSS. The other pad has a single port of pin VSS.

This means there are, in total, three ports with VDD and three with VSS.

Let's see how the tool processes the following command for this design:

```
assignBump -pgnet VDD VSS -ratio {3:2 2:1}
```

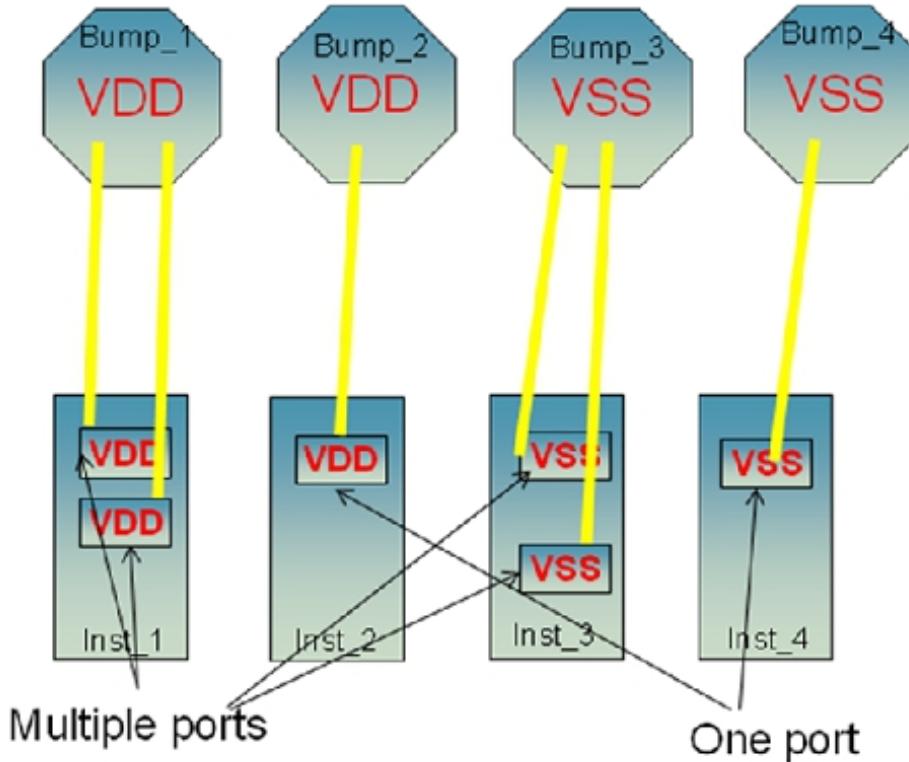
1. The tool confirms that both ratios are in their simplest forms.
2. The tool maps ratios to nets as follows:
  - VDD - 3:2
  - VSS - 2:1
3. The tool calculates the number of pad/port groups and maximum number of ports in each group as follows:
  - For VDD:
    - As the total number of ports with VDD (3) is an integer multiple of *port\_num* (3) in the ratio, therefore:  
$$\text{Number of groups} = \text{total\_port} \div (\text{port\_num}/\text{bump\_num}) = 3 \div (3/2) = 2$$
    - As in the ratio for VDD, *port\_num* (3) is not an integer multiple of *bump\_num* (2), therefore:  
$$\begin{aligned}\text{Maximum number of ports in each group} &= [\text{port\_num} \div \text{bump\_num}] + 1 \\ &= [3/2] + 1 = [1.5] + 1 = 1 + 1 = 2\end{aligned}$$
  - For VSS:
    - As the total number of ports with VSS (3) is not an integer multiple

of *port\_num* (2), therefore:

$$\begin{aligned}\text{Number of groups} &= [\text{total\_port} \div (\text{port\_num}/\text{bump\_num}) ]+1 \\ &= [3 \div (2/1)]+1 = [1.5] + 1 = 1 + 1 = 2\end{aligned}$$

- As in the ratio for VDD, *port\_num* (2) is an integer multiple of *bump\_num* (1), therefore:  
Maximum number of ports in each group =  $\text{port\_num} \div \text{bump\_num} = 2/1 = 2$

The multi-PG pad to bump assignment is depicted below.



## Distributed Co-design

Distributed co-design is a flow in which the package design and IC design are done in a distributed manner, and the package and IC design teams share data through text file exchange. The data shared includes information about die size, I/O pad ring placement, pin and bump placement, and so on.

The following command in EDI enable reading the I/O and bump -- placement and assignment information from SiP layout into EDI System:

- [readIoUpdate](#)

The package balls in the package file dumped out by the SiP layout in XML format can be correctly displayed in EDI System 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 EDI System floorplan view using the [readPackage](#) command.

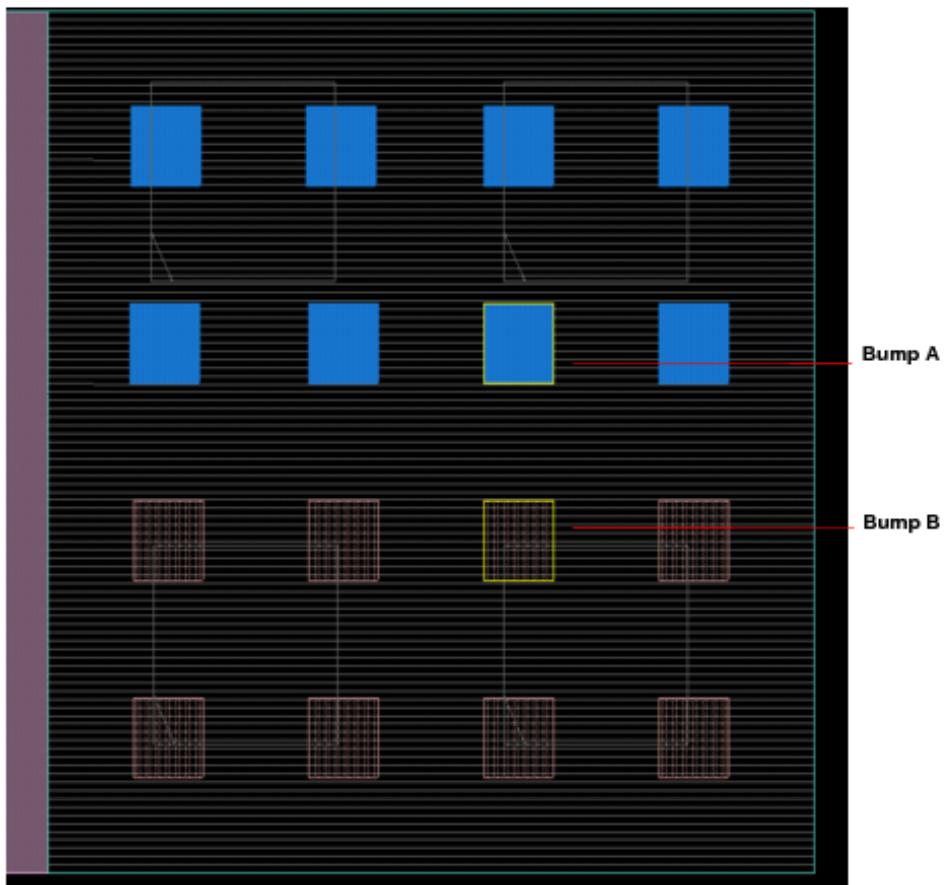
For more information, see the *Flip Chip Commands* chapter in the *EDI System Text Command Reference*.

## Swapping Signals

Signal swapping allows you to swap signals between bumps. Signals must be assigned to either one or both of the bumps to be swapped.

1. Click on the two bumps for the signals you want to swap.

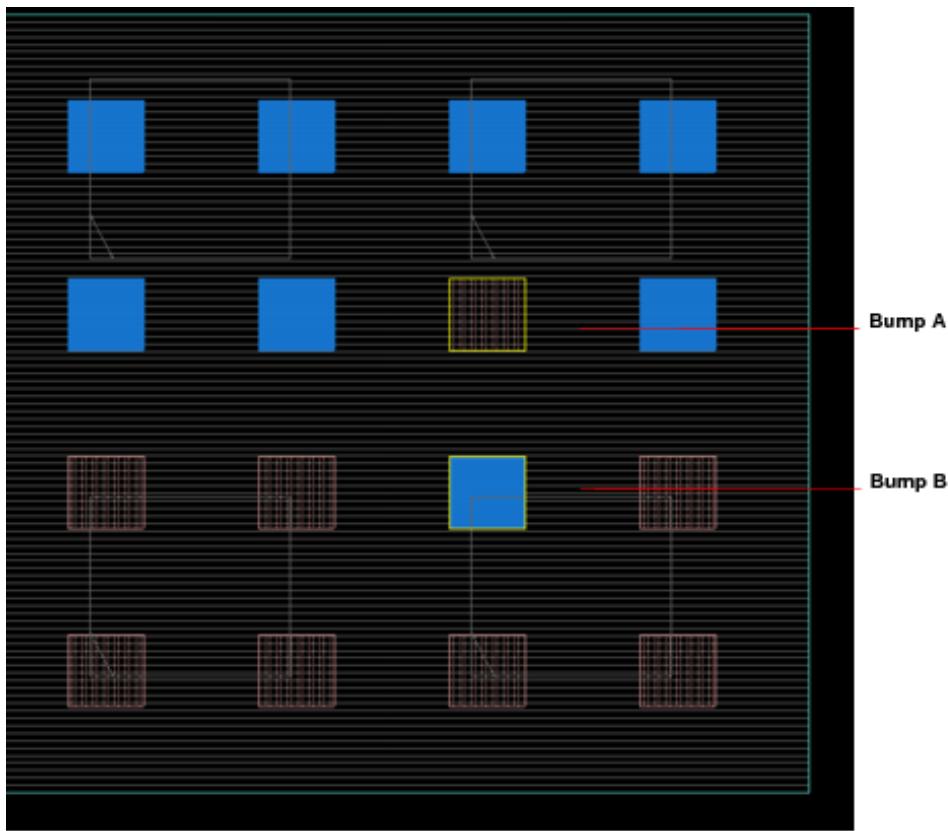
The following figure shows two highlighted bumps with signals to be swapped (bumps A and B).



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

2. Select *Tools - Flip Chip - Swap Signal*.

The following figure shows the signals after swapping.



## Creating Constraints for Flip Chip Routing

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

### Specify Routing Nets

#### Syntax

NETS

  WIDTHRANGE *value*

  WIDTHSTEP *value*

  SPACING *value*

  PINSPACING *dbUnitSpacing*

```
<nets>

END NETS

Example

NETS

    WIDTHRANGE 5:10

    WIDTHSTEP 1

    SPACING 0.1

    PINSPACING 0.2

    out[10] out[11] out[14] out[19] out[8] out[9]

    out[15] out[16] out[17] out[18] resetn

    clk out[12] out[12] out[13] out[15] out[3] out[3]

END NETS
```

## Define Differential Pairs

### Syntax

```
DIFFPAIR

    THRESHOLD value

    <2 nets>

END DIFFPAIR
```

Here, THRESHOLD specifies the value for the difference of the wire length of the two differential pair nets. The default value is 0.2. The threshold is calculated by using the following formula:

Length of the longer net - Length of the shorter net

-----  
Length of the shorter net

For example, if the length of the constrained nets are 22 microns and 20 microns, the threshold would be calculated as  $(22 - 20) / 20 = 0.1$ , so you have to set THRESHOLD as 0.1 to overwrite the default value.

### Example

```
DIFFPAIR
```

```
    THRESHOLD 0.2
```

```
    port_pad_data_out[7]
```

```
    port_pad_data_out[8]
```

```
END DIFFPAIR
```

## Define Nets to Match Tolerance

### Syntax

```
MATCH
```

```
    TOLERANCE 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.

### Example

```
MATCH
```

```
    TOLERANCE 0.2
```

```
    tdigit[1] tdigit[2] tdigit[3]
```

```
END MATCH
```

## Define Splitting

## Syntax

SPLITSTYLE RIVER | MESH

SPLITWIDTH *value*

SPLITGAP *value*

SPLITKEEPTOTALWIDTH TRUE | FALSE

Here:

- SPLITSTYLE specifies the splitting style.
- SPLITWIDTH specifies the maximum width limit for each wire after the split. If you do not specify this keyword, the width of each split wire is the value in the LAYER statement of the LEF file. If LAYER rules are not specified, no splitting occurs.
- SPLITGAP specifies the minimum distance between split wire segments. If you do not specify this keyword, the distance between split wire segments is the default minimum spacing value that does not cause DRC violations .

## Example

MATCH

TOLERANCE 0.2

tdigit[1] tdigit[2] tdigit[3]

END MATCH

## Define a Shield Net

**Note:** Shielding is supported only in the fcroute AIO mode.

## Syntax

SHIELDING

SHIELDBUMP TRUE | FALSE

SHIELDWIDTH *value*

SHIELDGAP *value*

```
SHIELDSTYLE a | b | c

SHIELDNET netName

<nets>

END SHIELDING
```

Here:

- SHIELDBUMP specifies whether bumps are to be shielded. The default value is FALSE, which means bumps are not shielded. When set to TRUE, shields bump with the specified shield net. It only works in AIO mode and with Manhattan routing style.
- SHIELDWIDTH specifies the width of the Shield Net, measured in microns.
- SHIELDGAP specifies the distance in microns between the shield, which is the special net, and the shielded net, which is the signal net.
- SHIELDSTYLE specifies where you want the shield to be placed: Above, 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 specifies a special net (typically VSS) used to shield the net.
- <nets> specifies the shielded nets.

### Example

```
SHIELDING

SHIELDBUMP true

SHIELDWIDTH 0.4

SHIELDGAP 0.1

SHIELDLAYERS Above

SHIELDNET VSS

scan_out_2 port_pad_data_out[15]

END SHIELDING
```

## Route Multiple Nets with Different Widths

The following shows a constraint syntax that allows one `fcroute` command to route multiple routes with different widths.

```
fcroute -constraintFile file_name
```

### Example Constraints File

```
NETS
```

```
    WIDTH 24.0
    ROUTELAYERS 7:7
    SPACING 0.1
    ## Net Definition ##
    VDDPST           #apply two nets only
    VSSPST
```

```
END NETS
```

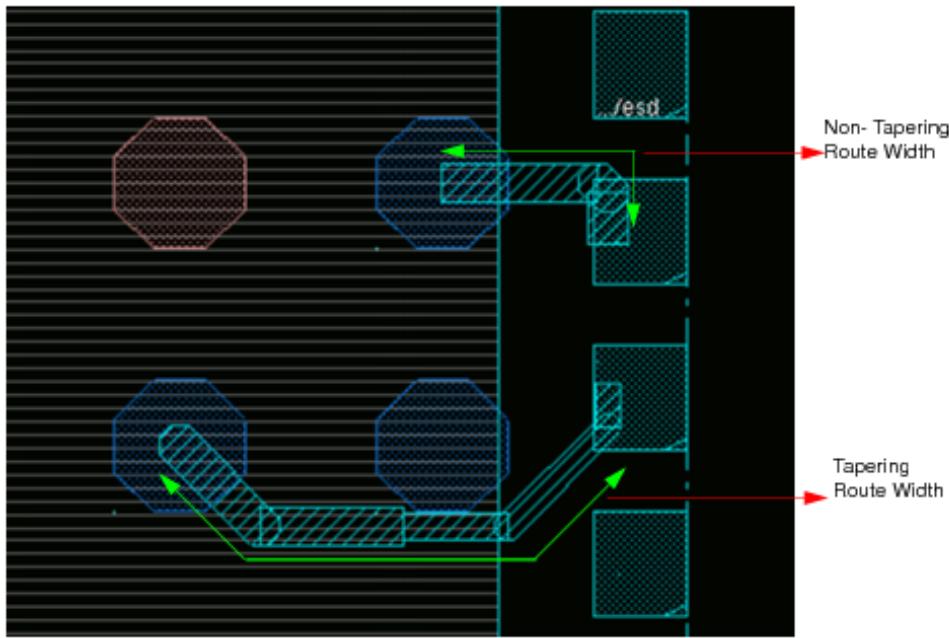
```
NETS
```

```
    WIDTH 20.0
    ROUTELAYERS 8:7
    SPACING 0.1
    ## Net Definition ##
    ~VDDPST          #negation - for all other nets
    ~VSSPST          #negation
```

```
END NETS
```

## Route Nets with Tapering Pin Widths

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 `-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
```

## Change Pin Access Direction

You can use the new `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 IO 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.

## Examples and Report Files

## Routing and Placement Constraints

The following listing is the constraint file that is used for both `fcroute` and `placePIO`.

For individual constraint descriptions, see [Route Flip Chip - Advanced -Routing Constraints](#) page in the *Tools Menu* chapter of the *EDI System Menu Reference*.

```
*****
```

```
Routing constraints: fcroute -designStyle aio | pio
```

```
*****
```

```
#One constraint file is used for fcroute and placePIO
```

```
VERSION 2
```

```
WIDTH 10          ; global constraint
SHIELDBUMP       ; global constraint
NETS
out[10]
END NETS
DIFFPAIR
WIDTH 20          ; can't accept, because of global constraint
MAXLENGTH 1000    ; can't accept, because of global constraint
SHIELDWIDTH 0.5   ; local constraint
SHIELDLAYERS abc ; local constraint
out[111] out[114] SHIELDNET VDD
END DIFFPAIR
```

DIFFPAIR

```
    out[119] out[120]
```

```
END DIFFPAIR
```

SHIELDING

```
    WIDTH 20           ; can't accept, because of global constraint
    SHIELDWIDTH 0.5   ; local constraint
    SHIELDLAYERS abc  ; local constraint
```

```
    out[18] out[19] out[115] out[116] out[117] out[118] resetn
```

```
END SHIELDING
```

# DIFFPAIR and MATCH results maybe different in -designStyle aio | pio

DIFFPAIR

```
    out[10] out[11] SHIELDNET VDD
    out[14] out[19]
```

```
END DIFFPAIR
```

#SHIELDING only works with fcroute -designStyle aio

SHIELDING

```
    SHIELDNET VDD (width spacing)
```

```
    in1 net2
    out1 out2
```

```
END SHIELDING
```

#NETS only work with fcroute -designStyle pio

NETS

```
    WIDTH 24.0
    ROUTELAYERS 8:7
    SPACING 0.1
    VDDPT          # apply two nets only
    VSSPT
```

```
END NETS
```

NETS

```
WIDTH 20.0
ROUTELAYERS 8:7
SPACING 0.1
~VDDPT          # negation - for all other nets.
~VSSPT          # negation
```

END NETS

BUMPREGION

```
AREA 3942.0 3545.0 3903.0 -3979.0 -3868.0 -3932.0 3774.0 -3521.0
```

```
VDD*
```

```
END AREA
```

END BUMPREGION

```
*****
```

Placement constraints: Only used with placePIO command

```
*****
```

FIXNETPAD

```
net_name_list
```

```
END FIXNETPAD      ;(All the pads associated with given nets are fixed)
```

FIXPAD

```
pad_name_list
```

```
END FIXPAD        ;(All the pads in the list are fixed)
```

FIXNETPADSIDE {EAST WEST SOUTH NORTH}

```
net_name_list
```

END FIXNETPADSIDE

FIXPADSIDE {EAST WEST SOUTH NORTH}

pad\_name\_list

END FIXPADSIDE

GROUPNET

net\_name\_list

END GROUPNET

GROUP

pad\_name\_list

END GROUP

FIXBUMP

net1 net2 net3

END FIXBUMP

\*\*\*\*\*

Resistance constraints for all (MAXRES) and/or individual nets (RESTABLE)

used by placePIO command

\*\*\*\*\*

NETS

WIDTHRANGE

ROUTELAYERS

MAXRES <resistance(ohms)>

NET\_1

NET\_2

END NETS

RESTABLE

#<Netname> <resistance(ohms)>

NET\_1 0.1

NET\_2 0.2

END RESTABLE

## IO\_FILE Example

The following sample is an IO\_FILE file showing bumps, I/O rows, and I/O instances. Format definitions follow the sample.

```
BumpCell: BUMPCELL Rect 1 Layer 6 0.000 0.000 80.000 80.000
Bump: bumpAry_16_3_3 BUMPCELL 697.440 696.800 DI[1]
Bump: bumpAry_15_2_3 BUMPCELL 497.440 696.800 DO[1]
Bump: bumpAry_14_1_3 BUMPCELL 297.440 696.800 DO[0]
Bump: bumpAry_13_0_3 BUMPCELL 97.440 696.800 SO
Bump: bumpAry_12_3_2 BUMPCELL 697.440 496.800
Bump: bumpAry_11_2_2 BUMPCELL 497.440 496.800
Bump: bumpAry_10_1_2 BUMPCELL 297.440 496.800
Bump: bumpAry_9_0_2 BUMPCELL 97.440 496.800
Bump: bumpAry_8_3_1 BUMPCELL 697.440 296.800 DI[0]
Bump: bumpAry_7_2_1 BUMPCELL 497.440 296.800
Bump: bumpAry_6_1_1 BUMPCELL 297.440 296.800 SI
Bump: bumpAry_5_0_1 BUMPCELL 97.440 296.800
```

```
Bump: bumpAry_4_3_0 BUMPCELL 697.440 96.800 CLK
Bump: bumpAry_3_2_0 BUMPCELL 497.440 96.800
Bump: bumpAry_2_1_0 BUMPCELL 297.440 96.800 SM
Bump: bumpAry_1_0_0 BUMPCELL 97.440 96.800

IORow: IOROW_1 520.100 596.400 I01 R0 V 100.800 2
IORow: IOROW_2 520.100 126.000 I01 R0 V 100.800 2
IORow: IOROW_3 119.700 596.400 I01 R0 V 100.800 2
IORow: IOROW_4 119.700 126.000 I01 R0 V 100.800 2

IOInst: test_clk/clk/inbuf 520.100 126.000 R0 -fixed
IOInst: test_clk/test/smbuf 119.700 126.000 R0 -fixed
IOInst: test_clk/test/sibuf 119.700 226.800 R0 -fixed
IOInst: test_clk/test/sobuf 119.700 697.200 R0 -fixed
IOInst: ioall/io_A/inbuf_0/inbuf 520.100 226.800 R0 -fixed
IOInst: ioall/io_A/inbuf_1/inbuf 520.100 596.400 R0 -fixed
IOInst: ioall/io_B/outbuf_0/outbuf 119.700 596.400 R0 -fixed
IOInst: ioall/io_B/outbuf_1/outbuf 520.100 697.200 R0 -fixed
```

### **Format Definitions**

- I/O Rows:

```
IOROW: iorow_name x y site_name [orient] [ [H | V] step num]
```

|            |                                                                                        |
|------------|----------------------------------------------------------------------------------------|
| iorow_name | Specifies the row name.                                                                |
| x y        | Specifies the x and y coordinates, in microns, of the origin.                          |
| site_name  | Specifies the site name. This must be defined in the LEF file.                         |
| orient     | Specifies the row orientation.                                                         |
| H   V      | Specifies either a <b>H</b> orizontal or a <b>V</b> ertical row.                       |
| step       | Specifies the site width or height (depending on orientation), in microns, of the row. |
| num        | Specifies the number of sites in the row (multiply by step for row length).            |

- I/O instances:

`IOInst: inst_name [x y [orient] [-fixed]]`

|           |                                                               |
|-----------|---------------------------------------------------------------|
| inst_name | Specifies the instance name.                                  |
| x y       | Specifies the x and y coordinates, in microns, of the origin. |
| orient    | Specifies the instance orientation.                           |
| -fixed    | Sets the placement status to fixed.                           |

For more information, see the [addAIORow](#) command in the "Flip Chip Commands" chapter of the *EDI System Text Command Reference*.

## 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 fcroute 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 resistance violation. In this case, the Total resistance section reports the current resistance versus the expected resistance in Constraint:xx. |
| routed               | If the net is successfully routed without any resistance violation, the status is reported as routed.                                                                                                                                                                                                                                            |

- **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.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Layer#             | <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> |
| Layer# to be split | <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</pre>                                                                                                             |

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 | <p><b>Layer 2<sup>nd</sup>_RDL to be split</b></p> <p>Path: Width 25 Length 50 Resistance 0.8</p> <p>Total length: 250</p> <p>Total resistance: 3.3</p> <p>END VSS</p>                                                                                                                                                                                                                                                                          |
| Layer# tapering | <p>Indicates that the tapering 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 routed</pre> <p>Layer TOP_RDL <b>tapering</b></p> <p>Path: Width 25 Length 200<br/>Resistance 2.5</p> <p>Path: Width 15 Length 50 Resistance 1</p> <p>Total length: 250</p> <p>Total resistance: 3.5</p> <p>END VSS</p> |
| Layer# widthOpt | <p>Indicates that width optimization 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 routed</pre> <p>Layer TOP_RDL <b>widthOpt</b></p> <p>Path: Width 25 Length 250</p>                                                                                                                                |

```
Resistance 3
Total length: 250
Total resistance: 3
END VSS
```

## 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

| ECO Change                        | Impact on Routing                                  |
|-----------------------------------|----------------------------------------------------|
| 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

| ECO Change                                         | Impact on Routing                                    |
|----------------------------------------------------|------------------------------------------------------|
| 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                                                    | <p>Invokes <code>fcroute -eco</code> (Working Mode II)</p> <p><b>Note:</b> The rerouting should preserve undeleted routings and do reroute on deleted routing only.</p> |
| 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
```

# Using ART in Hierarchical Designs

---

- [Overview](#)
- [Types of Active Logic Views](#)
  - [Flat Top](#)
  - [Critical](#)
- [Creating an Active Logic View](#)
- [The FlexView Flow](#)
  - [Overview](#)
  - [FlexView Flow Optimization Modes](#)
  - [Hierarchical FlexView Flow](#)
  - [Sample Summary Report](#)
  - [Sample timeDesign Summary](#)
  - [Sample optDesign Summary](#)
- [The flexILM PreCTS Closure Flow](#)
- [Debugging Timing Inside and Across Partitions](#)

## 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:

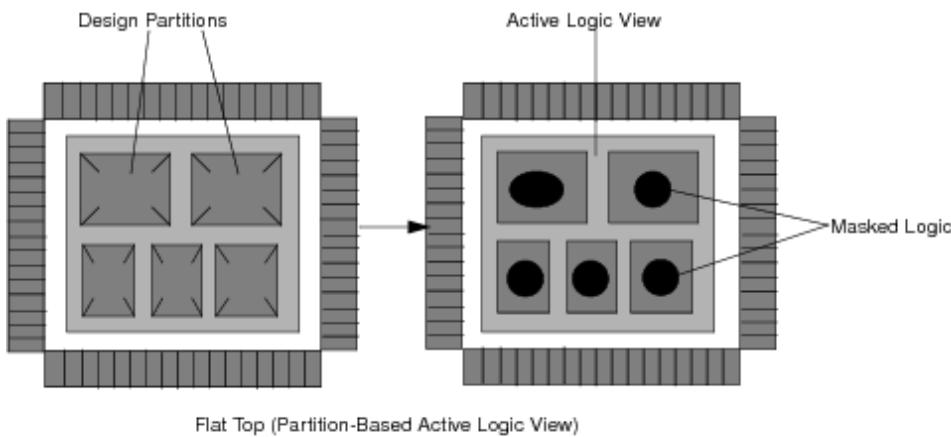
- Flat Top

- Critical

## Flat Top

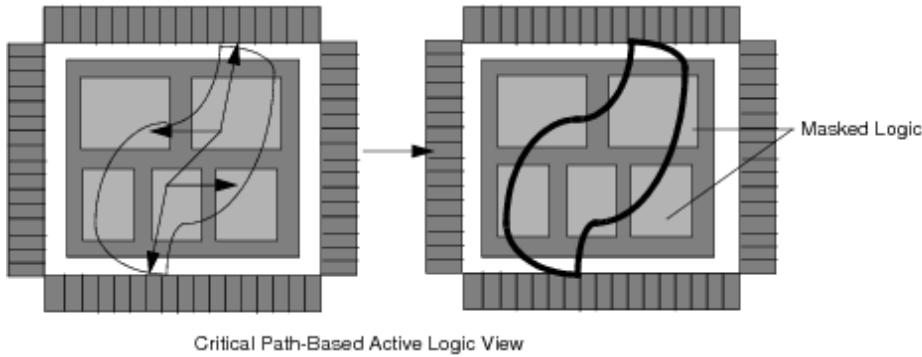
A flat top is a partition-based active logic view that activates the top-level paths and the interface path of partition blocks. The logic inside the partition blocks is excluded from the timing database.

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 logic in the design is masked.



## Creating an Active Logic View

To create an active logic view, load the entire chip as a design in the Encounter Digital Implementation System (EDI System) software, specify the partition, and then run the [createActiveLogicView](#) command with an appropriate option.

- i** 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 EDI System 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 FlexView Flow

- Overview
- FlexView Flow Optimization Modes
- Hierarchical FlexView Flow

### Overview

In the traditional hierarchical timing-closure flow, the partitions are optimized independently from the top level of a design. The top level just gets buffering and DRV fixing and the partitions are assembled after the timing is met. The timing optimization at the top level is done with the limitations of \*.lib or ILM models. The traditional hierarchical timing-closure flow can have unpredictable turnaround time.

The new FlexView flow assembles the top and each of the implemented partitions to perform full-chip level hold and setup optimization. In the FlexView flow, the timing based on both the physical data and the logical data. The physical data is used for extraction and the logical data is used to compute accurate timing. The FlexView flow eliminates limitations attached to the \*.lib and ILM models.

The FlexView flow supports nested partitions. It supports 2 level partitions with the second level being readOnly.

### FlexView Flow Optimization Modes

The following three optimization modes are supported in FlexView flow:

- Top-Only
- Top-and-Partition Interface Only
- Top and Full Partition

#### Top-Only

In the top-only optimization mode, the partitions are transparent for timing analysis. The optimization changes are allowed only at the top level of a design.

The top-only optimization mode is suited for designs ranging from five to 50 million instances depending upon the number of ILMs present in the design.

### **Top-and-Partition Interface Only**

In the top-and-partition interface only optimization mode, the partitions are transparent for timing analysis. The optimization changes are allowed at the top and interface logic of the allowed partitions. In this optimization, both the logical and physical hierarchy of a design is maintained.

The top-and-partition interface only optimization mode supports upto 10 million instances and upto 20 MMMC views in a design.

### **Top and Full Partition**

In the top-and-full partition optimization mode, the partitions are transparent for timing analysis. The optimization changes are allowed on top and inside any of the allowed partitions. In this optimization, both the logical and physical hierarchy of a design is honored.

The top and full partition optimization supports upto five million instances and 20 MMMC views in a design.

## **Hierarchical FlexView Flow**

Follow these steps to run the hierarchical FlexView flow:

1. Assemble the Top Data and the Block Data
2. [Verify Data Accuracy](#)
3. Define the FlexView Mode
4. [Optimize Data](#)
5. [Save the Hierarchical Data](#)
6. [Restore the Hierarchical Data](#)
7. [Verify the Optimized Data](#)

#### **Assemble the Top Data and the Block Data**

In this step, you assemble the design and specify partitions to be represented by ILMs.

To assemble a design, run:

```
assembleDesign  
-topDir top .enc.dat
```

```
-blockDir block1 .enc.dat
-blockDir block1 .enc.dat
-hierDir scratch_dir
```

where, `-topDir` is the top-level EDI database, `-blockDir` is the block-level EDI database ,and `-hierDir` first enables the FlexView flow. FlexView flow requires a scratch space to save some intermediate data and using the `-hierDir` parameter, a directory is specified.

To specify the partitions to be represented by ILMs, run:

```
assembleDesign -topDir top .enc.dat
-blockData block1 .def block1 .v
-blockData block2 .def block2 .v
-hierDir scratch_directory
specifyIlm -cell cell_name -dir ilm_data_dir
```

#### Verify Data Accuracy

Use `assembleDesign -mmmcFile` to load full chip MMMC constraints.

#### Define the FlexView Mode

Specify the FlexView mode by running the following command:

```
setModuleView -hinst partition1_inst -type <readOnly|interface|all>
setModuleView -topReadOnly [true|false]
```

To verify the FlexView mode, run:

```
getModuleView
```

FlexView flow supports the following different FlexView mode settings for different partitions:

```
setModuleView -hinst partition1_inst -type interface
setModuleView -hinst partition2_inst -type readOnly
setModuleView -hinst partition3_inst -type interface
setModuleView -hinst partition4_inst -type all
```

**Note:** FlexView flow supports deleting and adding filler cells as long as the `setFillerMode` is specified.

**Note:** Partitions are submitted for ECO routing based on the size (number of instances/nets). The bigger the size the higher is the priority to start ECO routing. You can set the priority of the order of ECO routing using the `eco_route_weight`/`top_eco_route_weight` parameters of the `setModuleView` command.

#### Optimize Data

Perform setup, hold optimization (including SI) by running the following commands:

```
optDesign -postRoute  
optDesign -postRoute -hold
```

The optimization and ECO routing can be performed separately using the following commands:

```
optDesign -postRoute [-hold] -noECORoute  
saveDesign output_directory -hier # This is an optional step  
ecoRoute -handlePartition
```

#### Save the Hierarchical Data

Save the hierarchical data by running the following command:

```
saveDesign output_directory -hier
```

The output directory contains the hierarchical database for top level and each of the partitions. For example, the output directory will contain the following files: Top.enc.dat, partition1.enc.dat, partition2.enc.dat, partition3.enc.dat, and partition4.enc.dat.

#### Restore the Hierarchical Data

Use the following command to restore the hierarchical data:

```
restoreDesign output_directory top_cell -hier
```

#### Verify the Optimized Data

To verify the optimized physical data, use the [checkHierRoute](#), [checkPlace](#), [verifyGeometry](#), and [verifyConnectivity](#) commands.

To check the final timing, run:

```
timeDesign -postRoute
```

## Sample Summary Report

The initial summary during optimization in FlexView flow is always reported using ART. As ART is used, this might make WNS, TNS, violating paths, and even the total paths less than the actual number.

The default behavior of `timeDesign` and `optDesign` final summary is not using ART technology. It will report the full design QoR with all the paths included.

## Sample timeDesign Summary

---

timeDesign Summary

---

| Setup mode       | all     | reg2reg | in2reg | reg2out | in2out | clkgate |  |
|------------------|---------|---------|--------|---------|--------|---------|--|
| WNS (ns):        | -2.207  | -2.207  | 0.515  | -1.083  | 61.544 | N/A     |  |
| TNS (ns):        | -2475.5 | -2464.9 | 0.000  | -10.635 | 0.000  | N/A     |  |
| Violating Paths: | 7188    | 7168    | 0      | 20      | 0      | N/A     |  |
| All Paths:       | 35747   | 35432   | 349    | 191     | 1      | N/A     |  |

## Sample optDesign Summary

| Initial Summary (with ART technology) |         |         |        |         |        |         |  |
|---------------------------------------|---------|---------|--------|---------|--------|---------|--|
| Setup mode                            | all     | reg2reg | in2reg | reg2out | in2out | clkgate |  |
| WNS (ns):                             | -0.740  |         |        |         |        |         |  |
| TNS (ns):                             | -76.752 |         |        |         |        |         |  |
| Violating Paths:                      | 250     |         |        |         |        |         |  |
| All Paths:                            | 8874    |         |        |         |        |         |  |

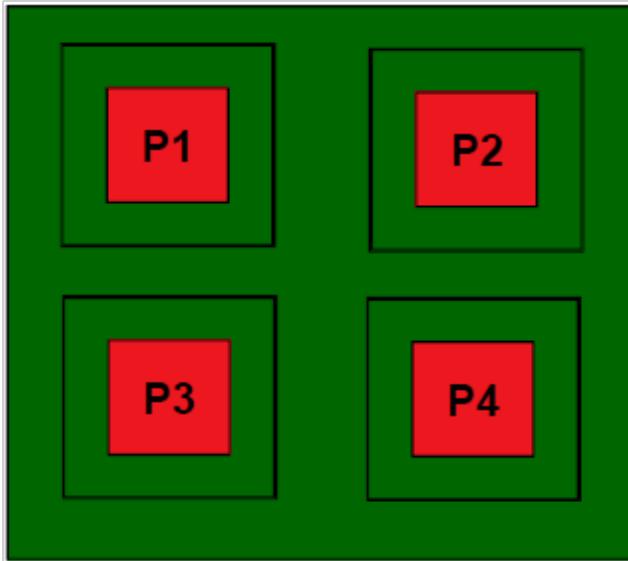
  

| optDesign Final Summary |         |         |        |         |        |         |  |
|-------------------------|---------|---------|--------|---------|--------|---------|--|
| Setup mode              | all     | reg2reg | in2reg | reg2out | in2out | clkgate |  |
| WNS (ns):               | -0.903  | -0.903  | 0.519  | -0.736  | 61.544 | N/A     |  |
| TNS (ns):               | -1146.6 | -1142.8 | 0.000  | -3.780  | 0.000  | N/A     |  |
| Violating Paths:        | 4983    | 4967    | 0      | 16      | 0      | N/A     |  |
| All Paths:              | 35747   | 35432   | 349    | 191     | 1      | N/A     |  |

In the above report, there are a total of 35747 paths of which 7188 are violating paths with a TNS of -2475.7 and a WNS of -2.2ns. The partitions in this case are selected to be read-only. Hence all the violating paths inside the partition will be filtered by ART reducing the WNS to 0.74ns with TNS of -

76ns, violating paths of 250 and the total paths to 8674. However the final summary is without using ART.

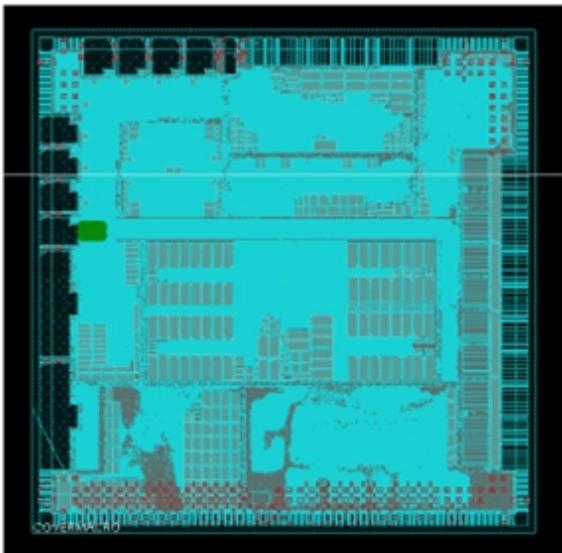
## The flexILM PreCTS Closure Flow



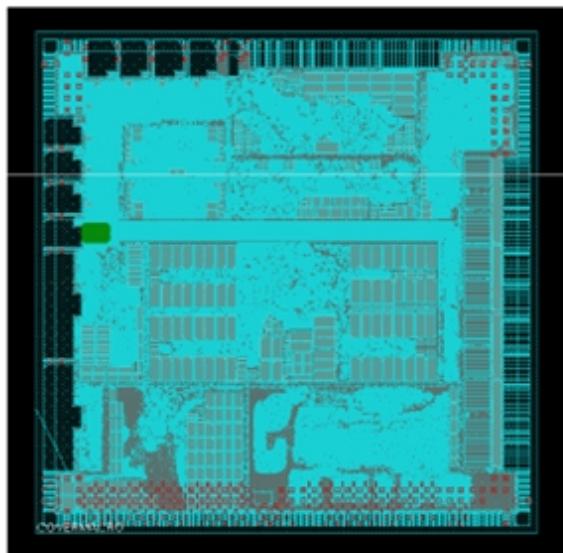
FlexILM creates reduced Netlist by trimming logic on internal path and keeping logic on interface path, and saves Netlist and DEF. In terms of Netlist and Placement ECO, FlexILM commits change on interface paths to original block data via the ECO process. It also contributes to memory reduction in timing graph and physical data.

### Preliminary Results on Large Design: Floorplan

Full Physical Data



FlexILM



#### 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.

You can use FlexILM Flow by:

FlexILM model generation

`createInterfaceLogic` can generate FlexILM to be used for either preCTS or postCTS optimization,

including converting reduced instances as placement blockages.

```
createInterfaceLogic -useType {flexILM}\n-optStage {preCTS | postCTS}
```

#### Specifying and committing FlexILM

User can easily specify FlexILM using `set_module_model` and assemble FlexILM using `commit_module_model`.

```
-cell <module_name> -dir <model_data_dir>\nset_module_model -type {flexILM}\ncommit_module_model
```

#### Distributed ECO process

`update_partition` command automatically saves post-optimization FlexILM data, generates ECO script for each FlexILM block, and launches distributed jobs to commit ECO change to the original block data.

```
update_partition {-flexILmECO | -genECOScript} \n-goldenDir <original partition data directory> \n-flexILmDir <directory to save postOpt FlexILM data>
```

## Debugging Timing Inside and Across Partitions

In the early stage analysis of hierarchical designs (during prototype and floor planning), the [check\\_partition\\_timing](#) command can debug timing inside and across partitions of very large designs (more than 100 million+ instances). This command divides tasks into smaller jobs suitable for inexpensive machines, runs analysis on those machines using distribution and presents results back to user in HTML format.

The `check_partition_timing` command works on full chip with top-level constraints and generates setup timing reports for top level and list of partitions in every view. Following parameters and timing paths are reported collectively in these reports.

- Total negative slack per partition per view
- Worst negative slack per partition per view
- Summarized timing paths per partition per view
- Detailed timing paths per partition per view

# Using Interface Logic Models in Hierarchical Designs

---

- [Overview](#)
- [Creating ILMs](#)
  - [Example ILM Creation](#)
  - [Preserving Selected Instances in ILMs](#)
  - [Creating ILMs for Shared Modules](#)
  - [Creating ILMs Without Using Encounter Database](#)
- [Specifying ILM Directories at the Top Level](#)
  - [Example Top-Level Implementation Flow with ILMs](#)
- [ILMs Supported in MMMC Analysis](#)
- [ILMs Supported in SI](#)
  - [SI Model Generation](#)
- [Merged ILM Model](#)
- [Interactive Use of ILMs](#)
- [Handling Interactive Constraints](#)

## Overview

Models are compact and accurate representations of timing characteristics of a block. An Interface Logic Model (ILM) is a structural representation of a block, specifically a subset of the block's structure including instances along the I/O timing paths, clock-tree instances, and instances or net coupling affecting the signal integrity (SI) on I/O timing paths.

Instead of using a blackbox at the top level, you create an ILM at the block level and use it as you would use a blackbox.

The advantages of using ILMs are as follows:

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

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

## Creating ILMs

In the hierarchical design flow, you create a detailed block-level implementation of a block, then specify the [createInterfaceLogic](#) command to create an ILM for the block. This command creates the specified directory containing ILM files.

You can also create ILMs for blocks that are in an intermediate stage of design, then use the data at the top level of the design for preliminary timing optimization.

**i** An ILM created for an incomplete block is not as accurate as an ILM created for a complete block. Always use ILMs for complete blocks to complete the top-level design.

The software generates ILM data for CTS, signal integrity, and other design stages (pre-CTS, post-CTS, post-route)

- ILM data for pre-CTS, CTS, post-CTS, and post-route
  - The model contains the netlist of the circuitry leading from the I/O ports to interface sequential instances (that is, registers or latches), and from interface sequential instances to I/O ports. The clock tree leading to the interface registers is preserved.

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 register-to-register paths, if internal logic is not part of the interface path.
- Internal paths (if `-noInterClockPath` is used): Internal paths controlled by different clock, or clocks connected to the ILM module through different ports.  
If the logic between the I/O ports is pure combinational, it is preserved in an ILM.
- ILM data for SI  
The model includes all of the above, plus aggressor drivers or nets which affect I/O paths. It also includes the timing window files in the ILM model directory.

Use `createInterfaceLogic -writeSDC` to generate block level constraints which can be used:

- During a bottom-up design flow to manually build a top-level constraint file from the block constraints. The generated block-level `.sdc` file contains references to the block instances or pins or nets which made it into the ILM model netlist.
- To validate a model at the block level. For example, an ILM netlist and the `.sdc` file can be read in a separate Encounter session and timing analysis can be run on all paths. Then, the results can be compared against timing for the same path during full-block implementation.

**Note:** When `createInterfaceLogic` is called, all views are generated for multi-corner, multi-mode (MMMC) analysis.

## Example ILM Creation

The following method creates a model that can be used in the top-level implementation flow by both `timeDesign` and `optDesign` for both setup and hold efforts, including post-route SI optimization. This model is also used during `clockDesign`.

```
createInterfaceLogic -hold -dir block_A.ILM
```

## Sample Summary Report

The following is a sample summary report generated at the end of the `createInterfaceLogic` command:

-----  
createInterfaceLogicSummary  
-----

| Model       | Reduced Instances | Reduced Registers |
|-------------|-------------------|-------------------|
| ilm_data    | 7153/7621 (93%)   | 174/285 (61%)     |
| 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.

**Note:** You can run the following commands for improving the reduction ratio:

- [setIlmMode](#) -highFanoutPort false

## Preserving Selected Instances in ILMs

You can force the selected instances and nets to be included in the ILM model by using the `createInterfaceLogic` - `keepSelected` parameter.

1. Select instances or nets using the `selectInst` or `selectNet` commands.
2. Specify `createInterfaceLogic` - `keepSelected`.

## Creating ILMs for Shared Modules

You can use the same sub-block module in different ILM blocks, enabling reuse of versatile modules. The `createInterfaceLogic` command considers constant propagate, so that only the enabled parts of a module are considered when creating ILMs for the reused modules. Because the Encounter database cannot handle the same module name in different circuits, the software automatically modifies the module names with the following rule:

`topModuleName+timestamp+$+moduleName`

As an example, one ILM block (`ModuleA`) uses an ALU module (`ALU`) as an unsigned ALU, and a second block (`ModuleB`) uses the ALU as a signed ALU. You can change the input signal to use the ALU differently, setting one ALU as sign enabled and the other to off. When you run the

createInterfaceLogic command, the software considers only the enabled parts of the ALU when creating ILMs for ModuleA and ModuleB. The software also ensures that the name of the ALU module in ModuleA and the name of the ALU module in ModuleB are different.

## Creating ILMs Without Using Encounter Database

If you do not have Encounter database for an implemented block but have a Verilog netlist, constraints, and SPEF for that block, then use the createILMDataDir command to store data in the ILM format.

Following is the usage of the createILMDataDir command:

```
createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -verilog myfile.v

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-spef max.spef.gz -rcCorner rcMax

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-spef typ.spef.gz -rcCorner rcTyp

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-spef min.spef.gz -rcCorner rcMin

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMaxMax.sdc -viewName funct-devSlow-rcMax

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMaxTyp.sdc -viewName funct-devSlow-rcTyp

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc tstMaxMax.sdc -viewName test-devSlow-rcMax

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMinMin.sdc -viewName funct-devFast-rcMin

createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc tstMinMin.sdc -viewName test-devFast-rcMin
```

**Note:** The createILMDataDir command does not support merged models. ILMs created using this command are non-merged models.

**Note:** In the current ILM flow, SDC constraints originally specified against the complete flat netlist for

the design get filtered when being applied against the ILM view of the design.

## Specifying ILM Directories at the Top Level

Use [specifyILm](#) to use the ILM data for a block at the top partition level rather than using the default .lib model. You can run `specifyILm` multiple times in the same session. Each time you run this command, the software overwrites the previous setting for the block. If master/clones exist in the design, the cell name will have the name of the master partition.

**Note:** You can use this command (and `unSpecifyILm`) only if the ILMs are unflattened (`unflattenILm`). You cannot change ILM settings in flattened or ILM view.

Use [unSpecifyILm](#) to revert to using the .lib model for the block.

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

## Example Top-Level Implementation Flow with ILMs

1. Before you start the Encounter tool, prepare the top-level Verilog file, if needed.

If you use the Encounter hierarchical flow in a previous Encounter session, then the `savePartition` command automatically creates the top-level data. Else, you need the following in the top-level directory:

- A Verilog netlist that includes dummy modules for the blocks (ILM or Liberty) in the design.
- A view definition file since ILMs are supported only in the MMMC mode. If you have a non-MMMC design, create or load a view definition file that contains the following:  
`set_analysis_views -setup {mode1_slowCorner} -hold {mode1_fastCorner}`

2. Start an Encounter 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
```

As an alternative, you can use the GUI to specify the ILM directories.

## Design Import - Advanced - Specify ILM

Specify the directory for each module, and the timing constraints file.

4. Load the floorplan.

```
loadFPlan top_floorplan
```

5. Place the design.

```
placeDesign
```

6. Run pre-CTS timing optimization.

```
optDesign -preCTS
```

7. Build the clock tree.

```
clockDesign
```

8. Run post-CTS timing optimization.

```
optDesign -postCTS
```

or

```
optDesign -postCTS -hold      ;#optional
```

9. Route the design.

```
routeDesign
```

10. Run post-route optimization for setup.

```
optDesign -postRoute
```

11. Run post-route optimization for setup and hold.

```
optDesign -postRoute -hold
```

12. Run post-route optimization for SI.

```
optDesign -postRoute -si
```

If you want to create an ILM of the resulting block for use in the next level up in the hierarchy, run the following steps with the above-mentioned flow:

1. Flatten the design as creating ILM calls timing analysis.

```
setILmType -model si
```

```
flattenILm
```

2. Perform timing analysis.

```
timeDesign -postRoute -si
```

3. Create ILM.

```
createInterfaceLogic -dir block_parent
```

## ILMs Supported in MMMC Analysis

Cadence strongly recommends that you use ILMs in the MMMC mode. If you have a non-MMMC design, create and load a view definition file that contains the following:

```
set_analysis_views -setup {mode1_slowCorner} -hold {mode1_fastCorner}
```

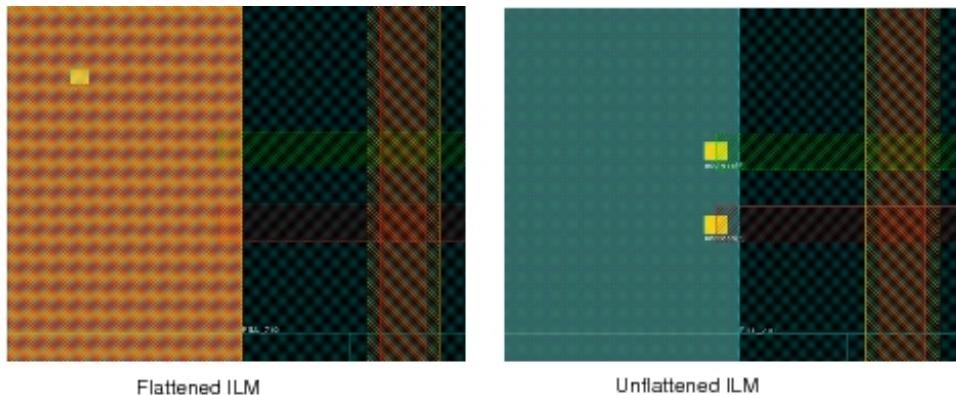
The MMMC analysis for designs including ILMs is identical to MMMC analysis for black box designs except for the following considerations:

1. Views, modes, and corners at the top and partition levels must have same names.
2. When you use `create_constraint_mode` to specify constraints for MMMC, you must specify the ILM constraints using the `-ilm_sdc_files` parameter (that is, timing in the presence of ILMs get constraints from the `-ilm_sdc_files` parameter, not the `-sdc_files` parameter). The `.sdc` files specified with the `-ilm_sdc_files` parameter are allowed to reference nets or pins internal to the ILM model.
3. The interactive constraint commands are currently not supported when using ILMs. Use the `update_constraint_mode -ilm_sdc_files` to change the current constraints files. When using ILMs, the `-ilm_sdc_files` is used. It allows references to nets or pins internal to the ILM model.

**Note:** In the current ILM flow, SDC constraints originally specified against the complete flat netlist for the design get filtered when being applied against the ILM view of the design.

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` parameter for `optDesign` and `timeDesign`. These commands automatically run `setILMType -model si` before calling `flattenILM` such that the SI ILM model is used. Therefore, your present post-route optimization scripts should run successfully in the presence ILMs (without any additional changes).

The following command can be used to get timing reports containing the SI push-out delays on nets using the `-setILMType -model` command:

```
setILMType -model si

# Flattens to the timing model

flattenILM

# Reflattens to SI model, then does not unflatten (All other design
# commands unflatten upon exit, regardless of the flattened/unflattened
# state before invocation)

timeDesign -postroute -si

# Adds incremental delay column (for SI push-out delays) in timing output:

set_global report_timing_format {instance arc cell fanout load slew delay
incr_delay arrival}

# Minimizes the width of the report such that it easily fits into the screen
# without wrapping

set_table_style -name report_timing -no_frame -indent 0

report_timing
```

**Note:** You can also invoke the Global Timing Debugger (Timing - Debug Timing - Generate)

## 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.

- 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 extractRCMode settings determine whether the SI model is generated (the extraction mode won't be changed internally to generate the SI model).

## Merged ILM Model

The new merged ILM model reduces the disk usage of an ILM model. It runs the top-level applications faster as compared to the earlier ILM model. 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 CTS model. All other registers are excluded.
- worst inter-clock paths for the timing model.

**Note:** The merged ILM model is not supported at the top level, that is, when some blocks are generated with merged models while others are generated with non-merged models.

## Interactive Use of ILMs

- Commands such as optDesign, timeDesign, clockDesign, and so on automatically take care of flattening and upon completion, leaves the design in an unflattened state.
- Timing commands require you to run flattenILM first so that the nets and instances internal to ILM are exposed to the timing engine.

```
encounter> flattenILM  
ilmView> report_timing
```

Notice that the prompt changes to `ilmView` after `flattenILM`.

- The Global Timing Debugger (GTD) also requires the design to be in a flattened state. GTD displays rows with instances or nets which are internal to the ILM as grayed out.
- The new `-ilm` parameter has been added to the `saveNetlist` command to write a netlist with ILM guts. This parameter can be specified only in the `flattenILM` state.

- The non-timing commands require the design to be in an unflattened state before invocation:

```
ilmView> unflattenIIm  
encounter>verifyGeometry
```

## Handling Interactive Constraints

You cannot specify the interactive constraints when ILMs are not flattened (`unflattenIIm`). In the flatten mode (`flattenIIm`), you can specify both interactive and modeless constraints and these constraints are used during various cycles of `unflattenIIm` to `flattenIIm`. During `saveDesign`, these constraints are honored.

To specify additional constraints while running `unflattenIIm`, 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 modeless constraints such as `timing_derates` and `group_path` in a separate file, and run:

```
loadTimingCon -ilmNonSdcFile <mode-less_constraint.sdc>  
set_global timing_defer_mmmc_object_updates false  
set_analysis_view -update_timing
```

For non-MMMC flow, run:

```
loadTimingCon -ilm
```

---

# What-If Timing Analysis

---

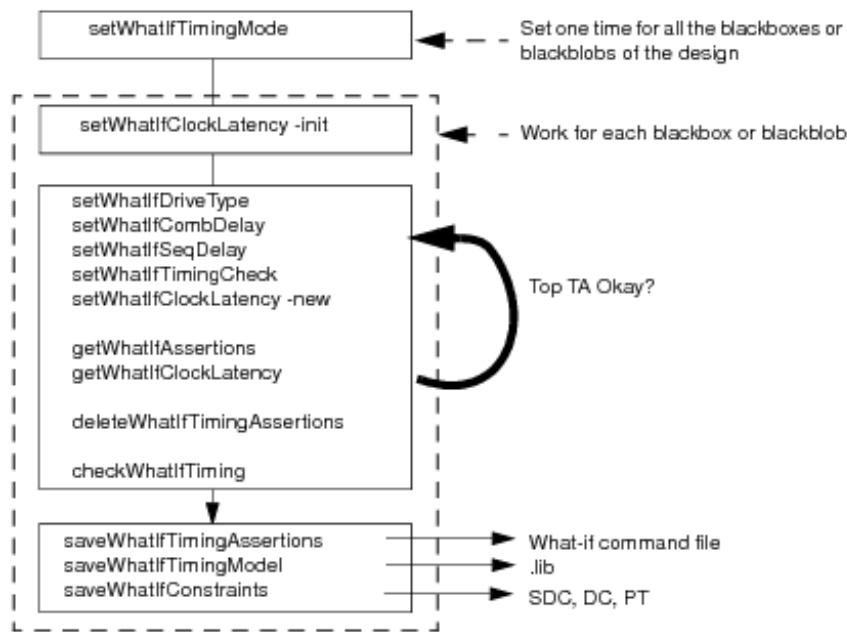
- [Performing What-If Timing Analysis](#)
  - [Prerequisite](#)
  - [Timing Models Supported for What-If Timing Analysis](#)
  - [Using the What-If Timing Commands](#)

## 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 Encounter® Digital Implementation System (EDI 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 Encounter software provides what-if timing commands to support what-if timing budgeting. For more information on what-if timing commands, see the chapter ["What-if Timing Commands,"](#) in the *Encounter Text Command Reference*.

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

The following diagram shows the what-if budgeting flow.



## Prerequisite

Prior to using what-if timing commands, you must load the what-if timing models into the database because the what-if timing commands simulate the modifications of the timing arcs.

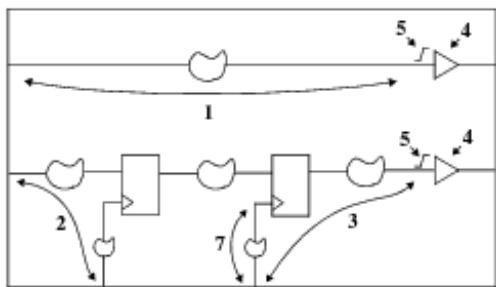
If you do not have timing models in the early design phase, you can use the [`setWhatIfClockPort`](#) command to create clock ports. You can then use the clock port to create timing arcs.

## Timing Models Supported for What-If Timing Analysis

The Encounter software supports two timing models for what-if timing analysis: intrinsic and normalized. You can select only one mode at a time.

Figure 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> |

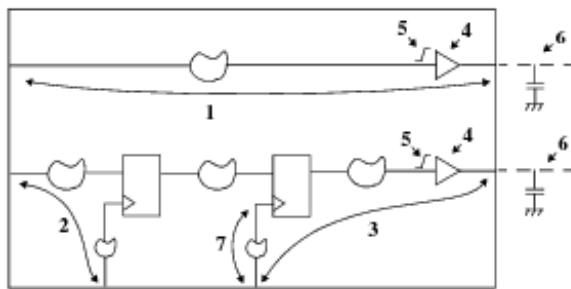
An intrinsic timing model uses the following formula for timing arcs ending on output ports:

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,

*constant delay* = Timing arc delay including driver delay

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

*constant delay - clock latency* must be greater than *driver delay*\*<sup>\*</sup>

In a normalized timing model mode driver input slew is always required. In this mode, timing arcs are only load dependant. If you do not specify the driver total output net capacitance, the software takes real net capacitance into account.

## Using the What-If Timing Commands

You can perform the following tasks with the what-if timing commands:

- Selecting Timing Model

Use the following command to select the timing mode:

- [setWhatIfTimingMode](#)

- Defining generated clocks on internal pins:

Use the following command to create an internal pin and to define a generated clock on the pin.

- [createWhatIfInternalGeneratedClock](#)

- Set the following values on the what-if ports, if required:

- Capacitance
  - Maximum capacitance
  - Maximum transition
  - Maximum fanout

Use the following command to set these values on the what-if ports:

- [setWhatIfPortParameters](#)

By default, the parameters specified with the [setWhatIfPortParameters](#) command are applied to all ports in the what-if timing analysis model. If you want to apply the values for a particular port, specify the port name with the [setWhatIfPortParameters](#) - port parameter.

- Selecting the precedence between the values set by `setWhatIfDriveType` command and the values set by the `setWhatIfPortParameters` command

On output ports, parameters such as capacitance value, maximum capacitance values, maximum transition value, or the maximum fanout value can come from the driver (`setWhatIfDriveType` command) or they can be set through the

`setWhatIfPortParameters` command.

Use the following command to define which of these values will take precedence in case of a conflict.

- [setWhatIfTimingMode](#)

- **Modifying Timing Arcs**

While what-if commands are the same for both intrinsic and normalized timing models, the delay value specified in the commands for the combinatorial and the sequential timing arcs has different meaning. The driver output net capacitance is a characteristic of the normalized timing model only. Whenever you create or modify a timing arc, the timing graph is updated automatically. The Encounter software recomputes the entire timing arc whenever any of the parameter such as clock insertion delay, timing arc delay or driver type is modified.

**Note:** The timing sense of the driver is taken into account in the combinatorial what-if timing arc description--while applying the drive type, the timing sense of the combinatorial arc is replaced by the timing sense of the driver's timing arc. For sequential arcs, the timing sense is always set to `non_unate`.

Use the following commands to modify timing arcs:

- [setWhatIfDriveType](#)
- [setWhatIfCombDelay](#)
- [setWhatIfSeqDelay](#)
- [setWhatIfTimingCheck](#)
- [setWhatIfClockPort](#)
- [setWhatIfClockLatency](#)

- **Getting Timing Arcs Assertions**

Use the following command to get what-if timing arc assertions:

- [getWhatIfTimingAssertions](#)

- **Saving Timing Arcs Assertions**

Use the following command to save what-if timing arc assertions:

- [saveWhatIfTimingAssertions](#)

- Deleting Timing Arcs Assertions

Use the following command to delete the what-if timing arc assertions:

- [deleteWhatIfTimingAssertions](#)

- Checking Timing Assertions

Use the following command to check the what-if timing assertions:

- [checkWhatIfTiming](#)

- Generating what-if timing Models

After modifying the what-if timing model (in memory) using the what-if command, you can generate an updated timing model (.lib).

Use the following command to generate an updated .lib file:

- [saveWhatIfTimingModel](#)

- Generating What-If SDC constraints

The Encounter software generates the what-if timing constraints considering the top-level environment of the blackbox or blackblob. It provides a higher convergence for a top-down flow. The software generates drive, load and transition as IN context. The software generates the input and output delays as OUT context taking into account the last modifications done when you use the what-if commands.

Use the following command to save the What-If constraints:

- [saveWhatIfConstraints](#)

## Bus Planning

---

- [Overview](#)
- [Bus Planning Flow in Encounter](#)
- [Creating a Bus Guide](#)
  - [Using the Edit Bus Guide GUI](#)
  - [Drawing a Bus Guide](#)
  - [Using Text Commands](#)
  - [Example](#)
- [Moving and Stretching a Bus Guide](#)
- [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 Encounter software enables you to plan and create bus guides which are used to guide the path of busses for floorplanning, partition pin optimization, feedthrough insertion, congestion prediction in trialroute, and final routing in nanoroute.

Most designs need bus planning for estimating the design size and routing channel widths. Without bus guides, the routers do not route all the bus bits together on the desired path. Routing the bus bits outside the desired path can have high cost implications. Hence it is very important to accurately plan the bus guide layouts.

Bus planning is critical in the prototyping stage of the hierarchical flow. Use the bus planning capability to guide the path of bus routing for feedthrough insertion, partition pin optimization, and congestion prediction. If you are in the implementation stage, use bus planning to guide the path of busses for detailed routing.

### Bus Planning Flow in Encounter

For hierarchical designs, you create bus guides before or after assigning the partition/black box pins. For flat or top-level designs, you create bus guides before routing. Normally, you create bus guides before pin assignment.

The following steps describe the bus planning flow in Encounter:

1. Importing the design  
Import the design into the Encounter environment.
2. Floorplanning the design  
If the design is a partition design then specify partitions. For more information, see [Specifying Partitions and Blackboxes](#) in the "Partitioning the Design" chapter of the *Encounter User Guide*.  
If it is a black box design then define black boxes and specify their sizes. You can manually preplace black boxes/macros or run [planDesign](#) to automatically place them. Further, adjust the floorplan if needed.
3. Defining net groups  
Group the bus bit nets together as net groups using [createNetGroup](#) and/or [addNetToNetGroup](#) commands.
4. Creating bus guides  
Create bus guides associated with the net groups, to guide routing for all the nets of the specified net group. Bus guides can be created using the Edit Bus Guide 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 Encounter placer ([placeDesign](#)) to move your macros and/or black boxes, set their placement status to **fixed** before running placement.  
**Note:** This is an optional step for designs that do not have standard cells at full-chip level.
6. (Optional) Routing the design  
Run [trialRoute](#) to route the design.

7. (Optional) Inserting feedthrough buffers  
Feedthrough can be inserted based on routing or placement. If `trialRoute` was run before this step, then feedthroughs are inserted based on routing. For more information, see the "Inserting Routing Feedthroughs" section of the [Partitioning the Design](#) chapter of the *Encounter User Guide*.
8. Assigning pins  
Assign pins using [`assignPtnPin`](#) command.
9. Committing partition  
Commit partitions using [`partition`](#) command.
10. Saving Partition  
Save the partition information using [`savePartition`](#) command.
11. Running NanoRoute at the top-level design  
Perform detailed routing using [`NanoRoute router`](#) at the top-level design.

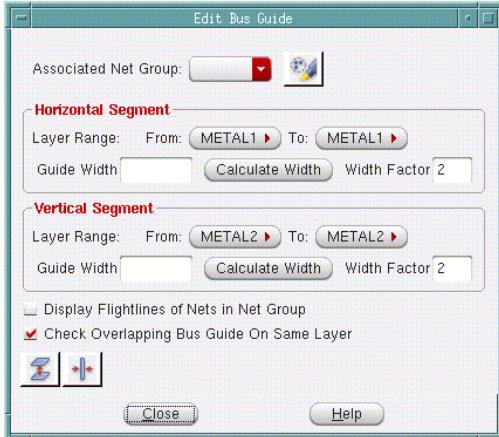
## Creating a Bus Guide

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 Encounter, allows you to create bus guides before or after assigning the bus pins. Using the *Edit Bus Guide* form, you can edit the bus guide properties and interactively create the bus guide. You can specify the net group associated with the bus guide, layer or layer range on which the bus guide is to be created, and the width of the bus guide segment. By default, the bus guide editor derives the default minimum guide width required to hold all the nets assigned to the bus guide. If the bus guide connects to placed pins on block edges, the bus guide editor automatically adjusts the width of the guide segment to cover all the pins of nets in the net group. The bus guide editor provides options to enable overlapping check for bus guides created on a specific layer and display flight lines of nets in the net group, when creating the bus guides.

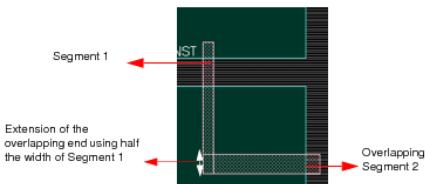


For more information on the *Edit Bus Guide* form, see [Edit - Object - Edit Bus Guide](#) in the [Edit Menu](#) chapter of the *Encounter Menu Reference*.

### Drawing a Bus Guide

To draw a bus guide in Encounter, you must first click the *Add Bus Guide* icon in the toolbar.

Once you are in the bus planning mode, you can draw the bus guide segment by clicking the left mouse button and dragging it along the points of center line for the guide segment. To end a bus guide segment, double-click the left mouse button. By default, the bus guide extends half width for the overlapping end of the created segment. However, if the guide segment overlaps with another segment that has bigger or smaller width, the bus guide 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:

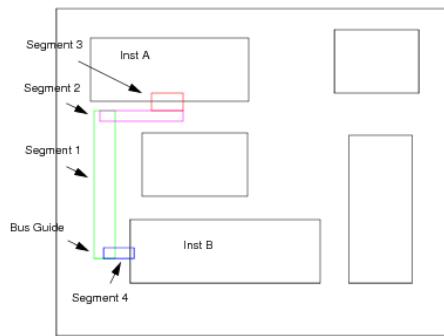


Figure 11-1

You can also draw a bus guide segment that connects to the placed pins of the associated net group.

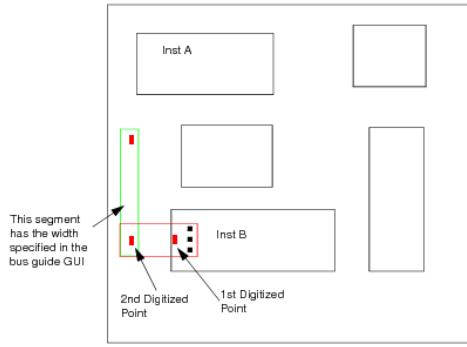


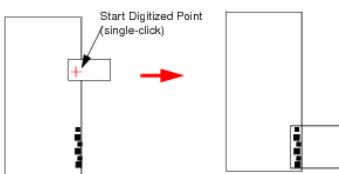
Figure 11-2

If you click on the partition boundary side where the pins are placed, the bus guide editor automatically snaps to these pins. If the width value specified in the bus guide editor is smaller than the width required to fully cover all these pins, the bus guide editor derives new width for the guide segment such that all the associated physical pin geometries are covered. If the width value is bigger than the width that needs to cover all pins, the editor will use the current width value without adjusting it.

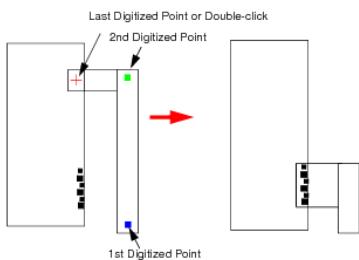
In the Figure 11-2, the width of the segment defined by the first and the second digitized points is derived based on the placed pin information such that the segment width can fully cover all the pins. The width of the next segment (defined by second and third points) is the width that is specified 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.

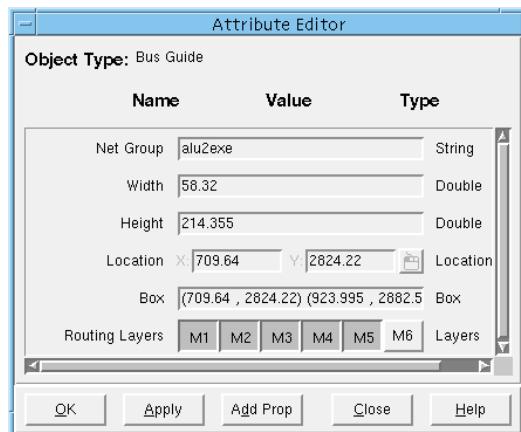
The following example illustrates the snapping behavior at the starting digitized point. The snapping occurs before you specify the second point:



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                         | Usage                                                                                                                                                  |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">createBusGuide</a>   | Creates a bus guide segment.                                                                                                                           |
| <a href="#">deleteBusGuide</a>   | Deletes a bus guide.<br><b>Note:</b> You can also delete a bus guide segment by selecting the segment and pressing the <b>Del</b> key on the keyboard. |
| <a href="#">deselectBusGuide</a> | Deselects a bus guide segment.                                                                                                                         |
| <a href="#">selectBusGuide</a>   | Selects a bus guide segment.                                                                                                                           |
| <a href="#">selectBusGuide</a>   | Selects a bus guide segment with its specified bounding box.                                                                                           |

For more information on the commands, see the "[Bus Plan Commands](#)" chapter in the *Encounter Text Command Reference*.

The following Example describes the steps to create bus guides using text commands.

### Example

This sample script creates 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

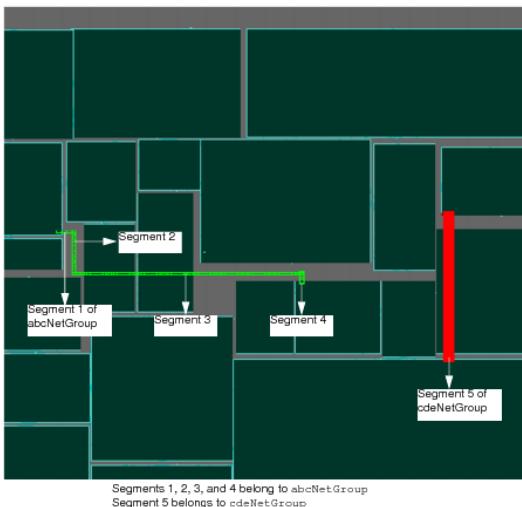
#Place the design since design has some top-level cells
placeDesign

#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...
```

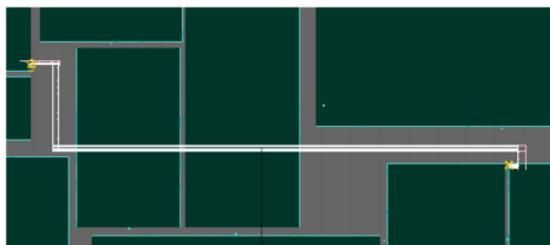
The following figure displays the bus guide associated with the net group `abcNetGroup`, highlighted in green, and the bus guide associated with the net group `cdeNetGroup`, highlighted in red:

After running `createBusGuide` to create 5 segments



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.

## Customizing the Bus Guide Display

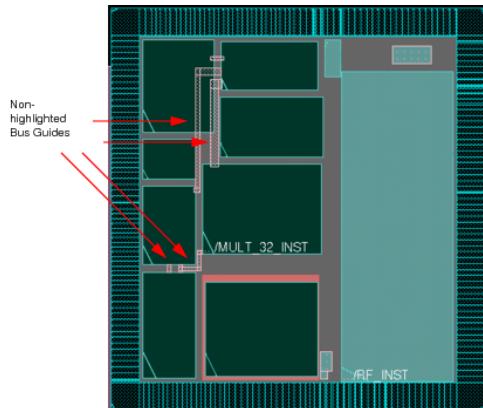
You can specify multiple colors for bus guide objects in the design, using the *Bus Guide Color Selection* form. (*Color Preferences -- Objects -- Bus Guide -- Bus Guide Color Selection* )

### Highlighting and Dehighlighting the Bus Guide

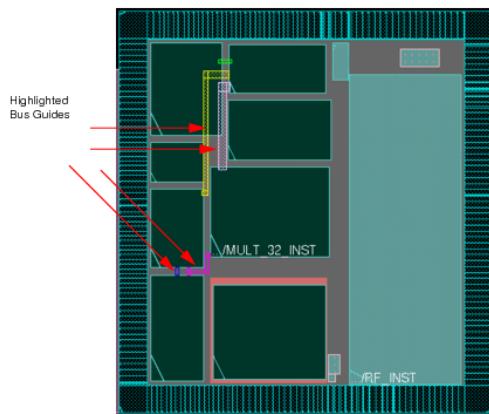
After specifying colors for bus guides, you can highlight the bus guides in the design using the *Edit -- Bus Guide -- Color* menu command.

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

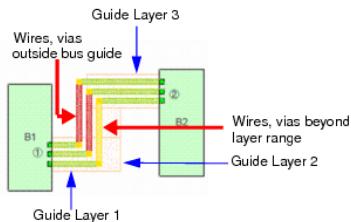
The bus guide data is stored in the floorplan spr file (.fp.spr file). You can save and restore this information using the [saveFPlan](#) and [loadFPlan](#) commands.

However, you cannot load the .fp.spr file having bus guide information from the 8.1 version, into an older version of Encounter.

## 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.

---

# 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](#)

## Overview

Most of the system-on-a-chip devices are designed in a traditional flat flow that avoids the effort to set up a design hierarchy. However, in multi-million gate designs, this could result in memory limitations and long run time. Designs team 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 Encounter Digital Implementation System (EDI System) 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` option should be used for prototyping and the `-highEffort` option should be used for tapeout mode. Use the `-handlePartition` or the `-handlePartitionComplex` parameter for channel-based designs. Use the `-handlePartitionComplex` parameter for channelless designs only after the feedthrough insertion step.

For 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.

- 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

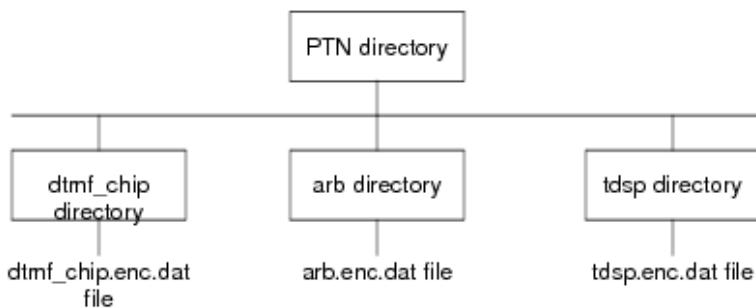
Chip assembly is the last stage in the top-down process and consists of bringing together the detailed information for the top-level and all of the blocks for full chip extraction, power, timing, and crosstalk analysis. Chip assembly is done using the [assembleDesign](#) command.

**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 EDI System 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 details of the syntax and the parameters, see the description of the [assembleDesign](#) command in the *EDI System Text Command Reference*.

For this example, you would run the [assembleDesign](#) command as follows:

```
assembleDesign -topDir PTN/dtmf_chip/dtmf_chip.enc.dat -blockDir PTN/arb/arb.enc.dat -blockDir PTN/tdsp/tdsp.enc.dat -topFP fullChip.fp
```

This assembles the entire design.

You can also use the [assembleDesign](#) command to bring back specified block data from OpenAccess database. Here is an example:

```
assembleDesign -topDesign testOALib DTMF layout -block testOALib ptn1 layout -block
```

```
test0ALib ptn2 layout
```

In this example, the OpenAccess database top-level library is `test0ALib`, 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***

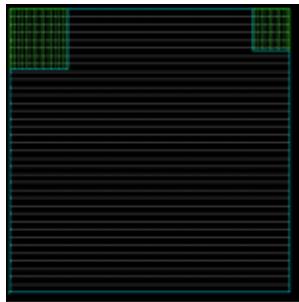
The size of a block-level design can be derived or adjusted using the *Floorplan - Specify Floorplan* menu command or the [floorplan](#) text command. The EDI System software can support a rectilinear block level design. You can use the same procedure to create a rectilinear partition to create a rectilinear 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](#) text command to assign I/O pins based on placement information.

You can specify initial I/O pin placement in an I/O constraint file. For more information, see the Generating the I/O assignment File section in the "[Data Preparation](#)" chapter of the *EDI System User Guide*. You can read in the I/O constraint file into the EDI System environment during the design import step, or use the [loadIoFile](#) text command after reading in the netlist.

If an I/O constraint file does not exist, an initial I/O pin placement can be derived from cell placement. After placing macros and standard cells, the placer can internally call the `assignIoPins` text command to place I/O pins based on current cell placement. By default, pins are placed under power areas on different layers.

**Note:** Use the [setPlaceMode](#) `-placeIoPins` option to disable I/O pin assignment during placement.

After I/O pins have been assigned, you can further refine the current I/O pin assignment by doing either of the following:

- Adjust pins (using the Pin Editor or the [editPin](#) text command). You can also use direct pin manipulation to manually move selected pins to different locations.
- Run incremental pin assignment by running the [assignIoPins](#) text command. This command honors fixed pins and re-assigns only the ones that have a *placed* or *unplaced* status.

**Note:** The `loadIoFile` text command automatically sets the I/O pin placement status to fixed. For the pins that need to be re-assigned, you must change their pin placement status.

You can use the [legalizePin](#) 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 option of the assembleDesign command.

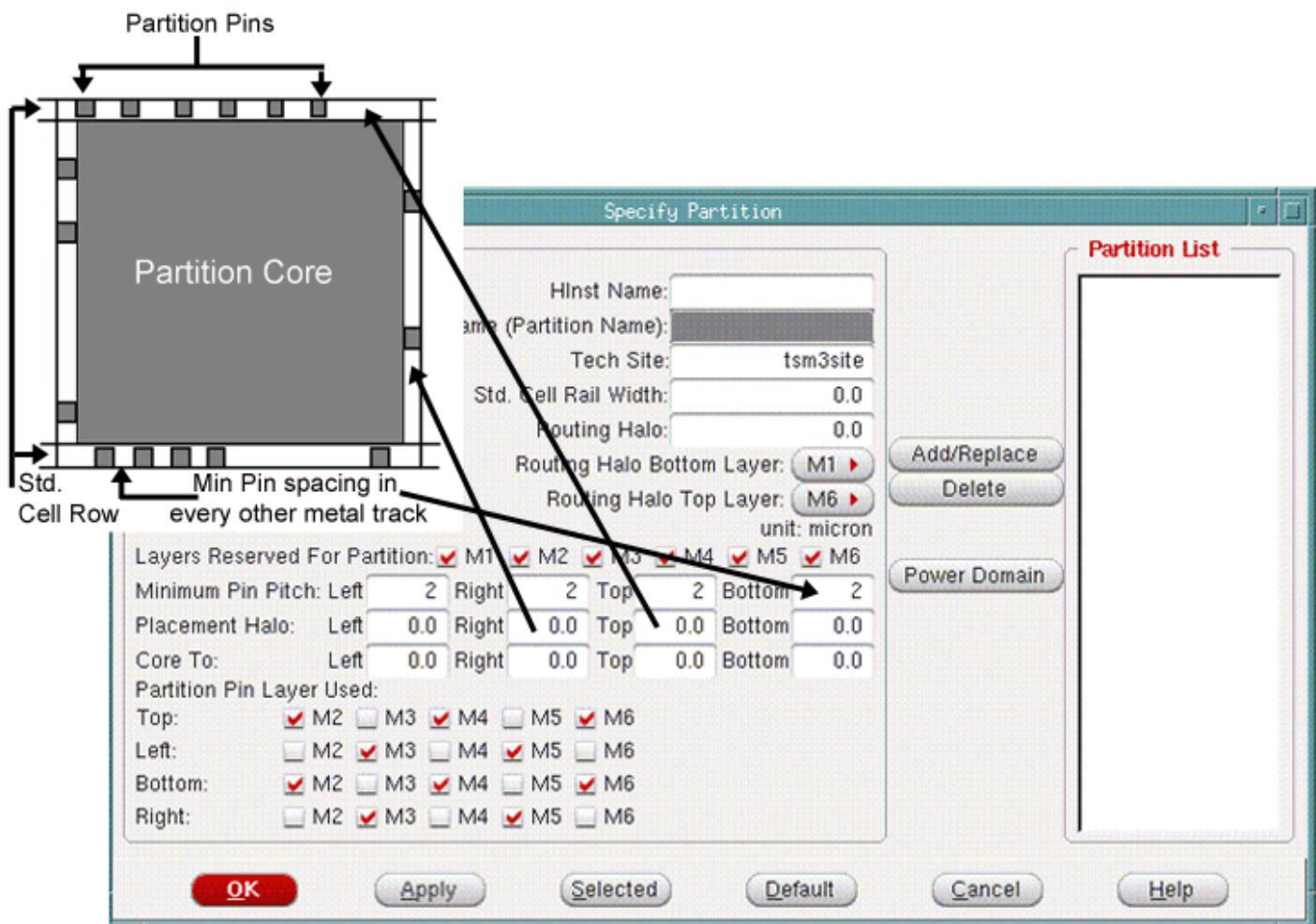
## **Specifying Partitions and Blackboxes**

- [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](#) text 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 *EDI System 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` text command, to define a new module boundary with its coordinates in the core area.

**Note:** A blackbox is a special partition where this restriction does not apply.

**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 M<sub>1</sub>, M<sub>2</sub>, M<sub>3</sub>, M<sub>4</sub>, and M<sub>5</sub> selected, and M<sub>6</sub> 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.

- **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 EDI System 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 objects 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 EDI System 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.

- 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 EDI System 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 EDI System software automatically convert the orientation of master blackboxes to R0.

In addition, you can also run the [changeBBoxMasterToR0](#) command to convert the orientation of the master blackboxes to R0. This would be useful for example, you restore a design and want to convert the orientation of all the master blackboxes to R0.

The following sections provide addition information about automatic conversion of orientation and about the [changeBBoxMasterToR0](#) command.

- Automatic Conversion of Orientation
- 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 [EDI System 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.

**i** There is no change in the design physically as a result of these transformations. Only the cell orientation and the instance representation are modified.

As an example, if the blackbox master instance is MX, then after the transformation:

- cell geometries are transformed to MX
- The orientation of the master instance is changed to R0.

### Performing R0 Transformation

For designs that contain blackboxes whose master instances have a non-R0 orientation, you can use the [changeBBoxMasterToR0](#) command to convert the orientation of the master blackboxes to R0. The syntax of the command is as follows:

```
changeBBoxMasterToR0 [-checkOnly] [{cellName | cellNameList}]
```

If a cell name, or a list of cell names, is not specified, the command converts the orientation of all the non-R0 master blackboxes to R0.

If the `-checkOnly` parameter is specified, the command does not actually convert the orientation of any master blackbox; it only displays the number of master blackboxes whose orientation would have been changed had the command been run without the `-checkOnly` parameter.

For more information, see the description of the [changeBBoxMasterToR0](#) command in the *EDI System Text Command Reference*.

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 EDI System software, or use the [`freeDesign`](#) text command.
3. Rerun the EDI System 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 EDI System 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 EDI System 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 EDI System 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.

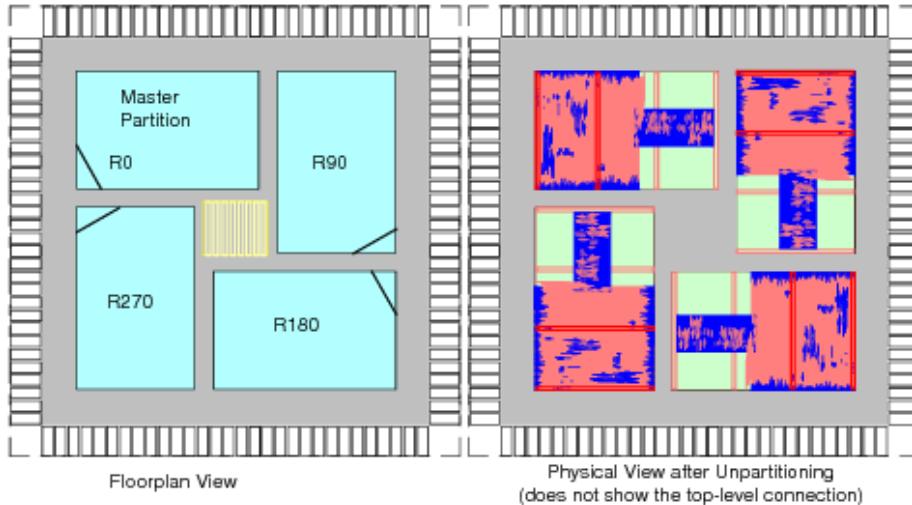
Use the [getClonePtnOrient](#) command to retrieve orientation information of a specific partition clone.

For routing purposes, the EDI System 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 EDI System 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:

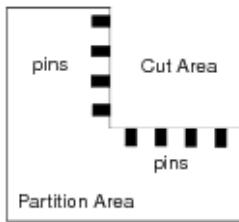


**Note:** The illustration above only shows the wire information inside the partition, and does not include the top-level connection.

## 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` text command. You should specify a module as a partition before using `createPtnCut`.

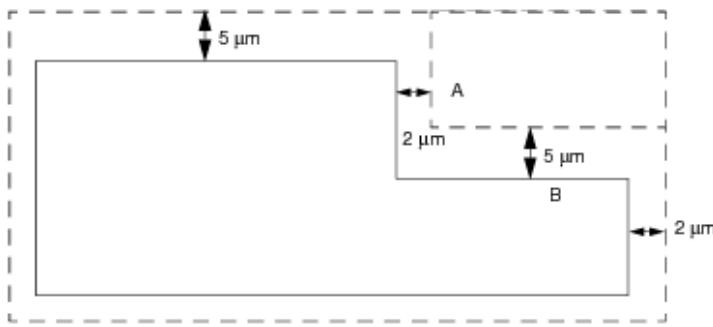
For repeated partitions or blackboxes, when you create a cut on one instance--either master or clone--the cut is applied to the other instances as well.

**Note:** If a cut is made on a blackbox that has pins assigned to it, the affected blackbox pins are automatically moved to the new edge boundary created by the cut.

## Specifying Core-to-I/O Distance for Partition Cuts

Core-to-I/O distance is specified in the Specify Partition form. If the partition has a partition cut, core-to-I/O distance is honored where the cut is specified. The specified top, bottom, left, and right core-to-I/O distances are automatically assigned for the cutting edges that face the north, south, west, and east side, respectively.

For example, if you specify a core-to-I/O distance of 5 m for the top and bottom, and 2 m for left and right sides:



The core to I/O distance for the edge A (facing east) should be 2 m. The core to I/O distance for the edge B (faced to north) should be 5 m, same as the top side.

## Working with Nested Partitions

The EDI System 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.

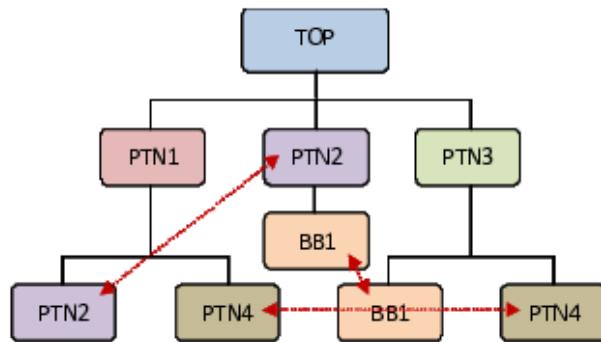
### Using the Multi-level Hierarchical Flow

**!** 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. To use this limited-access feature, you must have the `Encounter_Giga_Scale_GXL` license.

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 EDI System 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`](#) text command after pin assignment to make sure that all pins are assigned, are placed on routing grids, and are not overlapping.

Blackbox pins are assigned in the same way as partition pins.

Pin assignment supports the following:

- Rectilinear partitions and black boxes
- Repeated partitions and black boxes. Both master and clones are considered when assigning their pins.
- Designs with an arbitrary origin.
- Non-uniform tracks.

**Note:** Pin assignment assigns only signal pins but it does honor power/ground stripes and followpins. Power and ground pins are created when the design is partitioned.

**!** You cannot assign blackbox or partition pins when design is unplaced and unrouted. Make sure you place the design before partitioning; otherwise, all pins will be unplaced.

The following sections describe pin assignment in EDI System:

- [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 EDI System 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](#) text command or by using the [Edit Pin Group](#) GUI form (*Edit-Edit Pin Group* ).

You can add pins to a cell pin group with the [createPinGroup](#) text command or with the [addPinToPinGroup](#) text command.

Cell pin groups do not have to be associated with a partition pin guide because a pin group is not a constraint on any partition edge. In this case, the pin assignment program can freely place this group of pins on any edge of the partition. However, pins that belong to this pin group are still placed together in adjacent locations.

With a pin group you can:

- Optimize order of pins within a cell pin group to improve wire length using the `-optimizeOrder` option of the [createPinGroup](#) text command. If this option is not specified, the pin order is exactly as specified in the pin group.
- Specify pin spacing. The default minimum pin spacing between pins of a cell pin group is two tracks.

The following commands create a pin group pGroup1 that consists of 3 INT bus bit pins of the module ALU. These pins can be optimized within the pin group:

```
createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -optimizeOrder  
Or  
createPinGroup pGroup1 -cell ALU -optimizeOrder  
addPinToPinGroup -cell ALU -pinGroup pGroup1 -pin {INT[0] INT[2] INT[3]}
```

Use the [deletePinGroup](#) command to delete a pin group or all pin groups.

Use the [deletePinFromPinGroup](#) command to delete a pin from a pin group.

### **Net Group**

You can create a net group using the [createNetGroup](#) command or by using the [Edit Net Group](#) GUI form (*Edit-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 net pin order, alternate pin layers, and specify pin spacing for a net group.

The following commands create a net group nGroup1 that has two nets NET1 and NET2 with minimum pin spacing of 2 tracks.

```
createNetGroup nGroup1 -net {NET1 NET2} -spacing 2  
Or  
createNetGroup nGroup1 -spacing 2  
addNetToNetGroup nGroup1 -net {NET1 NET2}
```

Use the [deleteNetGroup](#) command to delete a net group or all net groups.

**Note:** When you delete a net group, any bus guide associated with the net group also gets deleted.

Use the [deleteNetFromNetGroup](#) command to remove a net from a net group.

#### **Pin Guides**

You can create a pin guide to constrain a bus, net, pin, net group, or pin group to be placed in specific areas. A pin guide is used for specifying a physical guided area where pins belonging to the pin guide will be placed.

**Note:** While creating a pin guide, you cannot optimize the order of pin members or specify minimum spacing. If you want to control the pin order and the pin spacing of the members of a pin guide, first create a net group or a pin group and associate this net group or pin group with a pin guide.

A pin guide can support multiple constraint pin layers. In addition, any bus, net, pin, net group, or pin group can be assigned to multiple pin guides.

You can create a pin guide using the [Create Pin Guide](#) widget from the GUI or through the [createPinGuide](#) text command. A physical location constraint can be specified either as a rectangular area or as an edge constraint. If you specify a physical location constraint as an edge constraint, you will also need to specify the partition/black box cell name.

Here are a few examples of using the [createPinGuide](#) text command to create a pin guide.

Example 1: The following command creates a pin guide for a net group nGroup1. The pin order within this net group will be optimized. The pin members of this pin guide can be placed on the top edge of the cell ALU. Pins will be placed on Metal2 or Metal4 layers:

```
createNetGroup nGroup1 -net {NET1 NET2} -optimizeOrder
createPinGuide -netGroup nGroup1 -edge 1 -cell ALU -layer {Metal2 Metal4}
```

Example 2. The following command creates a pin guide for a pin group pGroup1 of cell/module ALU. Pins of this pin guide will have a minimum spacing of 2 tracks:

```
createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -spacing 2
createPinGuide -area 678.52 371.25 778.53 787.33 -pinGroup pGroup1 -cell ALU
```

The pins will be assigned on the preferred layers.

Example 3. The following command creates a pin group pGroup2. This pin group can be placed on the top edge or the right edge of the cell TDSP. For top edge, pins can be assigned on the 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 Metal16}  
createPinGuide -edge 2 -pinGroup pGroup2 -cell TDSP -layer Metal5
```

You can also use the GUI to create a partition pin guide, as follows:

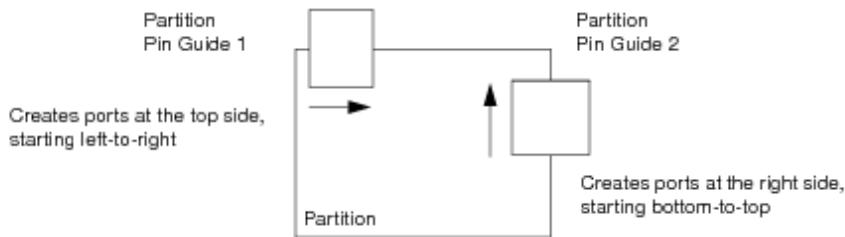
After you have determined a pin guide location in the design display area, you can create a partition port for a net or bus name and add a partition pin guide. To add a partition pin guide through the GUI, complete the following steps:

1. 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](#) text 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.

Example 1: 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
```

Example 2: 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 -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

Example1: 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
```

Example 2: 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* menu command). Alternatively, you can use the [setPinConstraint](#) -layer -edge command.

You can specify layer constraints at partition level, pin guide level, or pin level.

- Partition level

Layer constraint per edge can be specified at partition level using either

- the [Specify Partition](#) form (*Partition - Specify Partition* menu command), or
- the [definePartition](#) command with -pinLayerTop, -pinLayerBottom, -pinLayerLeft, and -pinLayerRight parameters. These layer constraints will be applied to all pins on a particular edge of the specified partition.
- the [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**

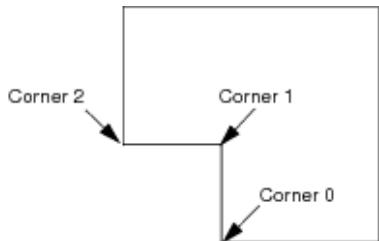
To keep pins away from partition/black box corners, you can set the pin-to-corner distance constraint.

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, 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](#) text command. These pre-assigned pins will have *fixed* placement status so pin optimizers will honor them. For more details, see the [Pin Editor](#) section in the "Edit Menu" chapter of the *EDI System Menu Reference*.

#### ***Setting Constraints on a Specific Pin***

Use the [setPinConstraint](#) command to specify the following constraints for a particular pin:

- Physical location  
A pin can be constrained by specifying its coordinate (x, y) location and its preferred routing layer. If specified location is not on valid track, the pin will be snapped to the closest location. To keep the pin on non-preferred routing layer or to not snap the pin, use the [editPin](#) command instead.

Besides an actual physical location, a pin can also be constrained to a particular edge.

- Layer
- Spacing

For example, the following command specifies that the pin `reset` of partition cell `mult_32` should be

placed on top edge with either Metal15 or Metal17 routing layer.

```
setPinConstraint -cell mult_32 -pin reset -edge 1 -layer {Metal15 Metal17}
```

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 EDI System 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.

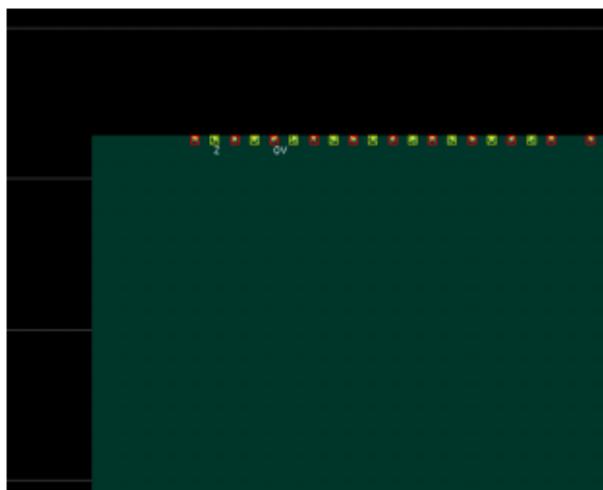
Example:

```
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 *EDI System Text Command Reference*

### Reporting QoR of Pin Assignment

You can use the [pinAnalysis](#) command to report certain Quality of Results (QoR) metrics for pin assignment. The [pinAnalysis](#) command deletes the existing routes, reroutes the design ensuring that the routes pass through partition pins, and reports pin assignment QoR metrics.

 The [pinAnalysis](#) command creates new routes. The original routes are not retained.

**Note:** The [pinAnalysis](#) command reroutes the design using `trialRoute -honorPin`. This command thus takes at least as much time as running Trial Route. Also, because Trial Route is run, you can use the generated routing information for other applications such as time budgeting.

When you run the [pinAnalysis](#) command after assigning pins but before committing the partition, the following are reported:

- Pin-deviation from routing cross-points
  - Estimated net-length and via-count
  - Comparison between net lengths, via counts, and congestion indexes of the original and the new routes
- Note:** *Original route* refers to the routing that was performed before the pin assignment step. *New route* refers to the routing performed by the [pinAnalysis](#) command.
- Number of two-pin nets that have aligned pins and number of two-pin nets that have unaligned pins
  - CPU time and memory usage

When you run the [pinAnalysis](#) command after committing the partition, the following are reported:

- Estimated total top-channel net-length and via-count
- Congestion report with overall congestion index values for top-level channel nets
- Number of two-pin nets that have aligned pins and number of two-pin nets that have unaligned pins
- Run time and memory usage

You can check the legality of pin assignment (similar to the functionality of [checkPinAssignment](#) command) by specifying the `-checkLegality` parameter with the [pinAnalysis](#) command.

You can also save the output of the command in a text file by specifying the `-outFile` parameter.

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 |
```

```
-----
| dtmf_chip | 0 | 0 | 0 | 0 | 0 |
```

=====  
Pin QoR Report:

-----  
Unaligned 2-pin Net: DTMF\_INST/t\_addrs[0].

Unaligned 2-pin Net: DTMF\_INST/m\_clk.

There are 2 unaligned 2-pin nets.

=====

-----| QoR Metric | Before Pin-Assignment | After Pin-Assignment | Percent Increase |

-----

| Horizontal | 0.000% | 0.032% | NA |  
| Congestion | | | |

-----

| Vertical | 0.000% | 0.697% | NA |  
| Congestion | | | |

-----

| Total | 6.089e+05um | 6.150e+05um | 1.011% |  
| Net-Length | | | |

-----

| Total | 48300 | 54036 | 11.876% |  
| Via-Count | | | |

-----

Completed pinAnalysis (CPU=0:00:07.6 MEM=5.9)

**Note:** The internal pins (shown in the Legality Report) are not checked for legality. Internal pins are the pins that are not on the partition boundary.

In the previous table, the Before Pin-Assignment column shows the results of the routing before pin assignment and the After Pin-Assignment column shows the results of the routing after pin assignment.

### Refining Pin Assignment and Fixing Pin Violations

After assigning partition or blackbox pins, you can further refine the current pin assignment and fix any pin violations using one or more of the following methods:

- Adjusting Pins
- Aligning Partition Pins
- Running Incremental Pin Assignment
- Adjusting Floorplan or Floorplanning the Design Again
- Performing Pin Assignment Again

These steps are explained in the following sections.

#### ***Adjusting Pins***

You can Adjust pins using the [Pin Editor](#) or the [editPin](#) text command. You can also use direct pin manipulation to manually move selected pins to different locations.

#### ***Aligning Partition Pins***

You can align partition pins with other block pins (using the Pin Editor or the [pinAlignment](#) text command).

The [pinAlignment](#) command can be used to align partition/black box pins with or without specified reference object(s). Reference objects can be hard macros, blackboxes, I/O pads, 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 (without using `-ptn` and `-pin` options):

```
setPtnPinStatus <partitionName> <pinName> fixed  
assignPtnPin
```

### ***Adjusting Floorplan or Floorplanning the Design Again***

Sometimes a sub-optimal floorplan can also lead to a bad pin placement result. If this is the case, re-adjust the floorplan and run pin assignment again.

### ***Performing Pin Assignment Again***

Perform pin assignment again. If the partitions or blackboxes have been committed, use the [flattenPartition](#) command to unassign the pins. If the partitions or blackboxes are not yet committed, use the [setPtnPinStatus](#) command to unplace the pins.

### ***ECO Pin Assignment***

The EDI System 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 EDI System 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 EDI System 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 *EDI System User Guide*.

- Randomly assigning I/O pins

You can create an I/O template file with random I/O pin assignment using the following steps. I/O placement is evenly distributed on design boundary:

- a. Import design
- b. Run the `saveIoFile` command with the `-template` option
- c. Use the `loadIoFile` command to load I/O file generated from the step 2

- Placing the design

After importing a design and floorplanning it, you should place the design. By default, the EDI System placer (`placeDesign`) internally invokes the I/O pin assignment to place I/O pins based on the current floorplan.

**Note:** Set the `-placeIoPins` option of the `setPlaceMode` command to `False` if you want to disable I/O pin assignment during the placement step.

## Refining Pin Placement

After I/O pins are assigned, you can further refine the current I/O pin assignment by one of the following methods:

- Manually adjust pins by direct pin manipulations or using pin editor.
- Use the `assignIoPins` command to further optimize I/O placement.

### ***Using the assignIoPins Command to Optimize I/O Placement***

The `assignIoPins` command assigns I/O pins based on placement information. The design must be placed before this command is run. The command supports:

- Rectilinear designs

- Non-uniform tracks
- User-specified constraints

By default, the `assignIoPins` command will honor fixed pins and only assign pins that have placed/unplaced placement status. If the initial I/O placement is generated by loading a constraint file (that is, the `loadIoFile` command automatically set I/O placement status to fixed) you should change I/O pins placement status to placed using [`setPtnPinStatus`](#) command before running I/O pin assignment.

To incrementally assign I/O pins, you can do one of the following:

- Specify pins that should be re-optimized using the `-pin` option.

Example: Re-assign all p\_address bus pins, int, and bio I/O pins of the design `tdsp_core`.

Optimize these specified pins even though they have fixed placement status.

```
assignIoPins -pin {p_address[*] int bio} -moveFixedPin
```

- Mark I/O pins that you want to keep with fixed status and run the `assignIoPins` command. This scenario can be used when you want to re-optimize most of I/O pins.

Example: Preserve `port_pad_data_in` and `port_pad_data_out` buses and clock pins, and re-optimize the rest.

```
setPtnPinStatus tdsp_core port_pad_data* fixed
```

```
setPtnPinStatus tdsp_core clk fixed
```

```
assignIoPins
```

## Validating Pin Placement

After assigning I/O pins, it is recommended that you check for I/O legalization.

Use the [`checkPinAssignment`](#) command to make sure that pins are legalized (such as they snap to routing grid, are on reserved routing layers, honor user-specified constraints, not cause any DRC violations, and so on).

You can check:

- All I/O pins

Example: Verify all I/O pins of the current design and output the result into the output file `pinLegality.rpt`.

```
checkPinAssignment -outFile pinLegality.rpt
```

- Specific I/O pins

Example: Verify all bus pins BG\_scan\_in, BG\_scan\_out, and the write pin of the design

```
checkPinAssignment -pin {BG_scan* write}
```

If any pin violation is detected, you can:

- Manually adjust pins by direct pin manipulation or using pin editor.
- Run the [legalizePin](#) command to automatically legalize pins. You can legalize all I/O pins or specific I/O pins of the design. Fixed pins will not be adjusted unless the -moveFixedPin option is specified.

Example1: `legalizePin`

With this example, the EDI System 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 EDI System 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 EDI System 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)  | <code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code>                                                                                                                           |
| 2      | PtnAware        | 42(223)  | <code>ram_256x16_test(DTMF_INST/RAM_256x16_TEST_INST)</code>                                                                                                               |
| 3      | PtnAware        | 32(223)  | <code>G1(DTMF_INST/G1_PH)</code>                                                                                                                                           |
| 4      | PtnAware        | 20(223)  | <code>results_conv(DTMF_INST/RESULTS_CONV_INST)</code><br><code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code><br><code>ram_128x16_test(DTMF_INST/RAM_128x16_TEST_INST)</code> |
| 5      | Normal          | 18(223)  | <code>ram_128x16_test(DTMF_INST/RAM_128x16_TEST_INST)</code>                                                                                                               |
| 6      | Normal          | 15(223)  | <code>results_conv(DTMF_INST/RESULTS_CONV_INST)</code>                                                                                                                     |
| 7      | Normal          | 3(223)   | <code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code><br><code>ram_128x16_test(DTMF_INST/RAM_128x16_TEST_INST)</code>                                                           |

|    |        |        |                                                                                                                                                                        |
|----|--------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 8  | Normal | 2(223) | G1(DTMF_INST/G1_PH)<br>results_conv(DTMF_INST/RESULTS_CONV_INST)<br>tdsp_core(DTMF_INST/TDSP_CORE_INST)                                                                |
| 9  | Normal | 1(223) | G1(DTMF_INST/G1_PH)<br><br>tdsp_core(DTMF_INST/TDSP_CORE_INST)                                                                                                         |
| 10 | Normal | 1(223) | G1(DTMF_INST/G1_PH)<br>results_conv(DTMF_INST/RESULTS_CONV_INST)                                                                                                       |
| 11 | Normal | 1(223) | G1(DTMF_INST/G1_PH)<br>tdsp_core(DTMF_INST/TDSP_CORE_INST)                                                                                                             |
| 12 | Normal | 1(223) | G1(DTMF_INST/G1_PH)<br>results_conv(DTMF_INST/RESULTS_CONV_INST)<br>ram_256x16_test(DTMF_INST/RAM_256x16_TEST_INST)<br>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:

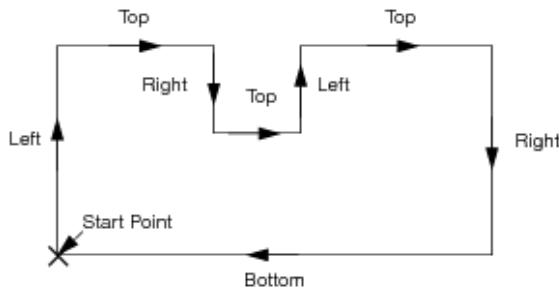
- The routing topology generated by the `trialRoute -fastRouteForPinAssign` command should be used only for the pin assignment flow.
- The net groups are sorted in descending number of nets in them.
- The net groups that have a significant number of inter-partition nets are routed in a partition-aware 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.

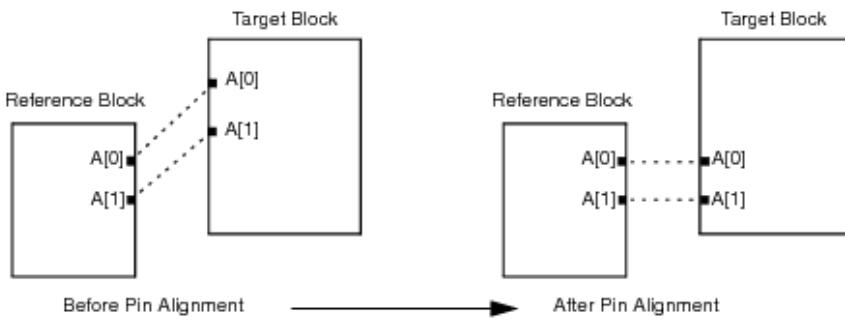
## Swapping Partition Pins

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:

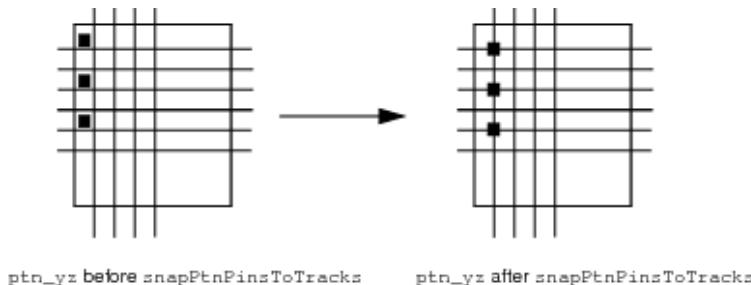
```
pinAlignment -block blockB -refBlock blockA {A0 A1}
```



## Snapping Pins to the Grid

To snap center of pins to nearest intersecting routing grid, where the horizontal and vertical routing tracks cross, use the [snapPtnPinsToTracks](#) text command. For example, the following command snaps center of partition ptn\_xy pins to the nearest intersecting routing grid:

```
snapPtnPinsToTracks ptn_yz
```



## 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.

- i** The bus guide feature is intended to guide partition pins and blackbox pins and *not* I/O pins. The I/O pin assignment feature ([assignToPins](#) 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.

- i** Before inserting feedthroughs, you should determine what stage the design is in, such as prototyping, intermediate, tapeout, and set the appropriate global options by running the `setMode` commands, such as `setPlaceMode` and `setTrialRouteMode`. For example, when inserting feedthroughs during prototyping, you could set modes with the following commands:

```
setPlaceMode -fp
setTrialRouteMode -floorplanMode true
setExtractMode -default
```

You can use the `insertPtnFeedthrough` command (or the [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 EDI System 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` text command inserts routing feedthroughs into the partitions without changing the design netlist. This command affects the design in the following areas:

- Manages only the physical aspect of a partition, not the logical aspect.
- No new ports are added to a partition and no buffers are added to the partition netlist.
- For channel feedthroughs, this creates channels for over-the-block routing on specified layers at the top-level design. These channels are pushed down as routing blockages on the correct routing layers at the partition level during Partitioning.
- For placement island feedthroughs, the EDI System software reserves these areas for inserting buffers at the top-level design after running the `insertRepeater` command. These island feedthroughs will be pushed down as placement blockages and routing blockages on all routing layers at the partition level during partitioning.

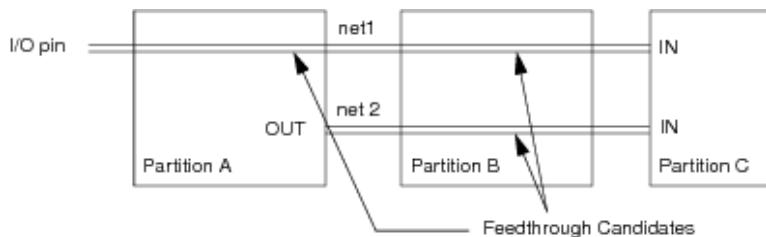
## Inserting Feedthrough Buffers

Partition feedthrough insertion manages partitioned designs that have nets that need to be pushed down to become a component of each partition design. That is, each feedthrough buffer must be added to the partitioned design, which changes the partition's netlist. This approach is typically used in channelless designs and in designs with limited channel resources.

A pure channelless design has no channel routing resource--connections among partitions are always done by means of module abutment and pin alignment. A mixed or partially channelless

design has limited routing resource in the channels; therefore, abutment and pin alignment is only performed on selected nets.

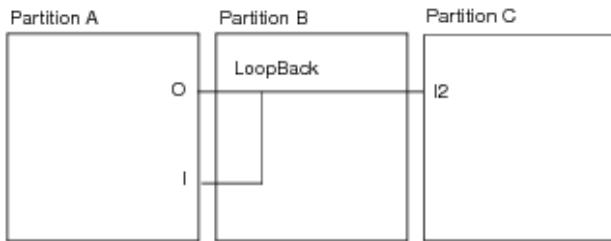
The following example shows how nets are selected for feedthrough buffers:



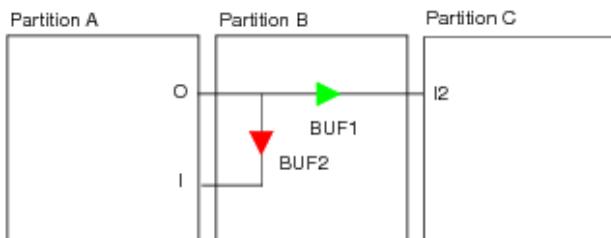
### Inserting Feedback Buffers

You can insert a feedthrough buffer to a net that loops back to an original partition to avoid the net routing over a partition area using the `insertPtnFeedBackBuffer` text command, which you should run after the feedthrough insertion step.

The following example shows a situation where net LoopBack connects to output pin `o` and input pin `I` of Partition A, and input `I2` of Partition C.



By inserting a feedthrough buffer (BUF1) with the `insertPtnFeedBackBuffer` text command, and inserting a feedback buffer (BUF2) with the `insertPtnFeedBackBuffer` text command, LoopBack now connects to the input pins of BUF1 and BUF2, as shown in the following figure:



### Limitations

- Each partition must be intact. A non-child instance cannot be preplaced in another partition. This would present a top-level net connection problem.

- Partition pin guides cannot be used during feedthrough insertion.
- A partition design that has repeated partition modules is not supported. Exclude all nets that connect into a repeated partition module.
- The Unpartition program cannot remove the inserted buffers for the feedthrough nets.
- Does not handle blackboxes.
- It might not handle clock nets efficiently because the `insertPtnFeedthrough` text command does not take timing into account.
- It cannot handle nets that are connected to two or more glue logic standard cells. This type of net should be excluded from feedthrough insertion.
- It might not provide good quality of results for high fanout nets. You should exclude high fanout nets and clock nets from feedthrough insertion to avoid timing and routing problems.

#### Procedure

1. Design the top-level floorplan for the partition design.
2. Run Placement.
3. (Optional) Run Trial Route.

**i** 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 `insertPtnFeedBackBuffer` text command's `-chanLess` option.

4. Create a file to identify which nets get buffers.

You can manually edit the file, create a script, or generate a wire crossing file (see [Generating the Wire Crossing Report](#)).

5. Generate the feedthrough buffers and nets.

Use the `insetPtnFeedthrough -chanLess` command, or `insetPtnFeedthrough -selectNet` with the created net file.

**Note:** Step 6 returns to the normal partition design flow.

6. Run Trial Route to completely connect the design, including the inserted feedthrough buffers.
7. Run Partition to generate the partition pins and change the partition module status to hard block.
8. Run Save Partition.

This saves the design and generates a top-level directory and partition directories.

#### Using a Topology File to Insert Feedthrough Buffers

You can guide the insertion of feedthrough buffers for specific nets by providing the feedthrough topology information for those nets in a topology file. You can manually create this file and subsequently edit it.

**Note:** If you are using topology files from releases prior to the 8.1 release, they will still work with this release.

**Note:** The syntax is case sensitive.

The syntax of the topology information in the file is as follows.

```
# Comment line

version version_string;

nametype netname

    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);

    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);

.

.

.

end nametype

nametype netname

    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);

    fromtype-totype from_name to_name route_data=(x,x,x,x,x,x);

.

.

.

end nametype
```

The description of the syntax is as follows

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nametype | <p>Can be net, bus, or netgroup. The value netgroup represents all nets in the net group. You should update the net group after feedthrough insertion step.</p> <p>Here are some examples of nametype:</p> <ul style="list-style-type: none"> <li>■ bus myBus[0:1] specifies bus bits</li> <li>■ net myBus[0:1] specifies a scalar net.</li> <li>■ bus myBus[1] specifies a bus bit</li> <li>■ net mybus[1] specifies a scalar net or a bus bit. In case both exist in the design, use the Verilog escape name and use the <code>dbgIsBackslashInNamesHiddenFlag</code> variable to resolve correctly.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| netname  | <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>■ io for I/O pins</li> <li>■ hinst for hierarchical instance name of a partition or partition clone</li> <li>■ instterm for top-level instance pins</li> </ul>                                                                                                                                                                                                                                           |
| totype   | <p>Can have one of the following values:</p> <ul style="list-style-type: none"> <li>■ io for I/O pins</li> <li>■ hinst for hierarchical instance name of a partition or partition clone</li> <li>■ instterm for top-level instance pins</li> <li>■ hinstfb for hierarchical instance name of a partition or partition clone. This can only be used as part of the combination hinst-hinstfb, which specifies a feedback buffer path.</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 insertPtnFeedthrough 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

#####
```

### Replicating Feedthrough Insertions Across ECO Netlists

While performing a feedthrough insertion through the [insertPtnFeedthrough](#) command, you can save the feedthrough buffer topology tree information in a file by specifying the `-saveTopoFile` parameter as follows:

```
insertPtnFeedthrough -saveTopoFile TopoFileName
```

where `TopoFileName` is the name of the file in which topology information is saved.

When you run the `insertPtnFeedthrough` command on another ECO netlist, you can use the saved file to replicate feedthrough buffer insertions by specifying the `-topoFile` parameter as follows:

```
insertPtnFeedthrough -topoFile SavedTopoFileName
```

where `SavedTopoFileName` is the name of the file that was saved earlier using the `-saveTopoFile` parameter.

This way, you can save a file with feedthrough buffer topology tree information and use it to create the same feedthrough buffer insertions across multiple netlists.

The flow can be summarized as follows:

1. Import a design.
2. Perform floorplanning on the design.
3. Perform feedthrough buffer insertion and save the feedthrough buffer topology tree information in a file (use the `-saveTopoFile` parameter of the `insertPtnFeedthrough` command).
4. Import design with a new ECO netlist.

**Note:** The ECO netlist should not contain the original inserted feedthrough buffers.

5. Perform feedthrough buffer insertion with the topology file saved from step 3 (use the -  
`saveTopoFile` parameter of the `insertPtnFeedthrough` command).

**Note:** If you use the `-topoFile` parameter, only those nets that are specified in the topology file are considered for feedthrough buffer insertion.

**Note:** If a net does not exist in the design, it should not be in the topology file. For example, if ECO changes remove a net, that net should be removed from the topology file.

6. Repeat steps 4 and 5 for more ECO netlists, if required.

#### Reducing the Number of Buffers and Ports Added for Route-based Feedthrough Insertions

You can use the `-reduceAddedPort` parameter of the `insertPtnFeedthrough` command to specify that feedthrough insertion should follow the routing topology more closely. This can help reduce the number of added ports and buffers.

The ports are created at the route crossing points. The status of the added ports is set to *Fixed*. Subsequent use of Trial Route will make the routes pass through these pins. Therefore, there is no need to create partition pin guides for these pins.

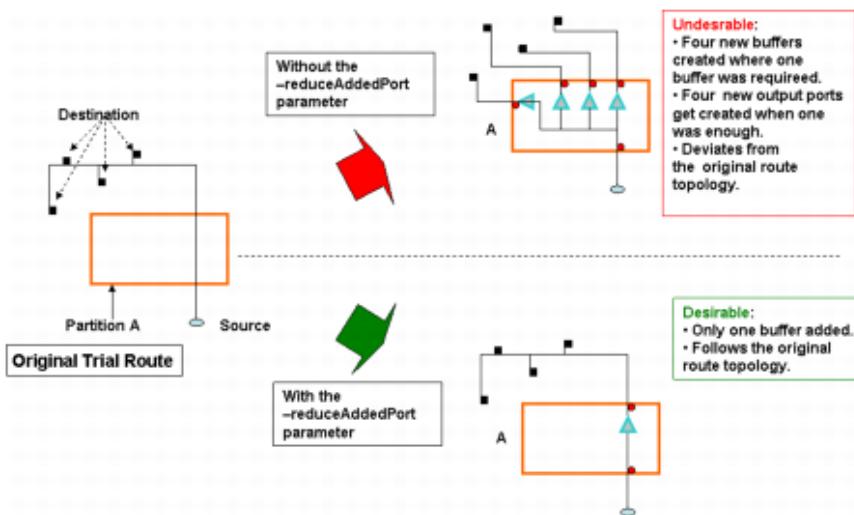
**Note:** The `-reduceAddedPort` parameter is applicable only for route-based feedthrough insertions.

This behavior is illustrated through the following scenarios:

- Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels:
- Net Connecting Through Adjoining Partition

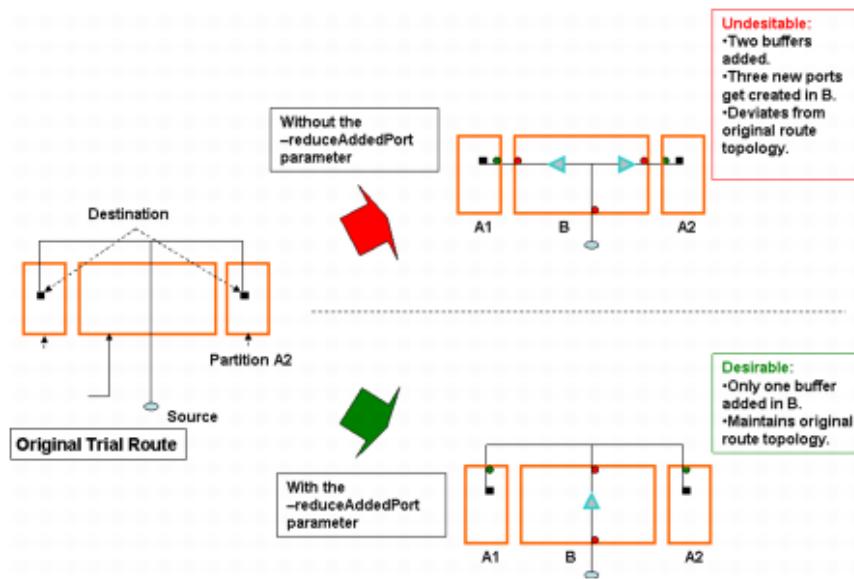
#### ***Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels***

The following diagram illustrates the improvement in feedthrough insertion where a net connects to a non-partition instance terminals in the top-level routing channels.



### Net Connecting Through Adjoining Partition

The following diagram illustrates the improvement in feedthrough insertion between partitions where there is another partition in between.



### Abbreviating Lengthy Feedthrough Net Names

You can abbreviate inserted feedthrough net names so that the net names will not extend too long if you run the `insertPtnFeedthrough` or `insertPtnFeedBackBuffer` commands multiple times. With the `-useShortName` option, you can eliminate the use of the old net name and partition names, and instead use a running count for the new net names.

For example, if a feedthrough net reset connects two partitions `tt_chiplet` and `video_chiplet`, the

feedthrough net name is:

FE\_FEEDX\_NET\_C\_tt\_chiplet\_video\_chiplet\_reset

The net name abbreviation convention for feedthrough buffer insertion when using the `insertPtnFeedthrough -useShortName` command are:

|              |                                                 |
|--------------|-------------------------------------------------|
| Net Names    | FE_FTn_ <i>n</i> , where <i>n</i> is an integer |
| Buffer Names | FE_FTb_ <i>n</i> , where <i>n</i> is an integer |

The net name abbreviation convention for feedback buffer insertion when using the `insertPtnFeedBackBuffer -useShortName` command are:

|              |                                                    |
|--------------|----------------------------------------------------|
| Net Names    | FE_FB_NET_ <i>n</i> , where <i>n</i> is an integer |
| Buffer Names | FE_FB_BUF_ <i>n</i> , 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.

**i** 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`

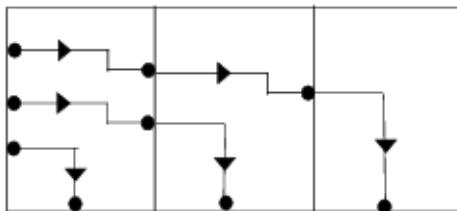
where `x` is a unique numeric identifier and `netName` is the name of the original net.

- 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](#).

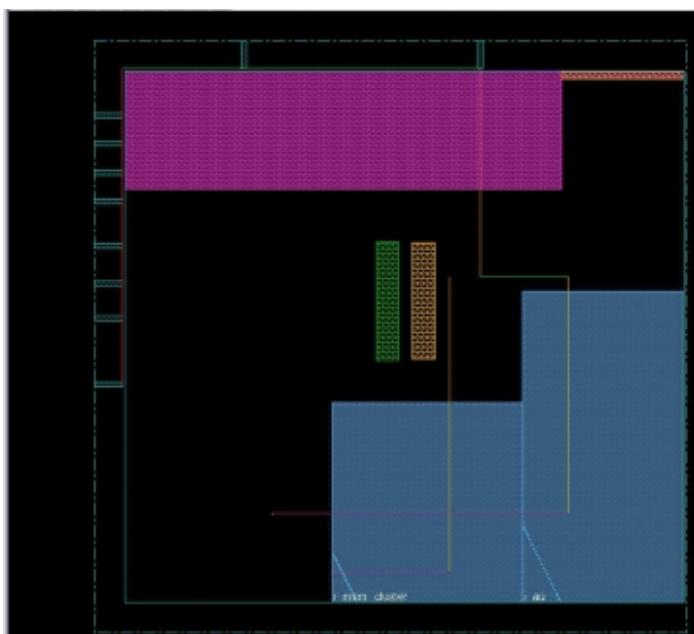
#### ***Feedthrough Connection for Abutted Macros***

For abutted custom macros, the [`connectMacroFeedthrough`](#) command detects the paths formed by the abutted feedthrough pins. The EDI System software considers only the end points of the detected paths, and picks those feedthroughs that will give good results.

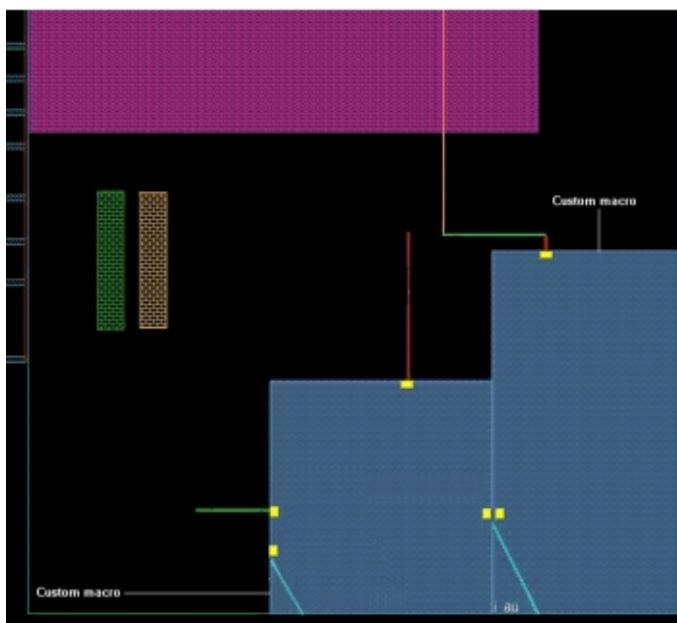
The following figures show how EDI System selects the feedthroughs for insertion in the abutted custom macros.



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 [connectMacroFeedthrough](#) command. If a mapping file is not specified with the [connectMacroFeedthrough](#) command, EDI System 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 (I<sub>N</sub>O<sub>T</sub>) 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.

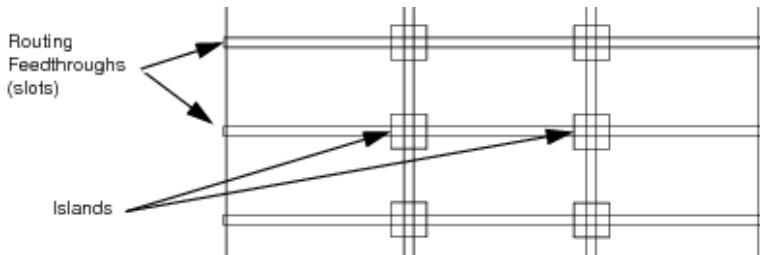
### **Inserting Routing Feedthroughs**

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.

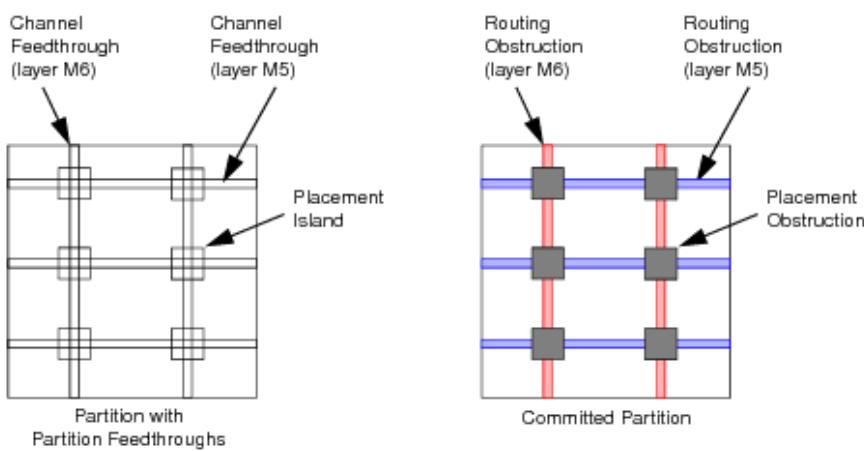
**Method 2:** If you want to specify narrow feedthroughs or several of them on a given partition, choose *Partition-Create Physical Feedthroughs* to open the *Create Physical Feedthrough* form. To specify which partition you want, click on the partition in the design display area, then click *get selected*. 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.

- 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 EDI System software determines the buffer tree topology, so not all specified nets will receive inserted feedthroughs. For example, nets that connect directly between adjacent partitions are not candidates for feedthrough insertion.

## Interpreting the Wire Crossing Report

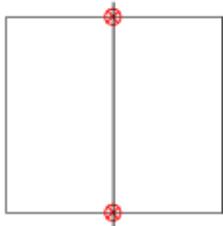
The wire crossing report section lists the nets, their wire lengths, in micrometers, and the shape of the wire in relation to the partition. For example, the following report segment is for a partition module named ptn01:

```
#####
# Nets that cross partition module ptn01
# Box (335 335) (833 567)
# Format: Net <netName> <wireLength> <shape>
#####

Net A 65 I
Net B 80 L
Net C 1050 T
Net D 132 X
...
```

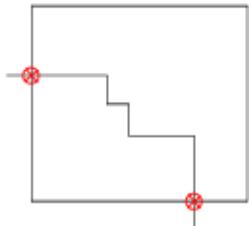
The first net in the report, A, has a wire length of 65 micrometers in an 'I' shape, which indicates that the net crosses the partition on opposite sides, as follows:

Net A 65 I



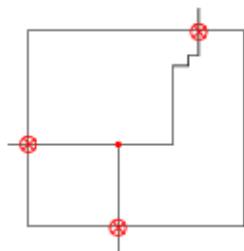
The second net in the report, B, has a wire length of 80 micrometers in an 'L' shape, which indicates that the net crosses the partition on adjacent sides, as follows:

Net B 80 L



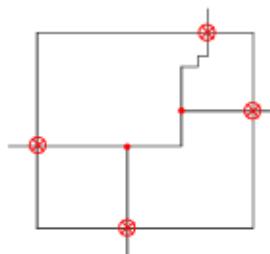
The third net in the report, c, has a wire length of 105 micrometers in an 'T' shape, which indicates that the net crosses the partition on three sides, as follows:

Net C 105 T



The fourth net in the report, d, has a wire length of 132.30 micrometers in an 'x' shape, which indicates that the net crosses the partition on all four sides, as follows:

Net D 132 X



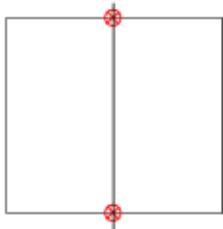
In the report, you can also include the total length of the wire crossing the block in the horizontal X direction and total length of the wire crossing the block in the vertical Y direction using the -delta option of the showPtnWireX command. For example, the following report segment is for the same partition module named ptn01 using the - delta option:

```
#####
# Nets that cross partition module ptn01
# Box (335 335) (833 567)
# Format: Net <netName> <wireLength> <shape> <deltaX> <deltaY>
#####

Net A 65 I 0 65
Net B 80 L 38 47
...
```

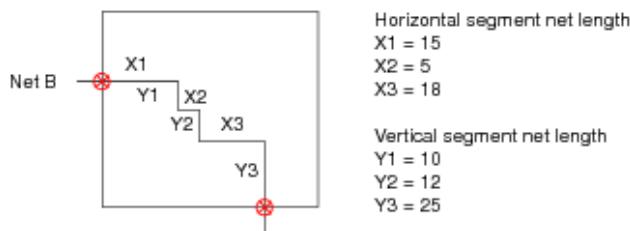
The first net in the report, A, has a wire length of 65 micrometers in an 'I' shape, with a total of 0 length in the horizontal X direction, and 65 in the vertical Y direction:

Net A 65 I 0 65



The second net in the report, B, has a wire length of 80 micrometers in an 'L' shape, with a total of 38 length in the horizontal X direction, and 47 in the vertical Y direction:

Net B 80 L 38 47



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 EDI System 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.

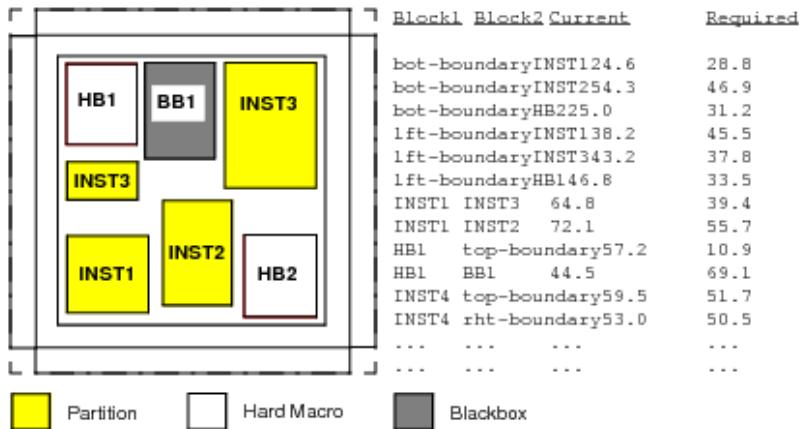
Channel width estimation also considers topology constraints to drive block placement. These constraints are block-to-block boundary, block-to-block distance, block order and alignment, block aspect ratio, net weight (from global trialRoute), and block halo. The channel width estimator also respects 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 EDI System 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.

**Tip:** 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 EDI System 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.

- i** 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.

- i** 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.

- i** 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> </ul> </li> </ul> |

|                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                | <ul style="list-style-type: none"> <li>▪ On layers reserved for partition, the stripe is pushed down to the block-level design.</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 EDI System prior to 6.1, the area I/O cells had to be part of the top-level netlist; otherwise, DRC violations were reported during block implementation. From the 6.1 release onwards, the area I/O cells can be at the top level or be a part of the partition netlist. This section describes the behavior for both the cases.

After the partition, LEF obstruction is cut against the overlapping bumps at the top. This is done for all the bumps (power/gnd/signal/unused). Similarly the routing blockages inside the partition is cut against the pushed down bump.

The following scenarios are discussed:

- [Area I/O Cells are Part of the Top-level Netlist](#)
- [Area I/O Cells are Part of the Partition Netlist :](#)
  - [Bumps and Routing are on Top Routing Layer--Behavior with the stripStayOnTop parameter](#)
  - [Bumps and Routing are on Reserved Routing Layer--Behavior with the stripStayOnTop parameter](#)
  - [Bumps and Routing are on Top Routing Layer--Default Behavior](#)
  - [Bumps and Routing are on Reserved Routing Layer--Default Behavior](#)

#### **Area I/O Cells are Part of the Top-level Netlist**

When area I/O cells are part of the top-level netlist, signal bumps and routes remain bumps and wires at the top level, but become routing blockages at the partition level. This allows routing at the block level while preserving the space for the signal bumps and routes.

Power and ground bumps and routes are copied and pasted (duplicated) from the top level to the partition. This allows power analysis at the block level. When the design is flattened, the duplicate power and ground bumps and routes are removed from the block level.

|                       | <b>Top Level</b> | <b>Partition Level</b>         |
|-----------------------|------------------|--------------------------------|
| Area I/O cell         | Area I/O cell    | Placement and Routing Blockage |
| Signal bump           | Signal bump      | Placement and Routing Blockage |
| Signal route          | Signal Route     | Routing blockage               |
| Power and ground bump | Bump             | Bump (copied and pasted)       |

|                        |       |                           |
|------------------------|-------|---------------------------|
| Power and ground route | Route | Route (copied and pasted) |
|------------------------|-------|---------------------------|

## Area I/O Cells are Part of the Partition Netlist

When area I/O cells are part of the partition netlist, the pushdown behavior depends on:

- whether the `stripStayOnTop` parameter has been specified with the [partition](#) command.
- whether the bumps and routing are on the top routing layer or the reserved routing layer

**ⓘ** In this case (that is, area I/O cells are part of the partition netlist), the behavior applicable to area I/O cells is also applicable to any other instance to which the bump is logically connected.

If the area I/O cell and the bump connection pass through a partition pin, the pin will not be assigned when you assign partition pins. These partition pins are assigned only when you run the [partition](#) command. If the bump overlaps the partition, a partition pin is created, with a geometry similar to that of the bump. If the bump does not overlap the partition, the pin is created during special route pushdown. The pin is created on the partition boundary where the routes between the bump and the area I/O cross the partition boundary.

For floating partition pins that are connected to a bump, the [assignPtnPin](#) command will check if the bump physically overlaps with the partition. If so, the command will not assign the pin and a partition pin is created, with a geometry similar to that of the bump, only when the [partition](#) command is run. Otherwise, the pin is assigned on the partition boundary by [assignPtnPin](#) command.

The following sections discuss the behavior for the following cases:

- [Bumps and Routing are on Top Routing Layer--Behavior with the stripStayOnTop parameter](#)
- [Bumps and Routing are on Reserved Routing Layer--Behavior with the stripStayOnTop parameter](#)
- [Bumps and Routing are on Top Routing Layer--Default Behavior](#)
- [Bumps and Routing are on Reserved Routing Layer--Default Behavior](#)

**Note:** For all the listed scenarios, the push down behavior for signal routes is similar to the behavior described in the [How Top-level Stripes Are Pushed Down](#).

### ***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:<br/>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:<br/>The signal bump is pushed down as a routing blockage.</li> </ul>                                                                                                                                                                                                                               |
| Signal Routes | Signal Routes routed on the top routing layers stays at top.                                                                                               | <ul style="list-style-type: none"> <li>▪ If the signal route overlaps the partition and is also connected to a area I/O cell inside the overlapping partition and a signal bump at the top, the signal route is copied and pasted to the partition. The pushed down net will be the internal net in the partition and will be named based on the partition port it is connected to inside the partition.</li> <li>▪ If the signal route overlaps the partition to which it is not connected (that is, it is not connected to any instance inside the partition but to a bump at top), these routes are copied and pasted as routing blockages inside the overlapping partition.</li> </ul> |

#### **Bumps and Routing are on Reserved Routing Layer--Behavior with the `stripStayOnTop` parameter**

The following table summarizes the behavior when the bumps and the routing are on the reserved routing layer and you run the [partition](#) command with the `-stripStayOnTop` parameter.

| Object Type   | Top Level                                                                                           | Partition Level                                                                        |
|---------------|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Area I/O cell | Not applicable because area I/O cell is already part of the partition netlist.                      | Area I/O cell is retained in the partition netlist.                                    |
| Signal bump   | Signal bump stays on top and, additionally, an equivalent pin is created in the partition LEF file. | Bumps get pushed down to the partition as an equivalent pin in the partition DEF file. |
| Signal route  | Signal routes are removed from the top.                                                             | Routing gets pushed down inside the partition block                                    |

#### ***Bumps and Routing are on Top Routing Layer--Default Behavior***

The following table summarizes the default behavior when the bumps and the routing are on the top routing layer. The default behavior in this context refers to the behavior that occurs when you run the [partition](#) command *without* the -stripStayOnTop parameter.

| Object Type   | Top Level                                                                      | Partition Level                                                                                                                                                                                                                       |
|---------------|--------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Area I/O cell | Not applicable because area I/O cell is already part of the partition netlist. | Area I/O cell is retained in the partition netlist                                                                                                                                                                                    |
| 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<br><br>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.

| <b>Object Type</b> | <b>Top Level</b>                                                                       | <b>Partition Level</b>                                     |
|--------------------|----------------------------------------------------------------------------------------|------------------------------------------------------------|
| 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.

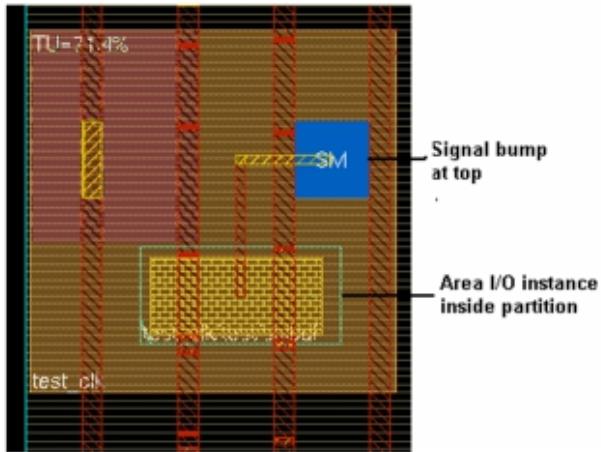
The following examples illustrate the behavior:

- [Case 1: All Routing Layers Reserved for the Partition](#)
- [Case 2: Top Layer Not Reserved for Routing](#)

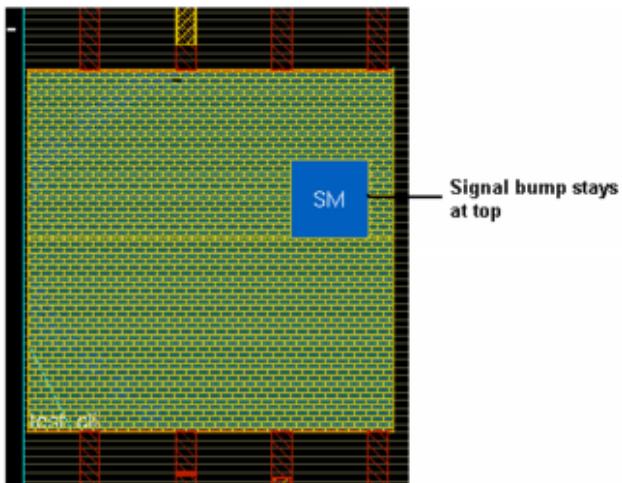
### ***Case 1: All Routing Layers Reserved for the Partition***

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.

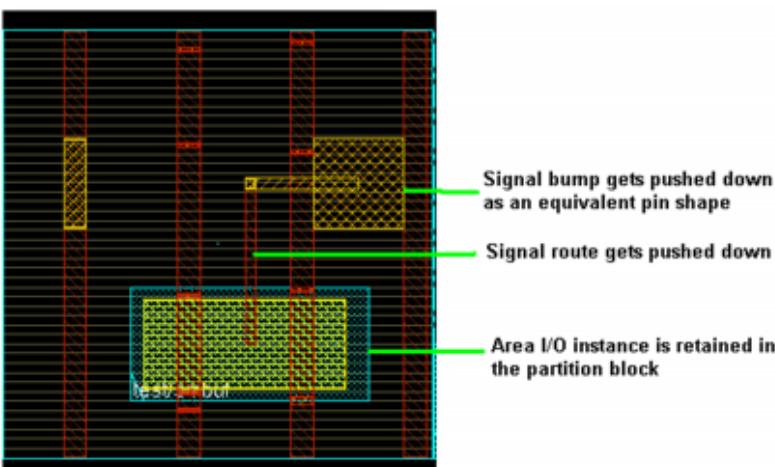
The following diagram shows the floorplan view before partitioning.



The following figure shows the view at top after partitioning.



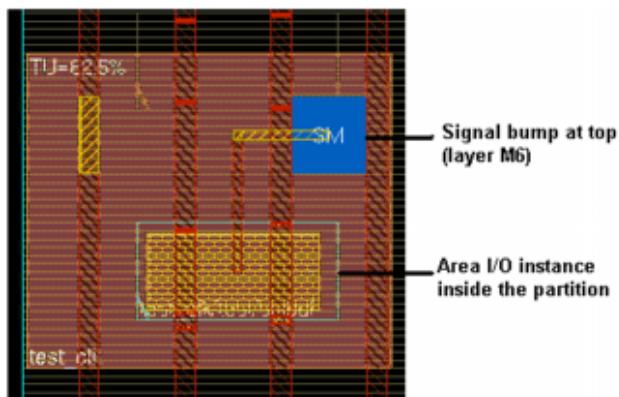
The following figure shows the view inside the partition



### Case 2: Top Layer Not Reserved for Routing

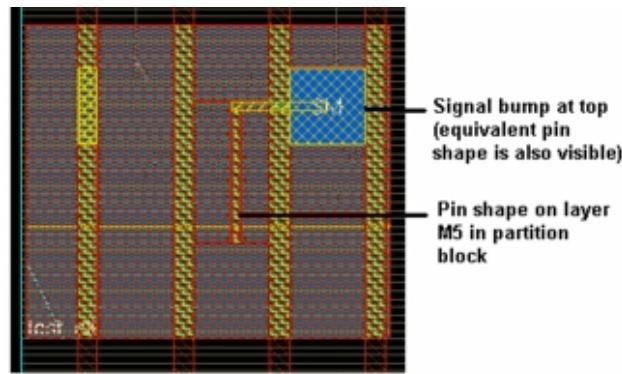
The design has six routing layers. Layers M1-M5 are reserved for the partition. M6 is the top routing layer.

The following diagram shows the floorplan view before partitioning.

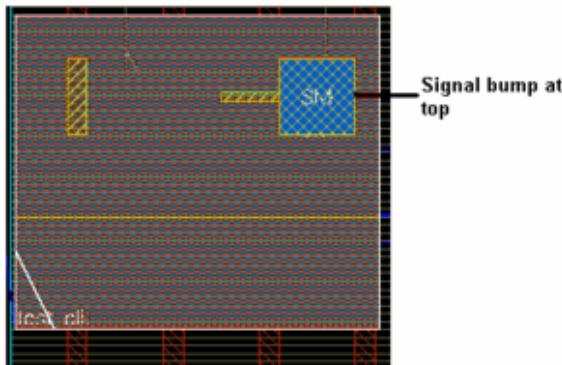


The following diagram shows the view at the top after partitioning with the `-stripStayOnTop` parameter specified.

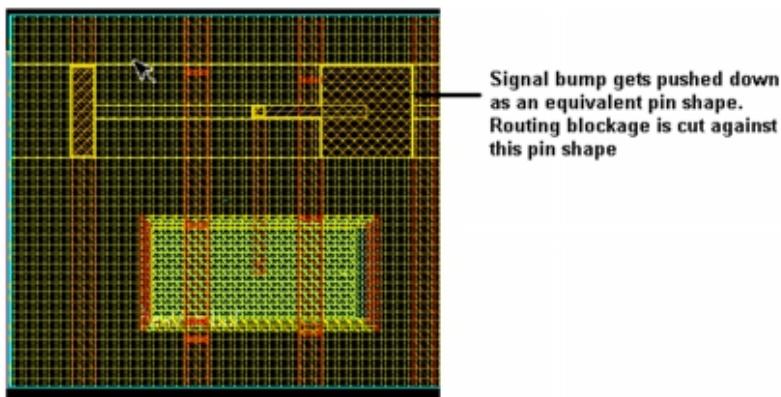
The following figure shows the view on the top after partitioning with the pins visible.



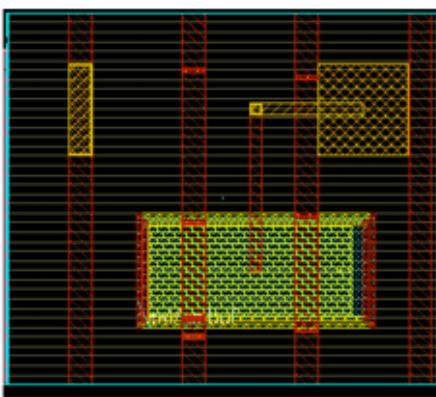
The following figure shows the view on the top after partitioning *without* the `-stripStayOnTop` parameter specified.



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](#)).

-  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 EDI System 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*.

-  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 EDI System 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 EDI System software. Files necessary to run back-end tools in DEF, PDEF, and TDF formats can be selected when saving partitions.

To save a partition, use the [Save - Partition](#) form or the [savePartition](#) text command.

 **Do not use the Save Design form to save a partition.**

You can also save partitions in the OpenAccess database format. For more information, see [Working with OpenAccess Database](#).

## Loading Partitions

After completing the design work for each partition and the top level, you can restore a partitioned design to the top level, which includes loading all the partition design directories and its data.

To restore a saved partition design, use the [Load - Partition](#) form.

## Unpartitioning with Routing Data

When loading a partition, it is important that the loaded routing results correctly correspond to the new netlist. To ensure that the netlist and routing file are consistent, you need to unpartition with the routing data using the following steps:

1. Load the original flat design.  
This is the original design before running the partition steps.
2. Specify the partition and save to a file (to be loaded later in step 7).
3. Run partitioning with pin assignment.
4. Save the partitions and the top level.

5. For each partition and the top-level, run the block-level implementation with the following commands:

- encounter
- restoreDesign (for the block or top level)
- trialRoute OR nanoroute OR wroute
- saveRoute

6. Load the original design (the same design loaded in step 1).

If the netlist has been modified after step 1 (for example, in the case where a netlist is modified after in-place optimization or running NanoRoute) use the updated netlist instead.

To specify the updated netlist, you must first specify top-level netlist, then the block-level netlists in the *Verilog Files* field of the Design Import form's *Design* page.

For example, `top.v block1.v block2.v ...`

 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 *EDI System Menu Reference* .
- [trialRoute](#)
- [setAllowedPinLayersOnEdge](#)

#### **Setting Pin Constraints**

[Inserting Feedthroughs](#)

# Floorplanning the Design

---

- [Overview](#)
- [Common Floorplanning Sequence](#)
- [Viewing the Floorplan](#)
- [Module Constraint Types](#)
- [Grouping Instances](#)
- [Creating and Editing Rows](#)
- [Using Vertical Rows](#)
- [Using Multiple-height Rows](#)
- [Performing I/O Row Based Pad Placement](#)
- [Editing Pins](#)
- [Running Relative Floorplanning](#)
- [Saving and Loading Floorplan Data](#)
- [Resizing the Floorplan](#)

## Overview

Floorplanning a chip or block is an important task of physical design in which the location, size, and shape of soft modules, and the placement of hard macros are decided. Depending on the design style or purpose, floorplanning can also include row creation, I/O pad or pin placement, bump assignment (flip chip), bus planning, power planning, and more. For example, floorplanning is very important when preparing the design for timing closure and detailed routing. Floorplanning, in conjunction with placement and trial routing, can be an iterative design process.

The Encounter® Digital Implementation System (EDI System) 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 *EDI System Text Command Reference* .
- For information on floorplan GUI, see the [Floorplan Menu](#) chapter, in the *EDI System Menu Reference* .

EDI System includes several keyboard shortcuts for use with the floorplanning feature. Make sure you type the bindkey while the main EDI System 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 EDI System 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 replacing blocks, modules, and submodules according to the prepared floorplan. All other modules or blocks not in the prepared floorplan are left outside the chip area.

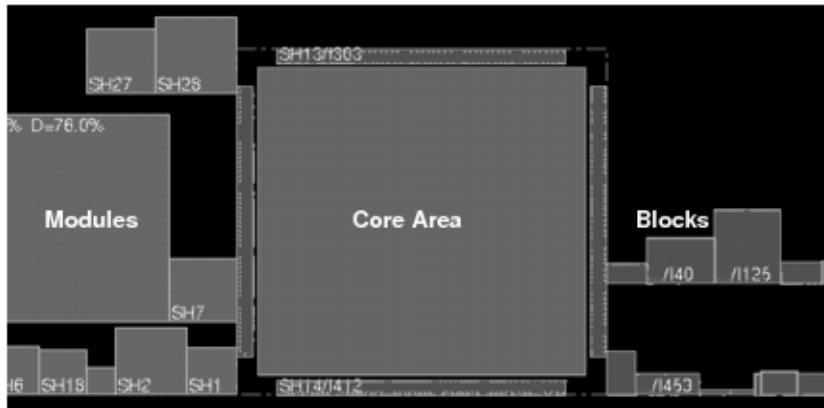
The following steps describe the most common sequence for floorplanning:

1. Importing the design.
2. Studying the design's connectivity.
3. Performing the minimum amount of floorplanning based on the chip design floorplan, or do no floorplanning at all.
4. In some cases, no floorplanning is required. For example, a front-end designer might want to predict the quality of the design's netlist by initially placing the entire design without any floorplanning. This iteration provides a good indication of how the blocks should be located and arranged together with the larger modules. After a few iterations, it should be clear how to position the blocks and modules in the floorplan.
5. Running placement and Trial Route to view placement and routing congestion.  
Optionally, running [resizeFP](#) to enlarge or shrink the die after placement and routing. See [Resizing the Floorplan](#).
6. In this case, floorplanning is done to detail the pre-placement of all blocks, most likely done by a back-end designer to gauge the feasibility of a prepared floorplan.
7. The placer places all remaining blocks that were not preplaced in the floorplan, and also recognizes the floorplan object, such as power and ground routes.
8. If you are at the design's top-level in the display area and want to generate a guide for a submodule, ungroup the top module until you have reached the submodule.
9. Using the full chip placement to refine block (hard macro and blackbox) locations.  
(Optional) Based on the full chip placement results--placement density and routing congestion, running resize floorplan to enlarge or shrink the die.
10. View the placements of blocks to determine if you need to change the alignment or orientations.
11. Looking for congestion in modules and change heavily congested modules' placement density to a lower percentage (using the [createDensityArea](#) text command).
12. (Optional) If you made any changes in step 5, or especially step 6, rerun placement.

- 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.

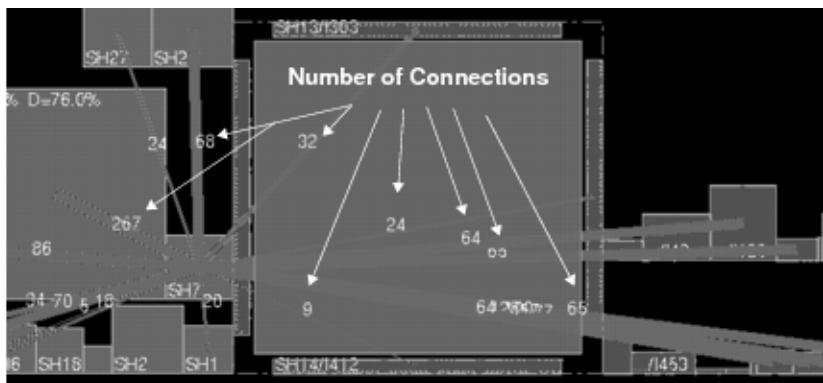
## Viewing the Floorplan

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.

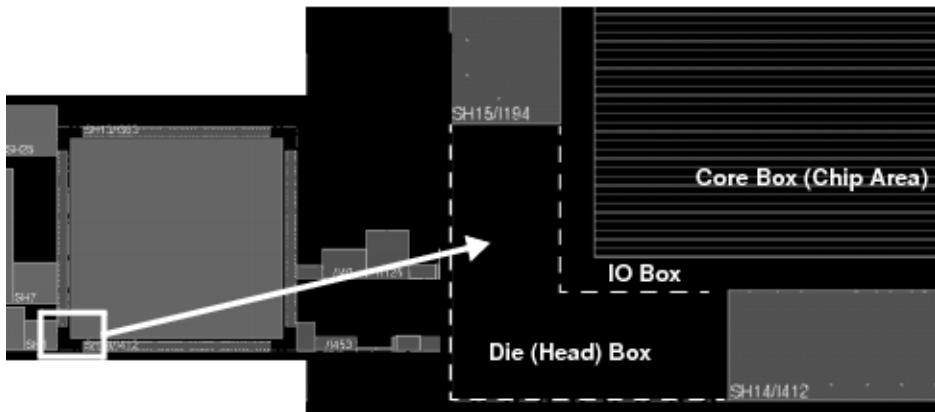
In Floorplan view, you can view the block pins and connection flight lines by clicking on a block or module. Flight lines show the connections and number of connections between the selected module or block to any other modules and blocks.



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 EDI System 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 are determined by either the core utilization or the core width and height specifications. The imported design modules can have one of the following constraint types:

- **None**--The module is not pre-placed in the core design area. The contents of the module are placed without any constraints.
- **Guide**--The module is preplaced in the core design area.  
A module guide represents the logical module structure of the netlist. The purpose of a module guide is to guide placement to place the cells of the module in the vicinity of the guide's location. The preplaced guide is a *soft* constraint, which is discussed later in this section. After the design is imported, but before floorplanning, you can locate module guides on the left side of the core area, which appear as pink objects (by default) in the Floorplan view.

When a module is preplaced in the core design area, it snaps to a standard cell row in the vertical direction and to a *metal 2* pitch in the horizontal direction (the default). This default can

be changed to snap to the manufacture grid (in the Preferences form's *Floorplan* page).

To create a guide for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the [createGuide](#) command or select *Guide* from the Attribute Editor's *Constraint Type* pulldown menu.

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 EDI System 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 EDI System software cannot move the cells out of the group.

- Soft Guide--This constraint is similar to a guide constraint except there are no fixed locations. This provides stronger grouping for the instances under the same soft guide. The soft guide constraint is not as restrictive as a fence or a region constraint, so some instances might be

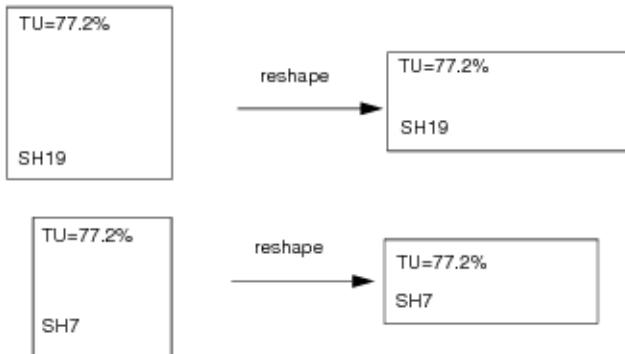
placed further away if they have connections to other modules.

To create a soft guide for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, select *SoftGuide* from the Attribute Editor's *Constraint Type* pulldown menu.

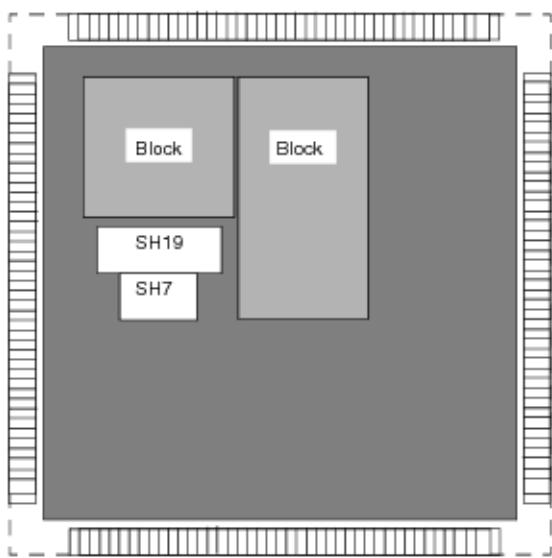
## Target Utilization Display

Module constraints display a target utilization (TU=%) value to represent their physical design size. This is an estimation of module utilization for the given size of the module where only standard cell and hard macro areas are considered; floorplan constraints, such as placement blockages, are not considered. This value is calculated by the standard cells area plus the hard macros area, divided by the module area. The initial TU values are calculated during design import.

The TU percentage helps judge the physical size of a module guide to customize the shape of the module in the floorplan. For example, modules SH19 and SH7 have a TU values of 77.2%. If the modules are reshaped with the same area, they retain their TU values, as shown in the following figure:



You can place them in the core area so they are preplaced close to one another, as shown in the following figure:



The position of the module guide is a placement constraint, and the final definition of the module is determined by several factors. The most important factor--the highest priority of constraint--is the connectivity between itself and other modules. Other floorplan constraints, such as neighboring preplaced module guides, preplaced blocks, placement blockages, and routing blockages, are also considered, but at a lower priority than connectivity.

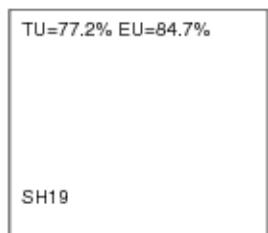
**Note:** You can use a stronger constraint for keeping modules SH19 and SH7 close together using the Group Instances form, and even a stronger constraint by saving the regrouped netlist.

Unlike module guides, the positions of fences and regions is a hard placement constraint and are not moved by the same factors.

## Effective Utilization Display

For fences and regions, you can display the effective utilization (EU=%) value. The EU value takes into account the actual cells and hard macros in the fence or region, placement or routing blockages, partition cuts, and other floorplan constraints. It is a good practice to update the EU value before running placement.

Click the *Display/Calculate Effective Utilization* toolbar widget (the % button above the design display area) to display the EU value for each fence and region, as shown in the following figure.



**Note:** The displayed EU values are not automatically updated. You must click the *Display/Calculate Effective Utilization* toolbar widget each time you want to display the updated EU value. This calculation could be time consuming, especially for larger designs.

**Note:** If the EU value is at or exceeds 100% for a fence or region, placement changes the fence or region to a guide. To avoid this, before you run placement, make sure to check and update the EU value, if necessary.

## Calculating Density

When specifying the floorplan, you can determine the core and module sizes by total density or standard cell density using the *Core Utilization* or *Std. Utilization* options, respectively, in the [Specify Floorplan](#) form.

Core Utilization determines the initial size of the core area and the initial size of the pink module guides off to the left of the die area. The total density is calculated as follows:

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

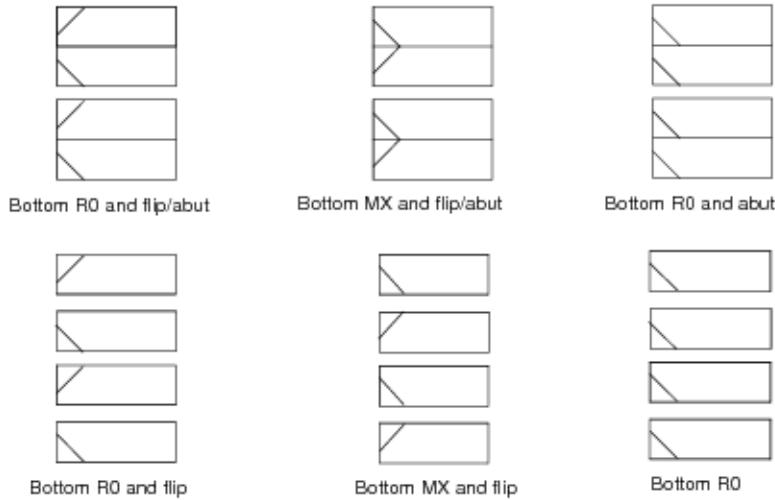
In determining the size of the core area and module guides, standard cells and hard macros are treated the same. However, you can determine how densely objects can be packed by weighing the standard cell density separately from the hard macro density. The standard cell density is calculated as follows:

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

The size of the core is smaller once you specified your floorplan by using *Std. Utilization*.

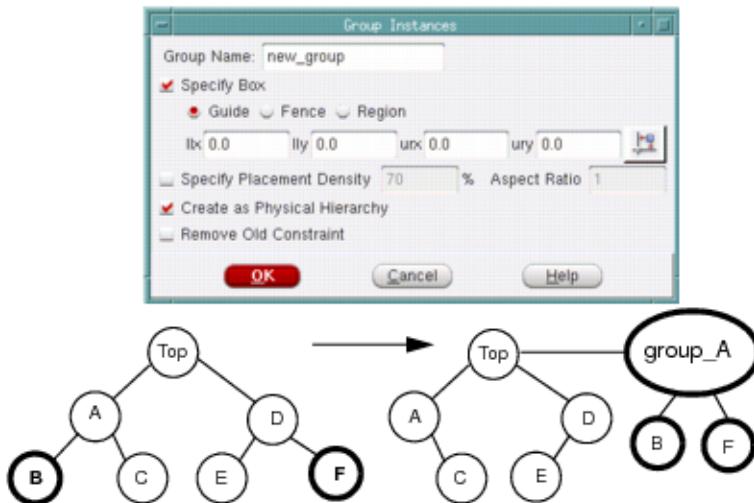
## Standard Row Spacing

To configure the rows, use the [`setFPlanRowSpacingAndType`](#) command, or the options from the *Standard Cells Rows* panel of the *Specify Floorplan* form. The following row configurations are supported:



## 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

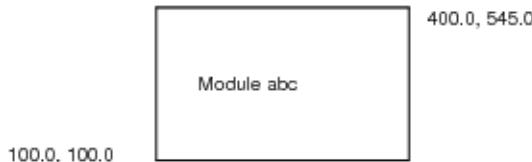
During floorplanning, you can use the `setObjFPlanBox` command to define a bounding box of a specified object, and the `setObjFPlanBoxList` command to define rectilinear shape of an object, which is comprised of two or more boxes.

This section provides graphical information to illustrate some of the command examples in the Floorplan Commands chapter of the *EDI System 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:

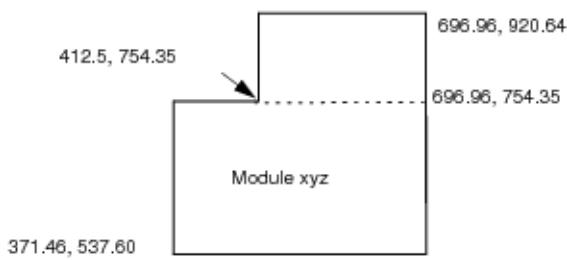
```
setObjFPlanBox Module abc 100.0 100.0 400.0 545.0
```



### `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
```



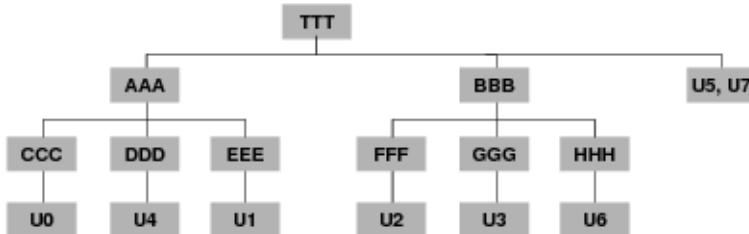
## Adding Logical Hierarchy Without Creating Additional Hierarchy

The EDI System 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 *EDI System 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

For more information, see the [runRcNetlistRestruct](#) command in the *EDI System Text Command Reference*.

#### Moving Instances to a New Top Module

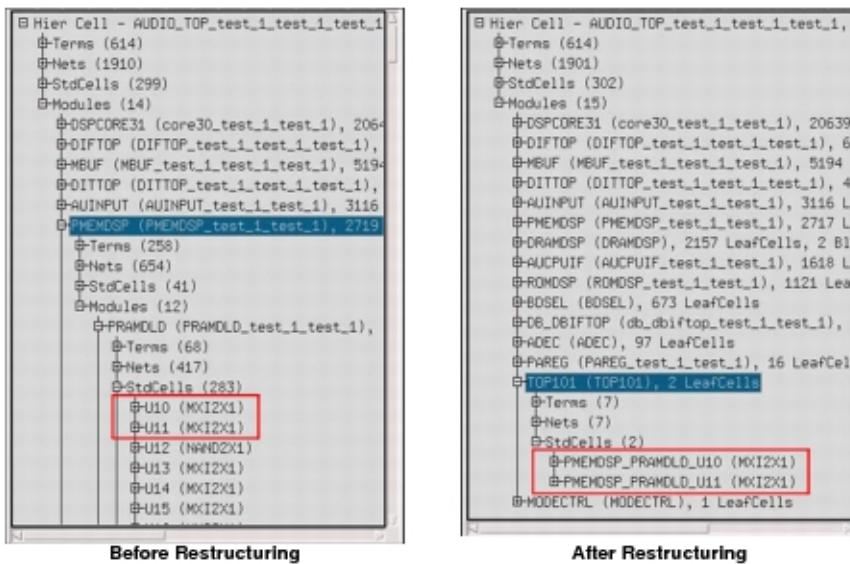
- To move an instance to a new top module named TOP101, you can do the following:

```
createInstGroup TOP101 -isPhyHier

addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U10

addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U11

runRcNetlistRestruct
```



#### Moving Instances to an Existing Module

- To move an instance to an existing module named DRAMDSP/DRAMDLD, you can do the following:

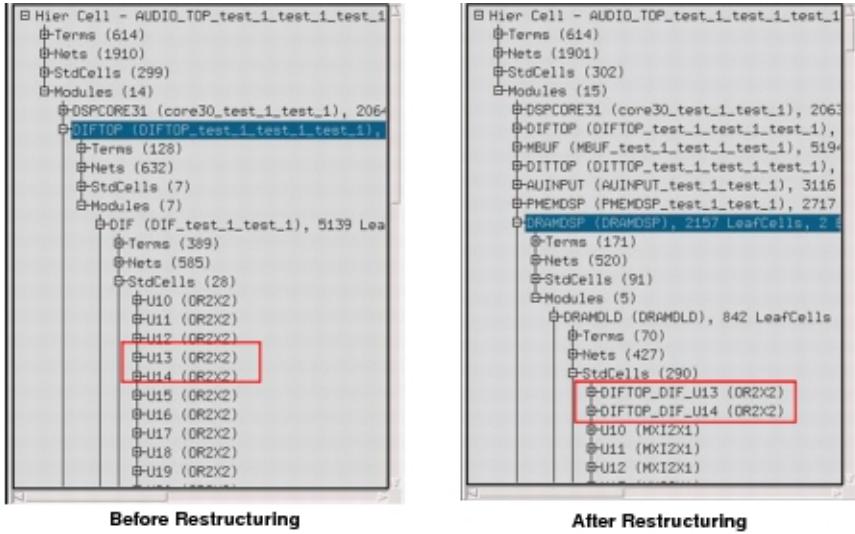
```
createInstGroup /DRAMDSP/DRAMDLD -isPhyHier

addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U13

addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U14
```

runRcNetlistRestruct

**Note:** The leading / (slash) is required for an existing module.

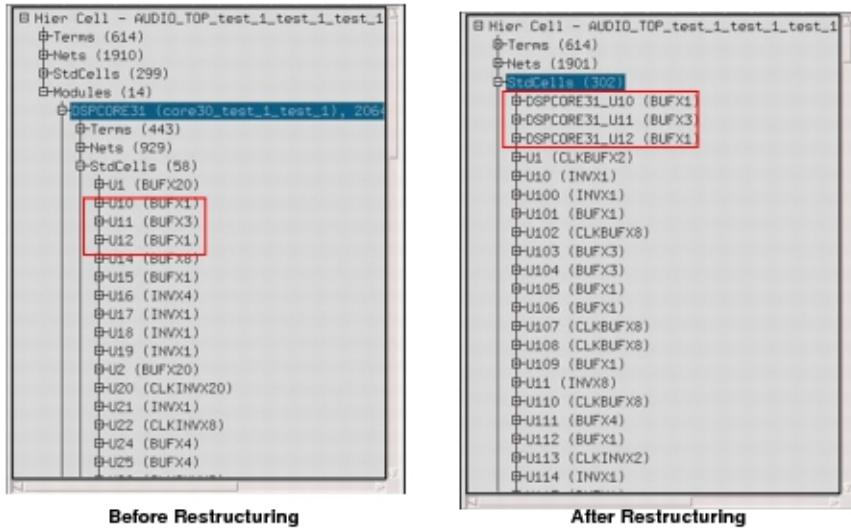


### Moving Instances to the Top Root Level

- 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.

|                             |                                                                                                                                                                                                                                                                     |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">createRow</a>   | 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. |
| <a href="#">cutRow</a>      | Cuts site rows that intersect with the specified area or object.                                                                                                                                                                                                    |
| <a href="#">deleteRow</a>   | Deletes the specified row(s).                                                                                                                                                                                                                                       |
| <a href="#">stretchRows</a> | Stretches selected rows. For example, you can specify that the left edge of all selected rows should be aligned to the left-most edge among all selected rows.                                                                                                      |

For more information on using these commands, see the *EDI System 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

**– 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, EDI System 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.

While specifying a floorplan, you can specify that the rows are vertical by using the `-verticalRow` parameter of the [floorPlan](#) command, or by specifying *Vertical Rows* in the *Row Direction* field of the [Specify Floorplan](#) form in the GUI.

The EDI System software generates vertical rows, provided the design data or the library meets the following prerequisites:

- The SITE width is more than the SITE height.
- Each SITE is stacked one above the other, when rows are created.
- MY and R0 row orientations are supported.
- The cells are stacked vertically, when they are placed and their power rails run vertically.
- The preferred direction of M1 is vertical.

### Limitations

The following limitations apply to vertical rows:

- Vertical rows cannot be created inside power domains.
- Non-integer and multiple integer vertical rows are not supported.
- Vertical rows cannot be created or edited directly. That is, the [createRow](#), [cutRow](#), [deleteRow](#), and [stretchRows](#) commands are not supported for vertical rows.

**Note:** You can, however, select rows to query, and delete rows with the delete key.

## Using Multiple-height Rows

In many cases, designs contain cells with different standard cell heights. For example, a design might utilize multiple standard cell libraries-possibly from different foundries or library vendors-which might have different standard cell heights.

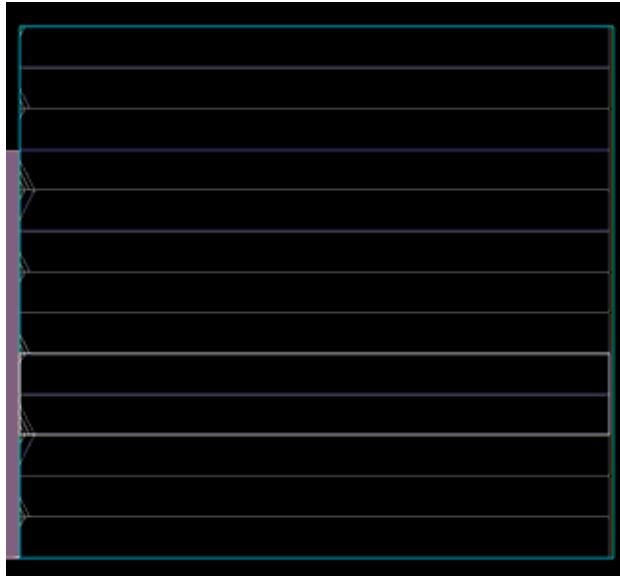
Standard cell designers create multiple-height standard cells for improving performance. Also, in a design with multiple power domains, standard cells with different voltages will probably have different footprints and different heights.

The EDI System 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 EDI System 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

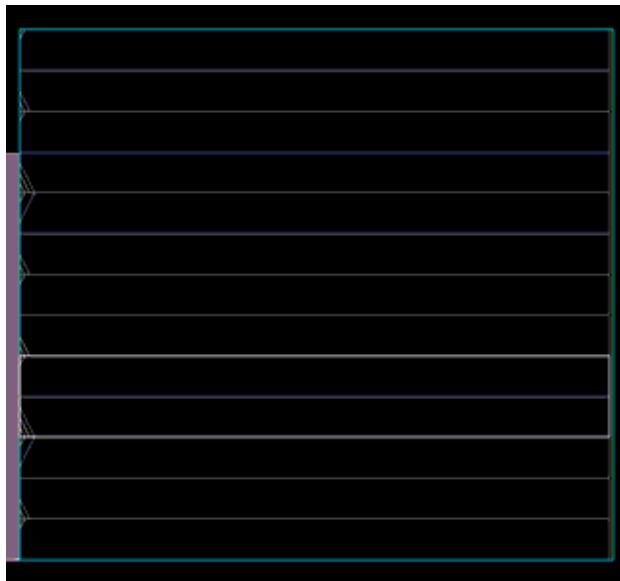
SITE doubleHeightSite
      SYMMETRY X Y ;
      CLASS CORE ;
      SIZE 0.660 BY 10.080 ;
END doubleHeightSite

MACRO DFFX64
      CLASS CORE ;
      FOREIGN DFFX64 0 0 ;
      ORIGIN 0 0 ;
      SIZE 21.12 BY 10.080 ;
      SYMMETRY X Y ;
      SITE doubleHeightSite ;
...
END DFFX64
```

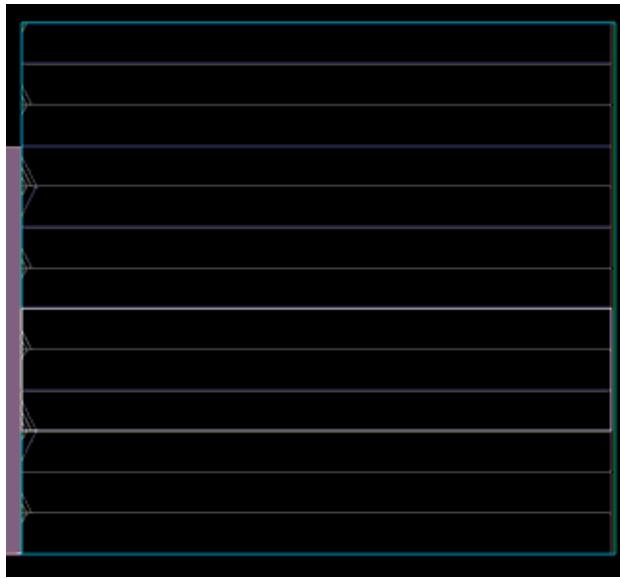
When you create integer multiple-height rows, the rows are automatically aligned with the single-height row. You cannot create unaligned integer multiple-height rows.

For information on creating and editing rows, see [Creating and Editing Rows](#)

The following figure illustrates a double-height row flipped to align with the orientation of the single row.



The following figure illustrates a triple-height row flipped to align with the orientation of the single row.



## Using Non-Integer Multiple-height Rows

You can also use non-integer multiple-height (NIMH) rows in your designs.

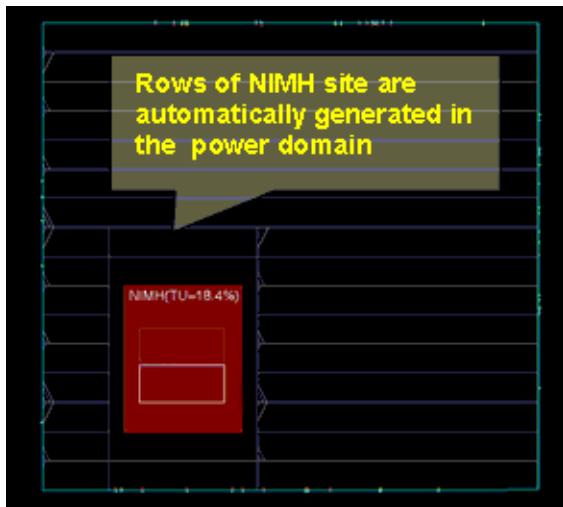
While creating NIMH rows, ensure the following:

- NIMH rows must be created *only* in those areas that have power domains associated with them.
- any newly created NIMH rows in an area must be an integer multiple of any existing rows in the area.

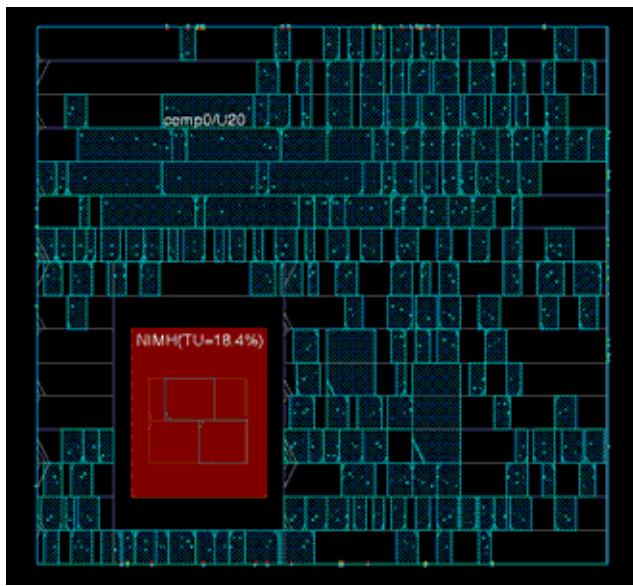
Each hierarchical instance to be declared as a power domain can only have one type of NIMH standard cells. In other words, NIMH rows inside any particular power domain must have the same height. Multiple types of NIMH standard cells require multiple power domains to be created. For example, if you want 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, EDI System automatically detects the SITE that is common to all the cells and creates the rows inside the power domain.

The following diagram illustrates how rows are automatically generated within the power domain for standard cells of the hierarchical instance.



The following diagram illustrates how standard cells of the hierarchical instance are all placed on the row inside the power domain.



The modules and/or instance groups can be moved outside the core boundary but within the die. Since new rows are not created automatically in this area, you can use the [createRow](#) command to create new rows.

Manual editing of rows might not be preserved by [floorPlan](#), [resizeFP](#), and [initCoreRow](#) commands.

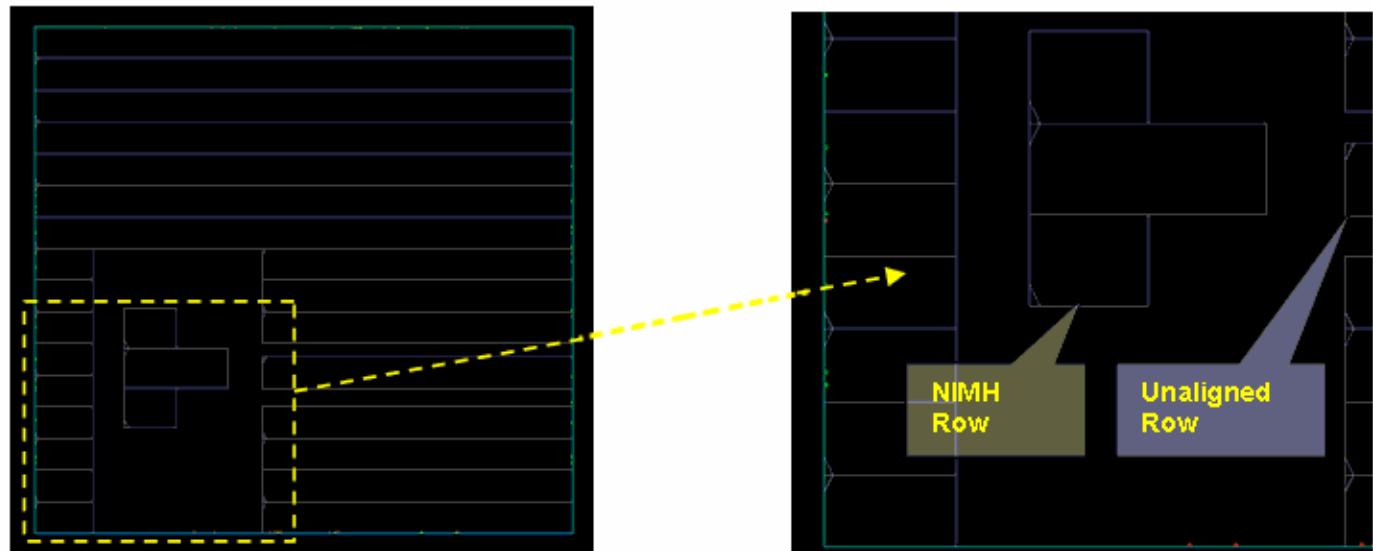
EDI System displays a warning message if you try to move a power domain outside the core boundary, and snaps the power domain inside the core.

## Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows

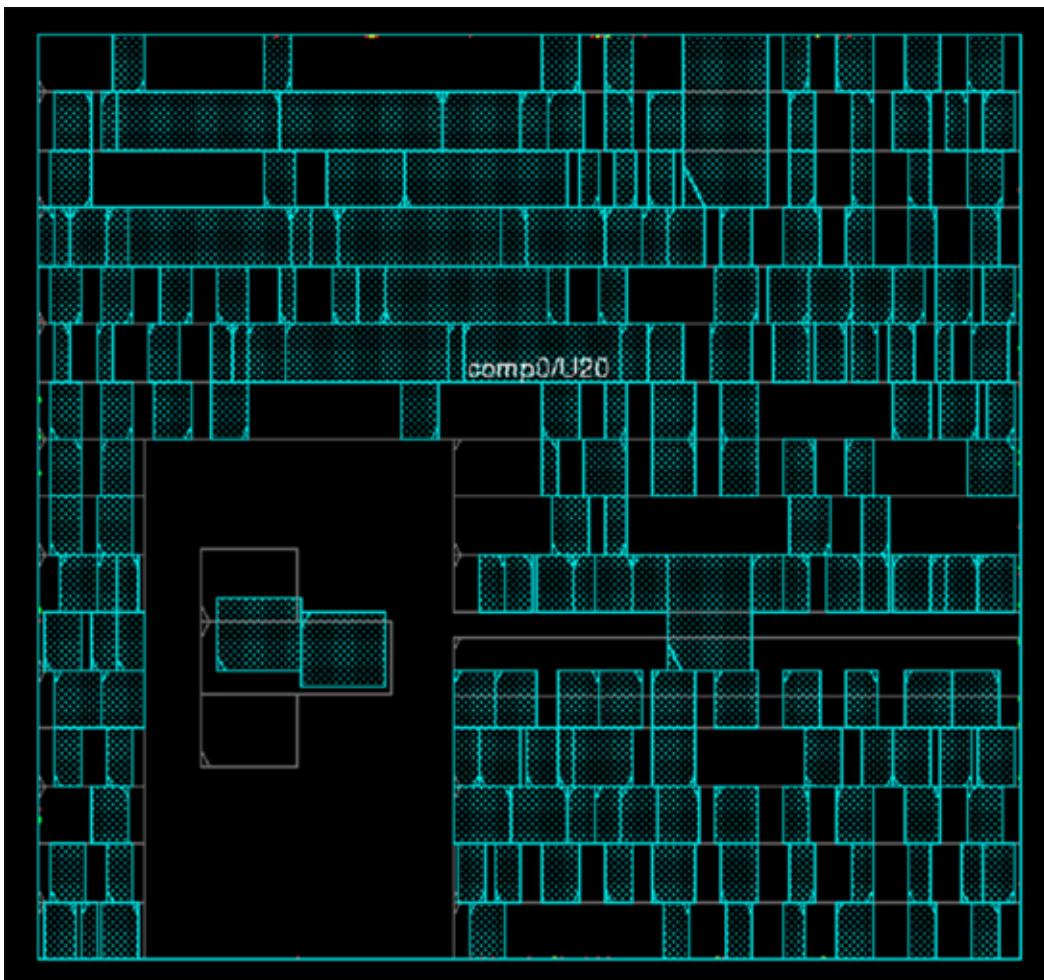
In case of integer multiple-height rows, as long as the rows are overlapping the single-height rows, standard cells will by default be legally placed on their corresponding site or row definition.

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.

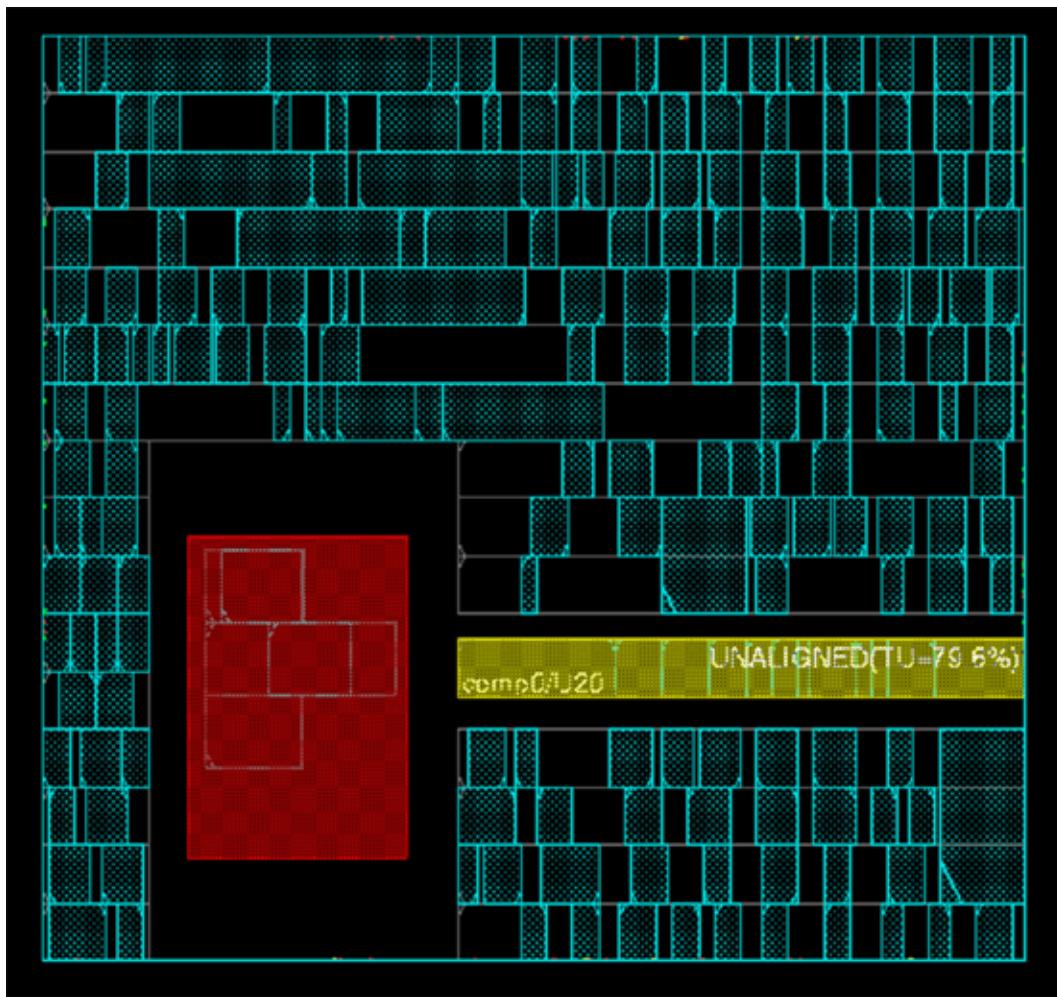
The following figure illustrates a user-defined floorplan brought in through a DEF file.



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 with the `-bringBackRow` parameter to preserve the row information. The [`flattenPartition`](#) command also supports rotated partitions.

Power domains are automatically created and associated with each of the partition that have NIMH or unaligned rows. These automatically created power domain have the following characteristics:

- The `minGap` value is the same as the placement halo defined for the partition at the full-chip

level.

- The timing library is the same as that specified at the full-chip level.
- The global net connection is the same as that for the full-chip level, which is the same as the partition floorplan global net connection.
- The value of the `RouteSearchExt` field is set to the default value of `0..0`.
- The core-to-edge distances are *not* preserved as an attribute of the power domain, but are preserved in the merged floorplan.
- Row Parameters are not translated or detected.
- The power domain is set to `alwaysOn`.

The recommended use model for bringing back non-integer multiple-height rows or unaligned rows is as follows:

#### Top-down flow

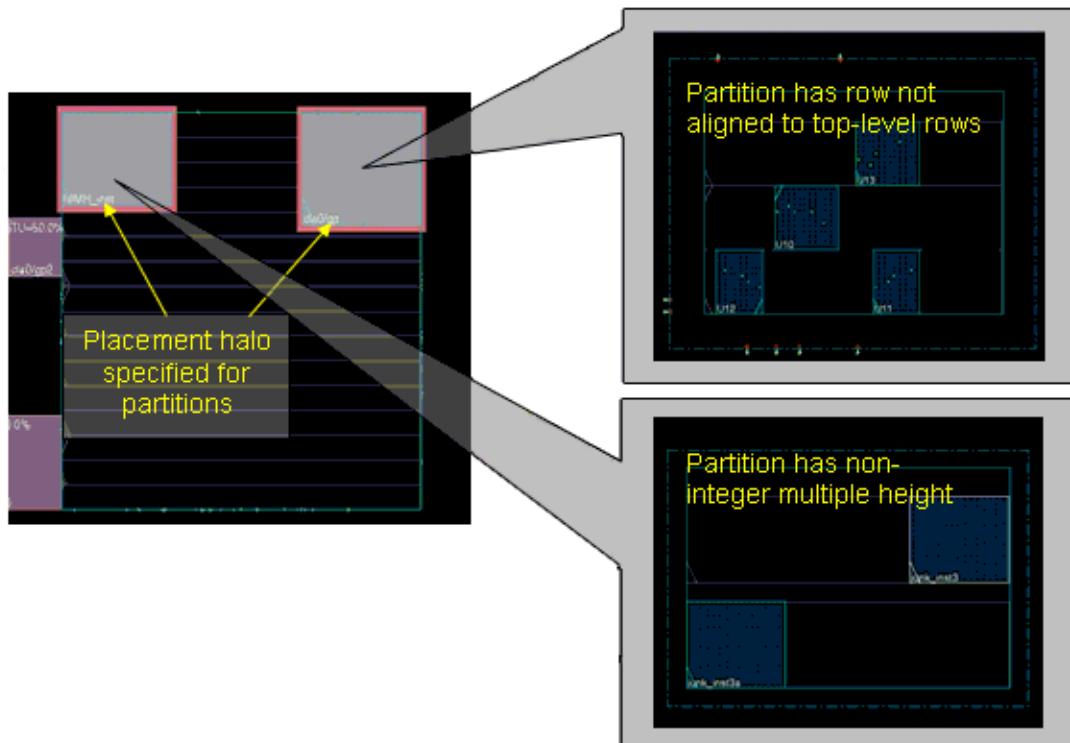
1. Create power domains at full-chip level design.
2. Specify the same hierarchical instance for power domain as defined for the partition.

#### Bottom-up flow

1. Create power domains at top-level design.
2. Create one PD for each of the partitions.
3. Assign instance blocks as a member of the created power domain.

For more information on the [flattenPartition](#) command, see the *EDI System Text Command Reference* .

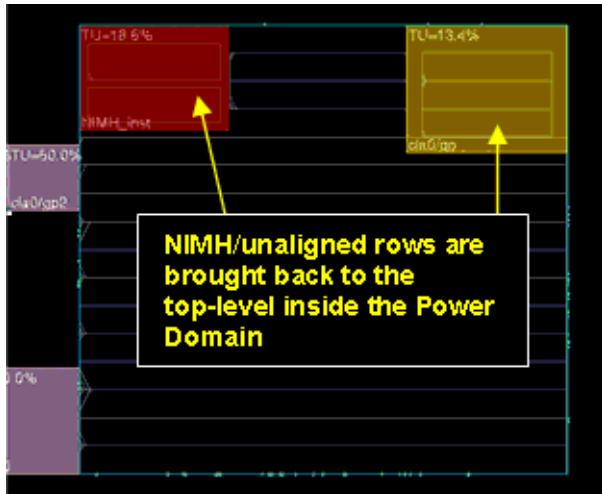
The following figure shows a full-chip view with the partition halos specified.



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 EDI System 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 EDI System, you must first define I/O SITE for each type of I/O cell in the LEF I/O macro (LEF technology file).

2. Each I/O cell LEF must have correct CLASS and SITE type specified.

Consider the following examples that define LEF I/O SITE and CLASS PAD MACRO in the I/O assignment file, each I/O CLASS PAD macro is referenced with the I/O SITE:

### Example 1

The I/O SITE IOPFC is referenced from the I/O CLASS PAD MACRO pn1\_qdr\_vp:

```
SITE IOPFC SYMMETRY y;  
    CLASS PAD ;  
    SIZE 0.1 BY 321.94 ;  
END IOPFC
```

```
MACRO pnl_qdr_vp CLASS PAD ;
  FOREIGN pnl_qdr_vp ;
  ORIGIN 0.000 0.000 ;
  SIZE 35.000 BY 321.940 ;
  SYMMETRY X Y R90 ;
  SITE IOPFC ;
  ...
END pnl_qdr_vp
```

### Example 2

The corner site, IOPFCCRNR , is referenced from the CLASS PAD MACRO pnl\_qdr\_iocrnr:

```
SITE IOPFCCRNR
  SYMMETRY y ;
  CLASS PAD ;
  SIZE 321.94 BY 321.94 ;
END IOPFCCRNR
```

```
MACRO pnl_qdr_iocrnr
  CLASS PAD ;
  FOREIGN pnl_qdr_iocrnr ;
  ORIGIN 0.000 0.000 ;
  SIZE 32.940 BY 321.940 ;
  SYMMETRY X Y R90 ;
  SITE IOPFCCRNR ;
  ...
END pnl_qdr_iocrnr
```

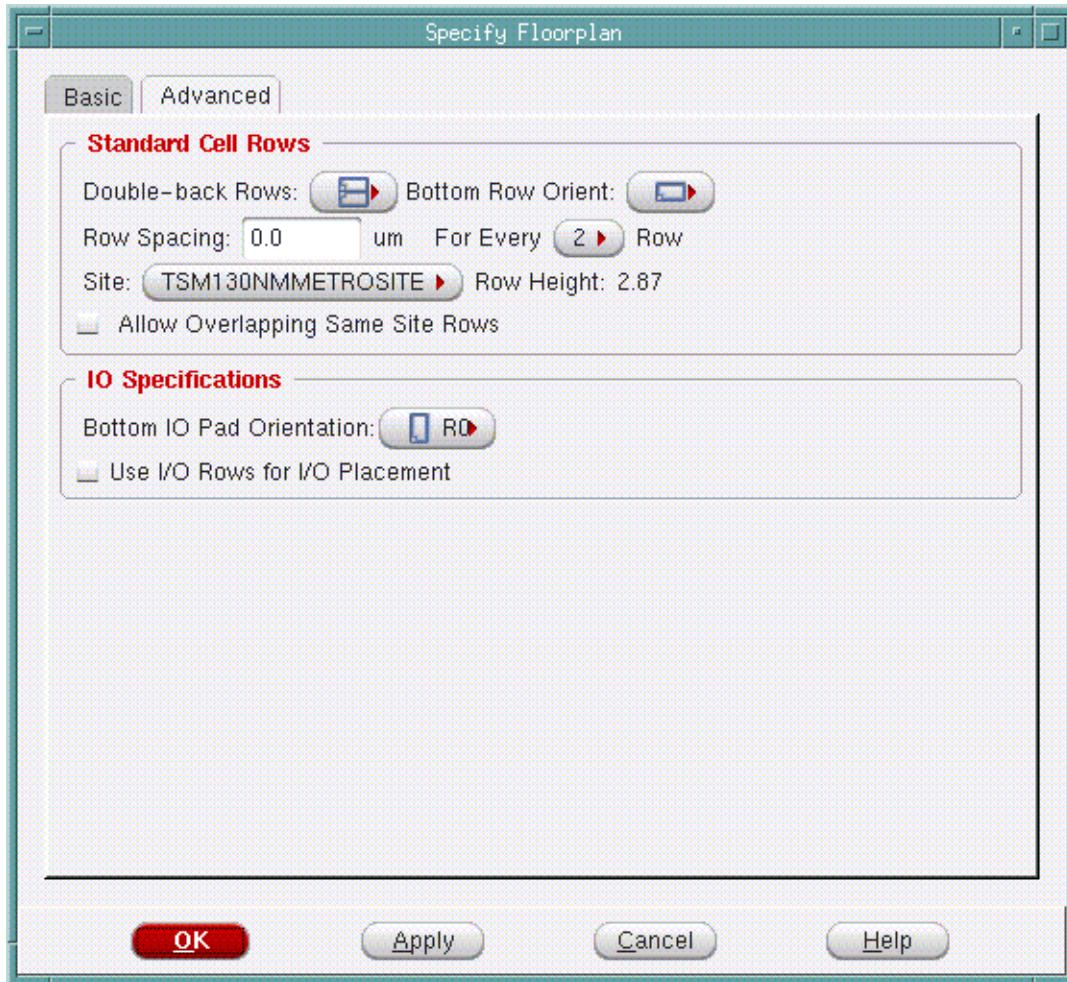
For more information, see "[Generating the I/O Assignment File](#)" in Data Preparation chapter of the *EDI System User Guide*.

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 EDI System software issues a warning and no I/O rows are created. EDI System does not automatically expand the core or die boundary to accommodate all the I/O pads.

## Enabling the I/O Row Flow in EDI System

To start using the I/O row flow in EDI System, you must enable the I/O row flow by doing one of the following:

- Specify `setUserDataValue conf_use_io_row_flow 1` in the EDI System 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.

**Note:** By default, the I/O row flow is *Off*.

## Use Models

### Starting a new design

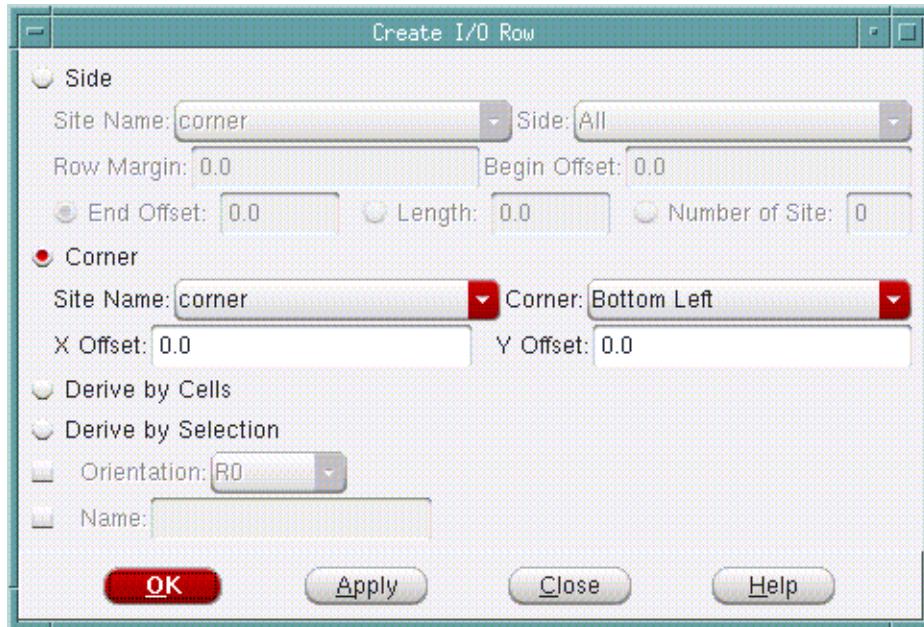
1. Import the design in EDI System
2. Set the I/O row flow by selecting the *Use I/O Row for I/O Placement* check box in the [Specify Floorplan - Advanced](#) GUI form.

[Advanced](#) GUI form.

3. By default, one I/O row is created on each side of the chip, between the core boundary and the die boundary. If the design has multiple I/O sites in the library, then rows are created for each side based on the number of I/O sites used in the design.

By default, the I/O pads are placed randomly on the I/O rows.

4. In the resulting design, you can create I/O rows using the [Create I/O Row](#) form.



5. Edit (move/stretch/rotate/flip) the I/O rows using the [Edit I/O Row](#) form or using the text commands.

The text commands that can be used for creating and editing I/O rows are described in the following table:

| Commands                           | Usage                                                                                                                                                                      |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">createIoRow</a>        | Creates an I/O row.                                                                                                                                                        |
| <a href="#">flipOrRotateObject</a> | Flips or rotates the selected objects.                                                                                                                                     |
| <a href="#">stretchRows</a>        | Stretches the selected I/O rows.                                                                                                                                           |
| <a href="#">moveSelObj</a>         | Moves the selected row to a specific location. Since the pads are already placed on the I/O rows, by default when you move the I/O rows, the pads also move with the rows. |
| <a href="#">deleteRow</a>          | Deletes the selected I/O row. You can also delete the row by selecting the row and pressing the Del key on the keyboard.                                                   |
| <a href="#">setIoRowMargin</a>     | Sets the distance from the die boundary edge to the I/O row starting edge location. You can use this command for multiple I/O rows.                                        |

|                             |                                                                                                         |
|-----------------------------|---------------------------------------------------------------------------------------------------------|
| <a href="#">snapFPlanIO</a> | Snaps the I/O pads onto the correct side of the I/O rows if the pads are not already on the rows.       |
| <a href="#">spaceIoInst</a> | 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 EDI System 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 EDI System (v6.2 and above) and you want the design to be read into current version of EDI System to use the new I/O row flow:

1. Restore the design with placed pads, using [restoreDesign](#) command or load the floorplan information from the file, using the Load FPlan File form or the [loadFPlan](#) command.

Optionally, load the I/O constraint file using [loadIoFile](#) command.

2. Turn on the I/O row flow by setting the I/O flow flag ([setIoFlowFlag](#)) to 1.

3. Create the initial I/O rows using the [createIoRow](#) -deriveByCells command. This command creates I/O rows based on the existing pad placement.

Once you have rows and pads placed in the design, you can continue to create more rows or edit the rows.

Use the [changeIoConstraints](#) command to change the constraints of the I/O row read from the I/O constraints file.

You can also use the [Attribute Editor](#) to change the I/O pad location or pad orientation from the GUI.

4. After editing the I/O rows and the I/O row constraints, run the [snapFPlanIO](#) command to snap the I/O pads and area I/O's onto the legal sites/rows.

5. Save the design in a floorplan file (\*.fp) using the [saveDesign](#) or [saveFPlan](#) command.

### Resizing Rectilinear Blocks

Given an initial rectilinear block in the floorplan, EDI System automatically resizes its bounding box by enlarging or shrinking the edges of the box proportionally in the X and Y directions, ensuring that the

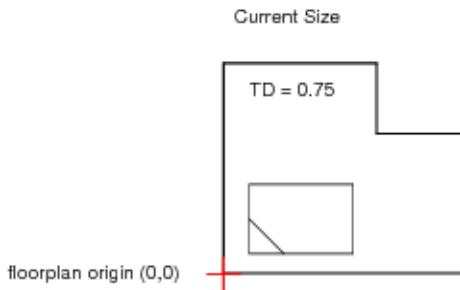
specified target utilization is met.

During this process, to retain the original shape of the rectilinear block, you must specify the -keepShape parameter in the [floorPlan](#) command. Consider analog designs where you have digital blocks that need to be fit into the analog chip and the shape of the block is already pre-defined. In such mixed-signal designs, you can retain the block shape during resizing, and also meet the specified target utilization value by shrinking or expanding the floorplan using the floorplan -keepShape util value, where util is the target utilization value.

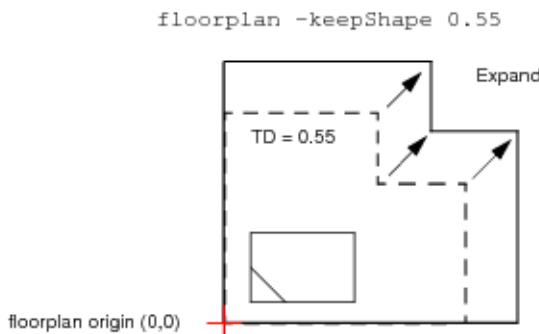
### Use Models

1. Import a DEF file or a floorplan file (\*.fp) that contains a rectilinear block boundary or you cut one corner of a rectangular block. This results in the core utilization missing the target value.
2. Run the floorplan -keepShape util command to automatically shrink or expand the block boundary, proportionally, in the X and Y directions, trying to meet the specified target utilization.

Consider the following example of a rectilinear block whose current target utilization value is 0.75.

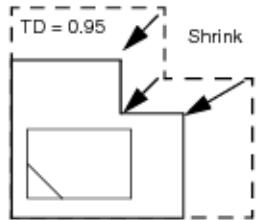


To meet a target utilization of 0.55, expand the rectilinear block.



To meet a target utilization of 0.95, shrink the rectilinear block.

```
floorplan -keepShape 0.95
```



In both the cases, the shape of the block is retained, and the required target utilization is met.

For I/O pins, prior to resize, EDI System saves the I/O file sequence internally and loads the file back after resize. The side and sequence of the I/O pin remains the same as in the old block, but the pins get distributed evenly. To redistribute the I/O pins, you must edit the pins manually in the resize block.

### Assumptions

The automatic resizing of rectilinear blocks is based on the following assumptions:

1. The rectilinear blocks 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 blocks are as follows:

1. The shape of the block 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.

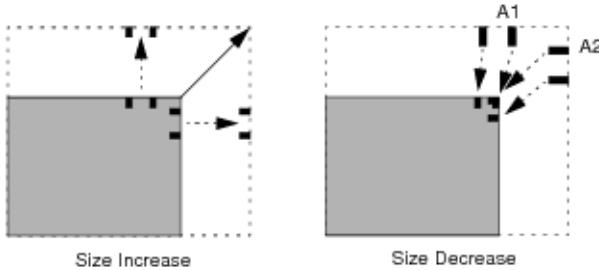
## Editing Pins

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 EDI System 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.

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 *EDI System 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.

The following sections provide details on the four pin-spreading methods supported by the Pin Editor.

#### Using a Pin as the Starting Point for Spreading Pins

For this method, you select a group of pins and sort them in the desired order. The first pin in the list serves as the starting point (anchor) for spreading the other pins in the group.

You must also provide the pin spacing distance if you are spreading more than one pin.

Assume that your design contains four pins (A1, A2, A3, and A4) that are currently spaced 2.0  $\mu\text{m}$  apart. You want to spread the pins to the right with 3.0  $\mu\text{m}$  spacing, using A1 as the starting point. To do this you must

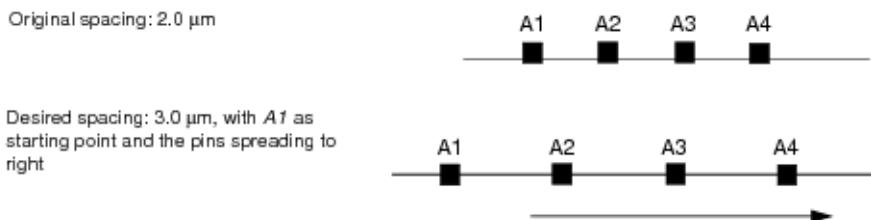
1. Sort the pins so that A1 is the first pin in the list.

The coordinates of A1 appear in *Starting X/Y*.

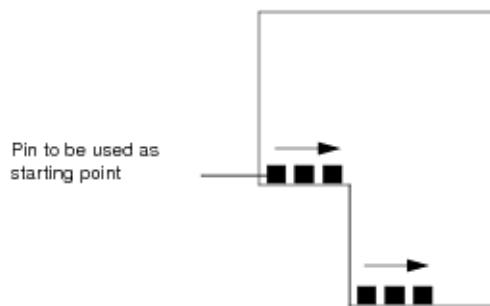
2. Select *Spread - From Starting Point* on the Pin Editor form.

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.



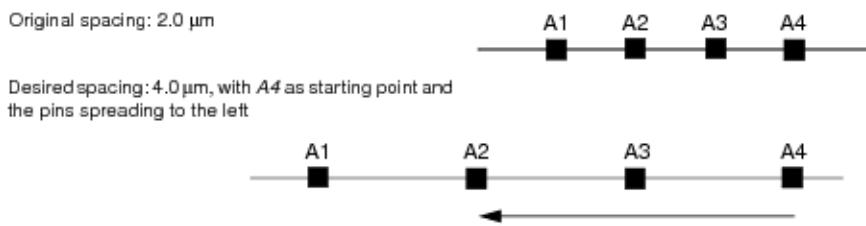
Now assume that you want to spread the pins to the left with  $4.0 \mu\text{m}$  spacing, using A4 as the starting point. To do this you must

1. Sort the pins so that A4 is the first pin in the list.

The coordinates of A4 appear in *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

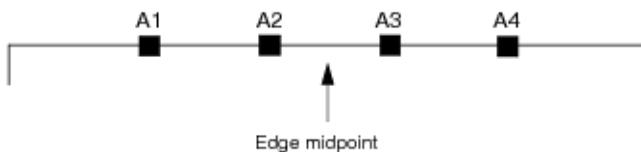
1. Sort the pins in the desired order (optional).
2. Select *Spread - From Center* on the Pin Editor form.
3. Specify a *positive* spacing value: 3 . 0.

The following figure illustrates this situation:

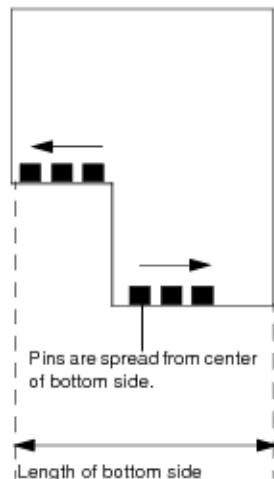
Original pin spacing



New spacing, with the pin group centered on the midpoint of the block edge



The following figure shows how pins are spread from a center point in case of rectilinear partitions where a side with multiple edges has been specified. In this case, the specified side is bottom, and the pins are therefore spread from the center of the bottom side.



## Spacing Pins Evenly Along an Edge or Side

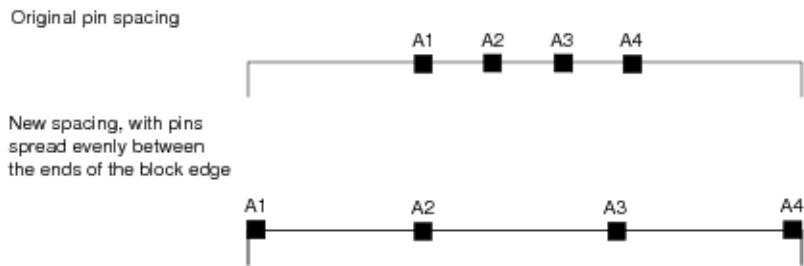
For this method, you select a group of pins and sort them in the desired order.

You do not specify a pin spacing distance because the Pin Editor calculates the appropriate distance, based on the length of the block edge or side, and spaces the pins evenly along the block edge or side.

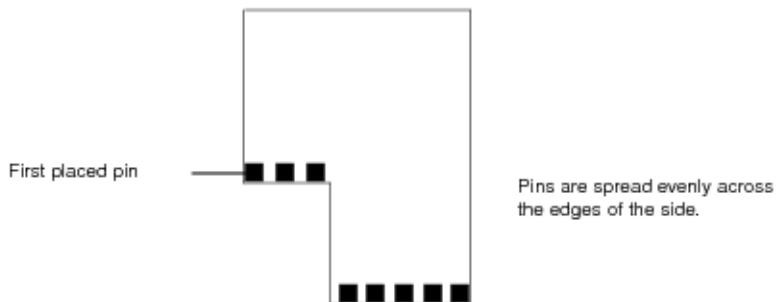
Assume that one edge of your design contains four pins (A1, A2, A3, and A4). You want to spread the pins evenly along the block edge. To do this you must

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.

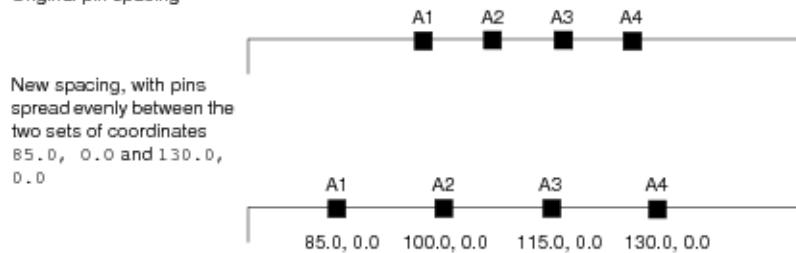
You do not specify a pin spacing distance because the Pin Editor calculates the appropriate distance, based on the specified starting and ending points, and spaces the pins evenly along the edge or side.

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.

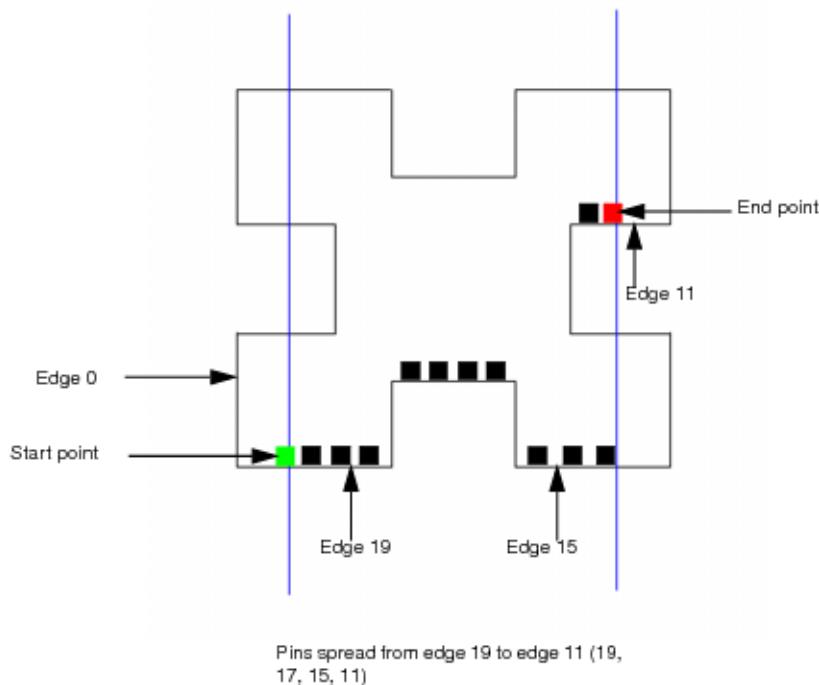
The following figure illustrates this situation:

Original pin spacing

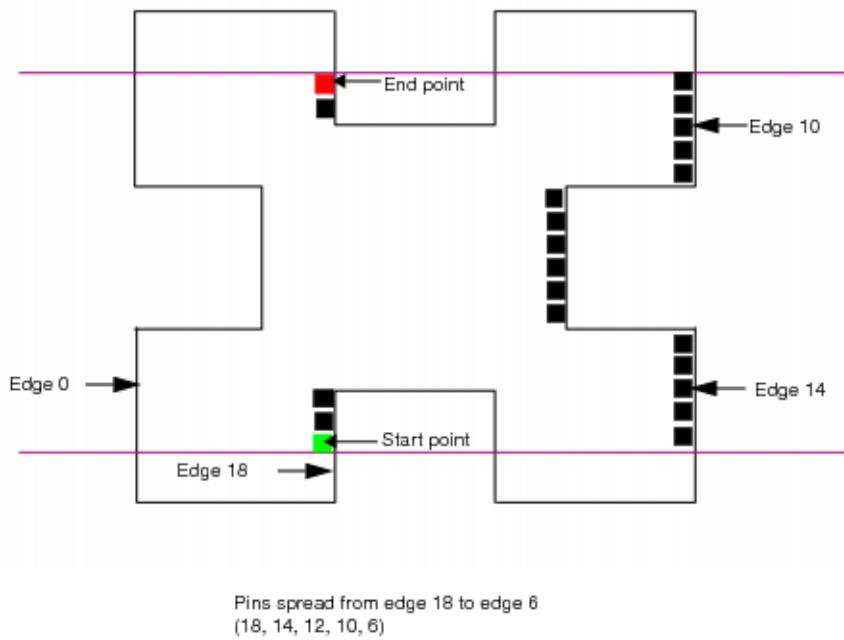


The following two figures shows how pins are spread along a side in case of rectilinear partitions where a side with multiple edges has been specified.

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.



## 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 EDI System session.

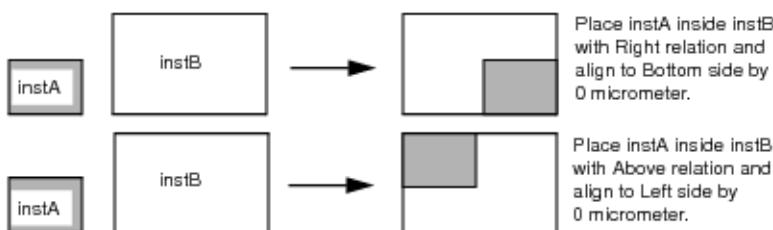
### Orientation Key

The following table is a key to the orientation of placed objects:

| Value | Definition                                                                             |
|-------|----------------------------------------------------------------------------------------|
| R0    | <input type="checkbox"/> No rotation                                                   |
| MX    | <input type="checkbox"/> Mirror through X axis                                         |
| MY    | <input type="checkbox"/> Mirror through Y axis                                         |
| R180  | <input type="checkbox"/> Rotate counter-clockwise 180 degrees                          |
| MX90  | <input type="checkbox"/> Mirror through X axis and rotate counter-clockwise 90 degrees |
| R90   | <input type="checkbox"/> Rotate counter-clockwise 90 degrees                           |
| R270  | <input type="checkbox"/> Rotate counter-clockwise 270 degrees                          |
| MY90  | <input type="checkbox"/> 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

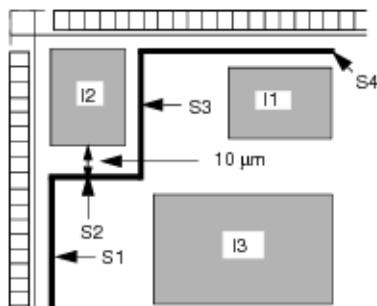
You can generate pre-routed relative floorplan information automatically after you have pre-routed a relative floorplan. The following example and illustrations show how the relative floorplan pre-route feature operates.

The following commands specify that S3, S2, and S1 are relative to object I2 and the core:

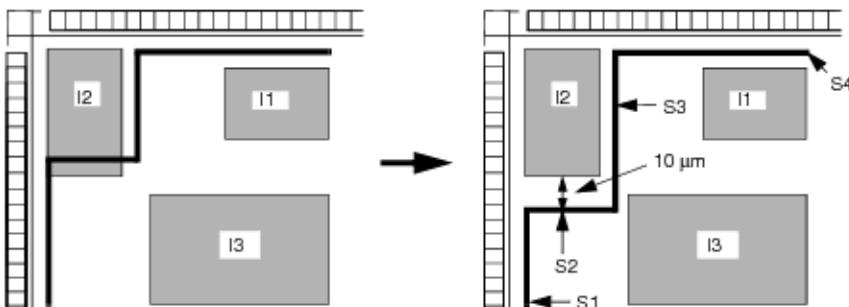
```
relativeFPlan --preRoute VDD Metal1 Metal2 0.44 0.44 I2 BR X2 Y2 I2 TR
X3 Y3
```

```
relativeFPlan --preRoute VDD Metal1 Metal2 0.44 0.44 I2 BL X1 Y1 I2 BR
X2 Y2
```

```
relativeFPlan --preRoute VDD Metal1 Metal2 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.



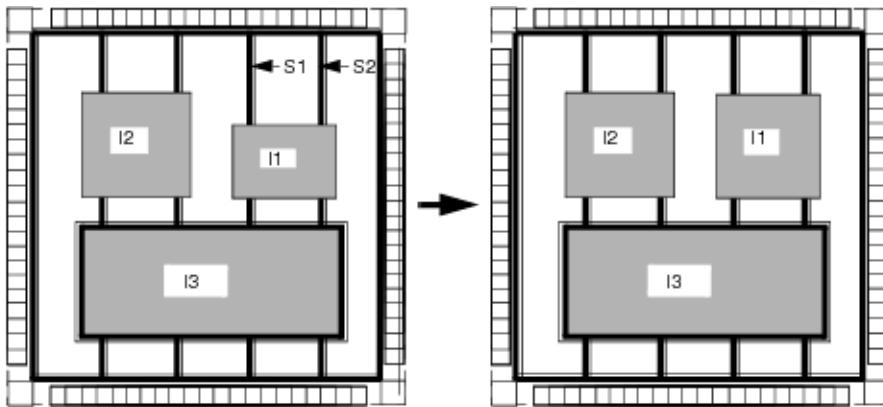
The following commands specify that S1 and S2 are relative to the object I2 and the Core\_Boundary:

```
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 EDI System software automatically saves all the executed menu and text commands for the relative floorplanning actions in the encounter.cmd file.

To save all the relative floorplan commands that were executed during a session, click the Save button on the Relative Floorplan form. This saves a script that can be used later for updating or adjusting an existing floorplan based on the new blocks' size and position. You can also save the constraints associated with an object through the [saveRelativeFPlan](#) command.

To restore the relative floorplan information to the design display area, use the [restoreRelativeFPlan](#) text command.

For more information, see "Floorplan Commands" in the *EDI System Text Command Reference*.

## Saving and Loading Floorplan Data

You can save and load floorplan data at any time during a session.

- To save the floorplan information to a file, use the Save FPlan File form or the [saveFPlan](#) text command.
- To load the floorplan information from a file, use the Load FPlan File form or the [loadFPlan](#) text command.

**Note:** You can save and load floorplan data in the XML format.

## 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](#) by using the [setResizeFPlanMode](#) command. Parameters that you specify with `setResizeFPlanMode` 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 EDI System GUI to perform the resize functions in the design.

**Note:** You can use the [getResizeFPlanMode](#) command to view the information about a specified `setResizeFPlanMode` parameter in the EDI System log file and in the EDI System 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 *EDI System 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 *EDI System 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 EDI System

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, EDI System does not support a scenario where resizing line by expanding and shrinking, both occur on the same direction. For example, the following command specifies a resize line in vertical direction with a resize width of 100 microns:

```
setResizeLine -direction V (x1 y1) (* y2) -width 100
```

The following command again resizes the floorplan in the same X direction with a negative value of -200 microns:

```
resizeFP -xSize -200
```

EDI System displays a warning message in such a situation.

## Snapping Resize Values

The resize values (shrink/expand) can be snapped to a multiple integer of the metal layer pitch.

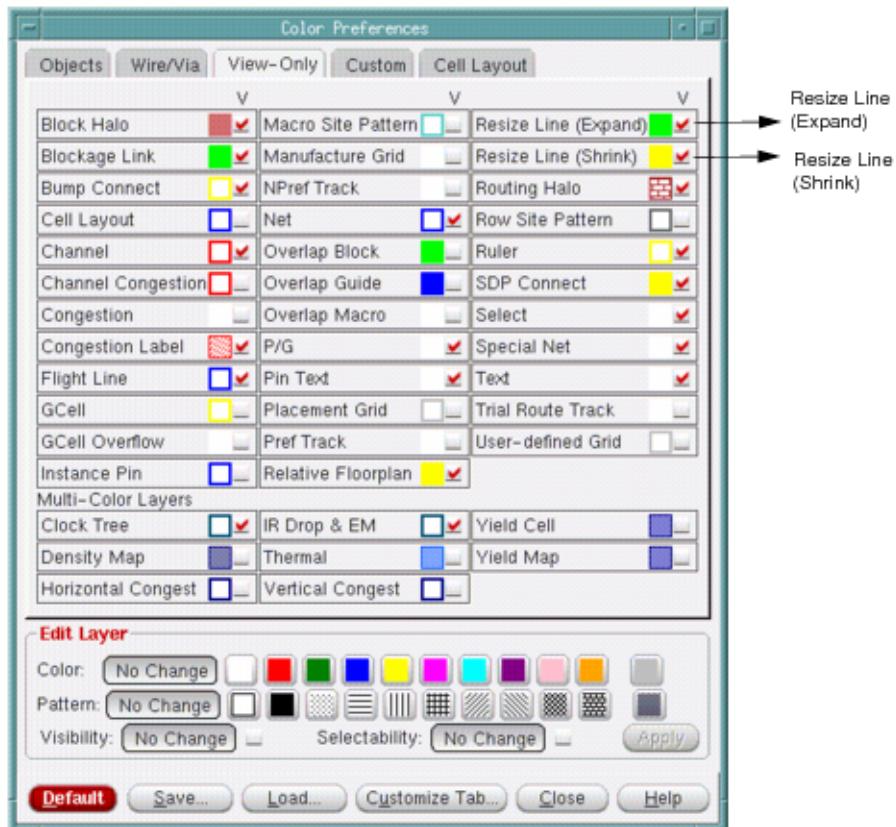
**Note:** Specify the `setResizeFPlanMode -snapToTrack` option to snap resize values.

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 *EDI System Menu Reference* for more information.

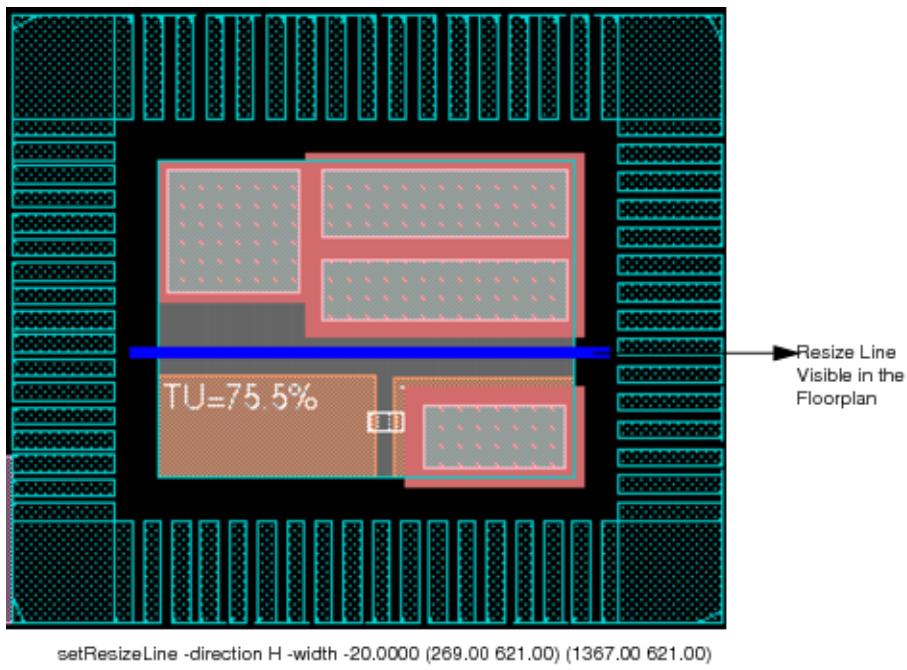
## Viewing Resize Lines using Color Preferences

Once you specify the resize lines to perform shift-based resizing, you can display the resize lines in EDI System by setting the Resize Line option in the *Color Preferences - View- Only* form.

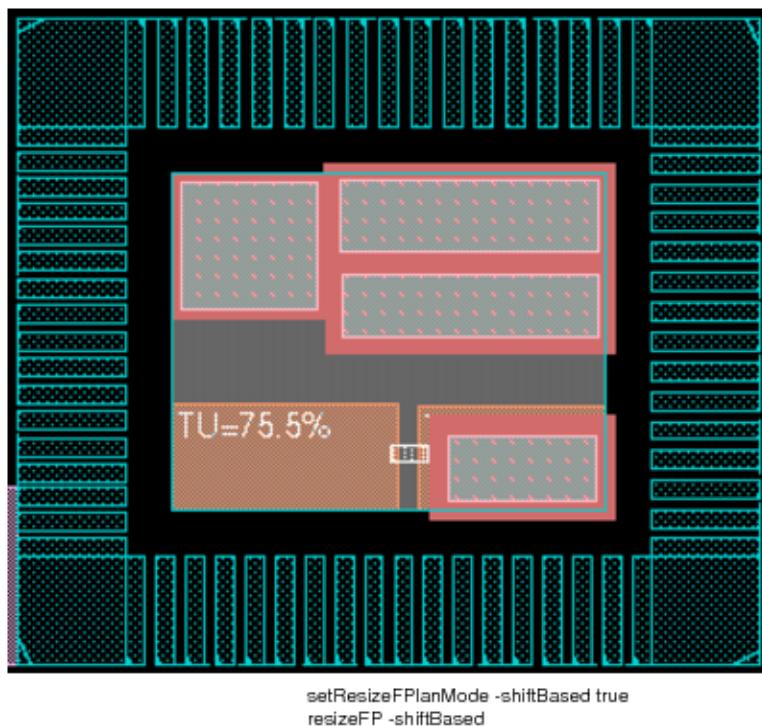


However, the resize lines disappear once you run the [resizeFP](#) command.

### **Example 13-1 Setting a Resize Line Before running resizeFP**



### Example 13-2 Resize Line Disappears After Running resizeFP in Shift Based Mode



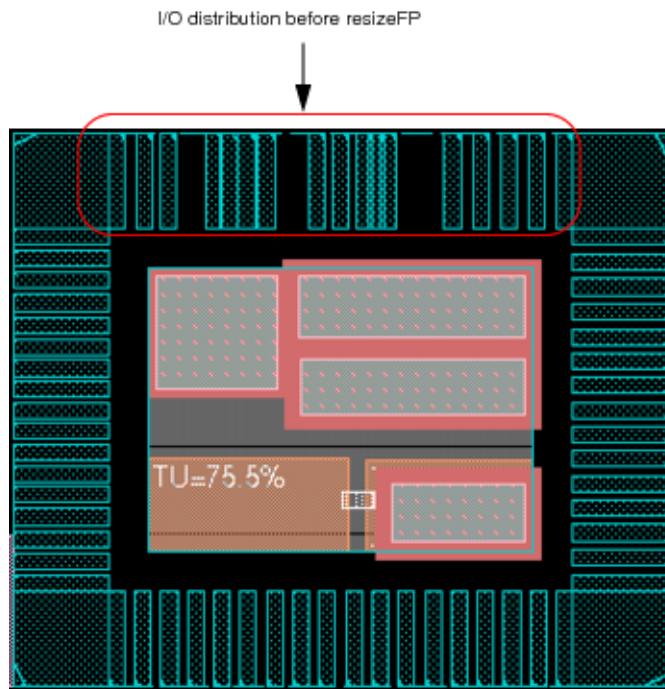
**Note:** During resizing, the target size may not be achievable. You have to force resize to meet the target size as much as possible, using the `resizeFP -forceResize` option.

## Distributing I/O's 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,

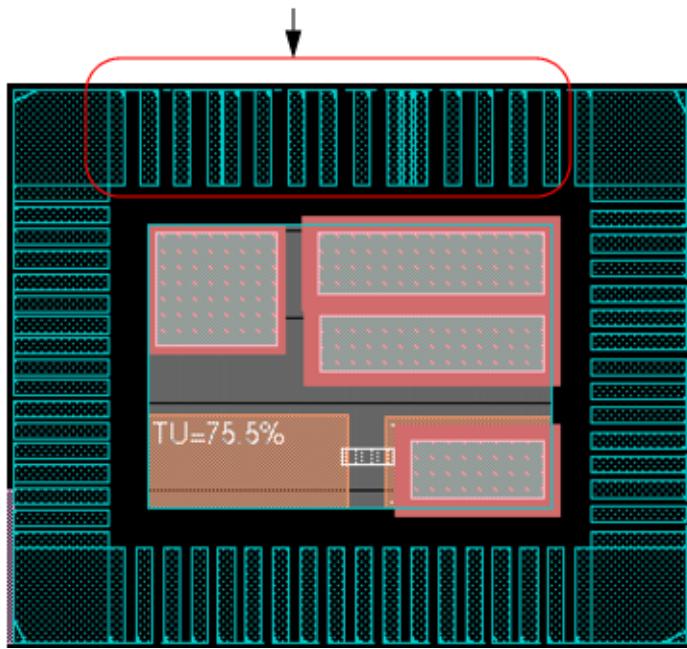
- The space between I/O pads can be adjusted evenly or proportionally. By default, the I/O's are distributed proportionally.
- The I/O side constraints and order constraints are honored.
- The offset that exists between I/O's and the design boundary is preserved.

### Example 13-3 Distribution of I/Os Before Resize Floorplan



### Example 13-4 Evenly Distribution of I/O's After Resize Floorplan

Evenly I/O distribution after  
setResizeFPlanMode -ioProportional false  
resizeFP -xSize 50



---

# Prototyping Foundation Flow

---

- [Overview](#)
- [The Prototyping Foundation Flow Stages](#)
  - [Generate Models](#)
  - [Debug Constraints and Prototype Design](#)
  - [Defining Partitions](#)
  - [Finish and Save Partition](#)
- [Advantages of Using Prototyping Foundation Flow](#)

## Overview

The prototyping foundation flow has 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.

## The Prototyping Foundation Flow Stages

The prototyping foundation flow runs in the following stages:

- Generate Models
- Debug Constraints and Prototype Design
- Analyze Floorplan and Adjust
- Defining Partitions
- Finish and Save Partition

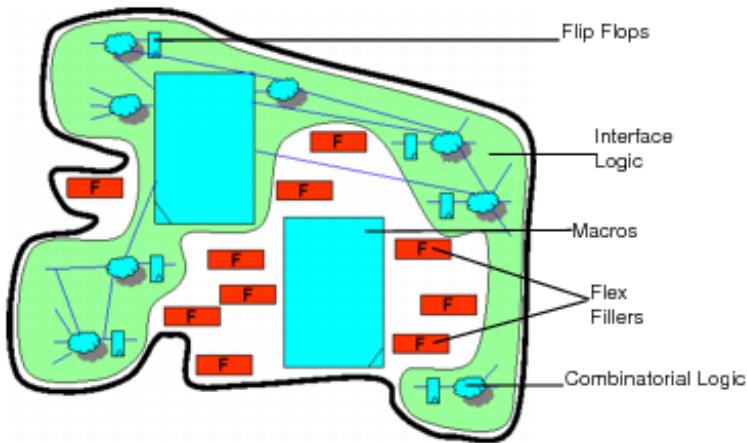
The following diagram shows the various stages of prototyping foundation flow.



## Generate 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.

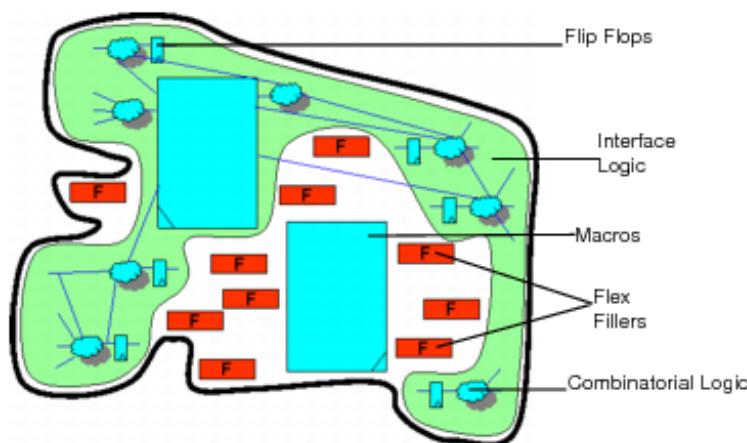
A flexModel is a Verilog netlist that can contain macros, interface standard cells, and flex fillers. The flex fillers reserve space for the removal of internal register-to-register logic. The flex fillers have no timing model associated with them and they connect such that the model holds together during placement. So the placer will place them together in one group.



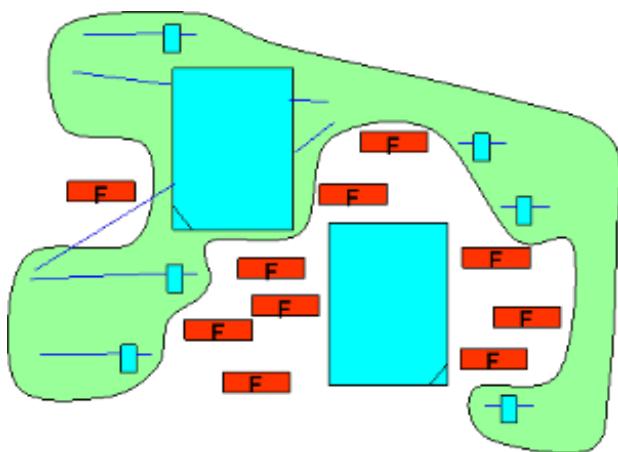
A flexModel netlist is usually one tenth the number of instances of its full netlist. It is used during early design planning to reduce the run time and memory while accurately modeling the timing and area.

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

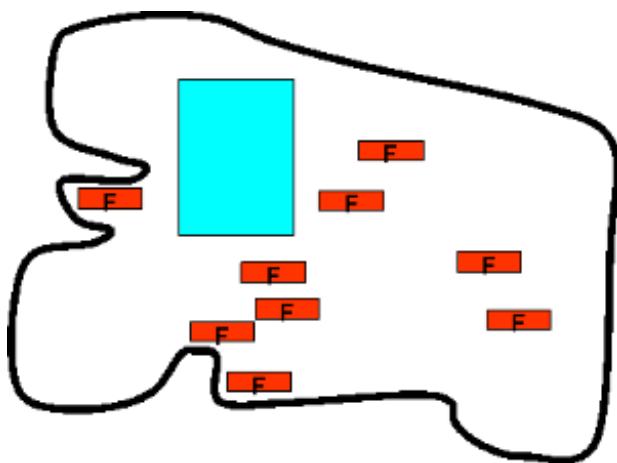
estimation.



If you 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.



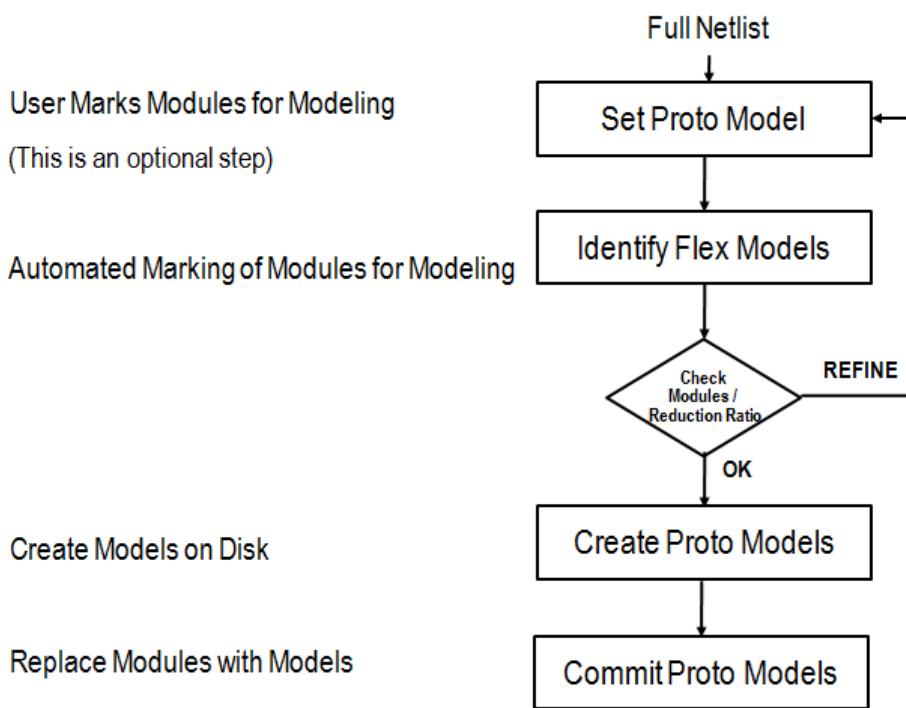
In case of empty netlist, your flexModel will have only the macros and flex fillers. Using empty netlist, you can solve only the congestion problem but you cannot estimate timing.



Following is the comparison table between different supported EDI 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 offull chip)</b>  | ✓        | ✓       | ✓         | ✓         |
| <b>Runtime (&lt;1/10 offull chip)</b> | ✓        | ✓       | ✓         | ✓         |

## The Model Generation Flow



#### Examples

- The following example shows the use of `set_proto_model` command using which you can mark the modules for modeling.

```
encounter> set_proto_model -module prog_bus_mach -type flexmodel
```

```
encounter> set_proto_model -module port_bus_mach -type flexmodel
```

```
encounter> set_proto_model -module mult_32 -type flexmodel
```

```
encounter> set_proto_model -module mult_32 -create_total_area 9770
```

```
encounter> get_proto_model -all
```

| module        | type      | source | create_total_area |
|---------------|-----------|--------|-------------------|
| prog_bus_mach | flexmodel | user   |                   |
| mult_32       | flexmodel | user   | 9770              |

```
port_bus_mach      flexmodel   user

encounter> get_proto_model -all -tcl

{{name prog_bus_mach}      {type flexmodel}  {source user}}

{{name port_bus_mach}      {type flexmodel}  {source user}}

{{name mult_32}            {type flexmodel}  {source user}  {create_total_area
9770} }

encounter> get_proto_model -type_match {flexmodel} -name -tcl

prog_bus_mach  port_bus_mach  mult_32
```

- The following example shows the use of `identify_flexmodel` command that is used for automated marking of modules for modeling.

```
encounter> set_proto_mode -identify_partition_min_insts 205

encounter> set_proto_mode -identify_partition_max_insts 4190

encounter> identify_flexmodel
```

- The following example shows the use of `create_proto_model` command that is used to create models on the disk.

```
encounter> setMultiCpuUsage -remoteHost 2 -cpuPerRemoteHost 1

encounter> setDistributeHost -local

encounter> create_proto_model
```

```
encounter> report_model -created
```

- The following example shows the use of `commit_proto_model` command that replaces modules with the prototyping models.

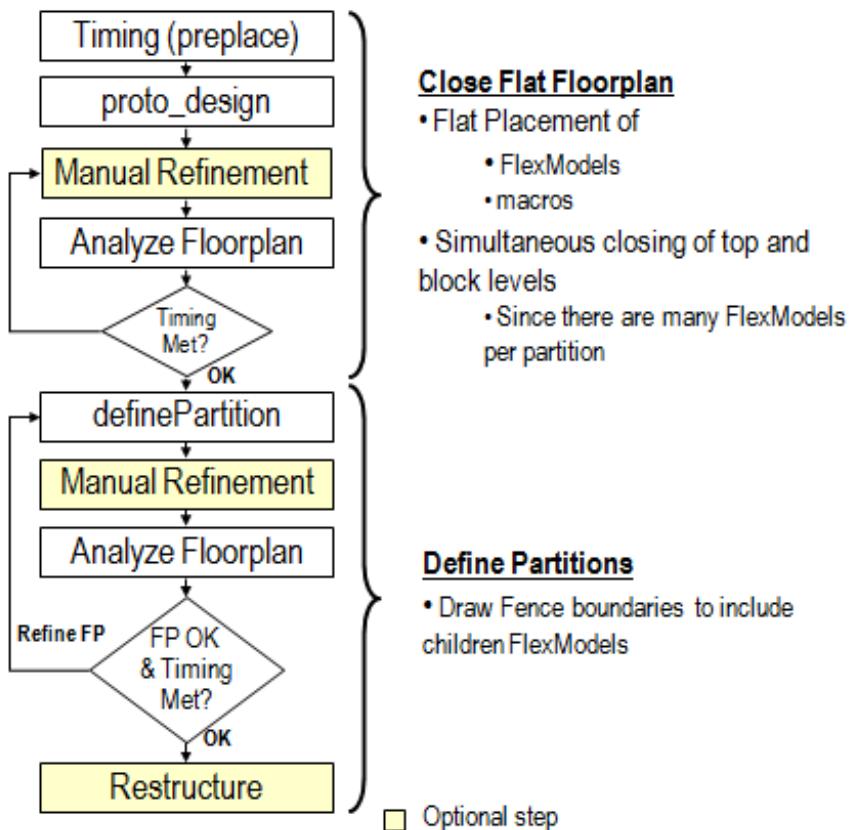
```
encounter> commit_proto_model
```

```
encounter> report_model -committed
```

For more information about the prototyping foundation flow commands, see the [Prototyping Foundation Flow Commands](#) chapter of the *EDI Text Command Reference*.

## Debug 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 low or medium -effort timeDesign pre-placement. This is done to check the initial timing and constraints.

```
set_proto_mode -effort low
```

```
timeDesign -proto -preplace
```

The timeDesign -proto parameter allows you to call the prototype timing if flexModels are present.

The low-effort 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

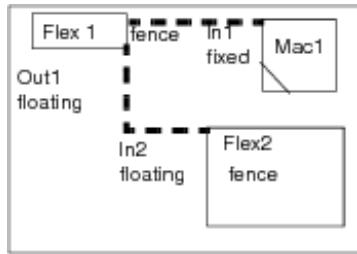
The prototyping foundation flow accepts `set_proto_mode -timing_ps_per_micron`. This parameter represents the amount of total delay (buffer and wire) that a technology can drive a signal at a certain distance. If value of `-timing_ps_per_micron` is not specified EDI will automatically create a pico-second 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 `ln1`. If you run `planDesign` without running standard cell placement, then the standard cell driving `out1` within `Flex1` is unplaced. If the macro `ln1` terminal location is known, then the distance is calculated as the minimum manhattan distance between the `Flex1` fence and the `ln1` terminal.



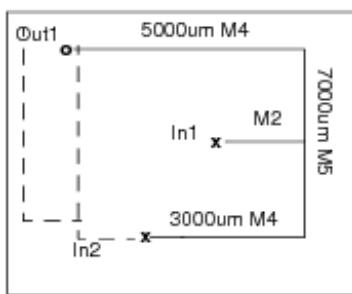
If a standard cell containing `ln2` within `Flex2` standard cell is unplaced, then the distance is calculated as the minimum distance between the two `flexModels`.

The delay is zero for any case where one of the terminals is of a 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 -effort` is `low` OR `medium`, 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 the effort is set to high, and if no routing exists, the delay calculation is same as with the low/medium effort. Else, actual routing distances and layers are used for calculating the delay:

M4delay + 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.

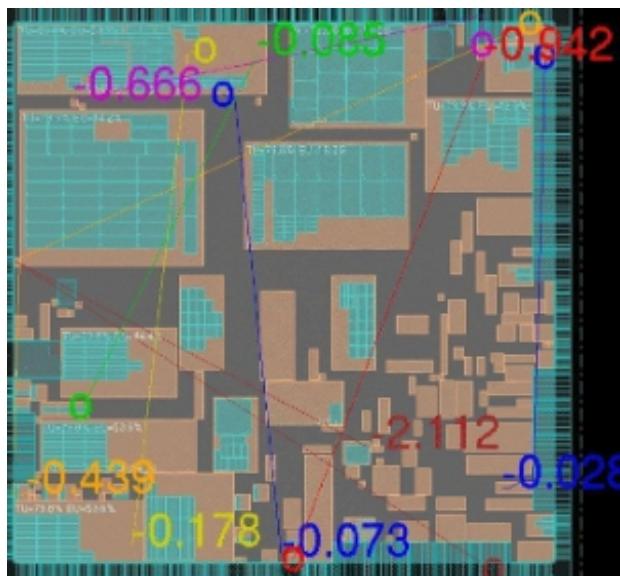
### **Analyze Floorplan and Adjust**

After `proto_design`, some manual refinement is usually necessary, followed by running these commands:

```
ff_checkFPlan  
analyzeFloorplan
```

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.

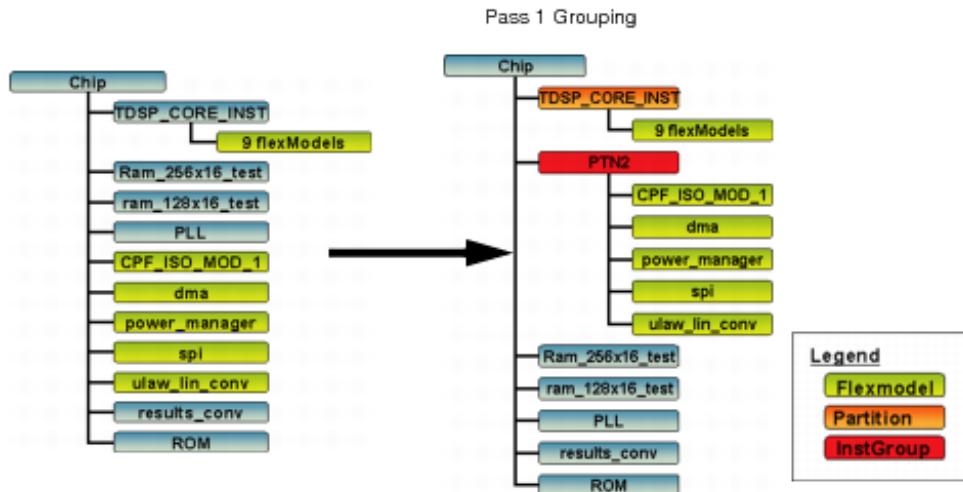
The `load_timing_debug_report -proto` command is automatically called by `analyzeFloorplan`. It creates timing categories for each `flexModel` pair. These timing categories are color-coded and thousands of timing paths are condensed into few paths. For example, in the figure below, thousands of timing 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

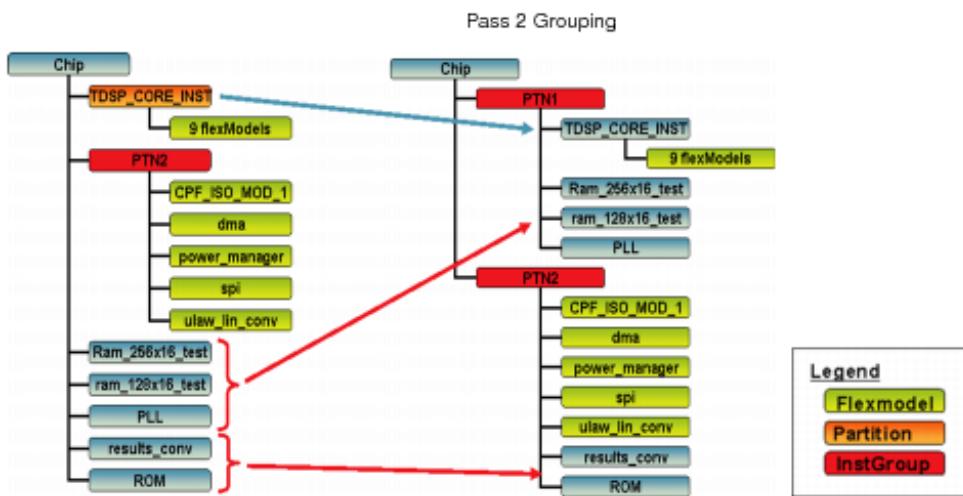
After running `planDesign` and getting a flat `flexModel` placement that closes timing, you need to define partitions. In the first pass for defining a partition, you are recommended to use the modules from the original design hierarchy. For example, in the design hierarchy shown below, the `TDSP_CORE_INST` partition has nine `flexModels`. There are other `flexModels` as well. 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\_model -identified command.

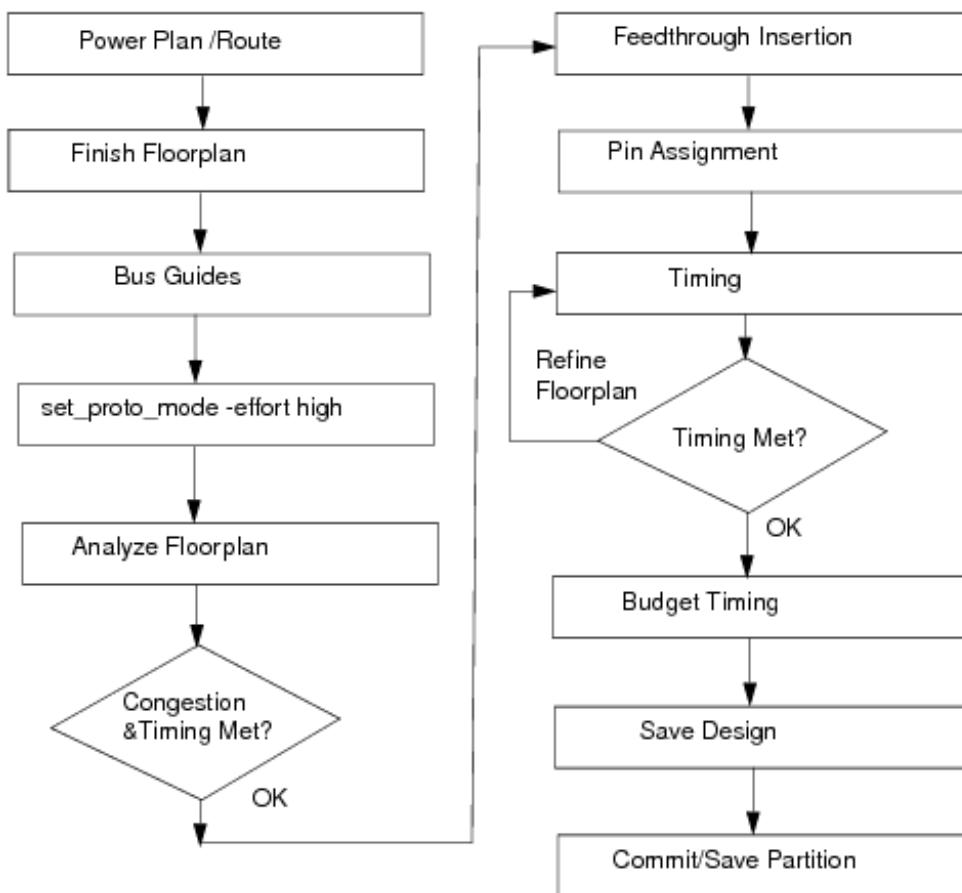
In the second pass, you define the partitions and perform manual refinement. Here you delete the TDSP\_CORE\_INST partition and create a new instance groups PTN1.



Finally you restructure your netlist using ff\_restructure\_netlist\_hier.

## Finish and Save 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 must set the value of `set_proto_mode -effort high` so that `trialRoute` will be called during `analyzeFloorplan -cong` 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
- Setting the Effort
- 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.

## Setting the Effort

The `-effort` parameter of `set_proto_mode` command should be set to `high` for finishing step. Doing this:

- `analyzeFloorplan` will automatically run placement, timing-driven `trialRoute` and `timeDesign -proto`.
- By default, `placeDesign` will be invoked to generate detailed placement with no overlaps.
- `timeDesign -proto` will use existing routes for delay calculation and timing analysis.

## Analyzing the Floorplan

The `analyze` floorplan step with `set_proto_mode -effort high` 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.

By default, both `-cong` and `-timing` parameters of `analyzeFloorplan` will be enabled when the `-effort` parameter of `set_proto_mode` is set to `high`.

| <b>-cong</b> | <b>-timing</b> | <b>Behavior</b>                                                                                                                                                                             |
|--------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| On           | Off            | <ul style="list-style-type: none"> <li>▪ <code>placeDesign</code> (which honors effort setting)</li> <li>▪ Zero iteration of timing-driven <code>trialRoute</code></li> </ul>               |
| Off          | On             | Only iterations one through four of timing-driven <code>trialRoute</code>                                                                                                                   |
| On           | On             | <ul style="list-style-type: none"> <li>▪ <code>placeDesign</code> (which honors effort setting)</li> <li>▪ Iterations zero through four of timing-driven <code>trialRoute</code></li> </ul> |
| Default      | Default        |                                                                                                                                                                                             |

To do this, there are two use models:

- First Model: Run `analyzeFloorplan` with `-cong` to check congestion and then run

`analyzeFloorplan -timing.`

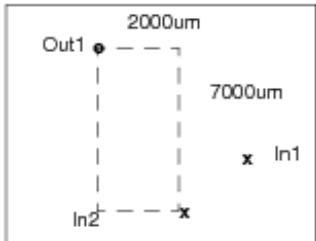
- Second Model: Run `analyzeFloorplan -cong -timing` (equivalent to `analyzeFloorplan`)

**Note:** The first model is recommended since you can find congestion problems and fix them early without waiting for full run time.

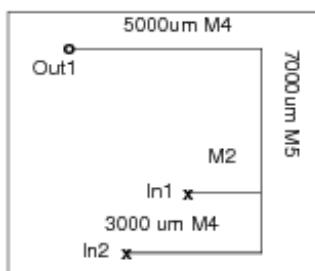
### ***Calculating Delay Using Timing-Driven trialRoute***

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, the low-effort `timeDesign -proto` calculates the delay from `out1` to `in2` 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

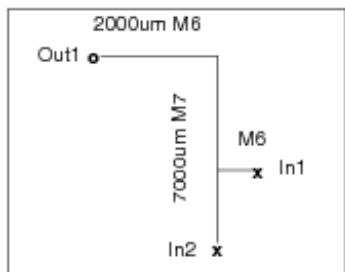


- assuming best and non-detouring layers
  - After running the initial iteration of trialRoute, then the actual routing layer is used and the layer length is multiplied by its delay factor. This results in more delay as compared to the earlier stage.
- For example, M4 delay ( $0.25 * 8000$ ) + M5 delay( $0.25 * 7000$ ) = 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.

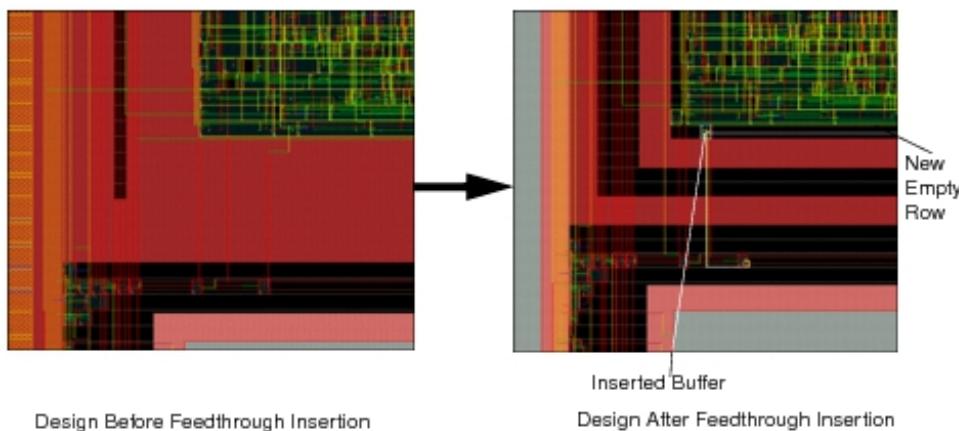
It is calculated as: M6 delay ( $0.14 \times 2000$ ) + M7 delay ( $0.14 \times 7000$ ) = 1260 ps



### 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,

```
ff_add_partition_border_blockage -num_row 1
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

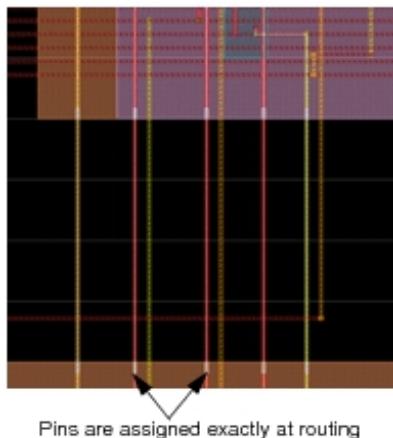
After running trialRoute, you perform pin assignment. To do this:

- 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 need to:

- Run optDesign with a flexModel design to get accurate timing budget.
- Derive timing budgets for partitions in a design.
- Run optDesign -preCTS to optimize design with virtual buffering for accurate budgeting.

For example,

```
setDelayCalMode -ignoreNetLoad false
ff_restore_default_mode_setting
ff_set_dont_touch_for_models 0
setOptMode -preserveModuleFunction true; setOptMode -honorFence true

setBudgetingMode -trialIPO false; setPlanDesignMode -reset; setPlaceMode -reset

    set maxLayer [getMaxRouteLayer; setTRailRouteMode -reset; setMaxRouteLayer $maxLayer
optDesign -preCTS
setBudgetingMode -noHoldView 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 access the prototyping commands and utilities, load the `procs.tcl` script. For example,

```
source SCRIPTS/PROTO/procs.tcl
ff_restoreDesign model_gen
```

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_IS0_MOD_1`, `spi`, and `dma`:

```
restoreDesign DBS/PTN_L1/PTN2.enc.dat dtmf_recv_core
```

```
set_proto_model -module ulaw_lin_conv -type full  
  
set_proto_model -module CPF_ISO_MOD_1 -type full  
  
set_proto_model -module spi -type full  
  
set_proto_model -module 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 Prototyping Foundation Flow

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.

## 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

EDI 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.

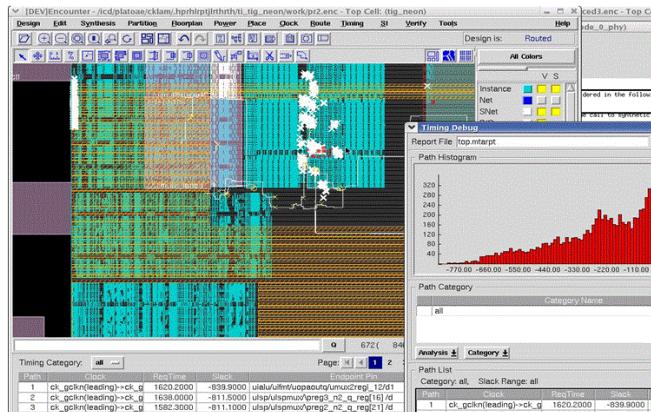
SDP is a semi-custom methodology that requires manual intervention so you need to have detailed design knowledge in order to get better speed, power, and area. The EDI System tool makes it easy for you to try different SDP experiments and evaluate their impact on congestion, timing, and power. However, the tool still relies on the relative placement information you specify for placing and handling SDP elements.

Furthermore, currently EDI System does not identify SDP elements automatically. You must script them based on naming conventions and detailed design knowledge.

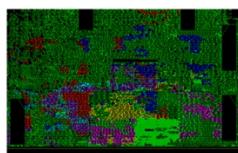
In addition, SDP capabilities can be used for high speed register or flip flop (FF) column methodology to help reduce clock latency and/or skew.

### Benefits of Using SDP

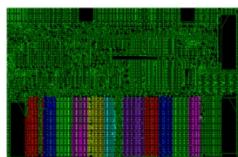
SDP provides a uniform environment for data path and control logic for placement, routing, and timing analysis.



SDP cells can be placed concurrently with other standard cells to get the optimal placement

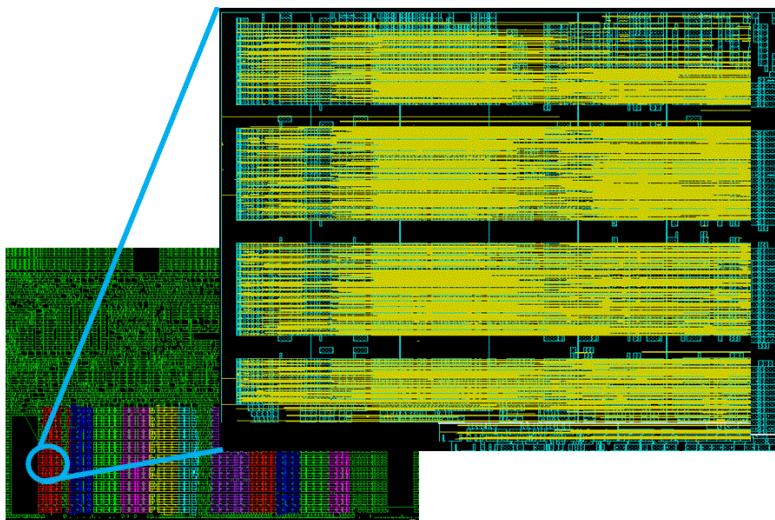


Traditional Placement



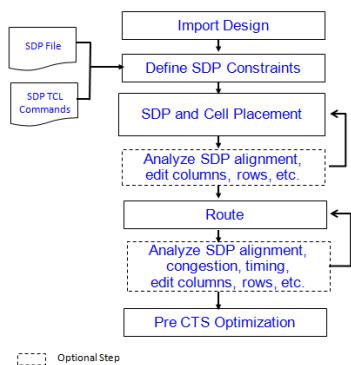
SDP Placement

The main advantage of this SDP placement is that it ensures uniform routing.



### General SDP Flow

The general SDP flow is as follows:

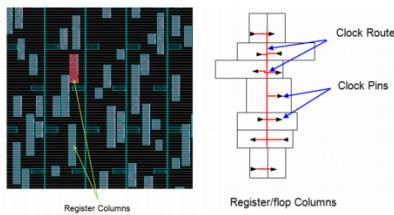


As shown in the flow diagram, after importing the design, you can read in an SDP file to define SDP constraints. Alternatively, you can source a TCL file that has SDP TCL commands, such as `createSdpGroup`, `addSdpGroupMember`, `addSdpObject`, and so on to define relative SDP placement. All SDP TCL commands 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.

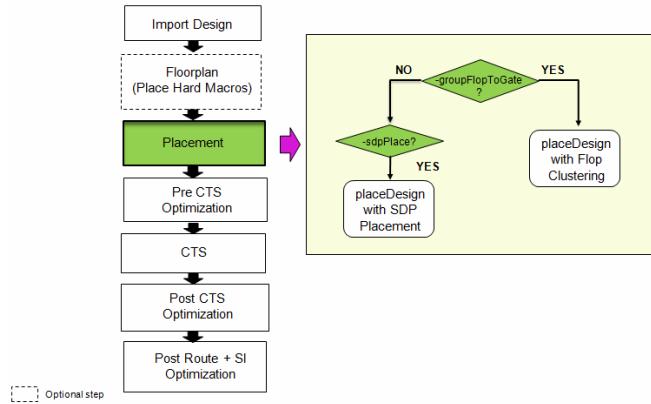


Some advantages of using high-speed register methodology are:

- 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

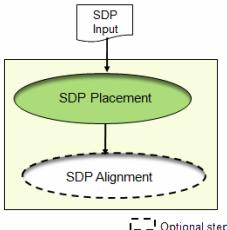


According to this flow, placement can be done in two ways, depending on whether or not you require flop clustering:

- placeDesign with SDP Placement
- Placement with Flop Clustering

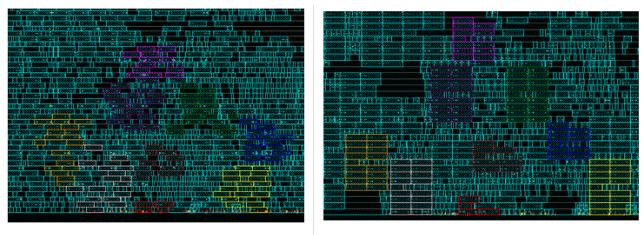
#### placeDesign with SDP Placement

`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 -sdpPlace | placeDesign with -sdpPlace and - sdpaLignment |
|----------------------------|-----------------------------------------------|
|----------------------------|-----------------------------------------------|



**Note:** Cadence recommends that you use `placeDesign -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
```

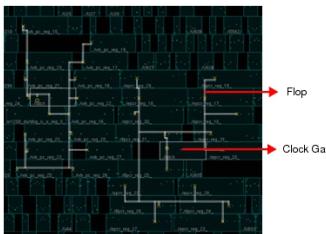
```
placeDesign
```

#### 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.

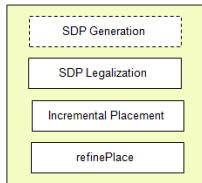
1. To enable flop clustering placement, set the `-groupFlopToGate` option of `setPlaceMode` to `true`.
2. Specify the size of the bounding box for each ICG-flop cluster by using the `setPlaceMode -groupFlopToGateHalfPerim` parameter to set the box's half perimeter. The default bounding box's half perimeter 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

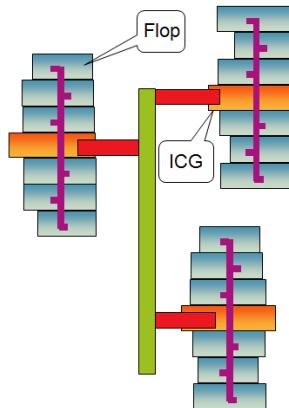
To enable SDP alignment, you must set `setPlaceMode -sdpAlignment true`. The SDP alignment process comprises four sub-steps:

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  $100 * \text{pitch}$  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_recvr_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. EDI System 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 *EDI System Text Command Reference*.

EDI System 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 *EDI System Menu Reference*.

### SDP Relative Placement File

You can bring SDP information into the EDI System 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
```

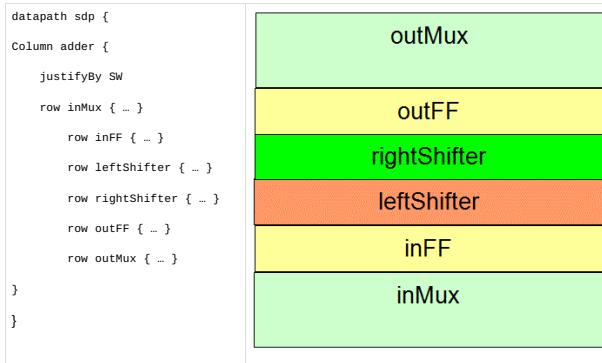
- Allows pre-place location

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.

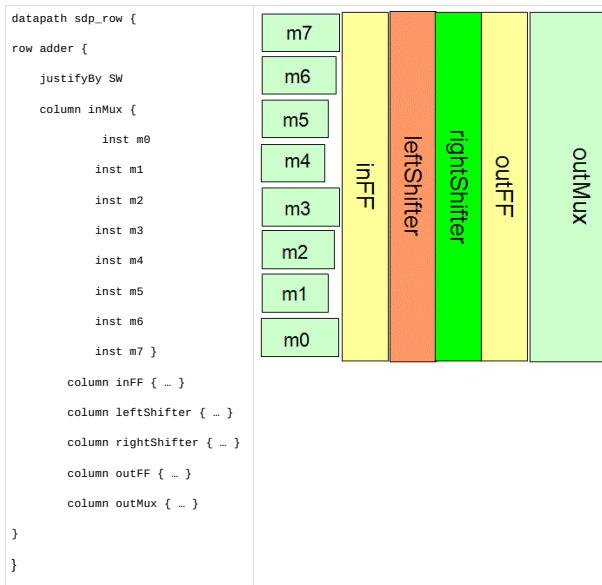
The SDP format provides you the ability to create rows or columns.

### SDP File Examples

Following is an example of an SDP file for creating a column of rows:



SDP also supports hierarchical construction (SDP within SDP, rows within a column and vice versa), as shown below.



#### 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
            | inst instanceName [orient R0|R90|..] [justifyBy ...] [flip X|Y|XY]
            | column name { ... } ]...
    }
}

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
    | inst instanceName [orient R0|R90|...] [justifyBy ...] [flip X|Y|XY]
    | row name { ... } ]...
}
}

# 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**.

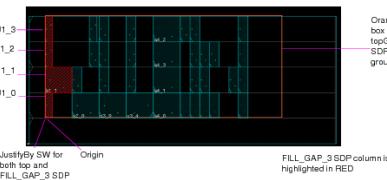
Example 1:

```

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.



- **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. 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).

Following is the example of **skipSpace**:

```

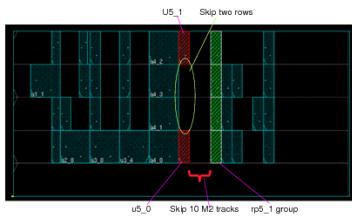
...
column FILL_GAP_9 {
    justifyBy SW

```

```

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>
    }
}
...

```



- **inst** : Specifies one or more instance names. Use this keyword if you want to create the SDP group from specified instances.
- **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.

#### 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:

```

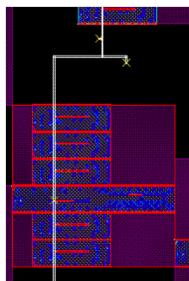
datapath DP_one {
    hierPath PTN1
    column col_arr {
        justifyBy SW
        skipSpace 1
        inst trn_1 orient R0
        row row_3_4 {
            inst top0/g1 orient R0
            inst top0/CKGA orient MY
        }
        row row_3_5 {
            inst top_1/CKGA_dcap0 orient MX
            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 f`

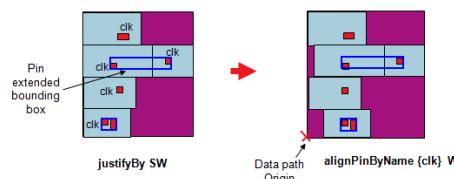
[ justifyBy NW | SW | SE | NE | W| E | N | S | MID]

[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|m|o` parameters to specify the pin names by which you want to align SDPs and the pin edge to be used for alignment.

## Figure 5

- 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 Encounter 10.11-s079_1
#
# Generated on: Fri Feb 25 11:04:50 2011
#
# HostName: icdlnx08s
#
# 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. EDI System 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`.

# Design Methodology for 3D IC with Through Silicon Via

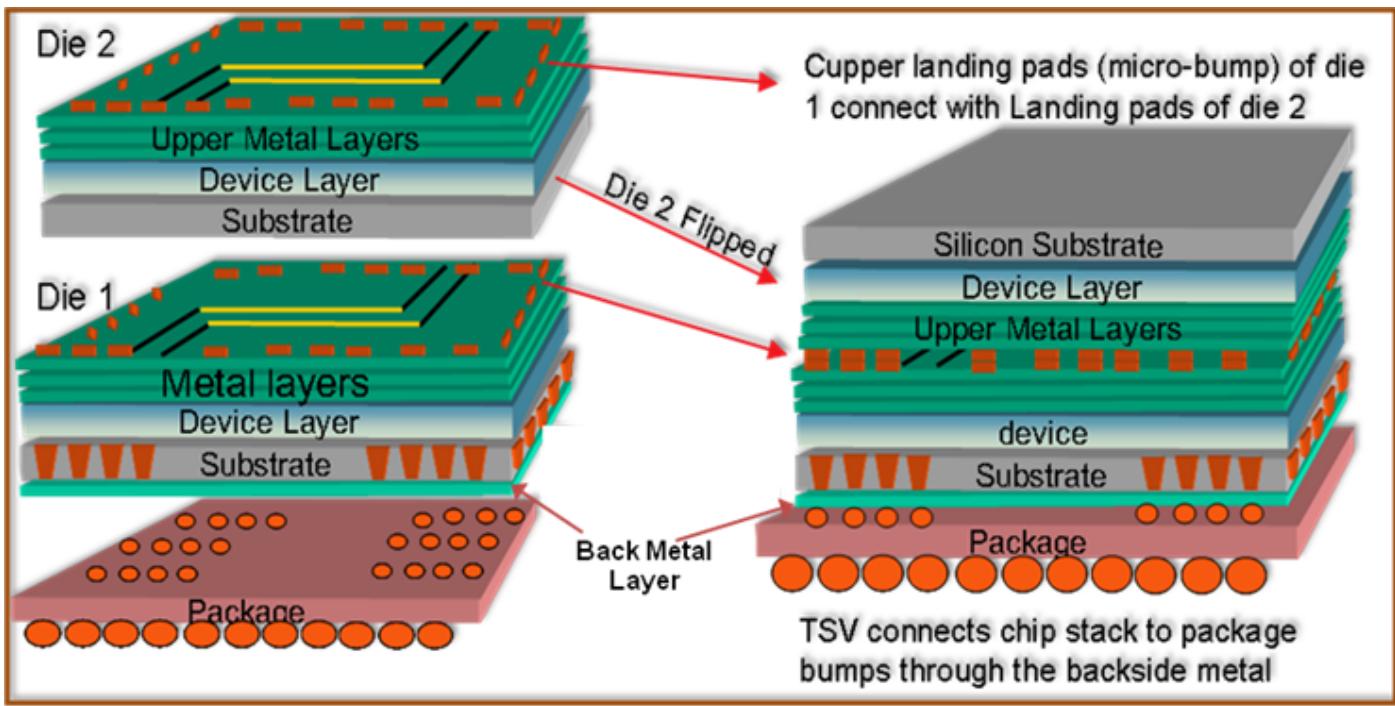
---

- [Overview](#)
- [TSV/Bump/Back Side Metal Modeling in EDI](#)
- [Defining Keep Out Area in Hard Macros](#)
- [3D IC Flow in EDI system](#)
- [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 16-1 Scheme of 3D IC Profile**



The EDI System supports TSV designs. In EDI, 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 EDI, 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

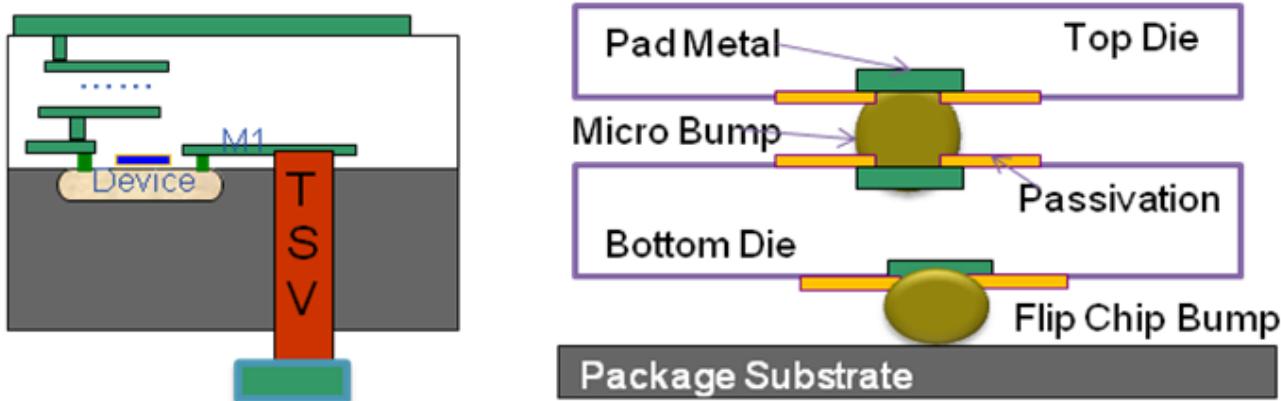
- "Through Silicon Via Design Commands" chapter of the EDI System Text Command Reference.
- "TSV Tool Box" section of the Tools Menu chapter in the EDI System Menu Reference.

## TSV/Bump/Back Side Metal Modeling in EDI

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 EDI. 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 16-2 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. EDI System will not create bumps on 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 ABOVE DIEEDGE ; " ;
END TOPDIE
```

### **A new layer for bottom die edge**

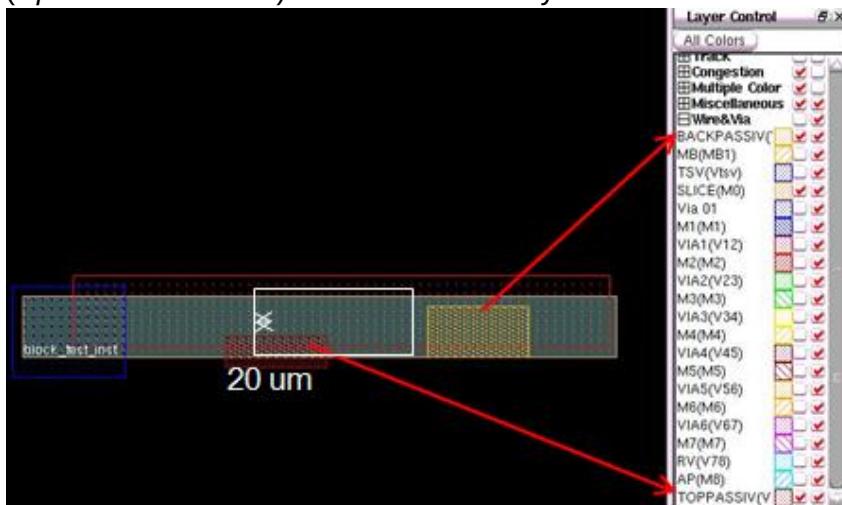
```
LAYER BOTTOMDIE
  TYPE MASTERSLICE ;
  PROPERTY LEF58_TYPE "TYPE BELOW DIEEDGE ; " ;
END BOTTOMDIE
```

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.



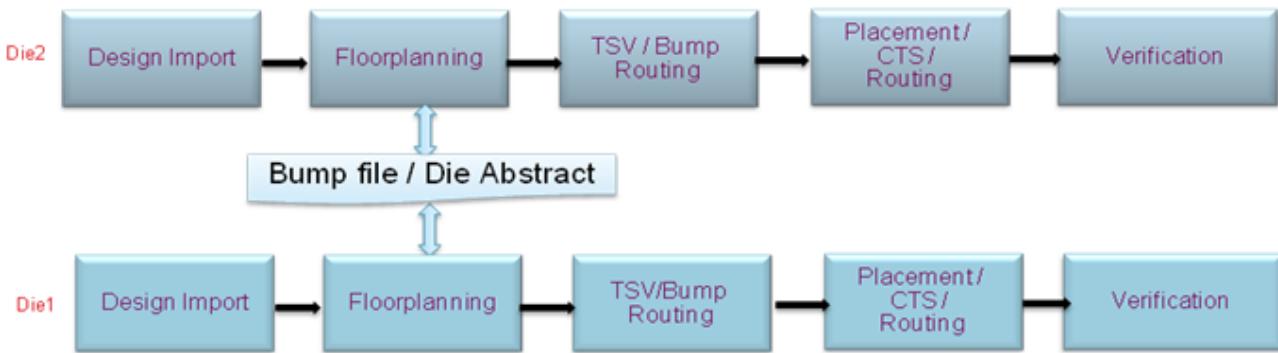
### Check Bump Keep Out Area Violation

Use the `readBumpLocation -checkAlignment` command parameter to check the overlap between the OBS layer and bumps (both front and back bumps). You can use the [verify\\_stacked\\_die](#) command 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 EDI system

The following figure shows the general EDI System 3D IC design flow. Only 2 dies are included in this flow chart, and it can be extended to multiple dies.

**Figure 16-3 TSV Design Flow in EDI**

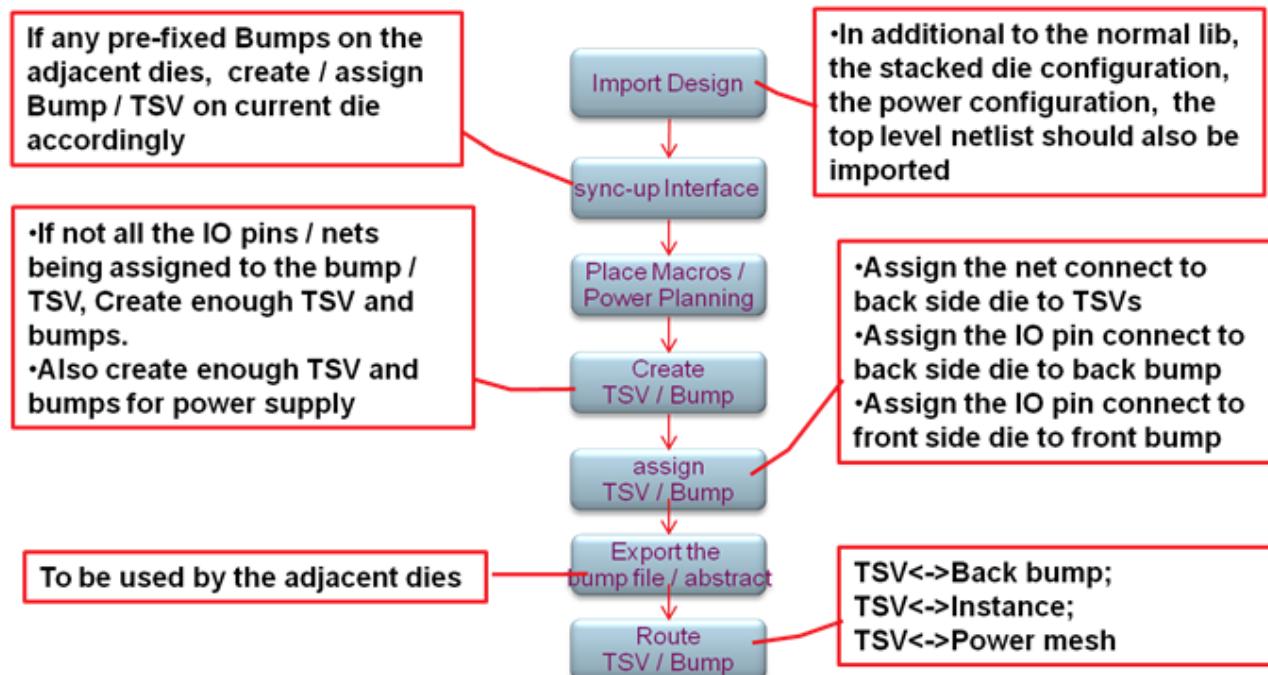


Encounter designs one die at a time. When designing one die, the micro bump and the instance on the adjacent die are honored and the micro bump on the current die is synchronized and optimized accordingly. The design flow for each die is quite similar to the normal design, except the following steps:

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 16-4 shows the detailed floorplan flow for a 3D IC design.

**Figure 16-4 Detailed Floorplan Flow for One of the Dies**



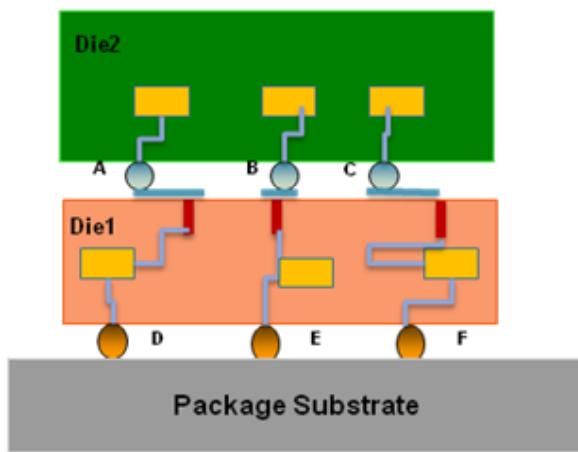
## Design Import

EDI takes the conventional and the additional information for stacked dies as inputs for each die design.

## Stacked IC Verilog Input

EDI 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 16-5 shows an example of the top-level netlist.

**Figure 16-5** Top-Level Netlist Example



```
//die1.v
Module Die1 (A, B, C, D, E, F);
.....
endmodule

//die2.v
Module Die2 (A, B, C);
.....
endmodule

//top.v
module Wrapper(D, E, F);
.....
Die2 Die2_inst (.A(w1), .B(w2), .C(w3));
Die1 Die1_inst (.A(w1), .B(w2), .C(w3), .D(D), .E(E), .F(F));
endmodule
```

### 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 16-5 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

EDI 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 *EDI System Menu Reference Guide*.

## Power Connectivity Input

A text file is used to set up P/G net configuration on each die.

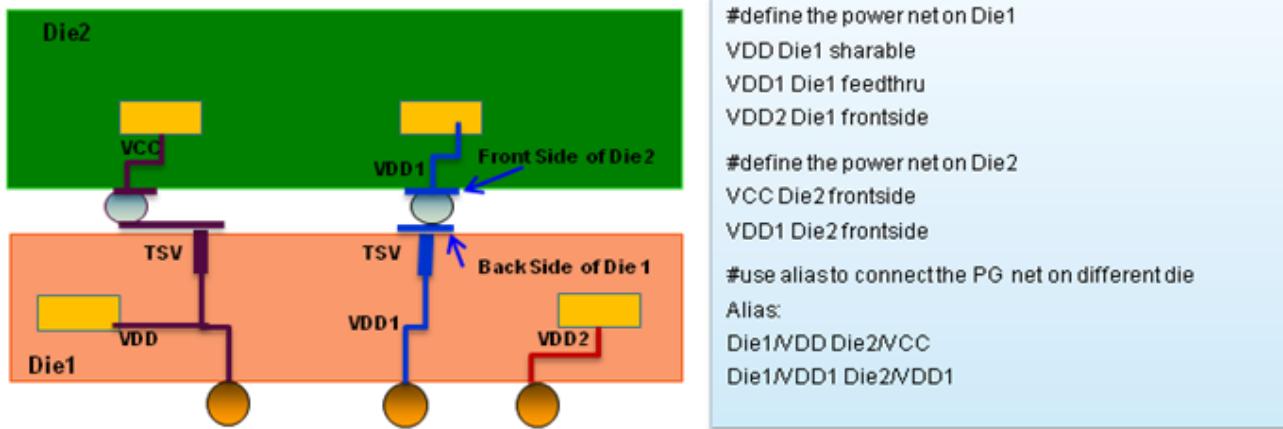
There are four types of P/G nets: frontside, backside, sharable, feedthru.

- 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
- feedthru: the P/G net is not connected to any instance of current; use alias to define the same

P/G net on different dies

Figure 16-6 shows an example of a power connectivity file.

### Figure 16-6 Example of Power Configuration File



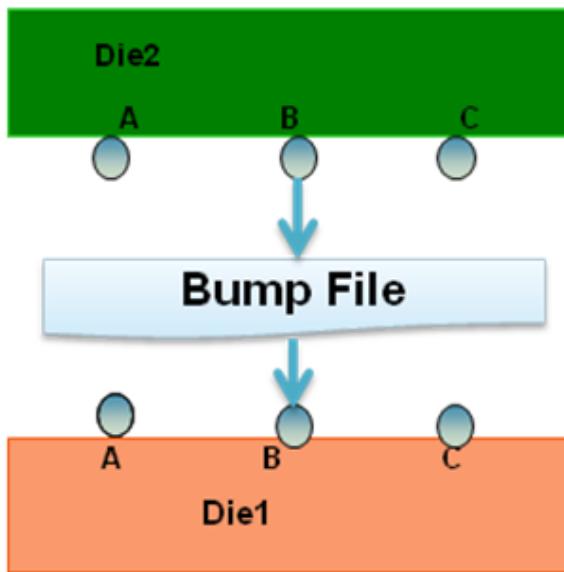
### Related Information

- "readTSVConfig" command in the "Through Silicon Via Design Commands" chapter of the EDI System Text Command Reference.
- "Load Stacked Die Config" form in "TSV Tool Box" section of the "Tools Menu" chapter in the EDI System 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 16-2, the pad of micro bump should be exactly aligned to build the connection between the dies. EDI 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 16-7 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 EDI System Text Command Reference.
- "Exchange Information Between Dies" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the *EDI System Menu Reference Guide*.

## TSV and Bump Manipulation

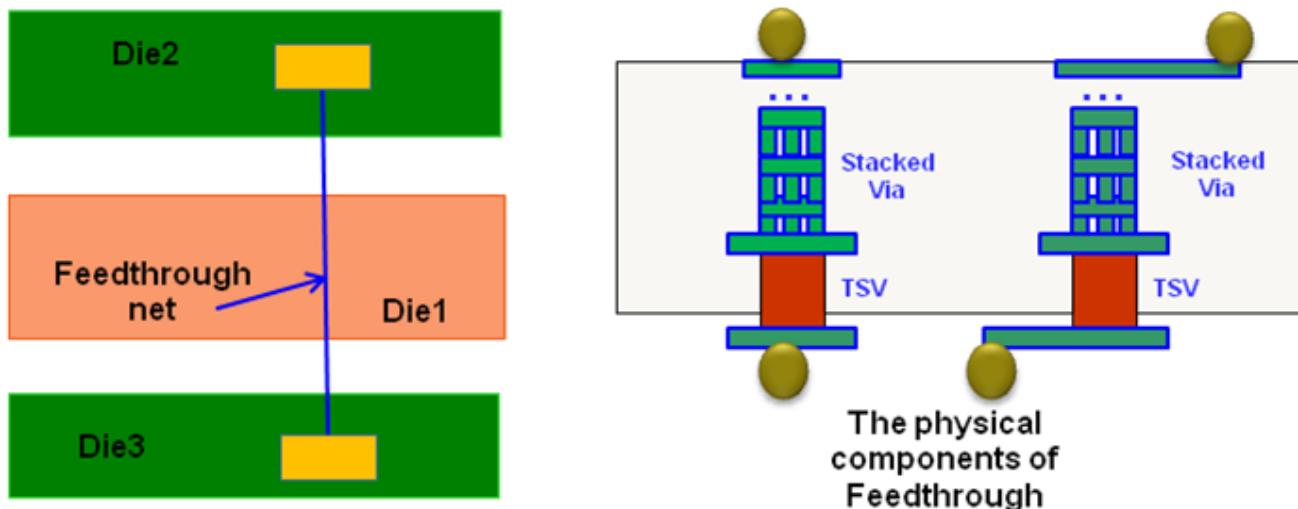
EDI provides the capability to create/delete and assign/unassign TSV and bumps. Run the `addTSV` command to create the TSV and bumps within the user-specified box, or with user specified number. Since TSVs and bumps are the path between the dies, place TSVs and back bumps close to the IOs/Blocks whose 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 16-8.

**Figure 16-8 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 EDI System Text Command Reference.
- "TSV/Bump Manipulation" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the EDI System Menu Reference.

## TSV and Bump Routing

Flip chip router and point to point router supports TSV routing, as shown in the following steps:

1. Route TSV to IO Pad.

```
fcroute -type signal -designStyle <aio|pio> -connectTsvToPad
```

2. Route TSV to back side bump.

```
fcroute -type signal -designStyle <aio|pio> -layerChangeTopLayer <layerName> -  
layerChangeBotLayer <layerName> -connectTsvToBump
```

3. Route TSV to power stripe.

```
fcroute -type power -connectTsvToRingStripe
```

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 EDI System Text Command Reference.
- "Route TSV/Bump" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the EDI System Menu Reference.
- "Flip Chip Methodology" chapter of the EDI System 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.

## Related Information

- "verifyConnectivity" command in the "Verify Commands" chapter of the EDI System Text Command Reference.
- "Verify Connectivity" form in the "Verify Menu" section of the Tools Menu chapter in the EDI System Menu Reference.
- "Identifying and Viewing Violations" chapter in the EDI System User Guide.

---

# 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](#)
- [Creating Differential Routing to Signal Bumps](#)

## 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 Encounter 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 Encounter power routing features to make the final power connections. The SRoute software creates pad rings and routes power and ground nets to the following power structures:

- Block pins
- Pad pins
- Standard cell pins
- Unconnected stripes

Use the SRoute form to specify routing to any or all of these structures. In addition, you can specify whether or not the software should make the connections by changing layers or allowing jogs.

## Before You Begin

Before you can begin power planning, the following conditions must be met:

- The design must be loaded into the current Encounter session.

The following input file is required:

- The design file

Before you can begin power routing, the following conditions must be met:

- The design must have power rings and stripes.

The following input file is required:

- A LEF file that contains technology and macro information.

## Results

After using the power planning software, your design has preliminary power structures that provide

the foundation for hooking up each cell to a power source.

After using the power routing software, your design has power connections between pins of specified nets on the blocks and pads to nearby rings or stripes. Your design is ready for combined power and rail analysis to determine whether power structures and connections provide sufficient power to the design.

## Loading, Saving, and Updating Special Route

To load special route data into a design, use the Load Special Route form. When loading the floorplan, the .fp and .fp.spr files are included. The filename extension entered in the Load FPlan form is .fp, not .fp.spr.

When creating special routes, use the Save Special Route form or the Save Design form to save the special route data plus vias.

Floorplan files are saved with the following extensions:

- .fp -- Contains the general floorplan information.
- .fp.spr -- Contains the special route data.

You can also use the Save DEF form to save special route information in the DEF file.

## 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 *Encounter Text Command Reference*.

## **globalNetConnect Command and Connections for Signal Pins and Power/Ground Pins**

The `globalNetConnect -type net` command automatically connects *only* signal pins to the global net--*not* power or ground pins.

To reconnect power or ground pins to the global net after using the `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 EDI System 12.x maps to the configuration file in EDI System 10.x and earlier.

## **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.

See the "Power Route Commands" in the *Encounter Text Command Reference* for more information.

## Fixing LEF Minimum Spacing Violations

If your design contains violations of the LEF minimum spacing rule, use the [fillNotch](#) command as a post-processing step. The command fill gaps, which removes same net violations between generated

vias and wires or pins where these violations are flagged by the Verify Geometry software. Using this command, you do not have to sacrifice via size by trimming vias in order to fix same net spacing violations.

See "Verify Commands" in the *Encounter Text Command Reference* for more information.

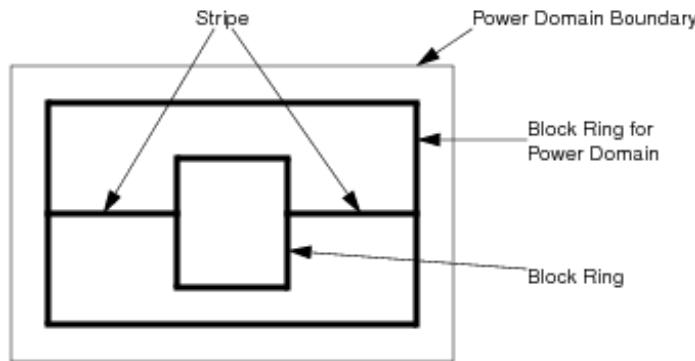
## Adding Stripes to Power Domains

When you use the *Each selected block/domain/fence* option on the [Basic](#) page of the Add Stripes form, stripes are placed according to the location of the power domain ring. Sometimes the location of the power domain ring was not specified at the time the power domain was created. In this case, you must indicate the location of power domain rings to the power planning software by issuing the following command:

```
modifyPowerDomainAttr -rsExts top bottom left right
```

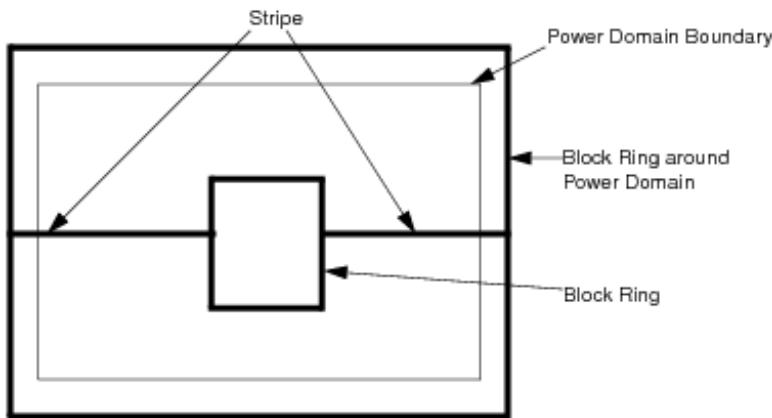
The *top*, *bottom*, *left*, and *right* values specify a distance from the edge of the power domain boundary. Rings between the power domain boundary and the specified distance are considered power domain rings.

- Specify negative values if the power domain ring is inside the power domain boundary. The power planning software considers any ring from the power domain boundary to the specified distance *inside* the boundary to be a power domain ring. When you add stripes, the software trims the stripes at the ring. The stripe also correctly recognizes block rings within the power domain and breaks at those rings if you specify the *Omit stripes inside block rings* option on the [Advanced](#) page of the Add Stripes form. The following illustration shows how the stripe is created when you specify negative values.

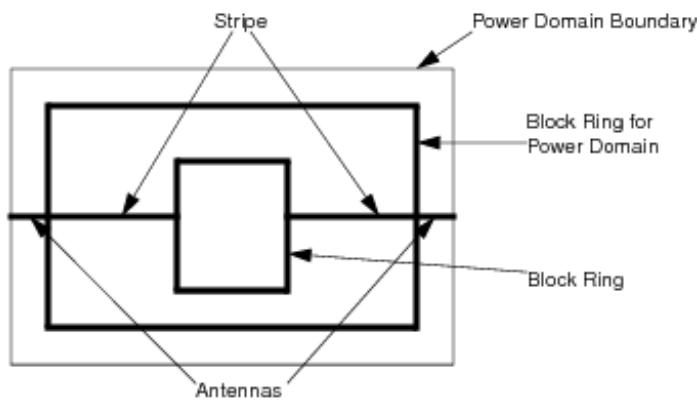


- Specify positive values if the power domain ring is outside the power domain boundary. The power planning software considers any ring from the power domain boundary to the specified

distance *outside* the boundary to be a power domain ring, and extends the stripes to that ring. The stripe also correctly recognizes block rings within the power domain and breaks at those rings if you specify the *Omit stripes inside block rings* option on the *Advanced* page of the Add Stripes form. The following illustration shows how the stripe is created when you specify positive values.



If the location of the power domain ring is not specified, the stripe begins and ends at the power domain boundary. If the power domain ring is inside the power domain, the stripe recognizes it as a block ring. This can cause the stripe to break in the wrong locations if you specify the *Omit stripes inside block rings* option on the *Advanced* page of the Add Stripes form, and can create antennas, as shown in the following illustration.



## Adding Stripe in Multi-CPU mode

In EDI System 12.x, [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 Encounter's *Multi-CPU Processing* GUI form.

**Note:** This will affect any application that can run in multiple-CPU processing mode

## Creating Differential Routing to Signal Bumps

Differential routing creates wires of the same length or configuration between a set of sources and targets.

You can create a constraint file to specify differential pair definitions, as well as shield net definitions and differential group definitions. The following example shows how to set up a constraint file:

```
##### Differential pair definition

###balanced routing with shielding

out[10] PAIR out[11] SHIELDNET VDD

###balanced routing without shielding

out[8] PAIR out[9]

##### Shield net defintition

out[5] SHIELDNET VDD

out[3] SHIELDNET VSS
```

##### Differential group definition

out[15] GROUP out[16] out[17] out[18] resetn

clk GROUP out[12] out[13]

---

## Low Power Design

---

- [Overview](#)
- [Power Domain Shutdown and Scaling](#)
- [Support for the Common Power Format \(CPF\)](#)
  - [CPF Version Support](#)
  - [EDI System Commands Supporting CPF](#)
  - [Loading and Committing a CPF File](#)
  - [Saving a CPF Database](#)
  - [CPF Documentation](#)
- [Multiple Supply Voltage Flat Flow](#)
  - [Preparing Data](#)
  - [Load the design \(init\\_design\)](#)
  - [Floorplanning the Design](#)
  - [Loading and Committing the CPF File](#)
  - [Setting the Power Domain Size](#)
  - [Setting the Power Domain mingap](#)
  - [Adding Power Switches](#)
  - [Verify Power Domains](#)
  - [Adding Well Tap Cells](#)
  - [Planning Power](#)
  - [Placing Standard Cells and Macros](#)
  - [Highlight Power Domains \(Optional\)](#)
  - [Adding Tie High/Low cells](#)
  - [Routing Power](#)
  - [Trial Routing](#)
  - [Optimizing Timing](#)
  - [Synthesizing Clock Trees](#)
  - [Optimizing Timing \(Post CTS\)](#)
  - [Routing the Design](#)
  - [Analyzing Timing](#)
  - [Analyzing Power](#)
  - [Optimizing Timing \(Post-Route\)](#)
- [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 EDI System](#)

- [Example of Top-Level CPF Generated by EDI System](#)
- [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 Encounter® Digital Implementation System (EDI System) software. A non default and non virtual power domain has a fence constraint 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

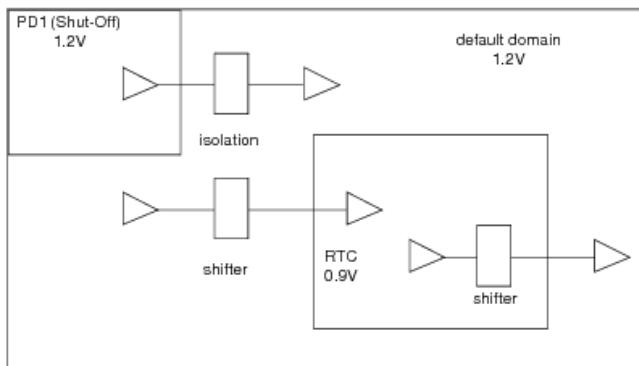
You can reduce power consumption either by shutting down a power domain or operating it at a reduced voltage (voltage scaling).

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 the default domain can share libraries since PD1 and RTC 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 the EDI system 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 EDI viewDefinition.tcl)
- Creating analysis view and defining library set for each power domain (Optional: users can define them in EDI viewDefinition.tcl)
- Defining operating conditions (Optional; users can define them in EDI 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 EDI System software supports the following versions of CPF:

- CPF 1.0
- CPF 1.0e
- CPF 1.1 (default)
- CPF 2.0

### EDI System 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 EDI System for error checking
- [commitCPF](#)  
Executes (commits) the CPF commands within the EDI System environment

## Saving a CPF Database

The [saveDesign](#) command prevents you from saving an EDI System database from obsolete EDI System 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 EDI System database with [saveDesign](#).

Complete the following steps to save a CPF database:

1. Save the CPF file from an EDI System database [saveCPF](#)
2. Exit the current session
3. Start a new session
4. Load the CPF file into EDI System [loadCPF](#)
5. Execute the commands in the CPF file [commitCPF](#)
6. Save the design [saveDesign](#)

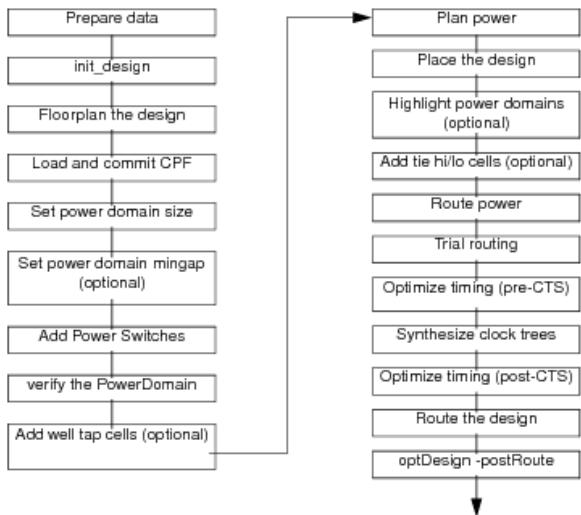
## CPF Documentation

For more information about CPF, see the following documents:

- [EDI System Text Command Reference](#)  
Documents the following EDI System commands supporting the CPF flow. Descriptions on obsolete native EDI System commands are provided.
- [EDI System Menu Reference](#)  
Documents the loadCPF/saveCPF GUI.
- [EDI System 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.

## Multiple Supply Voltage Flat Flow

The MSV flat flow is based on CPF. The flow includes the following steps:



This section includes the following topics:

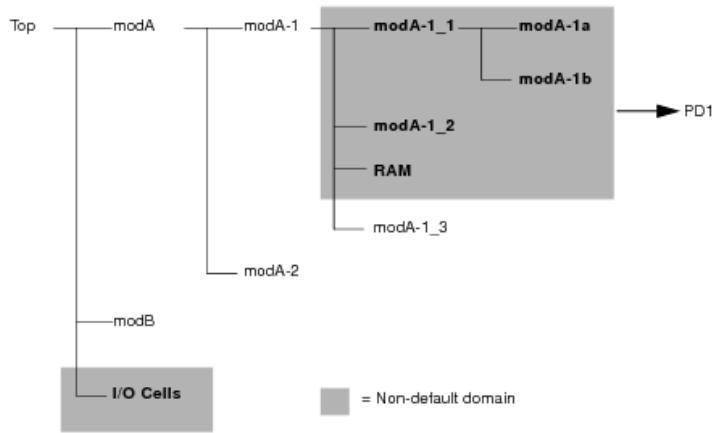
- Preparing Data
  - Load the design (`init_design`)
- Floorplanning the Design
- Loading and Committing the CPF File
- Setting the Power Domain Size
- Setting the Power Domain mingap
- Adding Power Switches
- Verify Power Domains
- Adding Well Tap Cells
- Planning Power
- Placing Standard Cells and Macros
- Highlight Power Domains (Optional)
- Adding Tie High/Low cells
- Routing Power
- Trial Routing
- Optimizing Timing
- Synthesizing Clock Trees
- Optimizing Timing (Post CTS)
- Routing the Design
- Analyzing Timing
- Analyzing Power
- Optimizing Timing (Post-Route)

## Preparing Data

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

- Preparing the Netlist
- Preparing Libraries
- Preparing CPF including level shifters

### Preparing the Netlist



- Power domains can contain one or more modules as members.
- A power domain can also be a partition in low power hierarchical design.

### Preparing Libraries

In an MSV flow, the same cell may have instances in different regions operating at different voltages. For such a cell, characterize a different NLDM (or table look-up) for each operating voltage.

Your design must satisfy the following requirements:

- All library files loaded into the system to represent different voltage characterizations must have different library names.
- Any given library must have one and only one voltage associated with it.
- Min and max libraries may share the same library name.

Two or more power domains may have the same set of min/max libraries bound to them, as is usually the case in shutdown methodologies. You can then associate each library with a unique operating condition (and hence voltage) associated with it; for example:

| Library File   | Library Name | OpCondName |
|----------------|--------------|------------|
| stdcell-1V.lib | Stdcell_1V   | 1V_min     |

|                |            |        |
|----------------|------------|--------|
|                |            | 1V_max |
| stdcell-2V.lib | Stdcell_2V | 2V_min |
|                |            | 2V_max |
| stdcell-3V.lib | Stdcell_3V | 3V_min |
|                |            | 3V_max |

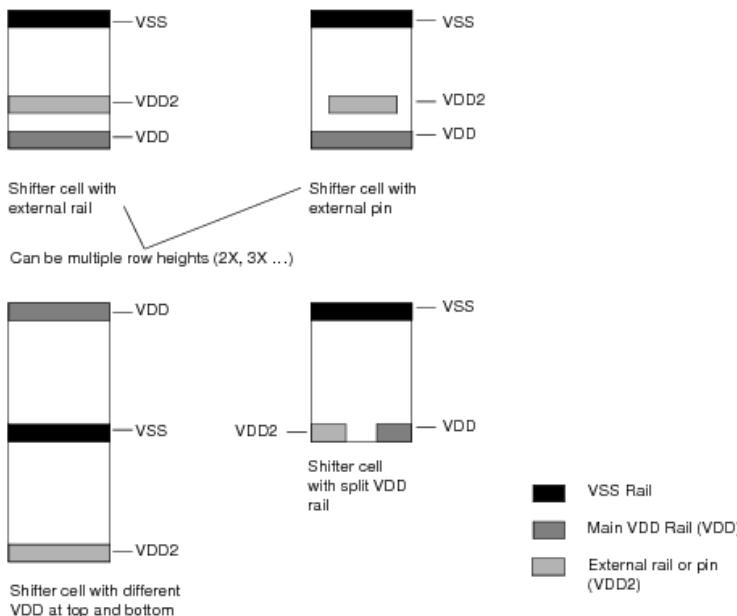
If you specify min and max timing libraries when you import the design, the timing library information appears in the `viewDefinition.tcl` or CPF. MMMC may be defined in CPF or `viewdefinition.tcl`. The command `init_design` can read the timing information based on either `viewDefinition.tcl` or CPF.

If you specify common timing libraries when you import the design, the timing library information appears in the `viewDefinition.tcl` or CPF.

#### Defining Level Shifter Cells

Level shifter cells are used to communicate between power domains of different voltages. The main characteristic of the level shifter cell is the presence of a secondary power rail or pin that operates at a different voltage than the primary power pins.

The EDI System software supports the following level shifter types:



The secondary power pin is modeled as a rail or pin. You must describe the secondary rail or pin as a

power pin. The following example shows the LEF description of a pin named VDDL:

```
PIN VDDL
DIRECTION INOUT ;
USE power ;
SHAPE ABUTMENT ;
PORT ;
LAYER METAL1 ;
RECT 5.860 1.700 8.110 1.990 ;
```

**Note:** The shifter cell can be an integral multiple of the standard cell height.

## Load the design (init\_design)

From EDI System 11.1, the syntax \*.conf is eliminated. The design data is read through the init\_design procedure. In Low Power design, users can define the MMMC in either viewDefinition.tcl or CPF. If viewDefinition.tcl is used, please define each domain binding through update\_delay\_corner -power\_domain. If CPF is used, init\_design calls a special loadCPF to generate the view definition file. The init\_design will load libraries and setup 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 issues an error.

The command init\_design works as follows:

1. If there is a viewDefinition.tcl for Low Power design, init\_design does not call special loadCPF to re-generate viewDefinition.tcl
2. If there is no viewDefinition.tcl for Low Power design, init\_design will call a special loadCPF to generate viewDefinition.tcl based on the CPF.
3. If both viewDefinition.tcl and CPF are provided and viewDefinition.tcl has higher priority, init\_design does not call the special loadCPF.
4. If there is no viewDefinition.tcl and CPF does not define analysis view, init\_design will error out.

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/...

Thus, if there is any network change, you can re-define the Library Primary Path and not need change anything else. To do this, the special loadCPF will generate the viewDefinition.tcl

**Note:** Since EDI System 11.1 supports only MMMC, Low Power only supports only MMMC Flow .

1. A. if design does not have viewdefinition.tcl, CPF should contain create\_analysis\_view.
2. B. If design has viewdefinition.tcl, CPF does not need to have create\_analysis\_view. But, CPF needs library set, nominal condition and power mode.
3. C. If CPF does not have create\_analysis\_view and design does not have viewdefinition.tcl, commitCPF will display an error.

## Floorplanning the Design

Create a floorplan for the design. For more information about creating floorplans, see "[Floorplanning the Design](#)" in the *EDI System User Guide* .

1. Place I/Os.
2. Use the Move/Shape/Resize icon to move control logic into the power domain.
3. Manually move the power domains into the core area.
4. (Optional) Change the shape of a power domain to a rectilinear shape by adding power domain cuts. The *Cut Rectilinear* icon allows you to interactively create power domain cuts that define rectilinear power domains. The power domain cuts mask out portions of the power domain to enable you to see the shape of the power domain.
5. (Optional) Preplace hard macros.
6. Save the floorplan file (.fp).

Use the saveFPlan command or select *Design - Save - Floorplan* from the GUI.

The following example shows the kind of information saved:

- Power domain information: *design .fp*  
POWERDOMAIN: NAME=TDSPCore  
  
VOLT=0.9680 LAYER=0 ALWAYSON=0  
  
NRLIB=4  
  
TIMELIB=tcbn451pbwp\_c060907bc0d88(common)  
  
TIMELIB=tcbn451pbwp\_c070208bc1d10d88(common)  
  
TIMELIB=tcbn451pbwphvt\_c070208bc0d88(common)

```
TIMELIB=tcbn45lpwp_c070208bc0d88(common)

MINGAPTOP=16.0000 MINGAPBOT=16.0000 MINGAPLEFT=16.0000

MINGAPRIGHT=16.0000

RSEXTTOP=50.0000 RSEXTBOT=50.0000 RSEXTLEFT=50.0000 RSEXTRIGHT=50.0000

ROWFLIP=3 SITE=core ROWSPACETYPE=2 ROWSPACING=0.0000

MODULE=HDRDID1BWPHVT POWER=(VDD_TDSPCore:VDD)

POWER=(VDD_TDSPCore_R:TVDD) GND=(VSS:VSS)

MODULE=LVLHLD2BWP POWER=(VDD_TDSPCore:VDD) GND=(VSS:VSS)

MODULE=PTLVLHLD2BWP POWER=(VDD_TDSPCore_R:TVDD) GND=(VSS:VSS)

MODULE=TDSP_CORE_INST POWER=(VDD_TDSPCore_R:TVDD)

POWER=(VDD_TDSPCore:VDD) GND=(VSS:VSS)

END_PD

POWERDOMAIN: NAME=PLL

...

END_PD
```

## Loading and Committing the CPF File

1. To read-in the CPF that contains CPF commands, use the following command:

```
loadCPF fileName
```

This command reads the file and performs lint, parsing, and semantics checking. This command does not execute any of the command within the CPF file.

1. To commit the CPF file, use the following command:

```
commitCPF
```

This command executes the CPF commands loaded by `loadCPF`. Running this command does the following:

- Creates power domains

- Creates logical power/ground net connections
- Specifies timing libraries for power domains
- Inserts shifter cells and isolation cells
- Replaces regular registers with state-retention (SRPG) registers

## Setting the Power Domain Size

→ To do specify the power domain physical size, use the following command:

```
setObjFPlanBox
```

## Setting the Power Domain mingap

→ (Optional) To set the power domain mingap, use the following command:

```
modifyPowerDomainAttr -minGaps
```

## Adding Power Switches

→ To add power switches, use the following command:

```
addPowerSwitch
```

The tool supports two types of power switch insertion: ring or column style. For more information, see Power Shutdown Techniques.

## Verify Power Domains

Use the [verifyPowerDomain](#) command to verify that the following conditions were met:

```
verifyPowerDomain -gconn -place -xNetPD
```

- Power and ground pins of instances within a power domain are connected to the power and ground nets of that power domain.
- Only those cells assigned to a power domain are placed within the boundary of the power domain.
- The correct level shifters are inserted on all nets crossing power domains.  
**Note:** The `verifyPowerDomain` command checks that libraries specified for each power domain of each `dc_corner` are complete for each power domain member list.

Correct all errors before continuing.

## Adding Well Tap Cells

→ To add well tap cells, use the following command:

```
addWellTap
```

For more information, see ["Leakage Power Optimization Techniques"](#)

## Planning Power

Each domain must have complete power and ground rings. To create the power plan, use one of the following approaches:

- Use the automatic power planner (APP) to assign a template to each power domain. For more information, see [Power Planning and Routing in the EDI System User Guide](#).
- Use GUI or text commands

To create rings, complete the following steps:

1. Create core rings (no changes for MSV).
2. Create block rings for power domains and hard macros.

To use the GUI to create rings around a power domain, choose *Power - Power Planning - Add Ring*. Select the *Block ring(s) around: Selected domain/fences/ reefs* option.

The equivalent text command is `addRing -type {block_rings -around power_domain}`

Create block rings for hard macros (no changes for MSV).

To create stripes for a selected power domain or for all power domains, complete the following steps:

1. Create stripes in either of the following ways:
2. Select a power domain, then use the *Each selected block/power domain/fence* option on the [Basic](#) tab of the Add Stripes form to create power stripes for a selected power domain.
3. The equivalent text command is `addStripe` with the `- over_power_domain` parameter set to `1`.
4. Use the *All domains* option on the [Basic](#) tab of the Add Stripes form to create power stripes for all power domains.
5. The equivalent text command is `addStripe` with the `- all_domains` parameter set to `1`.

**Note:** To create a mesh, you must use the `addStripe` command twice: once for vertical stripes and once for horizontal stripes.

**Note:** If the power and ground nets for the stripes do not match the power domain power and ground nets, a warning message displays.

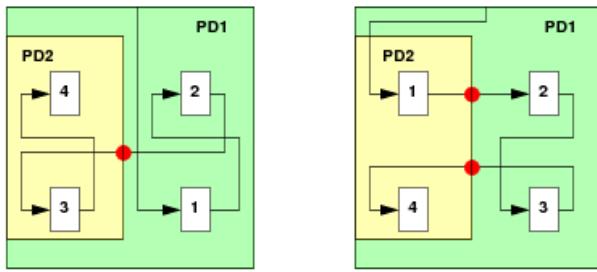
Create stripes to connect power domain rings to core rings.

## Placing Standard Cells and Macros

Place standard cells that have not already been placed. Use, `setPlaceMode -timingDriven placeDesign`

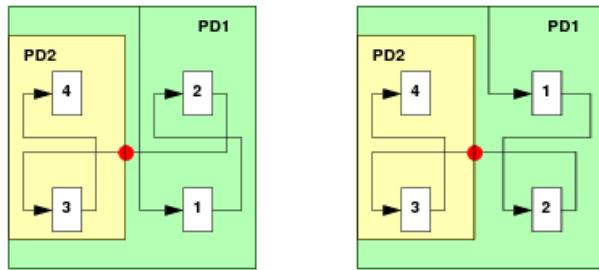
This software recognizes power domains automatically and ensures that the fences are respected. In scan reordering, to maintain the hierarchical ports so that shifters and isolation cells will not be changed, use the following command: `setScanReorderMode -keepHierPort`

The following figure shows scan reordering without `-keepHierPort`:



As shown in this figure, the software adds two hierarchical ports.

The following figure shows scan reordering with `-keepHierPort`:



A single hierarchical port is maintained.

The software places all standard cells and hard macros within the boundary of the power domain to which they are assigned. In addition, voltage level shifters are placed around the power domain boundary.

Voltage level shifters are placed along the inside edge of the power domain boundary, on the row ends. For top and bottom rows, multiple shifter cells are placed at row ends, if needed.

#### Highlight shifters (Optional)

To highlight shifters, use the following command: `reportShifter -highlight`

As an alternative, from the main menu, you can select *Power - Multiple Supply Voltage - Highlight Power Domains* to highlight the shifters.

**Note:** Shifter information is not saved in the database.

#### Verify power domains again (Optional)

To verify power domains (optional) and ensure that the design has no violations, use the following command: `verifyPowerDomain -xNetPD fileName -isoNetPD fileName`

## Highlight Power Domains (Optional)

After the design is placed, you can highlight the power domain in the following ways:

- Using the Floorplan View  
Select a power domain from the Floorplan view. When you switch to the Physical view, all power domain members are highlighted, except for I/O pads.
- Using the Design Browser  
The software displays power domains as groups in the Design Browser. Select the Group object to view all power domains listed in the PDGroups category. Select one of the power domains and click on the *Highlight* button. All power domain members (including the I/O pads) that were assigned to that domain are highlighted in the Physical view.
- Using the Text Command  
Use the `hilitePowerDomain` command to highlight all power domain members, including I/O pads, assigned to the power domain; for example:

```
hilitePowerDomain PD2
```

- Using the Highlight Power Domains GUIs:  
[\*Power - Multiple Supply Voltage - Highlight Power Domains\*](#)

This form has three tabs:

- [\*Power Domain\*](#)  
Highlights power domains, P/G nets, level shifters, and/or power switches in a power domains. You can highlight non-default power domains only.
- [\*Signal Net/HLS Cell\*](#)  
Enables you to select categories of signal nets or Hvt, Lvt, or Svt cells.
- [\*Highlight Set\*](#)  
Enables you to customize the highlight colors.

## Adding Tie High/Low cells

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

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

## Routing Power

The EDI System software uses the `sroute` command to route the followpin connections and the additional power pins contained in level shifters, just as it routes regular followpins.

→ To use the power router, use the following command:

```
sroute
```

The `sroute` command does the following:

- Connects power/ground nets from end to end, terminating at the power rings.
- Connects voltage level shifters pins to the closest segment of the ring surrounding the

power domain.

You can also use the sroute command to route level shifter pins as well as primary nets within a domain. For example:

```
sroute -powerDomains { TDSP } -secondaryPinNet { vdd } -allowJogging 1 -  
allowLayerChange 1 -nets { vdd_lp vss }  
  
sroute -powerDomains { DEFAULT } -secondaryPinNet { vdd_lp } -allowJogging 1  
-allowLayerChange 1 -nets { vss vdd }  
  
clearDrc
```

→ To use the *Sroute* GUI, complete the following step.

Select *Route - Special Route* to display the *Sroute - Basic* form.

- If you select the *Standard cell pins* option on the *Basic* tab of the SRoute form, SRoute connects power and ground nets from end-to-end, terminating at the power rings.
  - [Standard cell pins](#) automatically recognize voltage level shifters and connect the standard cells to the correct power nets.
  - If you select the *Level shifter pins* option, sroute connects voltage level shifter pins to the closest segment of the ring around the power domain.
- For more information, see *Sroute - Basic* in the "Route Menu" chapter of the *EDI System Menu Reference*.

→ To route always-on pins in SRPG cells, specify the following command before you run sroute:

[setPGPinUseSignalRoute](#)

For example, NanoRoute connects cell RSDF to pin TVDD, and cell PTBUFF to pin TVDD:

```
setPGPinUseSignalRoute RSDF:TVDD PTBUFF:TVDD  
routePGPinUseSignalRoute -nets VDD_TDSPCore_R
```

## Trial Routing

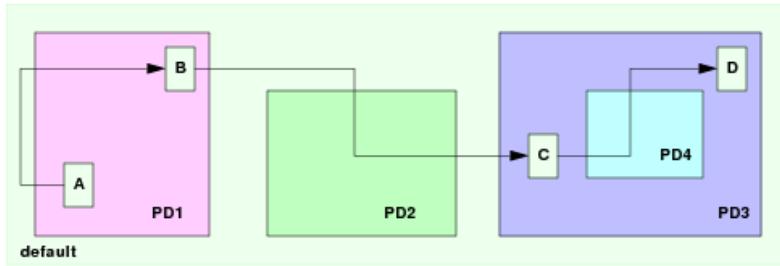
By default, the [trialRoute](#) command is not MSV-aware, so you must use the following options to [setTrialRouteMode](#):

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

If you have nested power domains, use the following `setTrialRoute` parameter:

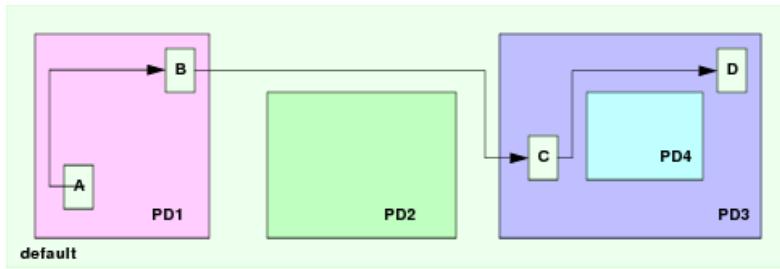
- `-handlePDComplex true`

The following figure shows `trialRoute` behavior without MSV-aware settings:



- Net from A to B is routed outside of PD1
- Net from B to C is routed through PD2
- Net from C to D routed through PD4

The following figure shows trial routing when you specify `setTrialRouteMode - handlePDComplex true` and `- handleEachPD true`:



- Net from A to B is routed only inside PD1
- Net from B to C is routed only through the Default power domain
- Net from C to D is routed only inside PD3

## Optimizing Timing

→ To optimize timing at this phase of the design, use the following command:

```
optDesign -preCTS
```

**Note:** In pre-CTS mode, the tool automatically synthesizes the buffer tree, but you must first specify always-on buffers in the CPF file for always-on buffering.

Instances that belong to different power domains are bound to different timing libraries. Instances are bound to TLF or .lib libraries when you create power domains. Each power domain has a set of associated timing libraries. During timing optimization, new instances (added or changed) are associated only with a timing library that is part of the power domain.

You do not need to set any special options to optimize timing for an MSV design.

Depending on the stage at which you want to optimize timing, use one of the following commands:

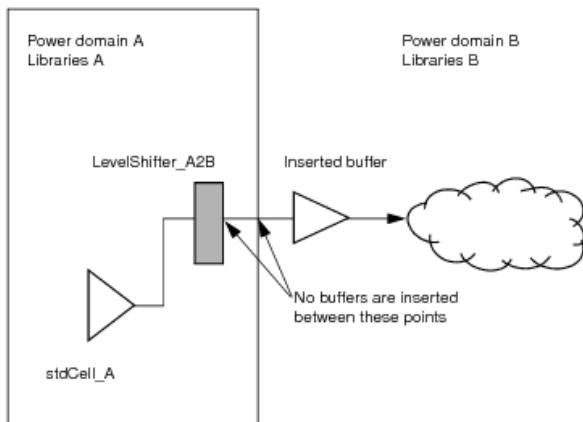
- `optDesign -preCTS`
- `optDesign -postCTS`

- optDesign -postRoute

Inserted instances are assigned to the appropriate power domain, and power and ground pins are connected to the correct power and ground nets. The software places the inserted cells within the appropriate domain boundary.

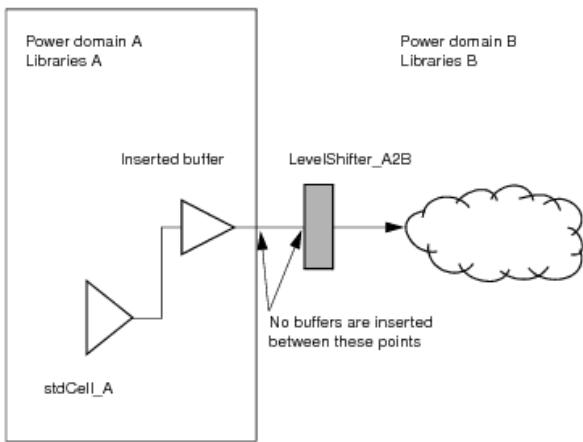
**Note:** The same scenarios apply to isolation cells.

The following scenario shows how the software optimizes timing across a net that crosses power domains of different voltages without always-on buffer.



- Logically, the buffer instance name is in B HInst, and the master cell is from library B.
- Physically, the buffer is placed as close as possible to the level shifter on the opposite domain location determined by the need to fix timing.

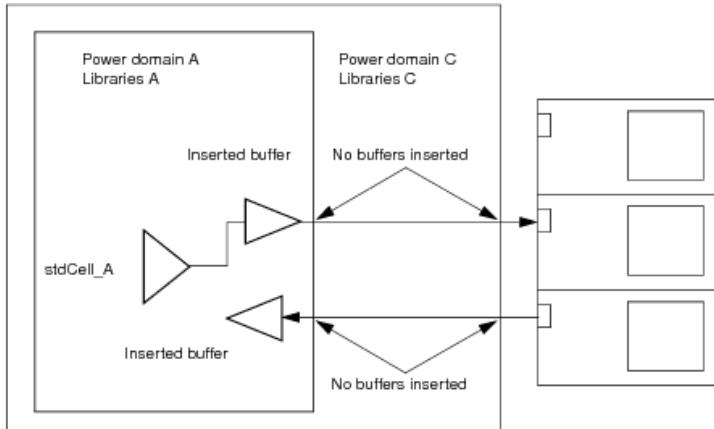
The following scenario shows how the software optimizes timing inside a power domain that does not contain a level shifter.



- Logically, the buffer instance name is in A HInst, and the master cell is from library A.
- Physically, the buffer is placed as close as possible to the level shifter on the opposite domain

location determined by the need to fix timing.

The following scenario shows timing optimization behavior for nets connecting for I/Os.



- Logically, the buffer instance names are in A HInst, and the master cells are from library A.
- Physically, buffers are placed within power domain A only.
- The same applies to isolation cells.

## Synthesizing Clock Trees

1. Before running [clockDesign](#), use the following command to make `clockDesign` power-aware:

```
setCTSMode -powerAware true
```

1. To synthesize clock trees, run [clockDesign](#).

For all clocks within a domain, CTS selects buffers from the domain's libraries and places buffers within the domain's boundary.

For all clocks crossing power domains, CTS assumes that level shifters are present. CTS does the following:

- Establishes a single entry/exit point for each domain
- Selects buffers from appropriate libraries
- Places the buffers within domain boundaries
- Performs cloning and decloning

## Optimizing Timing (Post CTS)

→ To optimize timing in post-CTS mode, use the following command:

```
optDesign -postCTS
```

## Routing the Design

Before running NanoRoute, use the following command to make nanoRoute MSV-aware: `setNanoRouteMode -routeHonorPowerDomain true`

To route the design, run `globalDetailRoute`.

## Analyzing Timing

Each power domain has a set of associated timing libraries. Instance cells that belong to different power domains are bound to corresponding timing libraries, which allows you to perform timing analysis with no further changes for MSV.

→ To analyze timing, run `timeDesign` as you would ordinarily.

## Analyzing Power

Power analysis software recognizes power domains and calculates power values accordingly.

To perform power analysis for a design with MSV, choose *Power - Rail Analysis* and use the Edit Pad Location form to create a power pad location file for each net.

The power analysis software can use an instance power file to calculate instance IR drop instead of a global net voltage.

You can specify that power analysis reports be organized by net. If you specify a report name, the report for all nets is placed in that one file. If you do not specify a report name, a separate file contains each report. For example, if the net names are `vdd1` and `vdd2`, the report names are `vdd1.report` and `vdd2.report`.

For more information about how power analysis supports MSV, see "Running Power Analysis with EDI System," in the Power Analysis chapter.

## Optimizing Timing (Post-Route)

→ After routing, use the following command to optimize timing:

```
optDesign -postRoute
```

## Multiple Supply Voltage Top-Down Hierarchical Flow

This section discusses the following topics:

- Overview
- 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

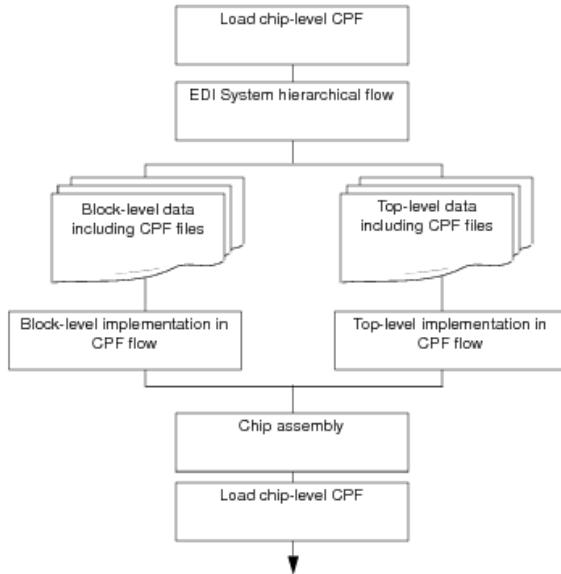
- Chip Assembly

## Overview

The EDI System tool supports the low power top-down CPF-based hierarchical flow built on the regular EDI System 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

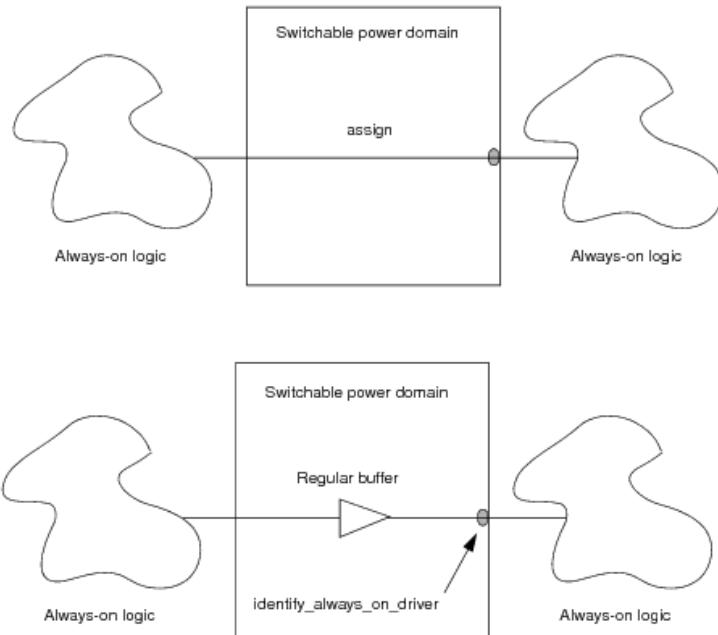
In the low power flow, when you commit CPF, the tool replaces assign statements with always-on buffers for the net connecting to always-on logic or a net whose driver is always-on. The tool identifies these nets automatically as follows:

- 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

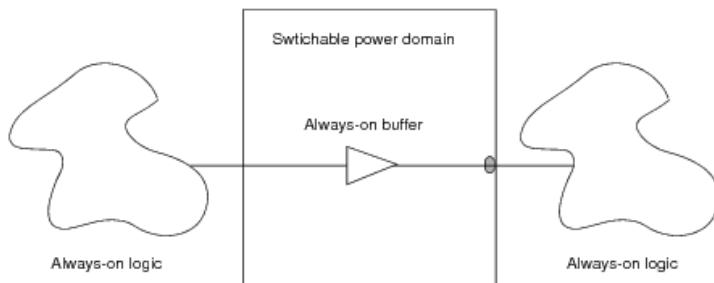
The tool can also insert a feedthrough buffer in a partition that resides in a shut-off power domain.

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:

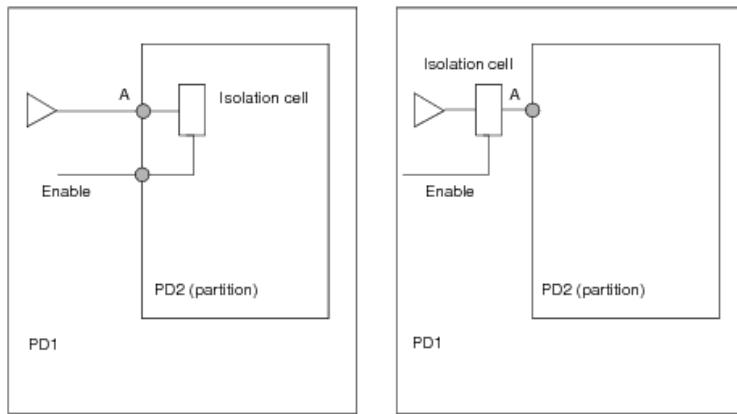
1. Derive the MMMC timing budget by `deriveTimingBudget` and `saveTimingBudget`. For more information, see the "[Timing Budgeting](#)" chapter of the *EDI System User Guide*.
2. Use `savePartition` to generates CPF files for each block-level design (partition), and a top-level CPF file for top-level 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.

- ❶ 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 EDI System

**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/A0wc0d72.lib}

define_library_set -name tdsp_wc_0v792 \
-libraries { ../../LIBS/N45/timing/wc0d72.lib}

create_operating_corner -name WC08COM_A0 \
-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_recv_core_dull.sdc}

create_analysis_view -name AV_LO_FUNC_MAX_RC1 \
-mode PM_LO_FUNC \
-domain_corners { AO@WC08COM_A0 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 {HDRRID 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_A0
-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 EDI System

```
.  
  
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/tcbn45lpwp_c060907wc.lib}

define_library_set -name ao_wc_0v792 \
-libraries { ../LIBS/N45/timing/tcbn45lpwp_c060907wc0d72.lib}

define_library_set -name tdsp_wc_0v792 \
-libraries { ../LIBS/N45/timing/tcbn45lpwp_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} {A0 A0} }

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 A0 \
-default

update_power_domain -name A0 \
-internal_power_net {VDD}

create_global_connection -net VDD_TDSPCore \
-domain A0 \
-pins VDDL

create_global_connection -net VDD \
-domain A0 \
-pins VDD

create_global_connection -net VSS \
-domain A0 \
-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}

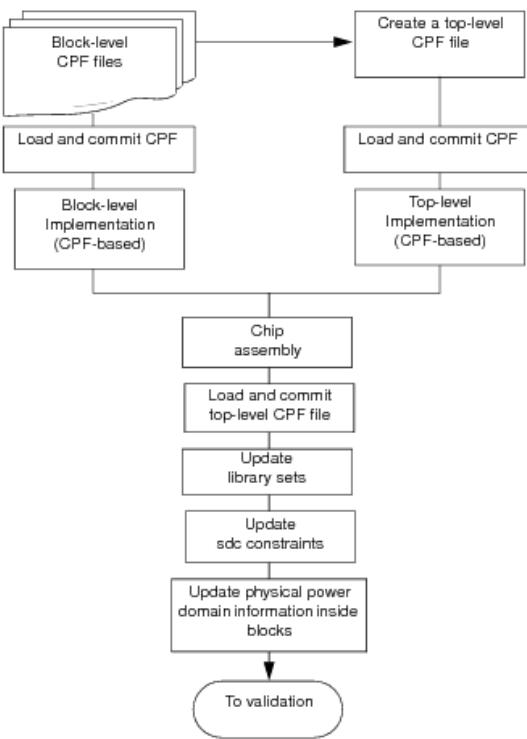
end_design
```

## Multiple Supply Voltage Bottom-Up Hierarchical Flow

This section discusses the following topics:

- Block-Level Implementation
- Top-Level Implementation
- Chip Assembly

The EDI System tool supports the low power bottom-up CPF-based hierarchical flow built on the regular EDI System 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,

1. Load and commit the block-level CPF files.
2. Implement the blocks using the block-level CPF implementation flow.

After completing block-level implementation,

1. Save the block-level design in def format for chip assembly.

`saveDesign -def`

1. 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 EDI System.

To implement the design,

- Use the CPF implementation flow using the hierarchical top-level CPF file.

After completing top-level implementation,

- Use the following command to save the top-level design in DEF format:

```
saveDesign -def
```

## Chip Assembly

- To assemble the design's physical data, use the following command:

```
assembleDesign
```

After chip assembly,

1. To restore the lower power setup for chip-level verification and chip finishing, load and commit the top-level hierarchical CPF file.
2. (Optional) Update power domain physical information inside block.
  - a. To update power domain shape, use the following command:

```
setObjFPlanBox
```

1. 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.

1. Report the total leakage power in the design.

```
reportPower -leakage
```

If you want to obtain a report file, run  
reportPower

-  
leakage  
with the  
-outfile *fileName*  
option.

The following example is a leakage report showing the total leakage power in microwatts, along with cell usage statistics. For each library, the number of cells used in the design and the total leakage power dissipated by the cells are listed.

```
Total leakage power = 785.079708uW
Cell usage statistics:
Library normalVt, 49265 cells (64.855650%), 733.007529uW (93.367269%)
Library highVt, 26696 cells (35.144350%), 52.072179uW (6.632725%)
```

1. 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.

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

```
reportPower -leakage -outfile fileName
```

### Optimizing Leakage Power While Running optDesign (Recommended)

You can optimize non-critical path logic for leakage power by using the setOptMode command in the following ways:

- If leakage power optimization is a second priority after timing convergence, then use `setOptMode -leakagePowerEffort low` after `placeDesign`. The leakage power optimization is automatically done by `optDesign -postRoute`.
- If leakage power optimization is as critical as timing convergence, then use `setOptMode -leakagePowerEffort high` right after `placeDesign`. The leakage power optimization is done by each `optDesign` step.

**Note:** When `setOptMode -leakagePowerEffort` is set to `low` or `high`, the hold fixing steps ensure to not insert any low-voltage threshold gates. Only high-voltage threshold gates are used to fix the hold-time violations.

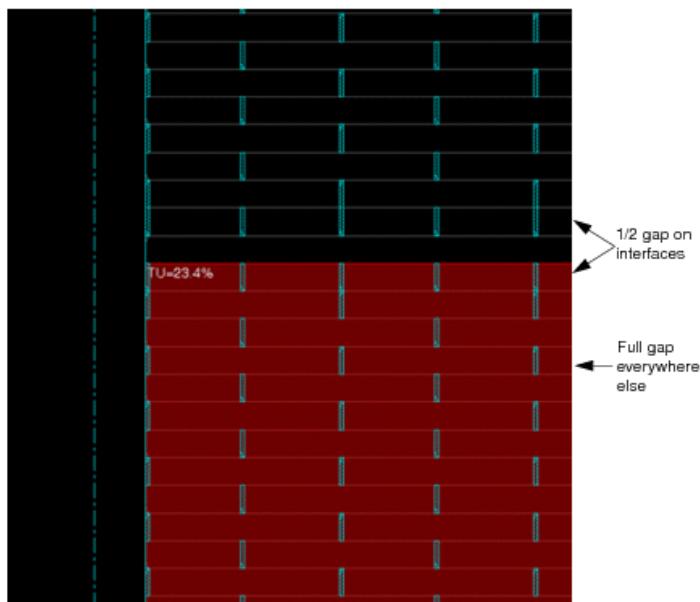
### Substrate Biasing

Substrate biasing is another technique for reducing leakage power. Changing the body voltage of the field effect transistor (FET) affects both the threshold voltage and the static leakage current.

To bias the substrate, insert biasing cells into a region of the design. In EDI System, 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:

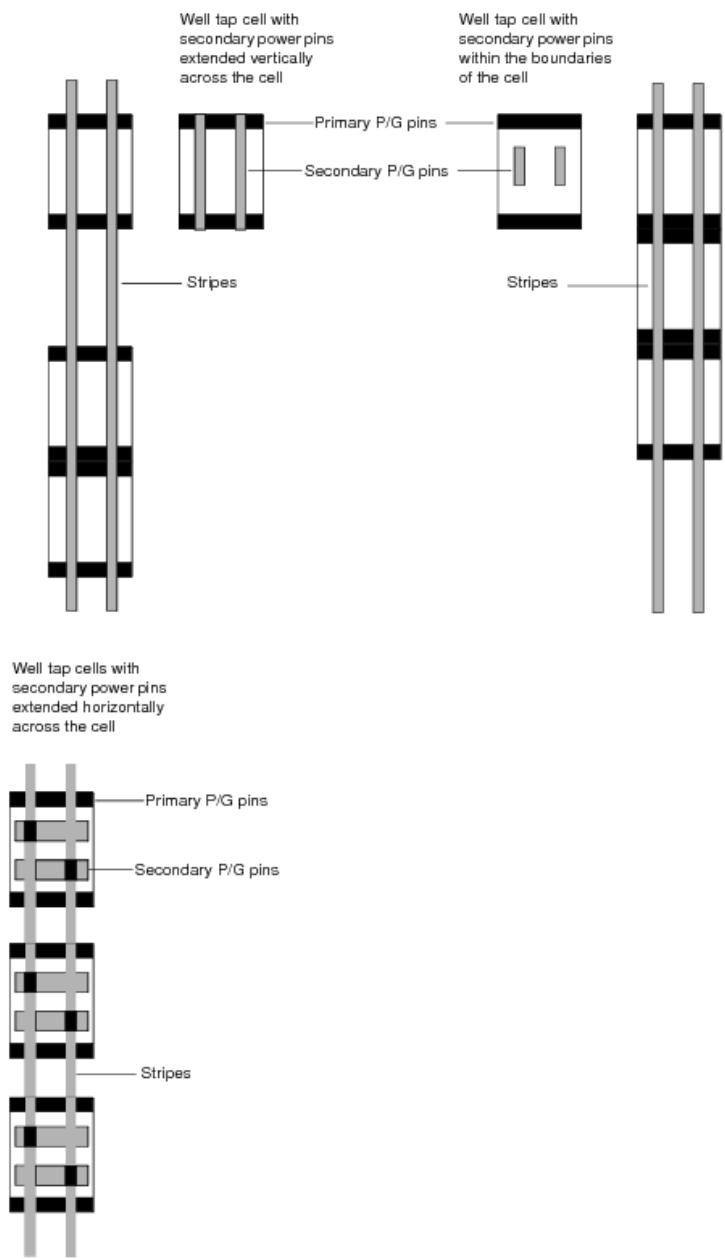


For well tap cells, you must add stripes to connect the secondary power/ground pins in the vertical or horizontal direction.

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.



## 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. 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:

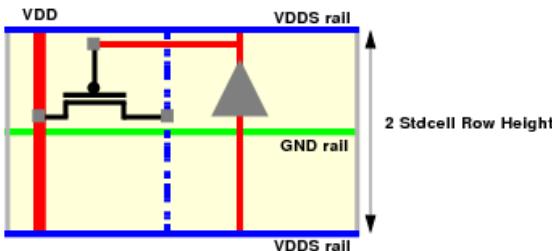
- For power planning, to ensure that the power stripes connect to the power switch cell
- NanoRoute connects enable signals. Abutment depends on the physical layout of the power switch cell.

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.

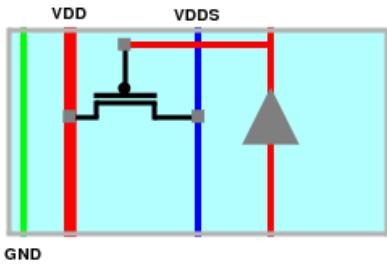
## Buffer Styles

The following figure shows column switch cell, which contains a buffer. This cell has a height of two times the standard cell height.



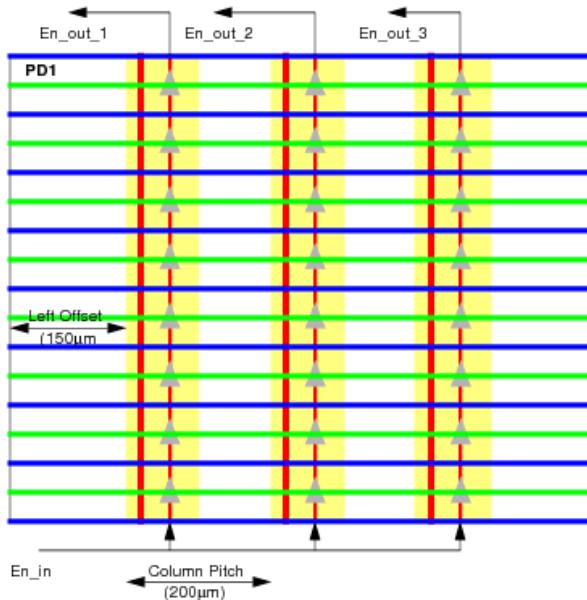
The following figure shows a ring switch cell, which contains a buffer. The cell has the same height as a standard cell. Ring switch cells can also contain two buffers with different directions.

**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.

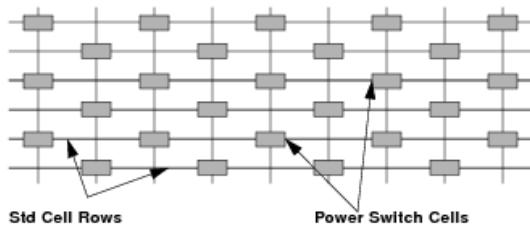
- Area in which the tool can place switches.
- Power/ground pin connections.
- Many other options.

Instead of using the text command, you could use the menu command as follows:

→ From the main EDI System 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.

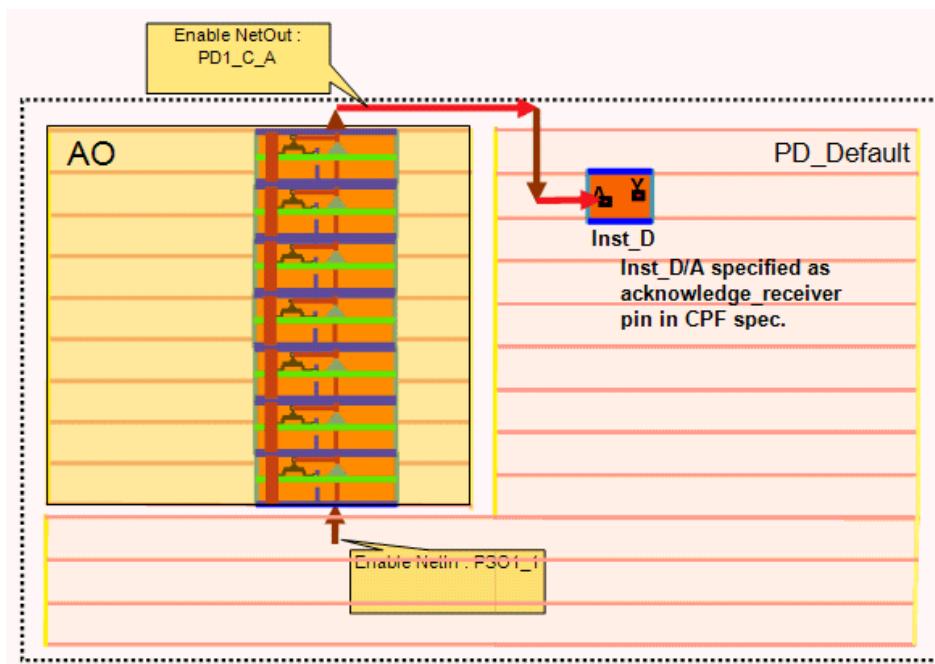
### Attaching the Acknowledge Receiver Pin

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 PS01_1 -globalSwitchCellName COLUMN_SW -leftOffset 3 -horizontalPitch 100
```

Verilog file after addPowerSwitch is run:

```
BUFXH Inst_D (.Y(switch_out), .A(PD1_C_A));
...
COLUMN_SW 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

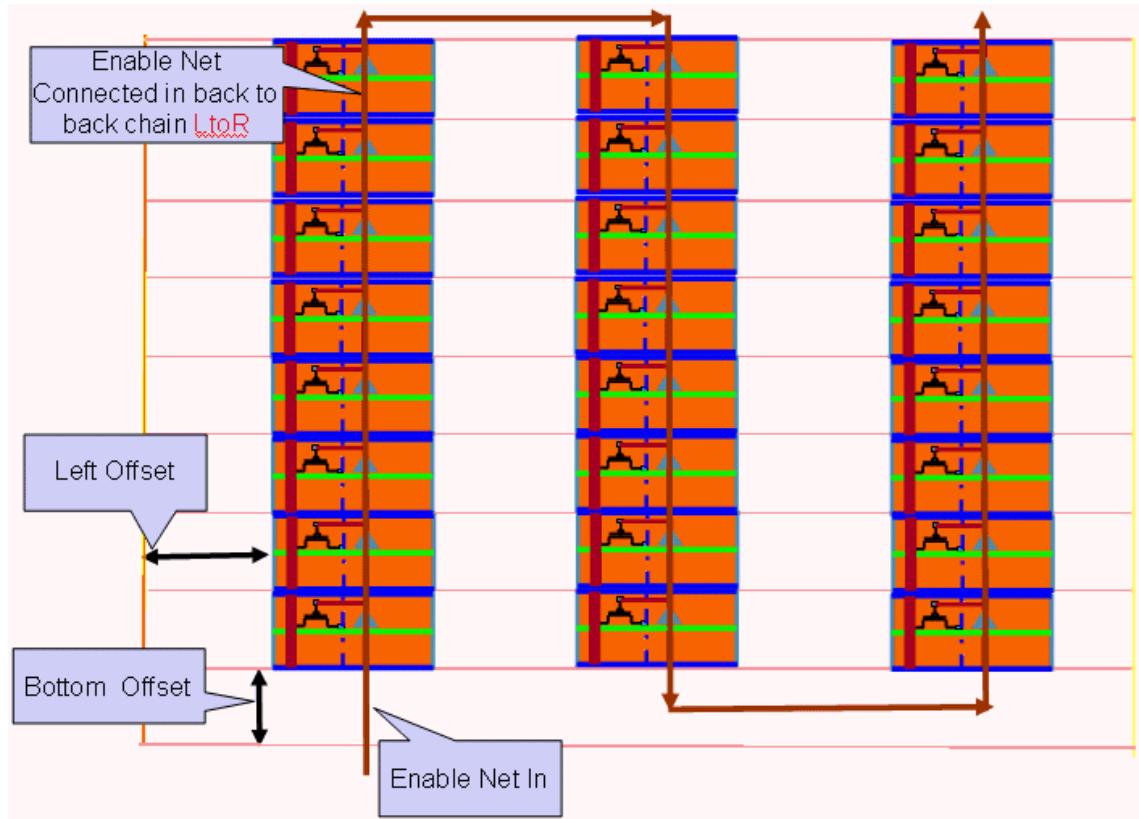
By default, the enableNetIn is connected to the bottom of each column and the enableNetOut exits

from the top of each column, in parallel. The following commands let you create columns with daisy-chain enables:

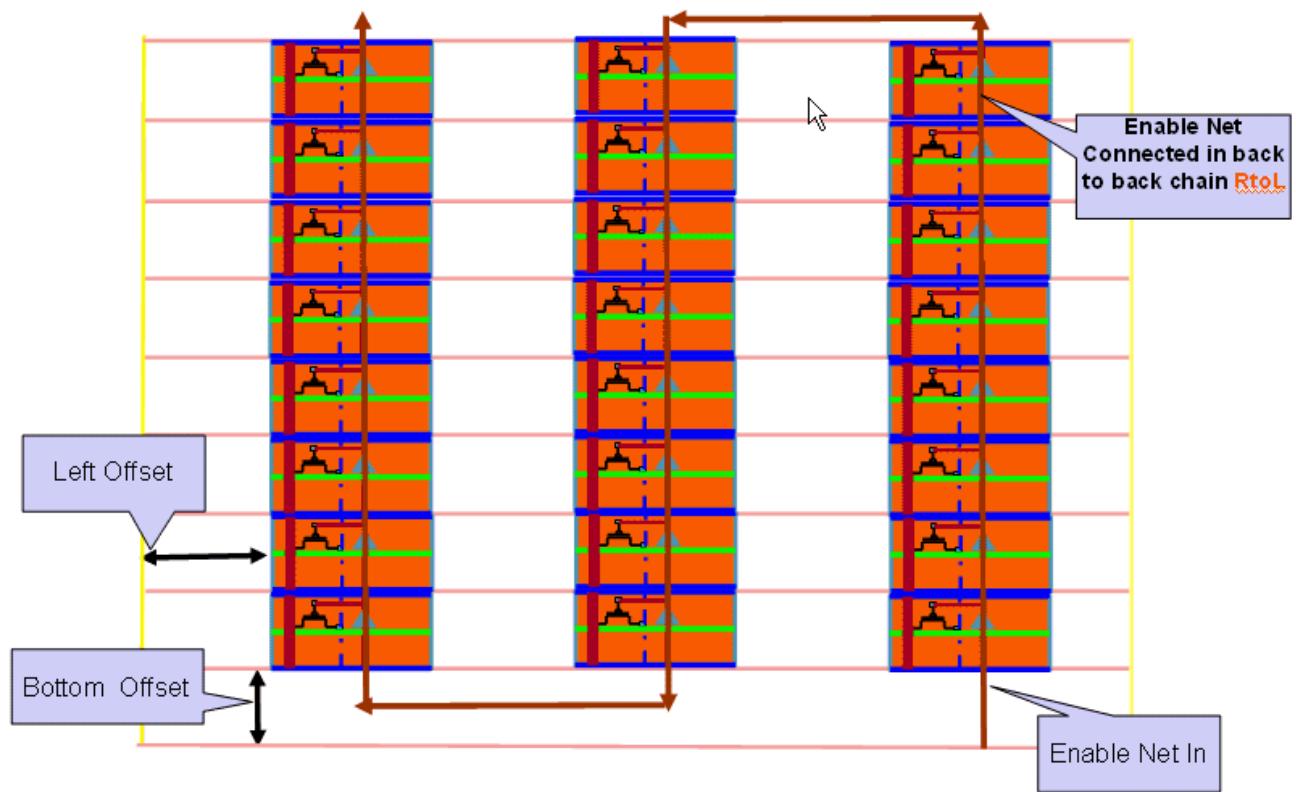
- -backToBackChain

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

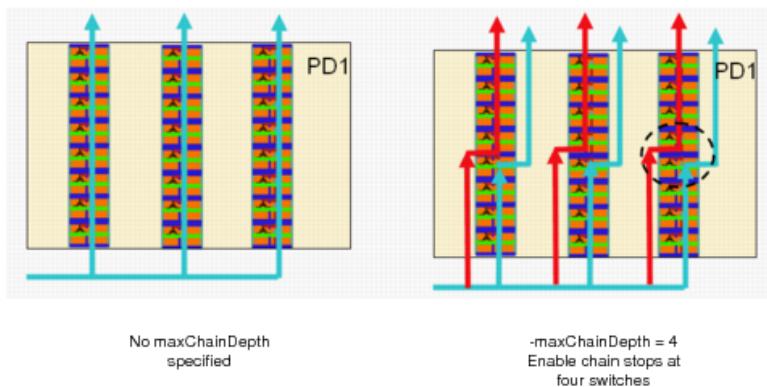
You can control the ramp-up time for the power domain by specifying the number of column switches are allowed in an enable chain before a new chain is started.

→ 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 EDI System tool can automatically create acknowledge trees that you would otherwise build manually. The acknowledge tree collects enable signals exiting the power domain and funnels them

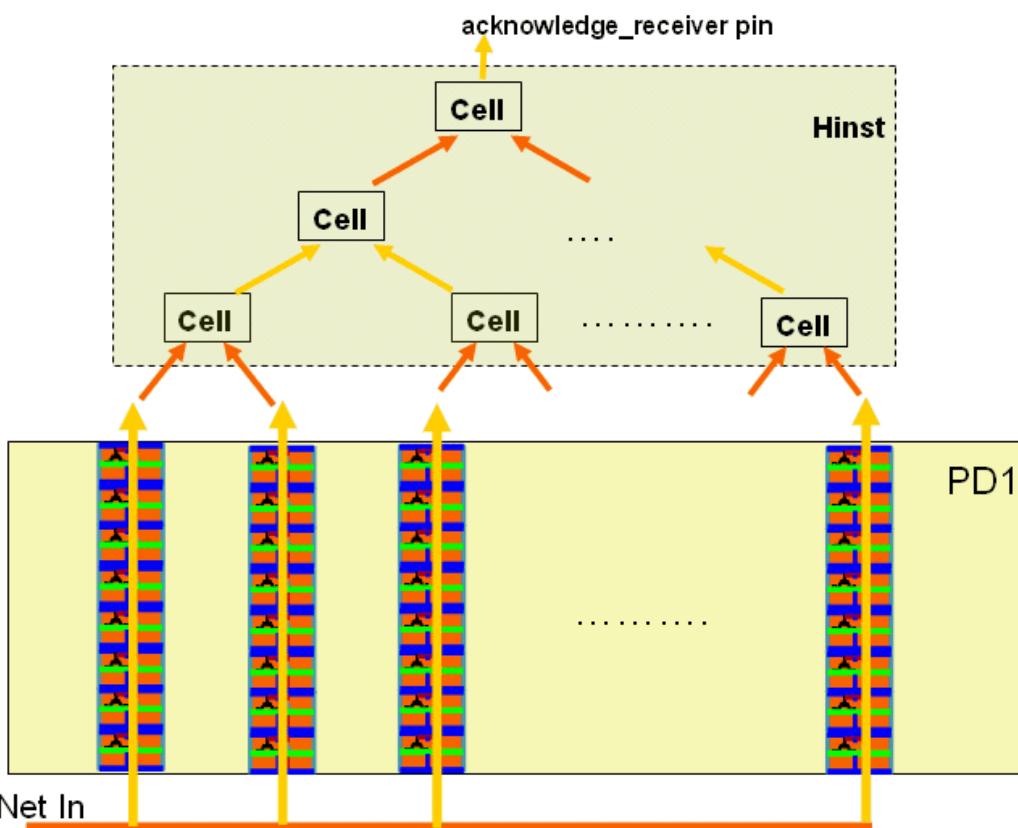
to an acknowledge receiver pin.

1. Use the following parameters to create the acknowledge tree:

- - acknowledgeTreeCell
- - acknowledgeTreeHierInstance

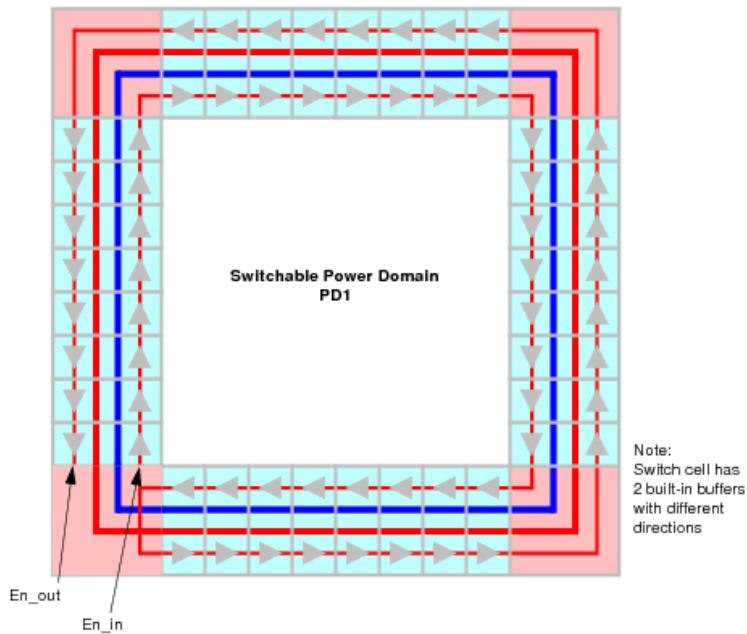
If you do not specify - acknowledgeTreeHierInstance, the tool places the cells in the top module.

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 EDI System 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}
-sideOffsetList {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}
```

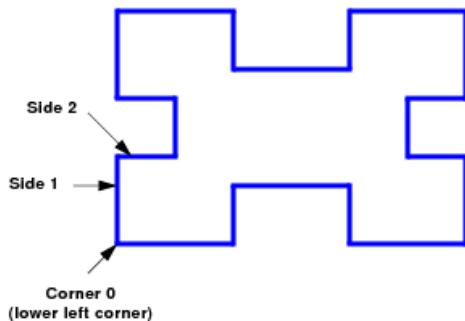
In this example, the command adds power switches in the following pattern:

```
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 EDI System software supports rectilinear power domains, such as the 20-sided power domain shown below:



The default side/corner numbering is clockwise from the starting corner (Corner 0), which is always the lower left corner of the power domain.

Corner numbering starts with 0. Side numbering starts with 1.

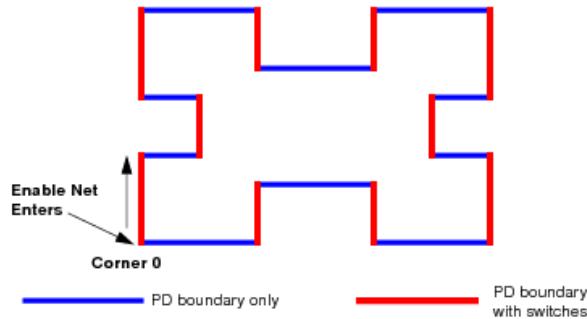
#### Specifying Sides in a Switch Ring

Use `addPowerSwitch -specifySides` to add switches around a power domain of any number of sides.

Each value provided in the `-specifiedSides` parameter corresponds to a side of the power domain. The 1 value indicates that the tool should add switches on a side, and 0 indicates that it should not. For example, for switches on all sides of a 4-sided power domain, use `- specifySides {1 1 1 1}`. By default, the tool adds switches on all sides.

The following example shows how you can place switches on every other side of the 20-sided power domain.

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

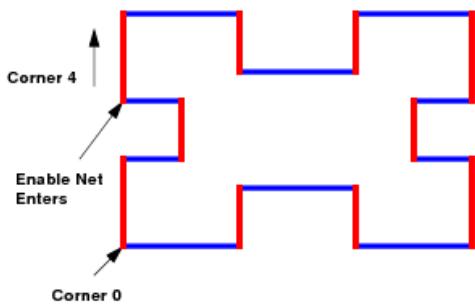


#### Starting the Enable Chain at a Different Corner

By default, the enable net enters at Corner 0. To select the corner at which you want the enable net to enter, use the `-startEnableChainAtCorner`. This example does the following:

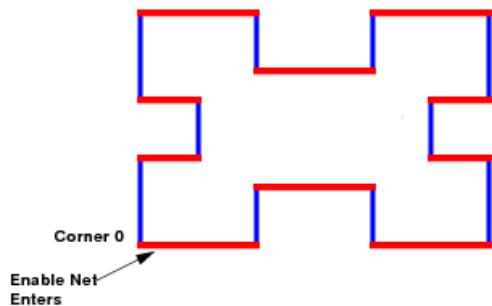
- Adds power switches on sides 1, 3, 5, 7, 9, 11, 13, 15, 17, and 19
- Starts the enable corner to corner 4.

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

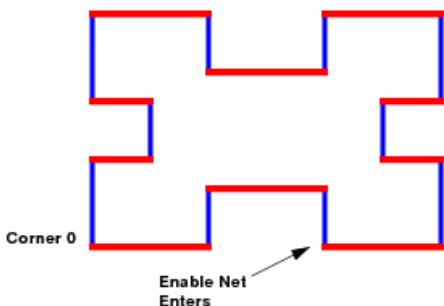


**Counter-Clockwise**

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.



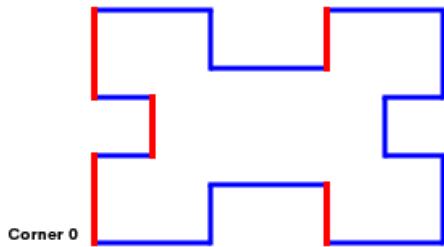
In the following example, side/corner numbering is counterclockwise, and the enable net enters at Corner 4. Note how location of the specified corner differs from the location of the corner specified without the -counterclockwise parameter.



## Left Sides

The following command adds switches only to the left sides of the power domain; Sides 1, 3, 5, 9, 17.

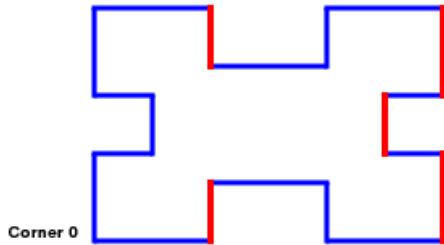
```
addPowerSwitch -ring -leftSide 1
```



#### Right Sides

The following command adds switches only to the right sides of the power domain: Sides 7, 11, 13, 15, and 19.

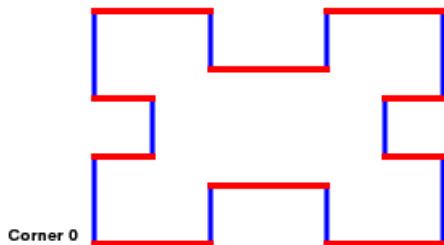
```
addPowerSwitch -ring -rightSide 1
```



#### Horizontal Sides

The following command adds switches only to the horizontal sides of the power domain: Sides 2, 4, 6, 8, 10, 12, 14, 16, and 18.

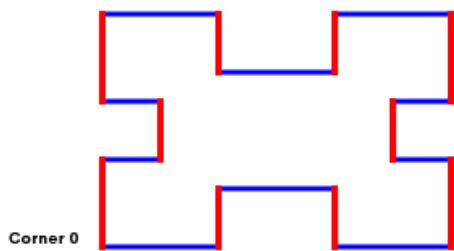
```
addPowerSwitch -ring -horizontalSide 1
```



#### Vertical Sides

The following command adds switches only to the vertical sides of the power domain: Sides 1, 3, 5, 7, 9, 11, 13, 15, and 19.

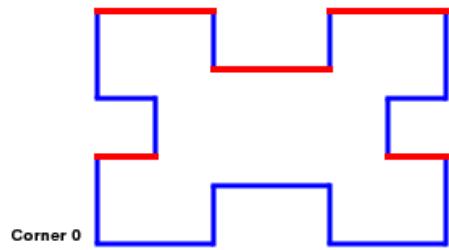
```
addPowerSwitch -ring -verticalSide 1
```



### Top Sides

The following command adds switches only to the top sides of the power domain: Sides 2, 6, 8, 10, and 14.

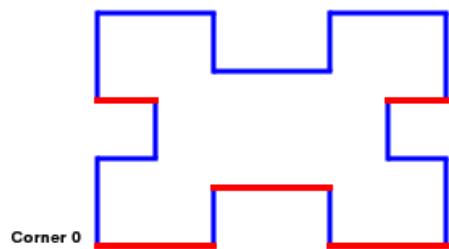
```
-addPowerSwitch -ring -topSide 1
```



### Bottom Sides

The following command adds switches only to the bottom sides of the power domain: Sides 4, 12, 16, 18, and 20.

```
-addPowerSwitch -ring -bottomSides 1
```



## Using Pitch Control and Offsets

You can control switch placement by using the following parameters:

- - globalOffset
- - bottomOffset
- - topOffset

- - leftOffset
- - rightOffset
- - horizontalOffset
- - verticalOffset
- - sideOffsetList { *value* ... }
- - startOffset
- - startOffsetBottom
- - startOffsetTop
- - startOffsetLeft
- - 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`.

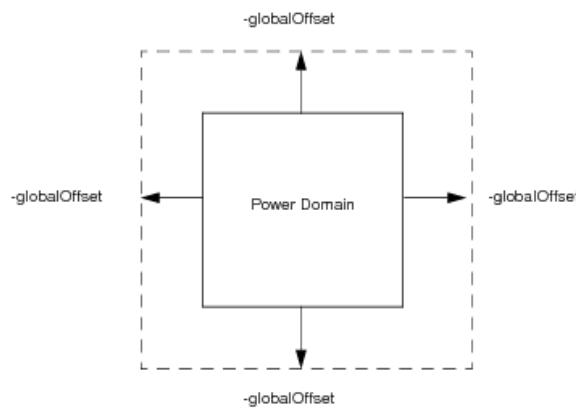
### Setting the Global Offset

Instead of placing switches against the power domain boundaries, you can place them away from the boundaries by the specified distances.

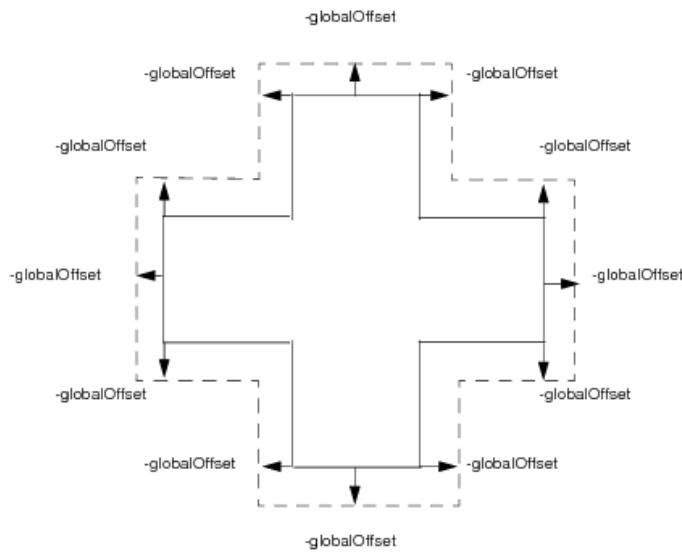
→ To specify the same offset for all sides, use the `-globalOffset` parameter.

The default offset value is 0 (no offset).

The following figure shows equal offsets for a rectangular power domain if `-forceoffset 1` is specified:



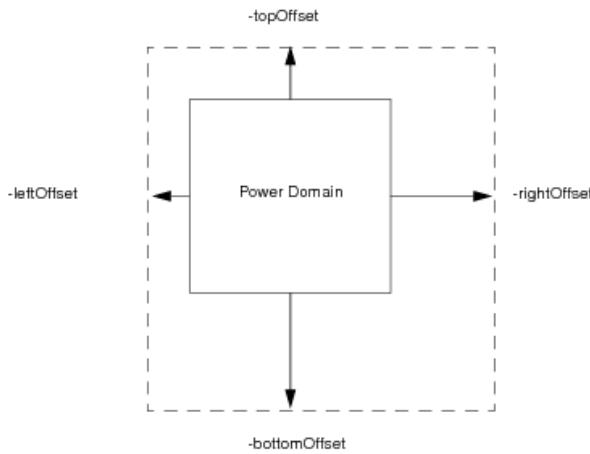
The following figure shows global offsets for a rectilinear power domain:



#### Setting Different Offsets for Different Sides

→ To specify different offsets for different sides, use the `-leftOffset`, `-rightOffset`, `-bottomOffset`, and/or `-topOffset` parameters.

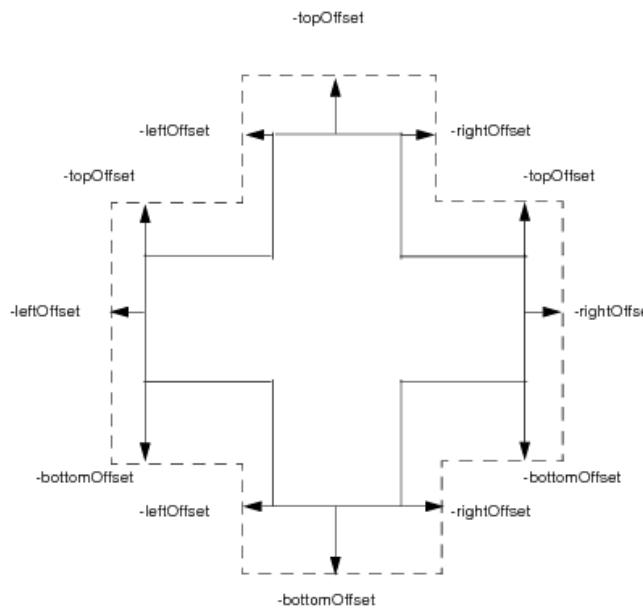
The following example shows a different offset for each side if `-forceOffset` is specified:



→ 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        |
|-----------------------------------|---------------------------------|-----------------------------------|-------------------|
| <code>-sideStartOffsetList</code> | <code>-sideEndOffsetList</code> | <code>-switchPitchSideList</code> | Specified side(s) |
| <code>-startOffsetBottom</code>   | <code>-endOffsetBottom</code>   | <code>-switchPitchBottom</code>   | Bottom side(s)    |
| <code>-startOffsetTop</code>      | <code>-endOffsetTop</code>      | <code>-switchPitchTop</code>      | 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, 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.

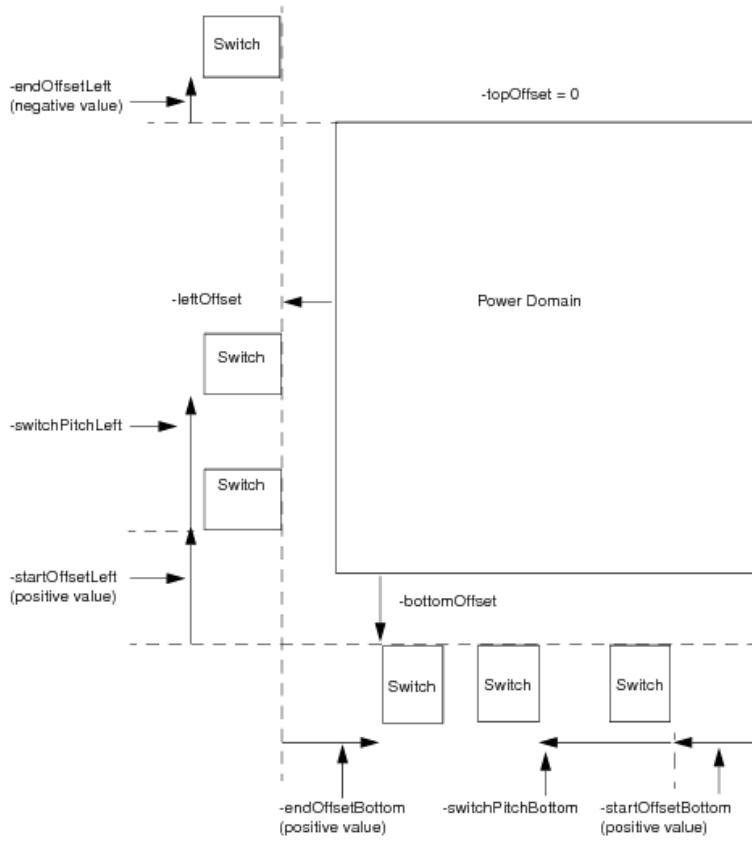
#### Example of Offsets

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  $- \text{startOffsetLeft}$  from the edge of the  $- \text{bottomOffset}$  boundary.
- The second switch on the left side is placed a distance  $- \text{switchPitchLeft}$  from the bottom edge of the first switch on the side.
- The last switch on the left side is placed a distance  $- \text{endOffsetLeft}$  *above* the  $- \text{topOffset}$  because the  $- \text{endOffsetLeft}$  value is negative. In this case, the  $- \text{topOffset}$  is 0, so the last switch is placed above the top power domain boundary.

- The first switch on the bottom side is placed a distance - startOffsetBottom from the right edge of the power domain. There is no - rightOffset or - topOffset specified.
- The second switch on the bottom side is placed a distance - switchPitchBottom from the right edge of the first switch on the side.
- The last switch on the bottom side is placed - endOffsetBottom to the *right* of the - leftOffset edge. The - endOffsetBottom parameter has a positive value.

## Power Switch Optimization

The [optPowerSwitch](#) command lets you perform power switch optimization in two ways:

- Optimize (reduce) the number of power switches in the ring and columns
- Perform ECOs to replace a filler cell with a switch or replace a switch with a bigger switch

You can use these two features in the following flow:

- Define the floorplan and power domains
- Synthesize the power grid
- Optimize power switches
- Place the design
- Run trialRoute
- Synthesize the clock tree
- Perform power switch ECO
- Perform buffer tree synthesis

## Power Switch Reduction

For power switch optimization (reduction), you can use the following options to optPowerSwitch:

- -readPowerSwitchCell
- -commit
- -effort
- -maxIRDrop
- -maxSwitchIRDrop
- -net
- -padFile
- -readInstancePower
- -reportFile

- -setDontTouchCells
- -setDontTouchInstances
- -totalPower

## Power Switch ECO

For power switch ECO, you can use the following options:

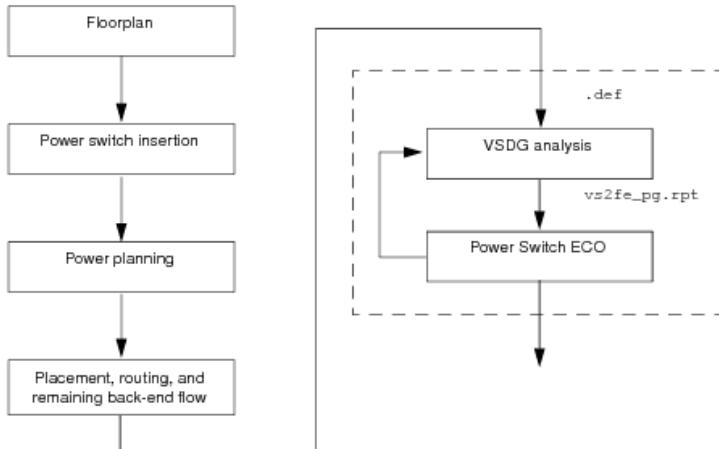
- -vsdgInFile
- -reportFile
- -fixViolations
- -reportViolationsOnly

You can now fix IR violations due to added power switches.

1. Add power switches to your design.
2. Create a complete power structure.
3. Pre-characterize power gating cells in Libgen.
4. Run Voltage Storm analysis to detect IR violations.
5. Use the new [optPowerSwitch](#) command to repair these violations.

**Note:** In power switch optimization, the inserted power switches could overlap standard cells. After running [optPowerSwitch](#), run [refinePlace](#) to move the standard cells away from the switches.

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



## Power Switch Prototyping

EDI System provides the facility for prototyping power switches. Power switch prototyping enables you to:

- Find optimal number of switches

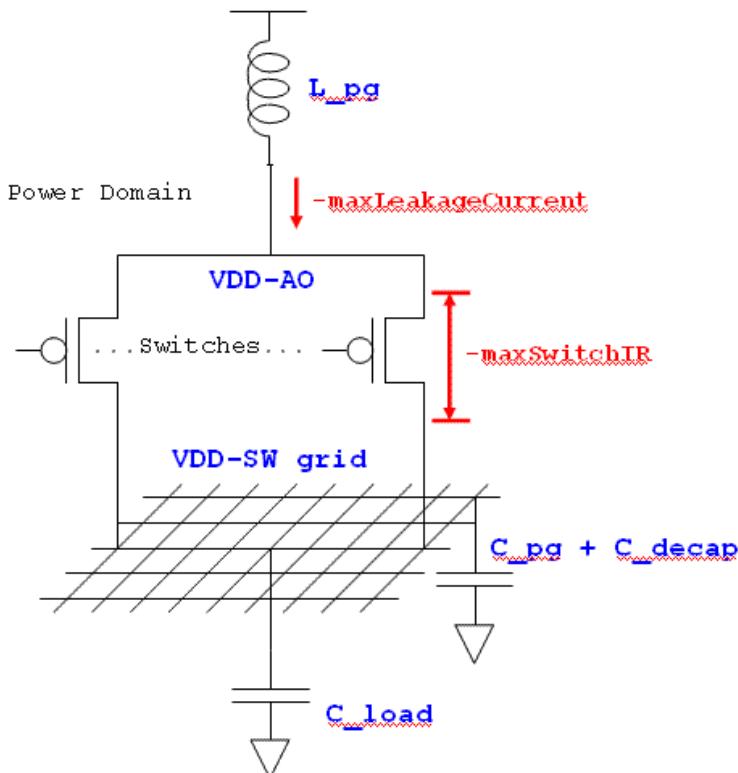
- Sweep (Enumerate) number of switches
- Given the ramp up time, find optimal number of ramp up switches
- Given number of ramp up switches, find ramp up time
- Sweep (Enumerate) number of ramp up switches
- Find number of ramp up switches constrained by maximum ramp up current
- Find best switch delay 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](#)
  - [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"](#).

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

## Loading a Design

Load a design by using the `restoreDesign` command or the `init_design` command

For more information, see

- "Importing and Exporting Designs"
- [restoreDesign](#) in the "Import and Export Commands" chapter of the *EDI System Text Command Reference*.
- [init\\_design](#) in the "Import and Export Commands" chapter of the *EDI System Text Command Reference*.

## Preparing for Placement

Before placement, run the following commands and correct problems. Some of these commands generate reports you can use as a baseline for comparisons later in the flow.

- Run `runN2N0pt` to remap and reoptimize the gate-level netlist to improve timing and area. For more information, see [runN2N0pt](#) in the "Netlist-to-Netlist Command" chapter of the *EDI System Text Command Reference*.
- Run `checkDesign` to check the integrity of the library and design data. For more information,

see [checkDesign](#) in the "Import and Export Commands" chapter of the *EDI System Text Command Reference*.

- Run `checkPlace` (or `checkDesign - place`) or use the Violation Browser to check for violations caused by preplaced cells or blocks. For more information, see [checkPlace](#) in the "Placement Commands" chapter of the *EDI System 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 *EDI System 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 *EDI System Text Command Reference*.
- Use one of the following methods to place and fix hard blocks. This step is necessary because `placeDesign` does not place blocks in default mode.
  - Run `planDesign`. For information, see [planDesign](#) in the "Floorplan Commands" chapter of the *EDI System Text Command Reference*.
  - Manually place and fix hard blocks.

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 *EDI System Menu Reference*.

A placement blockage has one of the following attributes:

|                   |                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Hard</i>       | The area cannot be used to place blocks or cells. By default, <code>createPlaceBlockage</code> creates blockages with this attribute.<br><br><b>Note:</b> In default mode, <code>placeDesign</code> does not place blocks.                                                                                                                                                                  |
| <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>    | Sets a percentage of the area that is unavailable for placement. Use the <i>Blockage Percentage</i> pull-down menu to select a percentage. For example, a partial blockage of 75 percent means that up to 25 percent of placement density is allowed in the area.<br><br><b>Note:</b> A partial blockage of 100 percent (or 0 percent placement density screen) behaves as a soft blockage. |
| <i>Macro-Only</i> | Enables <code>planDesign</code> to keep macros out of the placement blockage; however, it enables standard cells to be placed inside the box if no blockage is present.                                                                                                                                                                                                                     |

If the design has routing violations in the small channels between hard blocks, consider using the `setPlaceMode -autoBlockageInCannel true` which automatically adds soft placement blockages in these areas. Although the blockages obstruct standard cells during placement, they do not obstruct optimization operations. Using soft blockages can help improve both timing and routability.

For more information, see

- [`createPlaceBlockage`](#) in the "Floorplan Commands" chapter of the *EDI System 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 EDI System software blocks the placement of standard cells on *metal2* for a three-metal layer process. The software blocks placement on *metal2* and *metal3* for a four or more metal layer process.

You can change this behavior by using the following command before running placement:

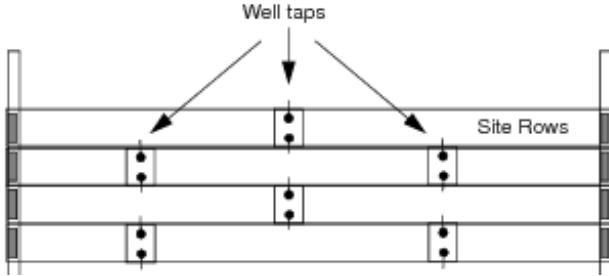
```
setPlaceMode -prerouteAs0bs
```

For more information, see [setPlaceMode](#) in the "Placement Commands" chapter of the *EDI System 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.

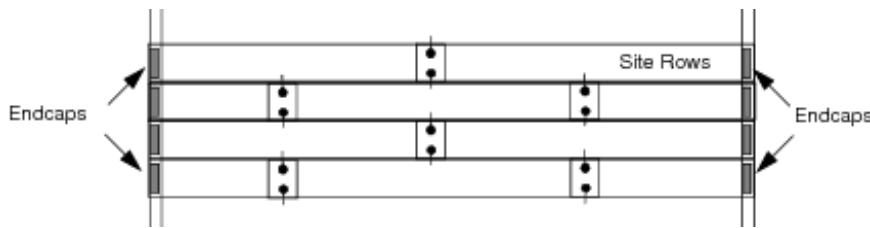
For more information see the following topics:

- [addWellTap](#) and [deleteFiller](#) in the "Placement Commands" chapter of the *EDI System 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. End-cap cells are placed in a preplaced status, so future placement commands do not move them. Add end-cap cells to the design before any other standard cells are placed, but after hard blocks are placed in the floorplan.

The following diagram shows an example of end-cap cell placement. The cells are placed at the ends of each site row.



To add end-cap cells, use the Add End Cap Instances form or the `addEndCap` command.

## Adding End Cap Cells to MSV Designs

In cases where there are different voltages in the same design, specify the power domain in which to insert the end cap cells by using the following command:

```
addEndCap -powerDomain
```

## Deleting End-Cap Cells

To remove end-cap cells, use the Delete Instances form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes end-cap cells that are completely contained within the area; it does not delete end-cap cells that cross the area boundary.

For more information see

- [addEndCap](#) and [deleteFiller](#) in the "Placement Commands" chapter of the *EDI System 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`

This parameter is valid whether `setPlaceMode - modulePlan` is true or false.

For more information on using this parameter, see "[Running Hierarchy-Aware Spare Cell Placement](#)".

**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 *EDI System 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 *EDI System 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 *EDI System 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:

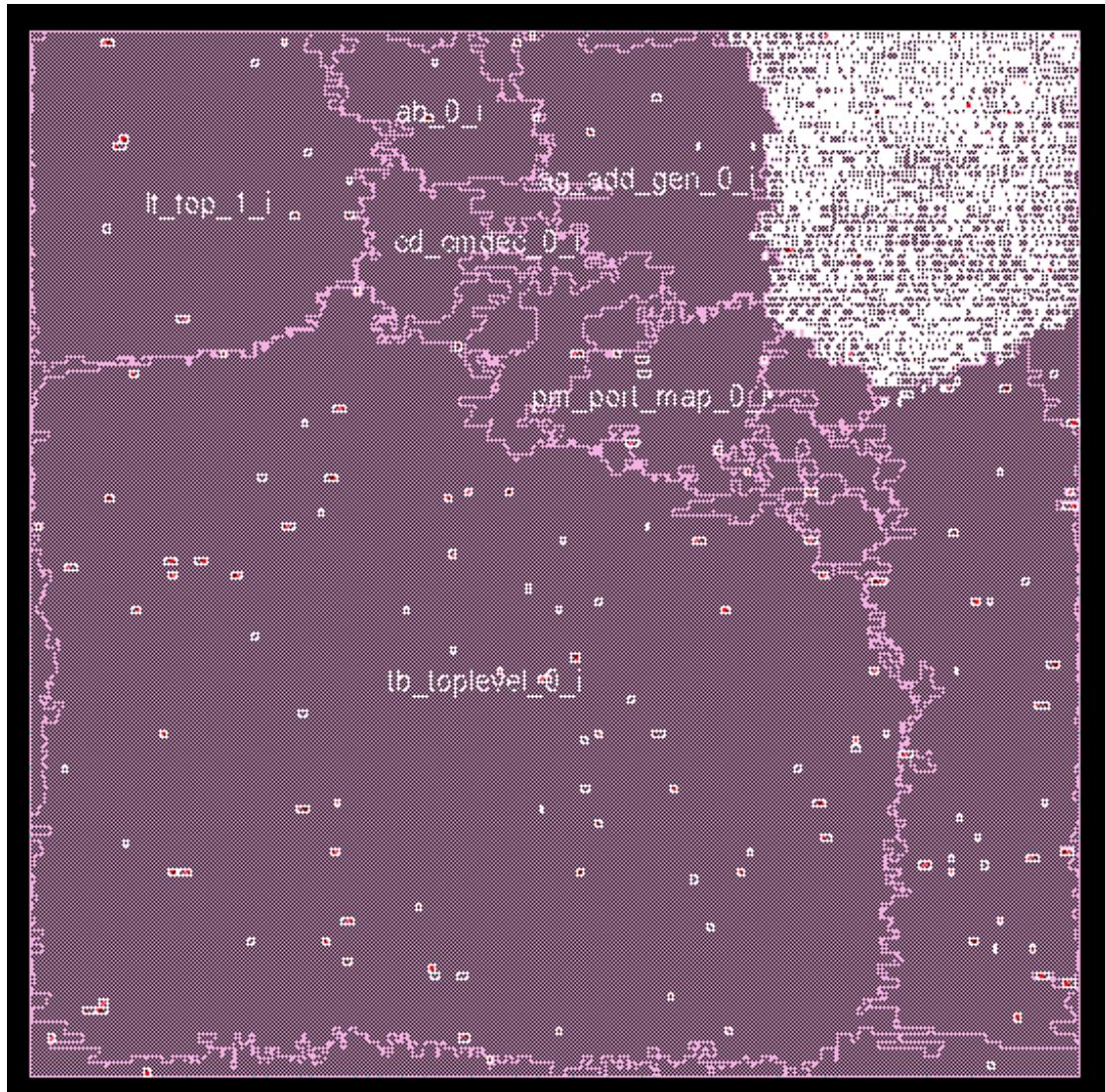
- specifySpareGate { -hinst | -inst}
- setPlaceMode -moduleAwareSpare {true | false}

The following examples and figures show how these commands affect placement.

```
specifySpareGate -inst blk1/spare_1/*
# Works also for specifySpareGate -hinst
setPlaceMode -moduleAwareSpare true
#Works also for -moduleAwareSpare true -modulePlan false
placeDesign
```

To place the spare cells evenly in the core, without binding them to the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {
    specifySpareGate -inst lt_top_0_i/spare_${x}i/*
}
placeDesign
```

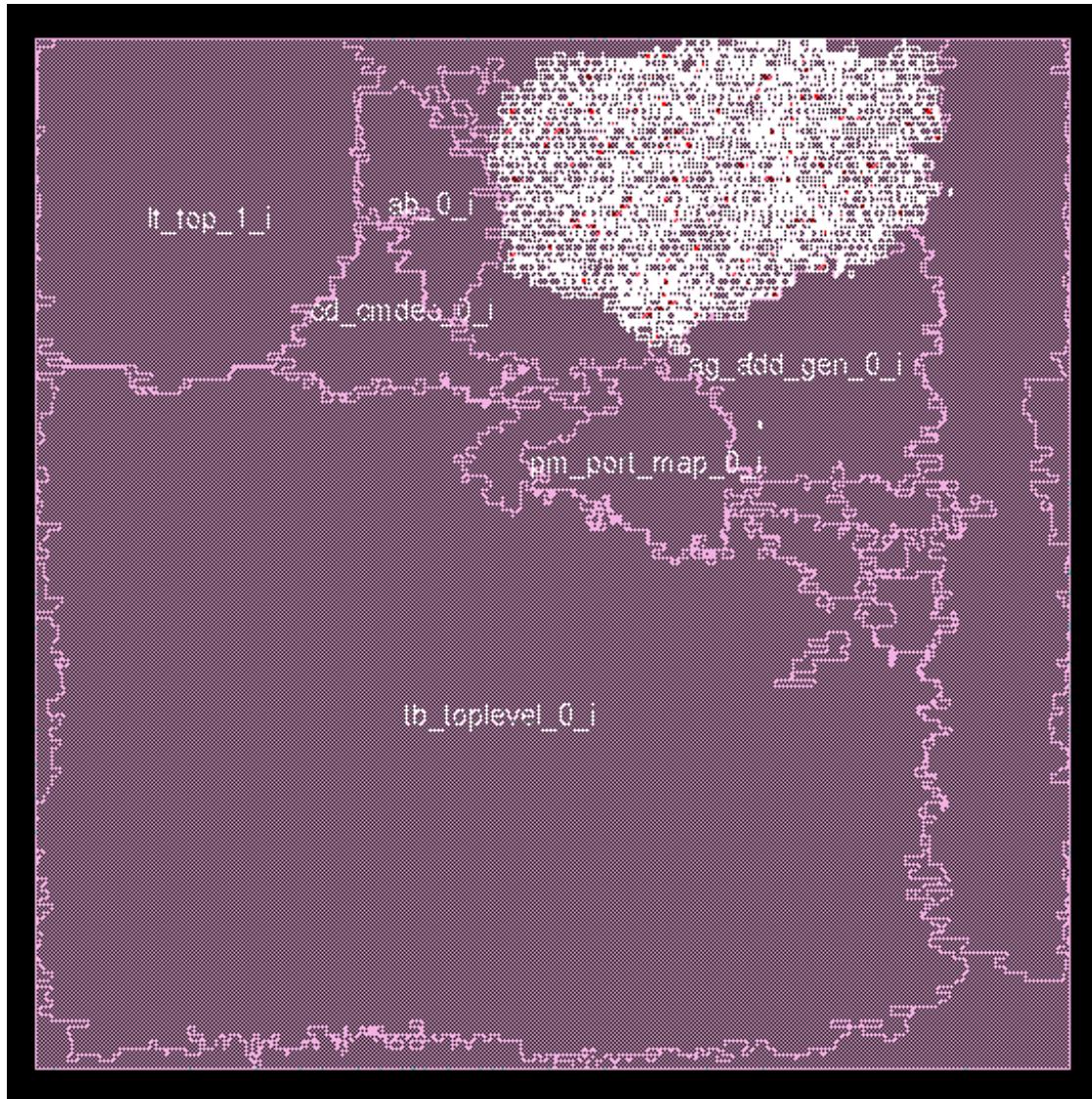


The log file, before Iteration 1, contains the following information: Identified 240 spare or floating instances, with no clusters.

To place the spare cells within the bounds of the `lt_top_0_i` hierarchy, using the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {
    specifySpareGate -inst lt_top_0_i/spare_${x}i/*
}

setPlaceMode -moduleAwareSpare true
placeDesign
```

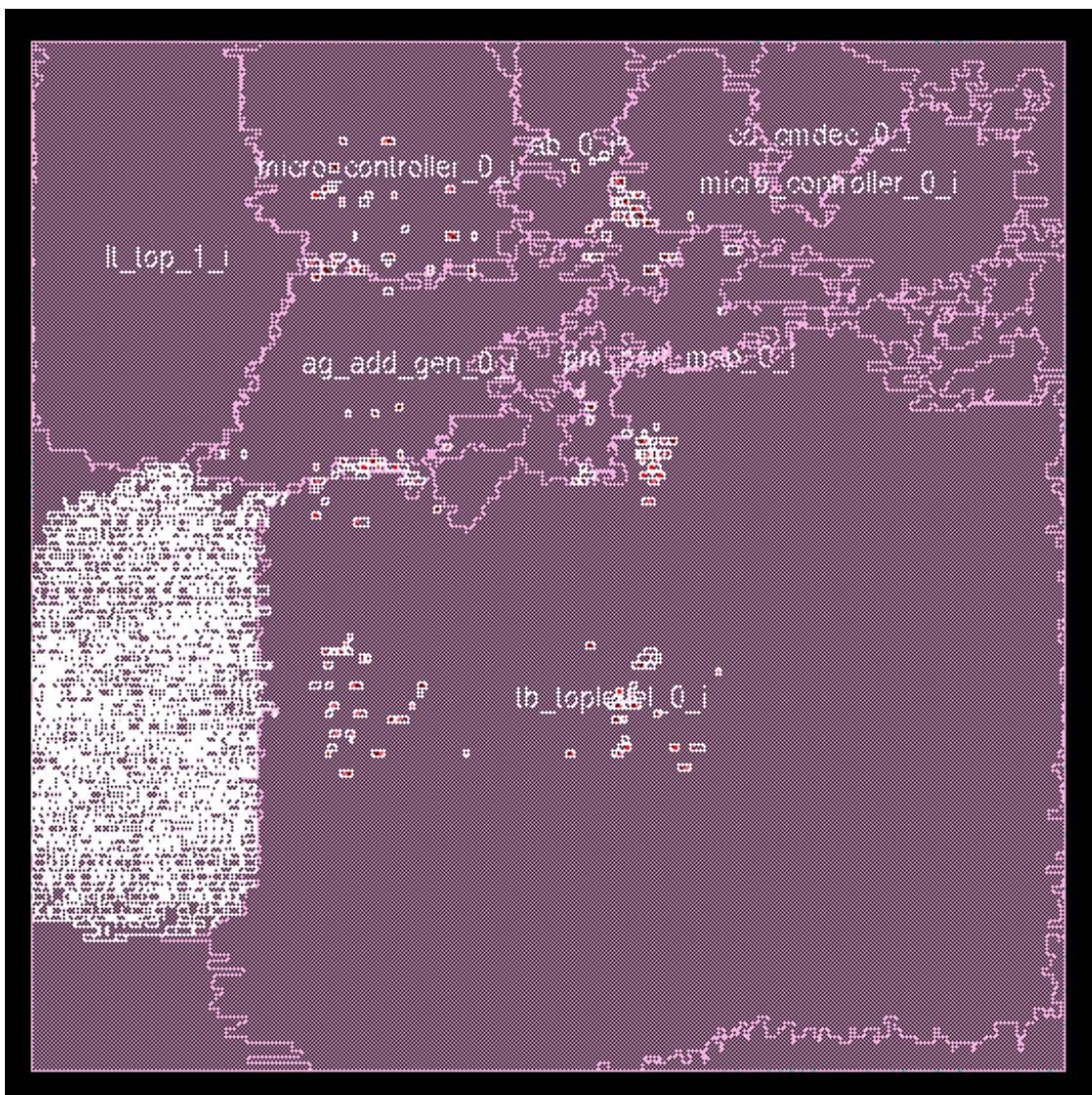


The log file, before Iteration 1, contains the following information: Identified 240 spares within logical modules.

To spread out the spare cells evenly as six clusters in the core, without binding them to the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x <6} {incr x 1} {  
    specifySpareGate -hinst lt_top_0_i/spare_${x}i/*  
}
```

`placeDesign`

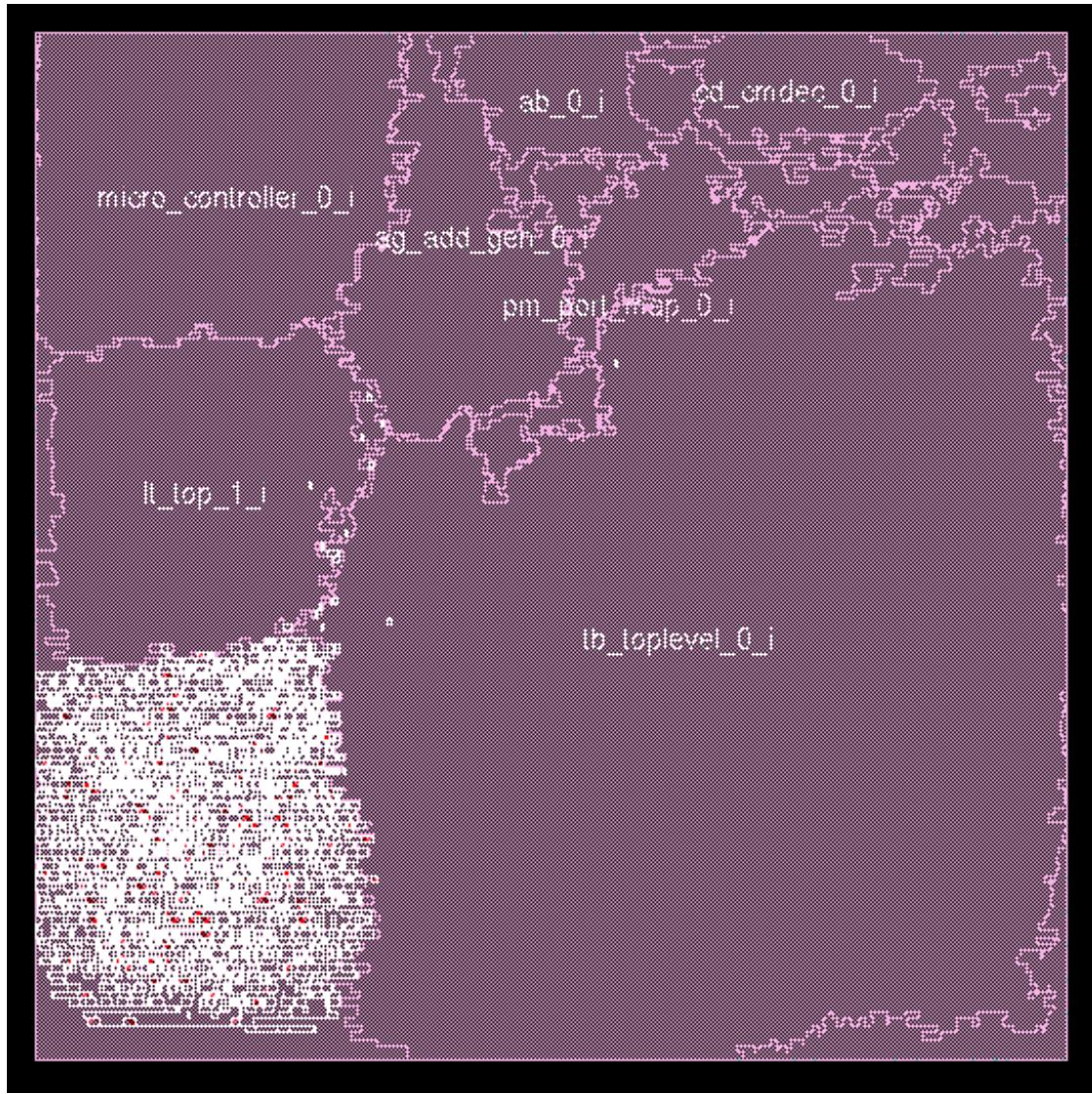


The log file, before Iteration 1, contains the following information: Identified 240 spares or floating instances, where some are grouped into 6 clusters.

To place the spare cells within the bounds of the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {
    specifySpareGate -hinst lt_top_0_i/spare_${x}i/*
}

setPlaceMode -moduleAwareSpare true
setPlaceMode -modulPlan false
placeDesign
```



The log file, before Iteration 1, contains the following information: Identified 240 spares within logical modules, of which 240 are in 6 spare-only modules.

## Adding Padding

Add padding to reserve placement space for cells or routing added after placement, for example, to make sure there is room to insert clock buffers when running Clock Tree Synthesis (CTS) on a highly localized clock. The software adds the padding on the right side of placed instances at a default metal/2 pitch dimension.

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

- If the clock in your design is concentrated in a tight area, reserve three to five percent of the targeted final utilization to add clock buffers. If your initial settings do not provide sufficient space for the buffers, add more padding and rerun placement after CTS or placement optimization.

## Adding Instance or Module Padding

Instance padding and module padding reserve space during global placement so it can be used later in the design flow, for cells added during placement legalization (`refinePlace`), Clock Tree Synthesis (CTS), or timing optimization.

**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 instances without padding, you can specify padding of two sites for each instance in the row. In this case, each instance in the row will then occupy two sites, and the row will fit only 15 instances.

1. Specify the following command:  
`specifyInstPad`

- To add extra padding on the most timing-critical instances on timing-critical paths, run the following command before placing standard cells:

```
setPlaceMode -tdInstPadding true
```

2. (Optional) Report instance padding by using the following command:  
`reportInstPad`

To delete instance padding, use the following command:

```
deleteInstPad
```

For more information, see the following commands in the "Placement Commands" chapter of the *EDI System User Guide*:

- [specifyInstPad](#)
- [reportInstPad](#)
- [deleteInstPad](#)

## Adding Module Padding

To reduce localized congestion, add module padding.

1. Specify the following command:

```
setPlaceMode -modulePadding module factor
```

This command adds padding within hierarchical modules by spreading out the standard cell instances within the modules. The padding is specified in terms of a factor that is applied to the instance area of all the cells within the module. For example, a factor of 1.2 increases the area by 20 percent.

This parameter is disabled when `-modulePlan false` is specified.

**Note:** The software ignores factors that are less than 1.0.

2. Run standard cell placement.

For more information, see [setPlaceMode](#) in the "Placement Commands" chapter of the *EDI System User Guide*.

## Adding Cell Padding

Cell padding adds hard constraints to placement. The constraints are honored by cell legalization, CTS, and timing optimization, unless the padding is reset after placement so those operations can use the reserved space. You can use cell padding to reserve space for routing.

- Specify the following command:

```
specifyCellPad
```

This command adds padding on the right side of library cells during placement. (Padding

location is dependent on the orientation of the cell. For example, if the library cell is flipped when it is instantiated, the padding is on the left side.) The padding is specified in terms of a factor that is applied to the placement SITE. For example, if you specify a factor of 2, the software ensures that there is additional clearance of two times the placement SITE 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 *EDI System User Guide*:

- [specifyCellPad](#)
- [deleteAllCellPad](#)

## Placing Standard Cells

Place standard cells with the `placeDesign` command. By default, the command runs preplacement optimization and standard cell placement. If you specified SDC timing constraints, it runs in timing-driven mode by default. If you specified scan information, it performs scan tracing and reordering by default.

- i If `placeDesign` does not place the standard cells, for example if all instances are fixed or if there is no placeable area, then `placeDesign` also skips I/O pin assignment.

This command was designed as a super command; that is, with the following exceptions, you can use it to place standard cells without specifying any placement options for your initial placement.

- If the design has more than 1,000 floorplan constraints or other types of complex floorplan

constraints, run the following command before placing standard cells:

```
setPlaceMode -modulePlan false
```

- If the design has clock-gating cells, run the following commands before placing standard cells:  
`specifyClockTree -clkfile fileName`  
`setPlaceMode -clkGateAware true`
- To reduce switching power on power-critical nets, consider running the following command before placing standard cells:  
`setPlaceMode -powerDriven true`

Tune the initial placement by trying the following techniques:

- If the design is congested, try the following command, then rerun placement:  
`setPlaceMode -congEffort high`
- If the design has local congestion, try the following command, then rerun placement:  
`setPlaceMode -modulePadding module factor`
- If the utilization increased by more than five percent after pre-CTS optimization, compared to what it was after `placeDesign`, try the following command:  
`placeDesign -inPlaceOpt`

The run time for this command is longer than the run time without the `-inPlaceOpt` parameter.

## Related Topics

- [placeDesign](#) and [setPlaceMode](#) in the "Place Commands" chapter of the *EDI System Text Command Reference*
- [specifyClockTree](#) in the "Clock Tree Synthesis Commands" chapter of the *EDI System Text Command Reference*

## Running Placement in Multi-CPU Mode

The `placeDesign` 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" Encounter 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.
- It is not supported when the `-modulePlan` parameter is set to `false`.
- 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 EDI System 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 `placeDesign` section of the log file.

```
(cpu for global=1:25:08) real=0:50:32***  
Placement multithread real runtime: 0:50:32 with 2 threads.  
Core Placement runtime cpu: 1:14:24 real: 0:41:42  
Starting refinePlace ...
```

The number to look for in the log is `Placement multithread real runtime`. In the preceding

example, the Placement multithread real runtime is 0:50:32.

### Calculating Multi-Thread Speed-Up

The amount of time used for global placement is calculated by using the following formula:

Global Placement = Core Placement + Timing Analysis + Congestion Analysis

In some designs, when timing and congestion analysis consume a high percentage of run time, the speed-up factor from multi-threading is not significant.

In the preceding example, if only one thread were used, the log would have reported the following:

```
(cpu for global=1:13:53) real=1:15:09***  
  
Core Placement runtime cpu: 1:04:53 real: 1:05:59  
  
Starting refinePlace ...
```

Comparing the times from the two log segments gives the following calculations:

- real time (1 thread) = 1:15:09 = 4509 seconds
- real time (2 threads) = 0:50:32 = 3032 seconds

$4509 / 3032 = 1.49$

**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 *EDI System Text Command Reference* .

## Checking Placement

Use the following methods to check placement:

- Amoeba view

For more information, see [The Main Window](#) chapter in the *EDI System Menu Reference*.

- checkPlace command

- For more information, see [checkPlace](#) in the "Placement Commands" chapter of the *EDI System Text Command Reference*.

- Placement density map

For information, see the following references:

- "Placement Commands" chapter of the *EDI System Text Command Reference*

- [getDensityMapMode](#)

- [reportDensityMap](#)

- [setDensityMapMode](#)

- "Placement Menu" chapter of the *EDI System Menu Reference*

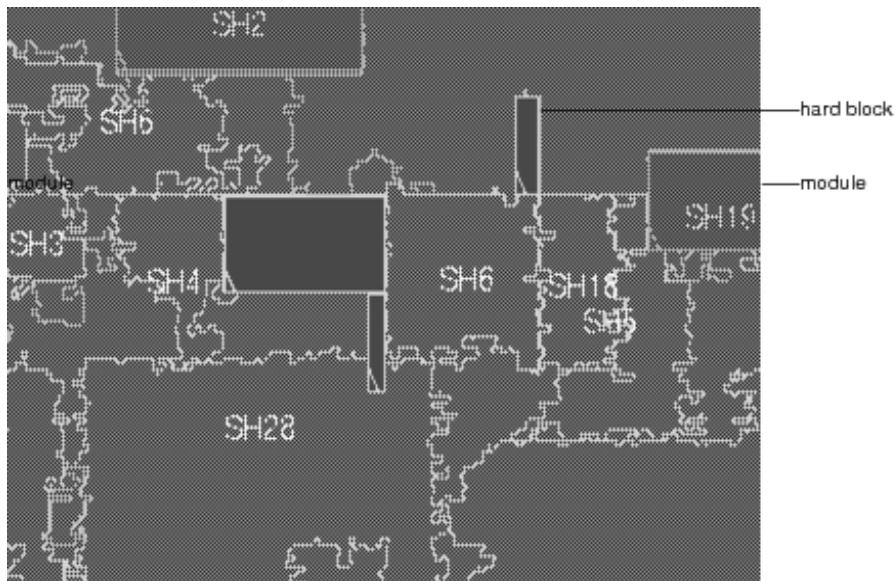
- [Display Density Map](#)

- 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 EDI System window.

For more information, see [The Main Window](#) chapter in the *EDI System 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 EDI System window.
- Clock the All Colors button to open the Color Preferences form, then select the *View Only* tab, and select *Density Map* , in the *Multi-Color Layers* section.

## Adding Filler Cells

The software uses filler cells to fill the gaps between standard cell instances. Filler cells also provide decoupling capacitance to complete the power connections in the standard cell rows and extend N-well and P-well regions. The reason to add them as the last placement step is that you cannot run in-place optimization after they are added. After routing, the software checks for DRC violations created by the added filler cells.

To add filler cells, use the [`addFiller`](#) command.

- Provide a list of filler cells so that at least one filler cell can be used to fill the space without causing DRC violations.

Add Filler recognizes whether a filler cell has implant layer geometries and attempts to add fillers that honor the implant layers' width and spacing rules. By judicious selection of filler cells, the software can correct implant layers' minimum spacing errors by putting in same voltage threshold implant layer fillers in spaces between two same implant layer cells. Add Filler also avoids creating implant layer minimum width errors by abutting fillers of same implant layer as the adjacent cells, thus extending the implant layer width.

- i** 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.

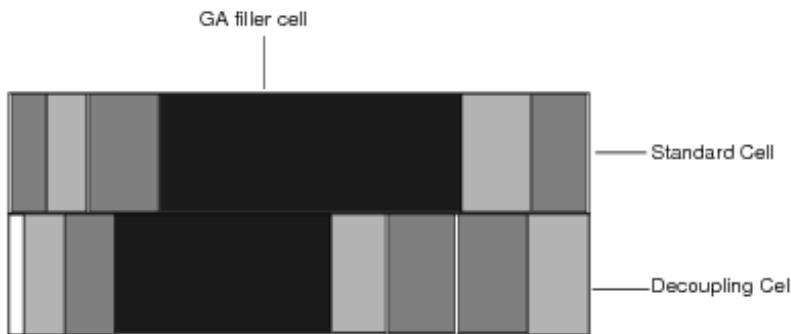
## Placing Gate Array Style Filler Cells for Post-Mask ECO

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.

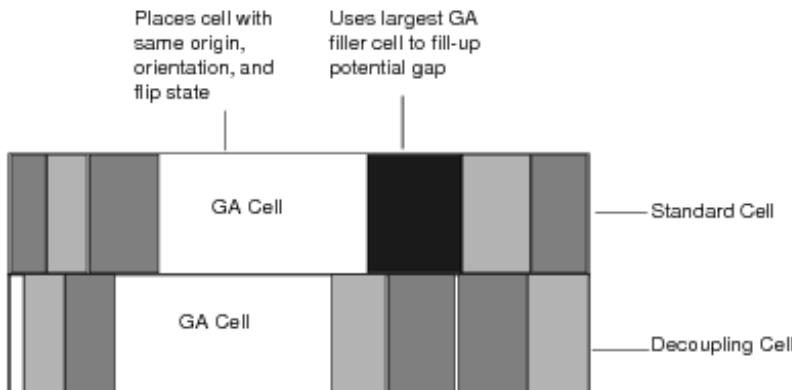
- Specify the following command:

```
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 *EDI System 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 EDI System 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 EDI System 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 this command.

- a. To undo the specification, use the following command:

```
unspecifyJtag
```

2. To place the instances or cells, use the following command:

```
placeJtag
```

3. (optional) To generate a report of the JTAG placement, use the following command:

```
reportJtagInst
```

To undo JTAG placement, use the following command:

```
unplaceJTAG
```

- If you do not want to place regular instances in the JTAG outer core area after running JTAG placement, specify a placement blockage prior to running placement.

## Related Topics

For more information, see the following commands in the "Placement Commands" chapter of the *EDI System User Guide*:

- [specifyJtag](#)
- [unspecifyJtag](#)
- [placeJtag](#)
- [reportJtagInst](#)
- [unplaceJtag](#)
- [traceJtag](#)

## Optimizing and Reordering Scan Chains

The `placeDesign` command reorders scan chains by default, unless it is in prototyping mode. If you decide not to reorder scan cells with `placeDesign`, use the information provided in this section to reorder scan chains.

## Related Topics

To see this step in the design flow, see "Place the Design and Run Pre-CTS Optimization" in the *EDI System 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, uniquify the incoming netlist and make sure that it does not contain Verilog assignment statements involving scan nets. A scan net is a net that resides along the scan datapath--that is, a net that connects the scan flip-flops in a scan chain.

If the netlist is unqualified, but contains Verilog assignment statements involving scan nets, use the following command to insert a temporary buffer into the netlist to enable reordering of these nets:

```
setDoAssign on -buffer bufferName
```

When the EDI System software reads the netlist, it outputs the following messages:

```
Reading netlist ...
```

```
Reading verilog netlist ".fileName"
```

```
Inserting temporary buffers to remove assignment statements.
```

If buffers were added by setDoAssign, these buffers remain in the final netlist and replace the Verilog assignment statements.

## Reordering Scan Chains

Use one of the following approaches to scan chain reordering:

- Native scan reordering

Use this approach in the following conditions:

- Single-clock domain, single-edge chains
- Multiple clock domain chain segments separated by data lockup elements

- Shared functional output signal chains
- ScanDEF-based reordering

Use this approach in the following conditions:

- All simple scan chain architectures (handled by the native approach)
- Implied domain transition scan chains (without data lockup elements)
- Scan chains with ordered segments
- Scan chains generated by LogicVision software

After reordering scan chains, save a netlist of the design using one of the following methods:

- [Save - Netlist](#) form (*Design - Save - Netlist*)
- [saveNetlist](#) command

### **Native Scan Reordering Approach**

Use the native approach to scan chain reordering when you do not have a scanDEF file.

This approach requires that you use the `specifyScanChain` command to identify the START and STOP signals of the top-level chains, or chain segments, in the netlist. Using this information, the software identifies the scan flip-flops along the scan chain when running the `scanTrace` command to analyze the scan flip-flop connections. You can also auto-detect data lockup latch elements using the `scanTrace -lockup` command.

If the scan cells are not listed in the timing library, you must specify them before tracing the scan chains. You can identify scan cells with the [specifyScanCell](#) command.

After `scanTrace` has identified the elements along the chain, complete the following steps:

1. (Optional) Ignore the scan connections:

```
setPlaceMode -ignoreScan true
```

2. (Optional) Set scan reorder options:

```
setScanReorderMode -skipMode [skipNone | skipBuffer | skipTwoPinCell]
```

3. Run placement:

```
placeDesign
```

The recommended flow for scan chains that have data lockup latches is as follows:

1. Specify a scan chain in the design:

```
specifyScanChain
```

2. Trace the scan chain connection with the automatic detection lockup latch elements:

```
scanTrace -lockup [-verbose]
```

3. (Optional) Ignore the scan connections:

```
setPlaceMode -ignoreScan true
```

4. (Optional) Set scan reorder options:

```
setScanReorderMode [-skipNone | -skipBuffer | -skipTwoPinCell]
```

5. Run placement:

```
placeDesign
```

The recommended flow for scan chains that have data lockup flip-flops is as follows

1. Specify a scan chain in the design:

```
specifyScanChain ...
```

2. Specify a cell or instance as a lockup flip-flop element:

```
specifyLockupElement ...
```

3. (Optional) Ignore the scan connections:

```
setPlaceMode -ignoreScan true
```

4. (Optional) Set scan reorder options:

```
setScanReorderMode -skipMode [skipNone | skipBuffer | skipTwoPinCell]
```

5. Run placement:

```
placeDesign
```

**Note:** The `scanReorder` command automatically calls `scanTrace` internally if you have not previously

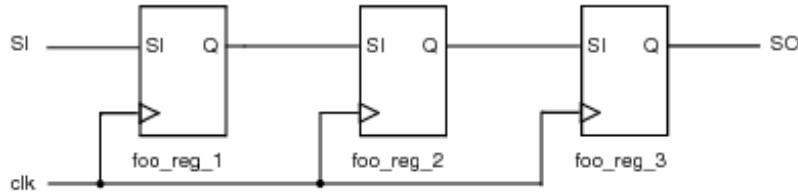
run `scanTrace`. By default, this internal `scanTrace` run specifies that the tracing will not detect lockup elements (`-noLockup`); therefore, if you have lockup latches, Cadence recommends using the `scanTrace -lockup` command before `scanReorder`, or specify `LockupElement` prior to running `scanReorder`.

### **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.

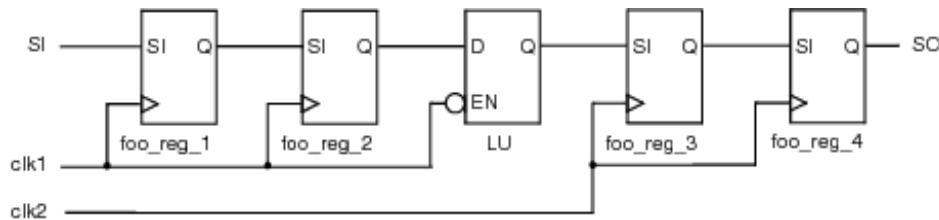


- `foo_reg_1`, `foo_reg_2`, and `foo_reg_2` scan flip-flops are triggered by `c1k1` (positive edge).

```
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 `c1k1` (positive edge).
- `foo_reg_3` and `foo_reg_4` scan flip-flops are triggered by `c1k2` (positive edge).
- LU represents a data lockup element of type latch.

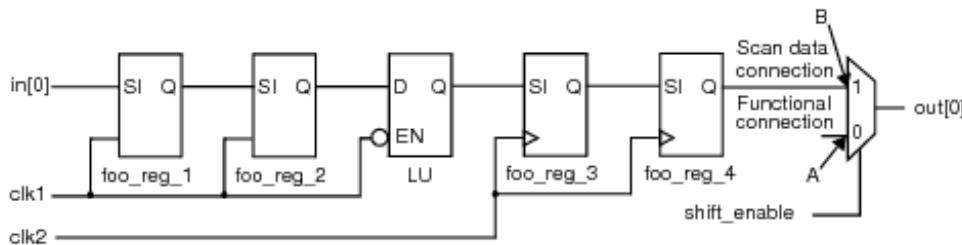
```
specifyScanChain chain1 -start SI -stop SO
scanTrace -lockup [-verbose]
```

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]
```

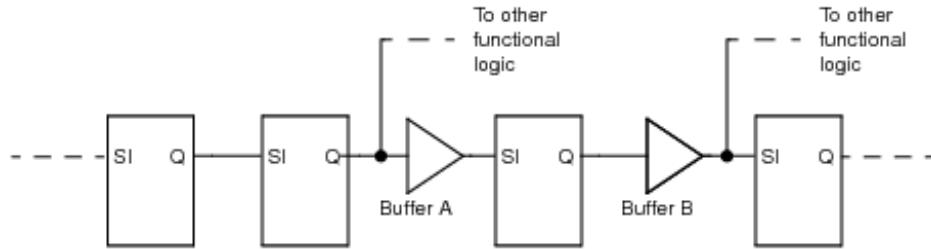
The following command sequence does not perform the forward trace from the last flip-flop through the MUX instance; scanTrace will not succeed:

```
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. (Optional) Ignore the scan connections:

```
setPlaceMode -ignoreScan true
```

3. Run placement:

```
placeDesign
```

### Using the `scanReorder` Command

When running the `scanReorder` command, the EDI System 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 EDI System 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, EDI System 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 EDI System, 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.

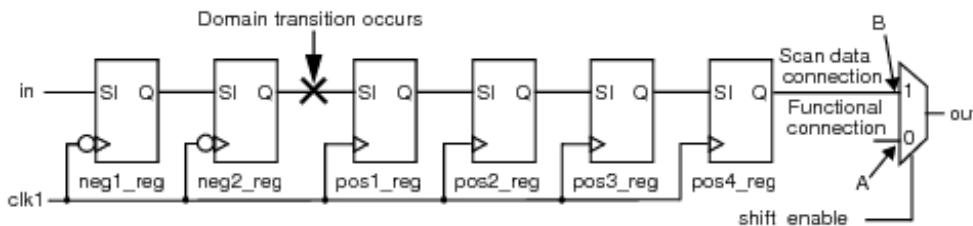
**Note:** You can use the following RTL Compiler command: `write_scandef > fileName`

### ***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:

- clk1 (negative edge) consisting of elements neg1\_reg and neg2\_reg
- clk1 (positive edge) consisting of elements pos1\_reg, pos2\_reg, pos3\_reg, and pos4\_reg

Because the domain transition is done implicitly (without a data lockup element), the scan chain must be segmented to be properly reordered. In the scanDEF format, the top-level chain becomes two scanDEF chains, segmented by clock domain and clock edge; the pos1\_reg scan flip-flop is sacrificed to anchor the domain transition. This register becomes an internal end and internal being point of scan DEF chains (chain1 and chain2 respectively):

```
SCANCHAINS 2 ;
- chain1
+ START pin in
+ FLOATING
    neg1_reg ( IN SI ) ( OUT Q )
    neg2_reg ( IN SI ) ( OUT Q )
+ STOP pos1_reg SI
;
- chain2
+ START pos1_reg Q
+ FLOATING
    pos2_reg ( IN SI ) ( OUT Q )
    pos3_reg ( IN SI ) ( OUT Q )
+ STOP pos4_reg SI
;
```

END SCANCHAINS

**Note:** The shared functional output signal (`out`) is not the `STOP` signal of the second scan chain segment. Instead, the scan chain is terminated to the `IN` pin of the last scan flop in the positive-edge triggered segment (BuildGates/PKS), or terminated to the data input pin of the MUX (other third-party tools).

- Scan chains with ORDERED segments

An order segment is a set of connected elements that can be reconnected along the scan chain based on its placement. Reconnection to the fixed segment occurs using the `IN` pin of the first element and the `OUT` pin of the last element of the ordered segment. The connections of the other elements in the ordered segment are presumed connected and remain as intact connections. When an ORDERED segment is reconnected in the scan chain, the location of the ORDERED segment appears as a comment in the FLOATING section and again in the ORDERED section in order to correlate the segment to its location in the FLOATING section. The notation is as follows:

```
# ORDERED segment integer ;
```

The integer corresponds to as many ORDERED segments as defined in the original scan chain. For example, a scanDEF chain with one ORDERED segment is as follows:

```
SCANCHAINS 1 ;
- chain0
  + START PIN scan_in
  + FLOATING
    out_reg_0 ( IN SI ) ( OUT Q )
    out_reg_1 ( IN SI ) ( OUT Q )
    out_reg_2 ( IN SI ) ( OUT Q )
    out_reg_3 ( IN SI ) ( OUT Q )
  + ORDERED
    out_reg_4 ( IN SI ) ( OUT Q )
    u_buf ( IN A ) ( OUT Y )
  + STOP PIN scan_out ;
END SCANCHAINS
```

After reordering the output, the scanDEF file is as follows:

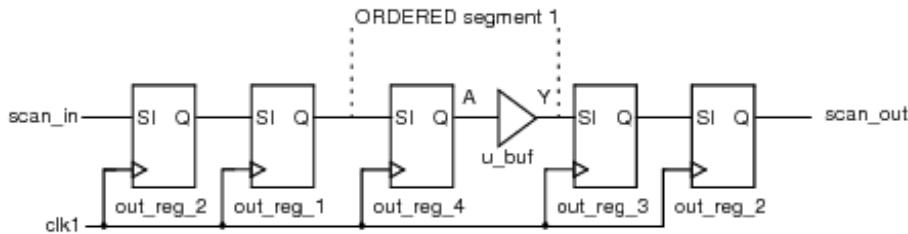
```
SCANCHAINS 1 ;
- chain0
  + START PIN scan_in
```

```

+ FLOATING
    out_reg_2 ( IN SI ) ( OUT Q )
    out_reg_1 ( IN SI ) ( OUT Q )
# ORDERED segment 1
    out_reg_3 ( IN SI ) ( OUT Q )
    out_reg_0 ( IN SI ) ( OUT Q )
+ ORDERED
# ORDERED segment 1
    out_reg_4 ( IN SI ) ( OUT Q )
    u_buf ( IN A ) ( OUT Y )
+ STOP PIN scan_out ;
END SCANCHAINS

```

Therefore, the connectivity of the elements along the reordered scan chain is as follows:



### Saving Scan Files

After scan reorder is run, save a DEF file using the following command:

```
defOutBySection -noNets -noComps -scanChains
```

With this command, you can view the new order of elements along the scan chain. However, you should use the scanDEF output file for viewing purposes only, not a subsequent reordering pass.

To save scan files, use the Save Scan File form or the [defOut](#) command.

### Loading Scan Files

To load scan files in DEF format, use the Load Scan File form. For DEF, use the [defIn](#) command.

---

# Synthesizing Clock Trees

---

- [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 Post-CTS Clock Tree Optimization](#)
  - [Using the ckECO Command for Post-CTS 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 Post-CTS 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](#)
  - [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](#)
- [Clock Concurrent Optimization](#)
  - [Overview](#)
  - [The CCOpt Flow in the EDI System](#)
  - [The Scripted Integration Mode](#)
    - [The Scripted CCOpt Flow](#)
  - [Native Integration Mode](#)
    - [The Native CCOpt Flow](#)
  - [CCOpt Properties and Configuration](#)
    - [Setting CCOpt Properties](#)
    - [Getting Properties from Tcl](#)
    - [Key-Value Pairs](#)
    - [Defaults for Key Value Pairs](#)
  - [CCOpt Clock Tree Specification](#)
    - [Clock Trees and Skew Groups](#)
    - [Automatic Extraction of Clock Trees](#)
    - [Defining Clock Trees](#)
    - [Defining Skew Groups](#)
    - [Skew Group Rank and Active Sinks](#)
    - [How Automatic Clock Tree Extraction Works](#)
    - [Reporting on Clock Trees](#)
    - [Reporting on Skew Groups](#)
  - [CCOpt Clock Tree Debugger](#)

## Before You Begin

Before you run CTS on your design, make sure the following files are available:

- Clock tree specification file
- Verilog netlist
- GDSII or LEF physical library
- Proper RC model from LEF, Encounter™ technology file, or Encounter® Digital Implementation System (EDI System) capacitance table  
For information on RC extraction in EDI System, see RC Extraction of the *EDI System 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 EDI System 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.

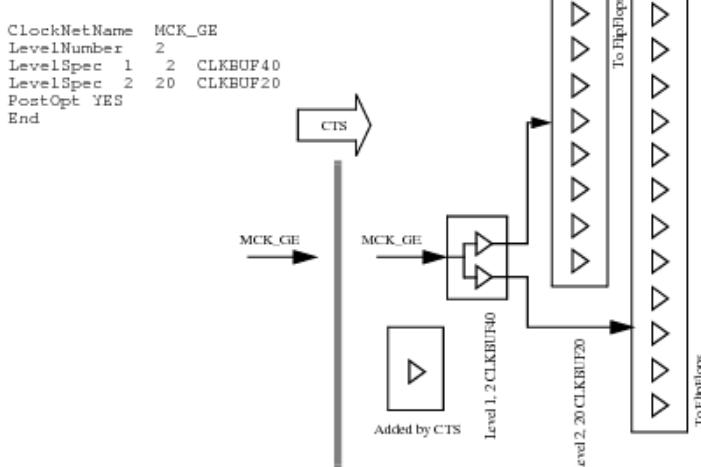
- Manual CTS mode allows you to control the number of levels and the number of buffers, and specify the types of buffers at each level.
- In automatic CTS mode, CTS automatically determines the number of levels and buffers based on the timing constraints in the clock tree specification file, such as 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:



## 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.

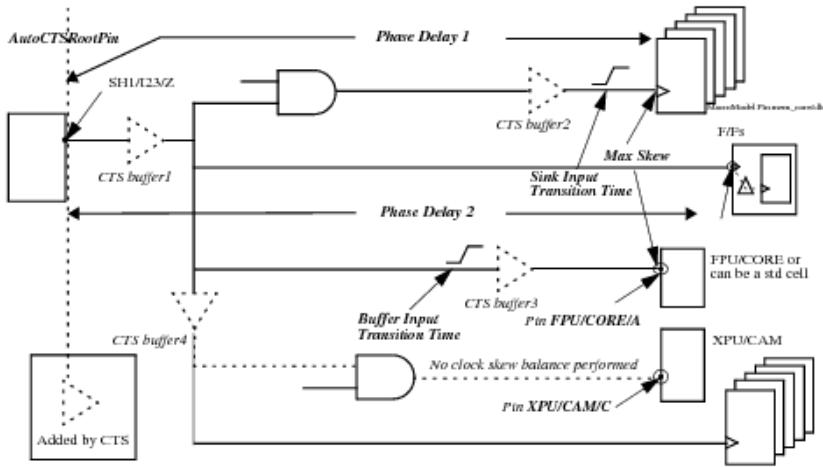
**Note:** Cadence recommends using the [ckSynthesis](#) -check command to check the gated clock tree of your design before running automatic gated CTS mode. After running the command, review the trace report file,*topCellName.cts\_trace*. If tracing fails, the -forceReconvergent parameter of the [ckSynthesis](#) 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:

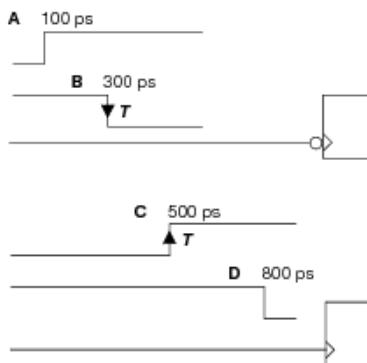
- FF1 rise: 3.0 ns; fall: 3.6 ns
- FF2 rise: 3.4 ns; fall: 4.0 ns

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 EDI System console, type `setCTSMode -routeGuide true`.
2. In the clock tree specification file, include `RouteClkNet YES`.

3. In the EDI System console, type [ckSynthesis](#).
4. Check your run directory for the clock tree timing report (*top\_level\_cell*.cts rpt).

#### Method 2

1. In the clock tree specification file, include RouteClkNet NO.
2. In the EDI System console, type [ckSynthesis](#).
3. In the EDI System console, type routeClockNetWithGuide [-clk *clock\_root\_pin\_name*]
4. Check your run directory for the clock tree timing report (*top\_level\_cell*.cts rpt).

## 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 macro model, then the tool displays an error message on specifying the view name.

### Example1

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
```

If you specify macro model for one pin without any view and the following statement is a macro model statement for the same pin but with a specific view name. For example, view2. Assuming there are three active views: view1, view2, view3, then view2 will get the delay value of the second statement (because the latest overrides the previous) and view1 (assuming view1 is the default setup view) will get the value from first statement 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
```

### Example2

Consider the following statements:

```
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
```

The tool interprets it as:

```
MacroModel pin inst1/CK 0.4ps 0.4ps 0.4ps 0.4ps 0pf view1  
MacroModel pin inst1/CK 0.4ps 0.4ps 0.4ps 0.4ps 0pf view2  
MacroModel pin inst1/CK 0.4ps 0.4ps 0.4ps 0.4ps 0pf view3
```

## Dynamic Macro Model

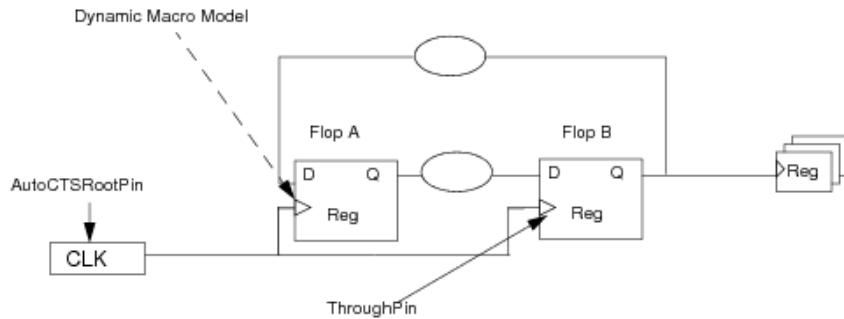
A dynamic macro model is used to minimize the skew between the reference pin and the target pin during CTS. The reference pin is a clock instance pin along a clock path. The target pin must be a leaf pin.

```
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.

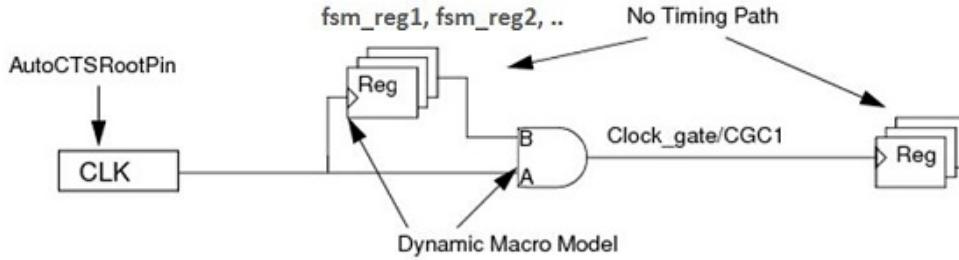
The `offset` parameter is optional. Instead of minimizing the difference between the arrival delays at the reference and the target pins, the `offset` parameter allows you to specify the offset arrival delay for the target pin. The offset arrival delay can be a positive or a negative value. A positive `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
...

```



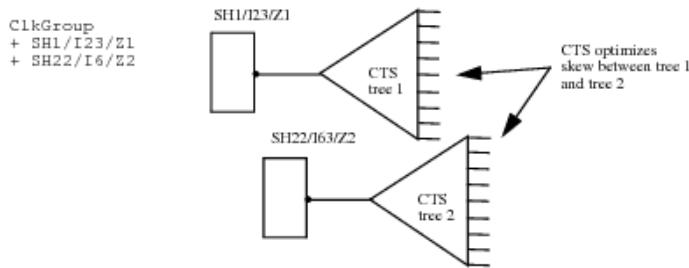
## Grouping Clocks

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

EDI System designs clock trees in a two-step, bottom-up fashion.

Within EDI System, 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.

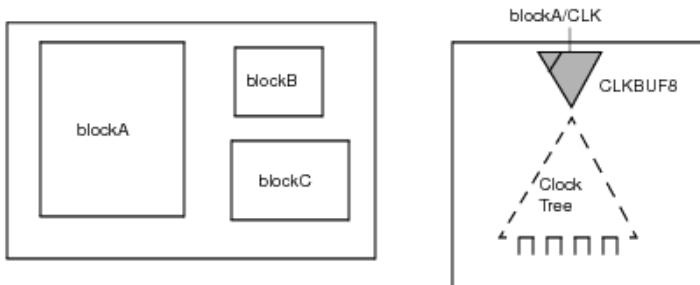
- i** The macro model specifications for each partition are at the top of the clock tree specification file.

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.

- Tip:** 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.

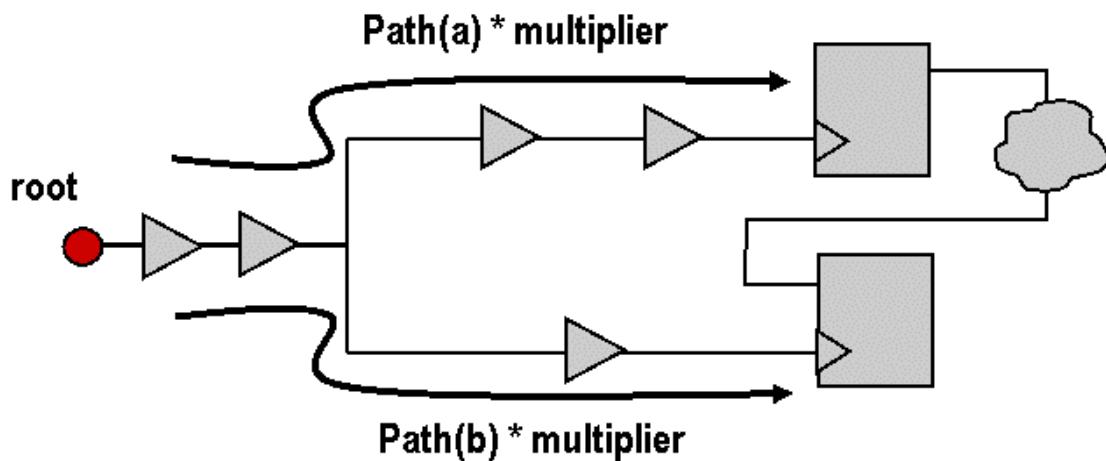
## Timing Model Requirement for Cells

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.

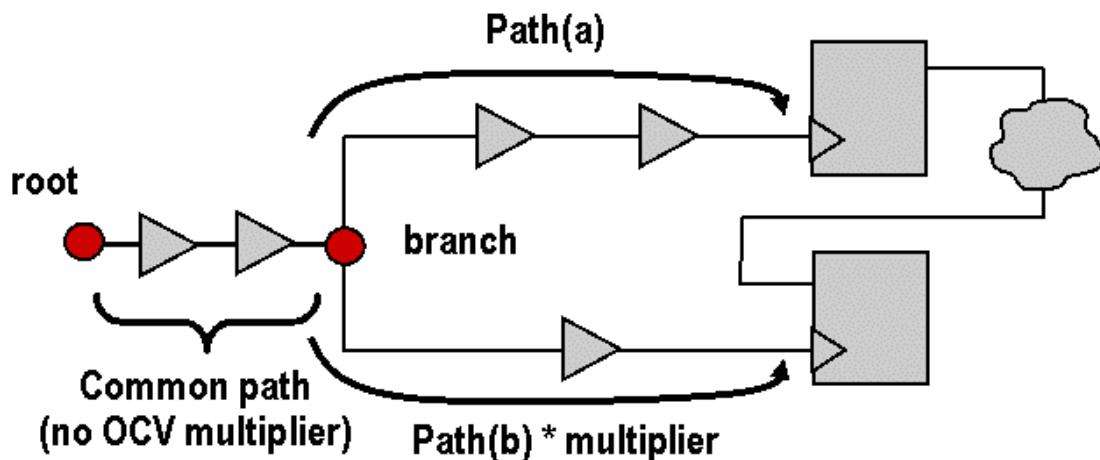
## Delay Variation and OCV

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 Post-CTS Clock Tree Optimization

### Using the ckECO Command for Post-CTS Clock Tree Optimization

Use the [ckECO](#) command for post-CTS optimization of clock tree(s) either in the same EDI 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 Encounter session (design import stage) when you want to perform local skew optimization.

At a minimum, the clock roots need to be defined in the SDC file so the [ckECO](#) command can identify the adjacent register pairs.

## Command Modes for the ckECO Command

The [ckECO](#) command can be run in 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 EDI session. Then run [reportClockTree](#) - `postRoute`. CTS then creates the clock report.

If clock nets are in a routed state you run the `ckECO` command, any wires that are disturbed by the `ckECO` command will automatically be rerouted using NanoRoute in an ECO mode.

## Running Post-CTS Optimization with the `ckECO` Command

You must load the clock tree specification file to run the `ckECO` command. (The clock tree specification file defines the clocks 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 Clock Tree Synthesis Commands chapter in *EDI System 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 EDI System 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.

- i 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.

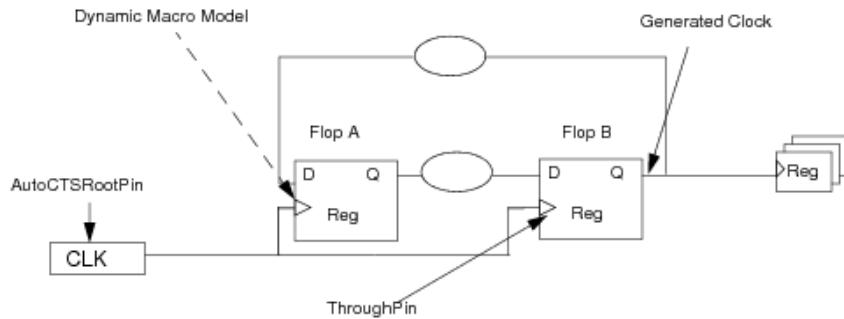
The auto clock tree specification file generator works differently depending on whether the software is in multi-mode multi-corner (MMMC) analysis mode or not.

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](#).

## 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

+ c/d/e/g\_reg

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><code>NameDelimiter # creates names with the format clk##L3#I2, rather than the default format, 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> |
| UseSingleDelim<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.

|                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UseCTSRouteGuide 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 <a href="#"><code>routeClockNetWithGuide</code></a> 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> |
| RouteTypeName <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 <i>ruleName</i>     | <p>Specifies the LEF <code>NONDEFAULTRULE</code> statement that the router should use.</p> <p><i>Default</i> : If you do not use this statement, the router uses the default routing rule.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| PreferredExtraSpace [0-3]          | <p>Specifies the spacing attribute, with which to add space around clock wires.</p> <p><i>Default</i> : If you do not use this statement, CTS uses a preferred extra space value of 1.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| TopPreferredLayer <i>number</i>    | <p>Specifies the top-most preferred routing layer.</p> <p><i>Default</i> : 4</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| BottomPreferredLayer <i>number</i> | <p>Specifies the bottom-most preferred routing layer.</p> <p><i>Default</i> : 3</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 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.             |
| <code>maxRiseDelay {ns ps}</code>                                        | Specifies the maximum rise delay in nanoseconds or picoseconds. |
| <code>minRiseDelay {ns ps}</code>                                        | Specifies the minimum rise delay in nanoseconds or picoseconds. |
| <code>maxFallDelay {ns ps}</code>                                        | Specifies the maximum fall delay in nanoseconds or picoseconds. |
| <code>minFallDelay {ns ps}</code>                                        | 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 delay_and_capacitance_values</i> |                                                                 |
|------------------------------------------------------------|-----------------------------------------------------------------|
|                                                            | Specifies the name of the macro model pin.                      |
| <code>maxRiseDelay {ns ps}</code>                          | Specifies the maximum rise delay in nanoseconds or picoseconds. |
| <code>minRiseDelay {ns ps}</code>                          | Specifies the minimum rise delay in nanoseconds or picoseconds. |
| <code>maxFallDelay {ns ps}</code>                          | Specifies the maximum fall delay in nanoseconds or picoseconds. |
| <code>minFallDelay {ns ps}</code>                          | Specifies the minimum fall delay in nanoseconds or picoseconds. |

The following table describes the entries for the global directive section.

- Global directive statements can not be used between `AutoCTSRootPin` and `End` statements.

|                                                                                              |                                                                                                                                                                                                                                                                                                                                                                      |                                                                           |
|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| <code>GlobalLeafPin<br/>+ pinName rising  <br/>falling<br/>+ ...</code>                      | Marks the input pin as a "leaf" pin for non-clock-type instances globally (throughout the design), stops tracing, and balances clock skew.<br><br><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.<br><br>Choose one of the following: |                                                                           |
|                                                                                              | <code>rising</code>                                                                                                                                                                                                                                                                                                                                                  | CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.  |
|                                                                                              | <code>falling</code>                                                                                                                                                                                                                                                                                                                                                 | CTS treats the input pin as a falling-edge-triggered flip-flop clock pin. |
| <code>GlobalLeafPort<br/>+ cellType /<br/>inputPinName rising  <br/>falling<br/>+ ...</code> | Marks the pin as a "leaf" pin for cells globally (throughout the design), stops tracing, and balances clock skew.<br><br>Choose one of the following:                                                                                                                                                                                                                |                                                                           |
|                                                                                              | <code>rising</code>                                                                                                                                                                                                                                                                                                                                                  | CTS treats the pin as a rising-edge-triggered flip-flop clock pin.        |
|                                                                                              | <code>falling</code>                                                                                                                                                                                                                                                                                                                                                 | CTS treats the pin as a falling-edge-triggered flip-flop clock pin.       |
| <code>GlobalExcludedPin<br/>+ pinName<br/>+ ...</code>                                       | Treats the pin as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.                                                                                                                                                                                                                                                |                                                                           |
| <code>GlobalExcludedPort<br/>+ cellType / inputPinName<br/>+ ...</code>                      | 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.                                                                                                                                                                                                                        |                                                                           |
| <code>GlobalUnsyncPin<br/>+PinName</code>                                                    | Treats the pin as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.                                                                                                                                                                                                                                                |                                                                           |
| <code>GlobalUnsyncPort<br/>+cellType/inputPinName</code>                                     | 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.                                                                                                                                                                                                                        |                                                                           |
| <code>GlobalThroughPort</code>                                                               | Traces through the pin of a particular cell type globally                                                                                                                                                                                                                                                                                                            |                                                                           |

|                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| +cellType/ <i>PinName</i><br>outputPinNames ] [                      | (throughout the design), even if the pin is a clock pin.<br><br>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.<br><br>For example, if you want CTS to trace through the output pin Q of library cell DFF, specify the following in the clock tree specification file:<br><br>GlobalThroughPort<br>+ DFF/Q                                                                                                                                  |
| GlobalPreservePort<br>+cellType /<br><i>inputPinName</i>             | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| GlobalThroughPin<br>+ <i>pinName</i> [<br>outputPinNames ]<br>+ ...  | Traces through the pin globally (throughout the design), even if the pin is a clock pin.<br><br>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.<br><br>For example, if you want CTS to trace through an instance top/mmcore from the input pin clk to clk01 and clk02, although this instance has another clock output clk03, specify the following in the clock tree specification file:<br><br>ThroughPin<br>+ top/mmcore/clk clk01 clk02 |
| GlobalPreservePin<br>+ <i>inputPinName</i><br>+ ...                  | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| GlobalValidCell<br>+ <i>cellName1</i><br>+ <i>cellName2</i><br>+ ... | Specifies all cells that can be used by CTS as GlobalValid 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 GlobalValidCell list. Once a cell type is                                                                                                                                                                                                                                                                                                                                                                                                                          |

|                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                               | <p>specified as GlobalValidCell, CTS ignores the dont_touch and dont_use attributes.</p> <p>This statement is honored by the ckSynthesis, ckECO, ckCloneGate, ckDecloneGate, and deleteClockTree commands.</p> <p><b>Note:</b> The GlobalValidCell statement must be specified outside the AutoCTSRootPin and End statement.</p>                                                                                                                                                                                                                                                                                                                                                                |
| DontTouchNet<br>+ <i>netName1</i><br>+ <i>netName2</i><br>+ ...               | <p>Specifies a list of nets for which the ckSynthesis and ckEco commands should not insert buffers. The deleteClockTree command does not delete buffers if their input or output nets have the DontTouchNet attribute.</p> <p><b>Note:</b> The DontTouchNet statement is not a physical parameter; any net specified in this statement can still be routed.</p>                                                                                                                                                                                                                                                                                                                                 |
| DontTouchFromToPin<br>+ <i>pinName1 pinName2</i>                              | <p>Instructs the ckSynthesis and ckEco commands to not insert buffers for nets that are between the specified start instance pin and end instance pin. Any nets between these pins are considered to have the DontTouchNet attribute.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| DontAddNewPortModule<br>+ <i>moduleName1</i><br>+ <i>moduleName2</i><br>+ ... | <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 DontAddNewPortModule statement applies only to the specified modules but not the nested submodules. When you specify this statement for any given module, only that module does not have a new port added to it but the nested submodule still might have a new port added.</li> <li>■ The DontAddNewPortModule statement might increase the latency of the clock tree and more buffers could be added because due to this restriction, the instances of different modules cannot be shared by the same buffer.</li> </ul> |
| LeafPinGroup <i>groupName</i>                                                 | <p>Instructs CTS to balance the group of leaf pins separately with the main clock tree. By using LeafPinGroup, the skew for leaf pin groups is reported separately.</p> <p>Specify LeafPinGroup 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</p>                                                                                                                                                                                                                                                                                                                       |

balanced.

**Note:** Specifying group name is optional. If group name is not specified, by default, CTS uses names such as `LeafPinGroup0`, `LeafPinGroup1`, and so on.

## Clock Grouping Data Section

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

|                                                                                                          |                                                                                 |
|----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <code>clkGroup</code><br>+<br><code>clockRootPinName</code><br>+<br><code>clockRootPinName</code><br>... | Specifies two or more clock domains for which you want CTS to balance the skew. |
|----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|

## Clock-Tree Topology Section

The clock-tree topology section provides a method for you to manually define buffers at particular levels. The following table describes the entries for the clock-tree topology section:

|                                                                               |                                                                                                                 |                                                                                                  |
|-------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| <code>ClockNetName netName</code>                                             | Specifies the name of the clock net.                                                                            |                                                                                                  |
| <code>LevelNumber</code><br><code>totalNumberOfClockTreeLevels</code>         | Specifies the total number of clock tree levels.                                                                |                                                                                                  |
| <code>LevelSpec levelNumber</code><br><code>numberOfBuffers bufferType</code> | <p>Provides the specifications for an individual clock level.</p> <p>Specify all the following information:</p> |                                                                                                  |
|                                                                               | <code>levelNumber</code>                                                                                        | Sets the level number in the clock tree.                                                         |
|                                                                               | <code>numberOfBuffers</code>                                                                                    | Specifies the total number of buffers CTS should allow on the specified level of the clock tree. |
|                                                                               | <code>bufferType</code>                                                                                         | Specifies the buffer type (based upon the LEF file).                                             |
| <code>End</code>                                                              | 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:

|                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GlobalPowerDomainBuffer</b><br>+<br><i>PowerDomainNames</i><br><i>ListofBuffers</i> | Specifies the power domain names and a list of buffers associated with the power domain.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>AutoCTSRootPin</b><br><i>clockRootPinName</i>                                       | Specifies the name of the clock root pin name from which to start tracing.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>CellObstruction</b><br>+ <i>cellName</i><br>Entire Ignored Detailed HV              | <p>Overrides CTS's obstruction modeling for a cell.</p> <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> <li>▪ <b>Detailed:</b> 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.</li> </ul> <p>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.</p> <ul style="list-style-type: none"> <li>▪ <b>HV:</b> Specifies the cell name and instructs CTS to honor routing obstruction of the cell for layers that obstructs the preferred routing layers.</li> </ul> <p>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.</p> <p><i>Default :</i> No manual overriding, the obstruction modeling is auto-computed by tool.</p> |
| <b>InstanceObstruction</b><br>+ <i>InstanceName</i><br>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</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                | <p>CTS to ignore routing obstruction due to this instance.</p> <ul style="list-style-type: none"> <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>Default :</i> No manual overriding, the obstruction modeling is auto-computed by tool.</p> |
| ForceMaxFanout {Yes   No}      | <p>Specifies the maximum fanout as a soft constraint. If the value is specified as yes, then the maximum fanout is specified as a hard constraint.</p> <p><i>Default :</i> No</p>                                                                                                                                                                                                                                                                                                                                                                                                         |
| SkipRouteGuide {Yes   No}      | <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> No</p>                                                                                                                                                                                  |
| MaxDelay <i>number</i> {ns ps} | <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>                                                                                                                                                                                                                                                                                                                                                                                                                        |
| MinDelay <i>number</i> {ns ps} | <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>                                                                                                                                                                                                            |
| SetBBoxPinAsSync<br>{true false}        | <p>Treats I/O pins as synchronous pins, instead of as excluded pins.</p> <p>For example, if you specify <a href="#">setCTSMode</a> as true and setBBoxPinAsSync as false, then all other clocks are true except the one that is specified as false in the clock tree specification file.</p> <p><i>Default</i>: false</p> |
| SinkMaxTran <i>number</i> {ns ps}       | <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>                                           |
| BufMaxTran <i>number</i> {ns ps}        | <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>               | <p>Specifies the maximum gate ratio for the clock tree. Gate ratio is defined as:</p> <p>sum of all cell delay / clock path delay (from clock root to all registers)</p> <p><i>Default</i>: 1</p>                                                                                                                         |
| MinGateRatio <i>ratio</i>               | <p>Specifies the minimum gate ratio for the clock tree. Gate ratio is defined as:</p> <p>sum of all cell delay / clock path delay (from clock root to all registers)</p> <p><i>Default</i>: 0</p>                                                                                                                         |
| MaxDeltaGateRatio<br><i>delta_ratio</i> | <p>Specifies the maximum delta gate ratio for the clock tree. Delta gate ratio is defined as the difference between fmGateRatio and toGateRatio, where:</p> <ul style="list-style-type: none"> <li>■ fmGateRatio is the sum of all cell delay divided by the clock path delay from the branching point to the</li> </ul>  |

|                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                      | <p>source register.</p> <ul style="list-style-type: none"> <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>                                                                                                                                                                                                                                                                                                                 |
| <code>MinDeltaGateRatio</code><br><i>delta_ratio</i> | <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> |
| <code>SrcLatency</code> <i>number {ns}</i>           | <p>Specifies an external latency value for each clock.</p> <p>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 MaxSkew 300ps ... End </pre>                                                                                                                                                                                                                                                                                                                                                                                                                             |

```

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:

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

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

*Default*: 0ns

`RootInputTran number {ns|ps}`

Specifies the input transition time for an input pin or an input port. Use this statement when you do not want CTS to use the default Encounter input transition time.

**Note:** If your SDC file contains a set\_input\_transition constraint, the RootInputTrans statement overrides that constraint.



You cannot use this statement with output pins or output ports.

`MaxSkew number {ns|ps} [viewName ]`

Specifies the maximum skew between sinks (clock pins).

You can specify different skew limits to be used for

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                               | <p>specific active analysis views (<i>viewName</i>) for multi-mode CTS.</p> <p><i>Default</i> : If you do not use this statement, CTS uses a maximum skew constraint of 300 ps.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| DefaultMaxCap <i>integer</i>  | <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 MaxCap and PinMaxCap statements.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| MaxFanout <i>integer</i>      | <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><u>setCTSMODE</u> - 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>                                                                                                                                                                                                |
| MaxNumLevel <i>number</i>     | <p>Specifies the maximum number of buffer levels that CTS builds in a clock tree that has no gating components.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 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. (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> $(\text{activity of clock net at gate's input}) \times \text{ClockGatingProb}$ <p>CTS computes the activity rate of a net driven by a buffer in the clock path as:</p> |

|                                  |         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                  |         | ( <i>activity of clock net at buffer's input</i> ) x<br><i>ClockGatingProb</i><br><br><i>Default:</i> 1 (clock is always on)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 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;"> <span style="color: #0070C0; font-size: 1.5em;">i</span> You cannot use this statement for gated clocks.     </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      | Allows CTS to trace through clock gating logic.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

|                                                                      |                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                      | <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<br><i>driver_cell_name</i>                             | <p>Places a driver cell name at the closest possible location to the clock port location for block-level CTS.</p>                                                                                                                                                                                                                                                                    |
| MaxDepth <i>number</i>                                               | <p>Sets the maximum depth of clock tree tracing.</p> <p><i>Default</i> : If you do not use this statement, CTS limits the number of levels of clock tree tracing to 1024.</p>                                                                                                                                                                                                        |
| RouteType <i>routeTypeName</i>                                       | <p>Specifies the name of the routing attributes definition.</p> <p><b>Note:</b> CTS uses the RouteType statement <i>only</i> if you specify RouteClkNet YES.</p> <p><b>Note:</b> If you use a RouteType statement, you must also use a corresponding <i>routeTypeName</i>.</p>                                                                                                       |
| LeafRouteType<br><i>leafRouteTypeName</i>                            | <p>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.</p> <p>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.</p> |
| cellRouteType<br>+ <i>cellName</i> <i>RouteTypeName</i><br>+ ...     | <p>Specifies the routing attribute for the output net driven by the specified cell.</p>                                                                                                                                                                                                                                                                                              |
| cellLeafRouteType<br>+ <i>cellName</i> <i>RouteTypeName</i><br>+ ... | <p>Specifies the routing attribute for the output net driven by the specified leaf cell. This applies only to the leaf nets.</p>                                                                                                                                                                                                                                                     |
| DetailReport YES   NO                                                | <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>                                                             |

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Obstruction YES   NO | Specifies whether CTS should take design obstructions into account during flip-flop clustering.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| RouteClkNet YES   NO | <p>Specifies whether CTS routes clock nets.</p> <p>The following combinations of the RouteClkNet specification file statement and the <a href="#">setCTSMODE</a> - routeGuide true command are possible:</p> <ul style="list-style-type: none"> <li>▪ If you specify RouteClkNet NO, CTS does not route clock nets. Timing is based on the pre-routed design using a steiner model. The RouteClkNet NO statement is useful for rapid prototyping.</li> <li>▪ If you specify RouteClkNet YES together with the setCTSMODE - routeGuide false command, CTS routes clocks using NanoRoute but CTS does <i>not</i> use a routing guide file. This combination of settings is useful for straightforward designs for which pre-route-post-route correlation is not an issue. CTS runs more quickly than when it uses a guide file. Note that NanoRoute routes the clocks but does not follow the pattern determined by CTS. Thus, this method cannot take advantage of the balanced routing that takes place during CTS.</li> <li>▪ If you specify RouteClkNet YES together with the setCTSMODE - routeGuide true command, CTS routes the clocks with NanoRoute and the CTS routing guide file. This method ensures appropriate pre-route-post-route correlation but at the expense of longer runtime.</li> </ul> <p><i>Default</i> : If you do <i>not</i> use this statement, CTS does not route clock nets. In other words, the default behavior is as if you had specified RouteClkNet NO.</p> |
| 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</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

|                                              |                                                                                                                                                                                                                                                                                                                                                                                                            |                 |                                                                           |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|---------------------------------------------------------------------------|
|                                              | <p>constraints as defined in the clock tree specification file. However, phase delay, buffer transition time, and sink transition times are not necessarily improved by using these two statements.</p> <p><b>Note:</b> If you specify PostOpt No, CTS does <i>not</i> resize gating components. If you specify PostOpt YES, CTS <i>attempts</i> to resize gating components, though it may not do so.</p> |                 |                                                                           |
| OptAddBuffer {YES   NO}                      | <p>Controls whether CTS adds buffers during optimization.</p> <div style="border: 1px solid #4682B4; padding: 5px; margin-top: 10px;"> <span style="color: #4682B4;">i</span> The OptAddBuffer statement is effective <i>only</i> if you specify PostOpt YES.         </div>                                                                                                                               |                 |                                                                           |
|                                              | <p>When used together, the Postopt and optAddBuffer statements try to meet the trigger edge skew constraints as defined in the clock tree specification file. However, phase delay, buffer transition time, and sink transition times are not necessarily improved by using these two statements.</p>                                                                                                      |                 |                                                                           |
| OptAddBufferLimit<br><i>positive_integer</i> | <p>Specifies the maximum number of buffers that CTS can add to a clock tree during optimization. The number you specify can be no smaller than 1. However, the higher the number you specify, the longer the runtime.</p> <p><i>Default :</i> If you do not use this statement, CTS inserts no more than 50 buffers.</p>                                                                                   |                 |                                                                           |
| AddSpareFF <i>cellName</i> <i>number</i>     | <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>      |                 |                                                                           |
|                                              | <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;"><i>cellName</i></td><td style="padding: 5px;">Represents the name of the flip-flop(s) to be inserted in the clock root.</td></tr> </table>                                                                                                                                                               | <i>cellName</i> | Represents the name of the flip-flop(s) to be inserted in the clock root. |
| <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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Buffer <i>cell1 cell2 cell3 ...</i>                   |               | <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 <i>cell1 cell2 cell3 ...</i>              |               | <p>Specifies the names of dummy buffer cells to use during the optimization process.</p> <p>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.)</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> |
| LeafBuffer <i>buffer_list</i>                         |               | <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 LeafBuffer 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>                                                                 |
| LeafPin<br>+ <i>pinName</i> rising   falling<br>+ ... |               | <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 LeafPin statement with an input pin.</p>                                                                                                                                                                                                                                                                                                                                                                                                          |

|                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                               |                                                                           |
|-----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
|                                                                                                           | Choose one of the following:                                                                                                                                                                                                                                                                                                                                                  |                                                                           |
|                                                                                                           | <code>rising</code>                                                                                                                                                                                                                                                                                                                                                           | CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.  |
|                                                                                                           | <code>falling</code>                                                                                                                                                                                                                                                                                                                                                          | CTS treats the input pin as a falling-edge-triggered flip-flop clock pin. |
| <code>LeafPort</code><br>+ <code>cellType / inputPinName</code><br><code>rising   falling</code><br>+ ... | <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>                                                                                                                                                                                                      |                                                                           |
|                                                                                                           | <code>rising</code>                                                                                                                                                                                                                                                                                                                                                           | CTS treats the pin as a rising-edge-triggered flip-flop clock pin.        |
|                                                                                                           | <code>falling</code>                                                                                                                                                                                                                                                                                                                                                          | CTS treats the pin as a falling-edge-triggered flip-flop clock pin.       |
| <code>ExcludedPin</code><br>+ <code>pinName</code><br>+ ...                                               | <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>                                                                                                                                                                                    |                                                                           |
| <code>ExcludedPort</code><br>+ <code>cellType / inputPinName</code><br>+ ...                              | <p>Treats the pin of a particular cell type as a non-leaf pin, and prevents tracing and skew analysis of the pin.</p>                                                                                                                                                                                                                                                         |                                                                           |
| <code>UnsyncPin</code><br>+ <code>pinName</code><br>+ ...                                                 | <p>Treats specified pin as a non-leaf pin, and prevents CTS tracing and skew balancing of the pin. <code>UnsyncPin</code> is specified between <code>AutoCTSRootPin</code> and <code>END</code> statement and it only affects a single clock.</p> <p><b>Note:</b> CTS fixes the DRV for the <code>UnsyncPin</code>, which is different than the <code>ExcludedPin</code>.</p> |                                                                           |
| <code>KeepPortPolarity</code><br>+ <code>moduleName /portName</code>                                      | <p>Preserves the polarity of specified module ports while running the <a href="#">deleteClockTree</a> command.</p>                                                                                                                                                                                                                                                            |                                                                           |
| <code>ThroughPin</code><br>+ <code>pinName [ outputPinNames ]</code><br>+ ...                             | <p>Traces through the pin, 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</p>                                                                                                            |                                                                           |

|                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                 | <p>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>                                                   |
| ThroughPort<br>+ <i>cellType</i> / <i>inputPinName</i><br>+ ... | <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 <code>ThroughPin</code> or <code>ThroughPort</code>.</p>                                                                                                                                                                                                                                                                                                                 |
| SetDPinAsSync YES   NO                                          | <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>Default</i> : If you omit this statement, CTS treats the Data pins of flip-flops as excluded pins.</p> |
| SetIoPinAsSync YES   NO                                         | <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>Default</i> : If you omit this statement, CTS treats I/O pins as excluded pins.</p>                                                                                                                                                                                                                                                    |

|                                                                                                                                                                                                               |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PreservePin<br>+ <i>inputPinName</i><br>+ ...                                                                                                                                                                 |                   | Preserves the netlist for the pin and pins below the pin in the clock tree. However, CTS considers any synchronized pins after the pin when computing skew.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| PinMaxCap<br>+ <i>instanceName / pinName1 capValue1 {pf   ff}</i><br>+ <i>instanceName / pinName2 capValue2 {pf   ff}</i><br>...<br>                                                                          |                   | <p>Specifies the maximum capacitance value for the specified pins.</p> <p><i>Default</i> : If there is no PinMaxCap statement specified in the clock tree spec file, but there is a DefaultMaxCap statement specified, the software uses the DefaultMaxCap value to constrain pins.</p> <p>If there are no PinMaxCap OR DefaultMaxCap statements specified in the clock tree spec file, but you specify setCTSMode - useLibMaxCap true, the software uses maximum capacitance values in the timing library.</p> <p>If there are no PinMaxCap OR DefaultMaxCap statements specified in the clock tree spec file and you do not specify setCTSMode - useLibMaxCap true, the software does not apply a maximum capacitance constraint.</p>                                                    |
| MaxCap<br>+ <i>bufferName1 capValue1 {pf   ff}</i><br>+ <i>bufferName2 capValue2 {pf   ff}</i><br>+ <i>clockGatingCell1 capValue1 {pf   ff}</i><br>+ <i>clockGatingCell2 capValue2 {pf   ff}</i><br>+ ...<br> |                   | <p>Specifies a maximum capacitance value for the specified buffers, inverters, or gating cells.</p> <p><i>Default</i> : If there is no MaxCap statement specified in the clock tree spec file, but there is a DefaultMaxCap statement specified, the software uses the DefaultMaxCap value to constrain buffers, inverters, or gating cells.</p> <p>If there are no MaxCap OR DefaultMaxCap statements specified in the clock tree spec file, but you specify setCTSMode - useLibMaxCap true, the software uses maximum capacitance values in the timing library.</p> <p>If there are no MaxCap OR DefaultMaxCap statements specified in the clock tree spec file and you do not specify setCTSMode - useLibMaxCap true, the software does not apply a maximum capacitance constraint.</p> |
|                                                                                                                                                                                                               | <i>bufferName</i> | Specifies the name of the buffer for which you specify a maximum capacitance value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|                                                                                                                                                                                                               | <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 ExcludedBuffer 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 ExcludedPin statement in the clock tree specification file.</p> <p><i>Default:</i> If you do not specify the ExcludedBuffer statement, the software chooses a buffer to insert from the Buffer 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 ForceMaxTran YES and run ckECO -fixDRVOnly, the software attempts to correct maximum transition violations.</p>                                                                                                                                                                                                                                                                                                                                                                                                                    |
| ForceReconvergent YES   NO | <p>Controls whether CTS allows or prevents reconvergence conditions for individual clocks.</p> <ul style="list-style-type: none"> <li>■ YES indicates that you want CTS to allow reconvergence on a particular clock. CTS synthesizes such clocks.</li> <li>■ 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.</li> </ul> <p>The following example illustrates how to specify different reconvergence conditions on clocks clk and clk2:</p> <pre>AutoCTSRootPin clk ... ForceReconvergent YES ...</pre> |

|                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                     | <pre> END  AutoCTSRootPin clk2  ... ForceReconvergent NO  ... END </pre> <p><b>Note:</b> You can override any ForceReconvergent statement by specifying <a href="#">ckSynthesis</a> - forceReconvergent . 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</i><br><i>instanceName</i> ...<br>+ <i>instanceName</i><br><i>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<br>+ <i>moduleName</i><br>+ <i>moduleName</i><br>+ ...                                                              | <p>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.</p> <ul style="list-style-type: none"> <li>■ Each line that begins with a + represents a module and its instances.</li> <li>■ You can specify only one module on each line.</li> <li>■ You can use the wildcards * and ?.</li> </ul>                                                                                                                                                  |

|                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| <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 <math>\mu\text{m}</math> 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 <math>\mu\text{m}</math>.</p> <p>If you specify 3 <math>\mu\text{m}</math> for BUFX4 and 4 <math>\mu\text{m}</math> for BUFX6 and if the two buffers are placed close to each other, then the spacing between these buffers is 4 <math>\mu\text{m}</math> 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.                           |
| <pre>MasterGateModule + <i>numClusters1</i> <i>module1</i> + <i>numClusters2</i> <i>module2</i> + ...</pre>                                                           | <p>Controls module cloning:</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                         |
|                                                                                                                                                                       | <i>numClusters</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Indicates how many times each module is to be cloned.   |
|                                                                                                                                                                       | <i>module</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Represents the module name.                             |
| <pre>MasterGateTargetFanout <i>number</i></pre>                                                                                                                       | <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</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                         |

|                                                                                                    |                                                                                                                                                                                                            |
|----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                    | gating cells might differ depending on the loads the gating cells drive.                                                                                                                                   |
| EquivGateInst<br>+ <i>instance1</i><br>+ <i>instance2</i><br>+ <i>instance3 instance4</i><br>+ ... | Specifies the instances to be cloned. <code>ckDeCloneGate</code> merges the instances you specify into one instance.<br><br>To declone groups of gating cells, specify more than one instance on one line. |
| End                                                                                                | Marks the end of an automated gated CTS section.                                                                                                                                                           |

## Log File Headings

Given particular settings for the `RouteClkNet` and `PostOpt` statements, and for the [reportClockTree](#) command, the `encounter.log` file reports results in the following sections of the log file:

| Clock tree specification file statements     | Example clock tree text command sequences                                                            | EDI System log file section heading: <code>Clock <i>clockName</i> plus--</code> |
|----------------------------------------------|------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <code>RouteClkNet NO</code>                  | <code>ckSynthesis<br/>globalDetailRoute<br/>reportClkTree -clk<br/><i>clock</i></code>               | Pre-Route Timing Analysis                                                       |
| <code>RouteClkNet YES</code>                 | <code>ckSynthesis<br/>globalDetailRoute<br/>reportClkTree -clk<br/><i>clock</i> -clkRouteOnly</code> | Clk-Route-Only Timing Analysis                                                  |
| <code>RouteClkNet YES<br/>PostOpt YES</code> | <code>ckSynthesis<br/>globalDetailRoute<br/>reportClkTree -clk<br/><i>clock</i> -postRoute</code>    | Post-Route Timing Analysis                                                      |

## CTS Report Descriptions

The standard CTS report (`top_level_cell.ctsrpt`) contains several sections by default, and several sections that are controlled by the settings of various CTS text commands.

**Note:** The report extracts that follow are based on various designs, and should not be construed as results from a single CTS run.

### General Information

The beginning of each CTS report contains the following general information about the clock tree:

- Library information

```
#####
# Complete Clock Tree Timing Report
#
# CLOCK: cgen/i_5/Y
#
# Mode: preRoute
#
# Library Name : slow
# Operating Condition : slow
# Process : 1
# Voltage : 1.62
# Temperature : 125
#
#####
```

```
# 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:

( activity of clock net at gate's input ) x ClockGatingProb

CTS computes the activity rate of a net driven by a buffer in the clock path as:

( activity of clock net at buffer's input ) x ClockGatingProb

CTS uses the same net activity for calculating the switching power of the net.

NetSwitchPower : 0.000997444(mW)

InstInternalPower : 0.00409096(mW)

InstLeakagePower : 1.9129e-06(mW)

LeafPinInternalPower : 0.0013708(mW)

Total Power : 0.00646111(mW)

## AC Current Density Violations

Information on AC current density violations appears in the standard CTS report only if your LEF file contains an ACCURRENTDENSITY statement.

AC Irms Limit Violation : 0.387332(mA) (17216.7%)

Information on AC current density violations appears in a special violations section:

\*\*\*\*\* AC Irms Limit Violation \*\*\*\*\*

Pin Name (Actual) (Required) (Violation)

-----  
ClkMux/Y 0.389581(mA) 0.00224974(mA) 0.387332(mA)

scanClk/Y 0.156409(mA) 0.00224974(mA) 0.154159(mA)

## Supported SDC Constraints

Clock Tree Synthesis supports only the following SDC timing constraints during tracing ([setCTSMODE -traceHonorConstants true](#)):

- set\_logic\_one
- set\_logic\_zero
- set\_case\_analysis
- set\_disable\_timing

The `createClockTreeSpec` command automatically translates certain SDC constraints into clock tree specification file statements. The following table shows the SDC constraints that the

`createClockTreeSpec` command automatically translates, and also shows the default values in those statements if an SDC constraint does not exist:

| SDC Constraint                               | Clock Tree Specification File Statement (default)                   |
|----------------------------------------------|---------------------------------------------------------------------|
| <code>create_clock</code>                    | <code>AutoCTSRootPin</code>                                         |
| <code>set_clock_transition</code>            | <code>SinkLeafTran and BufMaxTran (Default: 400 ps)</code>          |
| <code>set_clock_latency value</code>         | <code>MaxDelay (Default: clock period) MinDelay (Default: 0)</code> |
| <code>set_clock_latency -source value</code> | <code>SrcLatency value ns</code>                                    |
| <code>set_clock_uncertainty</code>           | <code>MaxSkew (Default: 300 ps)</code>                              |
| <code>create_generated_clock</code>          | <code>Add ThroughPin when necessary</code>                          |

## Clock Concurrent Optimization

### Overview

Clock Concurrent Optimization (CCOpt) replaces traditional CTS, combining advanced "timing-driven" CTS with datapath optimization.

With traditional CTS tool flows, an ideal clock model is used before CTS to simplify clock timing analysis. With the ideal clock model, launch (L) and capture clock (C) paths are assumed to have the same delay.

During CTS, the ideal clock model is replaced by a propagated clock model that takes account of actual delays along clock launch and capture paths. Traditional CTS solutions attempt to make the propagated solution match the ideal clock solution by balancing launch and capture clock path delays. This keeps clock skew (the difference between the longest and shortest clock tree path delays for sinks with a common parent) to a minimum.

However, a number of factors combine to make it impossible to produce perfectly balanced clock trees:

- OCV (on-chip variation) means that wires that are designed to be identical will differ once they are implemented in silicon.
- Clock gating relies on using datapath signals to switch clock propagation on and off. Clock path delays to gated sinks can only be calculated once the path delays to the clock gate are known (and are therefore unknown prior to CTS).
- Complex clock designs with muxes, clock generators, and clock gates render the

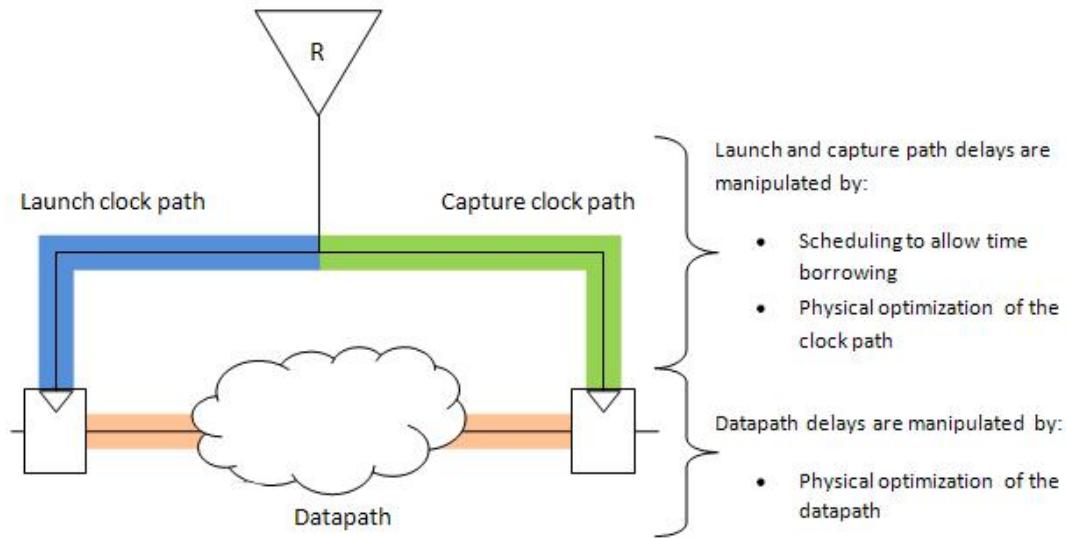
definition of clock skew itself non-obvious: rather than a tree or set of trees we have a network with hundreds of sources and hundreds of thousands of sinks. Commonplace scenarios exist where making launch and capture clock path delays equal for all flip-flops is mathematically impossible.

Whereas traditional CTS assumes launch and capture path delays to be equal in order to build a balanced clock tree, CCOpt merges physical optimization with CTS and directly manipulates all the variables that impact propagated clock timing:

- Launch clock path delay
- Capture clock path delay
- Minimum logic path delay
- Maximum logic path delay

CCOpt treats both clock delays and logic delays as flexible parameters that can be manipulated independently to optimize timing. This is shown in the figure below.

### Manipulating Clock Delays and Logic Delays



At each flop in the design, CCOpt can adjust delays to move slack around in the circuit.

### The CCOpt Flow in the EDI System

There are two ways you can run CCOpt within the EDI System using the [ccopt\\_design](#) command:

- Scripted Integration Mode – The EDI system produces a CCOpt control script, starts a separate CCOpt binary, optimizes the design in CCOpt, then returns to the EDI System

and loads the optimized design. The CCOpt Configuration is done using the [set\\_ccopt\\_mode](#) command.

- Native Integration Mode – All CCOpt optimization is performed within EDI, with no need to start separate binaries or produce control scripts. The CCOpt configuration is done using the [set\\_ccopt\\_property](#) command.

You can choose to run CCOpt in either of the two modes by using the [set\\_ccopt\\_mode](#) command's `-integration` parameter. You can use this command to specify the integration mode of your choice - scripted or native. By default, the scripted mode is selected.

Run the following command to switch to the native mode of integration:

```
set_ccopt_mode -integration native
```

You can also choose the native integration mode by using the [set\\_ccopt\\_property](#) command.

```
set_ccopt_property integration native
```

The CCOpt flow in both modes is described in detail in subsequent sections.

## The Scripted Integration Mode

In this mode, when [ccopt\\_design](#) command is run, the EDI System produces a CCOpt control script, starts the separate CCOpt binary, optimizes the design in CCOpt and then returns to EDI and loads the optimized design. The CCOpt configuration is performed using the [set\\_ccopt\\_mode](#) command.

The following commands are used to control and run CCOpt in the scripted CCOpt flow:

- [ccopt\\_design](#)  
Performs CCOpt on the current loaded design.
- [set\\_ccopt\\_mode](#)  
Configures options that control how `ccopt_design` performs optimization.
- [get\\_ccopt\\_mode](#)  
Displays current values for options set using the [set\\_ccopt\\_mode](#) command.
- [generate\\_ccopt\\_rc\\_factor](#)  
Calculates multipliers by comparing calculated capacitances and resistances with SPEF values. This results in more accurate estimates for resistance and capacitance values.

## The Scripted CCOpt Flow

This is the default flow for CCOpt. The basic flow for using CCOpt in the scripted mode is as follows:

1. Load your placed design in the EDI System.
2. Perform CCOpt configuration using the [set\\_ccopt\\_mode](#) `-integration`

scripted command.

3. Run CCOpt using the [ccopt\\_design](#) command to optimize the design.

The above points are detailed below:

### Step 1: Load Design

Load your design in the EDI System.

### Step 2: CCOpt Configuration

The [set\\_ccopt\\_mode](#) command lets you perform basic CCOpt configuration from within CCOpt.

For detailed information about the CCOpt commands, see [Clock Concurrent Optimization \(CCOpt Commands\)](#) chapter in the *EDI System Text Command Reference* document.

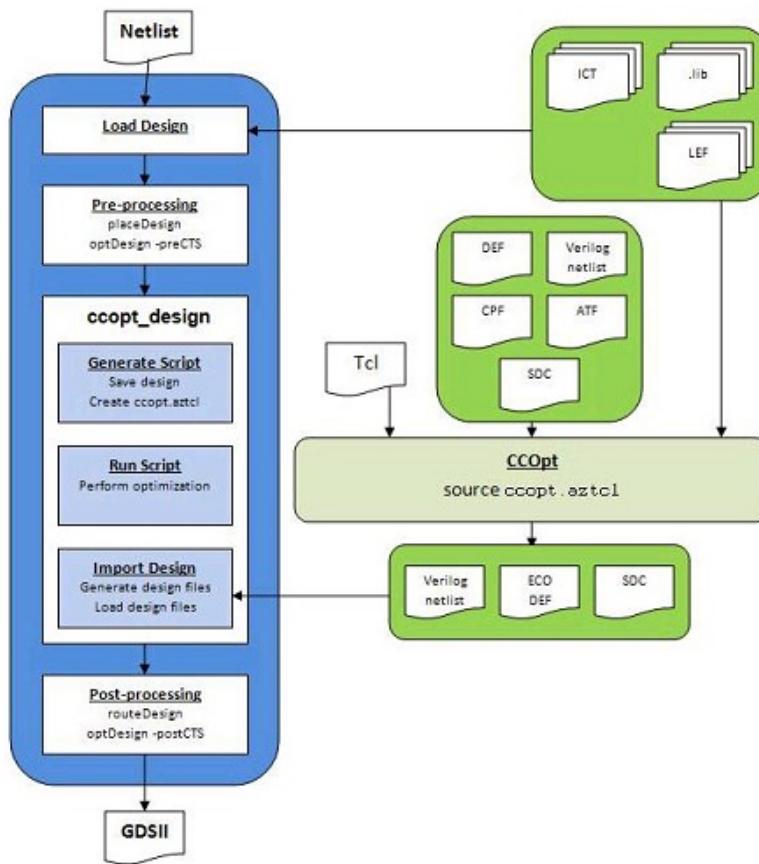
### Step 3: Running the `ccopt_design` Command

The [ccopt\\_design](#) command performs CCOpt optimization on the currently loaded design in the EDI System.

The basic flow of operations performed by the `ccopt_design` command is as follows:

1. **Generate script:** The EDI System saves the design, generating a number of files (netlist, DEF and so on) in the `az_ccopt` directory. It generates a Tcl script `az_ccopt/ccopt.aztcl` from the current EDI System configuration. This script is used to configure and control CCOpt.
2. **Run CCOpt:** The EDI System starts CCOpt, running the `az_ccopt/ccopt.aztcl` script, which does the following:
  - Loads the design files as exported into the `az_ccopt` directory in **Step 1**
  - Performs all required configuration
  - Saves a pre-optimization design database in `az_ccopt/<design>.pre.azdb`
  - Runs the CCOpt optimization
  - Saves a post-optimization design database in `az_ccopt/<design>.azdb`
3. **Import design:** The EDI System takes the optimized design database (`az_ccopt/<design>.azdb`) and generates the design files (Verilog netlist, DEF, SDC and so on) in the `az_import` directory. The design is then reloaded from these optimized design files in `az_import`. The process is illustrated in the figure below.

### The `ccopt_design` Command Flow



By default, the [ccopt\\_design](#) command performs all these operations as a single step. However, it is also possible to run [ccopt\\_design](#) in stages, in order to facilitate debugging or allow you to make changes to the Tcl script that controls CCOpt. The staged [ccopt\\_design](#) flow is as follows:

1. Export design files and generate the script that will be passed to CCOpt.

```
ccopt_design -genScriptOnly <script_filename>
```

Runs the "Generate Script" stage, saving the script with the specified filename.

```
ccopt_design -ckSpec
```

You can also use the `-ckSpec` parameter of the command to run the design using the EDI CTS specification file and generate the script that will be passed to CCOpt. This mode is a compatibility layer to enable the use of CCOpt technology with existing EDI CTS specification flows. It enables a translation step that transforms constructs in the EDI CTS specification file into a CCOpt clock tree specification tcl script.

2. Run CCOpt using a generated script

```
ccopt_design -runScript <script_filename>
```

Runs the "Run CCOpt" stage, starting CCOpt and sourcing the specified script.

3. Load the optimized design back into the EDI System

```
ccopt_design -import <azdb_filename>
```

Runs the "Import Design" stage, loading the specified CCOpt database (.azdb) into the EDI System.

For example, to modify the script you would edit the file <script.rbxtcl> between the first and second points listed above.

**Note:** For performing clock tree synthesis, you can choose either the EDI-CTS engine or the CCOpt engine, both with EDI clock specification file. This option is provided in the [setCTSMODE -engine {ck | ccopt}](#) command. When setCTSMODE -engine is set to ck, which is the default value, [clockDesign](#) uses the EDI-CTS commands ([ckSynthesis](#)/[ckECO](#)) to synthesize clock trees according to the EDI clock specification file. However, when setCTSMODE -engine is set to ccopt, the EDI clock specification file is automatically mapped to the Azuro clock specification file and [clockDesign](#) now calls the CCOpt engine to build the balanced clock tree.

For details of [clockDesign](#) options that are not supported by the CCOpt engine, see the [clockDesign](#) command in the *EDI System Text Command Reference* document.

## Native Integration Mode

**⚠** The CCOPT native integration mode is a limited-access feature in this release. It is enabled by a variable specified through the [setLimitedAccessFeature](#) command. To use this feature, contact your Cadence representative to explain your usage requirements, and make sure this feature meets your needs before deploying it widely. To use this limited-access feature, you need a license for the EDS210 product number.

This mode uses CCOpt functionality that has been integrated into the core EDI system. All CCOpt optimization is performed within the EDI system, with no need to start separate binaries or produce control scripts. The native integration mode has an advantage over the scripted integration mode in that it avoids the timing and routing correlation errors that result from performing CCOpt optimization with separate timing and routing engines, and it takes advantage of multi-CPU optimization to significantly reduce the runtime of CCOpt.

The configuration is performed using the [set\\_ccopt\\_property](#) command instead of the [set\\_ccopt\\_mode](#) command with the exception of [set\\_ccopt\\_mode -integration](#) parameter. This parameter is used to select either the scripted or the native mode of integration. By default, the scripted mode is selected. Some [set\\_ccopt\\_mode](#) parameters are translated into the [set\\_ccopt\\_property](#) command automatically - generally setting one global property value except where noted - and, therefore, also apply to native integration.

### The Native CCOpt Flow

The CCOpt configuration in the native flow is performed using the [set\\_ccopt\\_property](#) command, instead of the [set\\_ccopt\\_mode](#) command with the

exception of the `-integration` parameter.

The basic flow for using CCOpt in the native integration mode is as follows:

- Load your placed design in the EDI System
- Run [`create\_ccopt\_clock\_tree\_spec -immediate`](#) to extract CCOpt clock tree spec
- Perform CCOpt configuration using the [`set\_ccopt\_property`](#) command
- Set the native integration mode using the [`set\_ccopt\_mode -integration native`](#) command
- Run CCOpt using the [`ccopt\_design`](#) command to optimize the design

The list of parameters and the corresponding CCOpt properties set using the `set_ccopt_property` command is detailed in the table below.

| Name of <code>set_ccopt_mode</code> Parameter         | Corresponding Properties Set using <code>set_ccopt_property</code> Command                   |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>-cts_buffer_cells</code>                        | <code>buffer_cells</code>                                                                    |
| <code>-cts_inverter_cells</code>                      | <code>buffer_cells</code>                                                                    |
| <code>-cts_clock_gating_cells</code>                  | <code>clock_gating_cells</code>                                                              |
| <code>-cts_logic_cells</code>                         | <code>logic_cells</code>                                                                     |
| <code>-cts_use_inverters</code>                       | <code>use_inverters</code>                                                                   |
| <code>-cts_use_min_max_path_delays</code>             | <code>use_min_max_path_delays</code>                                                         |
| <code>-ccopt_auto_limit_insertion_delay_factor</code> | <code>auto_limit_insertion_delay_factor</code>                                               |
| <code>-cts_target_slew</code>                         | <code>target_max_trans -net_type leaf</code>                                                 |
| <code>-cts_target_nonleaf_slew</code>                 | <code>target_max_trans -net_type trunk</code><br><code>target_max_trans -net_type top</code> |
| <code>-cts_target_skew</code>                         | <code>target_skew</code>                                                                     |
| <code>-top_net_min_fanout</code>                      | <code>routing_top_min_fanout</code>                                                          |
| <code>-cts_opt_priority all</code>                    | <code>advanced_insertion_delay_optimization</code>                                           |

|                                                                                                                                |                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                | false<br>expand_multi_child_regions false<br>low_power_clustering false<br>recluster_to_reduce_power false                                        |
| -cts_opt_priority insertion_delay                                                                                              | advanced_insertion_delay_optimization<br>true<br>expand_multi_child_regions true<br>low_power_clustering false<br>recluster_to_reduce_power true  |
| -cts_opt_priority insertion_power                                                                                              | advanced_insertion_delay_optimization<br>false<br>expand_multi_child_regions false<br>low_power_clustering true<br>recluster_to_reduce_power true |
| -override_modify_clock_latency                                                                                                 | override_modify_clock_latency                                                                                                                     |
| -route_top_top_preferred_layer<br>-route_top_bottom_preferred_layer<br>-route_top_non_default_rule<br>-route_top_shielding_net | Translated into creating a compatibility route type for the native integration.                                                                   |

In the native integration mode, the following parameters of the [ccopt design](#) command are not applicable because there is no script generation in this mode.

- -genScriptOnly - if specified, the software gives an error
- -runScript - if specified, the software gives an error
- -import - if specified, the software gives an error
- -inDir - if specified, the software will ignore the option
- -outDir - if specified, the software will ignore the option

In this mode, you can use the -ckSpec parameter to run the design using the EDI CTS specification file and generate the script that will be passed to CCOpt. Additionally, if you use the optional -genScriptOnly parameter, the translated CCOpt clock tree specification tcl commands will be written out to a file and not be evaluated. Without the -genScriptOnly parameter, the CCOpt clock tree specification commands are immediately evaluated.

The [generate ccopt rc factor](#) command is not available in the native mode. The software will give an error if the command is run in this mode.

## CCOpt Properties and Configuration

In the native integration mode, you can set and retrieve values of various properties using the

[set\\_ccopt\\_property](#) and [get\\_ccopt\\_property](#) commands. The key considerations for setting properties is explained in the "Setting CCOpt Properties" section below.

The properties apply to different types of CCOpt configuration design object types:

- Properties of `clock_tree` objects
- Properties of `skew_group` objects
- Properties of `clock_spine` objects
- Properties of `clock_tree_source_group` objects
- Properties of `pin` objects
- Properties of `net` design objects
- Global options

For more information on how to get a complete list of all properties and detailed information for each property, see the "Getting a List of Properties and their Detailed Descriptions" section.

### **Setting CCOpt Properties**

This section details out some key considerations for setting CCOpt properties through the use of some examples. Before you learn how to set the CCOpt properties, you should know that there are some properties that cannot be set by you because these are read-only properties . This means that you can obtain information about these properties by using the [get\\_ccopt\\_property](#) command, but you cannot change that information.

For example, the `generated_by_sinks` property is read only and exists to let the user know which pins are used to trace the master clock tree paths to the generated clock tree.

However, there are writable properties that can be modified. These properties let you configure different aspects of CCOpt behavior. The key considerations for setting CCOpt properties are detailed below.

#### **Specifying the property name, value, object type, and object pattern**

The property name, value, object type, and object pattern need to be specified while setting CCOpt properties. For example:

- `create_ccopt_clock_tree_spec -name ct1 -source clk`
- `set_ccopt_property use_inverters -clock_tree ct1 true`
- `create_ccopt_skew_group -name sg1 -sources clk ...`
- `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`

Where:

- `set_ccopt_property [-<obj_type> <obj_pattern>]* <prop_name> <value>`
- `unset_ccopt_property [-<obj_type> <obj_pattern>]* <prop_name>`
- `create_ccopt_*`

where \* = `skew_group`, `clock_tree`, `clock_tree_source_group`, `clock_spine`

The `<prop_name>` and `<value>` clauses are positional, and must appear in that order. But the `<obj_name> <obj_pattern>` clauses can appear before, in the middle, or after the `<prop_name>` `<value>` clauses. This means that both the statements below are valid.

- `set_ccopt_property insertion_delay -skew_group sg1 1.5`
- `set_ccopt_property -skew_group sg1 insertion_delay 1.5`

#### Using Wildcards

It is possible to wildcard any of the `<value>` fields in [get\\_ccopt\\_property](#) and [set\\_ccopt\\_property](#) commands. 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 its `use_inverters` property set to `true`. However, if you had specified `set_ccopt_property -use_inverters true`, the new clock tree would get this property set to `true` when it was created. Leaving out a `-keyword value` pair means the `set_ccopt_property` command will affect all the relevant objects, even if they have not been created yet.

#### Using Properties with Switches

Properties are set using specific switches. If you leave out a switch for a particular property, there will be an implicit "forall" on that type value. The example below specifies that clock tree synthesis should use inverters when balancing clock tree `ct1`:

```
set_ccopt_property use_inverters -clock_tree ct1 true
```

The example below does not specify any switch:

```
set_ccopt_property -use_inverters true
```

So, this command will be equivalent to specifying the following:

```
For each ct, [get_ccopt_clock_trees *] {set_ccopt_property -use_inverters -clock_tree $ct true}
```

However, there are some exceptions to this, which are designed to catch user intent. The main exception are properties that have `-delay_corner` and `-early/-late` switches. When you omit these switches, the default behavior is that the property is not set across all delay corners but instead it will be set just for the primary corner. In the following example, the leaf-level properties are maintained internally and you can set one or more property at a time by simply specifying the extra information.

|                | <b>Early</b> | <b>Late</b> |
|----------------|--------------|-------------|
| <b>SetupDC</b> | Ignore       | Auto        |
| <b>Hold1DC</b> | Ignore       | Ignore      |

|                |        |        |
|----------------|--------|--------|
| <b>Hold2DC</b> | Ignore | Ignore |
|----------------|--------|--------|

If you specify `set_ccopt_property target_skew 0.05`, you will get the following:

|                | Early  | Late   |
|----------------|--------|--------|
| <b>SetupDC</b> | Ignore | 50ps   |
| <b>Hold1DC</b> | Ignore | Ignore |
| <b>Hold2DC</b> | Ignore | Ignore |

If you specify `set_ccopt_property target_skew -delay_corner Hold1DC 0.10`, you will get the following:

|                | Early  | Late   |
|----------------|--------|--------|
| <b>SetupDC</b> | Ignore | 50ps   |
| <b>Hold1DC</b> | 100ps  | 100ps  |
| <b>Hold2DC</b> | Ignore | Ignore |

**Note:** In the above example, the specification of “Hold1DC” matches two cells in the table, so both cells are updated.

If you specify, `set_ccopt_property -target_skew -early 0.075`, you will get the following:

|                | Early | Late   |
|----------------|-------|--------|
| <b>SetupDC</b> | 75ps  | 50ps   |
| <b>Hold1DC</b> | 75ps  | 100ps  |
| <b>Hold2DC</b> | 75ps  | Ignore |

### Getting Properties from Tcl

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 you ask for a range of values by leaving out some or all of the “–key\_name key\_value” switches or by providing a pattern for a key\_value, then in this case, if all the cells specified have the same value, then that value will be returned.

For example, using the following, if you specified, `get_ccopt_property -target_skew -early`, then three cells are selected, all of which have the value 0.075, and the value returned is 0.075.

|  | Early | Late |
|--|-------|------|
|--|-------|------|

|                |      |        |
|----------------|------|--------|
| <b>SetupDC</b> | 75ps | 50ps   |
| <b>Hold1DC</b> | 75ps | 100ps  |
| <b>Hold2DC</b> | 75ps | Ignore |

It is possible to return a number of cells at once by using the `-list` parameter of the `get_ccopt_property` command, which will cause all leaf cells that match to be returned as a list of `<index, value>` pairs. For example.

`get_ccopt_property -target_skew -list -delay_corner SetupDC` will return the following:

```
{ \
{ -delay_corner SetupDC -early -value 0.075 } \
{ -delay_corner SetupDC -late -value 0.05 } \
}
```

The syntax here is

```
{"( "-(<key_name><key_value>|<valueless_key>))+ "-value" <value> "}" )+ \
"}
```

In this, `<valueless_key>` can be `rise`, `fall`, `early`, or `late`.

Each possible value that can be set will be listed only once, with a fully qualified list of `key_names`. For example, the above table values will be listed as:

```
{ \
{ -delay_corner SetupDC -early -value 0.075 } \
{ -delay_corner SetupDC - late -value 0.05 } \
{ -delay_corner Hold1DC - early -value 0.075 } \
{ -delay_corner Hold1DC - late -value 0.1 } \
{ -delay_corner Hold2DC - early -value 0.075 } \
{ -delay_corner Hold2DC - late -value ignore } \
}
```

However, if only the `-list` parameter is specified, `get_ccopt_property -target_skew -list`, you will get a full list of the property values.

**Note:** If you specify the `-list` parameter, then you get the full list, even if many or all of the values are the same. When the `-list` parameter is not specified, the [set\\_ccopt\\_property](#) command default values for `-pin`, `-delay_corner`, and `-early/-late` also apply to the [get\\_ccopt\\_property](#) command. However, when the `-list` parameter is provided, the defaults are disabled.

#### Getting a List of Properties and their Detailed Descriptions

The list of all available individual CCOpt properties and their detailed information is available within the EDI System. For a list of all the available CCOpt properties - in alphabetical order - with brief descriptions of each property, run the following command:

```
get_ccopt_property * -help
```

The software displays the following information:

```
advanced_insertion_delay_optimization # Enables a number of CTS routines that are designed to reduce insertion delay.  
  
auto_detect_and_tie_lockup_latches # Automatically detect lockup latches and tie enable pin to clock pin of driving flop.  
  
auto_limit_insertion_delay_factor # Specifies the amount by which CCopt can increase the insertion delay of a clock tree.
```

.....

For detailed help on a specific property, run the following command:

```
get_ccopt_property <property name> -help
```

**Examples:**

- The following command provides a detailed description of the skew\_groups\_sink property:

```
get_ccopt_property skew_groups_sink -help
```

The software displays the following information:

The list of skew groups for which this pin is a sink.

Default: From CTS setup

Valid values: list skew\_groups

Applies to: pin

- The following command provides a detailed description of the use\_inverters property:

```
get_ccopt_property use_inverters -help
```

The software displays the following information:

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.

Default: auto

Valid values: auto true false

Applies to: Global, clock\_tree

## Key-Value Pairs

Following are the key-value pairs needed for setting properties:

- `-skew_group <sg>`: Objects created by the [create\\_ccopt\\_skew\\_group](#) command
- `-delay_corner <dc>`: The EDI System delay corners
- `-clock_spine <cs>`: `clock_spine` objects
- `-clock_tree_source_group <c>` :Objects created by [create\\_ccopt\\_clock\\_tree\\_source\\_group](#) command
- `-clock_tree <ct>`:Objects created by the [create\\_ccopt\\_clock\\_tree](#) command
- `-pin <p>` :The EDI System's pin / instTerms, referenced by name and not by pointer
- `-lib_pin <lp>`: The EDI System's pins on library cells, referenced by name not by pointer
- `-net <n>` :The EDI System's nets, referenced by name and not by pointer
- `-early | -late`: Selects half of a delay corner. These are mutually exclusive.
- `-rise | -fall`: Selects an edge. Usually the location is obvious. These are mutually exclusive.
- `-net_type <nt>`: Selects one each of trunk, top, and leaf nets.

**Note:** The `-early`,`-late`,`-rise`, and `-fall` clauses do not have values. This is to align with the same usage in various Common Timing Engine (CTE) commands.

## Defaults for Key Value Pairs

In general, each CCOpt property has the following:

- A number of values, which are held in a  $p \times q \times \dots$  table
- A list of types, which index the table

Leaving out one or more of the types means the set command affects a slice of the table, corresponding to a “forall” operation on the types that were missing. However, there are some exceptions to this rule:

- If you do not specify `-pin`, the software sets the property on all pins currently in the union of all defined clock trees.
- If you do not specify a `delay_corner`, the software selects the `delay_corner` that is the whole of the primary half-corner.
- If you do not specify `early` or `late`, the software selects `late` by default.

These exceptions are there to capture your intent in cases where they you have not been totally specific.

The two key example cases are:

### Example 1:

- `set_ccopt_property target_pin_max_trans 0.1 ;# per-pin max trans setting`
- `unset_ccopt_property insertion_delay ;# remove all pin insertion delays`

In both these cases, it's clear that only skew group sinks are intended.

### Example 2:

- `set_ccopt_property insertion_delay -pin p 0.1 ;# push sink up in clock tree`
- `set_ccopt_property target_skew -skew_group sg1 0.05 ;# tighten skew target`

In both these cases it is clear that the intention is to just set the property on the primary half-corner and not on all delay corners. Also, adding it to every corner will lead to an impossible balancing problem in the case of insertion delay.

## CCOpt Clock Tree Specification

This section describes how to define clock trees in CCOpt and the different methods you can use to control CTS.

### Clock Trees and Skew Groups

Before CCOpt can perform clock tree synthesis, it needs to know the following:

- The clock trees that are present in the design
- The sinks that need to be balanced together
- Targets for skew, slew, and insertion delay

Therefore, you need to do the following:

- Define clock trees to provide information about the physical connectivity between cells and sinks
- Define skew groups to specify the sinks that should be balanced together
- Define CTS constraints to specify skew, slew, and insertion delay targets

For many designs, CCOpt can automatically obtain all the required information by examining your design and the associated SDC constraints. For more information, see the "Automatic Extraction of Clock Trees" section.

A wide variety of balancing constraints can be applied to a design by using different skew group and clock tree configurations, including balancing between clock trees and balancing sinks with non-sink pins.

The same sink may belong to multiple skew groups. In this case, CCOpt attempts to balance the sink against the other members of all the skew groups, so that the sink meets the following:

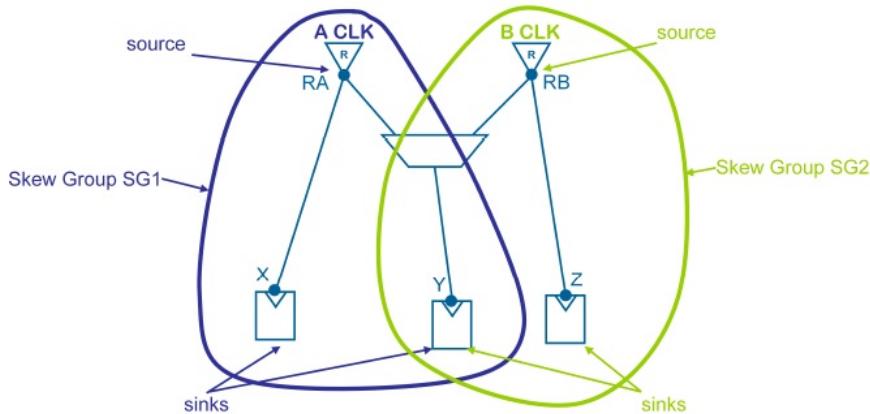
- The skew constraints of all the skew groups to which it belongs
- The maximum of the minimum insertion delays for all the skew groups to which it belongs
- The most aggressive of the slew targets for all the skew groups to which it belongs

Where sinks belong to multiple skew groups, there exists the possibility that constraints may conflict. In this case, CCOpt will try and resolve the conflicting constraints. For more information, see the "Defining Skew Groups" section.

The following examples show how clock trees and skew groups can be used to control CTS.

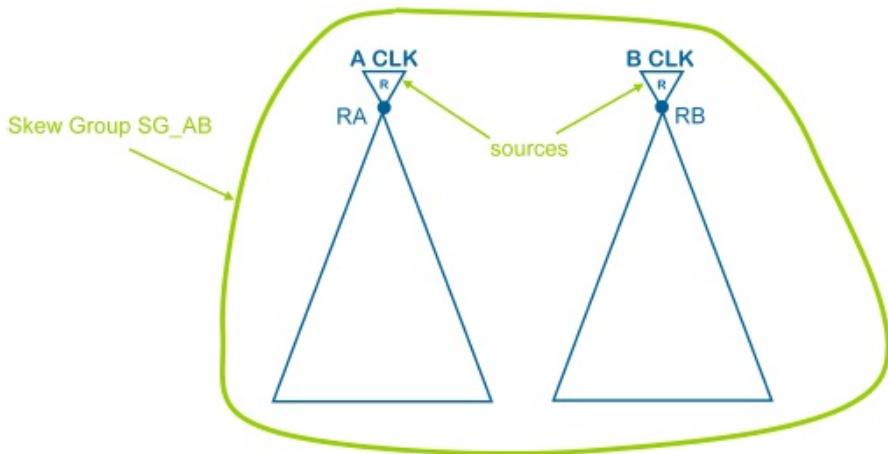
**Example1: Balancing Shared Sinks**

This example shows two clock trees, with a single flop shared between them. Two overlapping skew groups are used to balance these two separate clock trees to ensure that the shared flop is balanced against the remaining flops in both clock trees. The sinks X and Y belong to the same skew group (SG1) and are balanced together. Similarly, sinks Y and Z belong to the same skew group (SG2) and are also balanced together. However, there is no requirement to balance sinks X and Z together.



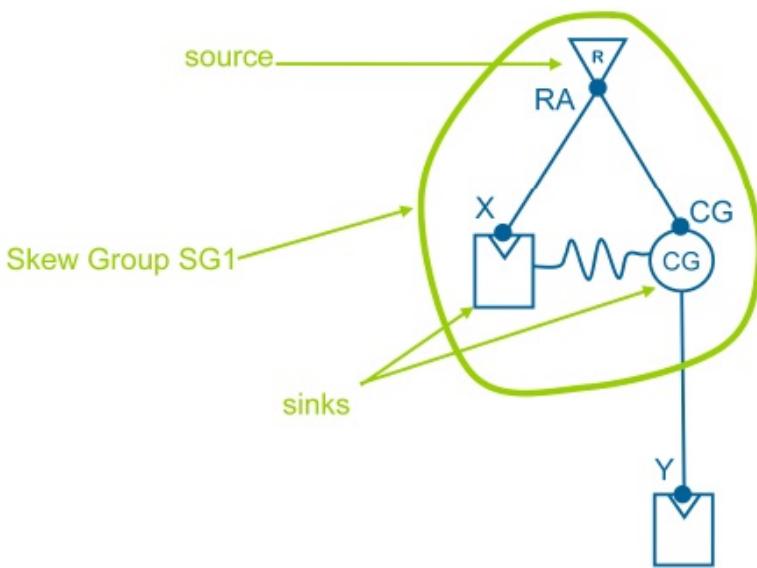
**Example2: Balancing Independent Clock Trees**

This example shows two independent clock trees. A single skew group is used to balance the sinks of both clock trees together. Because all sinks belong to the same skew group (SG\_AB), CCOpt balances all sinks together, even though they are part of separate clock trees.



#### Example3: Balancing Flops with Clock Trees

This example shows a flop (X) driving a clock gate enable (CG). In this case, to meet timing, flop X must be balanced against the clock gate rather than the sink flop (Y). To achieve this, create a skew group that contains the clock gate and its enable flop, but not the sink flop.



Before CCOpt can optimize, report on, or display clock trees, you need to define the clock trees and skew groups in your design. There are three ways to do this:

- **Automatic:** Use the [`create\_ccopt\_clock\_tree\_spec`](#) command to automatically analyze your SDC timing constraints and netlist to define the clock trees and skew groups in your design. Use this approach when you have a complete SDC file and relatively uncomplicated clock trees in your design.

- **Semi-Automatic:** Use the [`create\_ccopt\_clock\_tree\_spec -file <script filename>`](#) command to automatically analyze your SDC timing constraints and netlist and produce a script that will define the clock trees. You can then edit this script to make the required corrections. Use this approach when you have an SDC file that needs some correction, or when you have complex clock trees.
- **Manual:** Define the clock trees yourself using the [`create\_ccopt\_clock\_tree`](#) and [`create\_ccopt\_generated\_clock\_tree`](#) commands, along with other CCOpt clock tree definition commands, as required. Use this approach if you want to define clock trees and skew groups yourself without using the [`create\_ccopt\_clock\_tree\_spec`](#) command.

However you choose to define your clock trees - automatically, semi-automatically, or manually - CCOpt uses the following commands to define your clock trees:

- [`create\_ccopt\_clock\_tree`](#)
- [`create\_ccopt\_generated\_clock\_tree`](#)
- [`set\_ccopt\_property sink\_type stop -pin <pin name>`](#)
- [`set\_ccopt\_property sink\_type ignore -pin <pin name>`](#)
- [`set\_ccopt\_property insertion\_delay -pin`](#)
- [`create\_ccopt\_skew\_group`](#)

**i** With automatic clock tree extraction, CCOpt determines which of these commands to run by examining the netlist and the SDC file. With manual clock tree extraction, you specify the exact commands you want to run.

### Automatic Extraction of Clock Trees

CCOpt has the ability to look at your netlist and SDC clock definitions and automatically produce corresponding clock trees and skew groups. The [`create\_ccopt\_clock\_tree\_spec`](#) command performs this automatic configuration of clock trees and skew groups.

In many designs, this produces exactly the configuration you require. However, in some cases, you may need to hand-tune this process. Hand-tuning may be required when:

- The desired correspondence between timing clocks and clock trees is not simple
- You would like advanced control over how CCOpt balances interrelated clock domains
- The SDC file is inaccurate or incomplete

To hand-tune clock tree extraction, run the `create_ccopt_clock_tree_spec` command with the `-file` parameter:

[`create\_ccopt\_clock\_tree\_spec -file auto\_clock\_trees.tcl`](#)

This outputs a tcl script to the specified file containing the commands that would have been run, but does not actually run those commands. You can then edit the configuration script by adding,

removing, or redefining the clock trees, skew groups, and constraints, as required. For more information about exactly how [create\\_ccopt\\_clock\\_tree\\_spec](#) analyzes your design, see the "How Automatic Clock Tree Extraction Works" section.

## Defining Clock Trees

The basic process for clock tree definition is as follows:

- **Configuring Stop and Ignore Pins:** Configure any stop or ignore pins to tell CCOpt about any special places where you want clock tree extraction to stop. This means you specify the non-sink pins that you want to treat as sinks.
- **Defining Clock Trees:** Use the [create\\_ccopt\\_clock\\_tree](#) command to define your clock trees. CCOpt then traces through your netlist from the specified source until sinks are encountered.
- **Defining Generated Clock Trees:** Use the [create\\_ccopt\\_generated\\_clock\\_tree](#) command to define any generated clock trees. Again, CCOpt then traces through your netlist from the specified source until sinks are encountered.
- **Setting Pin Insertion Delays:** Configure insertion delays and CTS constraints.

This can either be done automatically by CCOpt, using the [create\\_ccopt\\_clock\\_tree\\_spec](#) command, or manually. These points are detailed in subsequent sections.

### Configuring Stop and Ignore Pins

Stop and ignore pins tell CCOpt to treat that pin as a clock sink, even though it would not normally identify that pin as a clock sink.

The [create\\_ccopt\\_clock\\_tree](#) and [create\\_ccopt\\_generated\\_clock\\_tree](#) commands, which trace through the netlist until clock sinks are encountered, also stop tracing when stop or ignore pins are encountered. Defining a pin as a stop pin tells CCOpt to treat that pin as a clock tree sink. CCOpt then balances the pin against all other sinks in the clock tree. For example:

```
set_ccopt_property sink_type stop -pin my_design/mod1/mod2/cellX/A
```

Defining a pin as an ignore pin tells CCOpt to treat that pin as a clock tree sink - just like a stop pin. However, CCOpt does not balance ignore pins against other pins in the clock tree. For example:

```
set_ccopt_property sink_type ignore -pin  
my_design/mod1/mod2/cellX/A
```

CCOpt minimizes the insertion delay of ignore pins.

### Defining Clock Trees

The [create\\_ccopt\\_clock\\_tree](#) command provides CCOpt with information about a clock tree in your design. The syntax of the [create\\_ccopt\\_clock\\_tree](#) command is as follows:

```
create_ccopt_clock_tree -name <n> -source <s>  
[-stop_at_sdc_clock_roots] [-no_skew_group] [-skew_group <sg>]
```

When you define a clock tree, CCOpt traces the netlist from the specified source pin, adding the nets and cells it encounters to the clock tree. CCOpt continues tracing the clock tree as it branches until it encounters any of the following:

- A clock pin of a flop, latch, or large macro
- A stop pin
- An ignore pin
- A cell with multiple outputs
- A cell that drives a net that has other drivers

The `-stop_at_sdc_clock_roots` parameter directs CCOpt to stop propagating the clock tree at points where SDC-generated clocks are defined. Normally, CCOpt subsumes these clock definitions, if possible, into the parent clock tree, which gives better results because CCOpt can physically move and size internal clock tree cells but not generated clock tree roots.

You also need to define the skew groups to which each clock tree's sinks belong. This determines which sinks should be balanced together. The [create\\_ccopt\\_clock\\_tree](#) command offers the following options for assigning sinks to skew groups:

- The `-skew_group` parameter lets you specify an existing skew group to which sinks are added. The clock tree will be added to the list of sources for the given skew group.
- The `-no_skew_group` parameter indicates that sinks should not be assigned to skew groups, leaving that task for a later time using the [create\\_ccopt\\_skew\\_group](#) command.
- By default, that is when either `-skew_group` or `-no_skew_group` is specified, CCOpt creates a new skew group for each clock tree and adds the sinks to that skew group automatically.

For more information about skew groups and how to assign sinks to skew groups, see the "Defining Skew Groups" section.

#### Defining Generated Clock Trees

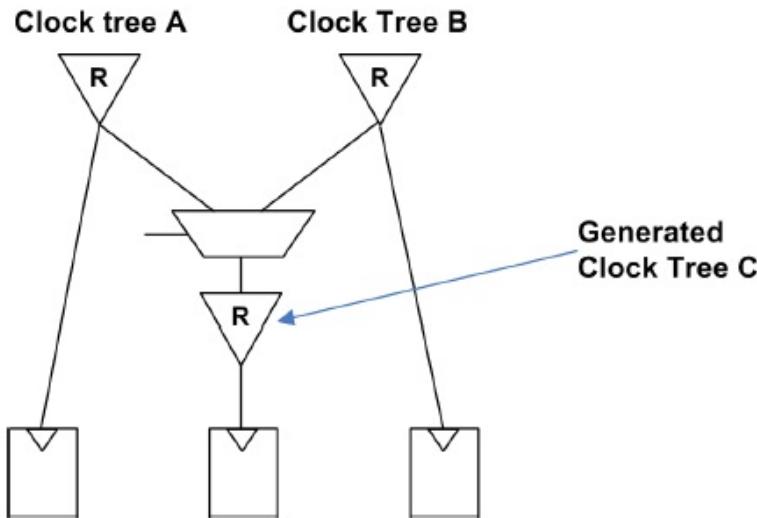
Many designs contain a clock that is generated from another clock, for example, by dividing down a fast clock. To specify a generated clock in CCOpt, use the [create\\_ccopt\\_generated\\_clock\\_tree](#) command.

The syntax of the [create\\_ccopt\\_generated\\_clock\\_tree](#) command is as follows:

```
[ -help ]  
[ -generated_by <pins> ]  
[ -name <clockname> ]  
[ -parents <parents> ]  
-source <pin>  
[-stop_at_sdc_clock_roots]
```

The `-parents` parameter specifies the clock tree or clock trees that is or are the parent of this generated clock tree. The parent clock tree must be defined before this tree. If the parent is unambiguous, the `-parents` parameter may be omitted. A generated clock tree may have multiple parent clock trees. This is shown in the figure below.

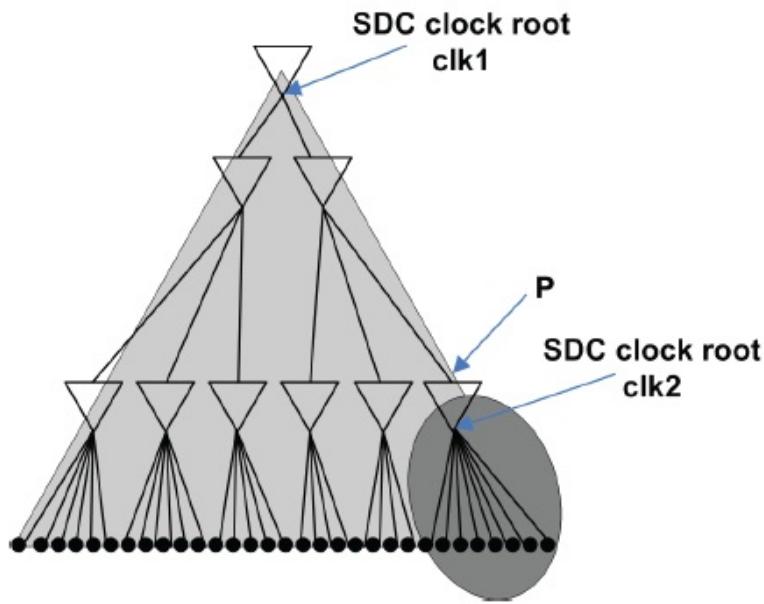
### Generated Clock Tree with Multiple Parents



In the above example, the generated clock tree C does not have a single parent clock tree. Clock tree C is a child of both clock trees A and B. This would be specified using `-parents {A B}`. The `-generated_by` parameter lets you identify the sink pins in the parent clock trees for which the delay between the parent and generated clock trees should be considered. Pins provided as generator sinks must either be existing clock tree sinks or input pins of existing clock tree nodes. If the `-generated_by` parameter is not specified, all possible generator inputs in the specified clock tree will be used.

There can be any number of generated clock trees under a parent clock tree, and generated clock trees can themselves be the parents of other generated clock trees. Note that if there are no state-holding elements between the parent clock tree and the generated tree, then the [`create\_ccopt\_clock\_tree`](#) command for the parent may grab the sinks of what you will later define to be the generated tree. If you want to stop this from happening, either set the input to the generator to be a stop pin, or use the `stop_at_sdc_clock_roots` parameter, if the generated clock tree has a corresponding SDC clock. The following diagram shows an example of a generated clock tree without a state- holding element between them.

### Extracting Generated Clocks Correctly



If you have a similar situation in your design where a generated clock, `clk2`, is derived from its parent `clk1`, without a state-holding element between them, you have the following options:

- Ignore the generated clock. If it is acceptable for the two SDC clocks to be carried by the same clock tree, there is no need to tell CCOpt about the generated clock. In this case, use a single clock tree `clk1`, which contains all the sinks belonging to `clk2`.
- Configure pin **P** to be a stop pin before defining the clock tree corresponding to SDC clock `clk1`.
- Use the `-stop_at_sdc_clock_roots` parameter when defining the clock tree corresponding to SDC clock `clk1`.

Sinks of generated clock trees are balanced against the other sinks of the parent clock, and against other generated clocks that share a common parent.

#### Setting Pin Insertion Delays

If you need several balanced pins high in the clock tree, then consider using either the `insertion_delay` CCOpt property, or using skew groups. For more information, see the "Defining Skew Groups" section. The `insertion_delay` CCOpt property specifies firstly, that the given pin is a stop pin, and secondly, that there should be an insertion delay offset applied to this pin. The format of this command is as follows:

```
set_ccopt_property insertion_delay -pin <pin> <delay>
```

where:

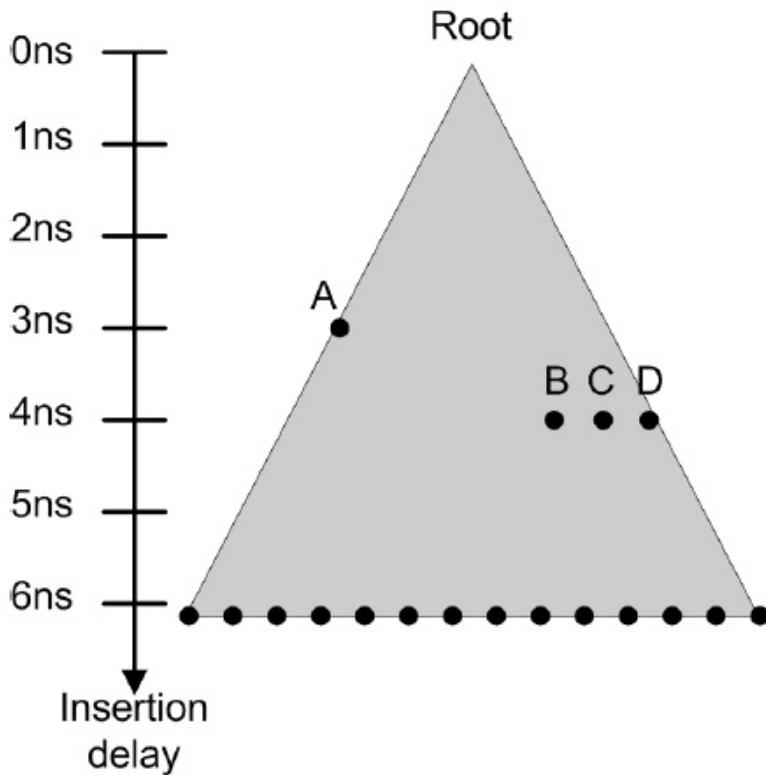
`pin` - applies the command to the specified pin

`delay` - specifies the delay for the pin in library units

You can use the `insertion_delay` property before or after defining a clock tree, although the stop pin behavior is not applicable if it is used after defining a clock tree. The example below shows the effects of setting pin insertion delays for four pins, **A**, **B**, **C**, and **D** in a clock tree, with library units of nanoseconds.

```
set_ccopt_property insertion_delay -pin A 3
set_ccopt_property insertion_delay -pin B 2
set_ccopt_property insertion_delay -pin C 2
set_ccopt_property insertion_delay -pin D 2
```

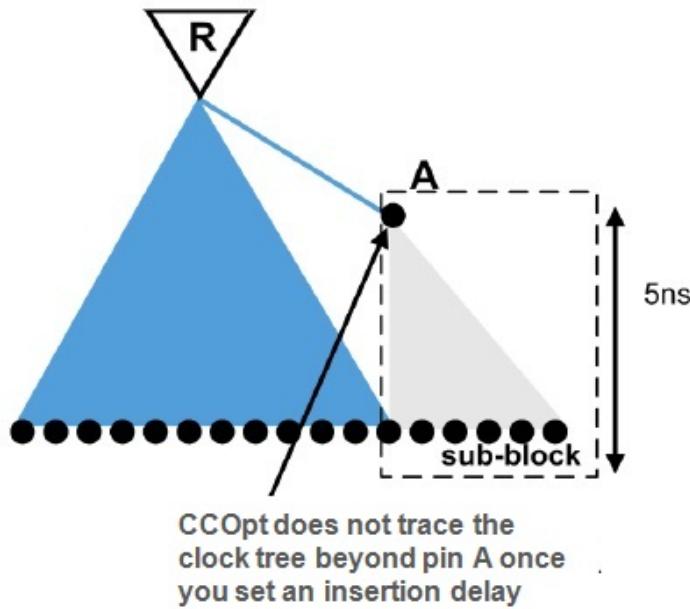
#### Effects of `insertion_delay` CCOpt Property



In this example, you can see that pin **A** has been placed 3ns before the other sinks, and pins **B**, **C**, and **D** have been placed 2ns before the other sinks. Note that because **B**, **C**, and **D** were all specified to have the same insertion delay, they have been balanced against each other. An example scenario of when you might need to set the insertion delay of a pin is if you have a sub-block with an optimized clock tree. In this case, you do not need CCOpt to trace the clock tree into the sub-block because you do not want the clock tree to be changed in the sub-block. However, since you know the timing characteristics of the sub-block, you know that the clock signal needs to reach the sub-block 5ns earlier than the other sinks.

#### Effects of `insertion_delay` CCOpt Property

```
set_pin_insertion_delay pin:A 5
```



Use the insertion\_delay property to set an insertion delay of 5ns on the clock input pin to the sub-block:

```
set_ccopt_property insertion_delay -pin A 5
```

#### Defining Skew Groups

Defining skew groups is the main method of defining how sinks will be balanced in CCOpt. Skew groups allow balancing of clock trees and balancing across clock trees. A skew group is defined as a set of sources, sinks, and ignore pins. Any input pin in the clock tree may be assigned to be an active skew group sink, and a pin may be an active sink in more than one skew group at any time. CCOpt will attempt to balance a pin according to the constraints specified in all the skew groups for which it is an active pin. By default, CCOpt balances clock trees. It does this by virtue of an automatic skew group that is created when you run the [create\\_ccopt\\_clock\\_tree](#) command without specifying the `-skew_group` and `-no_skew_group` parameters.

The sources of the skew group define the start point of paths in that skew group – it is where the insertion delay for that skew group (the longest path) is measured. The sinks of the skew group define the end points in that skew group. The ignore pins of a skew group prevent certain paths from being timed. The longest path through the skew group is the longest path traveling from any of the sources, through the clock tree structure, to any sink, stopping at any pin marked as an ignore pin.

The source of the skew group is a Tcl list of clock trees or pins. When using a list of non-generated clock trees, the paths of that skew group are considered to include the source latency of the clock trees in question, defined as the `source_latency` property on the `clock_tree` design object. The source latency of a skew group is the first cell arrival time when a skew group is defined with a non-generated clock tree source.

To define a skew group, use the [create\\_ccopt\\_skew\\_group](#) command. The syntax of the command is as follows:

```
[-help]
[-constraints <cts ccopt_initial ccopt>]
[-from_clock <clock_name>]
[-from_constraint_modes <constraint_mode_names>]
[-from_delay_corners <delay_corner_names>]
-name <skew_group_name>
[-rank <rank>]
[-target_insertion_delay <value>]
[-target_skew <value>]
[-sinks <pins> | -shared_sinks <pins> | -exclusive_sinks <pins> | -
auto_sinks | -filtered_auto_sinks <string> | -balance_skew_groups
<skew_groups>]
[-sources <pins> | -balance_skew_groups <skew_groups>]
```

To specify the sinks of the skew group:

- Use the `-sinks` parameter to specify a list of input pins in the design that CCOpt should consider as members of the skew group. If you use the `-sinks` parameter to specify sinks in this manner, you must also use the `-rank` parameter to specify the `exclusive_sinks_rank` of the skew group. For more information about skew group rank, see the "Skew Group Rank and Active Sinks" section.
- Use the `-auto_sinks` parameter to specify that you want to add all clock tree sinks reachable from the specified sources. The skew group is created with `exclusive_sinks_rank` of 0. This means that if any of these sinks already belong to another skew group, they will be members of both the original skew group and the new skew group, and they will be added as shared sinks.
- Use the `-filtered_auto_sinks` parameter to specify that you want to add those clock tree sinks that are both reachable from the specified sources and appear in the supplied list. This means that CCOpt will generate the same list of reachable pins as it would have done if `-auto_sinks` had been specified, but then it will use the supplied pin list as a filter. As with the `-auto_sinks` parameter, the skew group is created with an `exclusive_sinks_rank` of 0, resulting in shared sinks.
- Use the `-exclusive_sinks` parameter to add sinks to the new skew group and remove them from any skew groups to which they already belong. The skew group is created with `exclusive_sinks_rank` of 1 higher than the current highest `exclusive_sinks_rank` value, effectively removing the pins from all skew\_groups that they currently belong to.
- Use the `-shared_sinks` parameter to add sinks to the new skew group and let them remain members of any skew groups to which they already belong. This means that the skew group is created with `exclusive_sinks_rank` of 0.

For more information about skew group rank, see the "Skew Group Rank and Active Sinks" section.

The [create\\_ccopt\\_skew\\_group](#) command creates a new design object of type `skew_group` with the given name. The properties of the skew group may be manipulated via the design object mechanism.

The [delete\\_ccopt\\_skew\\_groups](#) command is used to remove a skew group.

The [modify\\_ccopt\\_skew\\_group](#) command is used to easily add and remove ignore or sink pins from a skew group.

The `-constraints` parameter lets you choose how the skew group affects optimization:

- If set to `ccopt`, the skew group constrains CCOpt - CCOpt will be restricted to skewing within this constraint.
- If set to `ccopt_initial` (the default) or `cts`, the skew group constrains the initial CCOpt balancing solution but CCOpt is allowed to skew these sinks arbitrarily, if necessary, for timing.

The [report\\_ccopt\\_skew\\_groups](#) command displays information about skew and insertion delays in skew groups.

### Skew Group Rank and Active Sinks

Every skew group is assigned a rank, accessible through the `exclusive_sinks_rank` property:

- Shared skew groups always have an `exclusive_sinks_rank` value of zero.
- Exclusive skew groups always have an `exclusive_sinks_rank` value greater than zero. When an exclusive skew group is created, CCOpt assigns that exclusive skew group a rank one greater than the highest existing skew group.

The rank of a skew group determines whether a member pin is an active sink in that skew group or not. A pin is only an active sink in the skew group or groups with the highest rank, out of all those 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 pin:top -shared_sinks [get_pins */D]
create_ccopt_skew_group -name SG2 -sources pin:top
-exclusive_sinks [get_pins *XYZ*/D]
create_ccopt_skew_group -name SG3 -sources pin:top
-exclusive_sinks [get_pins *XYZ_01*/D]
```

After running the first command, a single skew group **SG1** is created. This is a shared skew group, so it has an `exclusive_sinks_rank` value of zero. All **D** pins in the design are members of this skew group. In addition, all **D** pins are active sinks in skew group **SG1** - because **SG1** is the highest ranked skew group so far – even though it has a rank of 0.

The second command defines an exclusive skew group **SG2**. This is given an `exclusive_sinks_rank` value of 1, which is one higher than the current highest rank. Sinks 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. Sinks that don't match this pattern remain active pins in **SG1**.

The third command defines another exclusive skew group, **SG3**. This is given an

exclusive\_sinks\_rank value of 2, which is one higher than the current highest rank. Sinks that match the pattern \*XYZ\_01\*/D - which must also be members of **SG2**, because those sinks would also match the pattern \*XYZ\*/D - are now members of **SG1**, **SG2**, and **SG3**. However, they are only active sinks in **SG3**, which is the highest ranked parent skew group.

Sinks that matched the pattern \*XYZ\*/D but not \*XYZ\_01\*/D are members of **SG1** and **SG2** but only active sinks of **SG2**. Sinks that don't match the pattern \*XYZ\*/D are members of **SG1** and active sinks in **SG1**.

### How Automatic Clock Tree Extraction Works

Automatic CCOpt clock tree spec generation, performed by the [create\\_ccopt\\_clock\\_tree\\_spec](#) command, works by analyzing the incoming netlist and its associated SDC file to determine the clock tree structures and constraints in your design, then defining those clock tree structures and constraints in CCOpt.

CCOpt clock tree spec generation begins by looking for [create\\_clock](#) commands in your SDC file. For example:

```
create_clock -name clk1 -period 68 -waveform {0 34} root_pin
```

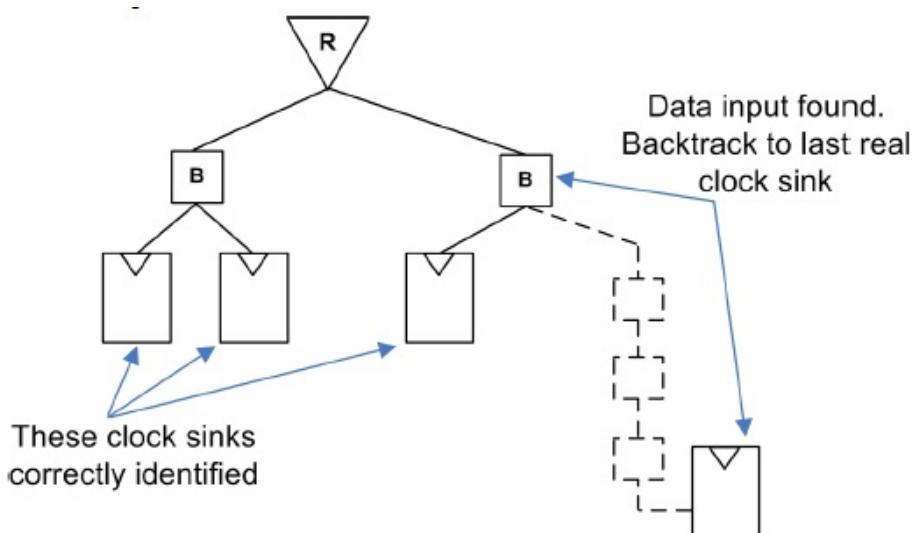
CCOpt creates one clock tree for every SDC clock that is not completely covered by another SDC clock. See below for more information about how to use `create_ccopt_clock_tree_spec -keep_all_sdc_clocks` to create a clock tree for every SDC clock even if a clock is wholly covered by another.

CCOpt then traces from the root pin, adding cells and nets to the clock tree as it progresses. If CCOpt finds a clock pin of a flop, latch, or large macro, it adds the pin to the clock tree and stops tracing that branch of the clock tree. If CCOpt finds a stop or an ignore pin, it adds the pin to the clock tree and stops tracing that branch of the clock tree.

If CCOpt traces a clock tree to a datapath logic input pin, it assumes a mistake has been made. CCOpt then backtracks through the clock tree until it finds a net that drives a clock sink, removes the mistakenly added portion of the clock tree, and adds an implicit ignore pin to the point where the mistake was made.

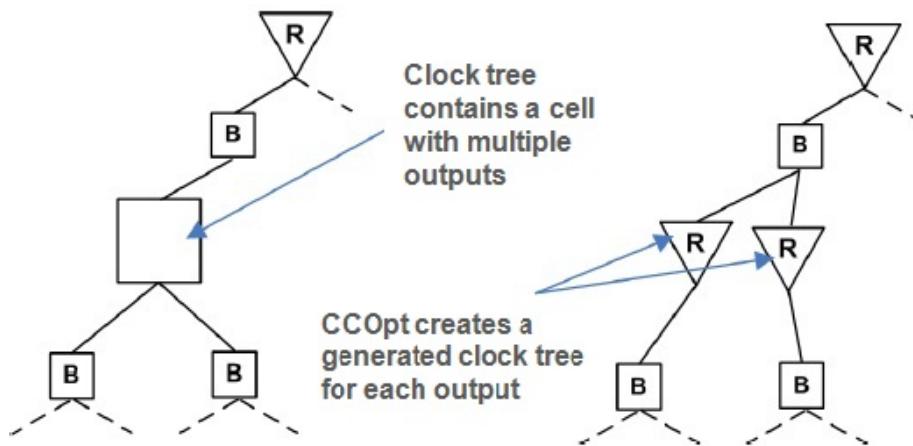
An implicit ignore pin is a pin that will be ignored for balancing purposes, but has not been explicitly requested. If you want the pin to be treated differently, or if you want the **D** pin of a flop to be a sink in a clock tree, you can explicitly mark such pins as stop or ignore pins with the appropriate commands. In this case, they will remain in the clock tree and be balanced as requested. The following figure shows backtracking during automatic extraction.

#### Backtracking During Automatic Extraction



If CCOpt finds a cell in the clock tree with multiple outputs, it automatically creates generated clock trees for each of the cell's outputs as shown below.

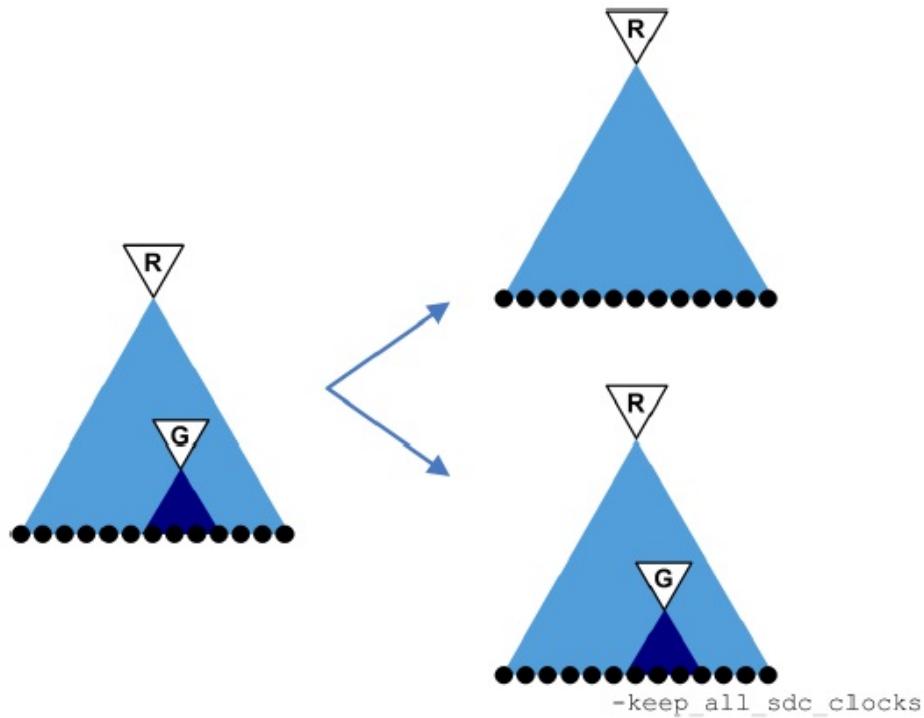
#### Automatic Extraction and Multiple Outputs



CCOpt will also create a generated clock tree when it encounters a clock divider. If CCOpt encounters a generated clock tree that is completely contained within another clock tree, with the same characteristics such as period, waveform, and so on, then CCOpt realizes that the generated clock tree is redundant and merges the generated clock tree into its parent clock tree. The `-keep_all_sdc_clocks` parameter of the [create\\_ccopt\\_clock\\_tree\\_spec](#) command lets you specify that CCOpt should not merge a generated clock tree even if it is redundant because it is part of a larger clock tree. This is not normally necessary, but it can make the visualizations and reports easier to understand. However, you should note that using `-keep_all_sdc_clocks` parameter can cause a reduction in quality of results or QoR compared to not using `-keep_all_sdc_clocks` because CCOpt will not physically move or size the cells marked as clock roots. It is therefore not recommended to specify this parameter, unless absolutely necessary.

Consider the following example, which shows a clock tree with root **R** that contains a generated clock tree, **G**, which is redundant.

**Keeping All SDC Clocks**



When [create\\_ccopt\\_clock\\_tree\\_spec](#) command is run without specifying the `-keep_all_sdc_clocks` parameter, the generated clock tree is merged with its parent clock tree. However, when this parameter is specified, the generated clock is retained.

The following table shows how SDC commands map to CCOpt clock tree definition commands.

| SDC Commands                                                                                                                              | CCOpt ( <a href="#"><u>create_clock_tree_spec</u></a> )                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre><a href="#"><u>create_clock</u></a> -name &lt;clk&gt; -period &lt;p&gt; -waveform &lt;w&gt; &lt;root_pin&gt;</pre>                   | <pre><a href="#"><u>create_ccopt_clock_tree</u></a> -name &lt;clk&gt; -source &lt;root_pin&gt; -no_skew_group</pre> <pre>create_ccopt_skew_group &lt;clk&gt; -sources &lt;clk&gt; -auto_sinks</pre>                                                                                                                                           |
| <pre><a href="#"><u>create_generated_clock</u></a> -name &lt;g_clk&gt; -source &lt;s&gt; &lt;root_pin&gt; -divide_by &lt;factor&gt;</pre> | <pre><a href="#"><u>create_ccopt_generated_clock_tree</u></a> -name &lt;g_clk&gt; -source &lt;root_pin&gt; -generated_by &lt;root_pin_clk&gt;</pre> <pre><a href="#"><u>create_ccopt_skew_group</u></a> &lt;g_clk&gt; -sources g_clk -constrains none -auto_sinks</pre> <p>&lt;root_pin_clk&gt; is the clock sink in the parent tree that</p> |

|                                                                    |                                                                                                                                                                                                                                                                                                                               |
|--------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                    | <p>drives the generated clock tree.</p> <p>The <code>create_ccopt_clock_tree_spec</code> command always defines reporting only (that is <code>-constraints none</code>) skew groups for SDC generated clocks.</p>                                                                                                             |
| <code>set_case_analysis 0 &lt;pinA&gt;</code>                      | <code>set_ccopt_property sink_type -pin &lt;pinB&gt; ignore</code> <p>The <code>ignore</code> pin <code>&lt;pinB&gt;</code> is determined by analyzing the <code>set_case_analysis</code> command. For example, if the case analysis is set on a mux, CCOpt makes the inactive pin an ignore pin.</p>                         |
| <code>set_disable_timing &lt;pin&gt;</code>                        | <code>set_ccopt_property sink_type -pin &lt;pin&gt; ignore</code>                                                                                                                                                                                                                                                             |
| <code>set_clock_latency &lt;pin&gt; &lt;delay&gt;</code>           | <code>set_ccopt_property insertion_delay -pin &lt;pin&gt; &lt;delay&gt;</code>                                                                                                                                                                                                                                                |
| <code>set_clock_latency &lt;clock&gt; &lt;delay&gt;</code>         | <code>create_ccopt_skew_group -target_insertion_delay &lt;delay&gt;</code> <p>This SDC command does not directly output a <code>create_ccopt_skew_group</code> command, but causes the <code>-target_insertion_delay</code> parameter to be supplied with the corresponding <code>create_ccopt_skew_group</code> command.</p> |
| <code>set_clock_latency -source &lt;clock&gt; &lt;delay&gt;</code> | <code>set_ccopt_property source_latency -clock_tree &lt;clock_tree&gt; &lt;delay&gt;</code>                                                                                                                                                                                                                                   |
| <code>set_max_transition &lt;clock&gt; &lt;time&gt;</code>         | <code>set_ccopt_property target_max_trans -clock_tree &lt;clock_tree&gt; &lt;time&gt;</code>                                                                                                                                                                                                                                  |

In addition, any stop or ignore pins that were defined before running `create_ccopt_clock_tree_spec` will be rerun. This means that if you run `create_ccopt_clock_tree_spec -file`, these stop and ignore pins will appear in the generated script.

### Reporting on Clock Trees

The clock trees report displays a summary of all defined clock trees. This report includes skew information, physical metrics, and a summary of the numbers of clock gates at different depths in each clock tree. You run a clock tree report by running the `report_ccopt_clock_trees` command.

By default, the lists of pins and insertion delay histograms are omitted unless you specifically

want to see these items by running with the `-list_pins` or `-histograms` parameters of the [`report\_ccopt\_clock\_trees`](#) command.

For details, see the [`report\_ccopt\_clock\_trees`](#) command in the *EDI System Text Command Reference* document.

#### Clock Tree Summary

For design debugging purposes, the fastest way to get an overall idea of insertion delay and clocks trees at a glance, is by using the `-summary` parameter of the [`report\_ccopt\_clock\_trees`](#) command. This command then outputs the clock tree summary, which reports on each clock tree, delay corners (if defined), and the worst leaf and non-leaf slew against the global skew.

#### Example

The following command displays the summary table in the report:

```
report_ccopt_clock_trees -summary:
```

```
=====
Clock Timing Summary:
=====
```

Target and measured clock slews (in ns):

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

| Clock tree | Timing Corner                        |              | Worst Rising  |
|------------|--------------------------------------|--------------|---------------|
|            | Worst Falling                        | Worst Rising | Worst Falling |
|            |                                      |              | Leaf Slew     |
| Leaf Slew  | Nonleaf Slew                         | Nonleaf Slew |               |
| coreClk    | default_delay_corner_max:setup.early |              | 0.111         |
|            | 0.110                                | 0.094        | 0.097         |
| coreClk    | default_delay_corner_max:setup.late  |              | 0.111         |
|            | 0.110                                | 0.094        | 0.097         |
| coreClk    | default_delay_corner_min:hold.early  |              | 0.111         |
|            | 0.110                                | 0.094        | 0.097         |
| coreClk    | default_delay_corner_min:hold.late   |              | 0.111         |
|            | 0.110                                | 0.094        | 0.097         |

```
-----
```

#### Restricting the Clock Trees Report to One Corner

To get a clock report on a particular corner, for example, the worst-case corner, use the `-delay_corner` parameter of the `report_ccopt_clock_trees` command to restrict the clock trees report. For example:

```
report_ccopt_clock_trees -delay_corner SLOW_tc
```

You can then open up the [CCOpt Clock Tree Debugger](#) with the appropriate view, for example, the skew clock tree and analyze the slow timing corner. At this point you can change your slew and skew targets to add more buffers.

### Reporting on Skew Groups

The skew groups report displays a summary of all defined skew groups and delay corner combinations. This summary includes the target insertion delay, target skew, actual insertion delay, and actual skew for each combination of skew group, delay corner, and early/late paths. You run a skew group report by running the [report\\_ccopt\\_skew\\_groups](#) command.

For details, see the [report\\_ccopt\\_skew\\_groups](#) command in the *EDI System Text Command Reference* document.

The skew groups report contains three sections:

The first table lists the insertion delays and skews (along with target values) for each skew group in each delay corner:

Skew Group Summary:

| Timing Corner | Skew ID | Min ID  | Max ID | Skew    | Skew     | Skew         |
|---------------|---------|---------|--------|---------|----------|--------------|
| Group         | Target  | Target  | Target | Type    | Type     | Type         |
| <hr/>         |         |         |        |         |          |              |
| slow_tc.early | CWah    | - 1.001 | 1.093  | ignored | - 0.092  |              |
| slow_tc.late  | CWah    | *1.000  | 1.027  | 1.106   | explicit | *0.020 0.079 |

An asterisk (\*) beside a target indicates that the target could not be met. Specifically, a target is marked with an asterisk in the following situations:

- If a target insertion delay is present but there is no target skew, an asterisk is shown if the minimum insertion delay is greater than the target insertion delay, or if the maximum insertion delay is less than the target insertion delay.
- If a target skew is present but no target insertion delay, an asterisk is shown if the skew is greater than the target skew.
- If both a target skew and a target insertion delay are present, an asterisk is shown on the target insertion delay if either the minimum insertion delay is less than [target insertion delay - (target skew\*0.5)] or the maximum insertion delay is greater than [target insertion delay + (target skew\*0.5)]. An asterisk is shown on the target skew if the skew is greater than the target skew.

The **second table** lists the start and end points of the min and max paths in the skew group:

Skew Group Min/Max path pins:

=====

```
-----
-----  
Timing Corner Skew Group Min ID Path ID Max ID  
Path ID
```

---

```
-----  
slow_tc.late Cwah 1.027 1 1.106  
2  
- min FrgletrH/TCH/ClsH02/Rvgv23_elz0/CK  
- max FrgletrH/WRJH/TvgoArigken_elz10/CK  
...  
-----
```

This table shows that, for `slow_tc.late`:

- The minimum insertion delay is 1.027.

The path that has that minimum insertion delay runs from the root to the pin

`FrgletrH/TCH/ClsH02/Rvgv23_elz0/CK`

In the table below, **Path ID 1** shows a full path trace for this path.

- The maximum insertion delay is 1.106.

The path that has that maximum insertion delay runs from the root to the pin

`FrgletrH/WRJH/TvgoArigken_elz10/CK`

In the table below, **Path ID 2** shows a full path trace for this path.

The remaining report contains detailed min and max path trace information for each skew group, in each delay corner. The path trace shows the arrival time at each pin along the path, together with additional information such as the slew at that pin, and the physical coordinates of the pin.

```
-----  
-----  
Name Lib cell Event Incr Arrival Slew Cap ...  
(ns) (ns) (ns) (pF)...  
-- Clockpath trace  
-----  
-----  
Cwah  
rise 0.000 0.005 0.089...  
FrgletrH/TCH/ClsH02/ED_WIN_YWZESP_Cwah_Z0_I2_3/z1/B  
NAND2X1 rise 0.002 0.002 0.005 ...  
FrgletrH/TCH/ClsH02/ED_WIN_YWZESP_Cwah_Z0_I2_3/z1/Y
```

```
NAND2X1 fall 0.030 0.032 0.027 0.002...
FrgletrH/TCH/ClsH02/ED_WIN_YWZESP_CWah_Z0_I2_3/z2/A
INVX3 fall 0.000 0.032 0.027 ...
FrgletrH/TCH/ClsH02/ED_WIN_YWZESP_CWah_Z0_I2_3/z2/Y
INVX3 rise 0.255 0.287 0.419 0.145...
FrgletrH/TCH/ClsH02/Rvgv10_elz0/CK
EDFFX1 rise 0.000 0.287 ...
-----
----
```

Note that if the first pin in the trace is the source pin of the skew group, and the source of that skew group is a non-generated clock tree, the arrival time at that first pin will be the source latency of the skew group. Otherwise, the arrival time at the first pin should always be zero.

## CCOpt Clock Tree Debugger

The *CCOpt Clock Tree Debugger* tool provides the GUI support for the CCOpt native integration mode. It provides clock tree debugging facilities that will help you better understand the quality of the CCOpt results. The tool displays a top-down tree view of the CCOpt clock trees with the vertical axis representing delays.

The *CCOpt Clock Tree Debugger* provides the following three basic views for clock tree visualization:

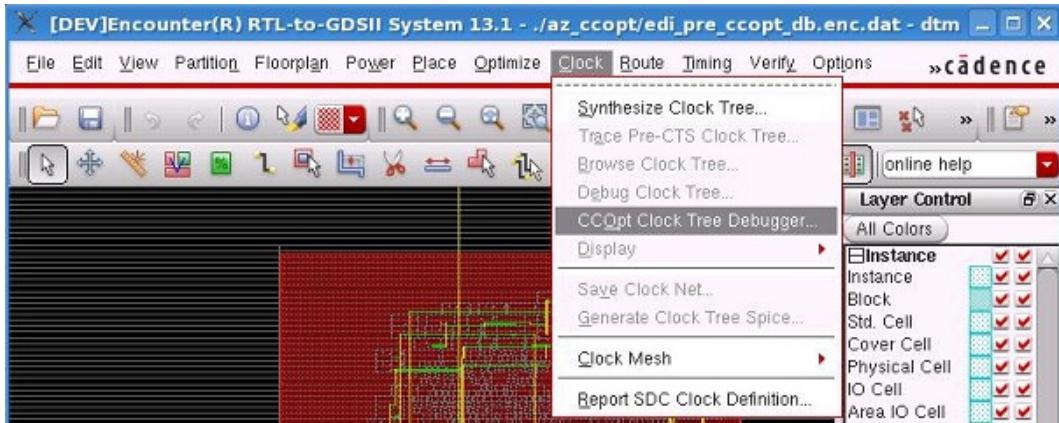
1. **Clock Tree Debugger:** Displays a top-down tree view of the CCOpt clock trees
2. **Clock Path Browser:** Displays clock path data in a table and provides the option for bringing up a clock path analyzer from its context menu or by double-clicking on a row in the table
3. **Clock Path Analyzer:** A view examining a single clock path

The *Clock Tree Debugger* has the following features:

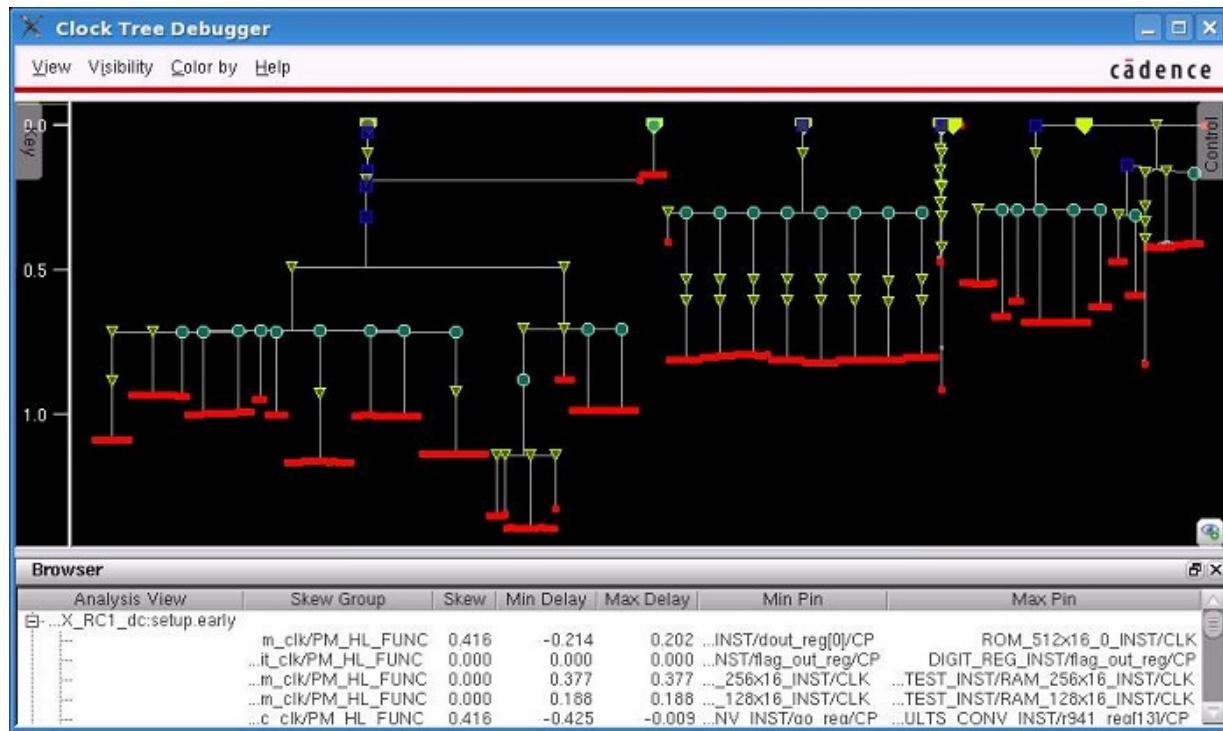
- The tool display shows a schematic-like representation of the CCOpt clock trees. You can control the level of details and the elements that you want to view.
- The results shown in the tool are based on the CCOpt clock tree spec file.
- The units on the vertical scale of the display represent time, so that distance between two elements indicates delays that you can measure.
- A comprehensive search mechanism lets you find instances, paths, and reconvergent clocks easily.
- The tool displays several clocks at the same time.
- The tool can display reconvergent and crossover clocks.

The tool can be accessed from the *Clock Menu* of the EDI System. Choose *Clock < CCOpt Clock Tree Debugger*.

**⚠** The CCOPT Clock Tree Debugger 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. To use the limited-access feature, you need a license for the EDS210 product number.



The main window of the *Clock Tree Debugger* opens. By default, the *Clock Path Browser* opens at the bottom of the window. The *Clock Path Analyzer*, when invoked, replaces the *Clock Path Browser* in its window.



The *CCOpt Clock tree Debugger* submenu only appears when you have created a CCOpt clock tree specification file using the [`create\_ccopt\_clock\_tree\_spec`](#) command. To view the menu, perform the following steps:

1. Load the placed design.
2. Run the [`create\_ccopt\_clock\_tree\_spec`](#) command.
3. Select *Clock < CCOpt Clock Tree Debugger* in the EDI System main menu.

The main area of the window displays the clock trees and the three menus available in the window, *View*, *Visibility*, and *Color by*, which provide you with options to customize the clock tree visualization.

For details of the *CCOpt Clock Tree Debugger*, see the [\*CCOpt Clock Tree Debugger\*](#) section in the *Clock Menu* chapter in the *EDI System Menu Reference* document.

---

# Working with Clock Mesh Structures

---

- Overview
- Clock Meshes Versus Clock Trees
- Creating Clock Meshes
  - Determining the Mesh Structure
    - Supported Mesh Styles
    - Clock Mesh Structure Characteristics
    - Multilevel Structure of a Mesh
  - Implementing the Clock Mesh
  - Analyzing the Clock Mesh
    - Pre-Route Wire Estimation
    - RC Extraction
    - Computing Mesh Delays
  - Generating Multiple Spice Run Deck For Big Clock-Mesh Networks
  - MultiSpine Clock Mesh

## Overview

The Encounter® Digital Implementation System (EDI System) 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  $M_2$ . If the mesh is using a vertical trunk on  $M_2$  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 [Edit Clock Mesh Specification](#) form. This makes it easier to experiment with different clock mesh structures.

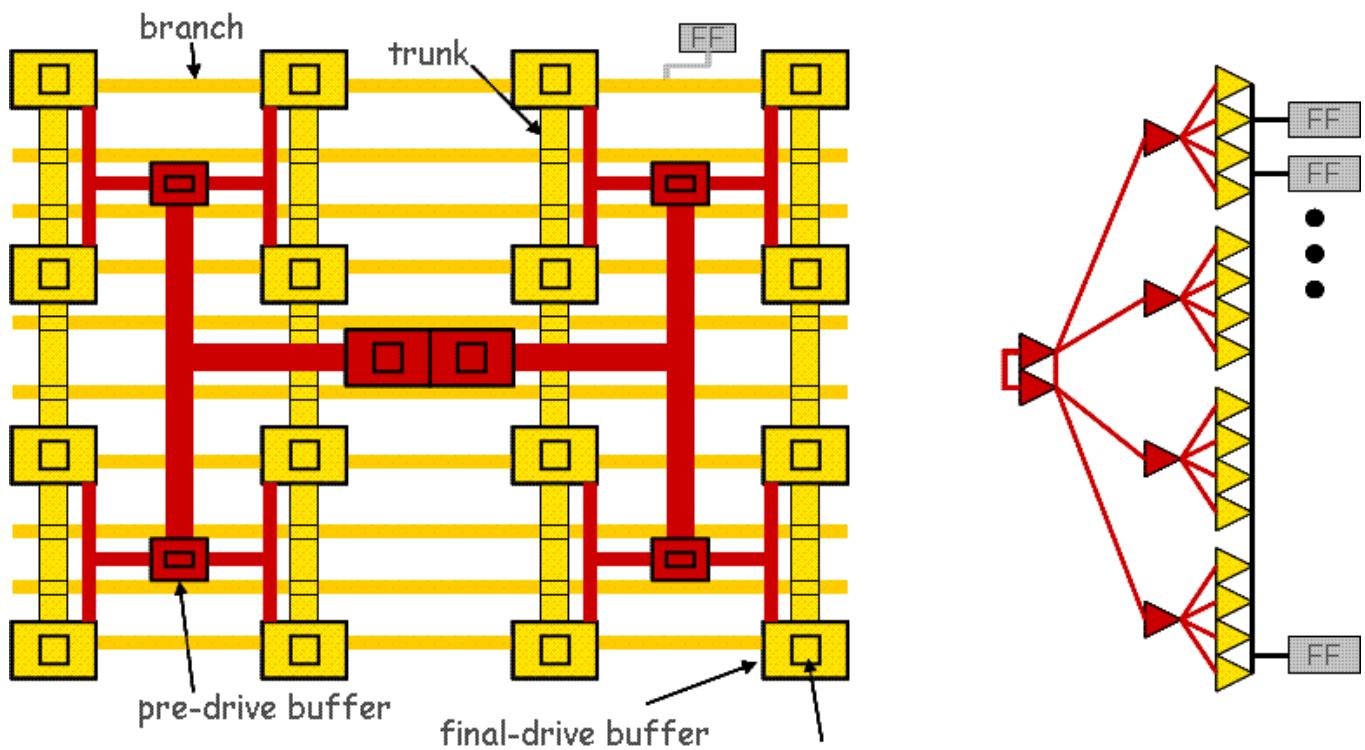
### Supported Mesh Styles

There is a wide variety of mesh styles currently in use. The Encounter software can synthesize the following mesh styles:

- H-tree + Mesh

This classic style uses a multilevel H-tree pre-drive, followed by a general mesh final stage. Although the pre-drive is a tree structure, it can still employ multiple drivers on a single net. The final stage consists of a rectangular grid of final drivers feeding a rectangular mesh grid of trunks and branches.

**Figure 21-1 H-Tree + Mesh Style**

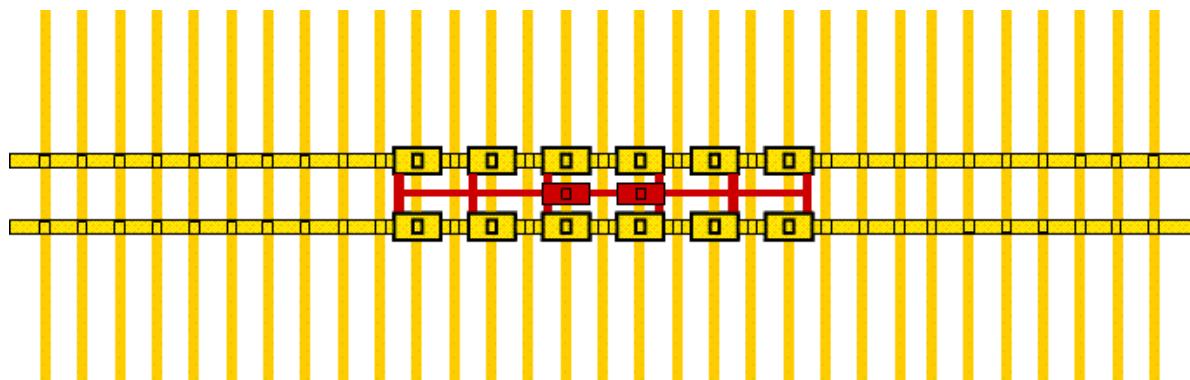


The advantage to this style is that the pre-drive is highly symmetrical and can achieve good skew control for large clock domains with many flip-flops. The drawback is that it can require higher power than other mesh styles.

- **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

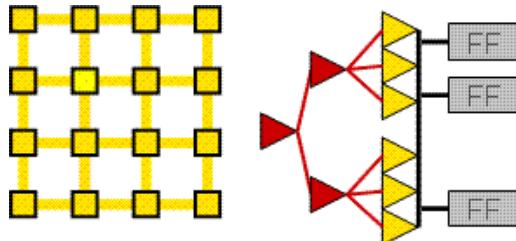
trunks. The fishbone style can be a very good style for smaller clock domains.

### 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 Encounter software have the following basic characteristics:

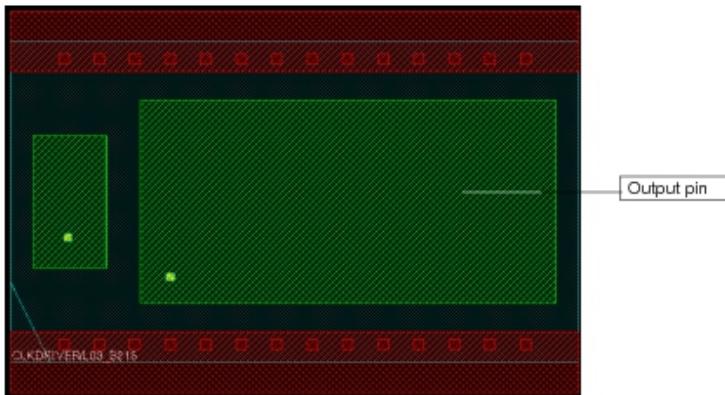
- Symmetry  
Meshes rely heavily on symmetry of netlist structure and routing pattern to achieve tight skew control and tolerance to variation.
- Routing patterns and topologies  
A major difference between clock meshes and clock trees are their routing patterns. Mesh routing commonly has the following characteristics:
  - Makes heavy use of non-tree routing topologies containing cycles or loops.
  - Uses wide wires to achieve low resistance and well controlled capacitance. For example it is not unreasonable for a mesh trunk to be 10  $\mu\text{m}$  wide.
  - Routing is implemented using a combination of special and regular routes.
- Parallel drive  
Clock meshes typically rely on multiple buffers or inverters working in parallel to drive a given net in the pre-drive and final-drive stages, whereas clock trees always use exactly one driver per net. Using multiple parallel drivers makes it possible to drive heavily loaded nets from multiple points. This allows a mesh to fan out with fewer stages and better skew control than a tree.

**Figure 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 EDI System 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

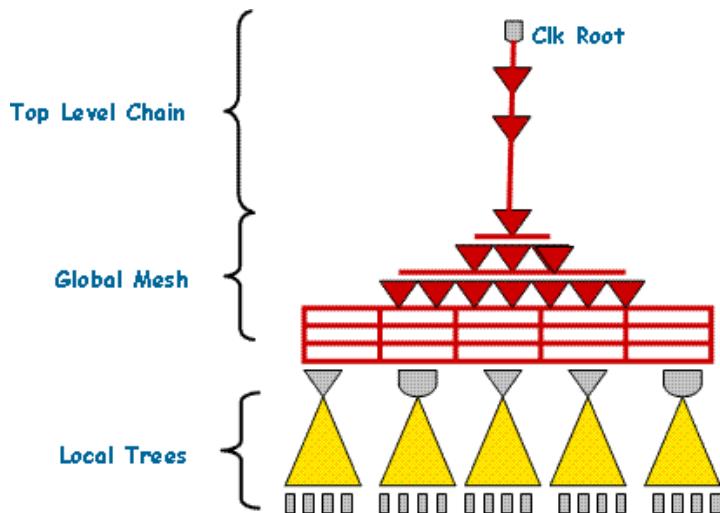
a single final global mesh net. This final mesh net can connect directly to the clock input pins, or there can be an additional level of local distribution.

- 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 EDI System clock mesh feature includes the following major implementation capabilities:

- Synthesis

The [synthesizeClockMesh](#) command implements the clock mesh according to the current specification, by inserting and placing buffers, and creating mesh special routes as needed.

- Mesh routing

The [routeClockMesh](#) command uses the native NanoRoute™ router to complete detailed routing connections for driver and flip-flop inputs, top-level cascade chains, and local trees.

- Wire trimming

After mesh routing is complete, the [trimClockMesh](#) command removes unused portions of the mesh trunks and branches, eliminates antenna violations, and reduces overall mesh capacitance.

## Analyzing the Clock Mesh

The Encounter software can analyze clock meshes at different stages of the implementation process (pre-route or post-route), using default extraction, detail extraction, or externally supplied RC data, and using either delay calculation technology or a circuit simulator. Various reports are available. To help visualize mesh quality, delay or transition information can be displayed graphically using a color scale.

Clock meshes have several characteristics that make them more difficult to analyze than clock trees:

- Meshes use a mixture of special routing and regular routing, which makes estimating and extracting wire RC information more challenging.
- Nets with multiple drivers need special consideration during delay calculation.
- A final stage clock mesh net might have thousands of drivers and tens of thousands of receivers, resulting in tens (or hundreds) of millions of driver-to-receiver timing arcs. This huge number of arcs can potentially lead to memory or performance problems during timing analysis.

The following sections describe how the Encounter software addresses some of the unique challenges to analyzing clock mesh timing.

### Pre-Route Wire Estimation

The software's clock mesh synthesis inserts and places buffers and draws mesh trunks and branches, but it does not immediately start NanoRoute to complete detail routing connections for mesh driver inputs and flip-flops; this is handled by a separate command. Therefore, some wire estimation is needed to complete pre-route clock mesh analysis.

For nets without any special net routing, such as top-chain nets and local distribution nets, the software uses a simple two-layer Steiner routing estimation based on the routing preferences given in the clock mesh specification.

For global nets, the software builds an RC network for the existing special net routing and then estimates nearest connections for each unconnected driver input and flip-flop. For these nets, nearest point-to-point connections are more appropriate than Steiner routing estimates because NanoRoute will connect them using "pattern trunk" connections.

### RC Extraction

There are several different ways to get RC information for clock mesh analysis:

- Default extraction

Default extraction is simple 1-D extraction based on per-unit-distance resistance and capacitance values for mesh routes. Default extraction is most appropriate before full detailed

routing is complete.

**Note:** Default extraction is the only available method to analyze pre-route clock mesh nets.

- Detail extraction

The Encounter software detail extraction can be used when mesh routing is complete.

**Note:** The clock mesh analysis commands will not automatically start detailed net extraction; to use this method, you must directly set the extraction mode then run the [extractRC](#) command prior to mesh analysis.

- External SPEF

To use RC data from an external parasitic extraction tool, load the SPEF prior to clock mesh analysis.

## Computing Mesh Delays

There are two ways of computing mesh delays and transitions times: delay calculation and circuit simulation.

- Delay Calculation

The fact that clock meshes rely on multiple buffers driving a single net presents some challenges for delay calculation. Although not all delay calculators can handle it, Encounter clock mesh uses delay calculation technology that supports multi-driven nets. For delay calculation to be accurate, all drivers on a given net must have similar input waveforms. This restriction is consistent with a well-designed mesh and probably does not present a major limitation in practice. The software generates warnings when the skew at multiple driver inputs exceeds a given fraction of their input slew.

- Circuit Simulation

The software supports simulation with Virtuoso® UltraSim™ fast SPICE simulator as an alternative to delay calculation for analyzing clock mesh delays and slews. Virtuoso® UltraSim™ simulator is a fast-SPICE simulator that is well-suited for analyzing clock meshes, even with post-route RC data. Virtuoso UltraSim simulator is not packaged with the Encounter software, therefore to use it, you must have the Virtuoso UltraSim simulator executable in your path, and an appropriate license.

The software also supports simulation with external SPICE-like simulators, but the flow is not as convenient as with Virtuoso UltraSim simulator. The steps are as follows:

1. Dump a SPICE netlist for the clock mesh.

- Manually run the simulator outside of the Encounter software.
- Backannotate the simulation measurement results. Encounter clock mesh provides commands to write the spice netlist and backannotate the results.

To simulate clock meshes, either with Virtuoso UltraSim simulator or with an external simulator, the Encounter software needs to have driver sub-circuit and model information available. This information must be provided via a plain-text cdb database file.

## 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, EDI System 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:

- Enable multi-part spice simulation with the `MultiPartSpice` keyword in the Analysis section of the clock mesh specification file.
- 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.
- Generate spice L1 and L2 run decks.
- Simulate the L1 spice deck. Measure delays and transitions for all points. Measure detailed waveforms for lowest level receivers.
- Use lowest level waveforms from previous step to generate corresponding voltage sources L2 runs.
- Simulate L2 spice decks.
- Use L1 and L2 measurement results to back-annotate complete mesh timing to EDI System.

EDI System 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.<ul style="list-style-type: none"><li>▪ Set <code>-preRouteAnalysis</code> or <code>-postRouteAnalysis</code> mode to UltraSim.</li><li>▪ Invoke an analysis command, such as <code>reportClockMesh</code>.</li></ul></li></ol> |
| Simulate and back-annotate                             | <ul style="list-style-type: none"><li>▪ Set up multi-part spice simulation in the</li></ul>                                                                                                                                                                                                                                                      |

|                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| automatic with a user-defined simulation procedure | clock mesh specification file.<br><ul style="list-style-type: none"> <li>▪ Set <code>-preRouteAnalysis</code> or <code>-postRouteAnalysis</code> mode to Spice.</li> <li>▪ Define a procedure that can invoke the user-defined Spice simulator.</li> <li>▪ Set <code>simSpiceProc</code> to name of procedure.</li> <li>▪ Invoke analysis command, such as, <code>reportClockMesh</code>.</li> </ul>                                                                                                                                                                       |
| Manual simulation and back annotation              | <ul style="list-style-type: none"> <li>▪ Set up multi-part spice simulation in the clock mesh specification file.</li> <li>▪ Use the <code>spiceClockMesh</code> command to dump the L1 and L2 spice decks.</li> <li>▪ Manually simulate L1.</li> <li>▪ Use <code>spiceClockMesh -genL2Sources {}</code> to create the voltage sources to stimulate the L2 runs.</li> <li>▪ Manually simulate L2.</li> <li>▪ Back-annotate results using <code>spiceClockMesh -readMapMeas {}</code>.</li> <li>▪ Invoke analysis command, such as <code>reportClockMesh</code>.</li> </ul> |

### 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

```

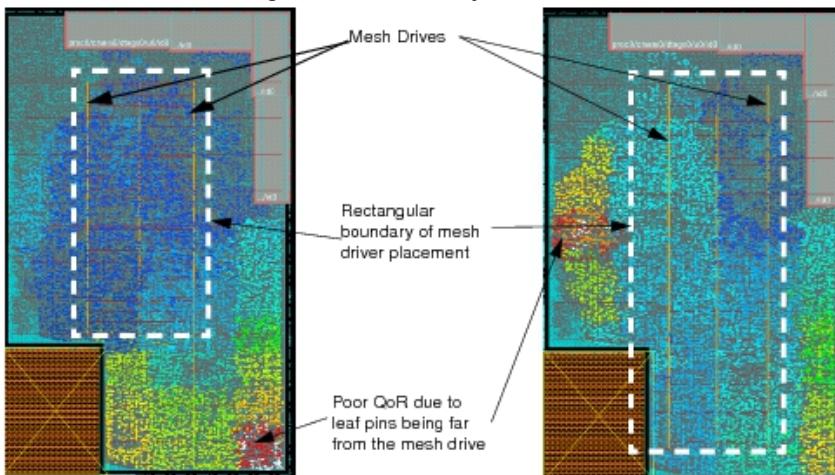
**Note:** The [setClockMeshMode](#) -simMultiPartSpiceBoundaryReceiverAsInst parameter controls whether the final receivers in L1 are simple pin capacitance values, or actual instantiated subcircuits.

**Note:** The [setClockMeshMode](#) -simMultiPartSpiceNrInstThreshold parameter combines the local portions of a clock network in one or separate SPICE files.

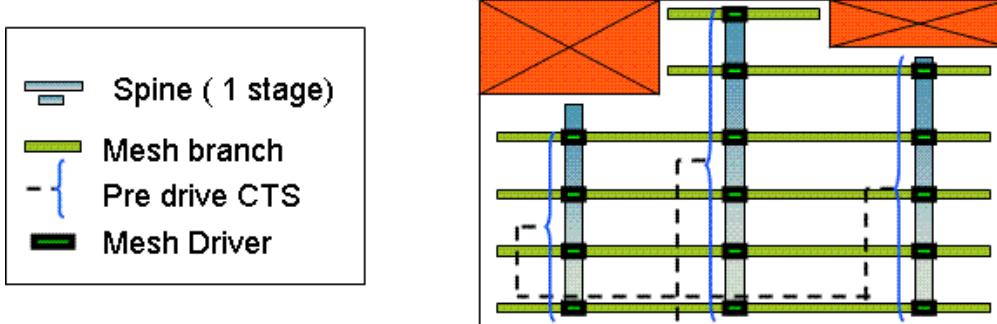
## **MultiSpine Clock Mesh**

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.

---

# Editing Wires

---

- [Overview](#)
- [Before You Begin](#)
- [Results](#)
- [Using Keyboard Shortcuts](#)
- [Selecting Wires](#)
- [Deleting Wires](#)
- [Moving Wires](#)
- [Adding Wires](#)
- [Cutting Shielding Wires](#)
- [Trimming Antennas on Selected Stripes](#)
- [Changing Wire Width](#)
- [Repairing Maximum Wire Width Violations](#)
- [Duplicating Wires](#)
- [Stretching Wires](#)
- [Changing Wire Layers](#)
- [Splitting and Merging 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
- Delete wires
- Merge selected wires
- Force wires to use specified widths
- Add vias

For power wires, you can perform all of the actions available for signal wires, as well as the following additional actions:

- Trim selected wires
- Split selected wires
- Duplicate selected wires
- Change wire width
- Fix wires wider than the maximum width
- Force wires associated with special nets to be created as signal wires
- Change vias

## Before You Begin

Before you can use the wire editing features, load the design into the current EDI System session.

## Results

After you use the wire editing features, the EDI System software saves the new and modified wires and vias in the database.

## Using Keyboard Shortcuts

The EDI System software includes keyboard shortcuts for use with the wire editing features. Type the keyboard shortcuts while the main EDI System 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                                               |
| M       |    | Move Wire                                            |
| O       |   | Add Via                                              |
| Shift+R |  | Move/Resize/Reshape<br>(non-connectivity-based move) |
| S       |  | Stretch Wire                                         |
| Shift+X |  | Cut Wire                                             |

For more information, see ["Tool Widgets"](#) in the *EDI System 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 | 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.

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.

|        |                                     |
|--------|-------------------------------------|
| 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 *EDI System 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.                                                                                                                                   |
| Control-w    | Deletes the last segment created in the design area. This allows you to remove one segment of the route at a time.                                                                             |
| Esc          | Removes the entire route.                                                                                                                                                                      |
| Number keys  | Change the added wire to a specific layer number. If you want the wire to be added to metal layer 1, use the 1 keyboard shortcut, use the 2 keyboard shortcut for metal layer 2, and so forth. |
| Single-click | Ends the segment, allowing you to continue the route in either the same direction or the orthogonal direction.                                                                                 |
| Double-click | Ends the route.                                                                                                                                                                                |

## Keyboard Shortcuts Used in Stretch Wire Mode

|   |                                                                                         |
|---|-----------------------------------------------------------------------------------------|
| 1 | Stretches or reduces horizontal wires from the left and vertical wires from the bottom. |
| 2 | Stretches or reduces horizontal wires from the right and vertical wires from the top.   |

For more information, see Stretching Wires.

## Keyboard Shortcuts Used to Change Vias

|   |                                                         |
|---|---------------------------------------------------------|
| N | Changes the selected via to the next available via.     |
| P | Changes the selected via to the previous available via. |

For more information, see Changing Vias.

## Selecting Wires

1. Click the *Move Wires* widget in the Tool Widgets area of the EDI System main window or press the **M** keyboard shortcut while the cursor is in the design display area.
2. Click a wire.

## Deleting Wires

To delete a wire without deleting the vias connected to it, complete the following steps:

1. Turn on [Auto Query](#).
2. Move the cursor over the wire to delete.
3. Use the **N** (next) or **P** (previous) keyboard shortcut to 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 special 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 EDI System software does not delete wires or vias with status COVER. Type the following commands to freeze the wires and vias on metal layers 1 and 2:

```
deselectAll
```

```
editSelect -layer {M1 M2 }
editSelectVia -cut_layers V12
editChangeStatus -to COVER
```

-  You cannot change the wires and vias back to their non-COVER status, so you should issue the `saveSpecialRoute` and `saveRoute` commands before changing the status.

## Moving Wires

You can 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 EDI System main window.  
The equivalent keyboard shortcut is Shift+R.
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 *Route* tab.
3. Specify a value, in microns, in the *Arrow increment* field.

This value defines the distance that the wire is to move each time you press an arrow key while holding the **shift** key. You can specify either a positive or negative number.

4. Click the *Move Wire* icon in the Tool Widgets area of the EDI System main window.  
The equivalent keyboard shortcut is **M**.
5. Click the wire to be moved.  
The selected wire is highlighted.
6. Hold the **shift** key, then press the up or down arrow key for a horizontal wire or the left or right key for a vertical wire.  
The selected wire moves in the direction of the arrow.

## **Moving Selected Wires or Vias**

To copy, paste, and move selected wires or vias, complete the following steps:

1. Select wires or vias.
2. Type the `editDuplicate` command (or use the **c** keyboard shortcut) to copy the objects.
3. Use the **shift+R** keyboard shortcut to switch to *unconn move* mode.
4. Move the mouse over any of the selected objects. A black dot appears.
5. Click once to start moving the selected objects.
6. Click again to place the objects in the desired location.

**Note:** To cut, paste, and move selected 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.

This places the EDI System software in the *Edit Wire* mode and the mouse cursor changes to a pencil. In addition, EDI System is automatically placed in the *Auto Query* mode, even if the Q widget below the design display area is not enabled.

The equivalent keyboard shortcut is Shift+A.

2. If pins are not visible, use the *Layers* tab of the [Color Preferences](#) form to make pins visible.
3. Place the cursor over the pin or wire at the starting point for the wire to be drawn, and then type 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. (Optional) Click the *Route* tab on the *Edit Route* form and adjust the values in the Layer and Width fields.

6. (Optional) Click the *Misc* tab and select a shape from the pulldown menu.

**Note:** Shapes are only defined for power wires. This value is ignored for signal wires.

7. Click the startpoint for the wire you want to add, then move the mouse to a new point.  
The wire is drawn interactively as you move the mouse.

8. (Optional) Click a new location to change the direction of the wire or continue in the same direction with a different segment.

**Note:** If there is a layer change, a via is automatically created.

9. (Optional) Press a number key to change the layer of the wire being added.

When the software is in 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.

10. (Optional) Press the Backspace key to erase the most recently drawn segment.

You can do this for as many segments as needed.

11. Double-click the mouse.

The wire ends at the location of the cursor.

**Note:** After double-clicking, you cannot use the Backspace key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the Edit Delete form.

## Adding Wires for Multiple Nets

To add parallel wires to multiple nets at the same time, complete the following steps:

1. Click the *Edit Wire* widget.

This places the EDI System 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 *Nets* tab on the Edit Route form and enter the net names into the *Nets* field.

**Note:** You can also specify a file that contains a list of nets. See Adding Wires that Automatically Extend to a Target for more information.

4. (Optional) Click the *Route* tab, then select horizontal and vertical layer names and specify width and spacing values.

**Note:** To use different width or spacing values for different nets, use the *Override* tab. See Using Override to Add Wire Groups with Multiple Widths and Spacing for more information.

5. (Optional) Specify a value in the *Drawing wire* field.

This specifies which of the nets (specified in the *Nets* field) corresponds to the mouse pointer location. By default, this value is 1, meaning the mouse position corresponds to the position of the left-most or bottom-most net of the group.

For example, if the *Nets* field contains vss VDD VDDA VSSA, the vss net is the bottom-most net for horizontal segments, and the left-most net for vertical segments. If the value in the *Drawing wire* field is set to 1, the mouse location corresponds to wires on the VSS net.

6. (Optional) Click the *Misc* tab and select a shape from the pulldown menu.

**Note:** Shapes are only defined for power wires. This value is ignored for signal wires.

7. Click the startpoint for the wires you want to add, then move the mouse to a new point.

The wires are drawn interactively as you move the mouse.

8. (Optional) Click a new location to change the direction of the wires or to continue in the same direction with a different segment.

**Note:** If there is a layer change, a via is automatically created.

9. (Optional) Press the Backspace key to erase the most recently drawn set of segments.

You can do this for as many sets of segments as needed.

10. Double-click the mouse.

The wires end at the location of the cursor.

**Note:** After double-clicking, you cannot use the Backspace key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the Edit Delete form.

## Adding Wires that Automatically Extend to a Target

To add wire groups to multiple nets that automatically extend to targets, complete the following steps:

1. Click the *Edit Wire* widget.

This places the EDI System 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. Create a text file that contains the names of multiple nets.

Make sure that each line in the file contains the name of one net, and that the nets are listed in the order in which you want to create the wire group.

4. Click the *Nets* tab on the *Edit Route* form.

5. Click the *Read* button.

This opens the *Open* form.

6. Select the file you created in step 3, then click *OK*.

The *Nets* field now contains the net names in the file.

7. Click the *Route* tab, then select horizontal and vertical layer names and specify width and spacing values.

**Note:** To use different widths or spacing values for different nets, use the *Override* tab. See [Using Override to Add Wire Groups with Multiple Widths and Spacing](#) for more information.

8. Click the *Misc* tab, and select the *Extend Start Wires* and *Extend End Wires* options. These options extend both ends of the wires until they connect to a target.

9. Click the point in the design display area where the left-most or bottom-most wire should start.

**Note:** The startpoint does not have to be at a target.

10. Move the mouse horizontally or vertically.

The wires are drawn interactively.

11. Double-click the mouse.

The startpoint and endpoint of the wire extend until they connected to a target. If no target is present, the wire does not extend.

## Using Override to Add Wire Groups with Multiple Widths and Spacing

To add pairs of power and ground wires, where wires have different widths and spacing, complete the following steps:

1. Click the *Edit Wire* widget.

This places the EDI System 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 *Nets* tab on the Edit Route form and enter the net names into the *Nets* field.

**Note:** You can also specify a file that contains a list of nets. See Adding Wires that Automatically Extend to a Target for more information.

4. Click the *Route* tab, then select the horizontal and vertical layer names and specify the width and spacing values.

5. Click the *Override* tab and enter a set of width and spacing values for the nets that do not have default width and spacing values.

For example, if you want wires for the third and fourth nets to have a wider width and larger spacing, specify the following values:

3 6 4

4 6

The first line indicates the third net (3) has a width of 6 microns and that the spacing value between the third and fourth net is 4 microns. The second line indicates that the fourth net (4) has a width of 6 microns.

**Note:** To specify a value of less than 1, you must include a 0 before the decimal point. For example, a value of .6 is not valid, and must be expressed as 0.6.

6. Click the point in the design display area where the left most stripe should start.

**Note:** The startpoint does not have to be at a target.

7. Double-click the mouse.

The wires end at the location of the cursor.

## Cutting Shielding Wires

1. Click the *Cut Wire* widget.

The cursor changes to the shape of a scissors, indicating that the EDI System 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.

**Note:** If multiple objects exist at the location of the cursor, press the spacebar to toggle the selection between them. To select multiple objects, press the Shift key while clicking.

7. Use the D keyboard shortcut to delete the selected objects.

**Note:** Wires can only be cut in the orthogonal direction. If you cut multiple wires, including wires in the same direction as the cut, the cut only affects wires in the orthogonal direction to the cut. Once cut, signal wire pieces maintain a 1/2 wire width extension, but power wires are not extended.

## Trimming Antennas on Selected Stripes

If your completed power structure contains stripes in a mesh configuration, physical antennas might remain on some stripes.

1. Use the D keyboard shortcut to display the Select/Delete Routes form.
2. Choose *Select* from the *Action* pulldown menu.
3. (Optional) Click *Nets*, then specify one or more nets for the wires to be trimmed.
4. (Optional) Click *Direction*, then click H to trim horizontal wires or V to trim vertical wires.
5. Click *Shapes*, then select *STRIPE*.
6. Click *Apply*.

The selected wires are highlighted in the design display area.

7. Use the Shift+T keyboard shortcut to trim the selected wires.

The selected power wires are trimmed back to the last connection point and deselected.

**Note:** Signal wires cannot be trimmed.

## Changing Wire Width

After running power analysis, you might need to increase the width of some power stripes to alleviate any IR drop or EM issues.

1. Make sure the software is in *Select* mode (you can use the 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 *Route* tab on the Edit Route form and enter values in the *Width* fields.  
Specify a width value in the *Horizontal* section for horizontal wires and a width value in the *Vertical* section for vertical wires.

4. Use the **Shift+W** keyboard shortcut.

This changes the width of the selected wire. Any via connected to that the wire is also updated based upon the new width.

**Note:** Widths for signal wires depend on the applicable LEF rule, no matter what value is populated in the GUI. To specify a wire width that is different from the default wire width value, select *Force Special* or a value other than *Default* from the *Rule* pull-down menu on the Nets tab of the Edit Route form.

## 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 Wires

After running power analysis, you might need to add some power stripes to alleviate any IR drop or EM issues. Instead of creating new wires interactively, you can duplicate existing wires.

1. Make sure the software is in the *Select* mode (you can use the **A** keyboard shortcut), then click the wire segment to duplicate.

2. Use the **E** keyboard shortcut.

This displays the [`Edit Route`](#) form without placing the software in the *Edit Wire* mode.

3. Click the *Route* tab on the Edit Route form and specify the vertical and horizontal layers for the duplicated wires.

**Note:** The widths of the duplicated wires are always the same as the original wires, but the layers are the ones specified in the form.

4. Click the *Duplicate selected wires* widget or use the **c** keyboard shortcut.

The duplicated wires are automatically selected and placed directly on top of the original wires.

**Note:** To duplicate a wire and change the layer, use the `editDuplicate` command and specify the layer for the duplicate wire. For more information, see [`editDuplicate`](#) in the "Wire Edit

Commands" chapter of the *EDI System Text Command Reference* .

## 5. 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 wires to the desired location.

## Stretching Wires

1. Click the *Select* widget in the Tool Widgets area of the EDI System main window.  
The cursor shape is an arrow, indicating that EDI System 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 EDI System 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. 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 arrows on the *Stretch Wire* widget change so that the outer arrow points to the right.

## Changing Wire Layers

You may need to change sections of wires to different layers in order to relieve congestion on a specific layer or to fix process antenna violations.

1. Make sure the software is in 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 *Route* tab on the Edit Route form and enter values in the *Layer* fields.  
Specify a layer value in the *Horizontal* section for horizontal wires and a layer value in the *Vertical* section for vertical wires.

4. Use the L keyboard shortcut.

This changes the layer of the selected wire. Any via connected to that wire is also updated based on the new layer.

## Splitting and Merging Wires

Stripes that spread over the entire die may need to be altered in specific locations. In this case, a stripe that is represented as a single piece of wire must be split into multiple segments. You can split a wire to 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 wires.

After splitting a wire, you can merge those wire segments that align back into a single segment.

1. Select a single segment.
2. Use the M keyboard shortcut.

This merges the wire segments into a single segment.

## Adding Vias

1. Select the *Add Via* widget. The equivalent keyboard shortcut is o.
2. Press the F3 key.  
This displays the *Edit Via* form.
3. Select *Geometry* in the *Create Via by* field.
4. Fill all of the fields in the form. For more information, see [Edit Via](#) in the *EDI System 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 *EDI System 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.
- 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 **N** (next) or **P**(previous) keyboard shortcut to display a via that has the same LEF rule as the selected via.
  - If a via is available, the display is updated with the new via when you press the keyboard shortcut.
  - If another via is not available, you will hear a warning beep when you press the keyboard shortcut. This can occur when only one via is defined in the LEF file, when the currently queried object is not a via, or when no object is currently queried.

- **Using the Edit Power Vias form**

For information, see [Edit Power Vias](#) in the *EDI System Menu Reference*.

**Note:** You cannot change vias using the Edit Route form.

## Moving Vias

You move vias the same way you move wires. The EDI System software moves vias without considering connectivity. For more information, see Moving Wires.

## 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 EDI System 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.

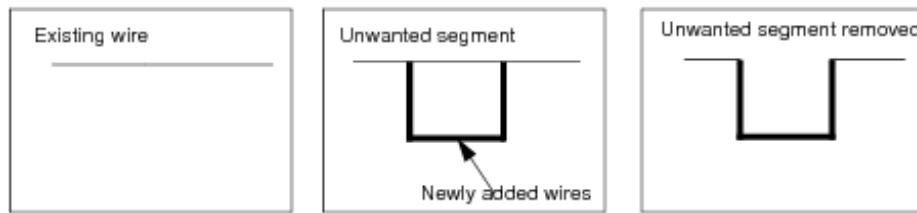
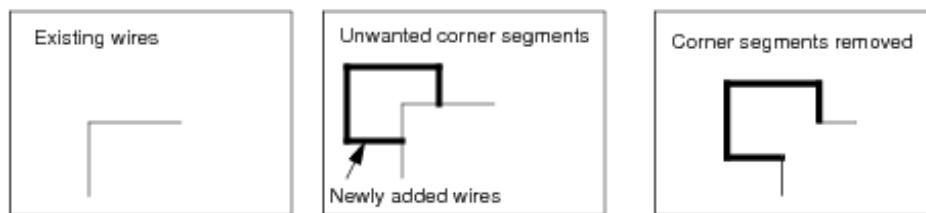
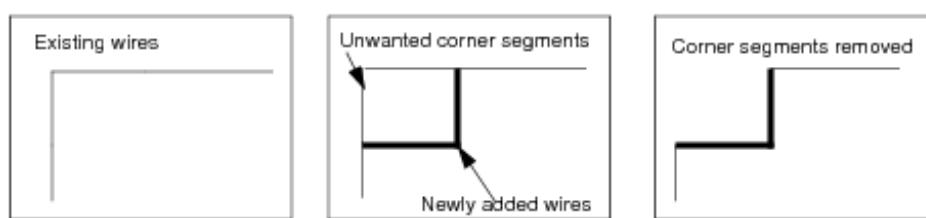
2. Select the Edit Route form from the *Edit* menu and click the *Misc* tab.

The equivalent keyboard shortcut is E.

3. Select the *Reshape* option on the form.

4. Add wires to the design, as described in Adding Wires.

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, *B/kg*, 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.

# 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](#)
  - [Congestion Markers in the Display](#)
  - [Congestion Distribution Report](#)
- [Improving Route Congestion](#)
- [Using Bus Guides](#)
- [Additional Information](#)
  - [Wire Overlap](#)

## 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.

- ❶ If you make any changes after running Trial Route that affect the placement data--for example, floorplan changes--you must run placement before rerunning Trial Route.

The design must be loaded into the current EDI System session.

The following optional input files are only required as necessary:

- DEF file
- Top Design Format (TDF) file containing routing data
- Routing guide file

## Routing A Flat Design

- Initial Design Routing

Run Trial Route for the first time to gauge the routability of the design. You can then examine the congestion map and congestion distribution report to identify congested areas that might cause routing problems later in the design session.

- a. Choose *Route - Trial Route* .

This opens the Trial Route Form.

- b. Select the *Prototyping* effort level and click *OK* .

You can also issue the following command:

```
trialRoute -floorplan
```



The prototyping (-floorplan) mode runs Trial Route quickly, which is important when prototyping large designs. However, note that components in your design might not be routed at legal locations.

- Post Clock Tree Synthesis Routing

Run Trial Route after clock-tree synthesis to recheck the routing congestion and to estimate parasitic values for timing analysis.

- a. Choose *Route - Trial Route*.
- b. Select the *Medium Effort* (default) or *High Effort* effort level on the Trial Route form.
- c. (Optional) Select the *Use Routing Guide* option, and specify the name of a guide file in the corresponding field. Trial Route follows the routing regions defined in the guide file and honors the specified pre-routed nets.
- d. Click *OK*.

Alternatively, you can issue the following command:

```
trialRoute -highEffort -guide my_chip.rguide
```

## Routing a Partitioned Design

In a flat design, Trial Route can route through guides, regions, and fences, as long as there are no routing blockages or hard blocks. However, fences are often defined as partitions, which become blocks after the design becomes hierarchical. Once partitions become blocks, the routes are no longer allowed, unless they use a proper feedthrough mechanism, such as inserted buffers or routing feedthroughs.

In channel-based routing designs, all top-level routing use channels to route around partitions. In a partitioned design in which the partitions have not been committed, you can use the Trial Route *-handlePartition* and *-handlePartitionComplex* parameters to force the routing into channels, simulating a channel-based design.

- Issue one of the following commands:

```
trialRoute -highEffort -handlePartition
```

or

```
trialRoute -highEffort -handlePartitionComplex
```

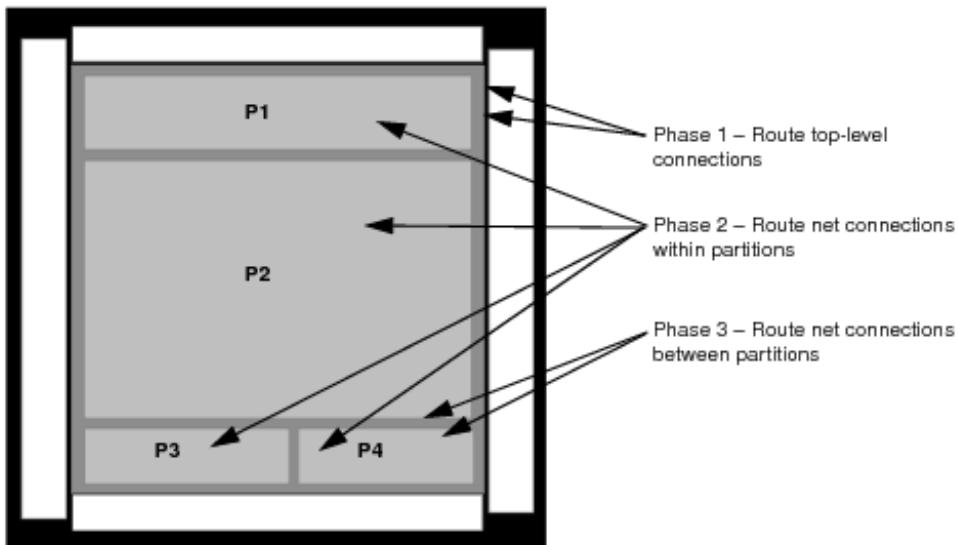
Use the `-handlePartition` parameter to route nets that are only connected within partitions. Use the `-handlePartitionComplex` to route nets that belong to more than one partition, so that the routing does not violate partition boundaries.

When the `-handlePartition` OR `-handlePartitionComplex` parameter is specified, Trial Route works in three phases:

- Phase 1 - Routes the top-level connections
- Phase 2 - Routes net connections within partitions
- Phase 3 - Routes net connections between partitions

For example, the design in Figure 23-1 has five metal layers, a top-level partition and four flattened partitions: P1, P2, P3, and P4. P1 reserves the *metal1*, *metal2*, and *metal3* layers for partitions, and P2, P3, and P4 reserve layers *metal1* through *metal5* for partitions.

**Figure 23-1**



If you specify `-handlePartition` or `-handlePartitionComplex`, Trial Route performs the following tasks to route the net connections:

- a. 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.

- b. Trial Route then routes nets connecting only at the top level. No connections to partitions or cells within the partitions are made.
- c. Trial Route blocks all areas outside of the defined partitions on all routing layers. For P1, Trial Route applies a routing blockage for layers *meta/4* and *meta/5*.
- d. 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.
- e. Trial Route removes the routing blockages and routes the net connections between partitions.  
If an I/O at the top level connects to P2:
  - If you specify -handlePartition, Trial Route uses all metal layers to route through P1.
  - If you specify -handlePartitionComplex, Trial Route uses only layers *meta/4* and *meta/5* 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

**i** 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 pre-route and post-route stage is improved, which results in reduced timing jumps between pre-route and post-route 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 congestions in the design graphically by analyzing the generated congestion map. For more information, on congestion maps, see Using the Congestion Map in [Using the NanoRoute Router](#) chapter of the *EDI System User Guide*.

**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.

## Loading and Saving Route Data

After initially running the Trial Route program, you can load or save Trial Route data at any time during an Encounter session.

- To load 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.

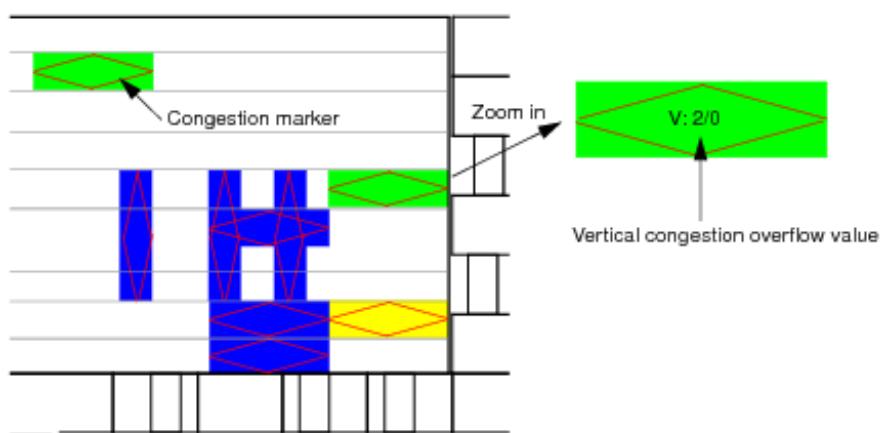
1. Visually check the route congestion markers.  
The red diamond-shaped congestion markers should not be very dense in a local area. These markers contain an overflow value to identify the number of tracks required for that grid, and the actual number of tracks available.  
See Congestion Markers in 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 EDI System 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.

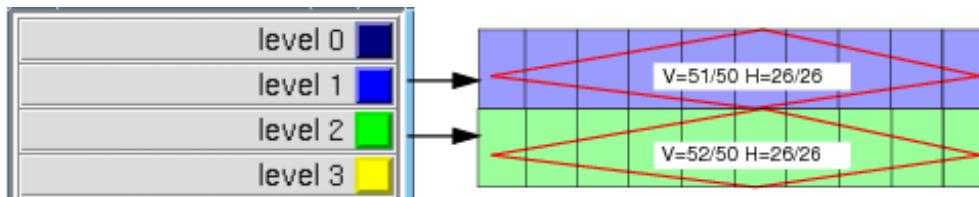
To change the size of super gcells, define the following variable:

```
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 *EDI System Menu Reference*.

## Congestion Distribution Report

After Trial Route completes, a congestion distribution report is created in the EDI System log file. The congestion distribution report provides usage and routing overflow percent values, as well as gcell overflow information (that is, the internal supply and demand for each gcell). There are two types of congestion distribution reports that can be generated: a default congestion distribution report; and a detailed congestion distribution report.

**Note:** The congestion information contained in the congestion distribution report can differ from the congestion information displayed in congestion markers in the EDI System 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)

OvInobst: 21 = 21/5777 (0.4% H) + 0/1587 (0.00% V)

Overflow: 192 = 1 (0.0% H) + 191 (0.07% V)

The Usage statement summarizes horizontal and vertical tracks used in gcells. In the above example, there are 1,734,514 horizontal tracks used in all gcells and 486,668 vertical tracks used in a gcells. Of the available horizontal tracks, 24.2 percent were used for horizontal routing (that is,

wires), and 35.8 percent were used for vertical routing. The total horizontal wire length used equals  $8.65e + 06$  m, and the total vertical wire length used equals  $1.314e + 07$  m.

The `ovInobst` statement summarizes obstructed gcell information. In the example, there are 5,777 horizontally obstructed gcells (where there are no available tracks). The approximate number of wires over horizontally obstructed gcells equals  $21/5,777$ .

The `Overflow` statement summarizes all overflowed gcells. In the example, there is one horizontally overflowed gcell, and 191 vertically overflowed gcells.

### **Gcell Overflow**

The following example illustrates the section of the congestion distribution report that summarizes gcell overflow information. A gcell has overflow if its demand exceeds its supply. Supply is the available routing resource, and demand is the amount of routing resource assigned to the gcell. Typically, the supply is the number of unobstructed tracks crossing the gcell, and the demand is the number of wires assigned to it.

| Remain | cntH   | cntV    |                 |
|--------|--------|---------|-----------------|
| -----  |        |         |                 |
| -6:    | 0      | 0 . 00% | 1 0 . 00%       |
| -5:    | 2      | 0 . 00% | 0 0 . 00%       |
| -3:    | 10     | 0 . 00% | 26 0 . 00%      |
| -2:    | 510    | 0 . 03% | 830 0 . 05%     |
| -1:    | 8100   | 0 . 47% | 17618 1 . 05%   |
| -----  |        |         |                 |
| 0:     | 78504  | 4 . 59% | 178501 10 . 63% |
| 1:     | 102934 | 6 . 02% | 214588 12 . 78% |
| 2:     | 76165  | 4 . 46% | 185168 11 . 03% |
| 3:     | 72832  | 4 . 26% | 179080 10 . 67% |

|    |        |         |        |          |
|----|--------|---------|--------|----------|
| 4: | 81555  | 4 . 77% | 180443 | 10 . 75% |
| 5: | 92704  | 5 . 42% | 158498 | 9 . 44%  |
| 6: | 106167 | 6 . 21% | 137707 | 8 . 20%  |

The following table defines the columns in the congestion report:

| <b>Column</b> | <b>Definition</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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      | ...      |
|--------|---------|----------|---------|----------|--------|---------|----------|
| - 6:   | 0       | 0 . 00%  | 0       | 0 . 00%  | 0      | 0 . 00% | 0        |
| - 5:   | 0       | 0 . 00%  | 0       | 0 . 00%  | 0      | 0 . 00% | 3        |
| - 4:   | 0       | 0 . 00%  | 0       | 0 . 00%  | 0      | 0 . 00% | 5        |
| - 3:   | 0       | 0 . 00%  | 0       | 0 . 00%  | 0      | 0 . 00% | 37       |
| - 2:   | 10      | 0 . 00%  | 28      | 0 . 00%  | 6      | 0 . 00% | 264      |
| - 1:   | 355     | 0 . 00%  | 476     | 0 . 00%  | 229    | 0 . 00% | 1254     |
| <hr/>  |         |          |         |          |        |         |          |
| 0:     | 8855290 | 65 . 73% | 8858305 | 65 . 76% | 70715  | 0 . 52% | 127176   |
| 1:     | 91903   | 0 . 68%  | 269941  | 2 . 00%  | 124321 | 0 . 92% | 460099   |
| 2:     | 166878  | 1 . 24%  | 441388  | 3 . 28%  | 192300 | 1 . 43% | 1052688  |
| 3:     | 250040  | 1 . 86%  | 559047  | 4 . 15%  | 347255 | 2 . 58% | 1109275  |
| 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      | ...    |
|--------|---------|--------|---------|--------|
| <hr/>  |         |        |         |        |
| 0:     | 8855290 | 65.73% | 8858305 | 65.76% |

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     | ...   |
|--------|-------|-------|--------|-------|
| <hr/>  |       |       |        |       |
| 1:     | 91903 | 0.68% | 269941 | 2.00% |

#### ***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    | ...    |
|--------|-------|-------|-------|-------|--------|
| <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% | 1214   |
| <hr/>  |       |       |       |       |        |
| 0:     | 34536 | 0.74% | 83715 | 1.78% | 18240  |
|        |       |       |       |       | 0.14%  |
|        |       |       |       |       | 101470 |
|        |       |       |       |       | 0.75%  |

|    |        |       |        |        |        |       |         |       |
|----|--------|-------|--------|--------|--------|-------|---------|-------|
| 1: | 91903  | 1.98% | 269941 | 5.75%  | 124321 | 0.93% | 460099  | 3.42% |
| 2: | 166878 | 3.59% | 441388 | 9.40%  | 192300 | 1.43% | 1052688 | 7.83% |
| 3: | 250040 | 5.38% | 559047 | 11.90% | 347255 | 2.59% | 1109275 | 8.25% |

### ***Track usage in gcells***

This section summarizes the percentage of tracks used for routing per gcell for each layer. It also reports the average number of tracks used in a gcell on each layer. The following example illustrates this section of the congestion distribution report:

Table for track usage in Gcells

| Usage(%)          | All M    | M2      | M3      | M4      | M5       | ... |
|-------------------|----------|---------|---------|---------|----------|-----|
| #Gcells avg trks: |          | 3.2     | 1.7     | 8.5     | 5.0      |     |
| <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 *metal2* are used.

### ***Real wire length***

This section summarizes the total real wire length used and number of vias for each layer in the section. The following example illustrates this section of the congestion distribution report:

\*\*\*Real Wire Length:

Total: 1.396e+08um, total number of vias: 3547270

M1(V): 9.076e+04um

M2(H): 1.490e+07um, number of M1/M2 vias: 1560437

M3(V): 1.652e+07um, number of M2/M3 vias: 1273925

M4(H): 3.241e+07um, number of M3/M4 vias: 360881

M5(V): 4.233e+07um, number of M4/M5 vias: 242047

M6(H): 2.492e+07um, number of M5/M6 vias: 75288

M7(V): 5.399e+06um, number of M6/M7 vias: 26668

M8(H): 3.013e+06um, number of M7/M8 vias: 8034

## Improving Route Congestion

For a module or submodule that has route congestion, complete one of the following actions to improve congestion, depending on the type and severity of the violation:

1. Change the block pin orientation.

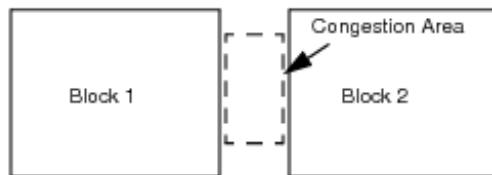
Route congestion usually occurs around blocks that have their pins facing incorrectly. You can identify these blocks by clicking on them in the design display area to see where the flight lines terminate. When the block pins are visible, you can rotate or flip the block(s) to alleviate the congestion.

For information on changing block orientation, see [Flip/Rotate Instances in the Floorplan Menu](#) chapter of the EDI System 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 EDI System software issues a warning message. To display warning messages, set [`setTrialRouteMode -printWiresOutsideBusguide`](#) parameter to `true` (or specify [`trialRoute -printWiresOutsideBusguide`](#)).

For more information on creating bus guides, see "Bus Planning" chapter in the *EDI System User Guide*.

The following limitations exist for the Trial Route bus guide feature:

- Bus bit ordering is not guaranteed; it depends on the pin ordering and topology requirements.
- Bus guides must be complete, and must cover the pins of the net they are to guide.

## Additional Information

## Wire Overlap

In certain situations, wire overlap can occur when using Trial Route.

A wire can overlap a routing blockage boundary if the blockage only partially covers the gcell. The covered tracks are counted as blocked tracks that are not available during global routing. However, Trial Route does not record the exact track location, which can result in wires being placed on a track which is already occupied by a routing blockage.

When using nondefault rules, wire width and space are considered during the global routing phase for congestion calculation, before track assignment. Because they are not considered during track assignment, overlapping nondefault wires can occur. However, because spacing was considered during congestion calculation, the routing congestion information is correct.

---

## Using the NanoRoute Router

---

- [About NanoRoute Routing Technology](#)
- [Routing Phases](#)
  - [Global Routing](#)
  - [Detailed Routing](#)
- [NanoRoute Router in the EDI System 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 EDI System Menu Commands and Forms](#)
  - [Running the NanoRoute Router with EDI System 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 EDI SystemText Commands](#)
  - [Using the EDI System 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](#)
  - [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](#)
- [Using a Design Flow that Includes Astro or Apollo](#)
- [Troubleshooting](#)

## About NanoRoute Routing Technology

The NanoRoute® router performs concurrent signal integrity, timing-driven, and manufacturing aware routing (SMART routing) of cell, block, or mixed cell and block level designs. The router is optimized for routing designs with the following features:

- More than 300K instances or nets and at least five routing layers
- 180 nanometer or smaller process technology
- Signal integrity critical
- Timing critical
- Detailed-model (full-model) abstracts

**Note:** The WRoute router is also included in the EDI System 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 EDI System 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 EDI System database.

## NanoRoute Router in the EDI System Flow

The NanoRoute router is part of the block implementation and the top-level implementation stages of the EDI System 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 EDI System. Before running the router, ensure your design meeting the following conditions:

- It is fully placed and the placement is legal, without any overlaps.  
Use the [checkPlace](#) command to check for overlaps.
- (Optional) Run the [verifyGeometry](#) command and fix any problems. In general it is easier to fix geometry problems before routing than after routing.
- Power is routed.  
Use the [sroute](#) command to route power structures.

## Checking Your LEF Files

You can avoid violations and save time if you ensure your LEF files are optimized for routing. Check the following statements and edit the files with a text editor if necessary:

- **MINSIZE**  
The router does not support specifying MINSIZE without specifying AREA. MINSIZE allows a geometry that is smaller than AREA.
- **UNITS**  
The router does not support a value of 100 for DATABASE MICRONS in the UNITS statement. If the LEF file specifies DATABASE MICRONS 100, issue the following command before you import the design:  
`setImportMode -minDBUperMicron 1000`
- **MANUFACTURINGGRID**  
The router requires that you define the manufacturing grid.
- **MACRO**  
To improve pin access, ensure that all standard cell macros are defined as CLASS CORE.  
  
You must use real shapes, not block-style abstracts, for the shapes on the layers where you expect the router to connect to pins of standard cell macros.
- **VIA**  
The TOPOFSTACKONLY keyword is unnecessary if there are LEF LAYER AREA statements, because the router automatically derives TOPOFSTACKONLY vias based on the AREA statements. If a default via satisfies the AREA statement, the router tags it internally as a TOPOFSTACKONLY via.  
  
If there is no AREA statement for a routing layer, the router looks for TOPOFSTACKONLY vias that go

up to the next metal layer. If `TOPOFSTACKONLY` vias exist, it derives the AREA rule from those vias—the smallest area of the bottom layer metal of all such vias becomes the AREA rule. This feature provides backward compatibility with LEF files that do not have AREA rule support.

Related Topics

- [Unsupported LEF and DEF Syntax](#) in the *EDI System User Guide*
- LEF Syntax chapter of the LEF/DEF Language Reference.

## Checking for Problems with Cells, Pins, and Vias

- Make sure that all power and ground pins in the `SPECIALNETS` section of the DEF file are marked `+ USE POWER` or `+ USE GROUND`.

- Overlapping cells

Overlapping cells make pins short each other and create violations on `metal1`. Check for overlaps by using one of the following commands:

- [verifyGeometry](#) `-wireOverlap`
- [checkPlace](#)

- Pins underneath power routes

Pins that are underneath power routes are inaccessible and cause violations on `metal1` and `metal2`. 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 *EDI System 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. For more information, see the [EDI System Library Development Guide](#).

## Generating Tracks

In the EDI System 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 *EDI System Text Command Reference*.
- For information on importing DEF files, see [Import and Export Commands](#) in the *EDI System 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 prompts you to confirm that you want to interrupt the command.

- To confirm, type `Y`.  
The software returns you to the EDI System prompt and the command does not continue to run.

 When you interrupt routing with `Ctrl-C`, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.

- To continue running the command, type N.

## Using the routeDesign Supercommand

The recommended Cadence design flows use the [routeDesign](#) command to run global and detailed routing and to optimize vias and wirelength after routing.

routeDesign honors the `setNanoRouteMode` and `setAttribute` settings and has the following advantages over using the `globalRoute` and `detailRoute` or `globalDetailRoute` commands:

- It runs SMART routing by default; that is, it runs in both timing- and signal integrity-driven mode by default.  
The other routing commands are not timing- or signal-integrity driven by default, but you can use the following `setNanoRouteMode` parameters to turn on timing- and signal-integrity-driven routing for those commands:
  - `-routeWithTimingDriven true`
  - `-routeWithSiDriven true`
- It changes the status of clock nets from `FIXED` to `ROUTED` so it can modify them during routing and routes them before routing other nets.
  - To keep clock nets' status `FIXED`, run the following command before running `routeDesign`:  
`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 *EDI System Text Command Reference* :

- [detailRoute](#)
- [globalDetailRoute](#)
- [globalRoute](#)
- [routeDesign](#)
- [setAttribute](#)
- [setNanoRouteMode](#)

## Results

The NanoRoute router outputs can include the following (depending on the run-time options you set):

- Section in the Encounter log file
- Routed DEF file
- GDSII file
- SDF or SPEF file
  - For information on outputting GDSII and DEF files, see [Importing and Exporting Designs](#).
  - 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 EDI System 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 EDI System 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).

- [getAttributesetAttribute](#)

Display and set net attributes.

- [getNanoRouteModesetNanoRouteMode](#)

Display and set run-time options for the router.

- [globalRoutedetailRouteecoRouteglobalDetailRouterouteDesign](#)

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:

| <b>Characteristic</b>             | <b>Attributes</b>                                                                                                                                                                                                                                                                 | <b>Options</b>                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Application                       | Apply locally to a net object                                                                                                                                                                                                                                                     | Apply globally to a process or command                                                                                                                                                                                                                                                                                                                                                                                       |
| Specification                     | <ul style="list-style-type: none"> <li>▪ NanoRoute/Attributes form</li> <li>▪ <code>setAttribute</code> command</li> <li>▪ Some attributes can only be specified by <code>setAttribute</code>.</li> </ul>                                                                         | <ul style="list-style-type: none"> <li>▪ NanoRoute form</li> <li>▪ <code>setNanoRouteMode</code> command</li> <li>▪ Some options can only be specified by <code>setNanoRouteMode</code>.</li> </ul>                                                                                                                                                                                                                          |
| Persistence                       | Saved with the database.<br>When you set an attribute and save the database and exit, the attribute setting is saved. If you reload the database, the object retains the attribute setting.                                                                                       | Saved with the database.<br>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.                                                                                                                                                                                                                                                |
| Format                            | <ul style="list-style-type: none"> <li>- <i>attribute_name</i></li> <li>▪ Example:<br/>-<i>avoid_detour</i></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>-<i>optionName</i></li> <li>▪ Example:<br/>-<i>drouteAutoStop</i></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 <a href="#">getAttribute</a> command                                                                                                                                                                                                                                      | Use the <a href="#">getNanoRouteMode</a> command                                                                                                                                                                                                                                                                                                                                                                             |
| More information available at ... | <a href="#">getAttribute</a> in the <i>EDI System Text Command Reference</i>                                                                                                                                                                                                      | <a href="#">setNanoRouteMode</a> in the <i>EDI System 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

EDI System 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 EDI System 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

#### *EDI System User Guide*

- [Multiple-CPU Commands](#) chapter in the *EDI System Text Command Reference*
- Multiple CPU Processing form in the [Options Menu](#) chapter of the *EDI System 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 EDI System 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 EDI SystemText Commands

The following commands show the basic routing strategy using the EDI System 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 EDI System 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*

The router builds the initial detailed routing database, but does not do any search and repair during this step.

3. Click *OK* .
4. Save the design as `droute0`.
5. Check the violations in the log file.

If you have many violations on *metal1* and *metal2*, you probably have pin-access problems, incorrect track settings, or overlapped cells. Check your LEF file and correct any problems.

See [Evaluating Violations](#) for an excerpt of a log file from a design with many violations on *metal1* and *metal2*.

For information on the LEF file, see the LEF/DEF Language Reference or the [EDI System Library Development Guide](#).

### Run Search and Repair

Break search and repair into two phases. Check congestion after each phase and repair violations.

To run the first phase of search and repair, complete the following steps:

1. Choose *Route - NanoRoute - Route* .
2. Set the following options on the NanoRoute form:
  - *Detail Route*
  - *Start Iteration 1*
  - *End Iteration 1*

During this phase, the router makes local changes to the database. It does not do detailed or global routing.
3. Click *OK* .
4. Save the design as `droute1`.
5. Check the violations in the log file and graphically.

To run the second search-and-repair phase, complete the following steps:

1. Choose *Route - NanoRoute - Route* .
2. Set the following options on the NanoRoute form:
  - *Detail Route*
  - *Start Iteration 2*
  - *End Iteration 19*
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 EDI System log file and the congestion map in the EDI System 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:

| Congestion Analysis: |                            |                            |                            |                            |                                        |
|----------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------------------|
|                      | OverCon<br>#Gcell<br>Layer | OverCon<br>#Gcell<br>(1-2) | OverCon<br>#Gcell<br>(3-4) | OverCon<br>#Gcell<br>(5-6) | OverCon<br>%Gcell<br>OverCon<br>(7-12) |
| Metal 1              | 22(0.01%)                  | 10(0.00%)                  | 0(0.00%)                   | 0(0.00%)                   | (0.01%)                                |
| Metal 2              | 5531(2.39%)                | 1680(0.73%)                | 370(0.16%)                 | 123(0.05%)                 | (3.33%)                                |
| Metal 3              | 4114(1.78%)                | 19(0.01%)                  | 0(0.00%)                   | 0(0.00%)                   | (1.79%)                                |
| Metal 4              | 1333(0.58%)                | 137(0.06%)                 | 0(0.00%)                   | 0(0.00%)                   | (0.64%)                                |
| Metal 5              | 5852(2.53%)                | 4(0.00%)                   | 0(0.00%)                   | 0(0.00%)                   | (2.53%)                                |
| Metal 6              | 27(0.01%)                  | 0(0.00%)                   | 0(0.00%)                   | 0(0.00%)                   | (0.01%)                                |
| Total                | 16879(1.22%)               | 1850(0.13%)                | 370(0.03%)                 | 123(0.01%)                 | (1.39%)                                |

#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 #Gcell1, list the number and percentage of gcells on each layer that are over-congested.
- The numbers in parentheses after overCon #Gcell1 indicate how many additional tracks within the gcell are needed to accommodate the global routing assignments. For example, overCon #Gcell1 (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 #Gcell1 (7 - 12).
- The NanoRoute router displays the maximum number of tracks needed in the last overCon #Gcell1 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 #Gcell1 (7-14). If the maximum number of tracks needed were only eight, the column would be labelled overCon #Gcell1 (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 Encounter software displays the congestion map in the main window when you complete the following steps:

1. Globally route the design.
2. Select *Physical view* in the *Views* area of the Encounter main window.
3. 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*.
7. Click *Apply*.

For more information on selecting the objects and colors, see [The Main Window](#) in the *EDI System Menu Reference*.

#### Interpreting the Congestion Map

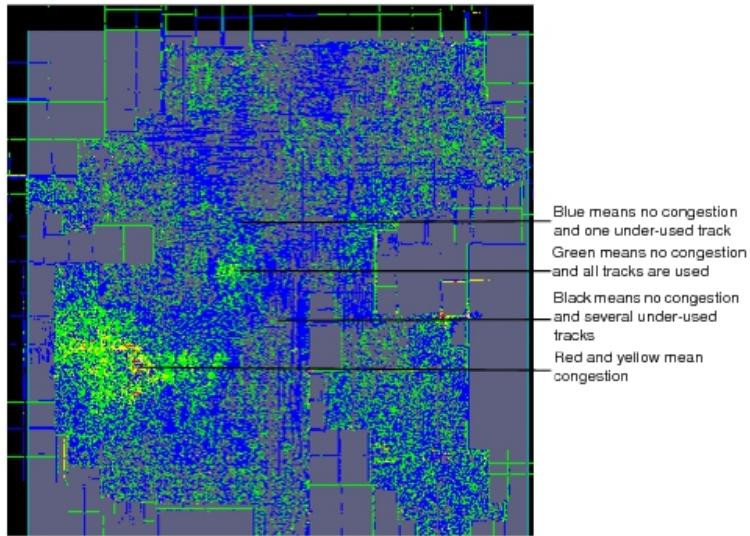
In the map, blue or black indicate an acceptable level of congestion; white indicates an unacceptable level. However, this depends on your design. For example, a design that is mostly uncongested might have small areas (often called hot spots) that are highly congested. You must look at the overall

congestion graphically to assess routability.

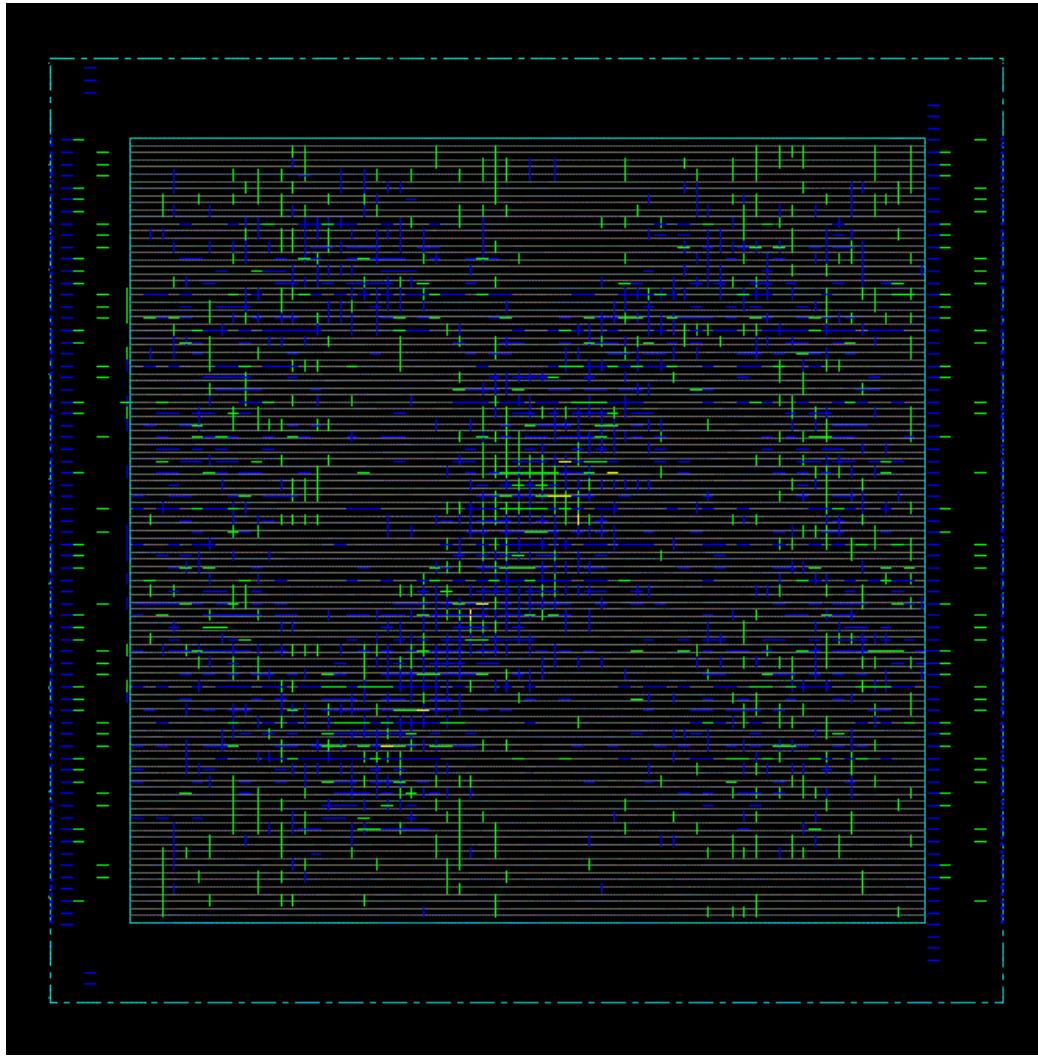
The following table explains the meaning of the default colors in the congestion map:

| 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 EDI System 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 EDI System 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 EDI System 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 EDI System 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 EDI System 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 - Release 13.1.

- Pins that are not aligned with routing tracks  
Align pins with routing tracks to assure the maximum number of pickup points. For more information, see the [NanoRoute Ultra Guidelines](#) chapter in the *EDI System Library Development Guide*.
- Pins that are above or underneath power stripes on the adjacent metal layer  
The router might not be able to access a pin if it is blocked by a power stripe.

For more information, see [verifyTracks](#) in the *EDI System Text Command Reference*.

## Resolving Additional Open Net Problems

If the router generates an open net message after you correct the problems reported by `verifyTracks`, or if `verifyTracks` does not report any problems, check for the following additional problems:

- Pin modelling or library problems
  - Pins without physical geometry
  - Pins that are less than the minimum width
  - Minimum-width pins that are placed off the manufacturing grid
  - Pins that are blocked for planar access, and are not accessible through a via without violating the adjacent-cut rule
  - Pins that trigger multiple-cut vias, but no multiple-cut vias are specified in the LEF file
- Floorplanning problems
  - Cell overlaps introduced during placement  
Use the `checkPlace` command to check for cell overlaps. For information, see [checkPlace](#) in the *EDI System Text Command Reference*.
- Problems caused by `setNanoRouteMode` option settings or conflicts between option settings and library specifications
  - No via access in pin but - `routeWithViaOnlyForStandardCellPin true` is specified
  - No via access in pin but - `routeBottomRoutingLayer` is too high or - `routeTopRoutingLayer` is too low for the router to connect without using a via
  - Via stacking is not allowed but - `routeBottomRoutingLayer` is higher than the pin layer (or - `routeTopRoutingLayer` is lower than the pin layer) so via stacking is required to reach the pin
- Problems caused by missing power or ground routing
  - Missing special routes for stripes or followpins to connect tie-high or tie-low nets causes open power or ground nets during global routing.

The global router issues open net warning messages such as the following:

```
#WARNING (NR) There is no prerouted stripe wire within routing layer range
1:9 for special net VSS.
#WARNING (NR) Please reroute special net wires before running NR.
```

## Running Timing-Driven Routing

In the Encounter environment, during timing-driven routing, the router uses the Common Timing Engine (CTE) by default. All the related tasks (route estimation for the timing graph, capacitance extraction, timing analysis, timing graph generation) are executed within the EDI System 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

For information on the timing constraints that are compatible with the EDI System CTE, see [Data Preparation](#).

- Extended capacitance table generated by the EDI System software
- Netlist in DEF or Verilog format
- Placed design in DEF

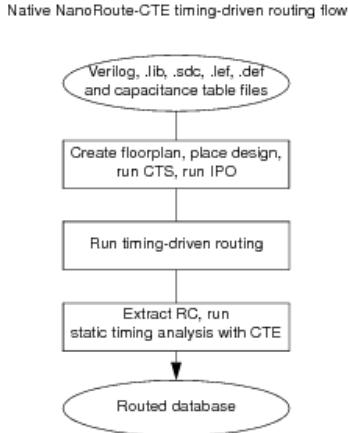
### Using the CTE and the NanoRoute Router in Native Mode

Figure 24-1 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 24-1**



### Using the CTE and Standalone NanoRoute

Figure 24-2 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 EDI System software and use the EDI System `writeDesignTiming` command.

```
writeDesignTiming design.tif
```

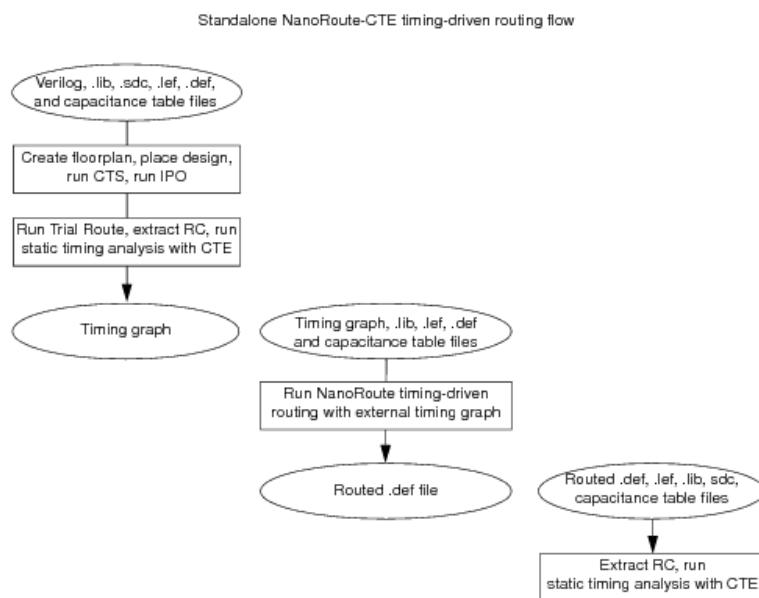
2. Routing

When you route the design, type the following commands:

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

3. Extracting capacitance and analyzing timing

**Figure 24-2**



## Routing Clocks

Route clock nets before routing the rest of the signal nets. If you are using the `routedesign` supercommand, the NanoRoute router changes the status of clock nets from `FIXED` to `ROUTED` so they can be moved and routes them before routing other nets.

This section gives additional information on you can use to route clocks manually.

Layer assignments for clock nets might not correlate in global and detailed routing. For tight control over clock timing, run global and detailed routing on clock nets before routing other nets. Fix the locations of the nets during detailed routing and unfix them during postroute optimization. Use net weights to ensure priority during search and repair.

## Setting Attributes for Clock Nets

If clock nets are marked USE\_CLOCK in the DEF file or you have defined a clock net in the Encounter database, the router automatically sets the following values. You can change the values by setting attributes on the NanoRoute Attributes form. If the clock nets are not defined, type the name of a clock net in the *Net Name(s)* text box to set attributes for the net.

- *Weight*

The default net weight for clock nets is 10 to give clock nets priority during global routing (the default net weight for other nets is 2). During global routing, the router goes from global routing cell to global routing cell within each switch box, and routes the nets with the highest weight first.

- *Bottom Layer*

The default bottom layer for routing clock nets is 3, to ensure that the router has access to *metal1* pins during routing. This attribute sets a soft limit, and the router might route some nets on lower layers, if necessary to complete the routing.

- *Top Layer*

The default top layer for routing clock nets is 4. This attribute sets a soft limit, and the router might route some nets on lower layers, if necessary to complete the routing.

- *Avoid Detour*

*Avoid Detour* is *True* for clock nets, so they are routed as straight as possible.

Set the following attribute in the EDI System console, using the `setAttribute` command

- `-preferred_extra_space 1`

`-preferred_extra_space` adds spacing around the clock nets, which improves coupling capacitance. It is not included on the NanoRoute/Attributes form.

For information on `setAttribute -preferred_extra_space`, see "[Route Commands](#)" in the *EDI System Text Command Reference*.

Select *SI Prevention True* to set *Weight*, *Avoid Detour* and `-preferred_extra_space` all at once. *SI Prevention True* sets *Weight* to 10, *Avoid Detour* to *True*, and `-preferred_extra_space` to 1 for clock nets.

## Routing Clock Nets Using the GUI Forms

- Specify the following options on the NanoRoute form:

- *Selected Nets*

Specify *Selected Nets* to route the clock nets first. Unlike the *Weight* attribute, which gives priority to routing nets within a switch box, *Selected Nets* is a global option that routes whole nets.

- *Global Route*

- *Detail Route*

Specify *End Iteration 19* to stop routing before the postroute optimization step.

## Running Postroute Optimization

To prevent rip-up and rerouting of clock nets during postroute optimization, specify the following:

- On the NanoRoute/Attributes form, keep the attributes you have already set, and specify *Skip Routing True*.
- On the NanoRoute form, specify *Start Iteration 20* and *End Iteration default*.

### Related Topics

- [Using the routeDesign Supercommand](#)
- [routeDesign](#) in the "Route Commands" chapter of the *EDI System Text Command Reference*
- "Synthesizing Clock Trees"

## Preventing and Repairing Crosstalk Problems

During SMART routing, the NanoRoute router automatically prevents crosstalk problems by wire spacing, net ordering, minimizing the use of long parallel wires, and selecting routing layers for noise-sensitive nets. The router performs these operations concurrently with other operations.

In addition to the operations it performs automatically, the router also performs shielded routing to protect critical wires from crosstalk.

During postroute signal integrity repair, the router performs these same operations.

The following sections describe the crosstalk prevention and repair operations the router performs, and whether you can set net attributes to control them.

- **Wire spacing**

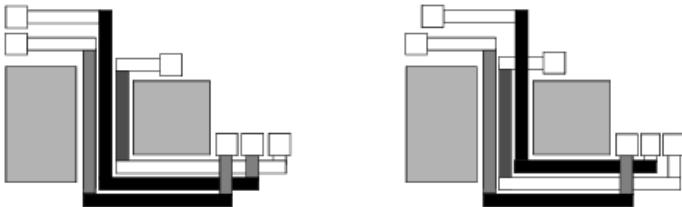
The router automatically adds extra space between critical nets. You can also use the `-preferred_extra_space` attribute to add space. For information on this attribute, see [setAttribute](#) in the *EDI System Text Command Reference*.



- **Net ordering**

The router automatically routes critical nets first and avoids detours on those nets so they are as short as possible.

- You can also use the `-weight` attribute to give priority to critical nets within a switch box, so they are routed first.
- You can use the `-avoid_detour` attribute to ensure that critical nets are routed as short as possible.



- Minimizing the use of long parallel wires

The router automatically minimizes the use of long parallel wires, based on an internal algorithm. You cannot set an attribute to control this feature.



- Selecting routing layers

The router automatically restricts routing layers for critical nets to reduce both coupling and resistance. It routes clocks on layers 3 and 4.

- You can set the `-bottom_preferred_routing_layer` and `-top_preferred_routing_layer` attributes to specify preferred layers for critical nets.
- You can specify how strictly to enforce these attributes by specifying the `-preferred_routing_layer_effort` attribute.



- Shielding

The router can shield critical nets with power or ground wires to protect them from coupling. Shielding is not an automatic operation--you control it with the `-shield_net` attribute.

#### Related Topics

- 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.

Optionally, you can also set the following options:

```
setNanoRouteMode -routeTdrEffort  
setNanoRouteMode -routeSiEffort
```

These options fine-tune the priorities the router assigns to timing, signal integrity, and congestion. Use these options together to minimize crosstalk. After meeting the timing requirements of your design, adjust the values and rerun routing, following these guidelines:

- If your design is congested, use a low timing-driven effort.
- If your design is not congested, use a high timing-driven effort

 Because designs with severe signal-integrity problems are usually not congested, use a high timing-driven effort for those designs.

- If increasing the timing-driven effort creates a jump in the number of timing violations, decrease the timing-driven effort.

For more information on these options, see [setNanoRouteMode](#) in the *EDI System Text Command Reference*.

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
```

**i** 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 EDI System ECO commands and flows, see [Interactive ECO](#).

## Evaluating Violations

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

**Note:** Use the **ctrl-c** key combination to stop search and repair. For information on using **ctrl - c**, see [Interrupting Routing](#).

The router marks four types of violations:

- Horizontal

A violation that falls on a horizontal wire is a horizontal violation.

- Vertical

A violation that falls on a vertical wire is a vertical violation.

- Via

A violation that falls on a via is a via violation. The router places via violation markers on the lower layer of the via, even when the violations are on the upper layer. For example, if the router finds a spacing violation between an M1-M2 via's *metal2* layer and another *metal2* shape, the violation marker on the via will be at *metal1* layer. If your routed design has via violations, the router corrects the violations during search and repair.

- Minimum area rule

If the AREA rule for the layer has not been satisfied, the router marks a minimum area rule violation.

For details on the violations, use the [verifyGeometry](#) command.

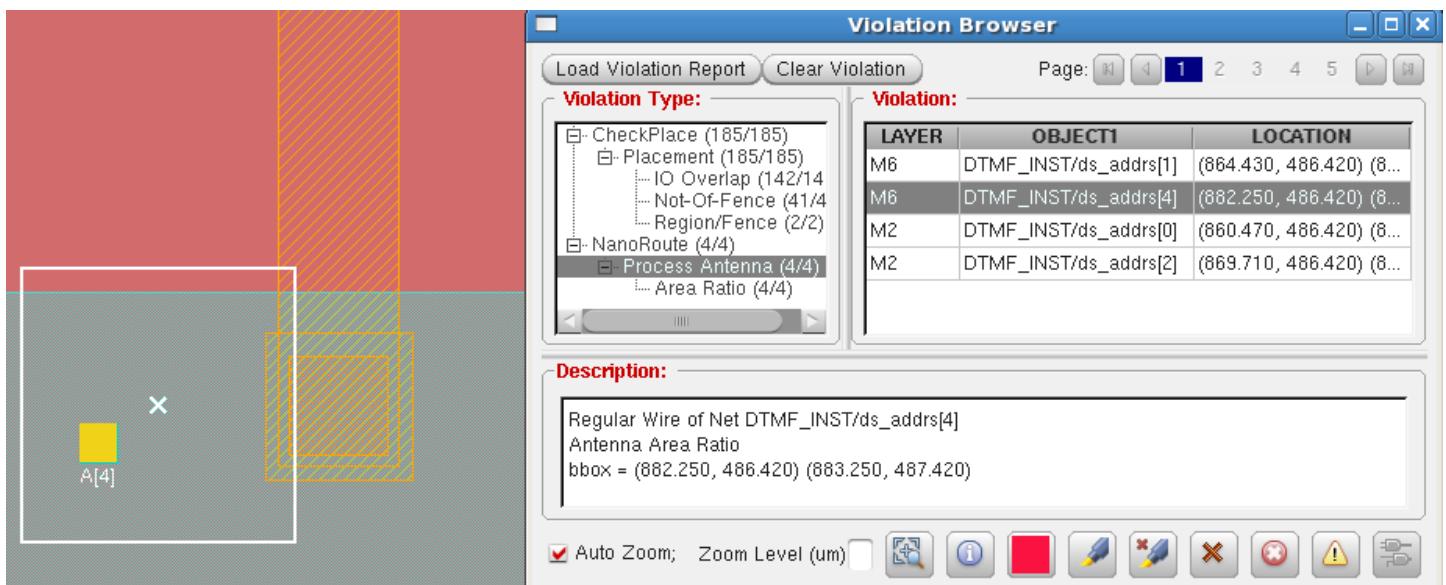
If you import a routed design from standalone mode, the EDI System software imports the violation markers as well. To delete the violations, rerun detailed routing. The software deletes the violations during search and repair.

The following excerpt is from a log file from a design after the nineteenth iteration of search and repair. There are many violations, mostly on layers *metal1* , *metal2* , and *metal6* .

```
#Total number of DRC violations = 1426
#Total number of violations on LAYER Metal1= 876
#Total number of violations on LAYER Metal2= 275
#Total number of violations on LAYER Metal3= 84
#Total number of violations on LAYER Metal4= 38
#Total number of violations on LAYER Metal5= 18
#Total number of violations on LAYER Metal6= 135
#Total number of violations on LAYER Metal7= 0
```

In the design window, violations on different layers are shown by different-colored markers. Violations can occur due to library or design problems, such as overlapping pins, improperly defined tracks, or an insufficient number of rotated vias. The marking box provides full information about the violation including error type, layer name, net object, location, and a description in the violation browser.

**Figure 24-3**



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

| Violations/Warnings                                                                                                                  | Check for ...                                                                                                                                                                                                                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Many violations on <i>metal1</i>                                                                                                     | <ul style="list-style-type: none"> <li>▪ Improper setup of routing tracks</li> <li>▪ Overlapping cells</li> <li>▪ Insufficient via rotation, causing inaccessible pins</li> </ul> <p>For information on fixing these problems, see <a href="#">EDI System Library Development Guide</a>.</p>                                                       |
| Many violations on <i>metal1</i> and <i>metal2</i>                                                                                   | <ul style="list-style-type: none"> <li>▪ Offgrid pins</li> <li>▪ Overlapping tracks</li> <li>▪ Overlapping cells</li> <li>▪ Improper X offset, causing offgrid standard cell pins</li> <li>▪ Pins buried under power routing</li> </ul> <p>For information on fixing these problems, see <a href="#">EDI System Library Development Guide</a>.</p> |
| Open net warning messages<br><br>An open net is one that the router cannot route because it cannot complete the connection of a net. | <ul style="list-style-type: none"> <li>▪ Pin modelling, track definition, data, floorplanning problems</li> <li>▪ Conflicts between option settings or options and library specifications</li> </ul> <p>For information on fixing these problems, see <a href="#">Resolving Open Nets</a>.</p>                                                     |

|                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Violations not decreasing after first iteration of detailed routing                                                                                                                                                                                                                                                                              | <ul style="list-style-type: none"> <li>▪ Offgrid pins</li> <li>▪ Overlapping tracks</li> <li>▪ Overlapping cells</li> <li>▪ Improper X offset, causing offgrid standard cell pins</li> <li>▪ Pins buried under power routing</li> </ul> <p>For information on fixing these problems, see <a href="#">EDI System Library Development Guide</a>.</p>                                                                                                                                                                                |
| Violations on upper layers, such as via-to-wire violations or shorts                                                                                                                                                                                                                                                                             | <ul style="list-style-type: none"> <li>▪ Improper routing pitch (not line-to-via)</li> </ul> <p>For information, see <a href="#">Violations on Upper Metal Layers</a>. For more information on correcting the routing pitch, see <a href="#">EDI System Library Development Guide</a>.</p>                                                                                                                                                                                                                                        |
| Localized congestion (also called hot spots) --areas in the congestion map that are red, magenta, or white. The congestion might be caused by one of the following: <ul style="list-style-type: none"> <li>▪ Limited pin access to a block</li> <li>▪ Congestion around the corner of a block and in the middle of the standard cells</li> </ul> | <ul style="list-style-type: none"> <li>▪ Improper placement or floorplanning</li> </ul> <p>For information on placement see <a href="#">Placing the Design</a>. For information on floorplanning, see <a href="#">Floorplanning the Design</a>.</p>                                                                                                                                                                                                                                                                               |
| False violations                                                                                                                                                                                                                                                                                                                                 | <ul style="list-style-type: none"> <li>▪ Violation markers on the lower metal layer of a via, even though the actual violation is on the upper layer.</li> </ul> <p>When it flags a via violation, the router places the violation marker on the lower metal layer of the via, whether the actual violation is due to a problem on the lower layer or the upper layer. To repair the violation, rerun detailed routing. The router finds and repairs the violation, even when the marker was reported on the incorrect layer.</p> |

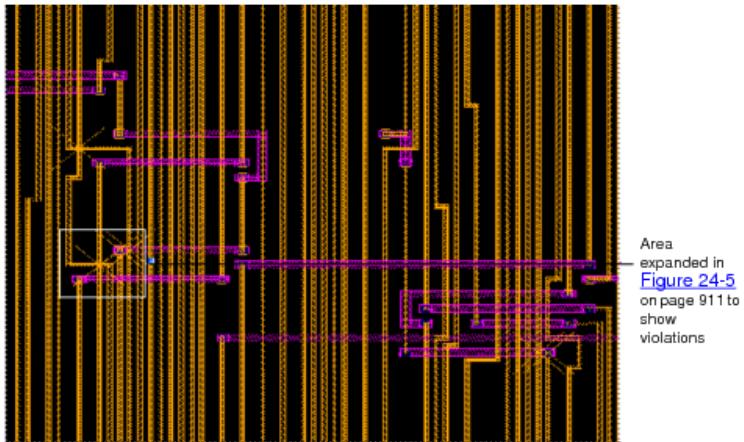
## Violations on Upper Metal Layers

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:

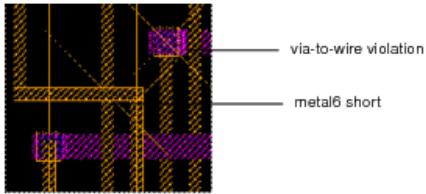
- Via-to-wire violations
- Shorts

Figure 24-4, Figure 24-5, and the LEF and DEF file excerpts that follow show a design with many violations on *meta/6*.

**Figure 24-4**



**Figure 24-5**



The relevant LEF file excerpt is:

```
LAYER Metal6
  TYPE ROUTING ;
  PITCH 0.46 ;
  WIDTH 0.2 ;
  SPACING 0.21 ;
  DIRECTION VERTICAL ;
END Metal6
LAYER Metal7
  TYPE ROUTING ;
  PITCH 0.82 ;
  WIDTH 0.4
  SPACING 42 ;
  DIRECTION HORIZONTAL ;
END Metal7
VIA via6 DEFAULT
  LAYER Metal6 ;
  RECT -0.19 -0.23 0.19 0.23 ;
  LAYER Via6 ;
  RECT -0.18 -0.18 0.18 0.18 ;
  LAYER Metal7 ;
  RECT 0.29 -0.2 0.29 0.2 ;
  RESISTANCE 0.68
END via6
```

The relevant DEF file excerpt is:

TRACKS X -4749270 D0 6324 STEP 460 LAYER Metal6

To repair the shorts and via-to-wire violations, align the tracks as much as possible without sacrificing them. Change the TRACKS statement in the DEF file to have at least line-to-via STEP (pitch).

The line-to-via calculation for *meta/6* is:

$$\begin{aligned}\text{Line to via metal6} &= 1/2 \text{ Width} + \text{Spacing} + 1/2 \text{ Via} \\ &= 0.1 + 0.21 + 0.19 \\ &= 0.5\end{aligned}$$

## Violations in Timing-Driven Routing

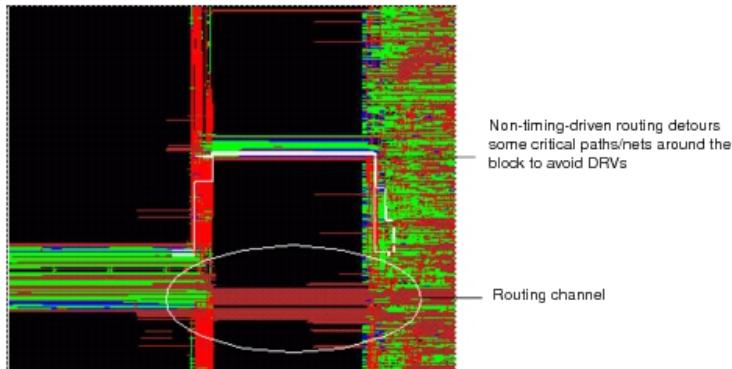
Run time and the number of violations often increase during timing-driven routing because the router restricts the routing of timing-critical nets.

During non-timing-driven routing, the router might detour some nets in order to avoid creating violations. In timing-driven mode, however, the router does not detour timing-critical nets. Instead, it forces them to be routed as short as possible, which can create congestion in the channels. Later, when design-rule checking takes precedence, the router detours timing-critical nets in overly congested channels.

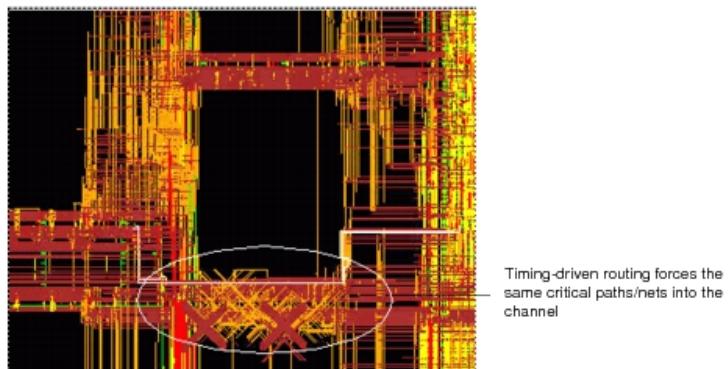
For information on the timing-driven routing flow, see [Running Timing-Driven Routing](#).

Figure 24-6 and Figure 24-7 illustrate non-timing-driven and timing-driven routing results for the same design.

**Figure 24-6**



**Figure 24-7**



## Deleting Violated Nets

To delete violated nets, use the [editDeleteViolations](#) command. After deleting the nets, use ECO routing, detailed routing, or the `globalDetailRoute` command to re-route the design.

### Related Topics

- [detailRoute](#)
- [ecoRoute](#)
- [globalDetailRoute](#)
- [setNanoRouteMode -routeWithEco](#)

## Using Additional Strategies to Repair Violations

### Process Antenna Violations

Repair process antenna violations with antenna repair options or the wire editing commands.

- For information on verifying process antenna violations, see "Verifying Process Antennas" in [Identifying and Viewing Violations](#) chapter of the *EDI System 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, type the following commands:

```
setNanoRouteMode -drouteUseMultiCutViaEffort {medium | high}  
detailRoute
```

For more information on this parameter, see [setNanoRouteMode](#) in the *EDI System Text Command Reference*.

## Postroute Via Optimization

The NanoRoute router can optimize vias on a fully routed design by replacing single-cut vias with multiple-cut vias or with fat vias (single or multi-cut vias with an extended metal overhang). The router does not replace multiple-cut vias during this step.

The router replaces vias by substituting vias in the following order:

1. Fat double-cut vias
2. Normal double-cut vias
3. Fat single-cut vias

Ensure the following before replacing the vias:

- Double-cut vias and fat vias are automatically generated or defined in the LEF file.  
Use the [generateVias](#) command to generate vias.
- The design is completely global and detailed routed.  
If you delete any wires after routing, reroute the design before replacing the vias.
- The design is free of all DRC violations.

Complete the following steps:

1. To run postroute via reduction, type the following commands:
  - `setNanoRouteMode -drouteMinSlackForWireOptimization slack`
  - `setNanoRouteMode -droutePostRouteMinimizeViaCount true`
  - `routeDesign -viaOpt`

**Note:** When you run these commands, the software replaces all multiple-cut vias with single-cut vias. Use these commands only if no concurrent via optimization was done.
2. To run postroute single-cut to multiple-cut via swapping, complete one of the following steps:
  - a. To run postroute single-cut via to multiple-cut via swapping, type the following commands:
    - `setNanoRouteMode -drouteMinSlackForWireOptimization slack`
    - `setNanoRouteMode -droutePostRouteSwapVia multiCut`
    - `routeDesign -viaOpt`
  - b. To run non-timing-driven postroute single-cut via to multiple-cut via swapping, type the following commands:
    - `setNanoRouteMode -routeWithTimingDriven false`
    - `setNanoRouteMode -droutePostRouteSwapVia multiCut`
    - `routeDesign -viaOpt`

### Related Topics

- Using the `routeDesign` Supercommand
- [routeDesign](#) in the "Route Commands" chapter of the *EDI System Text Command Reference*

## Optimizing Vias in Selected Nets

To optimize vias in selected nets, set the `-skip_routing` attribute to `true` for all nets, then set the `attribute` to `false` for nets with vias you want to optimize.

```
setAttribute -net * -skip_routing true
setAttribute -net ... -skip_routing false
globalDetailRoute
```

## Via Optimization Options

- `-droutePostRouteSwapVia`
- `-dbViaWeight`
- `-drouteUseMultiCutViaEffort`
- `-droutePostRouteMinimizeViaCount`
- `-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 [setNanoRouteMode](#) in the ["Route Commands"](#) in the *EDI System Text Command Reference*.

To minimize the number of vias, specify the following options:

```
setNanoRouteMode -droutePostRouteMinimizeViaCount true
setNanoRouteMode -drouteEndIteration default
```

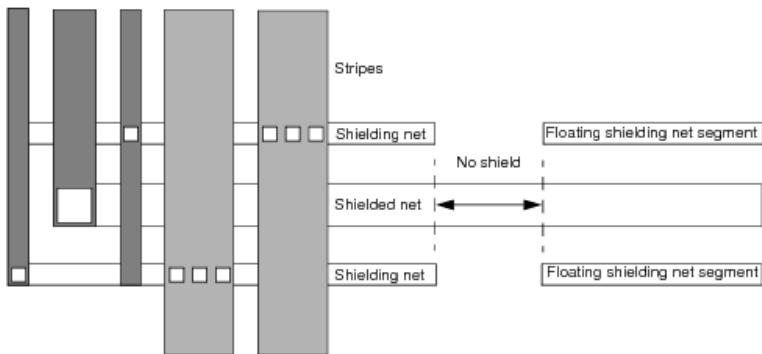
## Performing Shielded Routing

The NanoRoute router can protect noise-sensitive nets, such as a clock nets, from crosstalk by shielding them with power or ground wires. You typically route shielded nets before routing other nets. At the end of routing, the router deletes shielding wires that are not connected to power or ground wires. NanoRoute automatically generates a shielding statistics report after routing. For information on the report, see [Interpreting the Shielding Report](#).

Figure 24-8 shows a section of a design with a shielded net. In the figure,

- The signal net is shielded by a power net on one side and a ground net on the other side.
- Multiple vias can be dropped where a stripe crosses the shielding net at a right angle, if the stripe is wide enough to accommodate them.
- A segment of the signal net is not shielded.
- There are some floating shielding net segments.

**Figure 24-8**



## 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 shield 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 Encounter `selectNet` command to specify the net to shield. It must be the same as the net you specified on the NanoRoute/Attributes form.
6. From the main menu, choose *Route - NanoRoute - 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.

You can also skip routing on prerouted nets by issuing the following command:

```
setAttribute -net @PREROUTED -skip_routing true
```

@PREROUTED applies to a net that has any wiring, including partial wiring.

2. On the NanoRoute form, deselect *Selected Nets Only*.
3. Click *OK* or *Apply* to reroute the design.

## Performing Shielded Routing Using Text Commands

- The following commands shield net1 with both power and ground wires, and shield net2 with a ground wire:

```
setAttribute -net net1 -shield_net vdd \
             -shield_net vss
setAttribute -net2 -shield_net vss
globalDetailRoute
```

- The following commands show how to shield two nets (do not shield more than one net with the same command):

```
setAttribute -net net1 -shield_net abc_gnd
setAttribute -net net2 -shield_net abc_gnd
```

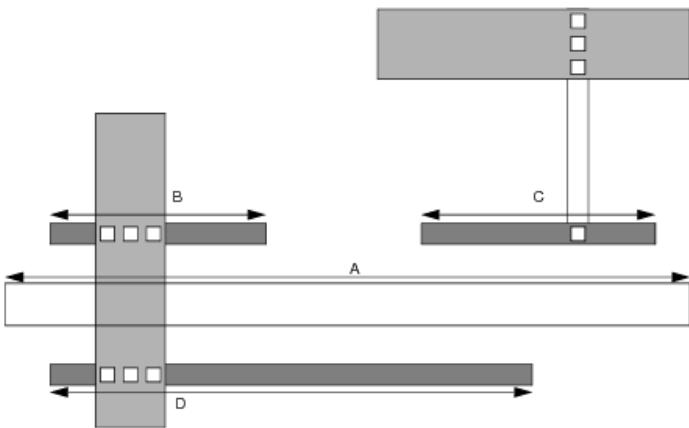
## Interpreting the Shielding Report

The software generates a shielding report for the NanoRoute router when you run the [reportShield](#) command. You can customize the report to output information on the whole design or on selected nets, and you can report per-layer statistics.

Following is a section of a report:

```
-----
Name      : Shielded net name
Length    : Shielded net length
Shield    : Total length of shielding wire
ratio     : Average shielding ratio
-----
Name      Length   Shield  Ratio    Layer:      Length   Shield  Ratio
-----
netA:
  211.5      378.3    0.894
                METAL2:      5.0       2.2    0.222
                METAL3:    107.4     180.1    0.839
                METAL4:     99.1     196.0    0.989
average:   211.5      378.3    0.894
                METAL2:      5.0       2.2    0.222
                METAL3:    107.4     180.1    0.839
                METAL4:     99.1     196.0    0.989
```

To help understand the report, see the following figure, which shows a section of netA:



In the figure,

|             |                              |
|-------------|------------------------------|
| A           | Represents the shielded net. |
| B, C, and D | Represent shielding wires.   |

The software calculates the shielding ratio by using the following formula:

$$\text{Shielding Ratio} = \frac{B + C + D}{A \times 2}$$

## Routing Wide Wires

The NanoRoute router automatically tapers wide wires when connecting to pins, including input/output pins of standard cells, macro cells, and block output pins. The tapered portion of a wire uses the minimum-width wire (the default width).

If you do not want tapering at the output pins, specify the following parameter:

```
setNanoRouteMode -drouteNoTaperOnOutputPin true
```

**Note:** The NanoRoute router prohibits tapering on top level pins, by default.

## 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.

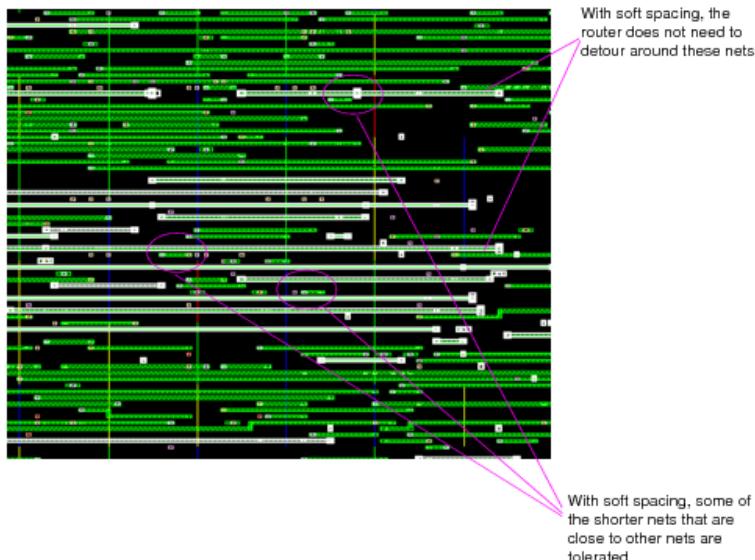
In congested areas, the router might violate nondefault spacing rules in order to avoid design-rule violations and complete the routing. Its flexibility with regard to nondefault spacing decreases the overall wirelength and benefits timing and signal integrity because it allows some shorter nets to be 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 24-9 illustrates nondefault spacing ("soft spacing") routing.

**Figure 24-9**



## Repairing Process Antenna Violations

The NanoRoute router can repair process antenna violations concurrently with DRC violations during the search-and-repair step. During the postroute optimization step, when there are no more DRC violations, the router repairs additional process antenna violations. This two-step methodology allows the router to use more aggressive methods to repair the process antenna violations early on and saves CPU time.

During postroute optimization, the router repairs antenna violations by changing layers (also called antenna stapling or layer hopping). It also repairs process antenna violations by inserting diode cells as close as possible to input gates to discharge current, and deleting and rerouting nets with violations.

**Note:** After routing, run the `globalNetConnect` command to ensure connectivity to power and ground pins in antenna cells added during process antenna repair. For information, see [globalNetConnect](#) in the *EDI System Text Command Reference*.

The router supports hierarchical process antenna calculations and repair.

For information on PAE calculations, and the LEF and DEF antenna parameters, see "Calculating and Fixing Process Antenna Violations" in the LEF/DEF Language Reference.

## Repairing Violations on Multiple-Pin Nets

On multiple-pin nets, the router does the following:

- On a two-pin net that has one input pin with antenna information and one output pin without antenna information, the router tries to repair the antenna violation based on input antenna information only.
- On a two-pin net that has one input pin without antenna information and one output pin with antenna information, there is usually no antenna violation on the output pin, so the router skips antenna repair.
- On a two pin-net where the router does not have any antenna information on either pin, the router skips antenna repair.
- On a three-pin net that has two input pins--one with antenna information and one without antenna information--and one output pin without antenna information, the router skips antenna repair.

## Changing Layers

The router automatically shortens wires whose area exceeds the gate/wire area ratio set in the LEF file. This process might not guarantee that it can resolve all antenna violations--if the routing area is congested, process antenna violations can still occur, just as shorts and spacing violations can occur.

## Using Diodes

The router inserts antenna diode cells or uses preplaced diode cells to repair violations. It can swap filler cells with antenna diode cells and fill the gap automatically if an antenna diode cell is not the same size as the filler cell it replaced. A later routing pass does not remove previously placed diodes.

The antenna diode cells must have the same LEF SITE definition as the standard cells. Specify the diode cell name using the *Diode Cell Name* option on the NanoRoute form or the -routeAntennaCellName option on the text command line.

## Deleting and Rerouting Nets with Violations

If the design has more than 100 DRC violations, and you are using LEF 5.4 or later, the router deletes and reroutes nets with process antenna violations.

## Repairing Violations on Cut Layers

The NanoRoute router detects antenna violations on cut layers and repairs them by inserting diodes. To repair these violations, you must specify a value in your LEF file for the ANTENNADIFFAREARATIO (or ANTENNACUMDIFFAREARATIO) for the cut layers. For each cut layer, the value for ANTENNADIFFAREARATIO (or ANTENNACUMDIFFAREARATIO) must be larger than the value for ANTENNAAREARATIO (or ANTENNACUMAREARATIO).

 If you do not use diodes to repair process antenna violations, the router cannot repair the violations on cut layers.

- 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 *EDI System Menu Reference*.

- To specify the diode cells, use - routeAntennaCellName.

- 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:

- setNanoRouteModeOptions:
  - -drouteFixAntenna
  - -routeAntennaCellName
  - -routeAntennaPinLimit
  - -routeDeleteAntennaReroute
  - -routeFixTopLayerAntenna
  - -routeIgnoreAntennaTopCellPin
  - -routeInsertAntennaDiode
  - -routeInsertAntennaInVerticalRow
  - -routeInsertDiodeForClockNets
- setAttribute -nets netName -skip\_antenna\_fix

## Examples

- The following commands shows the basic strategy for repairing process antenna violations:

```
setNanoRouteMode -drouteFixAntenna true
setNanoRouteMode -routeAntennaCellName "ANTENNA"
setNanoRouteMode -routeInsertAntennaDiode true
globalDetailRoute
globalNetConnect
```

The NanoRoute router runs global and detailed routing. After repairing DRC violations, it repairs as many process antenna violations as it can by layer hopping during postroute optimization. If any process antenna violations remain, the router repairs them by inserting antenna diode cells named ANTENNA.

- The following commands repair process antenna violations by inserting diodes and filler cells. The filler cells are specified by the `setFillerMode -core` command. They fill any gaps that is left when a diode replaces a large filler cell.

```
setNanoRouteMode -routeInsertAntennaDiode true  
globalDetailRoute  
globalNetConnect
```

For information on adding filler cells, see `setFillerMode` and `addFiller` in the ["Placement Commands"](#) chapter of the *EDI System Text Command Reference*.

## Using a Design Flow that Includes Astro or Apollo

The NanoRoute router uses the information in the Milkyway technology file to automatically map vias and layers in the design to Astro scheme format. The router maps only the routing data, so it does not map DEF vias or special routing.

To use Astro or Apollo in your design flow, you must have a LEF technology file that contains layer and via descriptions that match one-to-one with the descriptions in the Milkyway technology file.

 **Check with the foundry before making any changes to the original files.**

You can create a rule file to map layers or vias that are not mapped automatically.

In native mode, the following commands are used in this flow:

- [generateLef](#)  
Converts Milkyway technology file descriptions, Milkyway CLF files, customized LEF technology files from customers, and other older syntax and non-optimal LEF files to a LEF file with target rules and automatically generated vias. Issue this command before routing.
- [defOut](#)  
Outputs a routed database based on the mapping results. Issue this command after routing.

If you are running the router standalone mode, issue the following command:

- `pdi export_design -aef -rule`  
Finds the technology file and automatically maps the layers and vias from the LEF file to a format that Astro can read. Optionally, includes user-created rules for additional mapping.  
Issue this command after routing.

The following command outputs a file named MyoutputFile, using a rule file named tfo.map:

```
pdi export_design -aef -rule tfo.map MyOutputFile
```

If you do not need any additional mapping, the only line in tfo.map is

```
techfile apollo.tf
```

## Troubleshooting

If you have problems with your design, try the following troubleshooting tips:

1. Check the log file for errors and warnings. Correct the problems and continue routing or reroute, as appropriate.

For example, the router might stop routing automatically if it finds too many violations. If the router stops unexpectedly, check the log file for a message that the router has reached the maximum number of violations and the set the following [setNanoRouteMode](#) parameter to false to continue routing:

```
-drouteAutoStop
```

2. Verify connectivity and geometry before and after routing and compare results.

You can also use the [checkRoute](#) command to verify the connectivity. It is faster than [verifyConnectivity](#), but does not output a report.

3. Save the database after routing and restore it in a new session in the EDI System 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.

---

# 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 *EDI System 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 *EDI System 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.

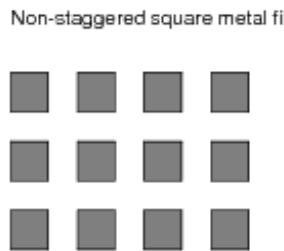
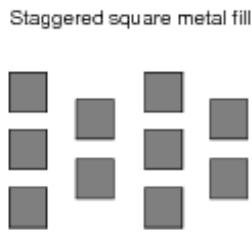
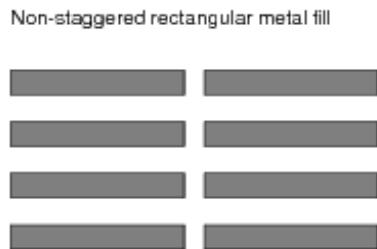
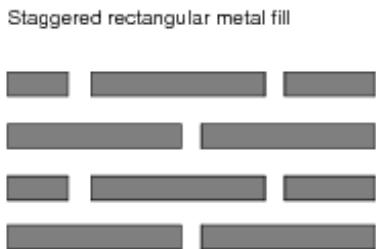
## Metal Fill Features

The metal fill has the following features:

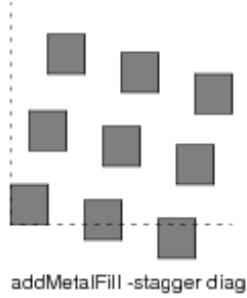
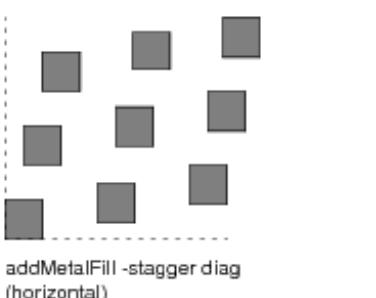
- It can be square or rectangular.
- It can be added in a staggered or non-staggered pattern.
- It can be connected to power or the ground (tied-off) or left unconnected (floating).
- It can be added in timing aware or non-timing aware mode.
- It can be part of the power and ground structure.

## 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  
-net 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:

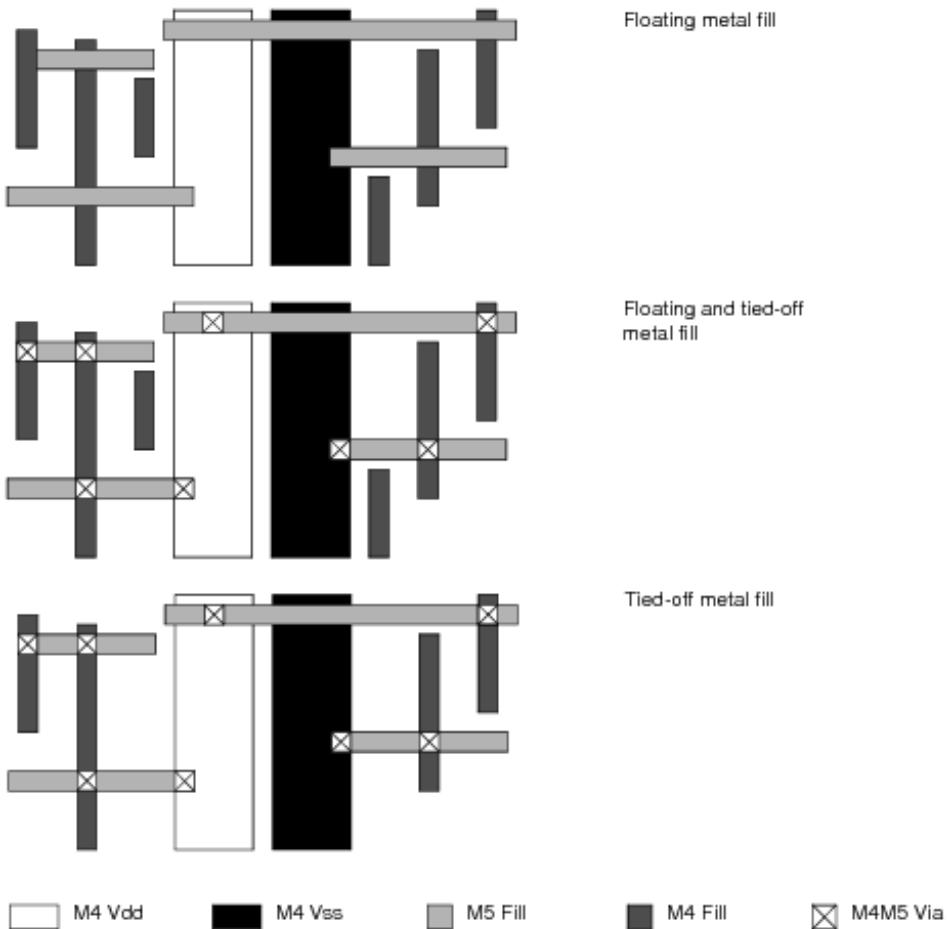
1. Lower cross coupling capacitance, especially if you specify short metal fill segments (long metal fill segments behave like they are really tied off).
2. Easier to trim when there are violations. You can trim a floating metal fill that causes DRC violations with the [trimMetalFill](#) command. If 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.

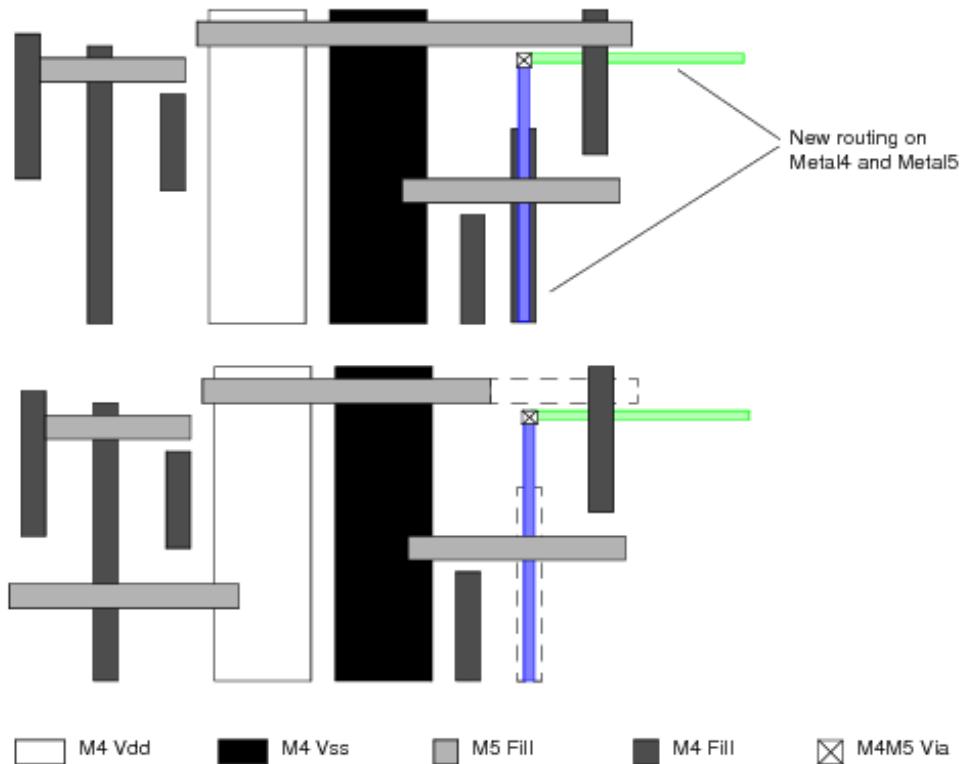
For more information, see the figures below and ["Trimming Metal Fill"](#).

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.

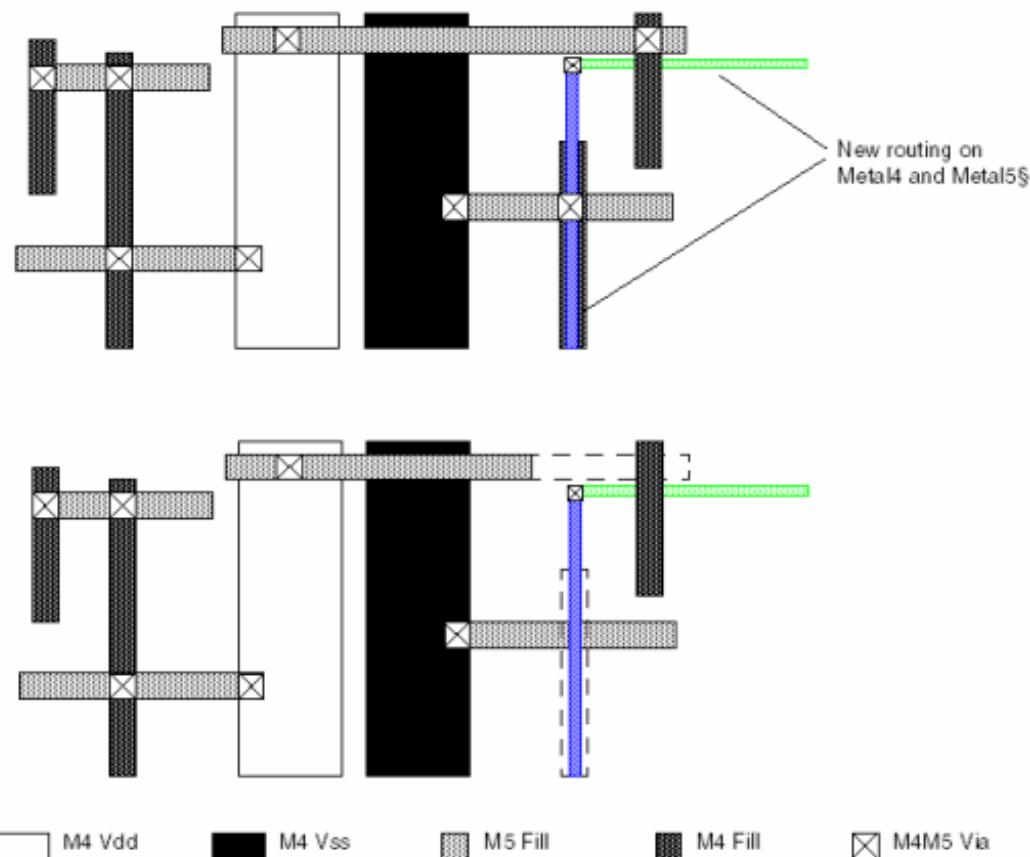


The figures below show the same design after an ECO, in which routing was added on *Metal4* and *Metal5*.

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.



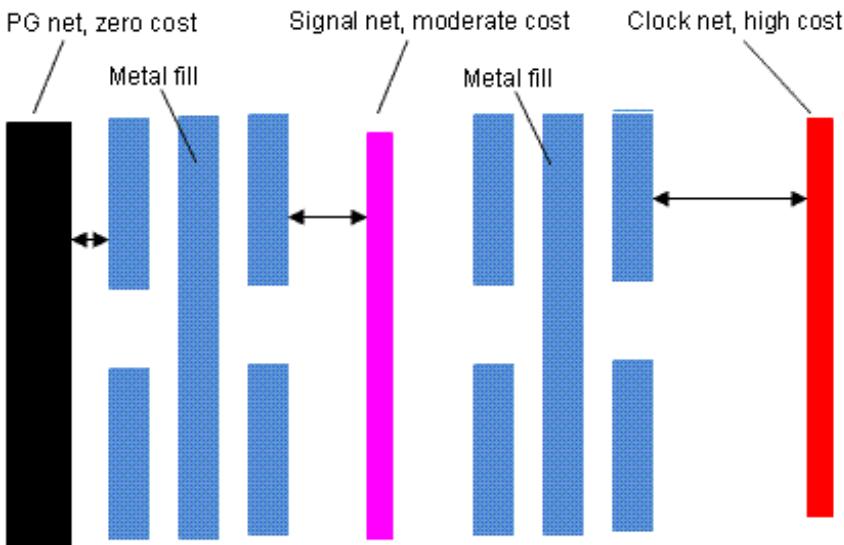
These figures show what happens when you use a tied-off metal fill. The first figure shows the design with the added routing. The second figure shows the design after the metal fill is trimmed. The dotted lines show where the metal fill was trimmed.



## Timing-Aware Metal Fill

When it adds a timing-aware metal fill, the EDI System 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 EDI System 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 EDI System 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 EDI System 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 EDI System software uses to examine metal density | DENSITYCHECKWINDOW | -windowSize | <i>Step Size X</i><br><i>Step Size Y</i>     |
| Distance the window moves for each metal fill iteration    | DENSITYCHECKSTEP   | -windowStep | <i>Window Size X</i><br><i>Window Size Y</i> |

The EDI System 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:

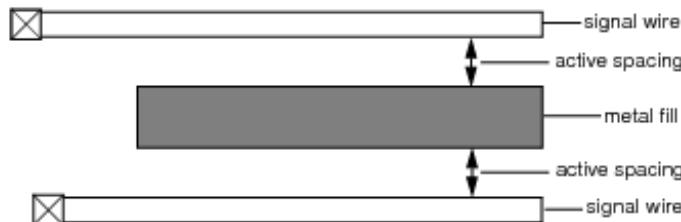
```
setMetalFill -layer "7 8"  
    -preferredDensity 35 -gapSpacing 0.4 \  
    -minWidth 0.2 -maxWidth 2.0  
    -minLength .2 -maxLength 20.0
```

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 EDI System software uses the `FILLACTIVEESPACING` value in the LEF file for the active spacing. If `FILLACTIVEESPACING` is not specified, you can set it manually by using one of the following methods:

- Specifying a value for `setMetalFill - activeSpacing` on the text command line  
**Note:** The `setMetalFill -activeSpacing` settings are used for creating regular `FILLWIRE` shapes. For `FILLWIREOPC` shapes, design rules are used.
- Specifying a value for *Active Spacing* on the Setup Metal Fill form

If no value is specified in the LEF file, and you do not specify one manually, the software uses 0.6 microns for thin layers (less than 0.24 microns) and 0.8 microns for thick layers as the default active

spacing value.

The default active spacing value is usually large enough that you can avoid using Optimal Proximity Correction (OPC) for the metal fill shapes. In addition, the default active spacing minimizes the increase in cross-coupling capacitance caused by the metal fill, which in turn reduces the additional timing delay.

As you increase the active spacing value, you reduce the space available for metal fill. A large increase--for example, 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.

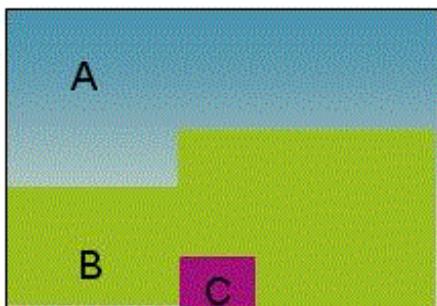
A routing blockage can have the following attributes:

- 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 EDI System 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 ...
```

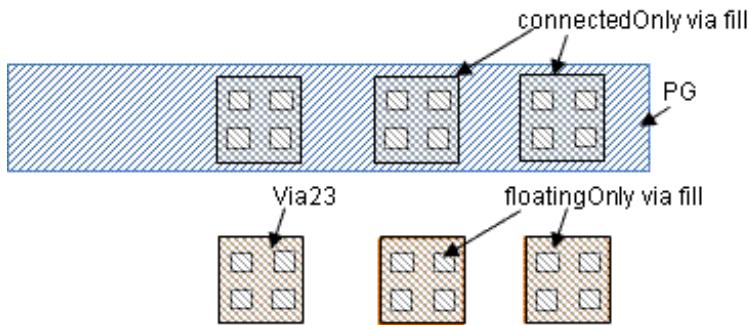
```
setMetalFill -layer "Metal2 Metal3" -activeSpacing 0.4 -gapSpacing 0.1 -maxWidth  
0.1 -maxLength 10 -windowSize 50 50 -windowStep 25 25 -minDensity 15 -maxDensity  
85 ...
```

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"
```

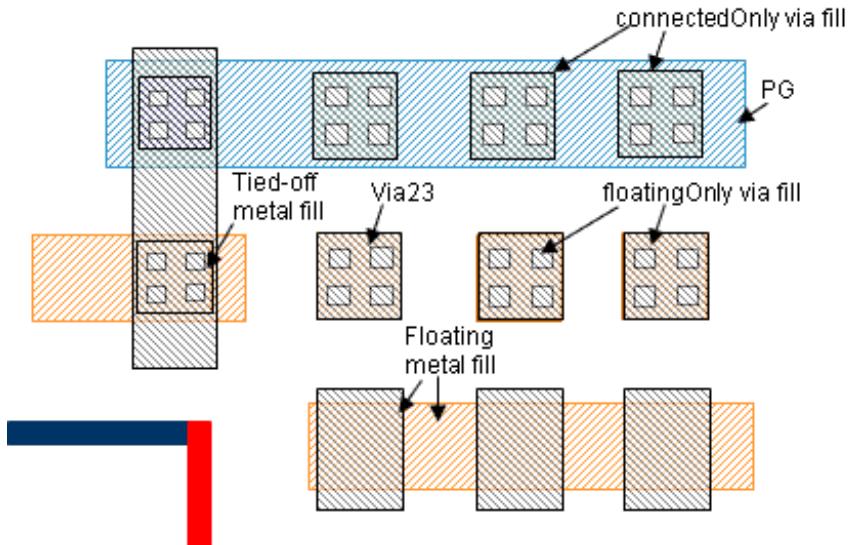
This will add via fill in white space to meet the via density requirements according to the specified rules. Via fill can be connected to power or ground nets (tied off) or unconnected (floating). Via fill cannot be connected to signal nets.



### 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.



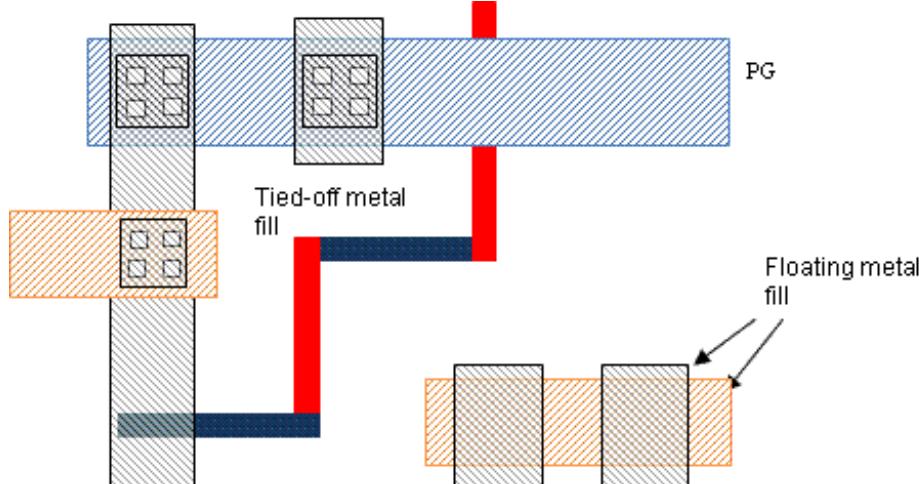
EDI now provides an alternative flow in which the software adds metal fill to free space prior to adding via fill shapes. It can connect metal fill with special via fill. In this alternative flow, you:

#### 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.

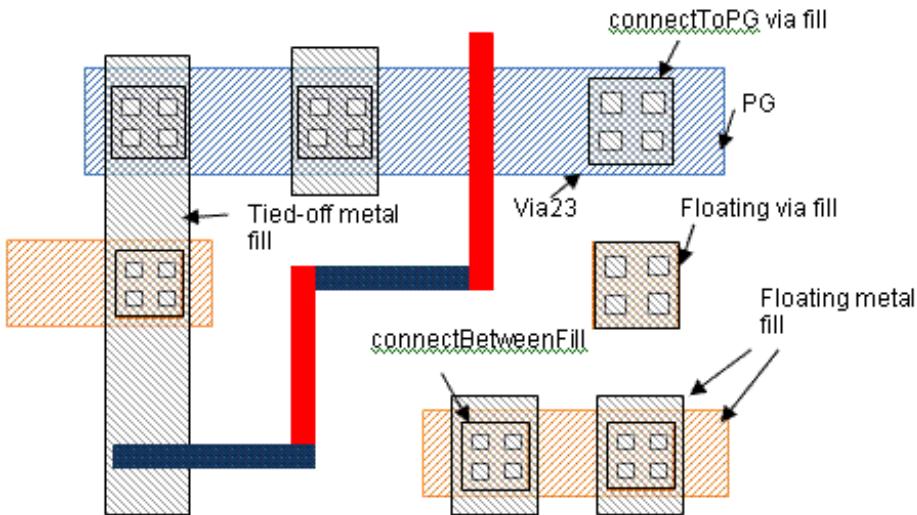
```
setViaFill -layer "Via23" -windowSize 50 50 -windowStep 25 25 -minDensity 0.005  
-maxDensity 30 ...
```

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 EDI System. 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:** EDI 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

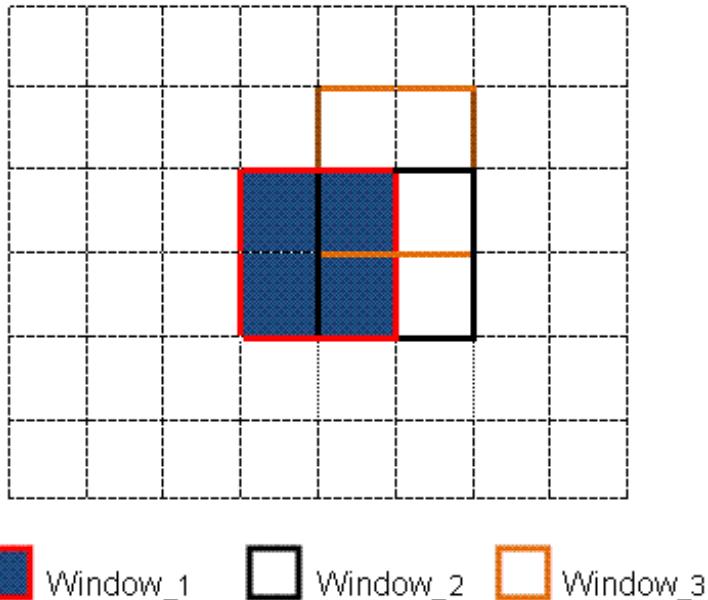
To prevent density in neighboring regions from varying too much, the addMetalFill targets a preferred density. This minimizes the variation in density from window to window. You can set the parameters as follows:

```
- minDensity 15 -maxDensity 85 -preferredDensity 35
```

```
addMetalFill -layer {Metal1 Metal2 Metal3}
```

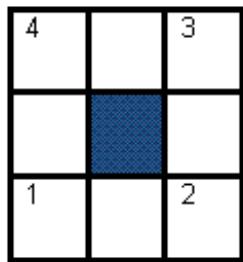
The metal fills are inserted into white space to meet the preferred density. When the metal density in a window is less than the minimum metal fill density value, addMetalFill adds metal fill to achieve a density slightly above the preferred density, if possible. If the density is larger than maximum 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.

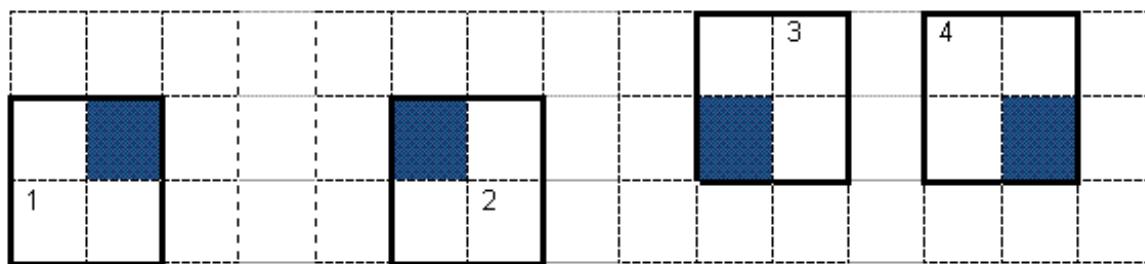


When add metal commands are applied, the engine calculates the Window\_1 density and tries to insert metal fill until the window density reaches the preferred density target. When Window\_1 is finished, the engine moves to the next window. The window step is specified in `setMetalFill` command. Note 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\*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
```

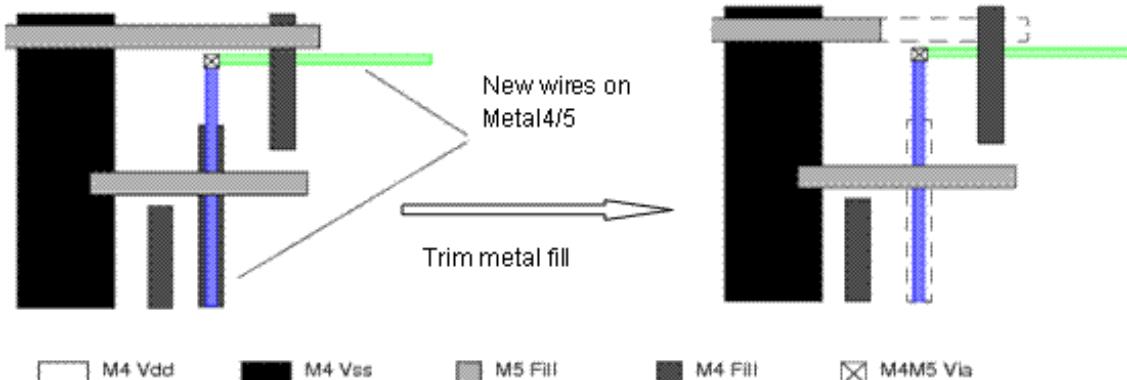
When required, you can use the [getMetalFillSpacingTable](#) command to print specified or all spacing tables used by `trimMetalFill`.

## 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.

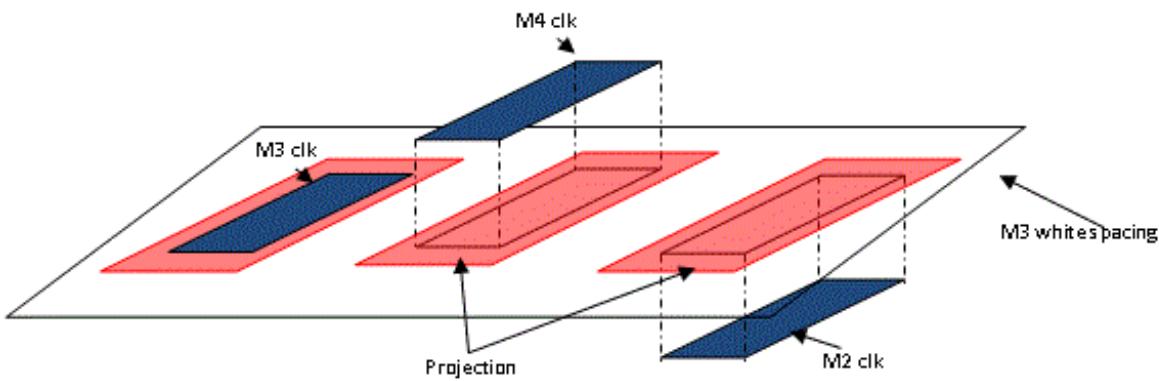
- To limit the area that in which metal fill is trimmed, use the -area parameter of the `trimMetalFill` command. This option is recommended for use with only floating metal fill. If you use `trimMetalFill -area` on tied-off fill shapes, some of the shapes may become isolated from the Power/Ground network.
- 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 EDI System 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 EDI System 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. EDI System calculates the metal density while trimming metal fill. If the metal density is less 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
```

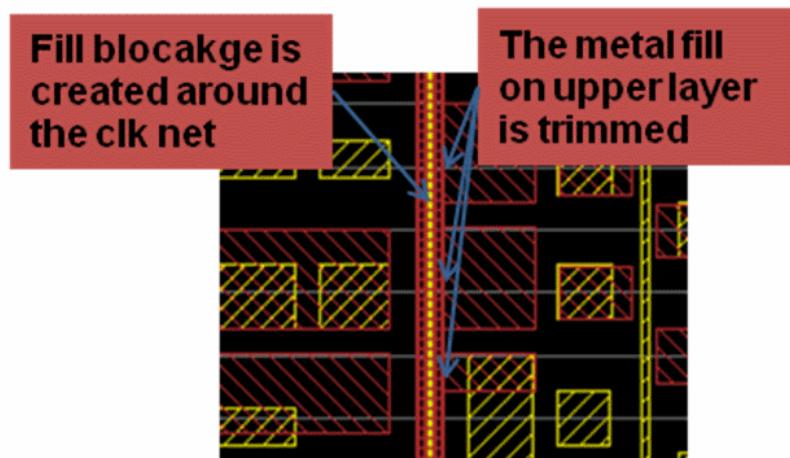
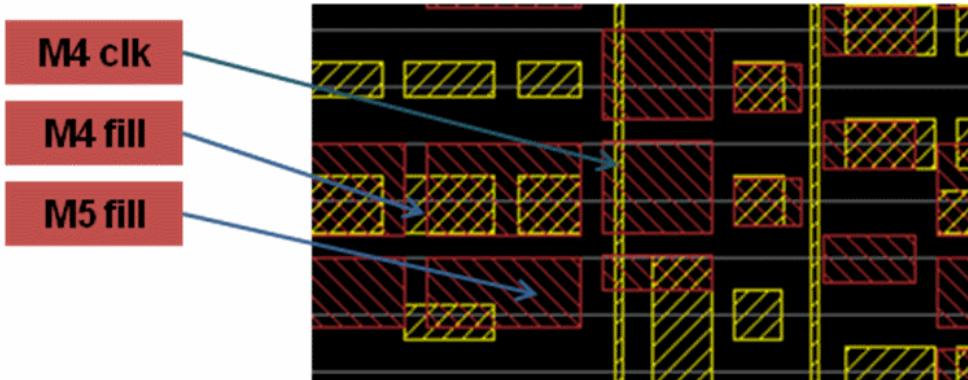
2. Trim metal fill with larger spacing near more critical nets.

```
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 *EDI System Text Command Reference*.

## Adding Metal Fill Using the GUI

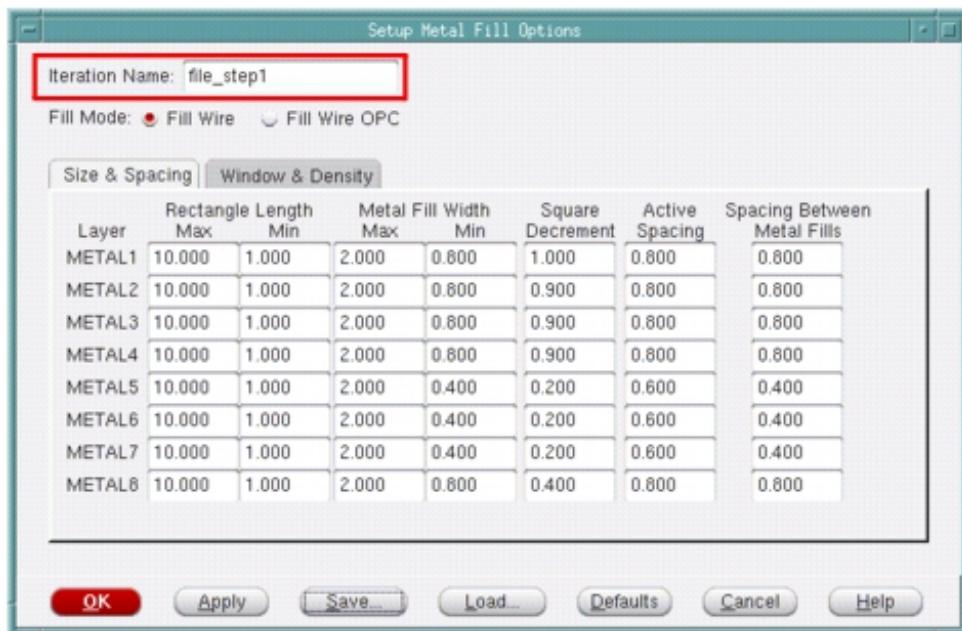
1. Determine the minimum and maximum size for metal fill shapes for each layer, then set these values on the *Size & Spacing* page of the Setup Metal Fill form.
  - If you are using rectangular metal fill, use the *Rectangle Length* and *Metal Fill Width* values.
  - If you are using square metal fill, 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 Metal1 -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
```

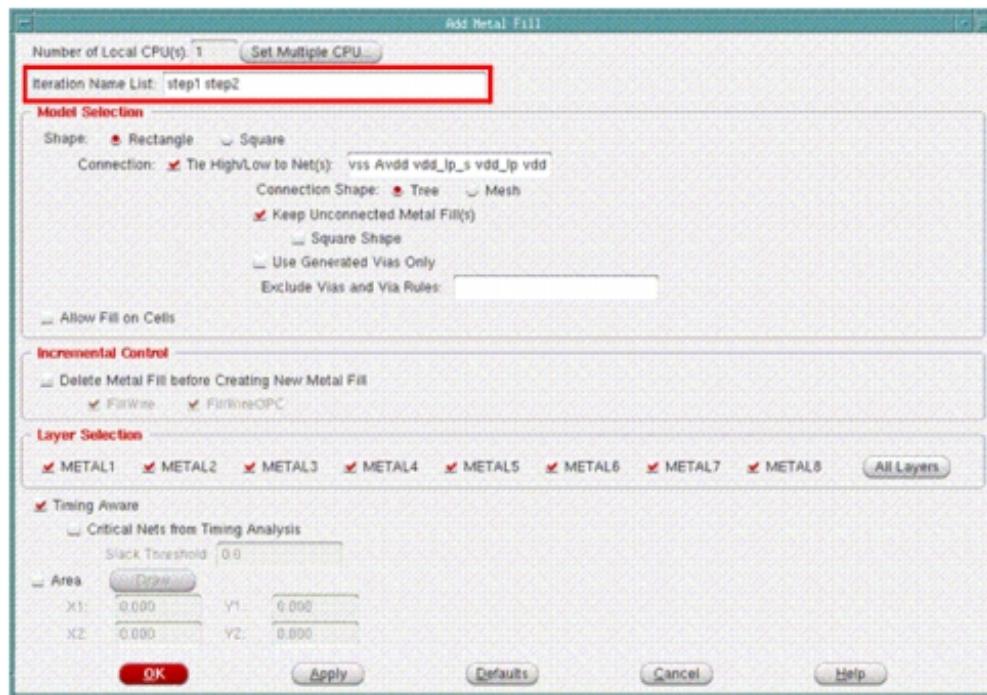
```
setMetalFill -iterationName step2 -layer -windowSize 100 100 -windowStep 50 50
```

```
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.

---

# 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](#)
- [Budgeting Output Files for MMMC Designs](#)
  - [Corner Cloning](#)
  - [Mode Cloning](#)
- [Setup and View Handling for MMMC Designs](#)
- [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](#)
  - [Power Pin Support in Budgeted Timing Models for Low Power Designs](#)
- [Calculating Timing Budgets](#)
- [Customizing Budget Generation](#)
- [Verifying Timing Budgets](#)
- [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](#)
- [Support for Distributed Processing in Budgeting](#)
- [Constraints Support in Budgeting](#)

- [Warning Report](#)
  - [Pin Constraint Values Greater than Available Time](#)
  - [Warning Report Example](#)

## Overview

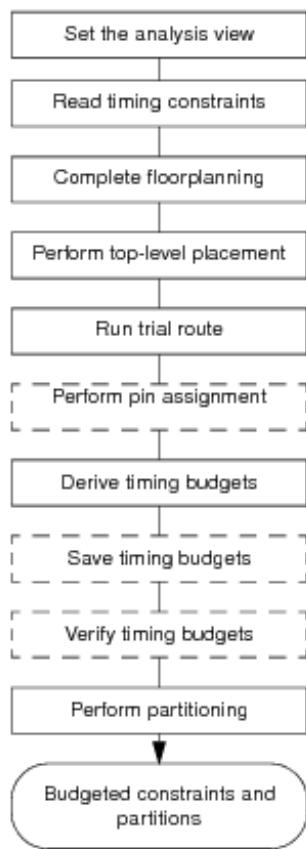
In hierarchical design flows, chip-level timing constraints must be mapped correctly to corresponding block-level constraints. The Encounter® Digital Implementation System (EDI System) software does this automatically to produce predictable timing convergence.

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. EDI System 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 MMMC mode, timing budgeting is run per-view, and generates view files for partition implementation and top-level implementation (not chip assembly).

The following flowchart shows how timing budgeting is performed within the overall design flow.



**Note:** You can set the analysis view at any time before performing timing analysis, but could also be done before reading the constraint file.

## Is My Design Ready for Budgeting?

In order to close timing on the hierarchical level, you must be able to close timing on the flat design first. If the fully flat placement of a design (based on the partition fences, pin placement, and so on) does not meet timing before partitions are committed, then it is unlikely that timing will close after the partitions are committed and the budgets generated.

To help you decide whether the design is ready for budgeting, run virtual IPO on the flat design. This builds a timing graph, which allows you to examine the design for failing inter-partition paths. When you find these paths, you can use the information to determine why the problems occur and how you can fix them. In a hierarchical implementation, you might need to iterate top-level floorplanning and virtual IPO several times before creating a floorplan that can meet timing.

When you are convinced that the timing will close at this stage of the design process, you can then run timing budgeting.

## Deriving Timing Budgets

You can generate timing budget constraint files for each top-level partition using either the partition graphical user interface (GUI) or the various text commands.

### Budgeting Using the GUI

To generate the constraints files using the GUI, complete the following steps:

1. Read in the timing constraint file during design import.
2. Complete the floorplan for the design to partition.  
The more complete the floorplanning, the better the timing budgeting results.
3. Run top-level placement and preferably trial route, choosing medium effort for both.
4. (Optional) Use the Assign Partition Pins form (*Partition - Assign Pins*) to assign the partition pins.
5. Derive timing budgets. Choose *Derive Timing Budget* from the *Partition* menu, and specify the options you need on the Derive Timing Budget form.
6. (Optional) Save timing budgets. Select *Design - Save - Save Timing Budget*.
7. Partition the design. Select *Partition - Partition*.

Use the *Save Partition* form (*Design - Save - Partition*) to save the partitions to their directories. The directories are called *topCellName* for the top-level and *partitionName* for the partitions.

For each partition, this procedure creates a timing constraint file named *partitionName.constr.pt* for PrimeTime format. These files are located in each partition directory. The library model files, *partitionName.lib*, are created in the top-level directory. The constraints file *top\_level.top.constr* is created for the top level.

The result is budgeted constraints and partitions.

### Budgeting Using Text Commands

To generate the constraints files using the text commands, complete the following steps:

1. Read the timing constraint file during design import.
2. Complete the floorplan for the design to partition.

The more complete the floorplanning, the better the timing budgeting results.

3. Run top-level placement and preferably trial route.
4. Perform pin assignment.
5. Derive the timing budgets using the [deriveTimingBudget](#) command with the -ptn parameter.
6. (Optional) Save the derived budgets using the [saveTimingBudget](#) command with the -ptn parameter.
7. Partition the design.

The result is budgeted constraints and partitions.

## Top-Level Budgets Derived by Using Active Logic View

The software provides a top-level interface timing analysis flow to perform partitioning and budgeting on a trimmed-down version of the timing graph: a virtual partition. This flow saves memory usage and provides faster run time on large designs. The results of the flow are comparable to results of partitioning and budgeting with the full timing graph.

To perform partitioning and budgeting using top-level timing analysis, complete the following steps:

1. Load the hierarchical design into the database. Specify the partition information in the database.

```
source init.enc
```

2. Run placement.

```
setPlaceMode -fp true -ignoreScan true  
placeDesign
```

3. Run Trial Route and perform pin assignment.

```
trialRoute  
assignPtnPin
```

4. Run Trial Route to honor assigned pins.

```
trialRoute -honorPin
```

5. Use the `createActiveLogicView` command to identify interface logic for all the partitions.

```
createActiveLogicView -type flatTop
```

This command marks the interface logic in all the partitions in the chip-level design. When you build the timing graph after this step, the software masks the core logic inside the partitions and ignores it to reduce the run time and memory usage.

6. Derive timing budgets.

```
deriveTimingBudget
```

The command rebuilds the timing graph as part of its operation.

7. (Optional) Evaluate the budgeting results.

```
justifyBudget -pin pinABC Partition1
```

8. Save the timing budgets.

```
saveTimingBudget
```

9. Clear the virtual partition marking after you save the timing budgets.

```
clearActiveLogicView
```

## Deriving Preliminary Budgets in Early Design Phase

The software provides a flow for deriving timing budgets in the early stages of the design to obtain a preliminary estimate of the budgets. The software uses the net delay and net load that you specify using the [setBudgetingMode](#) command. To perform the preliminary budgeting, you use the `-preliminary` parameter of the [deriveTimingBudget](#) command. The software does not use trialIPO operations for calculating budgets for this flow.

The software uses the top-level net delays that you specify using the `-topLevelDelayPerLen` and `-topLevelMinDelayPerNet` parameters of the [setBudgetingMode](#) command. The software calculates the total delay value using the value of the `-topLevelDelayPerLen` parameter and the total length of the net. If the total delay value that the software calculates is less than the `-topLevelMinDelayPerNet` parameter, the software uses the value of the `-topLevelMinDelayPerNet` parameter. Otherwise the software uses the value of the `-topLevelDelayPerLen` parameter. The software assumes the block-level delays are zero.

The software uses the lump capacitance that you specify using the [setBudgetingMode](#) `-overrideNetCap` command for both top-level and block-level nets. The software uses this value to calculate the cell delay.

The software assumes all the net delay is on the top-level and does not perform boundary net adjustments. Therefore, when you run the `justifyBudget` command after generating the preliminary budgets, the `Adjustment by Net Delay` value is zero in the budgeting report.

The software proportions the budget according to the calculated values. If you do not use the `-preliminary` parameter in the `deriveTimingBudget` command, the software adds three extra buffer delays to the delay of the positively slacked path that has positive slack. If you use the `-preliminary` parameter, the software distributes the positive slack equally between source block, destination block, and top-level. Therefore, the value of following fields in the budgeting report is zero:

- Virtual buffering adjustment

- External buffering adjustment

For negative slack, the software uses the `-freezeTopLevelNegPathOnly` parameter of the `setBudgetingMode` command.

To perform preliminary budgeting, complete the following steps:

1. Load the hierarchical design in the database. Specify the partition information in the database.

```
source init.ENC
```

2. Run placement.

```
setPlaceMode -fp -ignoreScan  
trialRoute -noDetour -floorplanMode  
placeDesign
```

**Note:** Steps 1 and 2 are optional. You can source a routed design instead.

3. Set the delay values to be used during budgeting.

```
setBudgetingMode -topLevelDelayPerLen value  
-topLevelMinDelayPerNet value  
-overrideNetCap value
```

4. Derive timing budgets.

```
deriveTimingBudget -ptn partitionName -preliminary
```

5. Verify the budgets.

```
justifyBudget -short -pins pinList partitionname
```

6. Save the budgets `saveTimingBudget -dir dirName -pt -rptNegSlackOnPorts value`

## Budgeting Output Files for MMMC Designs

In MMMC mode, timing budgets are derived automatically for per view. Use the following command:

```
deriveTimingBudget -justify
```

This command derives the budgets for all ports of the instances or partitions specified by `-inst` or `-ptn`. Use `deriveTimingBudget -justify` instead of `justifyBudget` to generate a report for per view/per partition.

Files are generated per view. The files are generated with the following structure:

```
budget_justify/Partition1/Partition1_view1.justify  
budget_justify/Partition1/Partition1_view2.justify  
budget_justify/Partition1/Partition1_view3.justify  
budget_justify/Partition2/Partition2_view1.justify
```

`budget_justify/Partition2/Partition2_view2.justify`

In cases of setup and hold timing budgeting:

`budget_justify/Partition1/Partition1_view1_setup.justify`  
`budget_justify/Partition1/Partition1_view1_hold.justify`  
`budget_justify/Partition1/Partition1_view2_setup.justify`  
`budget_justify/Partition1/Partition1_view2_hold.justify`

A view is a combination of a mode (.sdc files) and a corner (libraries). The delay of the timing arcs through the partition ports may be different in terms of the constraints files (modes) and the libraries (corners). This means that one mode at the chip level becomes two or more modes at the partition/ top level. In this case, modes must be cloned. Similarly, one corner becomes two different corners at the top level because the timing model of a partition is not the same for the two different views. In this case, corners are cloned.

- Corner cloning affects only top-level data
- Mode cloning affects partition and top-level data

## Corner Cloning

The following example shows budgeting output for views having the same corner (c1) and different modes (M1 and M2).

`View 1 (M1 C1)`  
`View 2 (M2 C1)`

The output data from `deriveTimingBudget` are for the top level are as follows:

`Top_view1.constr`  
`Top_view2.constr`  
`Ptn_view1.lib`  
`Ptn_view2.lib`

The output data are for the block level are as follows:

`Ptn_view1.constr.pt`  
`Ptn_view2.constr.pt`

At the top level, for the corners, the view definition file contains the following:

`View 1: Corner1 (pointing to corner 1 + Ptn_view1.lib)`  
`View 2: Corner1_budgeting1 (pointing to corner 1 + Ptn_view2.lib)`

Here, the corner is cloned.

## Mode Cloning

The following example shows budgeting output for two views with the same mode (M1), but different corners (c1 and c2):

```
View 1 (M1 C1)  
View 2 (M1 C2)
```

The output data from `deriveTimingBudget` are for the top level are as follows:

```
Top_view1.constr  
Top_view2.constr  
Ptn_view1.lib  
Ptn_view2.lib
```

The output data are for the block level are as follows:

```
Ptn_view1.constr.pt  
Prn_view2.constr.pt
```

At the top level, for modes, the view definition file contains the following:

```
View 1: Mode 1 (pointing to the Top_view1.constr)  
View 2: Mode1_budgeting1 (pointing to the Top_view2.constr)
```

In this case, the modes are cloned.

## Setup and View Handling for MMMC Designs

Views can be created for both setup as well as hold analysis. For example, the following command can set the same view for both setup and hold analysis:

```
set_analysis_view -setup [view] -hold [view]
```

The budgeting output constraint file contains setup as well as hold time constraints, but the `.lib` files are written separately for setup and hold.

The view definition file libraries for setup and hold are specified as follows:

```
create_library_set -name name _max -timing [... Ptn_max.lib]  
create_library_set -name name _min -timing [... Ptn_min.lib]
```

```
create_delay_corner -name corner-name
-late_library_set name _max
-early_library-set name _min
```

## 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`.

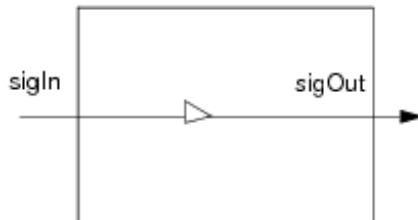
```
max(SIDm, SIDc1, SIDc2, ...),  
max(SODm, SODc1, SODc2, ...)
```

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:

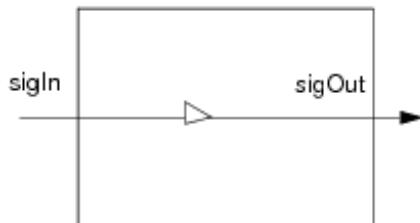
```
set_input_delay 3 sigIn  
set_output_delay 1 sigOut
```

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:

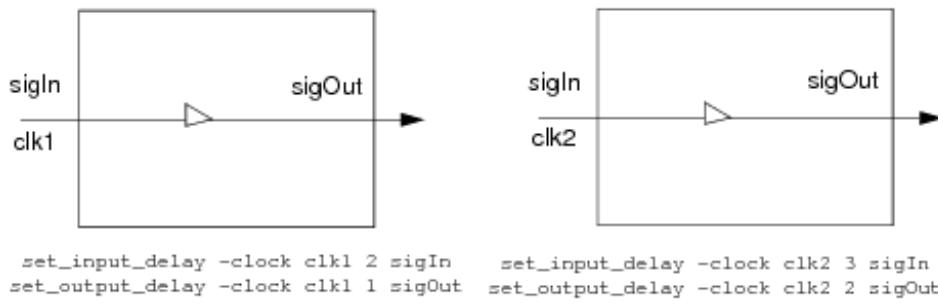
```
set_input_delay 3 sigIn  
set_output_delay 2 sigOut
```

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.

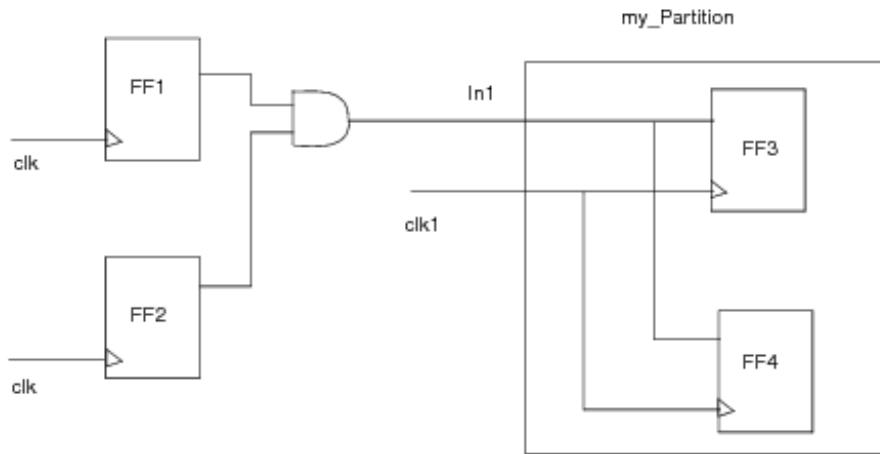
A pin might have multiple paths going through it. Multiple paths through the same port are handled by CTE budgeting. In case of multiple paths related to the same clocks and the same number of clock cycles, the tool automatically chooses the best path for deriving the budgets.

## 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

Two virtual clocks will be created for each same group of path-based exceptions during budgeting.

- Chip-level exceptions:  
`set_multicycle_path 2 -setup -from FF1 -to my_partition/FF3/D`
- For single cycles path and clk:  
`set_multicycle_path 1 -clock -from FF1 -to my_partition/FF3/D`

```
set_input_delay -clock clk_v0 number In1
```

(based on worst path from FF1)

- For multicycle path and clk:

```
set_input_delay -clock clk_v1 number In1
```

(based on worst path from FF1)

```
set_multicycle_path 2 -from clk_v1 -through in1 -setup -to FF3/D
```

## Case 2

Two virtual clocks are created for each same group of path-based exceptions.

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF4/D
```

```
set_false_path -from FF1 -to my_partition/FF3/D
```

```
set_multicycle_path 2 -setup -from FF2
```

- For multicycle paths from FF1:

```
set_input_delay -clock clk1_v0 number In1
```

(based on path from FF1 to my\_partition)

```
set_multicycle_path 2 -setup -from clk1_v0 -through In1 -to FF4/D
```

- For false path from FF1:

```
set_false_path -from clk1_v0 -through in1 -to FF3/D
```

- For multicycle path from FF2:

```
set_input_delay -clock clk2_v0 number In1
```

(based on worst path from FF2)

```
set_multicycle_path 2 -from clk2_v0 -through in1 -setup
```

### Case 3

Two virtual clocks are created.

- Chip-level exceptions:

```
set_multicycle_path 3 -setup -from FF1 -to my_partition/FF3/D
```

```
set_multicycle_path 2 -setup -from FF2 -to my_partition/FF4/D
```

- For multicycle 3 path and clk:

```
set_input_delay -clock clk_v0
```

(based on worst of paths from FF1)

```
set_multicycle_path 3 -from clk_v0 -through in1 -setup -to FF4/D
```

- For multicycle 2 path and clk:

```
set_input_delay -clock clk_v1 number In1
```

(Based on worst path form F2)

```
set_multicycle_path 2 -from clk_v1 -through in1 -setup -to FF4/D
```

### Case 4

Two virtual clocks are created:

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF4/D
```

```
set_false_path -from FF1 -to my_partition/FF3/D
```

```
set_multicycle_path 2 -setup -from FF2
```

- For multicycle 2 path from FF1:

```
set_input_delay -clock clk1_v0 number In1
```

(based on path from FF1 to my\_partition/FF4)

```
Set_multicycle_path 2 -setup -from clk1_v0 -through In1 -to FF4/D
```

- For false path from F1:

```
set_false_path -from clk1_v0 -through in1 -to FF3/D
```

- For multicycle 2 path from FF2:

```
set_input_delay -clock clk2_v0 number In1
```

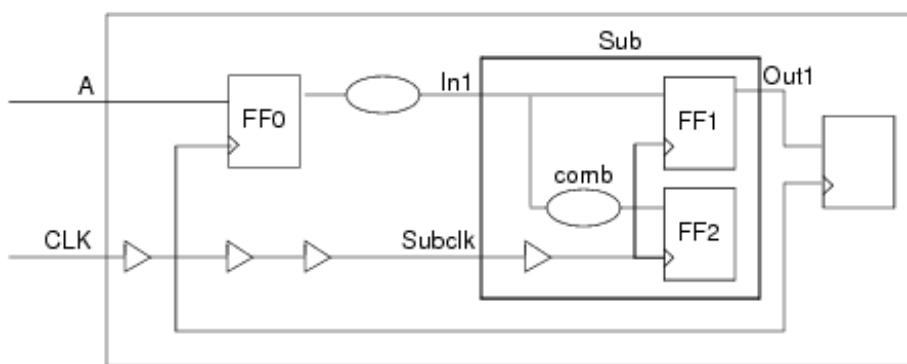
(based on worst of paths from FF2)

```
set_multicycle_path 2 -from clk2_v0 -through in1 -setup
```

## Budgeting Clock Latency in Propagated Mode

EDI System includes clock latency in the constraints generated for clocks in propagated mode. The clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The clock latency is added to `set_input_delay` and subtracted from the `set_output_delay`. This feature is useful when a clock tree is present in your design.

For multiple paths, both source and propagated clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The software adds the `-source_latency_included` and `-network_latency_included` constraints in the `set_input_delay` and `set_output_delay` constraints for all inputs and outputs related to clocks in propagated mode. Consider the following figure.



The `deriveTimingBudget` command result in the following constraints for the sub partition:

```
create_clock -clock subclk -waveform...
set_clock_latency -source (top_source + 0.2 + 0.2 + 0.2) subclk
set_input_delay -clock subclk (Input delay + top_source + 0.2)
    -source_latency_included -network_latency_included In1
```

```
set_output_delay -clock subclk (output_delay -top_source -0.2)
    -source_latency_included -network_latency_included Out1
```

Where,

*top\_source* = source latency of the clock at the top level

0.2 = delay through each buffer in the clock network

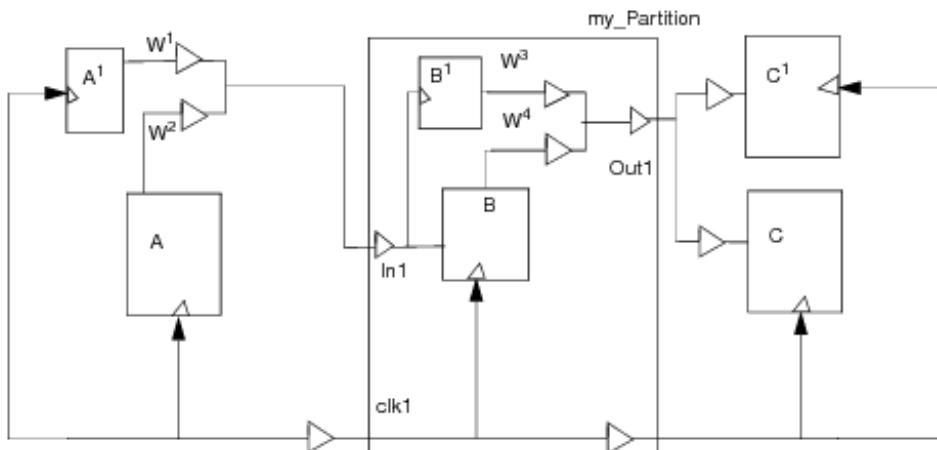
## Budgeting Libraries

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

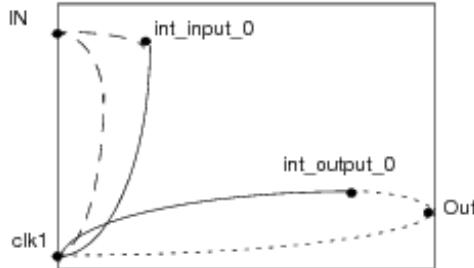


- Chip-level exceptions:

```
set_multicycle_path 2 -from A^1 /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.

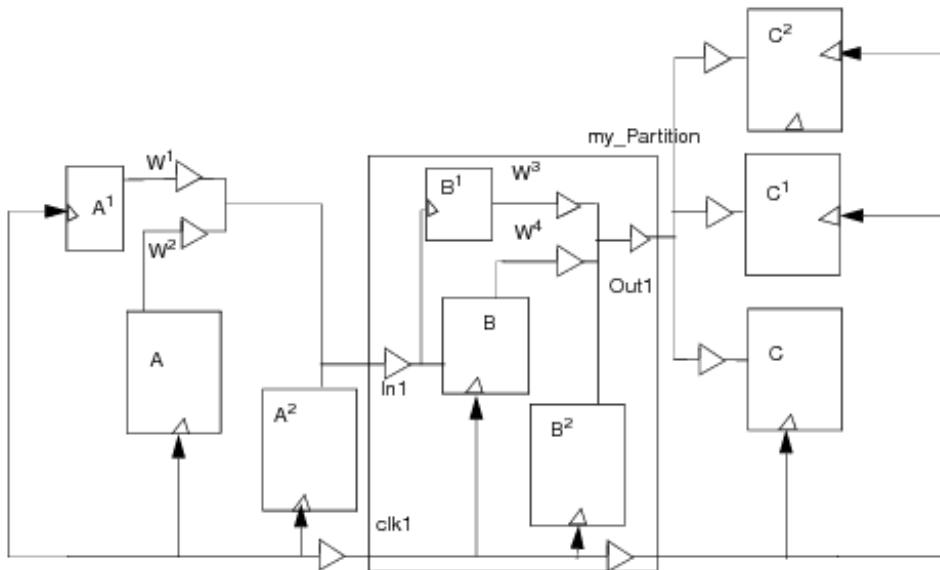


- Top-level exceptions:

```
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:

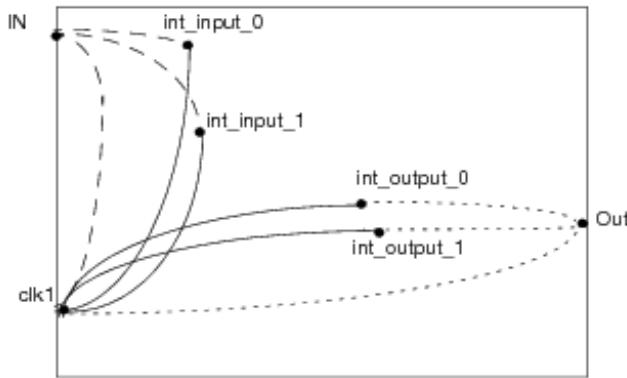
```
set_multicycle_path 2 -from [get_pins {A1/CK}] -to [get_pins {my_partition/B/D}]
```

```
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}]]
```

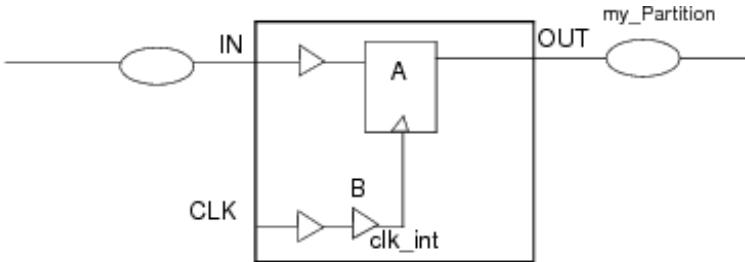
```
set_multicycle_path 2 -setup -through [list [get_pins {my_partition/int_output_1}]] -through [list [get_pins {my_partition/out}]] -to [list [get_pins {C1/D}]]
```

```
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.

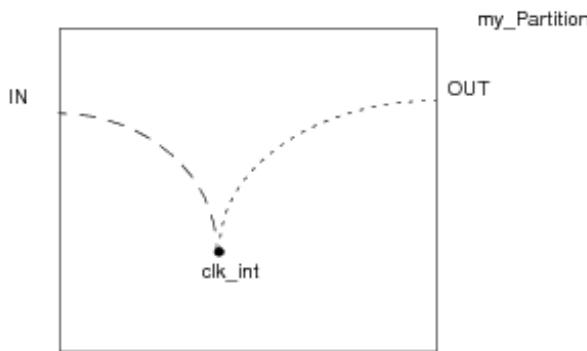


- Chip-level constraint:

```
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 `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/clk_int}]
```

## Power Pin Support in Budgeted Timing Models for Low Power Designs

In a variation of the CPF-based flow, the CPF file contains only power domain information and assigns timing libraries to them. Since voltages are not defined in the CPF file, the voltage information comes from these timing libraries through constructs like `voltage_map`, `pg_pin`, `related_power_pin` and `related_ground_pin`. It is important to have this voltage information in budgeted timing models to ensure that the top-level flow is able to use the budgeted .lib containing the power and ground pin information for an accurate CPF-based flow.

Timing Budgeting supports PG pins in budgeting timing models. Therefore, if a full-chip design has libraries defined with PG pins, timing libraries generated for each instance/block will have voltage information. The budgeting timing library imports the following constructs from the full-chip library:

- **Voltage Maps** - are written for all the voltages that are coming inside a partition/instance for which the timing model is being written.

```
voltage_map( vdd, 0.903 );
```

This information is at the library level. Every voltage has a name given and a value is associated with it.

- **Cell Level PG Pin Information** - For each voltage map, a related pg pin is defined for the cell. It will have a pin name and the voltage name it is to be connected to.

```
pg_pin ( vdd ) {  
    voltage_name : vdd;  
    pg_type : primary_power;  
    direction : input;  
    physical_connection : device_layer;  
}
```

This information is at the cell level. The pins described in this definition are global in the library scope. All the pins in the library are considered to be connected to one of these pins.

- **Pin Level PG Information** - At each port of the partition/instance, there will be a related pg\_pin construct that specifies from which power/ground pin is the port being driven.

```
pin (q) {  
    related_power_pin : vdd;  
    related_ground_pin : vss;  
    ....
```

## Calculating Timing Budgets

EDI System 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 provisioning 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 provisioning formula for a setup check (max budgeting) is:

$$SD + SD/TD * PS = BB \text{ (pos)}$$

For paths with positive slack, the provisioning formula for a hold check (min budgeting) is:

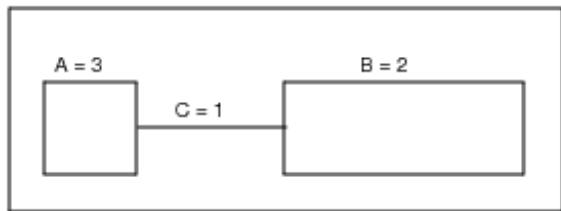
$$SD - SD/TD * PS = BB \text{ (pos)}$$

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.
- $BB$  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:

$$\text{Budget for block A} = 3/6 * 5 = 2.5 \text{ ns}$$

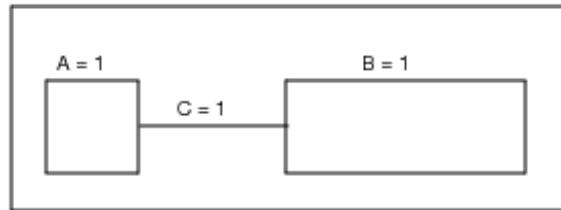
$$\text{Budget for block B} = 2/6 * 5 = 1.67$$

$$\text{Budget for net C} = 0.83$$

$$\text{Output delay at A} = \text{Budget for block B} + \text{Budget for net C}$$

$$\text{Input delay at B} = \text{Budgets for blocks A} + \text{Budget for net C}$$

### Example 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:

$$\text{Budget for A, B, and C} = 1 + 1/3 * 2 = 1.66 \text{ ns}$$

## Customizing Budget Generation

You can customize budget generation according to the design stage and timing requirements. To customize budget generation, use the following commands in EDI System:

- 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.
- The `setBudgetingMode -topLevel` command specifies the minimum percentage of available allowed time to be set aside for the top. For example, if the clock period is 10ns and this minimum value is set to 0.1, then 10 percent of the clock period is set aside for top-level. The remaining 9 ns will be proportioned between the partitions.

## Verifying Timing Budgets

EDI System provides feedback on how the budgets are generated. The feedback is provided in a budget report file. You can verify the timing budgets by analyzing the results in the report file. To verify timing budget values, generate the report file by completing the following steps:

1. Derive budgets for partitions. In the command tool (console) window, type the following:

```
deriveTimingBudget -justify -ptn partitionName
```

The `deriveTimingBudget` command generates the timing models and stores them in the partition directories. The command creates one justify report per view. The report contains debug data to justify timing budget for each pin and partition.

Views can be created for both setup and hold analysis, so the command generates budgets for setup and/or hold analysis type specified with `set_analysis_view`.

2. (Optional) Obtain a report consisting of headers for the reports generated per view:

```
justifyBudget -short
```

When in MMMC mode, `-short` is the only valid parameter for `justifyBudget`. If you want full reports, use `deriveTimingBudget -justify`.

3. Save the generated budgets by typing the following:

```
saveTimingBudget -ptn partitionName
```

The `saveTimingBudget` command saves time budget files of specified hierarchical instances to a specified directory. If you specify the `-setupHold` parameter in the `setBudgetingMode` command, the `saveTimingBudget` command saves both setup and hold budgets.

## 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 - 0.009 = 1.372

Path 1: MET Setup Check with Pin i\_sub2/sub\_FF/CP  
Endpoint: i\_sub2/sub\_FF/D (^) checked with rising edge of 'clk2'  
Beginpoint: i\_sub1/sub\_FF/Q (^) triggered by rising edge of 'clk1'  
Other End Arrival Time 0.000  
- Setup 0.269  
+ Phase Shift 1.000  
+ Cycle Adjustment 1.000  
= Required Time 1.731  
- Arrival Time 0.641  
= Slack Time 1.090  
Clock Rise Edge 0.000  
= Beginpoint Arrival Time 0.000

| Instance | Arc | Cell | Delay | Arrival | Required | Slew | Load | Instance |
|----------|-----|------|-------|---------|----------|------|------|----------|
|----------|-----|------|-------|---------|----------|------|------|----------|

|               |                |        | Time  | Time  |       |       |       | Location           |
|---------------|----------------|--------|-------|-------|-------|-------|-------|--------------------|
| <hr/>         |                |        |       |       |       |       |       |                    |
|               | clk1 ^         |        |       | 0.000 | 1.090 | 0.000 | 0.007 |                    |
| i_sub1/sub_FF | CP ^ -><br>Q ^ | df1qx1 | 0.182 | 0.182 | 1.272 | 0.063 | 0.005 | (43.12,<br>366.80) |
| buf1          | A ^ -> Z<br>^  | bfx0   | 0.455 | 0.637 | 1.727 | 1.003 | 0.040 | (43.96,<br>398.16) |
| i_sub2/sub_FF | D ^            | df1qx1 | 0.004 | 0.641 | 1.731 | 1.003 | 0.040 | (43.12,<br>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 Slack (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 Slack (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)

## Support for Distributed Processing in Budgeting

Encounter 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.tbMMMCDDistributed 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 {losdclko_rp}]] ]
```

Where, `losdclko_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

EDI System timing analysis requires that the `set_false_path` and `set_multicycle_path` constraints have valid startpoints and endpoints for the `-from` and `-to` options. This corresponds to the requirement of PrimeTime.

Valid startpoints are:

- - Input ports
  - Input part of bidirectional ports
  - Clock pins of sequential cells
  - Pins associated with `set_input_delay`
  - Pins associated with `set_path_delay -from`

Valid endpoints are:

- Output ports
- Output part of bidirectional ports
- Data pins of sequential cells
- Pins associated with `set_output_delay`
- Pins associated with `set_max_delay -to`

During budgeting, the 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 */
```

---

## RC Extraction

---

- [Overview](#)
- [Before You Begin](#)
  - [Results](#)
  - [Specifying Temporary File Locations](#)
- [Extraction Flow in EDI System](#)
- [PreRoute Extraction](#)
- [PostRoute Extraction](#)
  - [Native Detailed](#)
  - [TQRC and IQRC](#)
  - [Sign-Off Extraction Using QRC](#)
  - [Inputs for 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 QRC 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](#)
  - [Comparing SPEF Files Using a Perl Script](#)
  - [Defining the Scale Factor](#)
- [Distributed Processing](#)
  - [Setting-up Distributed Processing](#)
  - [Generating a Capacitance Table in Multi-CPU Mode](#)
  - [Performing IQRC, TQRC, and Standalone QRC Extraction in Multi-CPU Mode](#)
  - [Setting Scale Factors in Capacitance Table File for PreRoute Extraction](#)

## Overview

You can perform two types of extraction in EDI System:

- 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
- Turbo QRC (TQRC)
- Integrated QRC (IQRC)
- Standalone QRC

For more information, see [PostRoute Extraction](#).

The following table summarizes the types of extraction used during the design process.

| <b>Extraction Type</b> | <b>When</b>     | <b>QRC License Required</b>                                                                   |   |
|------------------------|-----------------|-----------------------------------------------------------------------------------------------|---|
| PreRoute               |                 | Used during optimization both before and after clock tree synthesis.                          | X |
| PostRoute              | Native Detailed | Used during postRoute and SI optimization flow in older technologies.                         | X |
|                        | TQRC            | Used during postRoute and SI optimization flow in newer technologies.                         | X |
|                        | IQRC            | Used after ECO and for near signoff.<br><br><b>Note:</b> For IQRC, QRC-XL license is required | ✓ |
|                        | Standalone 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 EDI System 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 TQRC. 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 QRC Techfiles. Scaling for engine with `effortLevel` set to `signoff` is not supported.

The following files are required for the EDI System 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 QRC Techfiles are not specified. For more information, see the section titled, "Generating a Capacitance Table".

- QRC Techfile(s) - Used by preRoute extraction engine for 32nm and below technology, TQRC/IQRC, and Standalone QRC engines.

**Note:** One captable and a matching QRC Techfile 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 TQRC 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/TQRC 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 EDI System

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 `setDesignMode -process processnode` 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 QRC 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), TQRC, IQRC, or Standalone QRC extraction, read in the QRC Techfile that contains the interconnect models used by these engine choices. For more information, see [Reading a QRC 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.

1. Use the `setExtractRCMode` command to set up extraction parameters and to specify the extraction engine to be used for subsequent extraction.
2. 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.
3. 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.
4. Use the `spefIn` command to load the existing SPEF files with the parasitic data.
5. 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.

## TQRC and IQRC

TQRC 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, QRC. The TQRC extraction engine is recommended for the implementation phase. This is because it is optimized for performance with a small tradeoff for accuracy. The IQRC extraction engine is recommended for the ECO flow. It has near signoff accuracy.

In TQRC and 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.

**Note:** The performance and accuracy of TQRC falls between that of native detailed extraction and IQRC. IQRC provides superior accuracy compared to TQRC. Both IQRC and TQRC support distributed processing. TQRC does not require a QRC license while IQRC requires a QRC-XL license.

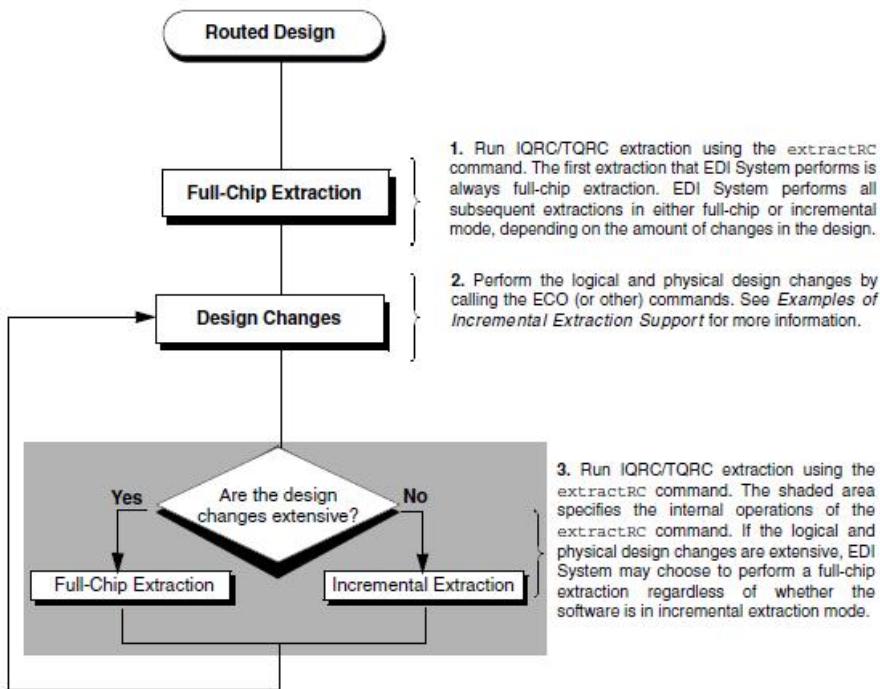
To invoke IQRC, use the following command:

```
setExtractRCMode -engine postRoute -effortLevel high
```

To invoke TQRC, use the following command:

```
setExtractRCMode -engine postRoute -effortLevel medium
```

The following figure shows the IQRC/TQRC RC extraction flow.



#### Examples of Incremental Extraction Support

Following are some of the examples for incremental extraction flow in the EDI System:

- **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 -si
```

**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 QRC

QRC Standalone extraction is accessible through the EDI System for generating detailed signoff quality parasitics. You can perform signoff extraction after the detailed routing phase.

**Note:** QRC Standalone extraction requires a separate license and requires installation of the EXT software package.

QRC can also be used with native extraction to generate an RC scale factor. For more information, see the section titled, "Correlating Native Extraction With Sign-Off Extraction".

You can run QRC extraction by using the [extractRC](#) command after setting the QRC extraction mode using the `-effortlevel signoff` option of the [setExtractRCMode](#) command.

**Note:** For TSV designs, you are required to provide the name of the layer map file for IQRC/TQRC and Standalone QRC.

**Note:** To generate scale factors, use the [generateRCFactor](#) command.

## Inputs for QRC Sign-Off Extraction

QRC Standalone extraction is a Design Exchange Format (DEF)-based flow. When you perform signoff extraction through the EDI System interface, a routed DEF is created automatically. The following inputs are required before you can start signoff extraction:

- **DEF file**--Contains the design-specific information of a circuit and is a representation of the design at any point during the layout process.
- **QRC technology file**--Contains the process-dependent model files and manufacturing effects used by the extractor to calculate resistance and capacitance.
- **LEF file**--Contains the relevant cell content that includes standard cells and macros. The file is loaded in the EDI System during design import.
- **QRC command file**--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.

## Scale Factor Setting

To better correlate native extraction results, both in preRoute and postRoute stages with sign-off extraction, the EDI System 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".

## Generating a Capacitance Table

A capacitance table is needed for accurate extraction by the preRoute and postRoute -effortLevel low extraction engines for above 32nm technology, and for 32nm and below technology only when QRC Techfiles 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. For more information about this feature, see [Setting Scale Factors in Capacitance Table File for PreRoute Extraction](#).
- **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, EDI System 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 TQRC 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 QRC Techfiles 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.

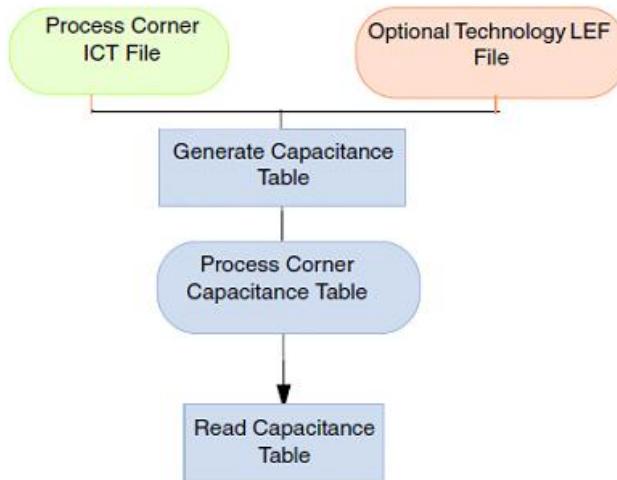
Fabrication process information in the ICT file can consist of the following:

- 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 Appendix A, [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 Appendix A, [Creating the ICT File](#).
2. Generate a capacitance table for each ICT file. Use the `generateCapTbl` command within the EDI System 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 EDI System hierarchy runs independent of the EDI System. 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 -0.01 -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 0.0216 0.0216 0.0215 0.0215  
0.0294 0.0286 0.0272 0.0257 0.0235 0.0224 0.0218 0.0216 0.0216 0.0216 0.0216  
0.0216 0.0216 0.0215 0.0215  
0.0286 0.0279 0.0269 0.0257 0.0235 0.0224 0.0218 0.0216 0.0215 0.0216 0.0216  
0.0216 0.0215 0.0215 0.0215  
...  
0.0264 0.0262 0.0259 0.0257 0.0235 0.0224 0.0218 0.0216 0.0215 0.0215 0.0215  
0.0215 0.0214 0.0213 0.0213  
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 [generateCapTb1](#) command to input a capacitance table and to specify the scale factors. Use the following parameters of the generateCapTb1 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 EDI 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 *EDI System 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 QRC Techfile

QRC Techfiles are required by preRoute (32nm and below), TQRC, IQRC, and signoff QRC extraction engines. Use the `-qx_tech_file` parameter of the [create\\_rc\\_corner](#) command to read in the QRC 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 EDI 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 *EDI System User Guide*.

## PreRoute Extraction Flow without Capacitance Table Data

QRC Techfiles are recommended for performing RC Extraction on designs at 32nm or below nodes. When QRC Techfiles are provided, capturable files are ignored. In this scenario, preRoute extraction uses QRC Techfiles (instead of capturable files) and postRoute extraction invokes "`-engine postRoute -effortLevel medium`" (i.e. TQRC).

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                                                                                                                                                                         |
| <b>PreRoute Extraction</b>  | Capturable not needed                                                   | Capturable Present         | Capturable Absent             | Capturable Present         | Capturable Present                                                                                                                                                                 |
|                             | QRCTechFile Present                                                     | QRCTechFile Absent         | QRCTechFile Present           | QRCTechFile Absent         | QRCTechFile Present                                                                                                                                                                |
|                             | Flow without Captable                                                   | Capturable-based Flow      | Flow without Captable         | Capturable-based Flow      | Capturable-based Flow                                                                                                                                                              |
| <b>PostRoute Extraction</b> | Detailed extraction is NOT allowed<br><br>TQRC extraction engine is run | Detailed extraction is run | TQRC extraction engine is run | Detailed extraction is run | TQRC extraction engine is run for 65nm and below – but detailed extraction is allowed by explicit setting.<br><br>For process nodes greater than 65nm, detailed extraction is run. |

## For designs at 32nm or below nodes

**Note:** The design mode is set in the EDI System using the [setDesignMode](#) -process command.

- Capturable files are not needed and QRC Techfiles are specified - preRoute extraction uses QRC Techfiles even when capturable files are provided. PostRoute extraction uses TQRC extraction engine and detailed extraction is not allowed. This is shown as Scenario 1 in above

table.

- Captable files are specified and the QRC 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 QRC Techfiles are specified - preRoute extraction uses QRC Techfiles. PostRoute extraction uses TQRC extraction engine. Detailed extraction is not allowed. This is shown as Scenario 3 in above table.
- Captable files are specified and the QRC 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.
- Captable files and QRC Techfiles are specified - preRoute extraction uses the capturable files. PostRoute extraction uses TQRC 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 captables nor QRC 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.

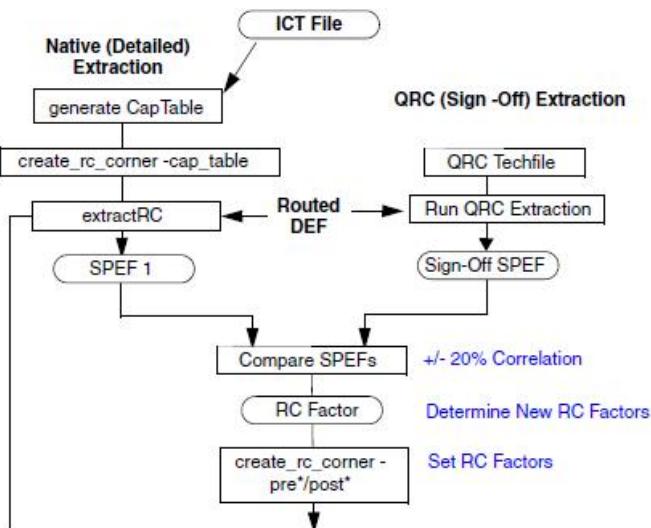
Complete the following steps to generate the RC scale factor to correlate native extraction results with sign-off extraction.

1. From the routed DEF, generate a SPEF file using the [extractRC](#) command. For more information on generating a SPEF file, see [Sign-Off Extraction Using 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 for above 32nm technology, and for 32nm and below technology only when QRC Techfiles are not specified.  
Optionally, when using preRoute (32nm and below), TQRC, IQRC, or Standalone QRC extraction, read in the QRC Techfile that contains the interconnect models used by these engine choices. For more information, see [Reading a QRC Techfile](#).
4. Generate a SPEF file using [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](#). The extracted values are accurate if the total capacitance scale factor has a deviation that is within 20 percent of 1.0 (that is, 0.80 to 1.20). You can also use the Perl script `spefCapCmp.pl` to compare the SPEF files, and generate scale factor for total capacitance. For more information, see [Comparing SPEF Files Using a Perl Script](#).
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 QRC signoff SPEF file.

The following figure shows the flow for generating RC scale factors.

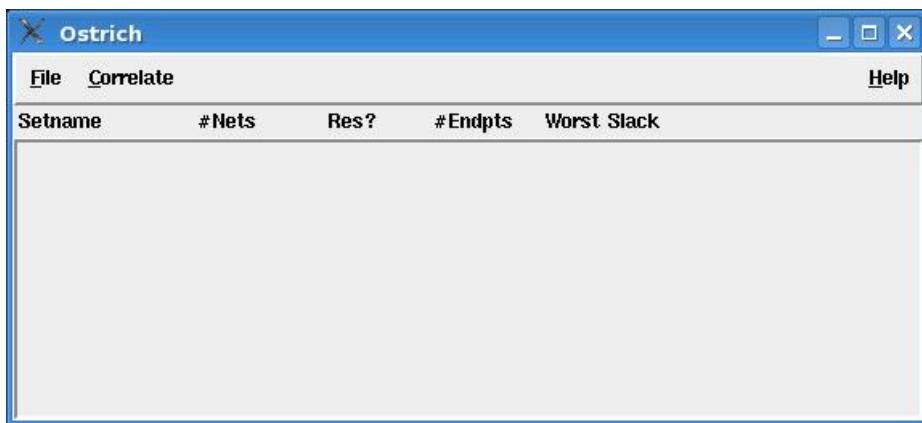


## Correlating SPEF Files Using the Ostrich Utility

Use Ostrich to correlate the SPEF files generated using native (postRoute -effortLevel low) extraction and signoff extraction. Ostrich is a standalone utility in the EDI System. 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:

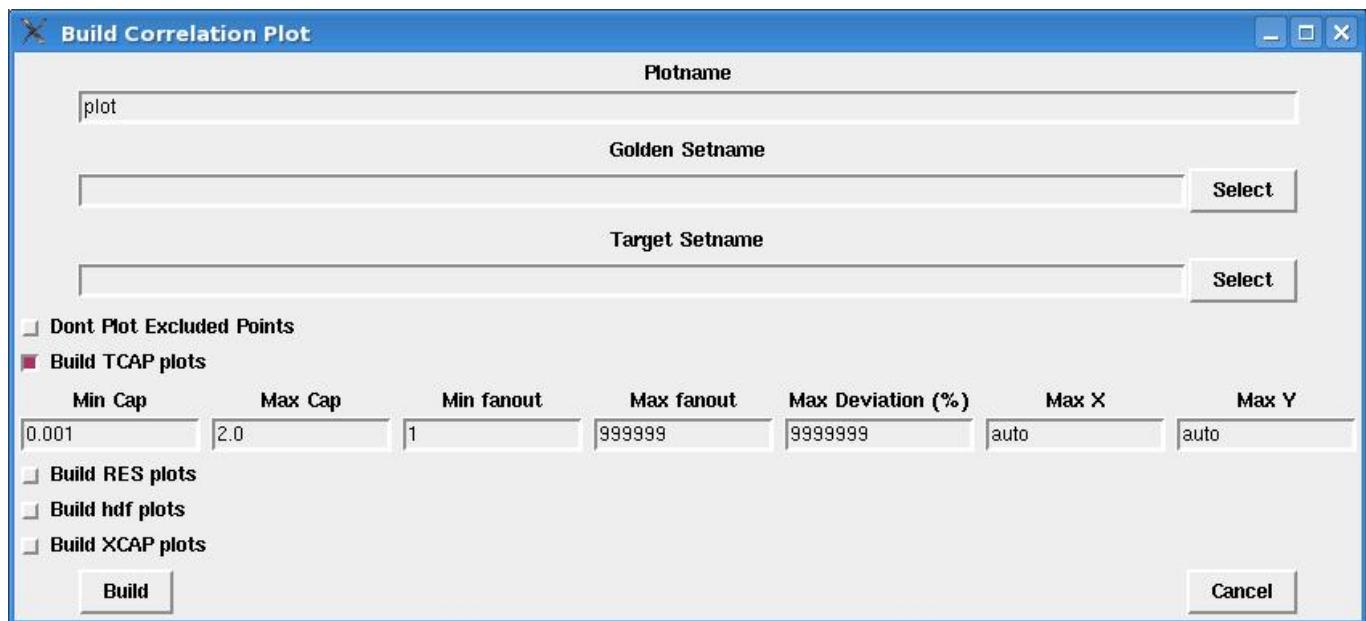
1. Type `ostrich` at the EDI System prompt. This opens the main Ostrich window.



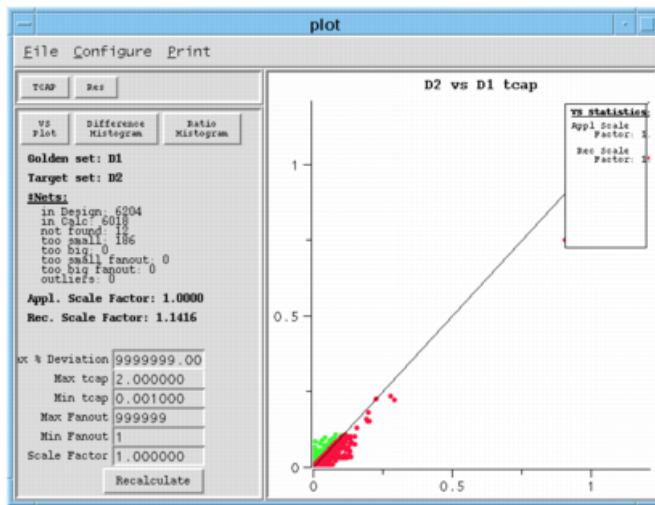
2. In the Ostrich window, click on *File - Import - SPEF*. This opens the SPEF Import form.



3. In the SPEF Import form, specify the name of the sign-off SPEF file and a name in the *Data Set Name* field. Next, click on the *Import* button to add the SPEF values in the Ostrich window. To correlate the resistance values, select the *Extract Resistance* option.
4. Similarly, import the native extraction SPEF file using the SPEF Import form.
5. Click on *Correlate - Build Plot* option in the Ostrich Window. This opens the Build Correlation Plot window.



6. Select the *Golden Setname* and *Target Setname* corresponding to the sign-off SPEF, and native extraction SPEF respectively.
7. Select *Build TCAP plots*, *Build RES plots*, and *Build XCAP plots* options. Click on the *Build* button.
8. Click on the Correlate - draw Plot - plot option in the Ostrich window. This opens the plot window.



The plot window displays the suggested scale factor.

## Comparing SPEF Files Using a Perl Script

Use the Perl script `spefCapCmp.pl` to compare the SPEF files generated by the `extractRC` command. You specify a range of total capacitance on a net to select the nets for comparison. This script resides in the `$ENCOUNTER/bin` directory. The Perl script generates the RC scale factors for total capacitance.

If the capacitance scaling is outside the range of +/- 20% (0.8-1.2), you need to reevaluate the flow for possible mistakes during parasitics generation.

To run the Perl script, use the following command at the UNIX prompt:

```
spefCapCmp.pl -ref signoff_file_name .spef -cmp file_name .spef [-minCap value ]
[-maxCap value ] [-ex openNetFile ]
```

|                                           |                                                                                                                                  |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>-ref signoff_file_name .spef</code> |                                                                                                                                  |
|                                           | Specifies the signoff/golden SPEF file.                                                                                          |
| <code>-cmp file_name .spef</code>         |                                                                                                                                  |
|                                           | Specifies the SPEF file generated by the <code>extractDetailRC</code> command.                                                   |
| <code>-minCap value</code>                | Specifies the minimum value, in picofarads, of the total capacitance on a net for the comparison script.<br><i>Default: 0.01</i> |
| <code>-maxCap value</code>                | Specifies the maximum value, in picofarads, of the total capacitance on a net.<br><i>Default: 5.0</i>                            |
| <code>-ex openNetFile</code>              | Specifies the name of the file that contains open nets to be excluded from the report file.                                      |

The Perl script generates the following output:

- Report file `spefCapCmp.rpt`, which contains the scale factors and statistical data for total capacitance. For more information, see Report File Example below.
- The `tcap.plot` file, which contains coordinates for plotting the comparison values for total capacitance.
- The `res.plot` file, which contains coordinates for plotting the comparison values for resistance.

To display a scatter plot of the comparison values, use the following command:

```
xgraph -P -nl file_name .plot
```

**Note:** `xgraph` is a public domain utility that you can download from the internet.

#### **Report File Example**

```
#Ref. Cap file: lambda_aftercrosstalkfix.spef
```

|                                                     |                                |      |
|-----------------------------------------------------|--------------------------------|------|
| #Cmp. Cap file:                                     | lambda_detailrc.spef           |      |
| #-----#                                             |                                |      |
| #                                                   | Total Capacitance Statistics   | #    |
| #-----#                                             |                                |      |
| #Total Capacitance range considered:                | 0.0100 - 5.0000 [pF]           |      |
| #Total number of nets:                              | 418399                         |      |
| #Total number of nets with nonzeoro lumped Cap:     | 213095                         |      |
| #Total number of nets discarded (small lumped Cap): | 152272.00                      |      |
| #Suggested Capacitance Scale Factor<br>0.9577)      | 0.9577 <---- (Ref. = FE(Cmp.)* |      |
| #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`](#) Tcl 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. The syntax for duplets and triplets is as follows:

```
value{ 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.  
No scale factor is applied to extraction with effortLevel signoff.

**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] |  
-rsh [-queue queue_name] [-resource resource_string] [-rsh_args rsh_arguments] |
```

```
-sgc [-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 EDI System. 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, TQRC, and Standalone QRC Extraction in Multi-CPU Mode

IQRC, TQRC, and Standalone QRC Extraction engines support distributed processing. You can use the [Multiple-CPU Processing Commands](#) to invoke IQRC, TQRC, and Standalone QRC Extraction in the multi-CPU mode.

### TCL Script for IQRC, TQRC, and Standalone QRC Extraction Invoked in the Distributed Mode

To run IQRC, TQRC, and Standalone 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, TQRC, and Standalone QRC Extraction Invoked in the Local Mode

To run IQRC, TQRC, and Standalone 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
```

## Setting Scale Factors in Capacitance Table File for PreRoute Extraction

**Note:** Captable is applicable for preRoute extraction for above 32nm technology, and for 32nm and below technology only when QRC Techfiles are not specified.

### Overview

Capability to set scale factors has been added in the header section of the capacitance table for preRoute extraction. If the capacitance table file is generated using EDI 10.1 release onwards, the following default scale factors will appear:

```
#preRoute_res 1.0
#preRoute_cap 1.0
#preRoute_clkcap 0.0
#preRoute_clkres 0.0
```

To apply scale factor values, uncomment the above values and provide proper scale factor values.

**Note:** If the same capacitance table is used for multiple corners and those corners have different scale factors, you will need to create unique versions of the file, each with their own unique scale factors.

If you have existing capacitance table files, add the following information in the header section:

```
---- Created using "First Encounter ...." on ....--#
#Process: captbl_cworst
VersionNumber 1.1
NominalTemperature 25
#preRoute_res 1.0
#preRoute_cap 1.0
#preRoute_clkcap 0.0
#preRoute_clkres 0.0
PROCESS_VARIATION...
LAYER M1
```

### Key Characteristics of the Feature

The scale factors provided in the capacitance table have a multiplicative effect on the scale factors set using the [create\\_rc\\_corner](#) or [update\\_rc\\_corner](#) command. During extraction, effective scale factors will be reported. Effective scale factor is the product of the scale factor set using `create_rc_corner` or `update_rc_corner` command and the scale factor specified in the capacitance table file. An example is provided below.

- Below are the scale factors set using `update_rc_corner` command after loading MMMC design with RC corners defined using `create_rc_corner` command:

```
<CMD> update_rc_corner -name rc_best -preRoute_clkres 0.98 -preRoute_clkc 0.99  
-preRoute_cap 1.01 -preRoute_res 1.1
```

```
.....
```

```
.....
```

Summary of Active RC-Corners :

Analysis View: setup\_view1

RC-Corner Name : rc\_best

RC-Corner Index : 0

RC-Corner Temperature : 25 Celsius

RC-Corner Cap Table : ''

RC-Corner PreRoute Res Factor : 1.1

RC-Corner PreRoute Cap Factor : 1.0

RC-Corner PostRoute Res Factor : 1

RC-Corner PostRoute Cap Factor : 1

RC-Corner PostRoute XCap Factor : 1

RC-Corner PreRoute Clock Res Factor : 0.98

RC-Corner PreRoute Clock Cap Factor : 0.99

RC-Corner PostRoute Clock Cap Factor : 1 [Derived from postRoute\_cap (effortLevel low)]

RC-Corner PostRoute Clock Res Factor : 1 [Derived from postRoute\_res (effortLevel low)]

- Below are the non-default scale factors specified in the capacitance table:

```
<CMD> update_rc_corner -name rc_best -cap_table new.captbl
```

Reading Capacitance Table File test.captbl ...

```
.....
```

```
.....
```

Cap Table preRoute\_cap Scaling Factor : 1.10000

Cap Table preRoute\_res Scaling Factor : 1.05000

Cap Table preRoute\_clkc Scaling Factor : 1.03000

Cap Table preRoute\_clkres Scaling Factor : 1.07000

- Below are effective scale factors applied during extraction. Note, these are multiplicative values:

```
<CMD> extractRC
```

Extraction called for design 'Design' of instances=42034 and nets=46484 using extraction engine 'preRoute' .

PreRoute RC Extraction called for design Design.

RC Extraction called in multi-corner(1) mode.

RCMode: PreRoute

Scale factors specified in the captable are multiplied with any user-specified scale factors. Resultant scale factors are as follows:

RC Corner Indexes 0

```
Capacitance Scaling Factor : 1.11100
Resistance Scaling Factor : 1.15500
Clock Cap. Scaling Factor : 1.01970
Clock Res. Scaling Factor : 1.04860
.....
```

- Earlier, the values for preRoute\_clkcap/res were derived from the values for postRoute\_cap/res. Starting EDI 10.1 release, if the values for preRoute\_clkcap/ res are specified in the capacitance table, then these values will be used for extraction. If, however, preRoute\_clkcap/res values are not specified in the capacitance table, then these will derived from values for postRoute\_cap/res.
- If any non-default scale factors are specified in the capacitance table, these will be reported while reading the capacitance table. An example is shown below.

```
<CMD> update_rc_corner -name rc_best -cap_table new.captbl
Reading Capacitance Table File test.captbl ...
```

```
.....  

.....  

Cap Table preRoute_cap Scaling Factor : 1.10000
Cap Table preRoute_res Scaling Factor : 1.05000
Cap Table preRoute_clkcap Scaling Factor : 1.03000
Cap Table preRoute_clkres Scaling Factor : 1.07000
```

---

# Calculating Delay

---

- [Overview](#)
- [Data Preparation](#)
  - [Operating Conditions](#)
  - [ECSM/CCS Libraries](#)
- [Delay Calculation Modes and Related Controls](#)
- [Choosing A Delay Calculation Engine](#)
- [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. The Encounter® Digital Implementation System (EDI System) provides three delay calculator engines that you can use:

- EDI System delay calculator engine  
Used to provide quick delay calculation for design prototyping, optimization, and general timing analysis.
- SignalStorm® delay calculator  
Used to perform final delay calculation and timing analysis. SignalStorm provides better accuracy, can calculate multi-driver nets, and uses more advanced effective current source model (ESCM) modeling.
- AAE delay calculator  
Used to perform delay calculation, timing analysis, and signal integrity (SI) analysis.

## Data Preparation

In order to perform delay calculation, you must have a placed design loaded in EDI System.

Delay calculation uses information from the following data files:

- A DSPF-format or SPEF-format netlist that contains the detailed parasitic resistance and capacitance of the interconnect, and the gates that are used to drive this interconnect.

- .lib file (Timing information)
- Timing constraints file
- LEF file

Delay calculation generates the following information:

- Optionally, an SDF file, which provides delay information for instances and RC interconnect.

## Operating Conditions

EDI 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.

## ECSM/CCS Libraries

By default, EDI System supports ECSM and CCS-based Liberty (.lib) timing libraries for performing delay calculation.

## Delay Calculation Modes and Related Controls

Delays are calculated differently, depending on the number of terminals (fanouts) a net has and the Elmore time constant. The following table lists the calculation modes and related controls used by the software for delay calculation.

|                  | <b>Fanouts <math>\geq 1,000</math></b> | <b>1,000 <math>&gt;</math> Fanouts<br/><math>\geq 100</math></b> | <b>100 <math>&gt;</math> Fanouts<br/>and<br/>Delay <math>&lt; 10\text{ns}</math></b> | <b>Delay <math>&gt; 10\text{ns}</math></b> |
|------------------|----------------------------------------|------------------------------------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------|
| <b>Algorithm</b> | Uses the default delay parameters.     | Uses simplified delay calculation mode.                          | Uses full RC delay calculation mode.                                                 | Uses simplified delay calculation mode.    |

|                                      |                                                                                                                                                           |                                                                                                                                            |                                                            |                                                                                      |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <b>Cell Delay</b>                    | <p>Uses lumped C lookup with a default value of 0.5 pF.</p> <p>The value is controlled by the <code>delaycal_default_net_load</code> global variable.</p> | 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>                    | <p>Uses the default value of 1 ns.</p> <p>The value is controlled by the <code>delaycal_default_net_delay</code> global variable.</p>                     | Uses Elmore delay.                                                                                                                         | Uses full RC.                                              | Uses Elmore delay.                                                                   |
| <b>Driver Slew Rates</b>             | <p>Uses the default value of 0 ps.</p> <p>The value is controlled by the <code>delaycal_input_transition_delay</code> global variable.</p>                | 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.<br>(Slews are degraded across interconnect.) | None                                                                                 |
| <b>Controls</b>                      | <p>Fanout threshold is controlled by the <code>delaycal_use_default_delay_limit</code> global variable. The default value is 1,000.</p>                   | 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> |

## Choosing A Delay Calculation Engine

1. Choose *Options - Set Mode - Specify Delay Calculation Mode*.

The Delay Calculation Mode form appears.

2. Select *Encounter-DC* to use the EDI System delay calculator, or *SignalStorm* to use the SignalStorm delay calculator.

Alternatively, you can specify the delay calculation engine by issuing the `setDelayCalMode` text command:

- Issue the following command to specify the EDI System delay calculator:

```
setDelayCalMode -engine feDc
```

- Issue the following command to specify the SignalStorm delay calculator:

```
setDelayCalMode -engine signalstorm
```

- Issue one of the following commands to specify the AAE delay calculator:

```
setDelayCalMode -engine aae
```

```
setDelayCalMode -engine default
```

**Note:** The default delay calculator is `aae`.

## Running Delay Calculation

The delay calculation engine set using the `-engine` parameter of the [`setDelayCalMode`](#) command is called automatically during timing analysis. You can choose to write the delays to a standard delay format (SDF) file using the [`write\_sdf`](#) command.

For example, the following command saves the results to the SDF file named `TOPCHIP_SP.sdf`:

```
write_sdf TOPCHIP_SP.sdf
```

## Calculating Delay in Multi-Thread Mode

Multi-threaded delay calculation is automatically enabled when you configure multiple-CPU processing for an MMMC design wherein full-chip delay calculation has not yet occurred. You can invoke multi-threaded delay calculation by running any command (for example, [`optDesign`](#), [`timeDesign`](#), [`report\_timing`](#), and so on) that needs timing information.

Use the following method:

Run the following command before running `report_timing` (any command that requires timing information):

```
setMultiCpuUsage
```

**Example:**

```
setMultiCpuUsage -localCpu 4
```

```
report_timing
```

---

# Timing Analysis

---

- [Overview](#)
- [Timing Analysis Features](#)
- [MMMC-On By Default Functionality](#)
- [Before You Begin](#)
- [Calculating Clock Latency](#)
- [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 Synopsys 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.

### **Wireload Model Generation in Hierarchical and Flat Format**

The delay information in the technology library applies to the timing arcs from input ports to output ports of each cell and the corresponding wire delays. The cell delays and the wire delays are expressed as a function of the physical characteristics of the nets in the design, such as wire capacitance and wire resistance. A wireload model uses the fanout count of a net and estimates its capacitance and resistance. The wireload models in the hierarchical format are generated for cells and instances based on external nets (hierarchical view). The wireload models in the flat format are generated for cells and instances based on internal nets (flat view).

### **Minimum and Maximum Timing Analysis**

To read in libraries with multiple operating conditions for minimum and maximum analysis, you can:

- Create a MMMC configuration file. You can use the EDI System GUI.
- Specify the operating conditions by using the `create_delay_corner -opcond` command.
- Specify the `setAnalysisMode` command.

### **Timing Analysis Ideal and Propagated Modes**

| <b>setAnalysisMode</b> |                                  | <b>Clock Propagation</b> | <b>Clock Latency</b> |
|------------------------|----------------------------------|--------------------------|----------------------|
| -skew false            |                                  | Forced Ideal             | No Effect            |
| -skew true             | -clockPropagation<br>forcedIdeal | Forced Ideal             | SDCs in Effect       |

|            |                                 |                 |                  |
|------------|---------------------------------|-----------------|------------------|
| -skew true | -clockPropagation<br>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 EDI v11.1, all designs need to be MMC-based to initialize into the EDI 11.1 release. The earlier versions of the software supported the two-corner, single-mode (minimum/maximum) operation. Starting from release 11.1 onwards, the EDI system 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 *EDI Menu Reference*.
- Specify the operating conditions to use for timing analysis.  
Use the operating conditions to specify process, voltage, and temperature (PVT) values. Operating conditions are defined in the timing library and read into the Encounter session when you import the design. You can use a single set of operating conditions for setup and hold analysis, or you can specify minimum and maximum conditions.
- 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 MMC configuration file you can define up to three different RC corners - typical, best, and worst.
- Specify the analysis mode you want to use for timing analysis. There are three types of

analysis modes: single, best-case worst-case (BC-WC), and on-chip variation. For more information, see "Specifying Timing Analysis Modes".

For more information, see "Importing and Exporting Designs" chapter of the *EDI User Guide*.

## Calculating Clock Latency

The EDI 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 EDI 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.

## Specifying Timing Analysis Modes

The Encounter software provides different timing analysis modes and performs different calculations for setup and hold checks for each mode. The timing analysis modes are divided as follows:

- Single Timing Analysis Mode

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.

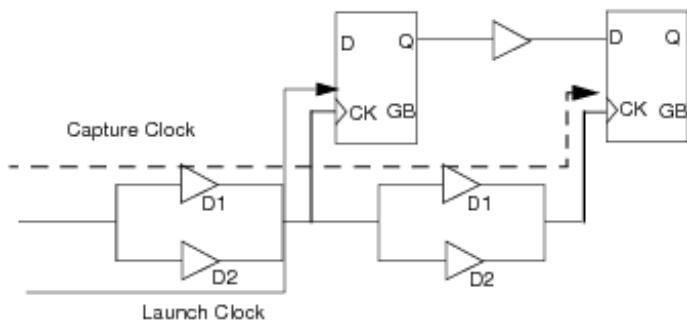
In OCV mode, the software calculates clock and data path delays based on minimum and maximum operating conditions for setup analysis and vice-versa for hold analysis.

## 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.

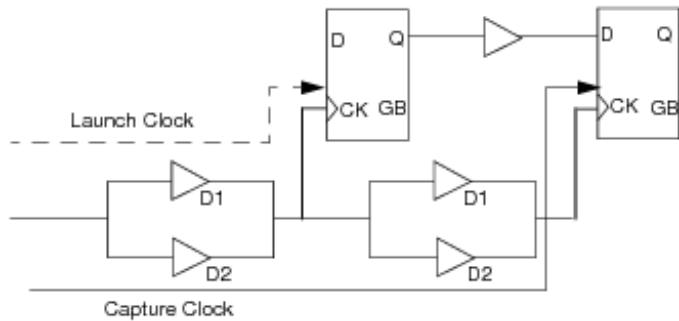
The following figure shows a setup check with late (shown in solid line) launch clock and early capture clock (shown in dotted line).

Figure 29-1 Setup Check



The following figure shows hold check with early launch clock (shown in dotted line), and late capture clock (shown in solid line).

Figure 29-2 Hold Check



## Single Timing Analysis Mode

In this mode, the Encounter software uses a single set of delays (using one library group) based on one set of process, temperature, and voltage conditions. To set the timing analysis mode as single, use the `-analysisType single` parameter of the [`setAnalysisMode`](#) command.

### Setup Check in Single Timing Analysis Mode

For setup check, the software checks the late launch clock and late data paths against early capture clock path.

For zero slack value in a setup check, the following condition should be met:

$$\text{launch clock late path} + \text{data clock late path} \leq \text{capture clock early path} + \text{clock period} - \text{setup}$$

### Hold Check in Single Timing Analysis Mode

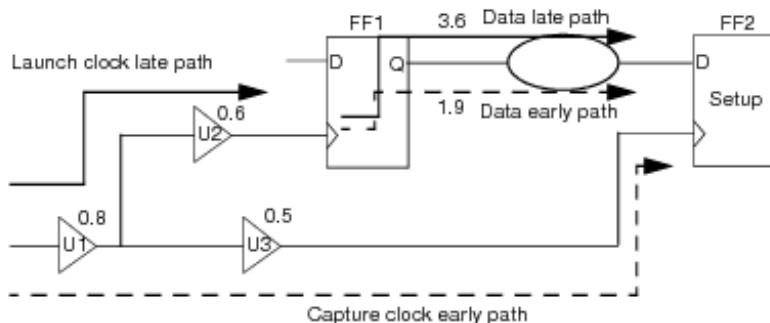
For hold check the software compares the early arriving data against the late arriving clock at the endpoint.

For zero slack value in a hold check, the following condition should be met:

$$\text{launch clock early path} + \text{data clock early path} \geq \text{capture clock late path} + \text{hold}$$

### Example 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

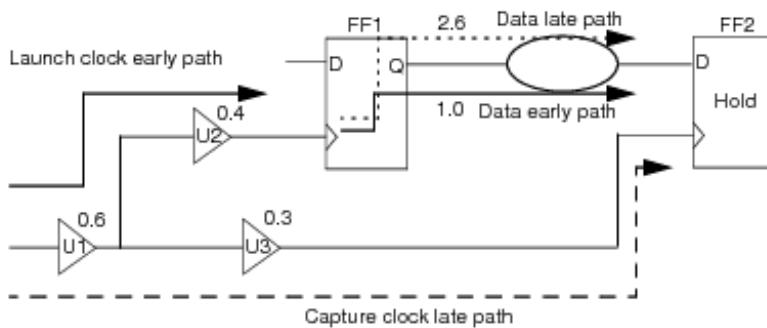
Data arrival time =  $1.4 + 3.6 = 5$

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
-cap_table /icd/libs/tech/6m1v-wc.capTbl

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 Encounter software considers two operating conditions. The software checks both operating conditions in one timing analysis run.

To set the timing analysis mode as BC-WC, use the `-analysisType bcWC` parameter of the [`setAnalysisMode`](#) command.

You can use the `set_clock_latency` constraint to set the source latency for a clock in both ideal and propagated mode for setup and hold checks. You can also use the constraint to set the network latency for an ideal clock. The specified source or network latency affects the early and late clock paths for both capture and launch clocks for both min and max operating conditions. The software considers the network latency that you set using the `set_clock_latency -max` or `-min` constraint for ideal clocks only.

### Setup Check in BC-WC Mode

For setup check, the software calculates delay values from the Max library group for data arrival time, and network delay of both launch and capture clocks (in propagated mode).

The software scales the delay values using the operating condition that you 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<br>(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>  |

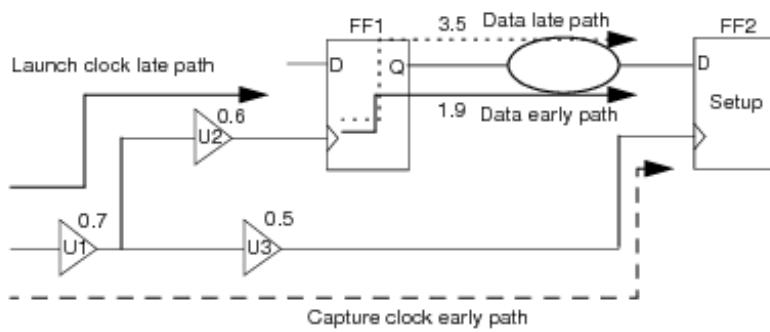
The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

| 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

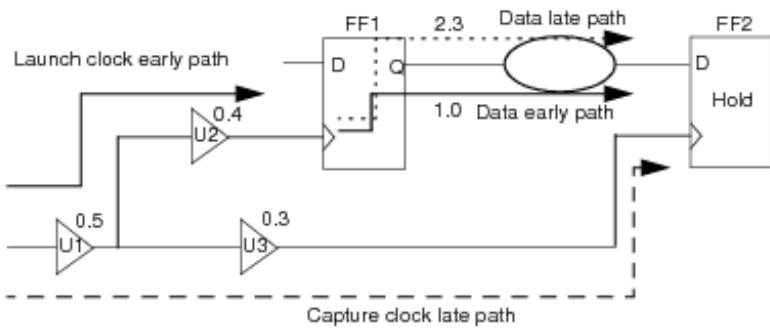
Data arrival time =  $1.3 + 3.5 = 4.8$

Data required time =  $4 + 1.2 - 0.2 = 5$

Slack =  $5 - 4.8 = 0.2$

#### **Example 29-4 Hold Check in BC-WC Timing Analysis Mode**

The following figure shows the hold check on the path from FF1 to FF2.



The software uses the Min library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock early path delay =  $0.5 + 0.4 = 0.9$

Data early path delay = 1.0

Capture clock late path delay =  $0.3 + 0.5 = 0.8$

Hold = 0.1

Data arrival time =  $0.9 + 1 = 1.9$

Data required time =  $0.1 + 0.8 = 0.9$

Slack =  $1.9 - 0.9 = 1$

### **Performing Timing Analysis in BC-WC Analysis Mode**

To perform timing analysis in BC-WC analysis mode, complete the following steps:

1. 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
-cap_table /icd/libs/tech/6mlv-wc.capTbl
create_rc_corner -name my_bc_corner_worst
-cap_table /icd/libs/tech/6mlv-bc.capTbl

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><br>Or,<br><code>set_clock_latency -source -late value</code>   |
| Capture clock early path (min)   | <code>set_clock_latency -source -early -min value</code><br>Or,<br><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

`set_clock_latency -min value`

### 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:

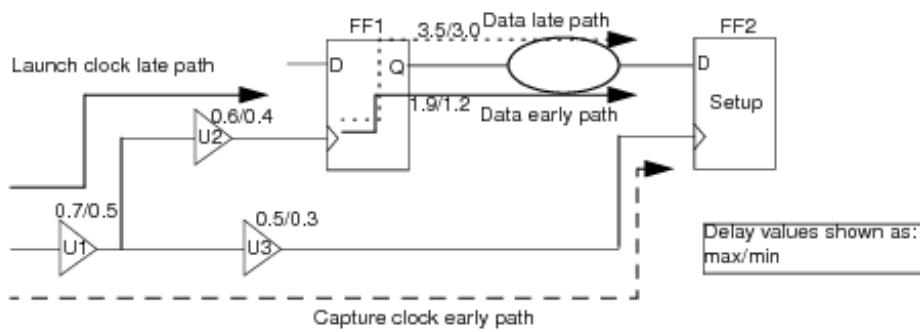
| Clock Path (Operating Condition) | Constraint Used                                                                                                        |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Launch clock early path (min)    | <code>set_clock_latency -source -early -min value</code><br>Or,<br><code>set_clock_latency -source -early value</code> |
| Capture clock late path (max)    | <code>set_clock_latency -source -late -max value</code><br>Or,<br><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

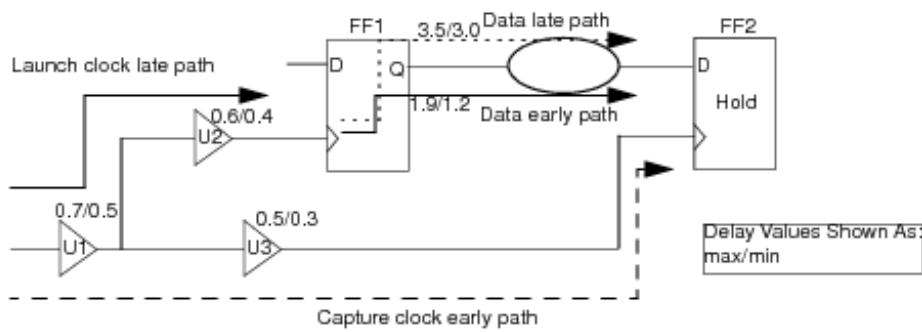
Data arrival time =  $1.3 + 3.5 = 4.8$

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.

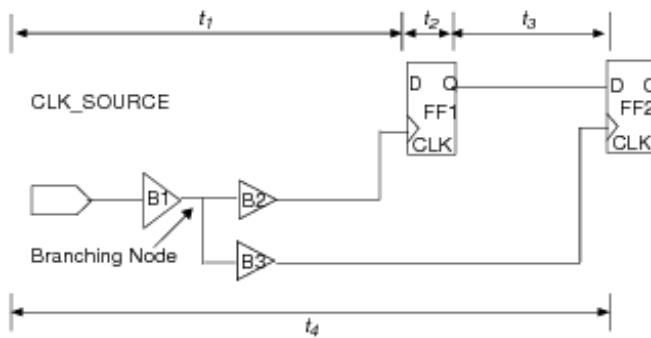
You can introduce early or late delay variations by using the `setAnalysisMode -analysisType onChipVariation` or by using derating factors in BcWc analysis mode. In CPPR mode, the difference between late and early delays is removed for common clock tree segments of the launching and latching devices.

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 29-3 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 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:

$$B1 * Late Derate - B1 * Early Derate = t_{CPPR}$$

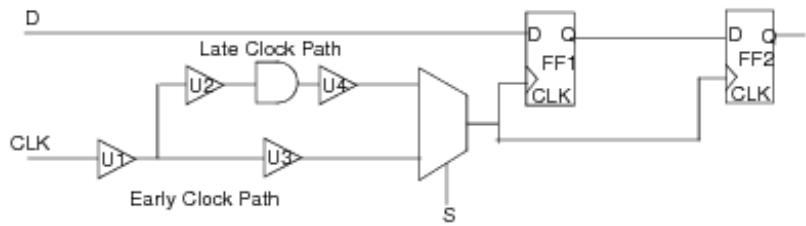
Therefore,

$$t_{CPPR} = 1.0\text{ns} * 1.0 - 1.0\text{ns} * 0.9 = 0.1\text{ns}$$

### 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.

Encounter 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 Encounter, 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 true
```

3. Set the derating values.

```
set_timing_derate -late 1 -early 0.9 -clock
```

4. Generate timing report. You use the `report_timing` command to remove delay pessimism from paths that have a portion of the clock network in common.

```
report_timing
```

### 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

| Instance | Arc         | Cell      | Delay | Arrival Time | Required Time |
|----------|-------------|-----------|-------|--------------|---------------|
|          | clk ^       |           |       | 0.000        | 0.508         |
| ck_0     | A ^ -> Y ^  | BUFX2     | 0.091 | 0.091        | 0.598         |
| ck_1     | A ^ -> Y ^  | BUFX2     | 0.097 | 0.188        | 0.695         |
| ck_2     | A ^ -> Y ^  | BUFX2     | 0.094 | 0.282        | 0.789         |
| ck_3     | A ^ -> Y ^  | BUFX2     | 0.092 | 0.374        | 0.882         |
| ck_4     | A ^ -> Y ^  | CLKAND2X2 | 0.150 | 0.524        | 1.032         |
| reg_1    | CK ^ -> Q v | DFFRHQX1  | 0.288 | 0.812        | 1.320         |
| t_1      | A ^ -> Y ^  | BUFX8     | 0.111 | 0.923        | 1.431         |
| t_2      | A ^ -> Y ^  | BUFX8     | 0.092 | 1.015        | 1.523         |
| t_3      | A ^ -> Y ^  | BUFX8     | 0.092 | 1.107        | 1.615         |
| t_4      | A ^ -> Y ^  | BUFX8     | 0.092 | 1.199        | 1.707         |
| t_5      | A ^ -> Y ^  | BUFX8     | 0.092 | 1.291        | 1.799         |
| t_6      | A ^ -> Y ^  | BUFX8     | 0.092 | 1.383        | 1.891         |

|       |            |          |       |       |       |
|-------|------------|----------|-------|-------|-------|
| t_7   | A ^ -> Y ^ | BUFX8    | 0.092 | 1.475 | 1.983 |
| t_8   | A ^ -> Y ^ | BUFX8    | 0.092 | 1.567 | 2.075 |
| t_9   | A ^ -> Y ^ | BUFX8    | 0.092 | 1.660 | 2.167 |
| t_10  | A ^ -> Y ^ | BUFX8    | 0.088 | 1.747 | 2.255 |
| t_11  | B ^ -> Y ^ | NAND2X1  | 0.066 | 1.813 | 2.321 |
| t_12  | A ^ -> Y ^ | INVX1    | 0.037 | 1.850 | 2.358 |
| reg_2 | D v        | DFFRHQX1 | 0.000 | 1.850 | 2.358 |

## Analyzing Timing Problems

In addition to the detailed timing violation report, the following report commands are helpful in analyzing timing problems:

- [check\\_timing](#)  
Performs a variety of consistency and completeness checks on the timing constraints specified for a design. Use the `check_timing` command after setting all constraints, but before any timing analysis commands, such as `report_timing`, to verify that the timing environment is complete and self-consistent.
- [get\\_property](#)  
Retrieves timing information for the specified pin property on the given pin.
- [report\\_analysis\\_coverage](#)  
Provides information about the timing checks in the design.
- [report\\_annotated\\_check](#)  
Reports coverage of annotated timing checks.
- [report\\_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.

- [reportCapViolation](#)

Reports the nets that exceed the maximum capacitance constraints set by the timing library and timing constraints file.

- [reportClockDomains](#)

Reports the clock domain setting for building the timing graph.

- [reportTranViolation](#)

Reports the nets that exceed the maximum transition constraints set by the timing constraints file.

- [report\\_constraint](#)

Reports constraint information of current design.

- [reportFanoutViolation](#)

Reports all the pins that exceed the maximum fanout constraints in the timing library and timing constraints file.

- [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.

## 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](#)
  - [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

Encounter provides the Global Timing Debug feature for debugging the timing results. The various Timing Debug forms provide easy visual access to the timing reports and debugging tools. Encounter provides different timing debug feature depending on the timing mode that you have selected.

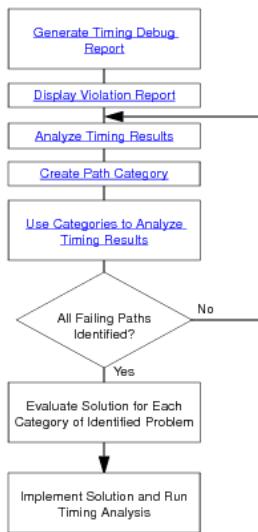
You can group all paths that are failing for the same reason and apply solutions for faster timing closure. You can cross-probe between the timing paths in the timing report and Encounter design display area.

**Note:** If you have a previously saved timing debug report, you can use the timing debug feature even when the design is not loaded in the Encounter session.

### Timing Debug Flow

You can generate a detailed violation report to list the details of all violating paths. You can then use the timing debug capability to visually identify problems with critical paths in this report. After identifying the problems, you group all paths with the same problem under a single category. You can define several categories to capture all problems related to the violating paths before fixing the problems and running timing analysis again.

Following is the flow for debugging timing results.



**Note:** The file gtd.pref.tc1 is loaded automatically at launch.

## Generating Timing Debug Report

Encounter uses a machine readable timing report to display timing debug information. The report is generated in the ASCII format and contains details of all violating paths. By default, the report has .mtarpt extension.

To generate a violation report, use one of the following options:

- Use [report\\_timing -machine\\_readable](#) command
- Use [timeDesign -timingDebugReport](#) command
- Use the Generate option in the [Display/Generate Timing Report](#) form.

You can also generate text-format report from a machine readable report.

To generate the text report, use the following:

- Use the [write\\_text\\_timing\\_report](#) command
- Use the [Write Textual Timing Report](#) form

## Displaying Violation Report

To analyze the timing results, you need to load the machine readable timing report in Encounter.

To display the violation report, use one of the following options:

- Specify the file name in the [Display/Generate Timing Report](#) form.  
**Note:** To select an existing file, deselect the *Generate* option before clicking on the directory icon to the right of the *Timing Report File* field.

By default, the global timing debug engine uses the following command to generate a machine-readable timing analysis report for the GUI display:

```
report_timing -machine_readable -max_points 10000 -max_slack 0.75
```

- Use the [load\\_timing\\_debug\\_report](#) command.

**Note:** Use the Append to Current Report option in the Display/Generate Timing Report form to load multiple reports in a single session.

## Analyzing Timing Results

Encounter provides Timing Debug feature to visually analyze timing problems.

You analyze the following data in the [Timing Debug](#) form:

- Visual display of passing and failing paths as a histogram. Failing paths are represented in red and passing paths are represented in green color. The goal of timing debug process is to identify paths that fall in red category.
- Details of the critical paths in the Path list. You identify a critical path in this list for further analyses using the Timing Path Analyzer form.
- Visual display of paths reported in different timing reports. When you load multiple debug reports in a single timing debug session, the paths are displayed in different colors corresponding to the report file they are coming from. You can move the cursor over a path to display the name of the report file.

You analyze the following data in the [Timing Path Analyzer](#) form:

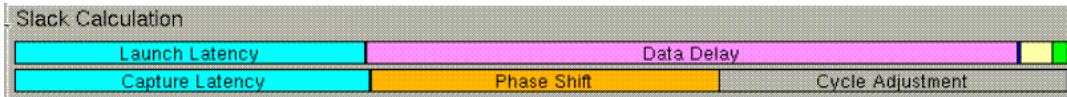
- Slack calculation bars for arrival and required times. You can identify clock skew issues, latency balancing or large clock uncertainty issues using these bars.
- The following examples illustrate the problems that you can identify using the slack calculation bars in the Timing Path Analyzer form.

Example 30-1



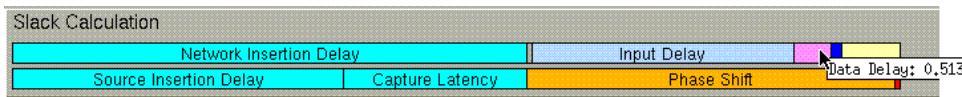
Launch and capture latency components are not aligned. Therefore there can be large clock-latency mismatch in this path.

Example 30-2



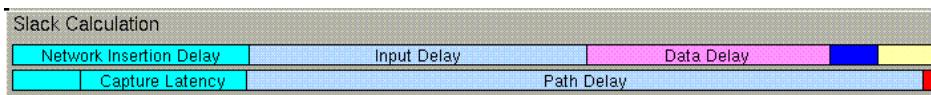
The cycle adjustment bar in the required time indicates presence of multicycle path.

Example 30-3



Large input delay in an I/O path is represented by the blue bar in the arrival time.

Example 30-4



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    | intance 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.

- 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 EDI System 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.

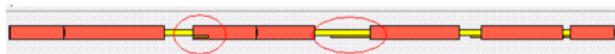
Timing interpretation for the path. This feature provides rule-based path analysis help you discover sources of potential timing problems in a path. By default, the software performs the following checks the following rules:

- Path structure
  - Transparent Latch in Path
  - Clock Gating
  - Hard Macros
  - HVT Cells
  - Buffering List
  - Net Fanout
  - Level Shifters
  - Isolation Cells

- Timing and constraints
  - Large Skew
  - Divider in Clock Path
  - Total SI Delay
  - SI Delay
  - External Delay
- Floorplan
  - Fixed Cells
  - Distance from start to end
  - Distance of repeater chain
  - Detour
  - Multiple power domains
- DRVs
  - Max transitions
  - Max capacitance
  - Max fanout

You can customize the type of timing information reported. The *Edit Timing Interpretation* GUI lets you add, modify, or delete rules you want the tool to check and report.

- Timing bar to analyze delays associated with instances and nets in a path. Use this information to identify issues related to large instance or net delays, repeater chains, paths with large number of buffers, and large macro delays. The small bars superimposed on net delays or within element delays show incremental (longer or shorter) delays due to noise effects:

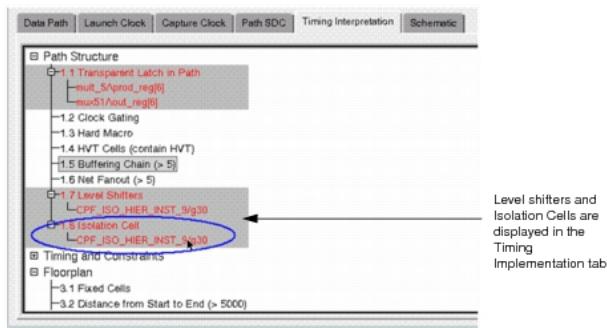


- Hierarchical representation of the path in the Hierarchy View field. This representation of the path-delay shows the traversal of a path through the design hierarchy drawn on the time axis. A longer arrow means that there are more instances on its path. Use this information to see the module where the path is spending more time or to identify inter-partition timing problems.

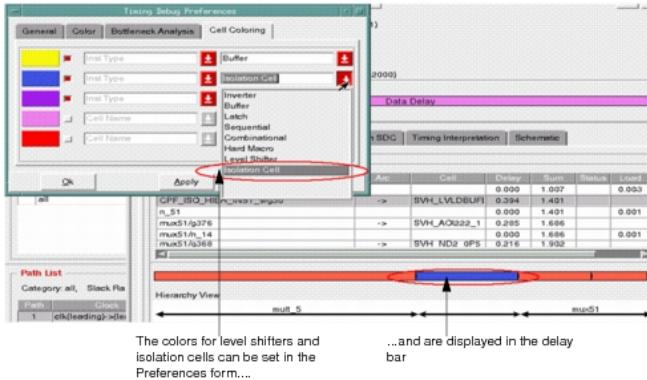
## **Viewing Power Domain Information**

While debugging critical paths on MSV designs, it is useful to be able to identify power domains and low power cells. The Timing Debug feature displays this information in the following ways:

- The level shifters and isolation cells are listed in the Timing Interpretation tab of the Timing Path Analyzer.



- 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 EncounterETS, you can either define your own category or use predefined categories to group your paths. The categories that you define are then displayed in the Timing Debug form in the Path Category field in the Analysis tab. The form also displays the paths associated with each category.

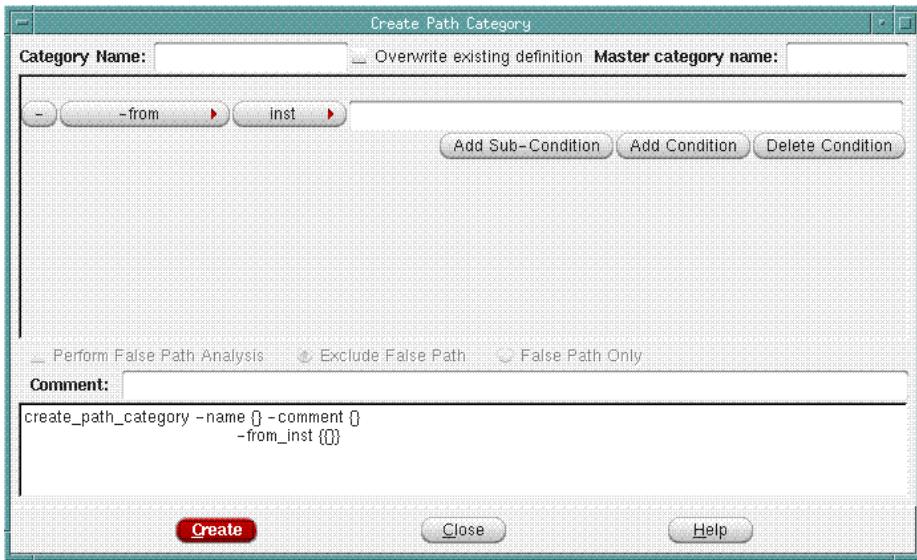
### Creating Predefined Categories

There are following predefined categories:

- Basic Path Group  
Creates standard path categories according to basic path groups.
- Clock Paths  
Creates categories according to launch clock - capture clock combinations.
- Hierarchical Floorplan  
Creates categories according to the hierarchical characteristics of a path.
- Hierarchical Port  
Creates categories according to the hierarchical characteristics of a path.
- View  
Creates categories according to the view for which the path was generated..
- False Paths  
Creates a category with paths defined as False paths.
- Bottleneck  
Creates categories based on instances that occur often in critical paths.
- 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.

### Creating New Categories

To define a new category, use the Create Path Category form. The Create Path Category form contains drop-down menus with conditions that you use to define a path category. The conditions are characteristics that a path must have to be added to the named category. You can define multiple conditions that a path must meet to be added to the category.



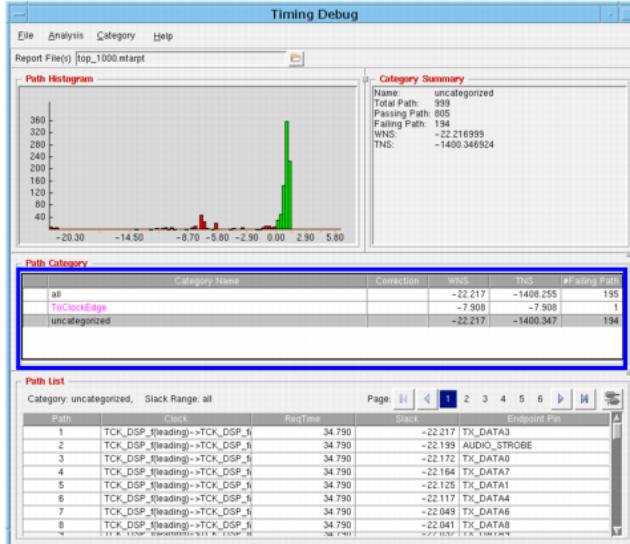
The category that you create is added in the Timing Debug form in the Path Category field in the Analysis tab. All paths that meet the conditions set for this category are grouped under the category name. Paths are separated automatically according to MMMC views into different categories, for example:

```
CLOCK1<View_test_mode>
CLOCK2<View_mission_mode>
```

Double-click on the category name in the Timing Debug form 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.

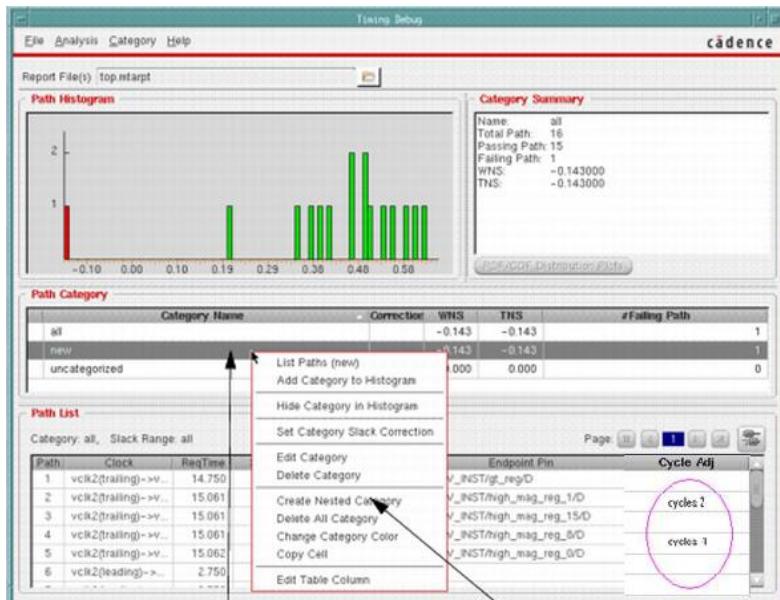
Following figure shows categories created by the Create Path Category form.



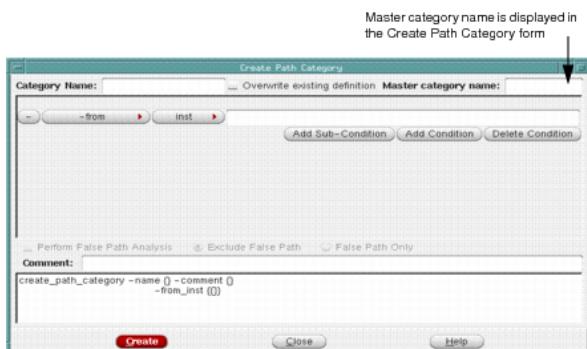
#### Creating Sub-Categories through the GUI

In the GUI, you can create a sub category as follows:

1. Right-click on a path category, and select *Nested Category*.



The Create Path Category form is displayed.



1. In the *Master category name* field, the name of the category you selected previously is displayed. Create one or more subcategories.

**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.

#### 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          | Correction |
|               | all                    |            |
|               | master_category1       |            |
| (1)           | nested_category_1a     |            |
| (2)           | nested_category_2      |            |
|               | (1) nested_category_1b |            |
|               | uncategorized          |            |

In this example:

- `master_category1` is the master category
- `nested_category_1a` and `nested_category_1b` are the sub-categories of `master_category1`.  
The prefix (1) is displayed with `nested_category_1a` and `nested_category_1b`.
- `nested_category_2` is the sub-category of `nested_category_1a`.  
The prefix (2) is shown with `nested_category_2`.

#### Hiding path categories

To remove a path category from the histogram display, right-click on a path and select *Hide Category*. The category name in the category list is not hidden, but is marked with an "H" as hidden.

#### Reporting Path Categories

To generate a report containing information about path categories, use the following options:

- Use the `write_category_summary` command
- Use the Write Category Report File GUI

The text file contains the following information:

- Category name
- Total number of paths
- Number of passing paths

- Number of failing paths
- Worst negative slack
- Total negative slack
- TNS

Sample report:

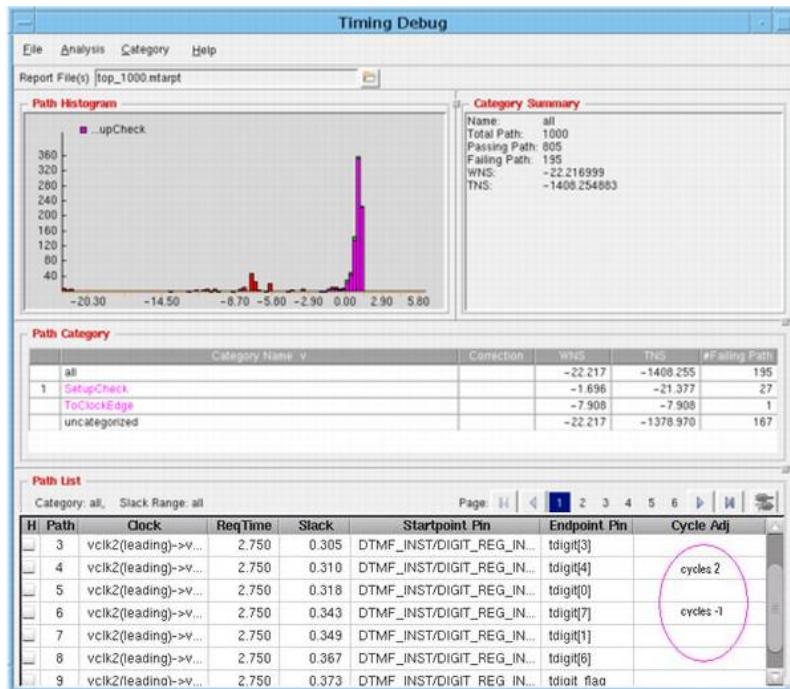
| <b>Category name</b>    | <b>Total path</b>                                                                                                       | <b>Passing Path</b> | <b>Failing path</b> | <b>WNS</b> | <b>TNS</b> |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------|---------------------|---------------------|------------|------------|
| test_clock<view_test_   | 3869                                                                                                                    | 768                 | 3101                | -5.699     | -4802.857  |
| 100MHz_1.00V>           | Clock Domain Analysis                                                                                                   |                     |                     |            |            |
| @->test_clock<view_test | 484                                                                                                                     | 55                  | 429                 | -2.292     | -378.432   |
| _100MHz_1.00V>          | Clock Domain Analysis                                                                                                   |                     |                     |            |            |
| my_clk-><view_mission   | 1                                                                                                                       | 2                   | 11                  | 1.931      | -1.931     |
| _166MHz_1.08V>          | Clock Domain Analysis                                                                                                   |                     |                     |            |            |
| my_clk_2x<view_mission  | 156                                                                                                                     | 154                 | 2                   | -.013      | -0.21      |
| _166MHz_1.08V>          | Clock Domain Analysis                                                                                                   |                     |                     |            |            |
| cat1875                 | 77                                                                                                                      | 13                  | 64                  | -3.524     | -100.893   |
|                         | Category of paths that cross i_1875 and start with test_clock - need to change the uncertainty value - advised --by Don |                     |                     |            |            |

## Using Categories to Analyze Timing Results

You can use the categories that you create to group the timing paths in the Timing Debug form. The Timing Debug form displays the category details in the Path Category field. You can perform the following tasks in the Timing debug form to analyze the timing results:

- Double-click on any category to display the details of the paths grouped in that category in the Path List field.
- Right-click on the category name and select Add to Histogram option. The paths related to the selected category are highlighted in a different color in the histogram. This gives you a visual representation of the number of paths that meet the conditions in that category and can possibly have the same timing problem.

For example, in the following figure the *SetupCheck* category was added to the histogram.

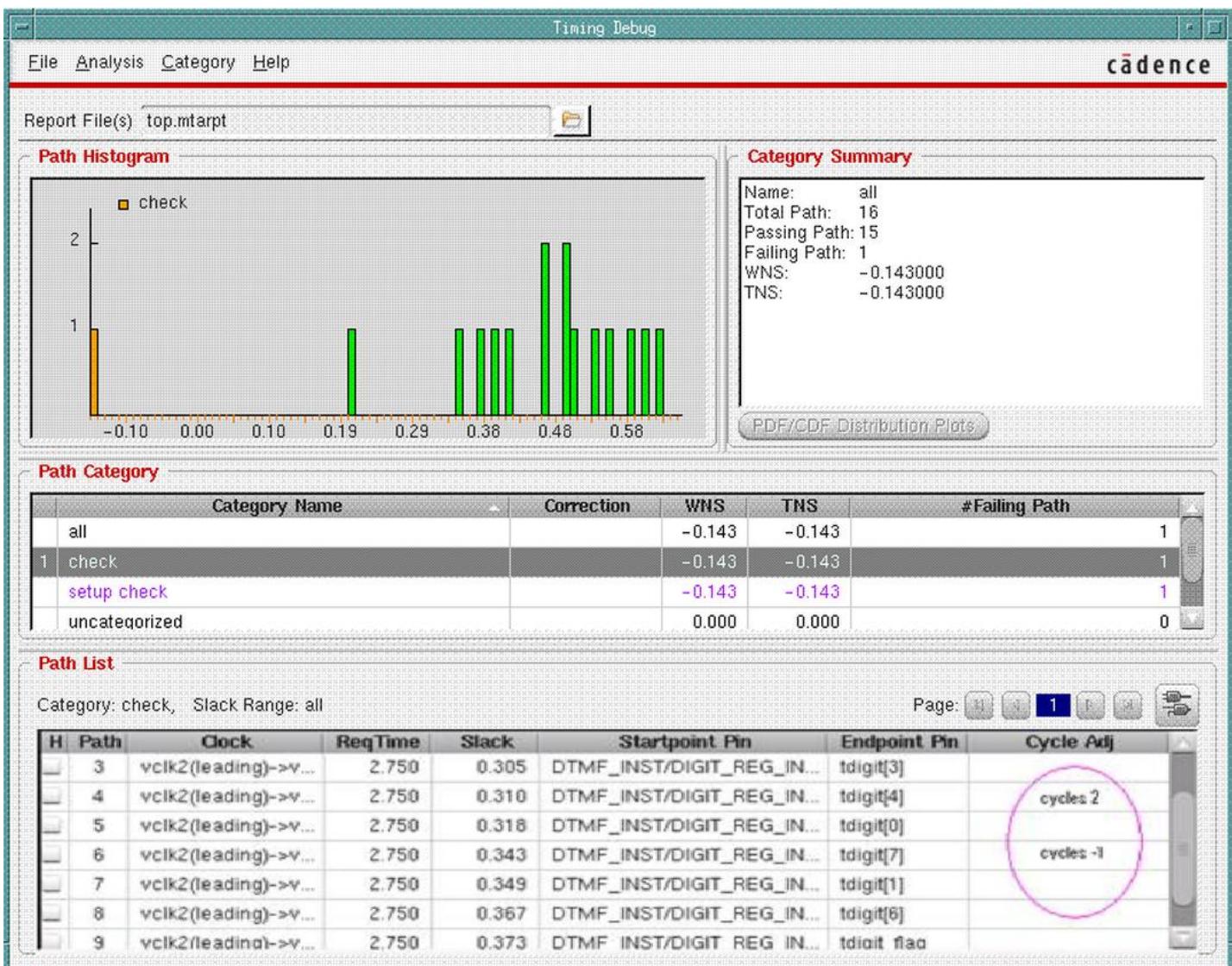


Analyzing the resulting the Timing Debug form gives you information for fixing problems related to larger sets of timing paths. After identifying the problems, you can make the required changes such as modify floorplan, script or SDC files and run timing analysis again for further analysis.

### Analyzing MMMC Categories

Paths are separated automatically according to MMMC views into different categories, for example, the following figure shows two categories based on MMMC views:

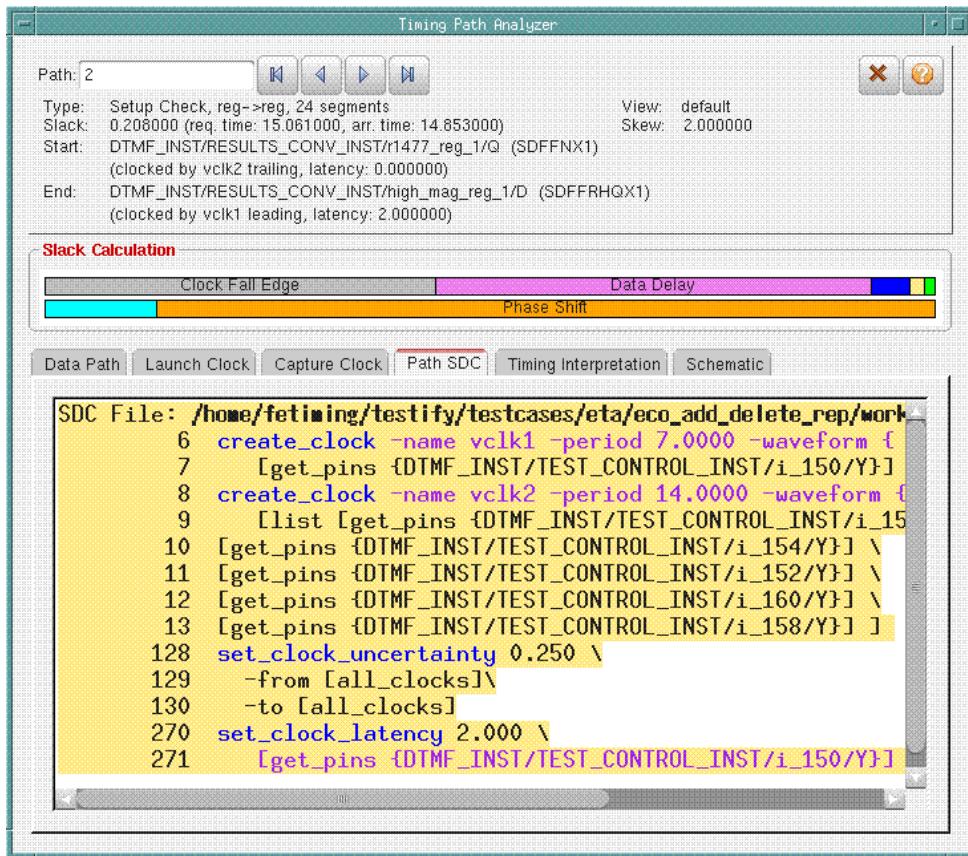
- view\_mission\_140MHz\_1.0V
- view\_mission\_166MHz\_1.8V



1. Right-click on one of `view_mission_140MHz_1.ov` and choose *List Paths*.
2. Right-click on one the paths and choose *Show Timing Path Analyzer*.

The Timing Path Analyzer is displayed.

1. Click on the [Path SDC](#) tab to display the SDCs:



Note that the SDCs relative to mode `mission_140MHz` that produced the path are highlighted.

### Manual Slack Correction of Categories

Use the Set Category Slack Correction form to specify the estimated slack correction for the selected category of paths. A slack correction that you apply to a category modifies all the paths in that category. If a path belongs to several categories, all the correction from the categories are added. The worst negative slack and total negative slack values of a category can be affected by the correction applied to another category.

Once you enable the slack correction, the histogram is updated to reflect the slack correction. An asterisk (\*) is added next to the slack value of paths that belong to this category in the *Path List* field in the Timing Debug window. Paths are reordered based on new specified slack. This allows you to filter out the paths that can be fixed and work on the remaining paths.

To access the Set Category Slack Correction form complete the following steps:

1. Choose *Timing - Debug Timing*.
2. Right click on the category name in the *Path Category* field.
3. Choose the *Set Category Slack Correction* option.



To disable the set slack correction value:

1. Right click on the category name in the *Path Category* field.
2. Choose the *Deactive Category Slack correction* option.

## Editing Table Columns

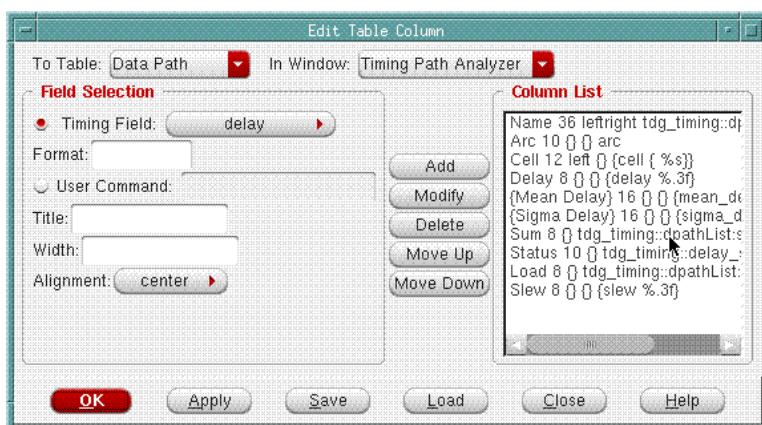
You can customize the dimensions and contents of table columns to suit your needs.

1. Open the *Timing Debug* or *Timing Path Browser* form., then right-click on a path.

A drop-down menu is displayed.

1. Select *Edit Table Column*.

The Edit Table Column form is displayed.



1. Choose the timing window that contains the table to want to customize.
2. Choose the table whose columns you want to customize. The selections change according to the timing window you choose.
3. 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 ""
    }
}
```

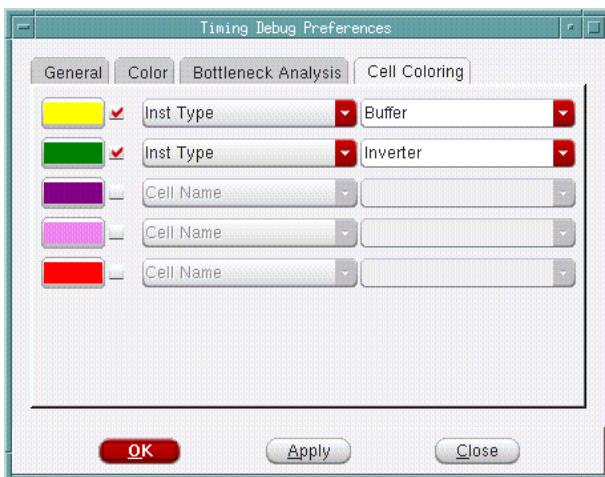
1. Build the column list.
  - *Add* adds a column to the column list.
  - *Modify* let you modify characteristics. Click on a column in the column list. Edit the information, then click *Modify*.

- *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.

3. (Optional) Click *Load*. This opens the GTD (Global Timing Debug) Preferences form. Specify a file name.

## Cell Coloring

Use the [Cell Coloring](#) page of the Timing Debug Preferences form to choose colors for specific cells in the delay bar.



When you assign colors, this same colors will be restored when you start a new session.

In the *Cell Name Selection Elements* field for each color, you can choose whether you are providing one of the following:

- Cell name
- Instance
- Procedure that you have defined

The procedure is invoked with the full instance name as the argument. You must source the file containing the procedure before you use this feature.

For example:

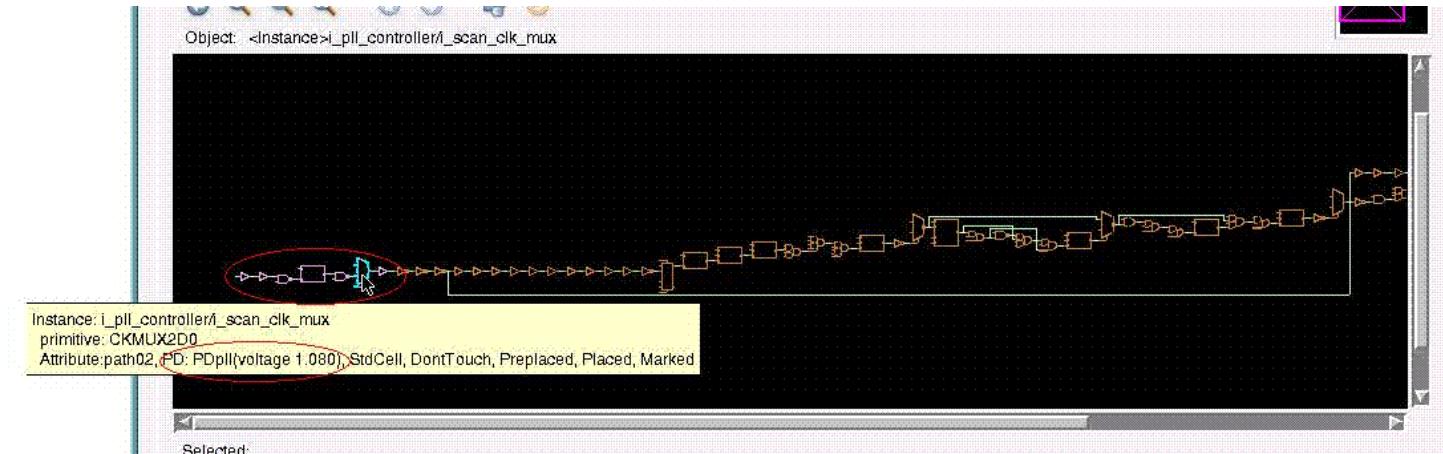
```
# Colors when the instance name contains "core/block1"
proc belongs_to_block1 {inst_name} {
    if [regexp {core/block1} $inst_name] {
        return 1
    } else {
        return 0
    }
}
```

## Viewing Schematics

The Critical Path Schematic Viewer displays the gate-level schematic view of the critical path that you select in the Debug Timing - Browser window. To display the Schematic Viewer, click on the schematics icon in the Timing Debug window. You can display additional paths in the Schematic

Viewer by using the middle mouse button to drag the path from path list to Schematic Viewer.

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:

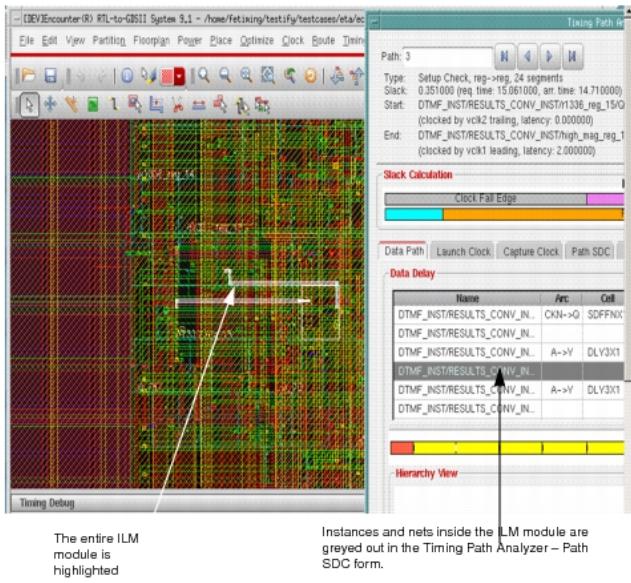
- View the Gate-level design elements.
- Select an element in the schematic.
  - Click on an object in the schematic to select and highlight it. When you move the cursor to an object, the object type and name of the object appear in the information box.
- Scroll over an object to display the object type and name of the object in the *Object* field.
- Cross-probe between the schematic and design display area.
  - Drag and drop an object from the schematic to the design display area to zoom in and highlight the object.
- Cross-probe between the Schematics window and *Path List* field.
  - Select a path and left-click on the Schematics button above the Path List Table. (This is the button at the far-right side, just above the table).
  - To show multiple paths, select another path, and drag and drop it to the Schematics window.
- Use the menu options provided in the Schematic Viewer. To access the menu options, you can either click on the menu bar or right-click on an object in the schematic. You can use the menu options to perform the following tasks:
  - Manipulate schematic views of fan-in and fan-out cones.
  - Trace connectivity between drivers, objects, and loads.
  - Move between different levels of instance views.

## Running Timing Debug with Interface Logic Models

You can use the timing debug feature with designs containing Interface Logic Models (ILMs).

- You must flatten ILMs before creating global timing debug reports. Use the [flattenIIm](#) command first. After loading the report, you can run timing debug in flattened or unflattened mode. The schematic feature is disabled in unflattened mode.
- The Timing Path Analyzer - Path SDC form displays ILM SDCs rather than the original SDCs.

- The software highlights the entire ILM module instead of the instances and nets inside the ILM. The instances and the nets inside the ILMs are greyed out in the Timing Path Analyzer - Path SDC form.



## Statistical Static Timing Analysis

---

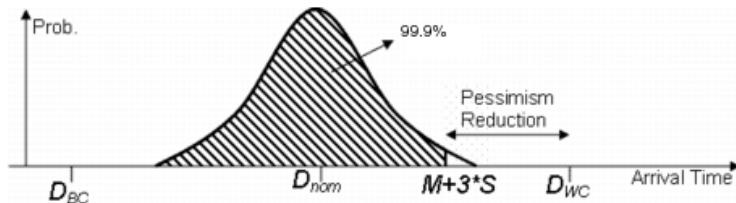
- SSTA Overview
- SSTA Inputs
- SSTA Flows
- SSTA Outputs
- SSTA Correlation With Monte-Carlo Analysis

### SSTA Overview

A major design challenge in 90nm and below designs are process variations. Process variations account for deviations in the semiconductor fabrication process. Process variations are due to variations in the manufacturing conditions such as temperature, pressure and dopant concentrations. To handle process variations, static timing analysis (STA) provides multi-corner analysis. However due to modelling of large number of process variation corners, STA methodology can be very complex and time consuming. Additionally STA can be very conservative in most cases and can be optimistic in some cases resulting in missed violations. To effectively handle process variation, you can now use Statistical Static Timing Analysis (SSTA).

SSTA represents the slack in terms of probability density function (PDF). PDF accounts for the variability of all process factors being modelled. This methodology targets a specific percentage yield for timing analysis and optimization.

SSTA removes the pessimism present in corner based STA. This helps in removing extra margins in the design and therefore improves design cycle time, and chip area. For example, consider the arrival time of signals on different chips. The arrival time on different chips is different because process variation delays are not constant. The following figure shows arrival time probability density function with different arrival times on x axis, and fractional number of chips on y axis.



In this figure,

$D_{nom}$  : Arrival time at nominal corner

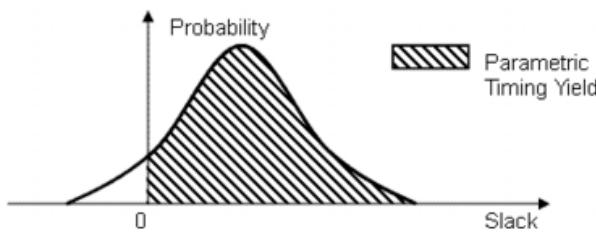
$D_{BC}$  : Arrival time at best case corner

$D_{WC}$  : Arrival times at worst corner

$M + 3\sigma$ : Worst arrival time for 99.9% yield.  $M$  is the mean value and  $\sigma$  is the standard deviation or sigma value.

In this example, none of the chips have the arrival time of  $D_{WC}$ . Therefore use of worst corner in STA will result in pessimistic delay analysis. SSTA analysis on the other hand considers complete distribution of arrival times and calculates worst arrival time for 99.9% yield. In this case,  $M+3*S$  delay is much smaller than worst corner arrival time  $D_{WC}$ .

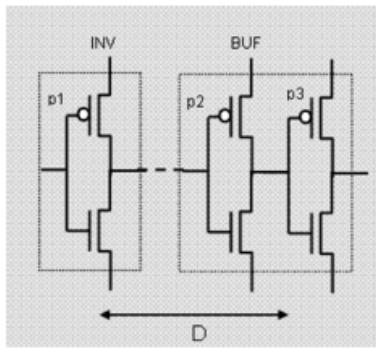
SSTA also provides data to trade-off between performance and yield. In STA, if a violation is reported at worst corner, it is actually violation on the worst chip in a lot. Since STA does not provide any information on the number of chips affected by the violation, you must fix this violation. This impacts cycle time and/or chip area. SSTA, on the other hand, provides the yield number (in terms of fraction of chips that can be affected by this violation) for a given frequency and vice versa. You can use this information to trade-off between yield and frequency. The following figure shows probability density function with slack value on X axis and fractional number of chips on the Y axis.



In this figure, parametric timing yield gives the percentage of chips with positive slack value.

The following figure shows process variations between different cell instances in a design.

**Figure 31-1 Variations between instances in a design**



In this figure,

$p_1$ ,  $p_2$  and  $p_3$  are process parameters for three different transistors

$D$  is Distance between the cell instances

SSTA accounts for the following process variations:

- Global or die-to-die variations

SSTA considers perfect correlation between all devices on the chip. Consider the variations between cell instances shown in Figure 1-1. The following equation shows the correlation between  $p_1$ ,  $p_2$  and  $p_3$  for global or die-to-die variations:

Correlation ( $p_1, p_2$ ) = Correlation ( $p_1, p_3$ ) = Correlation ( $p_2, p_3$ ) = 1

- Random variations

SSTA considers the process variation of all transistors on the chip to be uncorrelated to each other. Consider the variations between cell instances shown in Figure 1-1. The following equation shows the correlation between  $p_1, p_2$  and  $p_3$  for random variations:

Correlation ( $p_1, p_2$ ) = Correlation ( $p_1, p_3$ ) = Correlation ( $p_2, p_3$ ) = 0

- Spatial variations

SSTA considers the process variations inside the cell to be perfectly correlated. It also considers process variation correlation between transistors in different instances of the cells as function of distance between instances. Consider the variations between cell instances shown in Figure 1-1. The following equation shows the correlation between  $p_1, p_2$  and  $p_3$  for spatial variations:

Correlation ( $p_2, p_3$ ) = 1 Correlation ( $p_1, p_2$ ) = Correlation ( $p_1, p_3$ ) =  $f(D)$

Where,  $f(D)$  is a user defined function. The value of function decreases as  $D$  increases.

Therefore, the software models the process parameter as follows:

Where,

$P$  = Process parameter

$P_{nom}$  = Nominal process variation

$P_{global}$  = Global process variation

$P_{sys}$  = Systemic or spatial process variation

$P_{random}$  = Random process variation

## SSTA Inputs

SSTA requires specific input files containing sensitivities and distribution besides the inputs that you need to perform STA, such as a netlist and timing constraints.

The inputs required for SSTA are as follows:

- Libraries with sensitivities
- Statistical Parameter Distribution Format (SPDF) File
- Sensitivity-Based SPEF (S-SPEF) File
- Timing Constraints

The timing constraints supported for SSTA are the same as those supported for STA. For a list

of supported timing constraints, see the [Timing Constraints Commands](#) chapter in *EDI System Text Command Reference*.

- Netlist

## Libraries with sensitivities

SSTA requires S-ECSM libraries. These libraries provide sensitivities of delay, slew, and timing checks (such as setup and hold) to various cell process parameters. The libraries also contain sensitivities of voltage v/s time waveform. The S-ECSM libraries contain sensitivities for both global and random variations. Spatial variations are modelled as global parameters at the library-level.

You can use Encounter Library Characterizer (ELC) or other third party library characterizers to generate S-ECSM libraries. To generate S-ECSM libraries, the following inputs are provided to ELC:

- Spice models: Contains process parameters
- Spice circuits: Contains extracted spice netlist of library cells
- Configuration file: Contains variation of process parameters

For random variations, you can optionally combine the parameters in the library file. The following is a sample of S-ECSM file with sensitivity tables for global parameters, p1, p2, p3, and p4, and combined random parameter (ecsm\_random):

```
ecsm_timing_sensitivity(){  
    ecssm_parameter_type : p1 ;  
    ecssm_parameter_variation : 2.9 ;  
    fall_transition(delay_template_8x7) {....}  
    cell_fall(delay_template_8x7) {....}  
    rise_transition(delay_template_8x7) { ... }  
    cell_rise(delay_template_8x7) {....}  
}  
  
ecsm_timing_sensitivity() {  
    ecssm_parameter_type : p2;  
    ecssm_parameter_variation : 2 ;  
    ...  
}  
  
ecsm_timing_sensitivity() {  
    ecssm_parameter_type : p3;  
    ecssm_parameter_variation : 1.7 ;  
    ...  
}  
  
ecsm_timing_sensitivity() {  
    ecssm_parameter_type : p4;  
    ecssm_parameter_variation : 1.2 ;  
    ...  
}  
  
ecsm_timing_sensitivity() {  
    ecssm_parameter_type : ecssm_random;
```

```
ecsm_parameter_variation : 1 ;  
...  
}
```

For more information on library characterization, see *Encounter Library Characterization User Guide*.

## Statistical Parameter Distribution Format (SPDF) File

An SPDF file contains statistics of variation of process parameters. You can specify global, random and spatial variations in this file. You can create this file using the data provided by the foundry.

A sample of SPDF file is as follows:

```
{Standard_Parameter_Format  
  {SPDF_Version "1.0"}  
  {PROGRAM "SSTA"}  
  {PROGRAM_VERSION "1.0"}  
  {Units  
    {VOLTAGESCALE 100 mV}  
    {LENGTHSCALE nm}  
    {TIMESCALE 1 ns}  
    {Unit_Distance 1 um}  
  }  
  {PARAMETER  
    {A1, Cell,  
      {D2D Data}  
      {WID Data}  
      {Random Data}  
    }  
  }  
  {PARAMETER  
    {M1T, Interconnect,  
      {D2D Data}  
      {WID Data}  
    }  
  }  
}
```

The following sections provide information on the syntax used to specify various variations in the SPDF file.

### Specifying Global or Die-to-Die Variations in SPDF File

To specify the global or die-to-die variations, use the following syntax in the SPDF file:

```
{PARAMETER {Parameter ,Cell/Interconnect  
  {D2D  
    {Gaussian, (Mean ,Sigma )}  
  }  
}
```

### Specifying Random Variations in SPDF File

To specify the random variations, use the following syntax in the SPDF file:

```
{PARAMETER {Parameter , Cell/Interconnect
    {Random
        {Gaussian, (Mean , Sigma )}
    }
}
```

## Specifying Spatial Variations in SPDF File

To specify the spatial or with-in-die (WID) variation, use the following syntax in the SPDF file:

```
{PARAMETER {Parameter, Cell
    {WID
        {Spatial_Function}
        {Gaussian}
        {Mean_Polynomial, (  $\mu_0$  , q1, q2, q3, q4)}
        {Sigma_Polynomial, (  $\sigma_0$  , r1, r2, r3, r4)}
        {Exponential_Spatial_Correlation, FCD, (k0, k1)}
    }
}}
```

Where,

$\mu_0$  : mean value

$\sigma_0$  : sigma or standard deviation.

$k_0$  ,  $k_1$  : Constant factors that indicates the inverse relationship between correlation and separation of two devices.

FCD (Full-Correlated Distance): Size of grid for spatial correlation variation.

The values  $q_n$  ,  $r_n$  ,  $k_0$  and  $k_1$  are supplied by the foundry.

The mean and variation of process parameters can be modeled as quadratic functions in terms of distance.

$$\mu_n = \mu_0 + q_{1n}x + q_{2n}x^2 + q_{3n}y + q_{4n}y^2$$

$$\sigma_n = \sigma_0 + r_{1n}x + r_{2n}x^2 + r_{3n}y + r_{4n}y^2$$

Therefore if mean and sigma values are location independent, you can set  $q_n$  and  $r_n$  to 0. However the values for  $k_0$  and  $k_1$  should always be specified.

Spatial Correlation between two devices is modeled as an exponential function of distance.

## Sensitivity-Based SPEF (S-SPEF) File

Sensitivity-based extraction captures variations in interconnects. In sensitivity based extraction, in addition to nominal values of resistances and capacitances, the software also extracts the sensitivities of R and C to different interconnect process parameters.

Sensitivity-based extraction uses the variations in the following parameters:

- Metal width
- Metal thickness
- Dielectric layer thickness
- Resistivity
- Via
- Dielectric constant

The process of generating S-SPEF file is as follows:

1. Prepare a sensitivity techfile using TechGen. For more information, see *QRC TechGen Reference Manual*.
2. Set the variable to set sensitivity-based extraction in QRC command file and run extraction. For more information on running standalone QRC, see *QRC Extraction User Manual*.

## Loading the S-SPEF File

To load the S-SPEF file for SSTA, complete the following step:

```
spefIn -sspef SSPEF_file
```

**Note:** When you use the S-SPEF file, set the `-useNetSens` parameter in the `setDelayCalcMode` command for delay calculation to consider interconnect variations.

## SSTA Flows

You can perform block-based, path-based, or worstRC corner SSTA. By default the software runs block-based analysis.

- **Block-based SSTA**  
In block-based SSTA, the software analyzes the entire design statistically. The software uses the timing graph in leveled manner and propagates the arrival times from input ports to all end-points.
- **Path-based SSTA**  
In path-based analysis, the software first identifies the potential critical paths from STA and then path based statistical analysis is performed on the selected paths. Path-based SSTA uses path-based slew propagation, which makes path-based analysis more realistic (by removing pessimism in worst slew propagation). The reports generated in path-based SSTA are similar to those generated using STA.
- **WorstRC Corner SSTA**  
The worstRC corner mode can be used when the distribution of interconnect parameters is

unknown. In path-based worstRC corner mode, the software performs path-based SSTA on the most critical paths and reports the path-specific worst interconnect corners that are selected to provide worst case slack for the path. The reports generated in path based worstRC mode of SSTA report the interconnect parameter values that provide the worst case slack for the path.

## Running Block-Based SSTA

To run block-based SSTA, complete the following steps:

1. Set the analysis mode to statistical.
2. Load the design.
3. Read the SPEF file. Optionally read in the S-SPEF file to include the interconnect parasitic information.
4. Read the SPDF file. For more information on SPDF file, see [Statistical Parameter Distribution Format \(SPDF\) File](#).
5. Generate the timing report for top n endpoints.

### Example 31-1 SOC Command File for Running Block-Based SSTA

```
setAnalysisMode -timingEngine statistical
source init.globals
init_design
spefIn spefFile
read_spdf spdffile
report_timing -nworst n > rptfile
```

## Running Path-Based SSTA

In path-based analysis, the software first identifies the potential critical paths using STA and then runs statistical analysis on the selected paths. You can then generate path-based timing reports for your design.

To run path-based SSTA, complete the following steps:

1. Set the analysis mode to statistical.
2. Load the design.
3. Read the SPEF file.
4. Generate the timing report using the `-retime ssta` parameter in the [report\\_timing](#) command.

### Example 31-2 SOC Command File for Running Path-Based SSTA

```
setAnalysisMode -timingEngine statistical
source init.globals
init_design
```

```
spefIn spefFile
read_spdf spdfFile
report_timing -retiming ssta -max_paths num
```

## Running WorstRC Corner SSTA

When distribution of interconnect parameters are not known:

- Path based SSTA is applied on the top most critical paths (based on slack threshold).
- Path specific worst interconnect corners are reported.
- Interconnect parameter corners are selected to provide worst case slack.

Here's a sample path-based worst corner statistical timing analysis script:

```
setAnalysisMode -timingEngine statistical
source init.globals
init_design
spefIn -sspef test.spef
read_spdf test.spdf
setDelayCalMode -useNetSens true
report_timing -retiming ssta -worst_rc_corner
```

## SSTA Outputs

You can generate SSTA timing reports in block-based, path-based, and worst RC modes. The SSTA timing reports contain the following information in addition to the data present in the regular STA report:

- Sensitivities/PDF of Delay, Arrival Time, Required Time, Slack
- Yield for critical Paths
- Joint Yield (Design Yield)
- WorstRC corners (if interconnect process PDFs are not provided)

### SSTA Reporting Options

Standard report\_timing options are supported in Statistical analysis.

|                                                 |                                                                                   |
|-------------------------------------------------|-----------------------------------------------------------------------------------|
| <code>report_timing -max_paths num_paths</code> | Reports specified number of worst paths in the design regardless of the endpoint. |
|-------------------------------------------------|-----------------------------------------------------------------------------------|

|                                          |                                                                   |
|------------------------------------------|-------------------------------------------------------------------|
| <code>report_timing -nworst value</code> | Specifies the number of paths to be enumerated for each endpoint. |
|------------------------------------------|-------------------------------------------------------------------|

|                                              |  |
|----------------------------------------------|--|
| <code>report_timing -max_points value</code> |  |
|----------------------------------------------|--|

|                                                                                                                             |                                                                                                           |
|-----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
|                                                                                                                             | Reports the worst path to each endpoint upto the number specified by the <code>max_points</code> option.  |
| <code>report_timing -early/-late</code>                                                                                     | Generates the timing report for early paths (hold checks) or late paths (setup checks).                   |
| <code>report_timing -rise/-fall</code>                                                                                      | Reports the path with the specified edge on the endpoints.                                                |
| <code>report_timing -from/-through/-to pin_list</code>                                                                      | Reports paths starting from/passing through/ending at the pin(s) specified by the <code>pin_list</code> . |
| <code>report_timing -sort_slack_by {ssta_yield   sstaViolation   ssta_NSigma   ssta_path_criticality} -max_paths 100</code> | Sorts the SSTA reports in a specific order.                                                               |
| <code>report_timing -check_type check_type</code>                                                                           | Reports paths that end at a specific check type.                                                          |
| <code>report_timing -ssta_jpdf</code>                                                                                       | Reports the Joint Probability Density Function.                                                           |
| <code>report_timing -ssta_percentile float</code>                                                                           | Specifies Yield target for SSTA reports.                                                                  |

**Note:** All the above `report_timing` options are supported both in block based and path based and worstRC corner modes of SSTA.

## JPDF Report

A sample SSTA timing report with JPDF is as shown below.

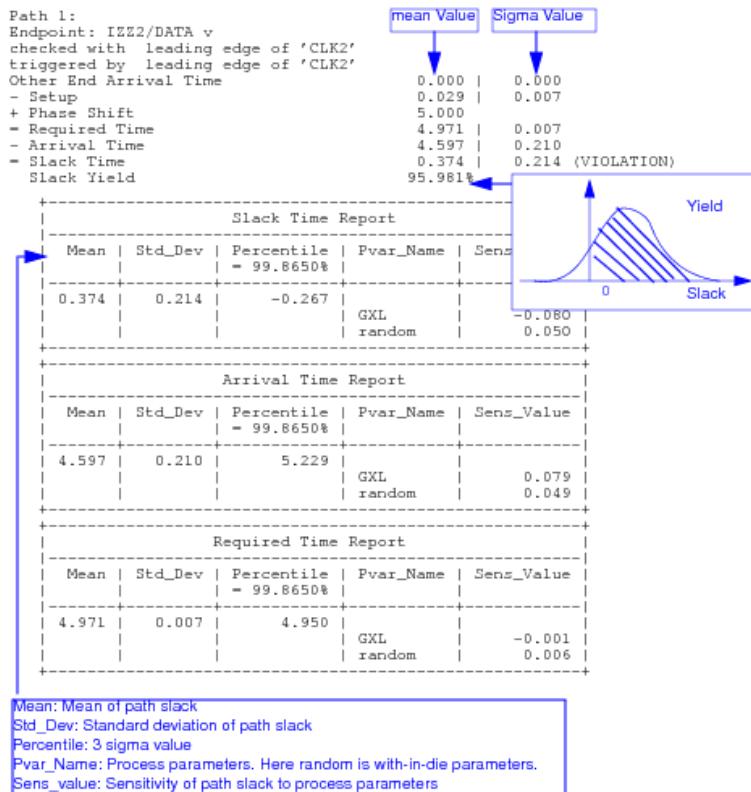
### Example 31-3 Joint PDF Report

```
SSTA Sigma Multiplier (N) (option -ssta_sigma_multiplier) = -3.000
+-----+
| Joint Distribution of Slacks |
+-----+
| Mean | Std_Dev | Mean+N*Std_Dev | Timing_Yield | Pvar_Name | Sens_Value |
+-----+-----+-----+-----+-----+
0.373	0.212	-0.263	96.078%		
+-----+
```

## Block-Based SSTA Report

Standard report\_timing command and options generate block based statistical timing reports in statistical analysis mode. A sample block based statistical timing analysis report is as shown below.

### Example 31-4 Block-based SSTA Report



## Path-Based SSTA Report

Path based SSTA reporting is enabled using report\_timing -retime ssta in statistical analysis mode. A sample path based statistical timing analysis report is as shown below.

### Path-Based SSTA Report

```
Path 1: VIOLATED Setup Check with Pin IZZ2/CLK
Endpoint: IZZ2/DATA (v) checked with leading edge of 'CLK2'
Beginpoint: IN1 (v) triggered by leading edge of 'CLK2'
Other End Arrival Time 0.000 | 0.000
- Setup 0.029 | 0.007
+ Phase Shift 5.000
= Required Time 4.971 | 0.007
- Arrival Time 4.526 | 0.217
= Slack Time 0.445 | 0.220
Clock Rise Edge 0.000
```

```

+ Input Delay 0.000
= Beginpoint Arrival Time 0.000

+-----+
| Instance | Cell | Arc | Retime | Retime | Retime Delay | Arrival |
| | | | Delay | Slew | Sensitivity | Time |
+-----+-----+-----+-----+-----+-----+
	IN1 v		0.100		0.000	
IA1	IVX1L		0.000	0.100		0.000
IA1	IVX1L	A v -> YB ^	0.353	0.463	XL 0.006	0.353
		random 0.014				
...						
...						
IA11	IVX1L		0.000	0.043	XL 0.000	4.436
IA11	IVX1L	A v -> YB ^	0.054	0.045	XL 0.001	4.489
		random 0.002				
IA12	IVX1L		0.000	0.045	XL 0.000	4.489
IA12	IVX1L	A ^ -> YB v	0.037	0.024	XL 0.001	4.526
		random 0.002				
		random 0.002				

| IZZ2 | DFX1L | | 0.000 | 0.024 | XL 0.000 | 4.526 |
+-----+
+-----+
| Slack Time Report |
+-----+
| Mean | Std_Dev | Percentile | Pvar_Name | Sens_Value |
|   | = 99.8650% | |
+-----+-----+-----+-----+
0.445	0.220	-0.216	
	XL	-0.080	
	random	0.069	
+-----+			
+-----+			
Arrival Time Report			
+-----+			
Mean	Std_Dev	Percentile	Pvar_Name
	= 99.8650%		
+-----+-----+-----+-----+			
4.526	0.217	5.177	
	XL	0.079	
	random	0.069	
+-----+			
+-----+			
Required Time Report			
+-----+			
Mean	Std_Dev	Percentile	Pvar_Name
	= 99.8650%		
+-----+-----+-----+-----+			
4.971	0.007	4.950	
	XL	-0.001	
	random	0.006	
+-----+

```

```
Path 2: VIOLATED Setup Check with Pin IZZ2/CLK
Endpoint: IZZ2/DATA (v) checked with leading edge of 'CLK2'
Beginpoint: IN4 (v) triggered by leading edge of 'CLK2'
Other End Arrival Time 0.000 | 0.000
```

Path-based SSTA reporting can be formatted to report the following additional data for all the stages of a path.

- Mean value of delay of all stages of the path
  - Delay sigma of all stages of the path
  - Delay sensitivities of all stages of the path
  - Mean value of slews of all stages of the path
  - Slew sigma of all stages of the path
  - Slew sensitivities of all stages of the path
  - This additional information can be reported in the timing report by setting the following global before the `report_timing` command in the script
- ```
set_global report_timing_format { instance cell arc retime_delay
retime_delay_sigma retime_delay_sensitivity retime_slew
retime_slew_sigma retime_slew_sensitivity}
```

## **WorstRC SSTA Report**

When distribution of interconnect parameters is unknown, WorstRC corner SSTA reporting is enabled using `report_timing -retime ssta -worst_rc_corner` in statistical analysis mode. A sample worstRC corner statistical timing analysis report is as shown below.

### **WorstRC SSTA Report**

```

Path 1: MET Setup Check with Pin REG_7/CP
Endpoint: REG_7/D (v) checked with leading edge of 'CLK'
Beginpoint: REG_5/Q (^) triggered by leading edge of 'CLK'
Other End Arrival Time      0.096 | 0.000
- Setup                      0.149 | 0.000
+ Phase Shift                2.200
= Required Time              2.147 | 0.000
- Arrival Time               2.105 | 0.000
= Slack Time                0.042 | 0.000
  Clock Rise Edge            0.000
  + Clock Network Latency (Prop) 0.000
  = Beginpoint Arrival Time    0.000
-----
  Instance          Arc        Cell       Retime Delay   Arrival Time   Required
                                                               Time
-----
          CLK ^           ND16      0.000      0.000      0.119
CK_L1_IO          I ^ -> ZN v  ND16      0.009      0.000      0.119
CK_L1_IO
...
...
REG_7
FF0             0.000      2.105      2.225
-----
Interconnect Worst Delay Corner:
M1_M0_T      9.500
M1_M0_W      8.300
M2_M1_T      3.000
M2_M1_W      5.200
M3_M2_T      6.000
...
...
-----
Slack Time Report
-----
Mean  Std_Dev  Percentile  Pvar_Name  Sens_Value
= 99.8650%
-----
0.042  0.000      0.042
-----
Arrival Time Report
-----
Mean  Std_Dev  Percentile  Pvar_Name  Sens_Value
= 99.8650%
-----
2.105  0.000      2.105
-----
Required Time Report
-----
Mean  Std_Dev  Percentile  Pvar_Name  Sens_Value
= 99.8650%
-----
2.147  0.000      2.147
-----
```

## SSTA Correlation With Monte-Carlo Analysis

To ensure the accuracy of your results, you can correlate the SSTA results with the Monte-Carlo

Analysis results. To perform Monte-Carlo analysis, complete the following steps:

1. Generate N sample of process parameters using their distributions.
2. Run Spice for each set of parameters.
3. Calculate delay histogram from N-samples.

Monte-Carlo analysis can be performed in a circuit simulator (e.g. spice, spectre), where each delay is calculated using spice simulation.

---

# Extracting Timing Models

---

- ETM Overview
- ETM Inputs
- Guidelines for Generating ETMs
- ETM Generation Flow
- ETM Outputs

## ETM Overview

Encounter® Digital Implementation System (EDI System) provides a mechanism to generate extracted timing models (ETM) in a hierarchical design flow. An ETM is the timing context derived for a digital circuit, which can be used in static timing analysis (STA).

Use of ETM in STA provides the following advantages:

- Reduces the memory requirements and improves the run time.
- Provides a mechanism to hide the proprietary implementation details of the block from a third party.
- Enables you to share the extracted timing models for blocks with other designers working on different parts of the design at the same time.
- Provides faster convergence of timing results for large design databases by providing the actual timing context of the lower-level module without requiring the software to analyze the complete logic of the module with each run.

During ETM generation, the software does not use boundary conditions, such as input transitions, output loads, input delays, output delays or clock periods. Therefore, you do not need to extract the models again if the boundary conditions change at a later stage in development. The software does consider the operating conditions, wire load models, annotated delays or loads, and RC data on internal nets defined for the design during ETM generation. Therefore, you do need to extract the models again if any of these elements change during a design phase.

An accurate ETM preserves the worst-case behavior of the original circuit. You can use ETM to

reproduce any timing violation that occurs in the original circuit. An ETM accurately represents critical paths that change with respect to a change in input slew value. In addition, an ETM preserves the self-loop timing checks such as minimum period and minimum pulse width checks.

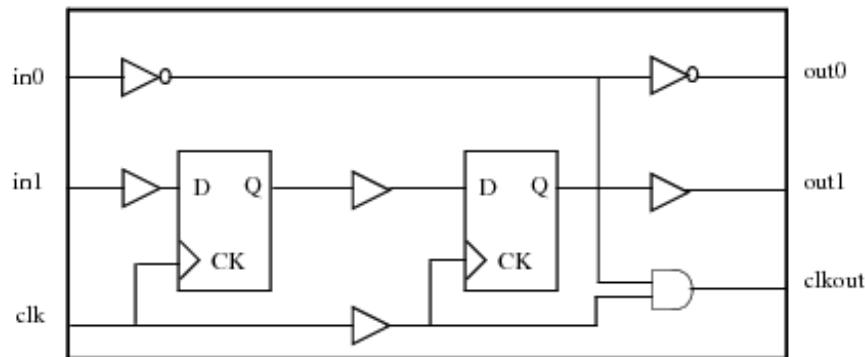
An ETM provides a timing representation by creating timing arcs for the following interface paths:

- Input to the register
- Input to output
- Register to output

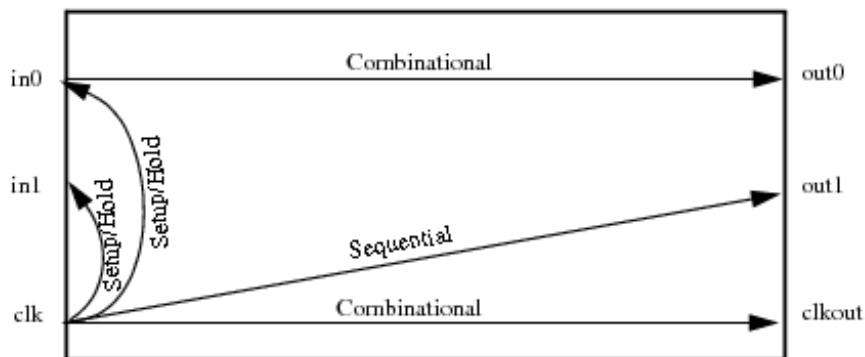
**Note:** Register to register paths are ignored because they do not affect the interface path's timing.

For example, consider the digital circuit shown in Figure 32-1.

**Figure 32-1 Gate-Level Netlist Representation of a Digital Circuit**



The ETM representation of gate-level netlist shown in the figure above is as follows:



## Using ETMs in Different Timing Analysis Modes

Use of ETM models for setup or hold analysis depends on the mode of timing analysis - Single, On-chip variation (OCV), or best-case worst-case (BcWc).

In single mode, the software uses late and early paths from a single corner for setup and hold analysis. The timing model that you generate in single mode has late and early paths from a single corner. Therefore, the timing model that you generate for setup is the same as that generated for hold analysis, and you can use the same model for setup or hold analysis.

In OCV mode, the software uses late paths from max corner and early paths from min corner for setup and hold analysis to generate the timing model. Therefore, the timing model that you generate for setup is the same as that generated for hold analysis, and you can use the same model for setup or hold analysis.

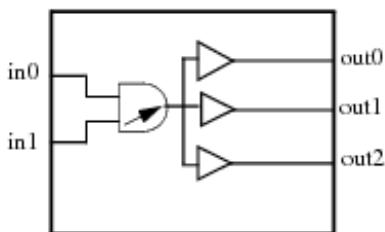
In BcWc mode, the software uses late and early paths from max corner for setup analysis and late and early paths from min corner for hold analysis. The model generated during setup analysis can not be used for hold analysis and vice versa because the software performs setup and hold analysis in different process corners.

## Limitation of Timing Models

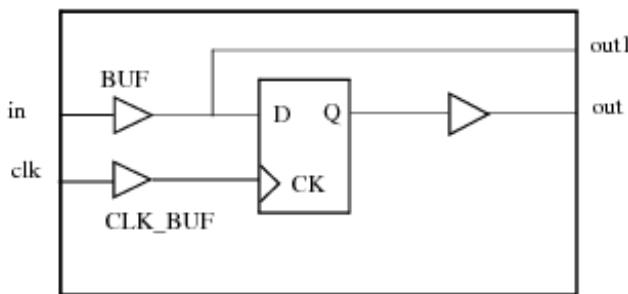
Timing models have the following limitations:

- An ETM does not preserve logical behavior of the pins. Therefore they do not contain any conditional arcs.
- Slew propagation from side paths may be inaccurate. The software assumes a single value for the input slew of ports during the extraction process. The software then propagates the input slews and calculates delays and uses these delays for the associated timing arc in the extracted model

For example, consider the circuit in the figure below. The software assumes that the input slews at both inputs of the AND-gate have the same value. Therefore the software propagates the worst slew value from B->Y transition. However, at the top-level, input slews can be different. Therefore the usage of worst case value from B->Y transition is not correct.



- The check value for the arc is extracted based on the current load. Therefore the output load dependent check paths do not have the load dependency in the extracted model. For example, consider the circuit in the figure below. In this circuit, the slew at point D depends on the loading at port out. Therefore the check arc value depends on the out loading. Extracted model do not support the 3D arcs for setup checks. Therefore in this case, the check arc value is calculated using the current loading at out.



- The software does not preserve the three state enable or disable arcs in an ETM. The software evaluates the three state enable or disable expressions and three state arcs with transitions to and from the high impedance state Z ( $0 \rightarrow Z$ ,  $1 \rightarrow Z$ ,  $Z \rightarrow 0$ , or  $Z \rightarrow 1$ ) are transformed to combinational arcs with transitions  $0 \rightarrow 1$ ,  $1 \rightarrow 0$ ,  $1 \rightarrow 0$ , or  $0 \rightarrow 1$ , respectively.

## ETM Inputs

Generation of ETM in Encounter requires the same inputs that you need to perform STA. To generate ETM, load the following data in Encounter

- Design netlist
- Timing libraries
- Timing constraints
- RC data of the nets

## Guidelines for Generating ETMs

You can use several parameters in the [do\\_extract\\_model](#) command and a timing global to define the characteristics of the ETM.

- Use the `-input_slew`, `-clock_slew` and `-output_load` parameters to specify slews at inputs and load at the output pins. If you do not specify these parameters, the software determines the characterization points from the interface elements of the design.

The software characterizes all the check arcs for the slew points as follows

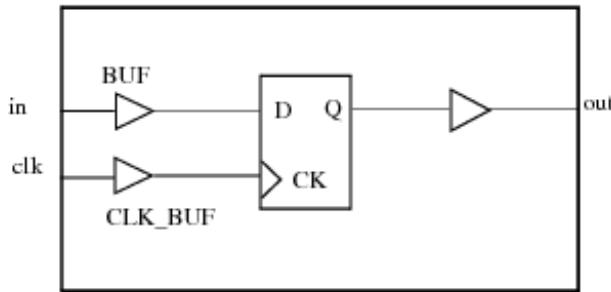
- Uses the reference slew points from the slew index of the first element after the port to which related pin is connected.
- Uses the signal slew points from the slew index of the first element after the port to which constrained pin is connected.

The software characterizes all the sequential arcs as follows:

- Uses the input slew from the slew index of the first element just after the clock port.
- Uses the output load from the load index of the last element just before the output port.

For example, Figure 32-2 shows a circuit used for generating ETM.

**Figure 32-2 Circuit for model extraction**



For this circuit, the library file has the characterization points as follows:

```
Cell (BUF)
{
    ...
    timing ()

    index_1 ("0.0500, 1.4000, 4.5000");
    index_2 ("1.0500, 6.5000, 10.0000");
    ...
}

Cell (CLK_BUF)
{
```

```
....  
timing ()  
index_1 ("1.0500, 2.4000, 3.5000");  
index_2 ("0.0500, 4.5000, 5.0000");  
....  
}
```

If you extract the model, without specifying the `-input_slew`, `-clock_slew` and `-output_load` parameters, the generated library file has the following characterization point for check arc between `clk` and `in`.

```
index_1 ("0 0.0500, 1.4000, 4.5000");  
index_1 ("0 1.0500, 2.4000, 3.5000");
```

Similarly the library file has the following characterization points for sequential arcs between `clk` and `out`:

```
index_1 ("0 1.0500, 2.4000, 3.5000");  
  
index_2 ("0 1.0500, 6.5000, 10.0000");
```

- Use the `-resolution` and `-tolerance` parameters to set the run time and accuracy of model extraction. Specifying greater resolution and tolerance results in less accurate models but improves the run time. The higher the tolerance, the faster is the extraction time. For example, you specify three characterization points for input slews and three characterization points for load values. In this case, the software uses a 3x3 table of slew and load indexes in the library file. However, if you also specify a range of resolution and tolerance, the software uses these values to simplify the table of slew and load indexes. If the middle slew and load values can be interpolated from the boundary values within the range of tolerance and resolution specified, then the software simplifies the 3x3 table to a 2x2 table. Therefore the software characterizes lesser number of slew and load values, thereby improving the run time of model extraction at the cost of accuracy.
- Use the [timing\\_extract\\_model\\_slew\\_propagation\\_mode](#) global to specify the type of slew propagation to use for generating ETM. You can specify worst slew propagation or path-based slew propagation.

In worst slew propagation mode, the software propagates the worst slew of all the incoming arcs at a converging point for extracting the arcs. This is the default mode.

In path based slew propagation mode, the software propagates the actual slew for the path elements for extracting the arcs. This is the recommended mode.

## ETM Generation Flow

Encounter supports two types of model extraction, blackbox and greybox models.

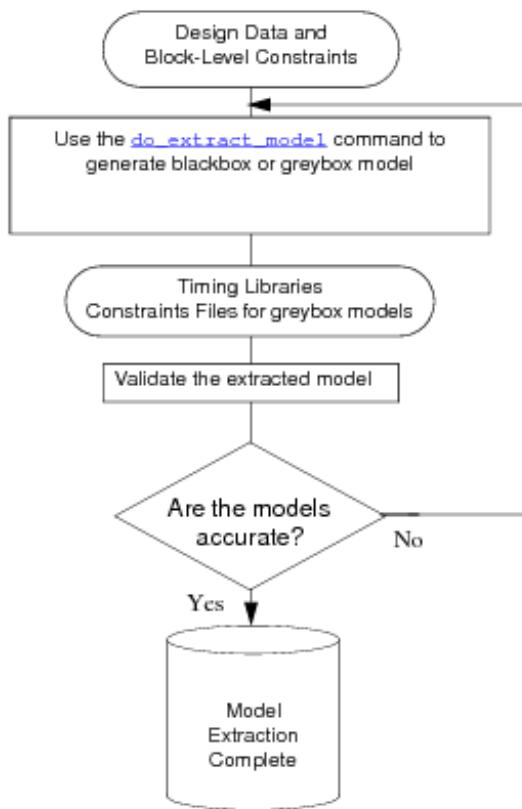
Both blackbox and greybox models preserve the worst-case behavior of the original circuit. The grey box models in addition retain internal pins in the extracted model. This enables better handling of constraints and latches.

Graybox models provide visibility into the internal pins in the circuit. Creating internal pins allows for accurate modeling of latch timing borrowing behavior and path exception handling. For greybox models, the software also generates a constraint file along with the extracted model. This constraints file is loaded incrementally when you use the extracted models at the top-level.

Graybox models have the following advantages:

- Supports arbitrary assertions.
- Guarantees that the model size is smaller than original netlist.
- The graybox model is clock-context independent as it does not adjust any multicycle path in the model as well it preserve the latch structure. That is, a greybox model is valid for clock waveforms that are different from the ones used to build the model.
- In greybox modeling, you do not need to re-extract the model if the path exceptions change. You can modify the generated constraint file instead. The greybox flow proves useful in the early stage of design where path exceptions are constantly changing.

The following figure shows the flow for generating ETMs.



The following procedure shows how to perform model extraction for black-box models:

1. Load the design data including the timing library on which you want to perform model extraction.
2. Specify the type of slew propagation to use for generating ETM. The worst slew propagation mode is recommended:

```
set_global timing_extract_model_slew_propagation_mode worst_slew
```

1. Perform model extraction:

```
do_extract_model extracted.lib \
-lib_name extracted_model \
-cell_name extracted_cell \
-tolerance 0.0 \
-verilog_shell_file top.v \
-verilog_shell_module test_top
```

```
write_model_timing -type arc netlist.rpt
```

The above steps will generate the following outputs:

- Extracted timing library (`extracted.lib`)
- Verilog wrapper (`top.v`), which will be used to instantiate the extracted model
- Interface timing characteristics of the original design (`netlist.rpt`)

## Validating the Generated Model

You can validate the timing models for their accuracy and coverage. An accurate ETM preserves the worst-case behavior of the original circuit. Encounter provides automated mechanism for validating the extracted models.

The following commands are used for model validation:

- [write\\_model\\_timing](#)

Writes the interface timing characteristics of the design in the specified timing model report file.

The report file contains the following sections:

- Worst-Case Arc/Slack: Contains details about the extracted arcs and the slack or delay across them depending on the argument specified with the `-type` option.
- Transition Time: Contains information about transition time at ports.
- Capacitance: Contains the capacitance values for the ports.
- Design Rules: Contains the design rules applied to the ports.

- [compare\\_model\\_timing](#)

Compares two reports that contain interface timing characteristics generated by the `write_model_timing` command.

This command compares the interface timing characteristics report generated by using the `write_model_timing` command on the original netlist with the report generated by using the `write_model_timing` command on the model extracted by the `do_extract_model` command for the same design.

To validate the extracted model, complete the following steps:

1. Ensure that the interface timing characteristics report file was generated by using the original gate-level netlist and constraints using the `write_model_timing` command.

2. Load the output files generated during model extraction, which include:
  - Extracted timing model (`extracted.lib`)
  - Verilog wrapper (`top.v`)
  - Design constraints (needed for greybox models)  
**Note:** In case of greybox models, load the additional constraints file generated during model extraction by the `do_extract_model` command.
4. Use the `write_model_timing` command on the extracted model (timing library) to generate a model report.

```
write_model_timing -type arc model.rpt
```

1. Use the `compare_model_timing` command to compare reports generated during model extraction and the report generated in step 4:

```
compare_model_timing
-ref netlist.rpt \
-compare model.rpt \
-outFile diff.rpt \
-percent_tolerance 2 \
-absolute_tolerance 0.003
```

## Reducing the Size of GreyBox Models

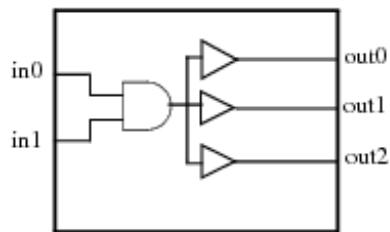
The software uses heuristic techniques to reduce the model size by further preserving some additional pins in a greybox model. To reduce the model size, use the `-gain` parameter in the [`do\_extract\_model`](#) command. The `-gain` parameters retains some existing internal pins in the circuit. These pins are called anchor points. Anchor points, if removed, might lead to an increase in model size. The software uses the numbers of input or output delay arcs to estimate the size of the model. The software assumes that all the delay arcs make equal contribution to the final model size. To select an anchor point, the software aims to minimize the number of delay arcs. Another criteria for selecting anchor point is the fact that the delay arcs are characterized with respect to a minimum number of input slew values. Therefore the software processes the incoming delay arc on each pin before removing them.

The gain value is calculated as follows:

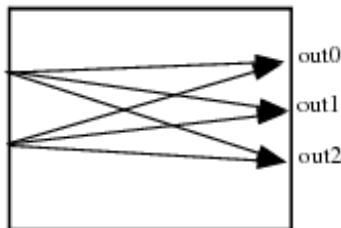
$$\text{Gain} = (\text{Number of incoming delay arcs} * \text{Number of outgoing delay arcs}) - (\text{Number of incoming delay arcs with anchor point} + \text{Number of outgoing delay arcs with anchor point})$$

For example, consider the circuit shown in Figure 32-3. If the software does not preserve an anchor point, the resulting 6 delay arcs are as shown in Figure 32-4. However if the software considers the AND gate output pin as an anchor point and creates an internal pin, the total number of arcs required is 5 as shown in Figure 32-5. There are 2 incoming arcs and 3 outgoing arcs at AND-gate output pin. Therefore the gain in this case is 1. Any pin that has gain equal or greater than value that you specify using the - gain parameter is considered by the software for creating an anchor point.

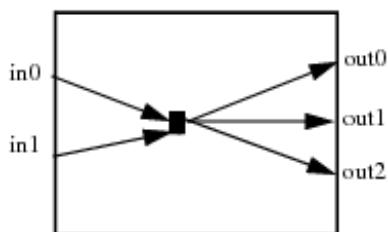
**Figure 32-3 Circuit for generating ETM**



**Figure 32-4 Extracted Model Without Internal Pin as Anchor Point**



**Figure 32-5 Extracted Model With One Anchor Point**



## ETM Outputs

ETM flow generates the following output:

- Timing Library File
- Timing Constraints Files

## Timing Library File

The timing model is generated as a .lib file. The following sections provide information on how these scenarios are handled in .lib file:

- Boundary Nets
- Internal Nets
- Timing Paths
- Minimum Pulse Width and Minimum Period
- Path Exceptions
- Constants
- Gating Checks
- Annotated Delays and Slews
- Design Rules
- Generated Clocks

### Boundary Nets

Boundary nets are the nets that are directly connected to the input or output ports of the block. Therefore the RC data for the boundary nets changes if the external context of the block changes at any point in the design phase. To generate context independent models, you can exclude the spef or dspef data for boundary nets from delay calculation. You can then stitch a separate file for spef or dspef data for boundary nets when you instantiate the ETM.

The software, by default, considers the RC data while extracting the timing model. ETS does not write out a spef file for boundary nets that can be used at top level. To use RC data with timing models, create the top level spef file for boundary nets or use the default behavior to consider the boundary nets RC data for model extraction.

### Internal Nets

Internal nets are the nets that drive or are driven by an internal instance pin of the block. Internal nets are context independent because they are only connected to internal instance of the block. Therefore the software uses the RC data defined for the internal nets for delay calculation and adds this information to the extracted paths. If you do not define any RC data for internal nets, then the software uses the wire load models to calculate the delay. If you use the set\_load or set\_resistance command

to annotate a load or resistance, then the annotated value overrides the annotated spef or the wire load model.

If you use the set\_annotated\_delay command or SDF annotation to annotate the delay on internal nets, the software uses the annotated delay instead of the calculated delay.

## Timing Paths

The following timing paths are included in an ETM:

- Input to Register Paths
- Register to Output Paths
- Register to Output Paths

**Note:** Register to register paths are ignored because they do not affect the interface path's timing.

### Input to Register Paths

Input to register paths are the paths from an input port to a register. In an ETM, the input to register paths are represented by equivalent setup or hold checks. These checks contain the calculated delay from the input port to the register, the setup or hold value of the library cell and the delay from a clock source to the clock pin of the register. The delay for the setup or hold checks is the function of the transition on the input port and the transition at the clock source

Setup check delay = delay (input to register) + delay (setup value of register) - delay (clock source to clock pin)

Hold check delay = delay (input to register) - delay (hold value of register) - delay (clock source to clock pin)

If multiple clocks reach a register, then the software extracts separate setup or hold arc for each clock source.

### Register to Output Paths

The register to output paths are the path from a register to an output port. These paths are a combination of a trigger arc of starting register and the combinational delay from sink of trigger arc to the output port. Therefore the software extracts an equivalent trigger or sequential arc for register to output paths.

The delay for these arcs is a function of the slew at the clock source and the capacitance at the output port. The delay of the arc is calculated as follows:

Sequential arc delay = delay (clock source to clock pin of register) + delay (register clock pin to output port)

The software generates two arcs representing longest and shortest path because the generated ETM can be used for both max and min analysis. The software generates different types of arcs for different valid clock edges such as rising\_edge or falling\_edge.

### **Input to Output Paths**

The input to output paths are the paths from an input port to an output port. These are combinational paths. Therefore the software generates equivalent combinational arc for these paths.

The delay for these arcs is a function of the slew at the input port and the capacitance at the output port. The delay for the arc is calculated as follows:

Combinational arc delay = delay (delay of all elements in the path)

The software generates two combinational arcs representing longest and shortest path because the generated ETM can be used for both max and min analysis. In case a path does not exist for a particular transition (rise or fall), the software generates half unate arcs such as `combinational_rise` or `combinational_fall`. The timing sense for the arc depends on the function of worst (early or late) paths.

### **Minimum Pulse Width and Minimum Period**

During timing model extraction the software transfers the minimum pulse width and minimum period constraints defined at the clock pin of the registers to the clock source pins. There might be several types of registers in the fanout of a clock source. Therefore, while transferring the minimum pulse width to the clock source, the software uses the worst minimum pulse width constraint value present on the fanout registers.

The software considers the delay and slew propagation differences of the clock network for rise and fall transitions while extracting the minimum pulse width. This is needed because some networks might have a significant difference in the rise and fall delays. So not considering these differences will cause differences in the minimum pulse width violations.

### **Path Exceptions**

False paths can only be modelled in greybox models. Blackbox models do not support extraction of false paths. In a greybox model, the software extracts the pins with false path or multicycle path assertions as internal pins. The software generates a constraint file for all internal pins that are extracted. You can then use this constraints file with the generated model to perform analysis.

**Note:** The software ignores the set\_max\_delay or set\_min\_delay constraints during model extraction. For an output port, the software extracts worst delay paths between two ports.

## Constants

The software propagates the case analysis and the constants on the netlist while extracting the model. The software does not consider the conditional arcs or paths that were disabled due to constants. If the constant value changes, you need to re-generate the timing model.

## Gating Checks

Clock gating checks are modeled as setup and hold checks between a clock pin and its enabling signal pin. Depending on the type of clock gating situation, setup and hold checks are inferred as follows:

- A clock-gating setup check is inferred with respect to the edge of the clock signal that changes the state of the clock gate from a controlling state to a non-controlling state.
- A clock gating hold check is inferred with respect to the edge of the clock that changes from a non-controlling state to a controlling state.

### Simple Clock Gating with AND Gate

For a simple AND gate, the setup checks are done with respect to the rising edge of the clock signal, and hold checks are done with respect to the falling edge of the clock signal.

Figure 32-6 shows a simple clock gating with AND gate.

**Figure 32-6 Simple Clock Gating with AND Gate**



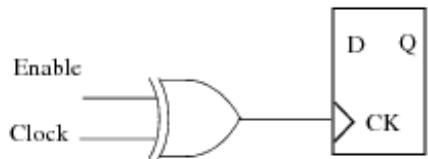
After extracting the clock gating check arc, the signal downstream the gate output is not propagated to the clock pin of the registers.

### No Clock Gating Logic

No setup or hold checks are inferred for clock gates with non-unate arcs, such as multiplexers and XOR gates because a clock signal cannot control the clock gate output.

Figure 32-7 shows an example where there is no clock gating logic:

### Figure 32-7 No Clock Gating Logic



### Annotated Delays and Slews

Model extraction uses the back annotated delay, slews, and the load information. This information is reflected in the extracted model.

- Annotated Delays

The annotated delays calculated using the SDF data or the `set_annotation_delay` command override the delay value of the arc, which was calculated using the library or RC data. The output slew of the arc, however, is calculated using the library or RC data. In case of incremental delays, the delta delay is added to the calculated arc delay.

- Annotated Slews

Slews annotated using the `set_annotation_transition` command override the slew value calculated using the library or RC data. As a result, the downstream delay and slews are calculated using the annotated slew.

- Annotated Load

The annotated load overrides the pin capacitance defined in the library, and is used for delay calculation.

### Design Rules

Model extraction uses the design rules defined in the timing library. The worst (lowest value for `max` design rules and highest value for `min` design rules) values among all the fanout pins is used for the input ports and the worst values among all the fanin pins is used for the output ports.

If the design rule limits are defined at the pin and library level, the design rule is chosen from the pin level because the pin-level information overrides the cell-level information, which in turn overrides the library-level information. However, the design rules defined in the SDC file have the highest priority and override the library values.

**Note:** The annotated delays are generated with a particular context and remain true for that context only. Therefore, annotating the delays or slews makes the model context dependent. To create a context independent model, it is recommended not to annotate the SDF.

## Generated Clocks

Generated clocks defined in a hierarchical block are preserved in the generated ETM model. The following rules are applied for modeling generated clocks in extracted timing models:

- The pin on which the generated clock is specified is preserved as an internal pin in the timing model.
- The name of this internal pin is changed to the generated clock name.
- A `generated_clock` construct is written out, which contains the definition of the `create_generated_clock` constraint that created the generated clock in the original netlist.

For example, if the original netlist contained the following generated clock statement:

```
create_generated_clock -name gclk -source [get_ports clk] -divide_by 2 [get_pins buf1/A]
```

During extraction, the pin `buf1/A` will be preserved as an internal pin in the library as `gclk`. The extracted model will be generated as follows:

```
generated_clock (gclk) {  
    master_pin      : clk;  
    divided_by     : 2;  
    clock_pin      : "gclk ";  
}  
  
pin (gclk ) {  
    clock: true ;  
    direction: internal ;  
    ....  
    ....
```

}

## Handling Multiple Clocks on Same Pin

When multiple generated clocks are defined on a pin in the original netlist, the pin is preserved as an internal pin in the extracted model. The name of the pin is assigned based on the generated clock that was last defined on that pin in the original netlist. In the extracted model, there will be one generated\_clock construct for each of the two clocks on that pin.

Consider the following example, which shows the constraints used in the original netlist to create generated clocks on the pin buf1/A:

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK -  
divide_by 2 [get_pins buf1/A]
```

```
create_generated_clock -name gclk2 -source [get_ports clk] -add -master_clock CLK -  
divide_by 2 [get_pins buf1/A]
```

During extraction, the pin buf1/A will be preserved as an internal pin in the library with a name gclk2. The extracted model will be generated as follows:

```
generated_clock (gclk1) {  
  
    master_pin      : clk;  
  
    divided_by     : 2;  
  
    clock_pin      : "gclk2";  
  
}  
  
generated_clock (gclk2) {  
  
    master_pin      : clk;  
  
    divided_by     : 2;  
  
    clock_pin      : "gclk2";  
  
}  
  
pin (gclk2) {  
  
    clock: true ;
```

```
direction: internal ;  
.....  
.....  
}
```

### Handling Generated Clock on Multiple Pins

When a generated clock is defined on multiple pins, all those pins are preserved as internal pins in the extracted model. Pin names of these internal pins are derived as follows:

- The generated clock name is used as a prefix
- All the pins on which the generated clock is specified are assigned a number on an incremental basis
- The generated clock name and the assigned number are merged using the underscore "\_" character. For example, if the clock name is clk, the corresponding clock pins will be named as clk\_1, clk\_2, and so on.

Consider the following example, which shows a netlist with clock definitions on multiple pins:

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK -  
divide_by 2 [get_pins {buf1/A buf2/A}]
```

During extraction, the pin buf1/A will be preserved as an internal pin in the library as gclk2. The extracted model will be generated as follows:

```
generated_clock (gclk1) {  
    master_pin      : clk;  
    divided_by     : 2;  
    clock_pin      : "gclk1 gclk_1 ";  
}  
  
pin (gclk1) {  
    clock: true;  
    direction: internal;
```

```
}

pin (gclk1_1) {
    clock: true;
    direction: internal;
    ...
}
```

## Timing Constraints Files

For greybox models, along with the .lib file, the software generates two constraint files containing a subset of original constraints. The constraint files contain constraints that are required for:

- Standalone model validation. This constraint file contains boundary environment such input transitions and output loads, along with other constraints.
- Top level stitching with the extracted timing model. This is the top-level constraints file.

The software extracts the following constraints in the two constraints files:

- set\_false\_path and set\_multicycle\_path constraints
- set\_disable\_timing and set\_case\_analysis
- create\_clock and create\_generated\_clock
- set\_input\_delay and set\_output\_delay
- Design Rules
- set\_load, set\_resistance and set\_annotated\_transition
- set\_annotated\_delay and set\_annotated\_check
- set\_input\_transition and set\_driving\_cell

### **set\_false\_path and set\_multicycle\_path constraints**

The arcs that you set using the set\_false\_path or set\_multicycle\_path constraints between ports

or clocks are preserved as is and written out in the extracted constraints file. The software does not change port names or clock names.

The arcs that you set using the `set_false_path` or `set_multicycle_path` constraints from, through or to internal pins are written out with modified pin names. The software prefixes the pin names with tcl variables. You can use the tcl variables to change the instance names.

## **set\_disable\_timing and set\_case\_analysis**

The `set_disable_timing` and `set_case_analysis` constraints are not written out in the constraints file because the software does not extract the arc that was disabled using the `set_disable_timing` or `set_case_analysis` constraints.

## **create\_clock and create\_generated\_clock**

The software extracts the `create_clock` constraints that you have defined for pins or ports. The software prefixes the internal pin names with tcl variables. You can use the tcl variables to change the instance names.

All the generated clocks are written into a library file as library defined generated clocks.

## **set\_input\_delay and set\_output\_delay**

The software writes out the `set_input_delay` and `set_output_delay` constraints that you define for ports as is. The constraints that you define for internal pins are not required for timing models and therefore are not written out.

The `set_input_delay` and `set_output_delay` constraints are not written out for top-level constraint file. This is because for an input port the input delay is the delay of path from top level register or port to input port. Similarly for an output port the delay is relative to top-level register or port.

## **Design Rules**

The software does not write out any design rule constraints for the timing model. This is because, design rules are defined in the .lib files.

## **set\_load, set\_resistance and set\_annotated\_transition**

The software uses the `set_load`, `set_resistance` and `set_annotated_transition` constraints defined on the internal pins for delay calculation during extraction.

The software writes out the `set_load`, `set_resistance` and `set_annotated_transition` constraints

defined on the port as is to the constraint file.

The `set_load`, `set_resistance` and `set_annotated_transition` constraints are not written out for top-level constraint file. This is because the software uses these values from the top-level environment.

### **`set_annotated_delay` and `set_annotated_check`**

The software uses the `set_annotated_delay` and `set_annotated_check` constraints for delay calculation during model extraction. Therefore the constraints are not written out to the constraint files.

### **`set_input_transition` and `set_driving_cell`**

The software writes the `set_input_transition` and `set_driving_cell` constraints as is to the constraint file.

The `set_input_transition` and `set_driving_cell` constraints are not written out for top-level constraint file.

# Optimizing Timing

---

- [Overview](#)
- [Before You Begin](#)
- [Results](#)
- [Interrupting Timing Optimization](#)
- [Performing Optimization Before Clock Tree Synthesis](#)
  - [Correcting Violations in Pre-CTS Mode for the First Time](#)
  - [Performing Rapid Timing Optimization for Design Prototyping](#)
  - [Using Additional Pre-CTS Timing Optimization Parameters](#)
  - [Performing Incremental Pre-CTS Optimization](#)
  - [Changing Default Settings in Pre-CTS Mode](#)
- [Performing Post-CTS Optimization](#)
  - [Correcting Violations in Post-CTS Mode](#)
  - [Using Additional Post-CTS Timing Optimization Parameters](#)
  - [Performing Incremental Post-CTS Optimization](#)
  - [Changing Default Settings in Post-CTS Mode](#)
- [Performing Postroute Optimization](#)
  - [About Postroute Optimization](#)
  - [Correcting Violations & Signal Integrity Issues using GigaOpt Technology in Postroute Mode](#)
  - [GigaOpt in PostRoute Setup Timing Flow](#)
  - [GigaOpt in PostRoute Hold Timing Flow](#)
  - [Changing Default Settings in Postroute Mode](#)
- [Optimizing Power During optDesign](#)
  - [Leakage Power Optimization](#)
  - [Dynamic Power Optimization](#)
- [Using Useful Skew](#)
  - [Using Useful Skew in Pre-CTS Mode](#)
  - [Using Useful Skew in Post-CTS 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](#)
  - [Optimizing Timing in OCV Mode Using the Sign-Off 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](#)

## 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 [Options - Set Mode- Mode Setup](#) form) to specify global timing optimization parameters. Use the [optDesign](#) super command (or the [Optimization](#) form) to optimize timing.

## Before You Begin

Before you optimize timing for the first time, complete the following steps:

1. Reserve placement space of more than five percent of the targeted final design utilization so that there is room to add buffers and remap the network to meet timing requirements.
2. If Assign statements exist in the Verilog® netlist, use one of the following procedures, which are equivalent, to enable optimization to work on Assign nets:
  - Specify `setDoAssign` on before loading the design data.
4. Specify default and detailed extraction scale factors by using the following commands:
  - `generateRCFactor`
  - `create_rc_corner`

For more information, see the "[RC Extraction Commands](#)" chapter in the *EDI System Text Command Reference*.
6. Use the following method to set input transitions for the high fanout nets for delay calculation:
  - Run the `delaycal_input_transition_delay` command.  
For more information, see [`delaycal\_input\_transition\_delay`](#) in the "Delay Calculation Commands" chapter of the *EDI System Text Command Reference*.
8. Create and load footprints. (Optional)

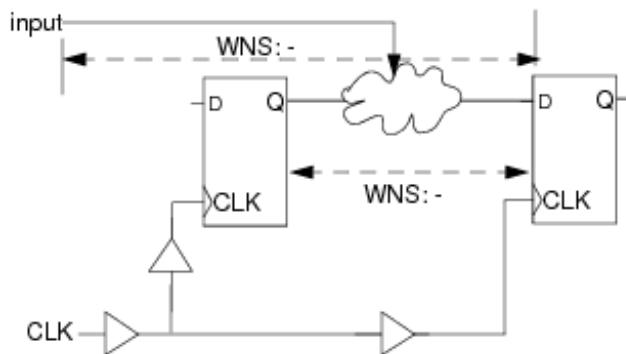
You are not required to specify footprints. For more information, see "Using the Footprintless Flow" .

## Results

After optimizing timing, the software appends the log file with the following information:

- Worst negative slack, total negative slack (TNS), and the number of failing (violating) paths. The software also reports hold violations if you specify the `-hold` parameter in post-CTS or postroute mode. It writes the values to the log file and writes reports to the working directory.  
**Note:** The overall TNS and number of failing paths of a design might not be equal to the total of the TNS and failing paths of the individual path groups. This is because the TNS and number of failing paths are based on the end-point of the path and are not path based.

For example, the following figure has a register with two paths, one from a primary input with a slack of -0.6ns and other from another register with a slack of -0.3ns.



In this case the overall TNS will be -0.6ns with 1 violating path (end point-based). But the individual reg2reg TNS will be -0.3ns with 1 violating path and in2reg with a TNS of -0.6ns with 1 violating path. Therefore, the sum total of individual path group TNS is not the same as overall TNS.

- Number of `max_tran`, `max_cap`, and `max_fanout` violations
- Utilization (density)

If you specify path groups, the software produces a slack file and `tarpt` report for them. If you do not specify path groups, the software produces the following four violation reports:

- Register-to-register
- Input-to-register
- Register-to-output
- Input-to-output

The reports contain information about the following violations for the top 50 critical paths:

- Setup violations
- Hold violations
- DRVs (maximum capacitance, maximum transition, and maximum fanout violations)

The software generates the reports and saves them in the file specified by `optDesign - outDir` (or in the `timingReports` directory if `- outDir` is not specified).

The filenames are

- `designName _preCTS_ pathGroup .tarpt`
- `designName _postCTS_ pathGroup .tarpt`

- *designName* \_postRoute\_ *pathGroup* .tarpt

The summary report has the following format:

```
-----  
optDesign Final Summary  
-----  
  
+-----+-----+-----+-----+-----+-----+  
| Setup mode | all | reg2reg | in2reg | reg2out | in2out | clkgate |  
+-----+-----+-----+-----+-----+-----+  
| WNS (ns):| -0.491 | -0.491 | N/A | 6.868 | N/A | N/A |  
| TNS (ns):|-866.856 |-866.856 | N/A | 0.000 | N/A | N/A |  
| Violating Paths:| 5273 | 5273 | N/A | 0 | N/A | N/A |  
| All Paths:| 69372 | 69343 | N/A | 29 | N/A | N/A |  
+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| | Real | | Total | |  
| DRVs +-----+-----+-----+  
| | Nr Nets(terms)| Worst Vio | Nr Nets (terms)|  
+-----+-----+-----+-----+  
| max_cap | 70(70) | -0.004 | 70(70) |  
| max_tran | 0(0) | .000 | 0(0) |  
| max_fanout | 0(0) | 0 | 0(0) |  
|  
+-----+-----+-----+-----+
```

Density: 53.455%

Routing Overflow: 0.00% H and 0.00% V

  
-----

For more information on timing reports, see [timeDesign](#), in the "Timing Analysis (Common Timing Engine) Commands" chapter of the *EDI System Text Command Reference*.

## Interrupting Timing Optimization

To stop timing optimization use the Control-C key combination. On pressing Control-C, the EDI System software exits at a legal location and outputs the database in a "reasonable" state; that is, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.

For more information, see [Interrupting the Software](#) in the "Getting Started" chapter.

## Performing Optimization Before Clock Tree Synthesis

- Correcting Violations in Pre-CTS Mode for the First Time
- Performing Rapid Timing Optimization for Design Prototyping
- Using Additional Pre-CTS Timing Optimization Parameters
- Performing Incremental Pre-CTS Optimization
- Changing Default Settings in Pre-CTS Mode

### Correcting Violations in Pre-CTS Mode for the First Time

- Before optimizing timing in pre-CTS mode, you must break all timing loops by disabling arcs in the constraint file. If you do not disable the arcs, the software cannot make a valid comparison of WNS between two different runs since it might not break the loops at the same point each time.

Use the following command:

```
set_disable_timing
```

For more information, see [set\\_disable\\_timing](#), in the "Timing Constraint Commands" chapter of the *EDI System Text Command Reference*.

- Use the following command to optimize timing:  
`optDesign -preCTS`
- To repair DRVs only, use the following command:  
`optDesign -preCTS -drv`

**Note:** By default, the `optDesign` does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

```
setOptMode -fixFanoutLoad true
```

For more information, see

- [optDesign](#), in the "Timing Optimization Commands" chapter of the *EDI System Text Command Reference*.

## Performing Rapid Timing Optimization for Design Prototyping

To optimize timing using low-effort mode for design prototyping, use the following commands:

```
setOptMode -effort low
optDesign -preCTS
```

In low-effort mode, optDesign resizes gates and performs global buffer insertion, but does not restructure the netlist or repair DRVs.

## Using Additional Pre-CTS Timing Optimization Parameters

You can use the following optDesign features separately or in combination.

- To run optimization with useful skew, use the following commands:  

```
setOptMode -usefulSkew true
optDesign -preCTS
```
- To run optimization on specific path groups, use the following commands:  

```
setAnalysisMode -honorClockDomains false
group_path -name groupName -from sourcePoint -to destinationPoint
setPathGroupOptions groupName -effortLevel high
optDesign -preCTS
```

In addition to custom path groups it is possible to use createBasicPathGroups to create the five most commonly used path groups:

- reg2reg
- in2reg
- reg2out
- in2out
- clkgate

For example, 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 Pre-CTS Optimization

Optimize timing incrementally to optimize setup times and area on critical paths. You can use the following features separately or together.

- To run incremental setup-only optimization, use the following command:

```
optDesign -preCTS -incr
```

- To run incremental optimization with useful skew, use the following commands:

```
setOptMode -usefulSkew true
```

```
optDesign -preCTS -incr
```

- To run incremental optimization on specific path groups, use the following commands:

```
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 "Using Additional Post-CTS Timing Optimization Parameters".

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 Pre-CTS Mode

You can change or add parameters for the following commands that optDesign runs automatically:

<a href="#">setAnalysisMode</a>	optDesign sets -clkSrcPath false and -clockPropagation forcedIdeal by default. You cannot override these values. You can add other parameters.
<a href="#">setExtractRCMode</a>	optDesign sets the extraction mode to default. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.

<a href="#"><u>setOptMode</u></a>	<p>optDesign sets the following parameters:</p> <ul style="list-style-type: none"><li>■ -drcMargin<ul style="list-style-type: none"><li>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. optDesign writes the margin value to the log file.</li></ul></li><li>■ -holdTargetSlack<ul style="list-style-type: none"><li>If you use <code>setOptMode -holdTargetSlack</code>, the value you specify is added to a dynamically calculated, internal margin. optDesign writes the hold target slack value to the log file.</li></ul></li><li>■ -setupTargetSlack<ul style="list-style-type: none"><li>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>. optDesign writes the setup target slack value to the log file.</li></ul></li></ul>
<a href="#"><u>setTrialRouteMode</u></a>	You can add parameters, but you cannot override the default settings. optDesign sets the <code>-handlePreroute true</code> parameter.

## Performing Post-CTS Optimization

- Correcting Violations in Post-CTS Mode
- Performing Incremental Post-CTS Optimization
- Changing Default Settings in Post-CTS Mode

### Correcting Violations in Post-CTS Mode

- To optimize timing after the clock tree is built, use the following commands:  
`optDesign -postCTS`

`optDesign` in postCTS fixes DRVs, reclaims area, and fixes setup violations.

**Note:** By default, the `optDesign` does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

```
setOptMode -fixFanoutLoad true
```

- To repair setup and hold violations, use the following commands:

```
optDesign -postCTS  
optDesign -postCTS -hold
```

- To repair design rule violations only, use the following command:

```
optDesign -postCTS -drv
```

- To repair hold violations only, use the following command:

```
optDesign -postCTS -hold
```

## **Skipping Path Groups or Clock Domains During Hold Fixing**

You can instruct the EDI System software to exclude path groups or clock domains from hold fixing by using the [setOptMode -ignorePathGroupsForHold](#) command.

This feature is useful, for example, when you want to fix only those hold violations that are in the core of the design and skip violations on I/O paths. This can be achieved by applying the following in the `clockDomains` support mode:

```
setOptMode -ignorePathGroupsForHold {in2reg reg2out in2out}
```

## **Using Additional Post-CTS Timing Optimization Parameters**

- To run optimization on specific path groups, use the following commands:

```
clearClockDomains  
setClockDomains -fromType sourcePoint -toType destinationPoint  
optDesign -postCTS
```

For a list of supported source and destination points, see "Correcting Violations in Pre-CTS Mode for the First Time".

**Note:** If you are using the CTE engine, you cannot use `setClockDomains` parameters other than `-fromType` and `-toType`.

For example, to run optimization on register-to-register paths, use the following commands:

```
clearClockDomains
setClockDomains -fromType register -toType register
optDesign -postCTS
```

- To take advantage of useful skew when optimizing timing in post-CTS mode, use the following commands:

```
setOptMode -usefulSkew true
optDesign -postCTS
```

If you have already performed detailed routing on the clock tree, the EDI System software performs global and detailed ECO routing automatically using the NanoRoute® router in post-CTS useful skew mode. If you do not want optDesign to do this, specify the -noECORoute parameter as follows:

```
setOptMode -usefulSkew true
optDesign -postCTS -noECORoute
```

If you specify -noECORoute before running optimization, the EDI System software performs trial routing to estimate clock delays.

- To run post-CTS optimization if your design has a clock mesh, use the following commands:

```
setOptMode -usefulSkew false
optDesign -postCTS
```

## Performing Incremental Post-CTS Optimization

- To run optimization on specific path groups, use the following commands:

```
clearClockDomains
setClockDomains -fromType sourcePoint -toType destinationPoint
optDesign -postCTS -incr
```

For a list of supported source and destination points, see "Using Additional Pre-CTS Timing Optimization Parameters".

**Note:** If you are using the CTE engine, you cannot use setClockDomains parameters other than -fromType and -toType.

For example, to run incremental optimization on register-to-register paths, use the following commands:

```
clearClockDomains  
setClockDomains -fromType register -toType register  
optDesign -postCTS -incr
```

- To optimize setup time incrementally and reduce area, use the following commands:  

```
setOptMode -reclaimArea true  
optDesign -postCTS -incr
```
- To take advantage of useful skew when optimizing timing in incremental post-CTS mode, use the following commands:  

```
setOptMode -usefulSkew true  
optDesign -postCTS -incr
```

If you have already performed detail routing on the clock tree, the software performs global and detailed ECO routing automatically using the NanoRoute router in post-CTS useful skew mode. If you do not want the software to do this, specify the `-noECORoute` parameter, as follows:

```
setOptMode -usefulSkew true  
optDesign -postCTS -noECORoute -incr
```

If you specify `-noECORoute`, `optDesign` performs trial routing to estimate clock delays.

- To run incremental post-CTS optimization if your design has a clock mesh, use the following commands:  

```
setOptMode -usefulSkew false  
optDesign -postCTS -incr
```

## Changing Default Settings in Post-CTS Mode

You can change or add parameters for the following commands and parameters that `-optDesign` runs automatically:

<a href="#"><u>setAnalysisMode</u></a>	optDesign sets -clockPropagation autoDetectClockTree and -clkSrcPath true by default. You cannot override these values. You can add other parameters.
<a href="#"><u>setExtractRCMode</u></a>	optDesign sets the extraction mode to default. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.
<a href="#"><u>setOptMode</u></a>	<p>optDesign sets the following parameters:</p> <ul style="list-style-type: none"> <li>■ -drcMargin If you use <code>setOptMode -drcMargin</code>, this value 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. optDesign writes the margin value to the log file.</li> <li>■ -holdTargetSlack If you use <code>setOptMode -holdTargetSlack</code>, this value is added to a dynamically calculated, internal margin. optDesign writes the hold target slack value to the log file.</li> <li>■ -setupTargetSlack If you use <code>setOptMode -setupTargetSlack</code>, this value is added to a dynamically calculated, internal margin. The default - setupTargetSlack value is <code>0</code>. optDesign writes the setup target slack value to the log file.</li> </ul> <p>You can override other parameters.</p>
<a href="#"><u>setTrialRouteMode</u></a>	You can add parameters, but never override the default settings. optDesign sets the <code>-handlePreRoute true</code> parameter.

## Performing Postroute Optimization

- About Postroute Optimization
- Correcting Violations in Postroute Mode

- Correcting Signal Integrity Violations

## About Postroute Optimization

In postroute mode, the EDI System software corrects setup violations and design rule violations unless you specify otherwise. It first operates on all path groups, then on register-to-register paths only. The software performs incremental RC and delay calculation, and runs the NanoRoute router to perform ECO routing. In case the final timing after ECO routing is degraded compared to what was expected before ECO routing, the tool will automatically call an incremental optimization to recover the Setup timing.

If filler cell definitions were provided during design import, `optDesign` removes or adds them as needed, following the information given by `setFillerMode`. For more information on this command, see [setFillerMode](#) in the "Placement Commands" chapter of the *EDI System Text Command Reference*.

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 pre-route optimization due to congestion-based detour routing
- Minor correlation issues between default and detailed RC extraction.
- Incremental delays due to parasitics coupling

**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 others. This cascading effect creates a situation where it becomes impossible to close on a final timing solution with no DRVs.

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

To take even more advantage of this feature, you can provide an external SPEF generated by a sign-off extraction tool for improved convergence. If you do, you must provide a full SPEF (reduced SPEF does not work) and one of the following conditions must be met:

- The SPEF must be generated with node locations using the starN format.  
or
- The resistance values in the LEF file must match those in the technology file used by signoff extraction to generate the SPEF, which enables the EDI System extraction engine to match the routes with the SPEF RC graph.

## **Correcting Violations & Signal Integrity Issues using GigaOpt Technology in Postroute Mode**

In the present release, GigaOpt Technology is the default for the postRoute flow, including setup/hold/power optimization. GigaOpt simplifies the postroute closure flow. Note that onChipVariation needs to be turned on & it is recommended to use CPPR as well:

```
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:

- 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 :

- External SPEF-flow
- External SDF-flow
- Filler cells deletion/insertion
- Smart ECO routing

GigaOpt hold fixing generates detailed diagnostic report for every remaining hold violated nets.

GigaOpt hold fixing performs:

- Buffer insertion along the route
- Wire and RC cutting
- Legal location searching

### **GigaOpt in PostRoute Use Model**

To optimize timing setup and hold with base and SI delay, use:

```
optDesign -postRoute
```

```
optDesign -postRoute -hold
```

To optimize timing setup and hold with base delay only, use:

```
setDelayCalMode -engine Aae -SIAware false
```

```
optDesign -postRoute
```

```
optDesign -postRoute -hold
```

## 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:

```
setOptMode -allEndPoints true
```

```
optDesign -postRoute
```

- To run GigaOpt for performing setup and leakage timing optimization on base and SI delay, use:

```
setOptMode -leakagePowerEffort high
```

```
optDesign -postRoute
```

- To run GigaOpt for performing setup with SI slews optimization on base and SI delay, use:

```
setSIMode -fixDRC true
```

```
optDesign -postRoute
```

- **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
```

- **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 the EDI System software to exclude path groups or clock domains from hold fixing. For more information, see [Skipping Path Groups or Clock Domains 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 same command as mentioned:

```
optDesign -postRoute
```

This will automatically flatten ILMs, optimize timing, 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), type the following:

```
spefIn -rc_corner cornerMax1 rcMax1.spef
spefIn -rc_corner cornerMax2 rcMax2.spef
spefIn -rc_corner cornerMin1 rcMin1.spef
spefIn -rc_corner cornerMin2 rcMin2.spef
optDesign -postRoute [-hold]
```

**Note:** You must provide SPEF information for each active `rc_corner` (setup and hold). If one corner does not have a SPEF, the tool will run RC extraction for each corner again.

- To run postroute 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:

```
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 and parameters that optDesign runs automatically:

<a href="#"><u>setAnalysisMode</u></a>	optDesign sets - clockPropagation autoDetectClockTree and - clkSrcPath true. You can override other parameters.
<a href="#"><u>setExtractRCMode</u></a>	optDesign sets the extraction mode to detail. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.
<a href="#"><u>setTrialRouteMode</u></a>	You can add parameters, but never override the default settings. optDesign sets the - handlePreroute true parameter.

## Optimizing Power During optDesign

During timing optimization, the tool is also able to reduce leakage power and dynamic power.

### Leakage Power Optimization

To activate leakage power optimization during timing optimization, run the following:

```
setOptMode -leakagePowerEffort none|low|high
```

- When effort is set to none, optDesign will not optimize the leakage power. This is the default option.
- When effort is set to low, optDesign will optimize leakage power only in the postroute stage. The leakage reduction will happen during postroute setup optimization and no high leakage cells will be inserted during hold fixing.
- When effort is set to high, optDesign will optimize leakage power in all stages. Leakage reduction will happen during each setup optimization and no high leakage cells will be inserted during hold fixing.

**Note:** To achieve the best leakage power results, load all the different Vth libraries.

## Dynamic Power Optimization

To activate dynamic power optimization during timing optimization, run the following:

```
setOptMode -DynamicPowerEffort none|low|high
```

- When effort is set to none, optDesign will not optimize dynamic power. This is the default option.
- When the effort is set to low, optDesign will only optimize dynamic power in the postroute setup optimization phase.
- When the effort is set to high, optDesign will optimize dynamic power in the entire setup optimization phases.  
**Note:** Cadence recommends that you provide an activity file. If you do not provide an activity file, EDI System automatically generates a default activity file based on a propagated toggle rate of 0.2 on each input port.

## Using Useful Skew

The useful skew feature in the EDI System software modifies the clock arrival time on sequential elements in order to improve the datapath timing between sequential elements.

The software provides two approaches to using useful skew, depending on whether you have run CTS:

- Pre-CTS mode  
Advances the clock signal for critical path start points. The start point must be a sequential element: No input paths are allowed.
- Post-CTS mode  
Delays the clock signal for critical path end points. The end point must be a sequential element: No output paths are allowed.

## Using Useful Skew in Pre-CTS Mode

To take advantage of useful skew during pre-CTS optimization, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -preCTS
```

The software determines the sequential instances whose clock signals can be advanced, then

generates the following two files:

- `latency_file.sdc`

This latency file models the proposed clock advancement for timing analysis.

- `scheduling_file.cts`

This file contains scheduling information for clock tree synthesis. You must specify this file when you specify the CTS constraints, for example:

```
specifyClockTree -clkfile scheduling_file.cts
specifyClockTree -clkfile original_constraints.cts
```

You can change the names of the latency and scheduling files by using the following commands:

```
setLatencyFile fileName
setSchedulingFile fileName
```

Use the following commands to report the names of the latency and schedule files:

- [getLatencyFile](#)
- [getSchedulingFile](#)

For more information, see "Timing Optimization Commands" in the *EDI System Text Command Reference*.

## Using Useful Skew in Post-CTS Mode

To take advantage of useful skew during post-CTS optimization, use the following commands:

```
setOptMode -usefulSkew true
optDesign -postCTS
```

In this case, the clock tree is already in place. The software determines the sequential instances whose clock signals can be delayed, and adds buffers or inverters to their clock nets accordingly. If the clock is already detail routed, these commands perform ECO routing on the clock tree after useful skew optimization.

## Controlling Useful Skew Optimization

You can control how the EDI System software employs useful skew, use the following command:

- [setUsefulSkewMode](#)

If you choose specific cells for clock tree synthesis, use `setUsefulSkewMode -useCells` to specify the cells to use for padding the clock nets. If you have no constraint on the type of cells allowed in the clock tree, you can omit this parameter, and the software selects the best combination of cells to achieve the required delay.

For example, if you want clock buffers or inverters only, specify the following command:

```
setUsefulSkewMode -useCells {...}
```

To advance or delay sequential elements more aggressively than it does by default, without degrading the worst negative slack, use the following `setUsefulSkewMode` parameter:

```
-maxSkew true
```

When you specify this parameter, the tool skews other registers as much as possible regardless of the worst slack on a particular register. This approach can help with difficult timing closure situations. In post-CTS mode, critical paths probably have been fully optimized, so further traditional optimization cannot dramatically improve timing.

To close timing, you might need to delay the endpoint clock pins more than the useful skew feature would do by default, by only padding the clock nets until the data path meets the target slack. To take advantage of this feature, use the following command:

```
setUsefulSkewMode -maxSkew true
```

To exclude boundary sequential cells in useful skew calculations, use the following command:

```
setUsefulSkewMode -noBoundary true
```

If you do not specify this parameter, the software takes boundary cells and ordinary sequential elements into account when calculating useful skew.

To use NanoRoute detailed routing to route nets that are added or changed during useful skew optimization, use the following command:

```
setUsefulSkewMode -ecoRoute true
```

To limit the amount of slack the EDI System software can borrow from neighboring flip-flops when performing useful skew operations, use the following command:

```
setUsefulSkewMode -maxAllowedDelay true
```

The EDI System delay calculation and RC extraction methods might differ from those of sign-off tools, so other setup violations might occur if the EDI System tool borrows too much slack. By having control over slack borrowing, you can prevent these setup violations. Limiting borrowed skew also limits the clock tree skew to avoid large hold violations. If you do not specify this parameter, the EDI System software automatically borrows the amount of slack needed (there is no maximum) to reduce setup violations.

To report the current `setUsefulSkewMode` settings, use the following command:

```
getUsefulSkewMode
```

For more information, see "Timing Optimization Commands" in the *EDI System Text Command Reference*.

## 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 EDI System, 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 EDI System 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

Optimize timing in on-chip variation (OCV) analysis mode to account for variations in process, voltage, and temperature (PVT) across the die. When it takes OCV into account, the software calculates early and late delays, and uses them to evaluate setup and hold timing checks. You introduce the delays into the analysis by specifying different min/max corner timing libraries and operating conditions. Early/late variation might also be present due to slew merging effects of multiple input gates in the clock path.

To enable the software to consider multiple libraries and operating conditions, you must specify a multi-mode/multi-corner (MMMC) environment. If the MMMC environment is not specified, and you try to run timing optimization in OCV mode, the `optDesign` command exits with an error message.

For more information on OCV and MMMC, see

- On-Chip Variation (OCV) Timing Analysis Mode
- 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:

```
setAnalysisMode -analysisType onChipVariation  
optDesign -postRoute [-hold]
```

For more information, see documentation for the following commands:

- [setAnalysisMode](#)
- [optDesign](#)

## Optimizing Timing in OCV Mode Using the Sign-Off Delay Calculator

After you specify the MMMC environment, use the following commands:

```
setDelayCalMode -engine signalStorm [-signoff]  
setAnalysisMode -analysisType onChipVariation  
optDesign -postRoute [-hold]
```

**i** The `setDelayCalMode - engine signalStorm` command is necessary for full support of the `rise/fall_pin_cap_range` construct.

For more information, see documentation for the following commands:

- [setDelayCalMode](#)
- [setAnalysisMode](#)
- [optDesign](#)

## Using Conformal Constraint Designer During Timing Optimization

The EDI System 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 EDI System-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 EDI System 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 have achieved the frequency target. Usually, this problem is due to tight timing constraints that are difficult to meet. By using CCD to verify whether the worst slack paths are valid, you may find that your critical paths are actually invalid and therefore need not be taken in account.

In this approach, the CCD tool is used post-processing only, and the EDI System timing closure flow is unchanged.

The following example shows the commands in the post-processing approach. Use these commands after running `timeDesign` or `optDesign` in setup mode.

```
deriveFalsePathCCD -outputDir . -outputFile trv.sdc
loadTimingCon -incr trv.sdc
```

The `deriveFalsePathCCD` command runs CCD in batch mode and generates a list of critical false paths. The `loadTimingCon` command reads the list into the EDI System software.

After running these commands, re-analyze timing by running `timeDesign`. The slack should be better because false paths have been removed from consideration.

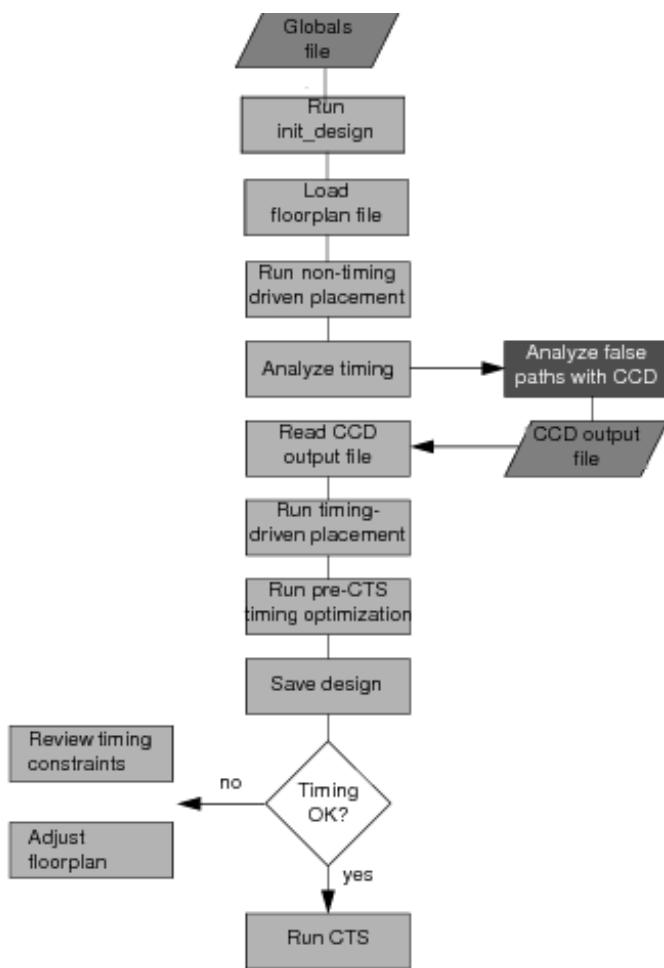
For more information, see documentation for the following commands:

- [deriveFalsePathCCD](#)
- [loadTimingCon](#)

## Integrated Approach

A second approach is to integrate CCD with the EDI System 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 *EDI System Text Command Reference*.

4. Read the CCD output file into the EDI System software.

For example,

```
loadTimingCon -incr trv.postnotdp.sdc
```

5. Place the design in timing-driven mode and run pre-CTS timing optimization. Save the design.

For example,

```
setPlaceMode -timingDriven true
placeDesign
timeDesign -preCTS -outDir timing_place_TD
setOptMode -effort high
optDesign -preCTS -outDir timing_optimized
saveDesign post_opt_prects.enc
```

If the design meets timing, you can go on to clock tree synthesis; if not, look at the timing constraints and the floorplan to make sure they are reasonable. If CCD reports too many false paths, the CPU run time for timing analysis and 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 EDI System 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 EDI System software creates an internal footprint structure based on cell functionality. This methodology is referred to as the footprintless flow, and has the following advantages over a flow that relies on footprint information from the libraries:

- The libraries do not need to contain footprints, and you do not need to specify a footprint file.
- The following commands are not necessary because the software detects the functionality for

inverters and buffers and decides whether a buffer is a delay cell, based on the cell's timing characteristic. The commands have no effect if specified in this flow.

- `setBufFootPrint`
- `setInvFootprint`
- `setDelayFootPrint`
- The software considers cells with the same functionality but different function syntax as equivalent and allows sizing between such cells.
- The software prints the list of usable and unusable ("don't use") buffers, inverters, and delay cells to the log file after reading in the libraries, for example:

```
Total number of combinational cells: 620
Total number of sequential cells: 247
Total number of tristate cells: 42
Total number of level shifter cells: 0
Total number of power gating cells: 0 Total number of isolation cells: 0
List of usable buffers: BFX1 BFX2 BFX3 BFX4
Total number of usable buffers: 4
List of unusable buffers: BFX20 BFX32
Total number of unusable buffers: 2
List of usable inverters: IVX1 IVX2 IVX3 IVX4
Total number of usable inverters: 4
List of unusable inverters:
Total number of unusable inverters: 0
List of identified usable delay cells: DLY2 DLY4 DLY8
Total number of identified usable delay cells: 3
List of identified unusable delay cells:
Total number of identified unusable delay cells: 0
```

To revert to the behavior in previous releases (that is, to rely on footprint information in the libraries), use the `loadFootPrint` command. As in those releases, you must specify buffers, inverters, and delay cell footprints according to what was loaded in the footprint file. For more information, see "[Using Cell Footprints](#)".

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 [setDontUse](#) in the "Timing Optimization Commands" chapter of the *EDI System Text Command Reference*.

## Using Cell Footprints

Timing optimization can use information in a footprint file. For example, the buffering mechanisms in optDesign add cells only if they are defined in the buffer footprint file.

To disable the footprintless flow (the default timing optimization flow) and load a footprint file, specify the following command:

```
loadFootPrint -infile footprint_file_name
```

For more information, see [loadFootPrint](#) in the "Timing Optimization Commands" chapter of the *EDI System Text Command Reference*.

Define footprints in your library or a footprint file by using the following commands, which are enabled when you specify `loadFootPrint`:

- `setBufFootPrint`
- `setInvFootPrint`
- `setDelayFootPrint`

**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 [Design Browser](#) in the "Tools Menu" chapter of the *EDI System Menu Reference*.

## Default Naming Conventions

<b>Prefix</b>	<b>Description</b>	<b>Command</b>
FE_MDBC	Instance added by multi-driver net buffering	<a href="#">optDesign</a>
FE_MDBN	Net added by multi-driver net buffering	<a href="#">optDesign</a>
FE_OCP_DRV_C	Instance added by DRV fixing	<a href="#">optDesign</a>
FE_OCP_DRV_N	Net added by DRV fixing	<a href="#">optDesign</a>
FE_OCP_RBC	Instance added by rebuffering	<a href="#">optDesign</a>
FE_OCP_RBN	Net added by rebuffering	<a href="#">optDesign</a>
FE_OCPC	Instance added by critical path optimization	<a href="#">optDesign</a>
FE_OCPN	Net added by critical path optimization	<a href="#">optDesign</a>
FE_OFC	Buffer instance added by rule-based buffer insertion	<a href="#">insertRepeater</a> / <a href="#">optDesign</a>
FE_OFN	Buffer net added by rule-based buffer insertion	<a href="#">insertRepeater</a> / <a href="#">optDesign</a>
FE_PHC	Instance added by hold time repair	<a href="#">optDesign</a>
FE_PHN	Net added by hold time repair	<a href="#">optDesign</a>
FE_PSBC	Instance added by buffer insertion in optDesign - postRoute	<a href="#">optDesign</a>
FE_PSBN	Net added by buffer insertion in optDesign - postRoute	<a href="#">optDesign</a>
FE_PSC	Instance added by postroute setup repair	<a href="#">optDesign</a>

FE_PSN	Net added by postroute setup repair	<a href="#">optDesign</a>
FE_RC	Instance created by netlist restructuring	<a href="#">optDesign</a>
FE_RN	Net created by netlist restructuring	<a href="#">optDesign</a>
FE_USC	Instance added during useful skew optimization	<a href="#">optDesign</a>
FE_PDC	Instance added by postroute DRV fixing	<a href="#">optDesign</a>
FE_PDN	Net added by postroute DRV fixing	<a href="#">optDesign</a>

---

## 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

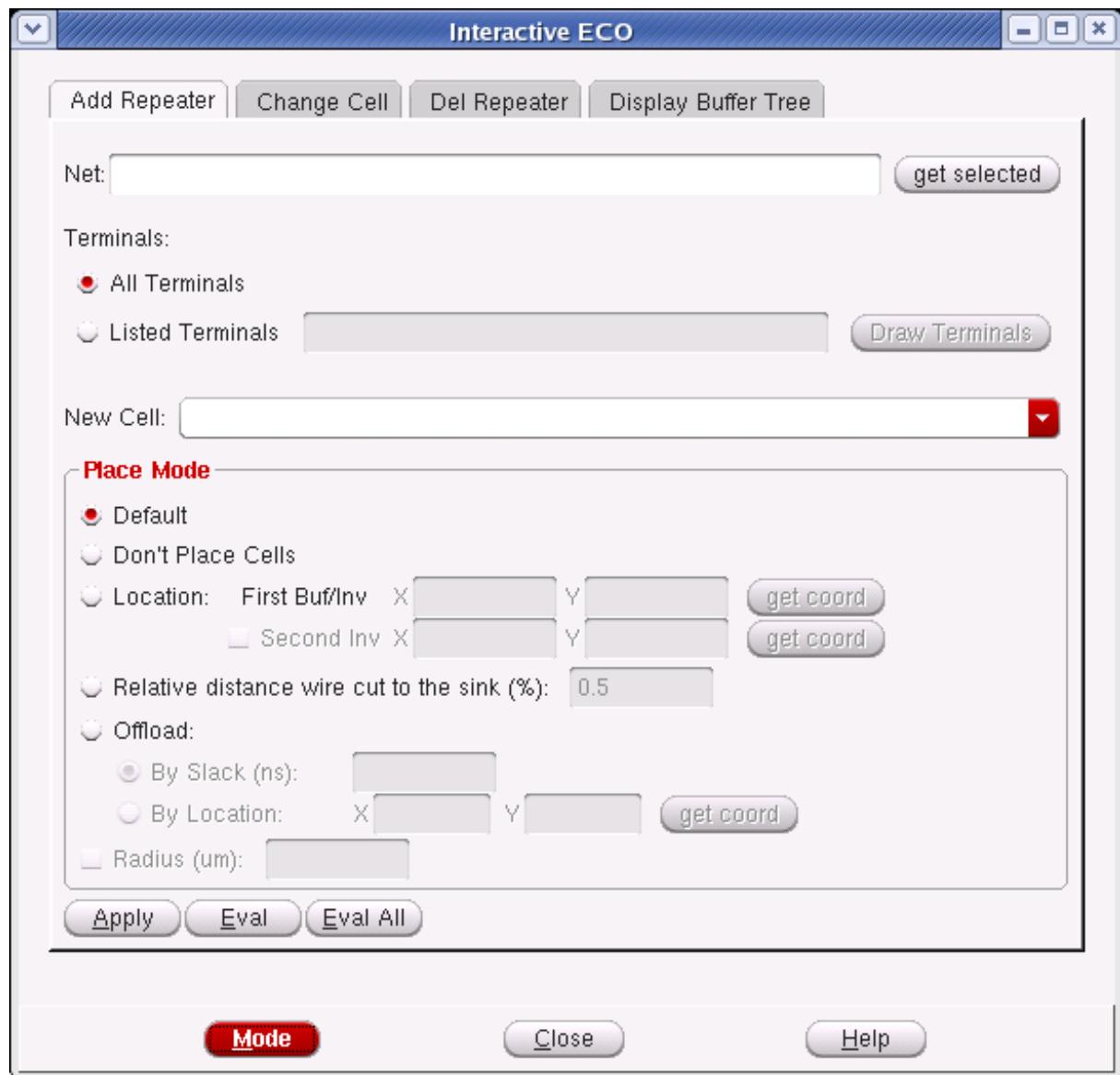
The following output files are generated:

- 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 *Optimize - Interactive ECO* from the EDI System 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 Mode* pane, you can choose one among several options:

- *Default*

The software automatically determines a location and places the new cell.

- *Don't Place Cells*

Specifies that the inserted cells should not be placed. Only the logical change in connectivity will be made.

- *Location*

Enter the location for the buffer using one of the following methods:

- You can use the automatically assigned locations, enter the locations, or click on an area in the design display window and click *get coord*.

- *Relative Distance to the Sink*

Specifies the location of the buffer based on its distance from the sink or the driver pin. The value is a number between 0 and 1. A low value (0.1) places the buffer near the sink; a high value (0.9) places the buffer near the driver. The fraction is based on the length of the wire.

This option works when one term is provided; it does not work if no term or multiple terms are specified.

- *Offload*

- i. 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.
- ii. 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 [attachIOPort](#) 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:

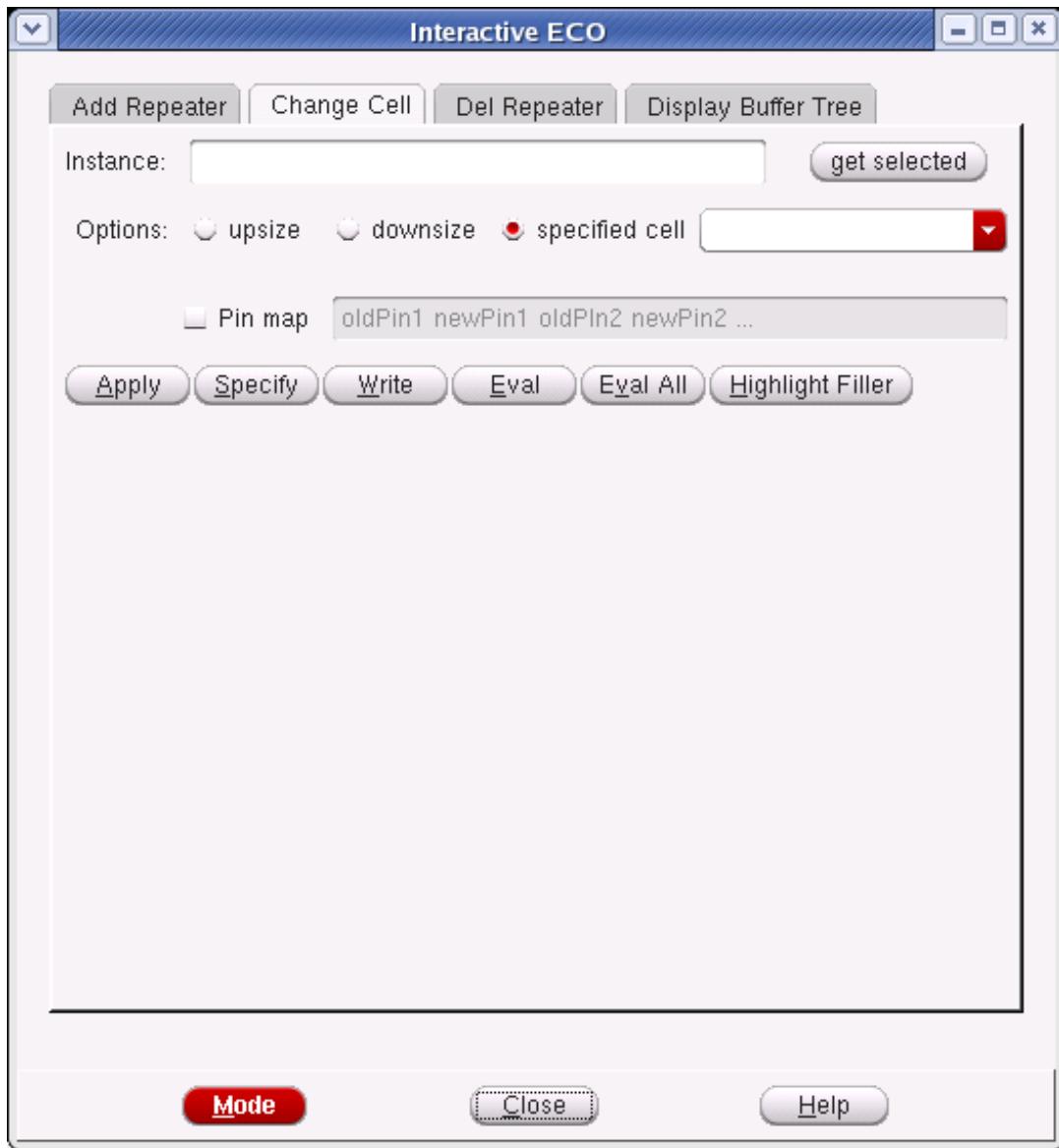
- [ecoAddRepeater](#)

For more information, see "[Interactive ECO Commands](#)" in the *EDI System 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 Optimize - Interactive ECO from the EDI System 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 *EDI System 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 Optimize - Interactive ECO from the EDI System 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:

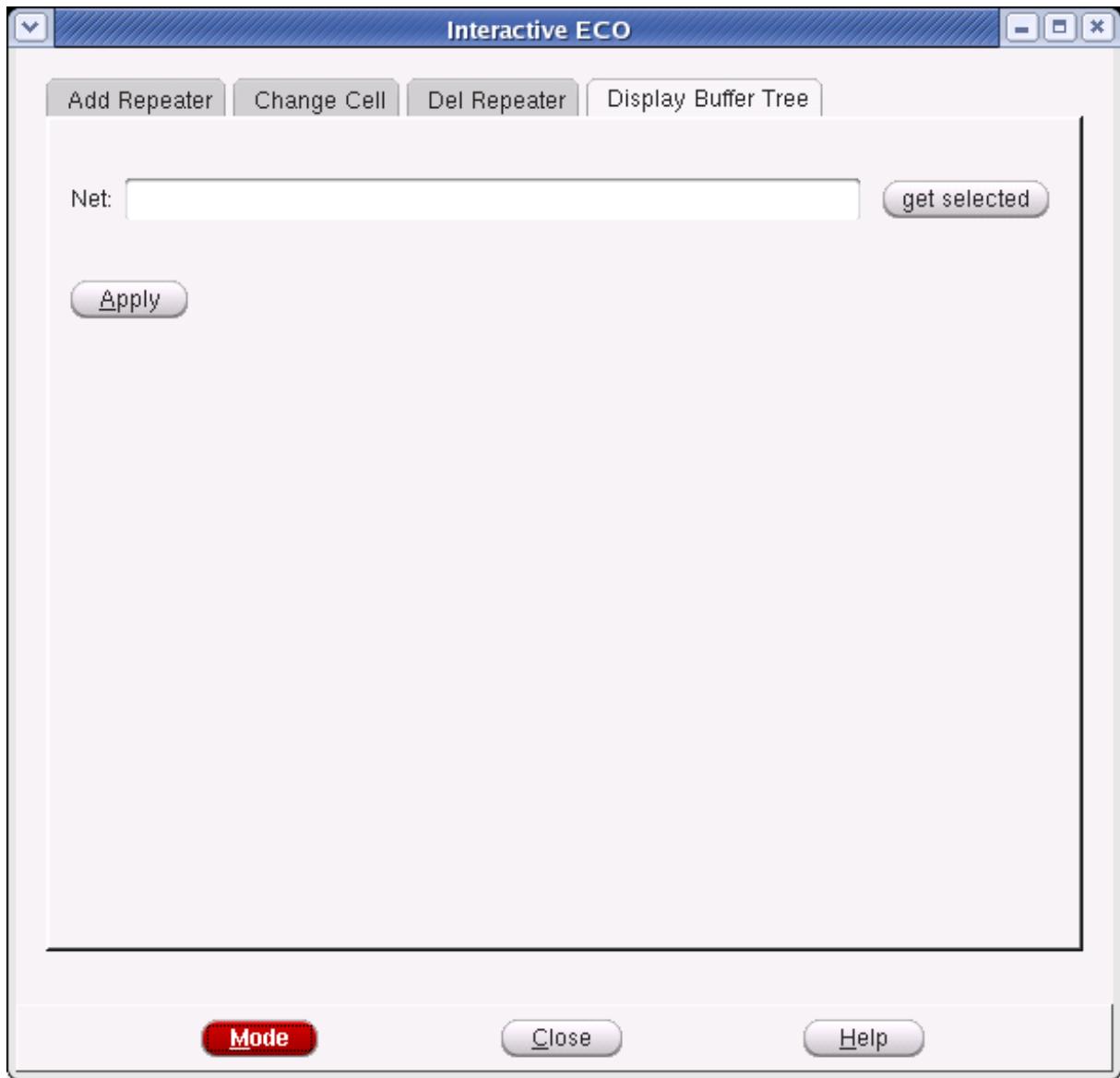
- [ecoDeleteRepeater](#)

For more information, see "[Interactive ECO Commands](#)" in the *EDI System 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 Optimize - Interactive ECO from the EDI System 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 *EDI System Text Command Reference*.

## Running ECO Placement

ECO placement updates the placement from a prior EDI System 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 EDI System 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 EDI System session.
3. Import the (ECO) design.
4. Load the floorplan.
5. Run ECO Placement.

This references the pre-ECO netlist placement data. The changes reflected in the new netlist are ECO'd into the pre-ECO placement. All designs are placed in the resulting placement.

After running ECO successfully, you can run Trial Route to view the routing congestion and analyze the design for timing.

## Naming Conventions for Interactive ECO

After running interactive ECO, you can use the Design Browser to view the newly added instance names, prefixed with FE\_. The interactive ECO operation naming conventions are described in the following table:

Name Prefix	Description
FE_ECON	A net added by interactive ECO
FE_ECOC	An instance added by interactive ECO

## Comparing Physical Design Data

After making changes to a DEF file, you can compare the new file to the information stored in the EDI System 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.

- CHANGECCELL statement

CHANGECCELL *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, EDI System considers this change as a deletion and addition of the instance. Rather than including a CHANGECCELL 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, EDI System 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, EDI System 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.

## Integration with LPA and CCP

---

- [Overview](#)
- [Before You Begin](#)
- [Results](#)
- [Running LPA from Encounter](#)
- [Running CCP from Encounter](#)

### Overview

The integration of Litho Physical Analyzer (LPA) and Cadence CMP Predictor (CCP) with Encounter® Digital 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 Encounter 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 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 Encounter window to configure LPA and CMP runs on the design. However, the *DFM* menu might not appear on your Encounter window or one of its submenu options might be disabled if the prerequisite conditions are not satisfied (See Before You Begin).

### Before You Begin

Before you can run LPA and CCP from Encounter, the following conditions must be met:

- You must have the LPA license to run litho sign-off from Encounter. However, to run LPA in Routing Layers Only mode, the `Encounter_DFM_GXL` license is sufficient and no separate LPA license is required.
- You must have the CCP license to run CCP from Encounter.
- You must have either `Encounter_Adv_Node_GXL` OR `Encounter_DFM_GXL` license.
- You must be able to launch version 9.2 (or a later version) of LPA and CCP from Encounter. In other words, the installation path to LPA and CCP must be present in your path variable.
- You must have LPA and 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.

### 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 *Encounter Violation Browser* and fix the reported hotspots using NanoRoute.

## Running LPA from Encounter

The integration of LPA with Encounter 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 Encounter 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.

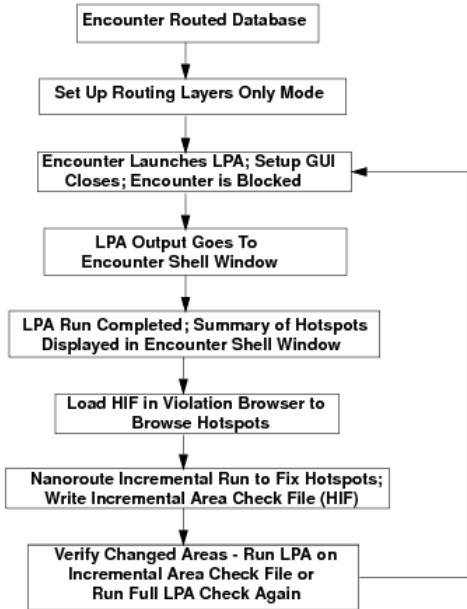
### Routing Layers Only Mode

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 Encounter DFM GXL Option license.

The following diagram shows the design flow for running LPA in Routing Layers Only mode.

#### LPA in Routing Layers Only Mode

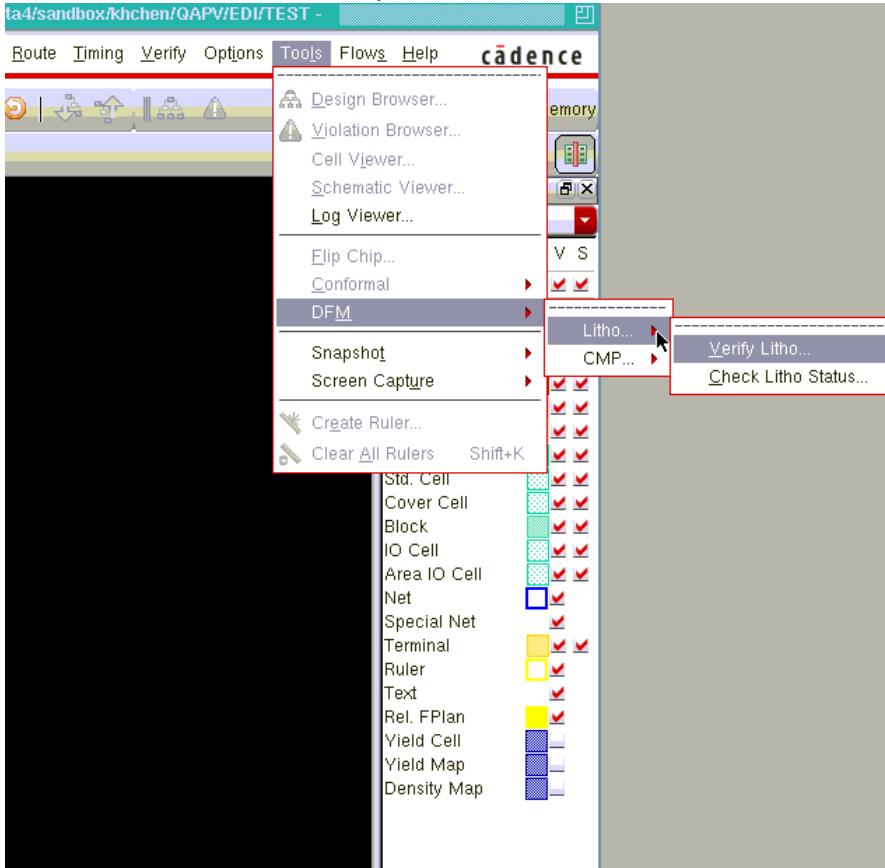


#### Running LPA in Routing Layers Only Mode

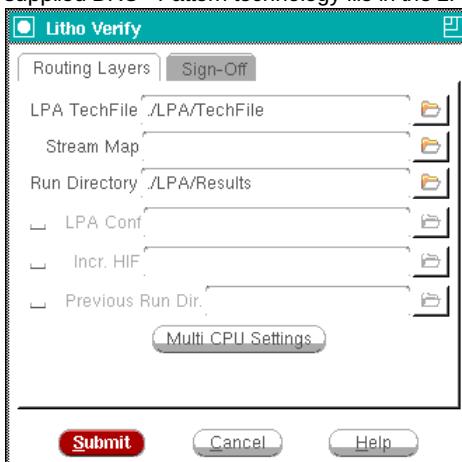
To run LPA in Routing Layers Only mode, launch Encounter by using the `encounter` command and load the design. When Encounter is invoked, it automatically loads all the required LPA files from the

LPA installation path. Once the Encounter GUI is displayed, perform the following steps:

1. Choose *Tools* -> *DFM* -> *Litho* -> *Verify Litho* from the Encounter menu.



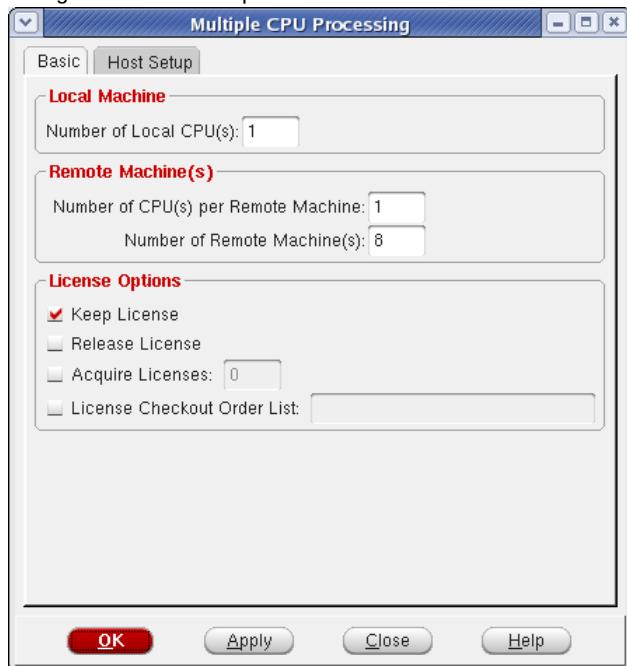
2. This opens the *Litho Verify* form, with the *Routing Layers* tab selected by default. In the *LPA TechFile* field, specify the path and name of the qualified LPA Mx- technology file that includes process-specific hotspot checking options and the LPA model. Alternatively, if you want to enable the GLOBALFOUNDRIES DRC+ flow, you specify the path to the foundry-supplied DRC+ Pattern technology file in the *LPA TechFile* field.



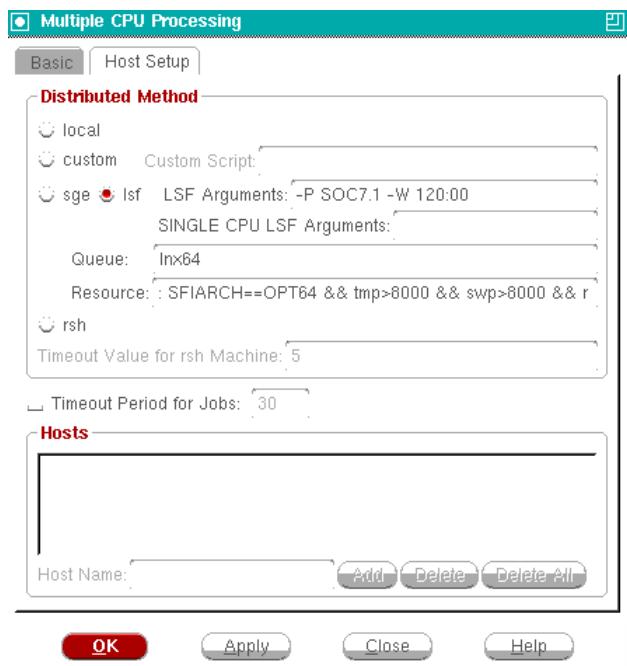
3. In the *Stream Map* field, specify the stream out map file, which maps the GDS stream to the layers in the EDI System database. Ensure that the GDS layer numbers in the stream out layer

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 Encounter's *Multi-CPU Settings* GUI. 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 Encounter commands.



9. On the *Basic* tab of the *Multiple CPU Processing* form, only the *Number of Remote Machine(s)* field has an effect on the LPA run. In this field, specify the number of LSF machines that you want to use for the LPA run.
  10. On the *Host Setup* tab, select the *lsf* radio button to set the distribution method as LSF and specify the LSF arguments in the *LSF Arguments* field.
- Note:** Only the LSF distribution method is supported for the Encounter-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 Encounter GUI until the LPA run is completed.

During the LPA run, the output is sent to the *Encounter Shell* window. After LPA is successfully completed, the LPA summary file with hotspot count is presented in the *Encounter Shell* window.

```
**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:

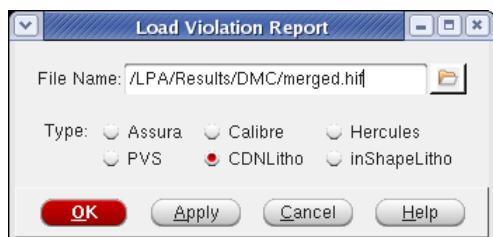
+---+---+---+---+
| Layer | L1 | L2 | L3 | L4 |
+---+---+---+---+
| M2 | 3 | n/a | n/a | n/a |
| M3 | 0 | n/a | n/a | n/a |
| M4 | 0 | n/a | n/a | n/a |
| M5 | 0 | n/a | n/a | n/a |
| M6 | 0 | n/a | n/a | n/a |
| M7 | 0 | n/a | n/a | n/a |
| M8 | 0 | n/a | n/a | n/a |
+---+---+---+---+
```

The terminal window has a menu bar with 'Session', 'Edit', 'View', 'Bookmarks', 'Settings', and 'Help'. At the bottom, there are tabs for 'Shell' (selected), 'Shell No. 2', and 'Shell No. 3'.

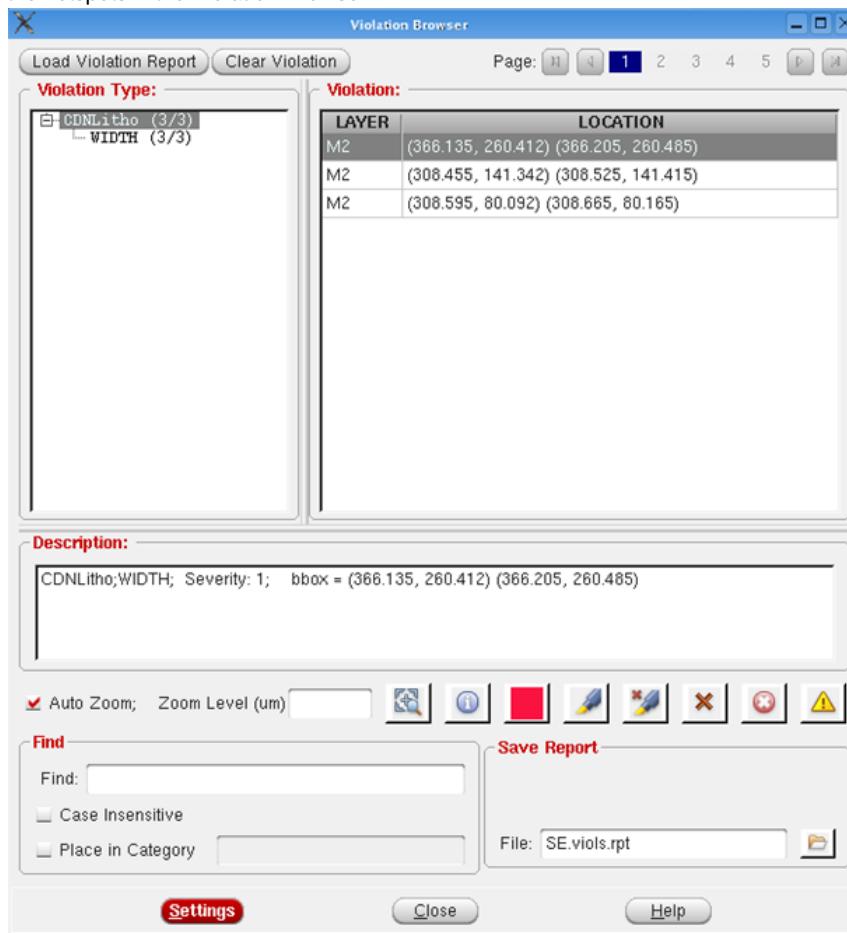
#### Viewing Hotspots

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:

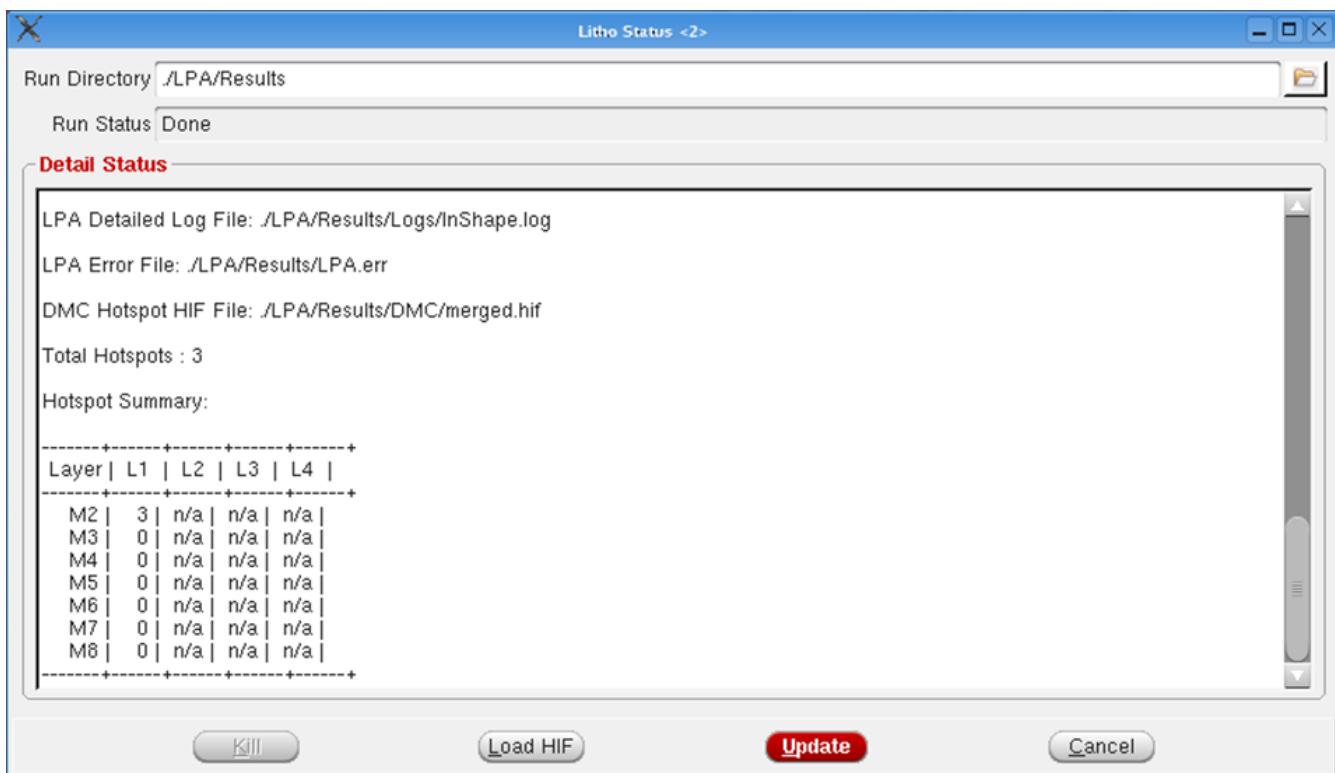
1. Select *Tools* -> *Violation Browser* -> *Load Violation Report*.
2. This opens the *Load Violation Report* form. Specify the path and name of the HIF file in the *File Name* field.



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. To do this, perform the following set of steps:

1. In the encounter command line, type the following to set up Nanoroute and run it on the design to repair the litho hotspots.  

```
encounter> setNanoRouteMode -droutePostRouteLithoRepair true
encounter> setNanoRouteMode -drouteMinimizeTopologyChange true
encounter> globalDetailRoute
```
2. After Encounter has completed `globalDetailRoute`, save the design for verification. This will also save the markers within the design, indicating the areas of change.  

```
encounter> saveDesign Litho_Fixed.enc
```
3. The Encounter 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.  

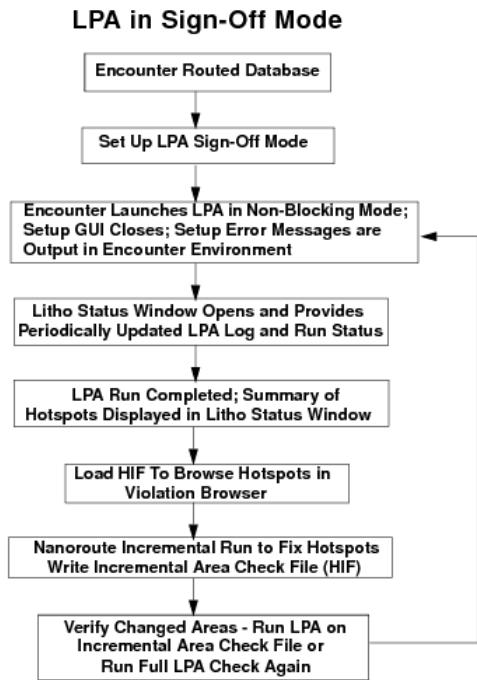
```
encounter> writeHif -file IncrVerify.hif
```
4. Now, run LPA in the incremental area to check for hotspots after Encounter has fixed the litho hotspots. Open the *Litho Verify* form, with the *Routing Layers* tab selected by default. Specify the name of the incremental HIF file in the *Incr. HIF* 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.

## 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.

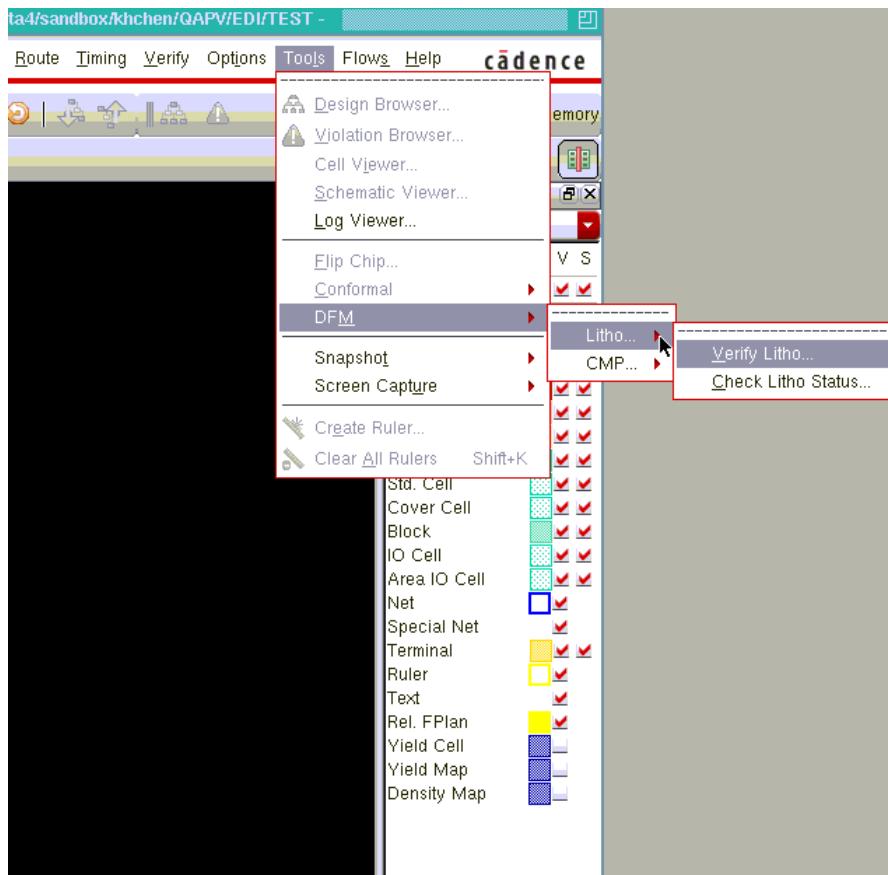
The following diagram shows the design flow for running LPA in Sign-Off mode.



### Running LPA in Sign-Off Mode

To run LPA in Sign-Off mode, launch Encounter by using the encounter command and load the design. When Encounter is invoked, it automatically loads all the required LPA files from the LPA installation path. Once the Encounter GUI is displayed, perform the following steps:

1. Choose *Tools -> DFM -> Litho -> Verify Litho* from the Encounter menu.

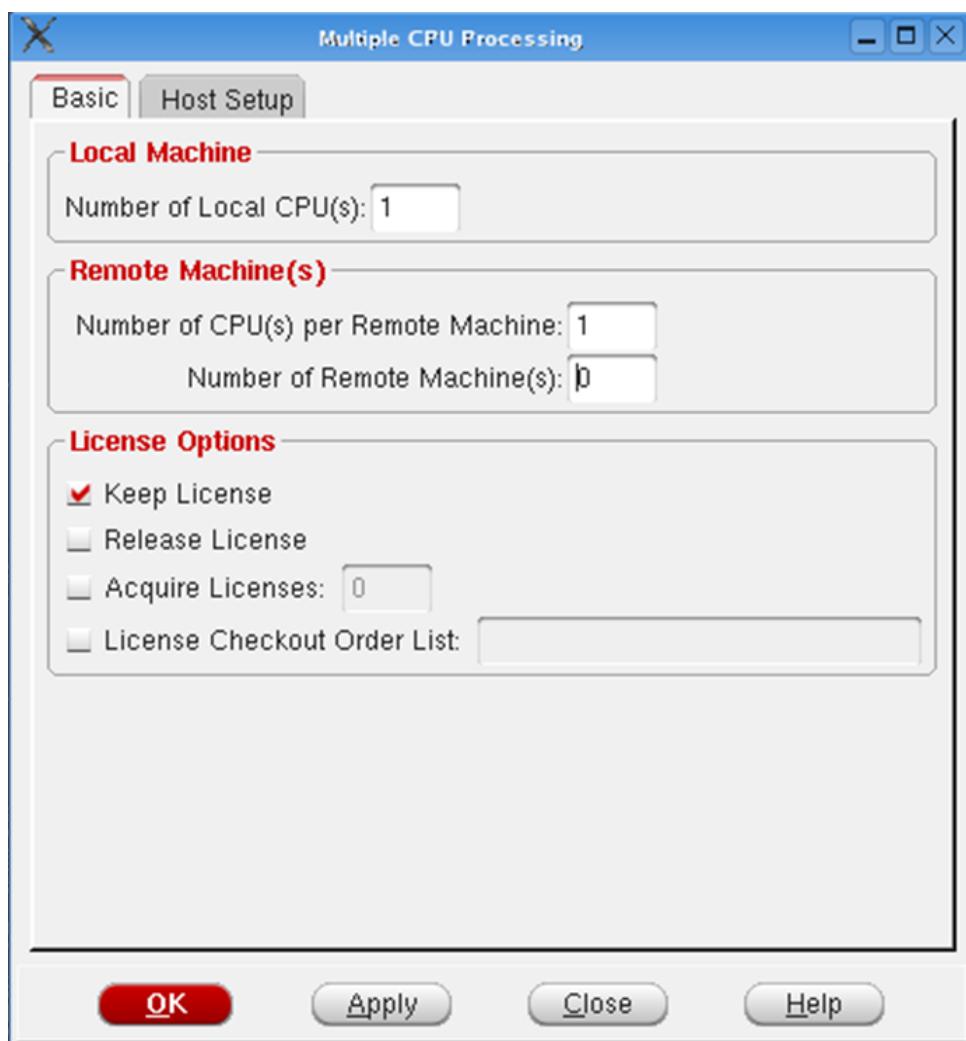


2. This opens the *Litho Verify* form, with the *Routing Layers* tab selected by default. Select the *Sign-Off* tab.

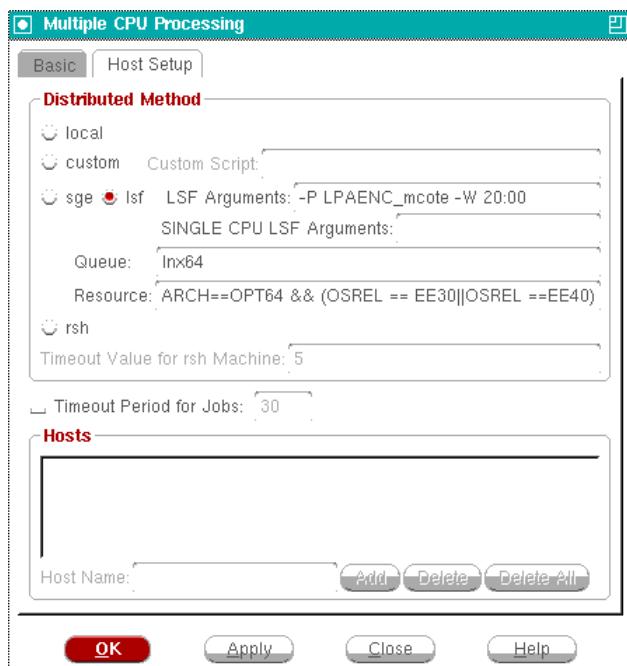
3. Specify the path and name of the LPA output directory in the *Run Directory* field. All output data from the LPA run will be stored under this directory.
4. Specify the name and path of the LPA configuration file in the *LPA Conf* field. This configuration file should contain the Techfile location 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 Encounter to map the GDS stream to the layers in the Encounter 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 Encounter's *Multi-CPU Settings* GUI. 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 Encounter commands.



10. On the *Basic* tab of the *Multiple CPU Processing* form, only the *Number of Remote Machine(s)* field has an effect on the LPA run. In this field, specify the number of LSF machines that you want to use for the LPA run.
11. On the *Host Setup* tab, select the *lsf* radio button to set the distribution method as LSF and specify the LSF arguments in the *LSF Arguments* field.  
**Note:** Only the LSF distribution method is supported for the Encounter-LPA integration.

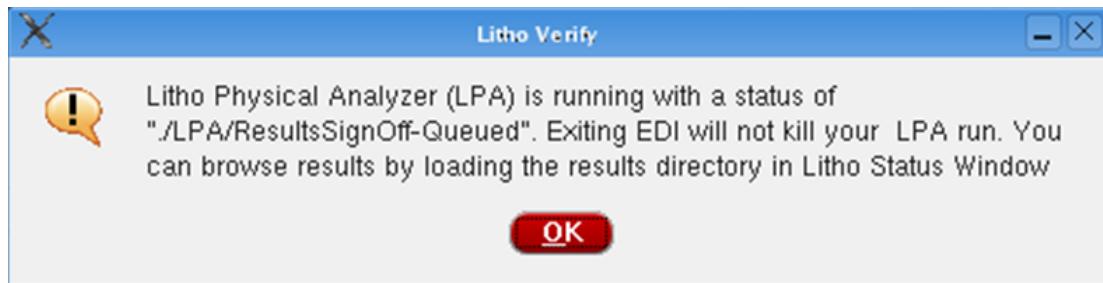


12. Specify the queue and resource string that is needed for the LSF launch in the *Queue* and *Resource* fields.

13. Select the *OK* button to confirm the multiple CPU settings and close the form.

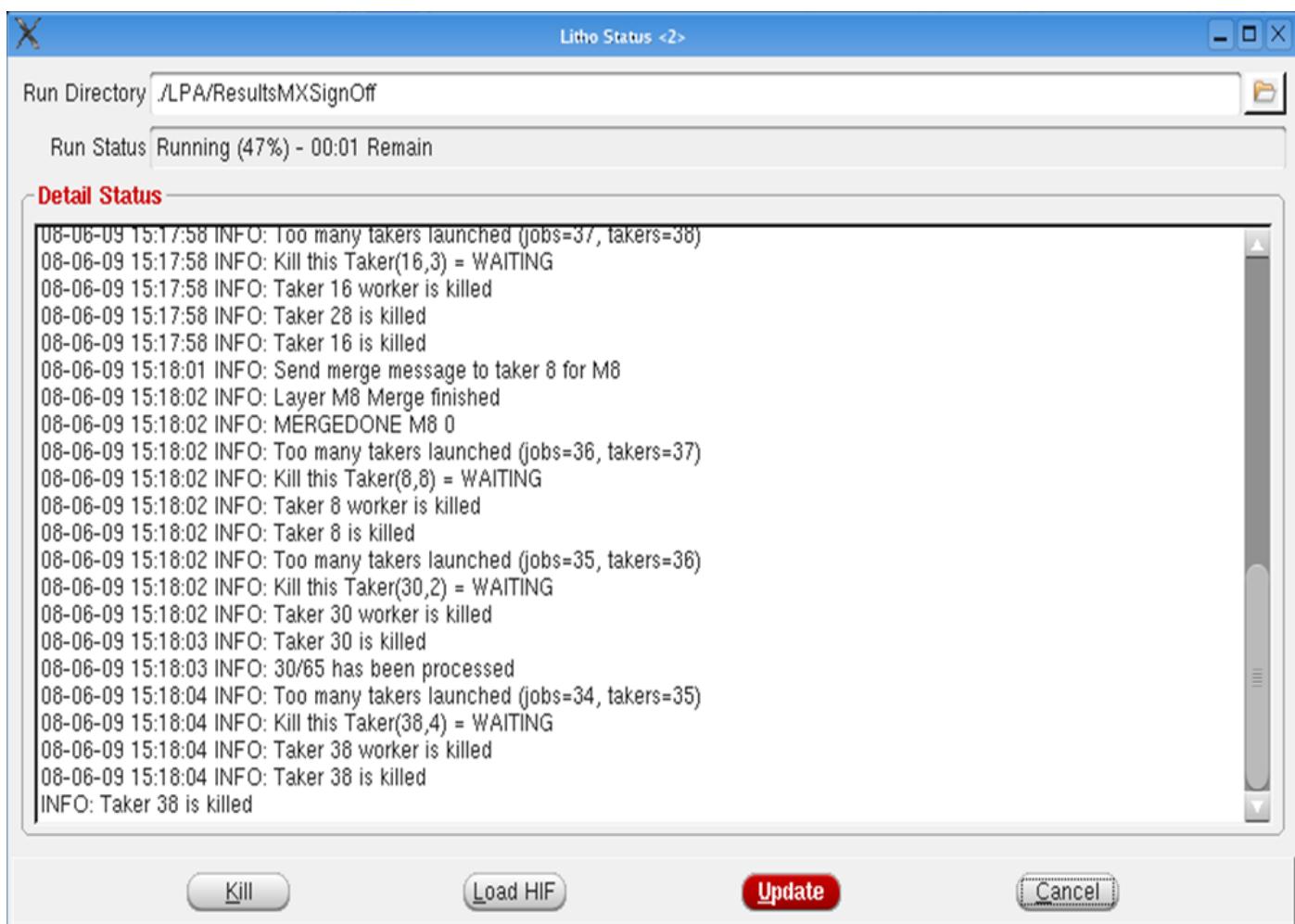
14. Select the *Submit* button in the *Litho Verify* form to launch the sign-off LPA run with the specified settings.

**Note:** In Sign-Off mode, LPA runs in non-blocking mode. You can continue working with the Encounter GUI and perform design activities in parallel with Litho sign-off. If you exit Encounter during this period, a warning is displayed informing you that LPA is still running and you can load the results later.

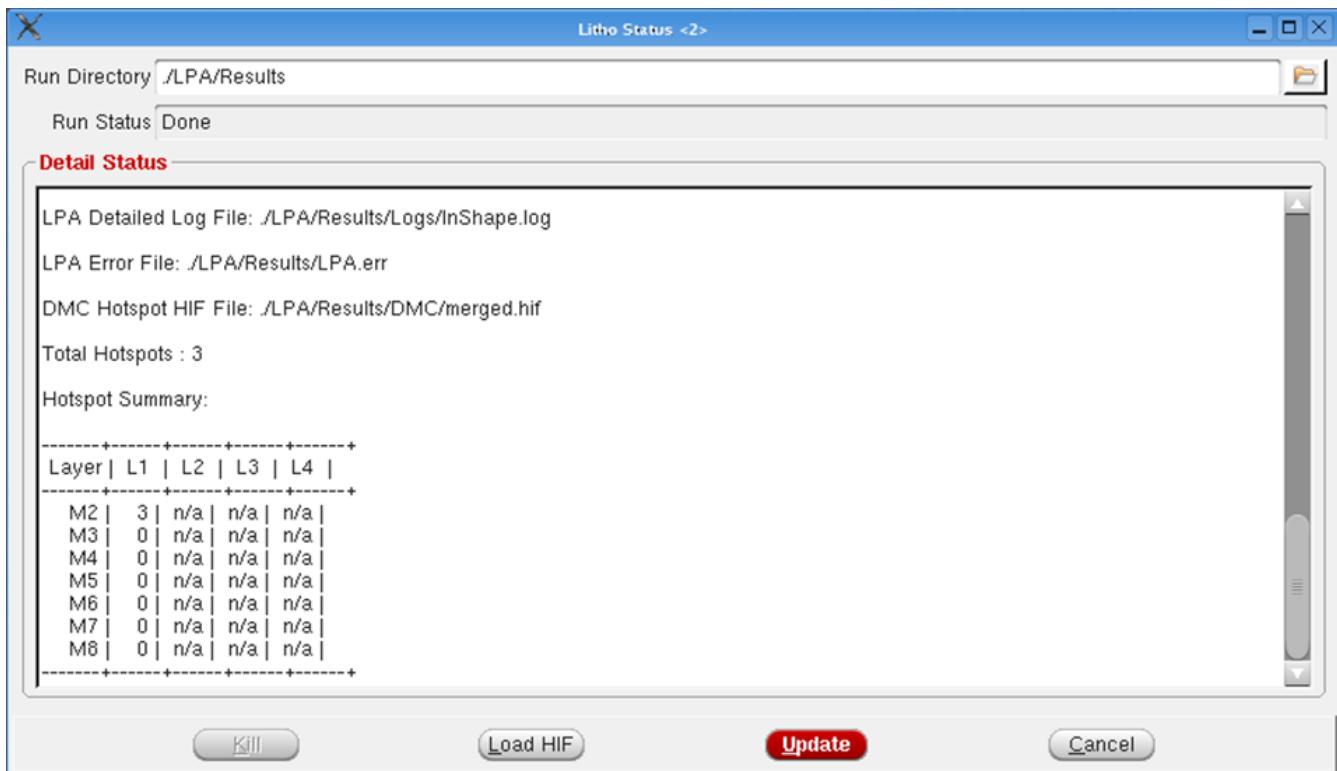


15. On the successful launch of LPA, the *Litho Verify* form closes and the *Litho Status* window is automatically displayed. The *Run Directory* field of this window is automatically populated from the *Litho Verify* form.

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* .

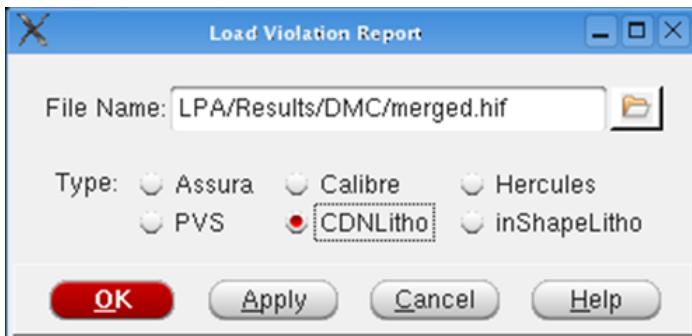


**Note:** You can also specify a different run directory name in the *Run Directory* field of the *Litho Status* window and then click the *Update* button to check the output of the specified LPA run.

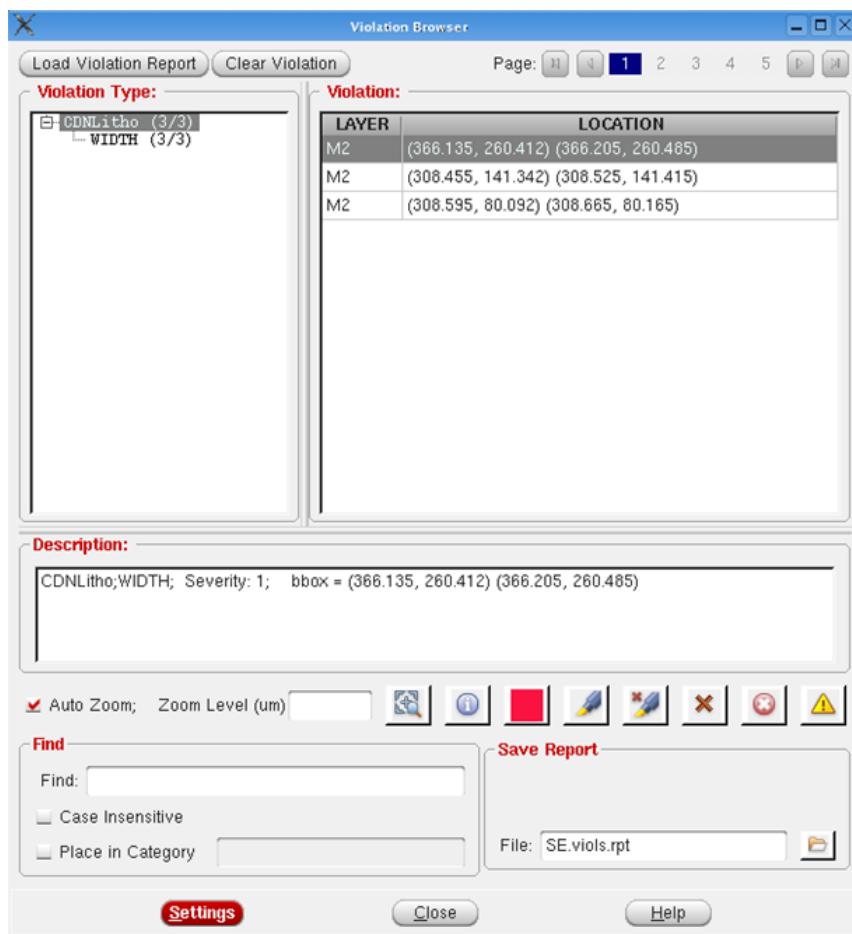
#### Viewing Hotspots

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:

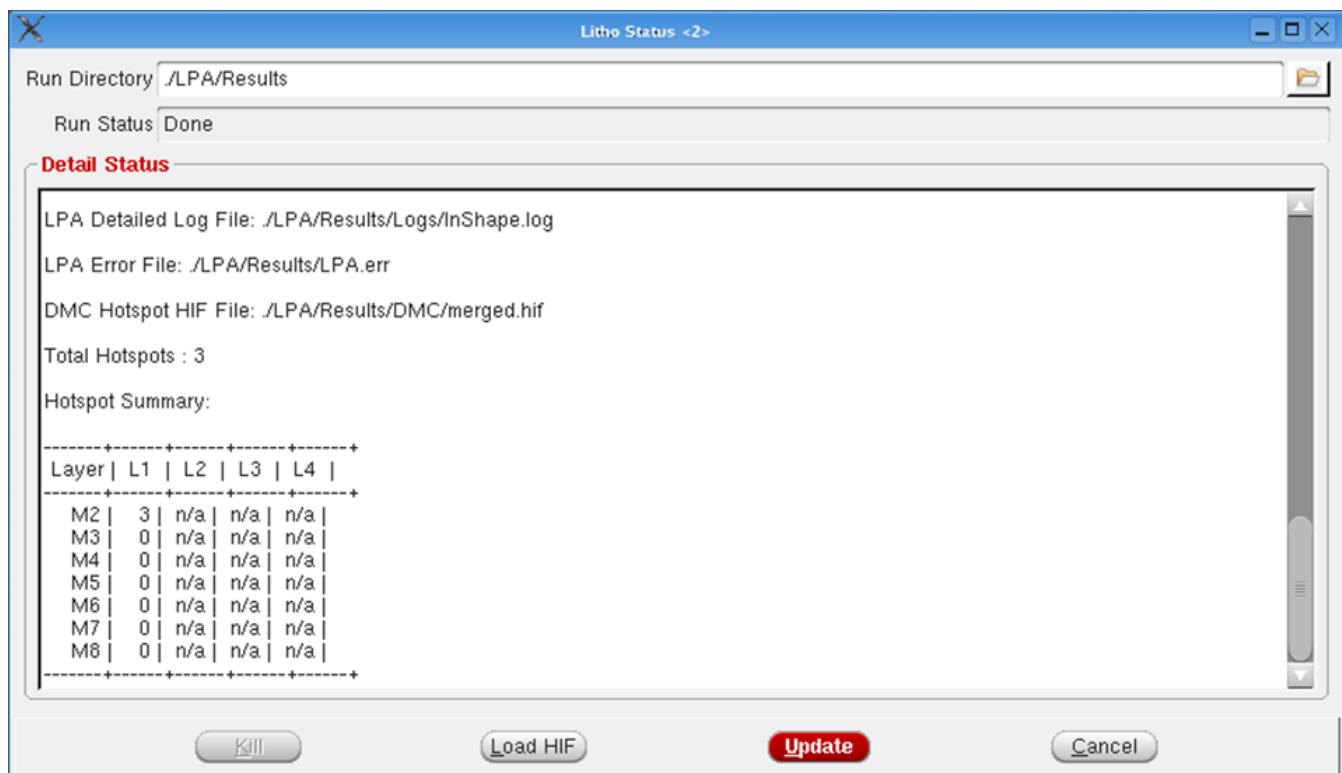
1. Select *Tools* -> *Violation Browser* -> *Load Violation Report*.
2. This opens the *Load Violation Report* form. Specify the path and name of the HIF file in the *File Name* field.



3. Select the *CDNLItho* radio button to specify the HIF format and select the *OK* button to view the hotspots in the *Violation Browser*.

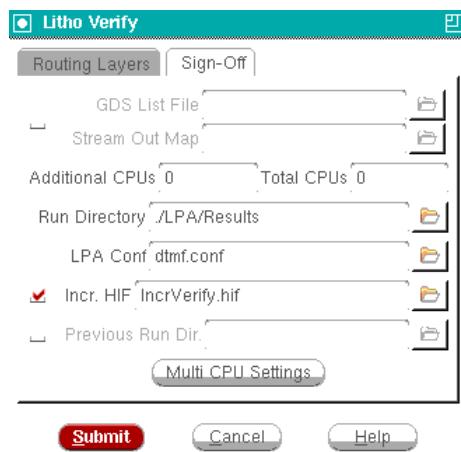


4. You can also load the HIF file from the *Litho Status* window (*Tools -> DFM-> Litho -> Check Litho Status*) by specifying the results directory name in the *Run Directory* field and selecting the *Load HIF* button. This will open up the *Violation Browser* to show the hotspot details.



#### Fixing Hotspots

Next, you fix the hotspots identified by the LPA run by using Nanoroute. The Encounter 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.



#### Running CCP from Encounter

The integration of CCP with Encounter 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 Encounter.

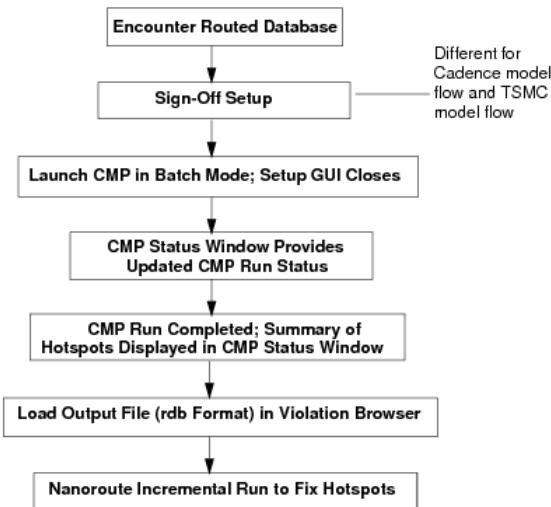
- Supports manufacturability checks of routed blocks
- Supports fast check and sign-off verification
- Eliminates translation and setup hassles
- Runs with default out-of-the-box setup

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 need to additionally set up the VCMP engine and process file.

### **CCP Flow in Encounter**

The following diagram shows the design flow for running CCP in Sign-Off mode.

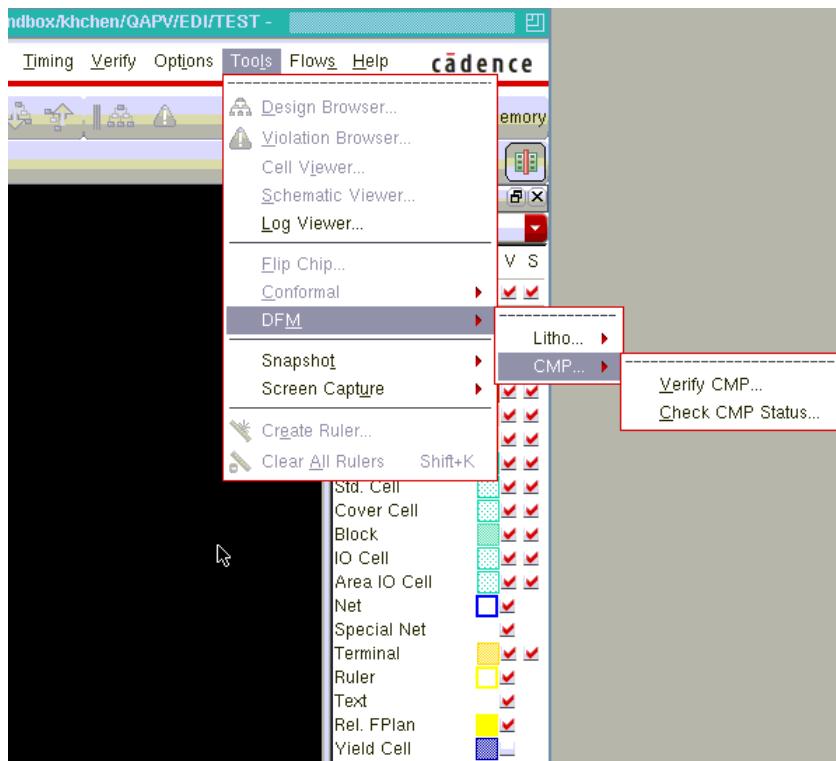
**CCP in Sign-Off Mode**



### **Running CCP in Cadence Model Flow**

To run CCP analysis on your design using the Cadence model flow, launch Encounter by using the `encounter` command and load the design. When Encounter is invoked, it automatically loads all the required CCP files from the CCP installation path. Once the Encounter GUI is displayed, perform the following steps:

1. Choose *Tools -> DFM -> CMP -> Verify CMP* from the Encounter 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. The extraction and prediction results of the CMP analysis are based on the specifications in your *vmp* file.

**CMP Verify**

**Sign-Off**

- GDS List File: Hier.GDS.list.txt
- Stream Out Map: StreamOut.map
- VMP Files: Generic\_calib\_all.xml Generic
- Output Directory: ./CCP/Results\_check
- VMP Name: Generic\_45nm\_Vmp
- CCP Conf: /path/to/ccp.conf
- Start Level: [ ] End Level: [ ]
- DP Setting**
- TSMC Process
- VCMP Engine: ./CCP/libvcmpDynamic.so
- Process File: ./CCP/VCMP\_N45GS.enc

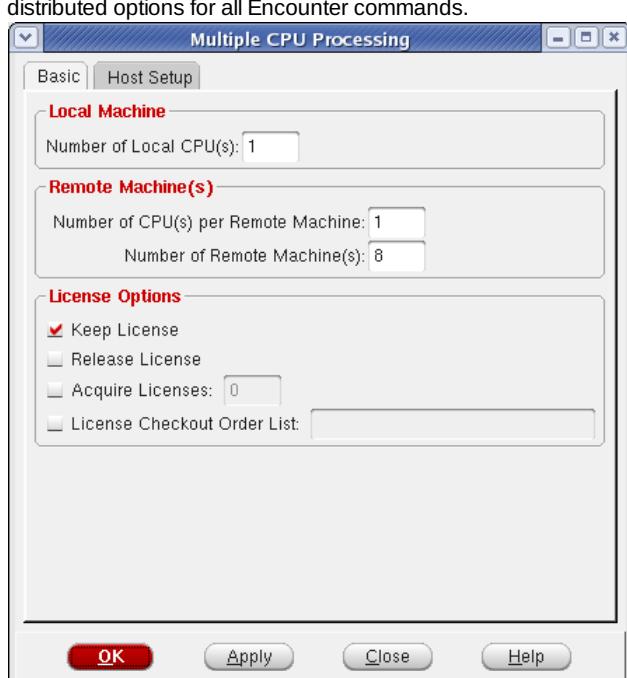
**Submit**   **Cancel**   **Help**

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 *vmp* process calibration 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.

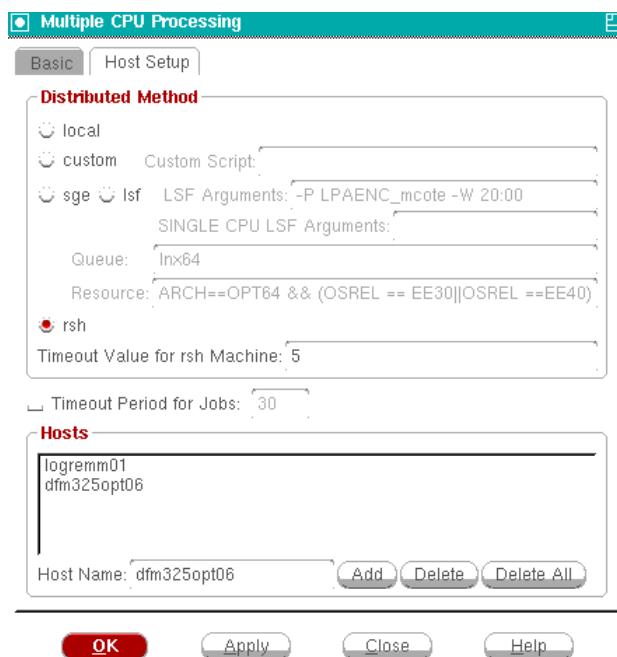
These fields are optional. You can specify the starting metal level and end metal level in the *Start Level* and *End Level* fields for the CMP simulation. You can either type in the values or select them from the drop-down menus associated with these two fields.

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 Encounter to map the GDS stream to the layers in the Encounter 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 Encounter's *Multi-CPU Settings* GUI. 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 Encounter 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 Encounter GUI and perform design activities while the CMP simulation is running. If you exit Encounter 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.



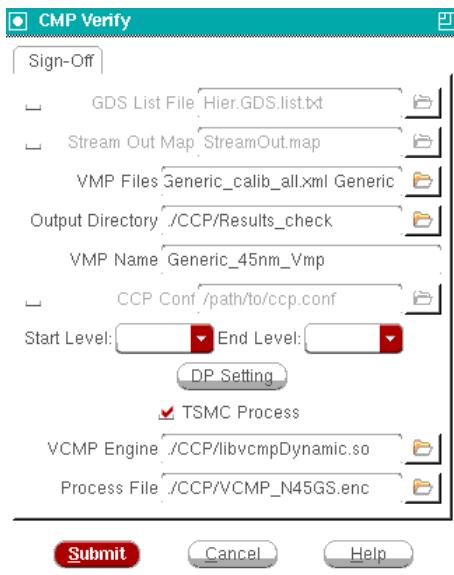
On completion of the CMP run, the *CMP Status* window displays the summary of the hotspots detected by the run and an *rdb* format output file is generated in the results directory.

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.



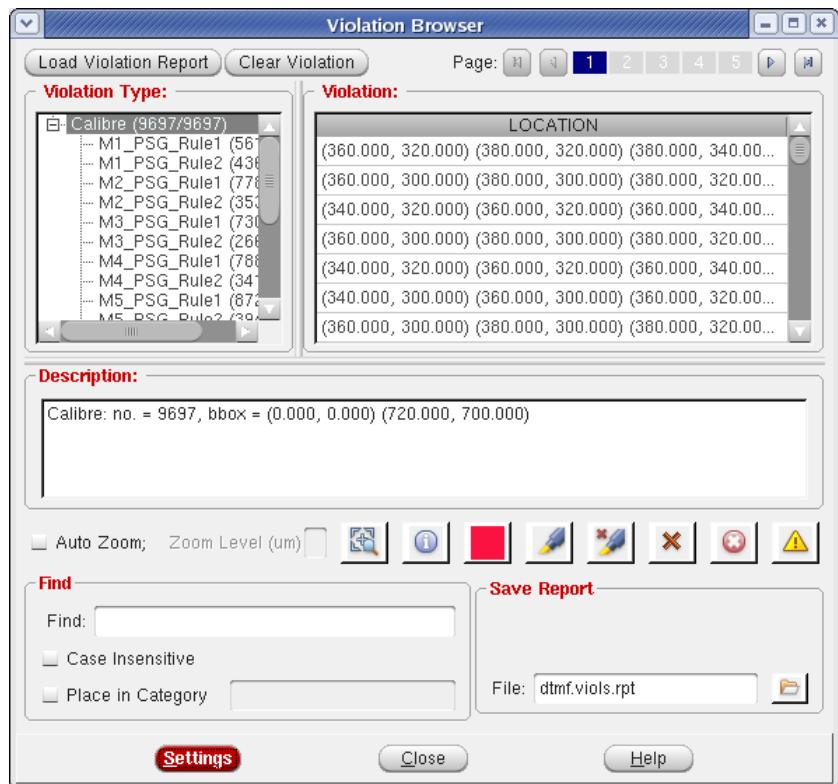
1. Select the *TSMC Process* check box if you want to use the TSMC model instead of the Cadence model.
2. In the *VCMP Engine* field, specify the location and name of the VCMP engine that is to be used for running CMP.
3. In the *Process File* field, specify the location and name of the VCMP process 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.



---

# Analyzing and Repairing Crosstalk

---

- [Overview](#)
- [Inputs and Outputs for SI Analysis](#)
- [Setting Up Encounter for SI Analysis](#)
  - [RC Extraction Settings](#)
  - [Noise Analysis Settings](#)
  - [Static Timing Analysis \(STA\) Settings](#)
  - [Advanced Settings for SI Analysis](#)
  - [Example of Setting Up Encounter for SI Analysis](#)
- [Preventing Crosstalk Violations](#)
- [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](#)
- [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 Encounter™ software supports signal integrity (SI) operations that include crosstalk prevention, analysis, and repair. The software uses an advanced crosstalk repair algorithm which features:

- Gate sizing
- Buffer insertion
- Victim spacing and protection

**Note:** NanoRoute™ is seamlessly integrated with these tools to perform all crosstalk analysis and repair operations.

Analyzing and repairing crosstalk is part of the signal integrity closure repair loop, which reduces both

timing and crosstalk violations starting from the prevention stage to the post-route optimization stage in the design flow.

## Inputs and Outputs for SI Analysis

The following design input files are required in order to repair crosstalk violations:

- Netlist
- SDC (timing information)
- Routed Encounter database or DEF file (placement and routing information)
- LEF file (physical library)
- .cdb noise library
- XILM data
- Liberty library (.lib)
- Encounter extended capacitance table file
- QRC standalone extraction technology file and library (optional)

**Note:** Before you begin, ensure that your routed design meets all the timing requirements.

When finished, the software produces an Encounter database optimized for crosstalk violations. The following files are generated:

- Incremental SDF file
- Incremental transition time file
- Timing reports with and without incremental delays

Use the `-detailedReports` parameter of the [setSIMode](#) command to control the report files generated during SI analysis. Set this parameter to `true` if you want to enable detailed reporting for debugging purposes.

## Setting Up Encounter for SI Analysis

- RC Extraction Settings
- Noise Analysis Settings
- Static Timing Analysis (STA) Settings
- Advanced Settings for SI Analysis
- Example of Setting Up Encounter for SI Analysis

### RC Extraction Settings

The RC extraction settings for SI analysis include the extraction engine and the extraction filters.

#### Extraction Engine

You can use one of the following post-route extraction engines:

- Detail
- Turbo QRC (TQRC)
- Integrated QRC (IQRC)
- Standalone QRC

For above 65nm technology, EDI System uses detailed extraction for post-route timing and/or optimization. However, for 65nm and below technology, if appropriate setup is available, EDI System uses TQRC as the default post-route extraction engine. For superior correlation with signoff extraction, use of TQRC and IQRC extraction engine is recommended. The Integrated QRC extraction engine provides the highest accuracy in implementation flow and is particularly recommended at ECO for incremental extraction.

**Note:** Integrated QRC extraction requires a separate QRC license.

To use either the TQRC or IQRC extraction engine, use the following command:

```
setExtractRCMode -engine postRoute -effortLevel [medium | high]
```

Where:

- `medium` invokes the TQRC engine
- `high` invokes the IQRC 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 the most significant impact on SI analysis and run time. EDI System automatically sets the default values for the RC extraction filters based on the process node specified using the -process parameter of the [setDesignMode](#) command. To set the process node, use the following command:

```
setDesignMode -process process_node
```

**Note:** For more information on the default values assigned to the filtering parameters with respect to the specified process node, see the [setDesignMode](#) command.

If you do not want to use the default filtering values for RC extraction, specify the following parameters of the [setExtractRCMode](#) command to tweak the coupling capacitance filters:

- `-total_c_th`: Specifies the threshold value (femtoFarads) that determines when the extractor lumps a net's coupling capacitance to ground. The software grounds the coupling capacitances for nets which have a total capacitance value less than the value specified with this parameter.
- `-coupling_c_th`: Specifies the threshold value that determines when the extractor lumps a net's coupling capacitance to ground. The software decouples the coupling capacitance of nets when the total coupling capacitance between the pair of nets is lower than the threshold specified with this parameter.
- `-relative_c_th`: Sets a ratio threshold value that determines when the extractor lumps a net's coupling capacitance to ground. If the total coupling capacitance between a pair of nets is less than the percentage (specified with this parameter) of the total capacitance of the net with the smaller total capacitance in the pair, the coupling capacitance between these two nets will be considered for grounding.

### Guidelines for RC Extraction Settings

Use the following guidelines while setting up extraction:

- Note that the detailed extraction engine (default extraction engine for SI analysis) is

significantly faster compared to TQRC or IQRC. However, it trades off accuracy of the extraction results for performance. TQRC is about 30% faster than IQRC. TQRC and IQRC both support distributed processing and their use is strongly recommended to offset longer runtime. Both TQRC and IQRC can take 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 Encounter software for SI fixing are the same as the ones used for SI signoff analysis.
- The default filtering values set by the Encounter 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 Encounter software) with the RCs from your signoff extractor.
- While setting the filtering thresholds, ensure that you retain small coupling capacitors because SI analysis lumps these together into a single virtual attacker model. Multiple small coupling capacitors can result in a significant virtual attacker.
- Exercise caution while setting low-value filters because this can increase the run time significantly.

## Noise Analysis Settings

Noise analysis settings include loading the input noise model, configuring the timing windows, choosing the noise analysis engine, setting the delta delay threshold, and specifying the virtual attacker mode.

### Noise Models

The primary input for noise analysis is a `cdb` library, which contains characterized noise data. In the absence of a `cdb` file, you can use a Liberty library (`.lib`) file. However, it is strongly recommended that you use a `cdb` noise library for SI analysis.

**Note:** For hierarchical designs, you need XILM data. For more information on XILM-based SI analysis, see the [Using Interface Logic Models in Hierarchical Designs](#) chapter in the *Encounter User Guide*.

### Timing Windows

Timing windows are used to filter out signals that are not switching simultaneously. The internal timing engine computes the timing windows and slew rates automatically. Alternatively, you can load the timing window information by using a timing window format (TWF) file.

### ***Internally Generated Timing Windows***

You can use the following timing window settings based on your requirement:

- By default, the Encounter software computes the timing windows and uses that information during noise analysis. The following is the default setting for timing windows:
  - `setSIMode -noiseTwfMode " "`
- To use a conservative approach for analysis, set the timing windows to infinite switching mode as shown:
  - `setSIMode -noiseTwfMode "-infsW"`
- To use the timing window files generated by common timing engine (CTE) during SI analysis, specify the following command:
  - `setSIMode -noiseTwfMode "-useCTE"`

### ***Externally Generated Timing Windows***

To use an external timing window format (TWF) file, specify the [read\\_twf](#) command to load the timing window information. The TWF information specified with the `read_twf` command is passed to the internal SI engine and is honored by the `optDesign` and `timeDesign` commands.

Consider the following points when loading an external TWF file using the `read_twf` command:

- The `-skiptw` parameter is passed to the SI engine if you have specified the [setSIMode](#) `-noisTwfMode -infsW` command.
- The [read\\_twf](#) command can be used to read multiple TWF files (for each view) in case of MMMC designs. For example:

```
read_twf twfFile1 -view view1

read_twf twfFile2 -view view2
```

`optDesign -postRoute -si`
- If you have one TWF file for one of the views of an MMMC design, you can load the available TWF file for that view and the software will create the timing window information for all other views internally.
- Multiple TWF files can be loaded for the same view. In this case, all the loaded files will be

passed on to the SI analysis engine.

- Only one iteration of SI fixing is performed when using the external TWF file, and the flow stops after routing. You will need to regenerate the external TWF files for the modified design (after timing optimization) and then run the `timeDesign` command to get the updated timing numbers.
- For non-MMMC designs, if you are running hold fixing using the `-fixHoldIncludeXtalkSetup` parameter of the [`setSIMode`](#) command, then:
  - specify the setup TWF file using the `-setup` parameter of [`read\_twf`](#)
  - specify the hold TWF file using the `-hold` parameter of [`read\_twf`](#)

## Noise Analysis Engine

The Encounter software uses native signal integrity analysis to perform crosstalk analysis. This engine is same as the one used by Encounter Timing System. If you are using Encounter Timing System as the SI signoff tool, the same settings can be applied in Encounter for performing noise analysis by using equivalent CeltIC NDC commands.

To use CeltIC NDC commands in Encounter, use the `-insCeltICPreTcl` and `-insCeltICPostTcl` parameters of the [`setSIMode`](#) command, as shown:

```
setSIMode -insCeltICPreTcl { command_name ; }
```

```
setSIMode -insCeltICPostTcl { command_name ; }
```

 The Cadence® Encounter® Timing System provides a sign-off timing and signal integrity solution for a design flow and is shipped with the ETS release.

## Delta Delay Threshold

You can set the delta delay threshold for noise-on-delay analysis using the following command:

- [`setSIMode -deltaDelayThreshold value`](#)

By default, the Encounter software uses the same default delta delay threshold value that is used by the internal SI engine, which is 1ps.

## Virtual Attacker Mode

To efficiently model the many small attackers on a victim net, the SI analysis engine replaces them with an imaginary net, called a virtual attacker. The two main characteristics of a virtual attacker are: coupling capacitance, and the voltage source waveform used as a transition on the virtual attacker.

There are two methods to create and control the characteristics of the virtual attacker.

- **Coupling Capacitance-Based Method**

Matches the total coupling capacitance of a virtual attacker with the small attackers that are being virtualized.

- **Current Matching-Based Method**

Models the transitions on virtual attackers using piece-wise linear (PWL) waveforms. The PWL waveforms are generated in a way that the current induced by the transition matches the total current induced by all virtual attacker components.

Use the following command to setup the virtual attacker mode:

```
setSIMode -insCeltICPreTcl { set_virtual_attacker option1 option2 }
```

## Guidelines for Noise Analysis Settings

- When using a standalone tool for SI analysis such as Encounter Timing System, use the same noise analysis settings in Encounter for best correlation results.
- If you are using Encounter Timing System or any other third-party signoff solution, use the following setting:
  - `setSIMode -analysisType pessimistic`

**Note:** Although these settings will capture most of the noise violations in a pessimistic way, it is still recommended that you match the settings with the settings of your signoff tool.

## Static Timing Analysis (STA) Settings

Static timing analysis settings include the timing analysis engine and the analysis conditions.

### Input Timing Library

The primary input for timing analysis is a Liberty (.lib) library.

### STA Engine

Encounter uses the native timing analysis engine to perform static timing analysis. This engine is same as the one used by Encounter Timing System. You can choose to use the built-in timing analysis engine or a third-party tool for performing timing analysis.

## Analysis Conditions

The important analysis conditions for running SI analysis include:

- Setting Up the Analysis Mode

Encounter software provides different timing analysis modes and performs different calculations for setup and hold checks for each mode. For SI analysis, set the analysis mode to On-Chip Variation using the [`setAnalysisMode`](#) command:

```
setAnalysisMode -analysisType onChipVariation
```

OCV mode assumes that capture clock, launch clock, and data path can have maximum or minimum delays in setup and hold analysis. This is the recommended analysis mode for noise analysis.

**i** To specify the OCV analysis mode, you must use the multi-mode multi-corner (MMMC) framework regardless of whether your design is an MMMC design. For more information on setting up Multi-Mode Multi-Corner, see [\*Performing Multi-Mode Multi-Corner Timing Analysis and Optimization\*](#).

- Removing Common Path Pessimism

To remove the common path pessimism, use the following command:

```
setAnalysisMode -cppr true
```

The `-cppr` parameter removes pessimism from clock paths that have a portion of the clock network in common between the clock source and clock destination paths. The pessimism is introduced when the timing analysis tools assume that the common path has different delay values for two different paths in case of on-chip variation.

- Enabling Accurate CPPR Analysis When Incremental Delays are Present

To enable accurate CPPR analysis in the presence of incremental delays, set the following variable:

```
set_global timing_enable_si_cppr true
```

When this variable is set to `true`, the incremental delay is not included in the CPPR calculation during setup analysis, but is included in the CPPR calculation during hold analysis.

## Advanced Settings for SI Analysis

- Multi-CPU Processing Settings
- Path Group Settings for SI Optimization

### Multi-CPU Processing Settings

Encounter supports multi-threaded, distributed, and super-threaded noise analysis for MMMC designs, and multi-threaded and distributed noise analysis for non-MMMC designs. Multi-CPU processing support, in general, reduces the noise analysis run time significantly.

The following command considers the multi-CPU processing settings during noise analysis:

- `optDesign -postRoute -si`
- `timeDesign -postRoute -si`

 For information on multi-CPU processing, see [Accelerating the Design Process by Multiple-CPU Processing](#).

### Setting Up Multi-Threading for Noise Analysis

Multi-threading enables you to run noise analysis on multiple CPUs of a single host. The host machine could be the one on which you are running Encounter or a different host.

- To setup multi-threaded noise analysis for non-MMMC designs on the same host, use the following command:

```
setMultiCpuUsage -localCpu number
```

```
optDesign -postRoute -si
```

- 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 -si
```

### ***Setting Up Distributed Processing for Noise Analysis***

Distributed processing enables you to divide a noise analysis job between two or more networked computers running concurrently.

- To setup distributed processing using RSH, specify the following commands:

```
setDistributeHost -rsh -add { host1 host2 host3 host4 ... hostN }
```

```
setMultiCpuUsage -remoteHost number
```

```
optDesign -postRoute -si
```

- To setup distributed processing using LSF, specify the following commands:

```
setDistributeHost -lsf -queue myLSFqueue
```

```
setMultiCpuUsage -remoteHost number
```

```
optDesign -postRoute -si
```

**ⓘ When setting up distributed processing for MMMC designs, ensure that there is a corresponding host computer for each view definition. For example, if your MMMC design has four view definitions, divide these between four host computers.**

### ***Setting Up Super-Threaded Noise Analysis for MMMC Designs***

Super-threading enables you to run a noise analysis job on both distributed processing and multi-threaded modes; that is, two or more networked computers, each with multiple processors, can be called to concurrently run noise analysis. The super-threaded mode is supported for MMMC designs only.

Enable super-threaded noise analysis for MMMC designs:

- To setup super-threading using RSH, use the following commands:

```
setDistributeHost -rsh -add {host1 host2 host3 .... hostN}
```

```
setMultiCpuUsage -remoteHost number \
```

```
-cpuPerRemoteHost number
```

```
optDesign -postRoute -si
```

- To setup super-threading using LSF, use the following commands:

```
setDistributeHost -lsf -queue myLSFqueue
```

```
setMultiCpuUsage -remoteHost number \
```

```
-cpuPerRemoteHost number
```

```
optDesign -postRoute -si
```

#### ***Examples of Setting Up Multi-CPU Processing***

- The following distributed host settings distribute one view each on host1 and host2 machines for a two-view MMMC design using RSH:

```
setDistributeHost -rsh -add { host1 host2 }
```

```
setMultiCpuUsage -remoteHost 2
```

```
optDesign -postRoute -si
```

- The following multi-threading settings run each view (in eight threads) for a two-view MMMC design sequentially:

```
setMultiCpuUsage -localCpu 8
```

```
optDesign -postRoute -si
```

- The following settings distribute one view each on the specified host machines for a five-view MMMC design using RSH:

```
setDistributeHost -rsh -add { host1 host2 host3 host4 host5 }
```

```
setMultiCpuUsage -remoteHost 5 -localCpu 4
```

```
optDesign -postRoute -si
```

**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 intend to 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 one view each on `host1` and `host2` machines for a two-view MMMC design using RSH, and run each view divided between eight threads each:

```
setDistributeHost -rsh -add { host1 host2 }
```

```
setMultiCpuUsage -remoteHost 2 \
```

```
-cpuPerRemoteHost 8
```

```
optDesign -postRoute -si
```

**Note:** In this case, a total of 16 CPUs will be used.

## Path Group Settings for SI Optimization

The SI optimization flow accounts for the SDC path groups. To enable path groups, use the following command:

- [setAnalysisMode](#) `-honorClockDomains false`

When you specify this command, the software disables the use of standard and user-defined clock domains during optimization and uses path groups for timing and SI optimization instead. This enables you to take advantage of the path group effort level during SI optimization. In other words, optimization is run on path groups for which the effort level is set to high ([setPathGroupOptions](#) -

effortLevel high).

In addition, you can use:

- The -slackAdjustment parameter of the [setPathGroupOptions](#) command for the slack adjustment value.
- The -acceptableLNS parameter of the [setSIMode](#) command for the target slack value.

The path group settings provide you the flexibility to run optimization on selective paths that belong to critical parts of the design.

The following command accounts for path groups:

- [optDesign](#) -postroute -si

 To create, modify, and report path groups, use the following commands:

- [group\\_path](#)
- [reset\\_path\\_group](#)
- [report\\_path\\_groups](#)
- [createBasicPathGroups](#)
- [setPathGroupOptions](#)
- [resetPathGroupOptions](#)
- [reportPathGroupOptions](#)

## Example of Setting Up Encounter for SI Analysis

The following example provides a summary of the settings that are required for performing SI analysis:

```
## Extraction Settings ##  
  
setDesignMode -process 65
```

```
setExtractRCMode -engine postRoute -effortLevel high \
    -capFilterMode relAndCoup

## SI Settings ##

setSIMode -noiseTwfMode "-infSW" \
    -insCeltICPreTcl { set_virtual_attacker -gtol 0.025 -mode current; }

## STA settings ##

## Make sure MMMC/SMSC is used

setAnalysisMode -analysisType onChipVariation -cppr true

## CPPR Removal

set_global timing_enable_si_cppr true
```

## Preventing Crosstalk Violations

The following steps enable you to prevent crosstalk violations:

1. Place and optimize your design. Set reasonable max transition thresholds.
2. Fix transition time violations on the Clock tree aggressively.
3. Run timing and signal integrity driven routing using the [setNanoRouteMode](#) - routeSiEffort command.

**Note:** To enable the reduction of SI effects while fixing the base timing during post-route optimization, set the `-postRouteSiAware` parameter of the [setOptMode](#) command to `true`. When you set this

parameter to true, the software reduces the SI violations which results in faster timing closure at the signoff stage.

## Fixing Crosstalk Violations

- Data Preparation
- Using optDesign to Fix Setup Violations with Crosstalk Effects
- Using optDesign to Fix Hold Violations with Crosstalk Effects
- Using optDesign to Fix Transition Time Violations with Crosstalk Effects

### Data Preparation

In order to analyze and repair crosstalk violations properly, you must prepare your data correctly.

1. Read in signal integrity related data and libraries.
  - a. Generate the noise library.

Use the `make_cdb` utility to create the characterized noise library. For blocks, you can save a block-level cDB in Encounter if the block will be represented as a black-box on the timing side.

The `make_cdb` utility can be run in interactive mode or in batch mode. For information on how to use the `make_cdb` utility, see the *make\_cdB Noise Characterizer User Guide*.

1. Check that the data and libraries are correct and that there are no timing violations.
  - a. Check your congestion map and resolve any highly congested areas using the following command:

`congRepair`

1. Correct transition time violations.
2. Perform RC Extraction and Correlation

You can use native detailed extraction, Turbo QRC, Integrated QRC, or standalone QRC sign-off extraction to extract the routed design. To correlate native extraction results with QRC sign-off extraction, you compare SPEF files from basic and sign-off extraction to generate the new scaling factors for total capacitance, cross-coupling capacitance, and resistance. Using these scaling factors, the native extraction results are closer to the sign-off extraction results, while only taking a fraction of the run time required for sign-off extraction. For more information, see "[Correlating Native Extraction With Sign-Off Extraction](#)".

- Read in a placed and routed database.

```
restoreDesign your_design .dat your_topCell
```

## Using optDesign to Fix Setup Violations with Crosstalk Effects

Use the [optDesign](#) command with the `-postRoute` and `-si` parameters to correct glitch and setup violations caused by incremental delays, which occur due to coupling capacitance. The `optDesign` command fixes additional setup time violations caused by incremental delay up to a target slack equaling the worst negative slack without incremental delays. It is recommended that a design be optimized to the worst negative slack without incremental delays before performing SI fixing. The `optDesign` command avoids the degradation of the worst negative slack without incremental delays while fixing the setup time violations with incremental delays.

The setup time violations are fixed before glitch violations. By default, two iterations are done for SI fixing. However you can use the `-maxNrIter` parameter of the [setSIMode](#) command to specify a different number for the iterations to be performed.

- i** For best results, setup Encounter SI analysis to match SI sign-off analysis before using the `optDesign` command.

## Using RC Data Generated by an External Tool for SI Fixing

You can load the RC data generated by an external tool by using the [spefIn](#) command in Encounter to do SI fixing. When RC data from an external tool is used, SI fixing is limited to one iteration. RC data needs to be regenerated using the third-party tool to perform SI analysis and generate reports after fixing.

- i** For MMMC designs, ensure that you use the [spefIn](#) command to load the parasitic data for each corner.

## Using SDF Data Generated by an External Tool for SI Fixing

You can use SDF data generated by an external tool in Encounter to do limited SI fixing. When SDF data from an external tool is used, SI fixing is limited to one iteration of fixing. SDF data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

#### ***Fixing Setup Violations Using External SDF for Non-MMMC Designs***

For non-MMMC designs, the SDF file specified with the `-setupSdfFile` parameter of the [setOptMode](#) command is used for fixing setup violations.

- Use the following command to specify the SDF file for SI fixing in non-MMMC designs:  
`setOptMode -setupSdfFile filename`

The other requirements for using the external SDF file for SI setup fixing flow are:

- Specify the target slack using the `-acceptableWNS` parameter of the [setSIMode](#) command:  
`setSIMode -acceptableWNS value`  
**Note:** This is a mandatory setting.
- Enable the external SDF based SI fixing flow:  
`optDesign -postRoute -si -useSDF`

#### ***Fixing Setup Violations Using External SDF for MMC Design***

For MMC designs, use the [read\\_sdf](#) command to load an SDF file for each view.

- Use the following commands to load separate SDF files for two different views:  
`read_sdf -view viewname1 filename1`  
  
`read_sdf -view viewname2 filename2`
- Enable the external SDF based SI setup fixing flow:  
`optDesign -postRoute -si -useSDF`

### **Using optDesign to Fix Hold Violations with Crosstalk Effects**

Use the [optDesign](#) command with the `-postRoute`, `-hold`, and `-si` parameters to correct hold violations caused by incremental delays, which occur due to coupling capacitance. The `optDesign` command corrects additional hold violations caused by incremental delays up to a target slack equaling the hold worst negative slack without incremental delays. It is assumed that a design has been optimized to the best hold worst negative slack without incremental delays before performing SI fixing. The `optDesign` command does not degrade the hold worst negative slack (without incremental

delays) and setup worst negative slack (without incremental delays) while fixing the hold violations with incremental delays.

- By default, setup SI analysis is not performed. However, you can choose to enable setup SI analysis while performing SI hold fixing. In this case, the `optDesign` command will try not to degrade setup violations with incremental delays while fixing hold violations with incremental delays. To account for crosstalk setup timing information while performing SI hold fixing, use the following setting:

```
setSIMode -fixHoldIncludeXtalkSetup true
```

### Using RC Data Generated by an External Tool for SI Hold Fixing

You can load the RC data generated by an external tool by using the [spefIn](#) command in Encounter to do limited SI hold fixing. When RC data from an external tool is used, SI hold fixing is limited to one iteration. RC data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

- For MMMC designs, ensure that you use the [spefIn](#) command to load the parasitic data for each corner.

### Using SDF Data Generated by an External Tool for SI Hold Fixing

You can also use SDF data generated by an external tool in Encounter to do limited SI hold fixing. When SDF data from an external tool is used, SI hold fixing is limited to one iteration. SDF data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

#### ***Fixing Hold Violations Using External SDF for Non-MMMC Designs***

For non-MMMC designs, the SDF file specified with the `-holdSdfFile` parameter of the [setOptMode](#) command is used for SI hold fixing.

- Use the following command to specify the SDF file for both setup and hold fixing:

```
setOptMode -setupSdfFile filename -holdSdfFile filename
```

The other requirements for using the external SDF file for SI fixing flow are:

- Specify the target slack using the -acceptableWNS parameter of the [setSIMode](#) command:  
`setSIMode -acceptableWNS value`  
**Note:** This is a mandatory setting.
- Enable the external SDF based SI hold fixing flow:  
`#When you are performing setup and hold fixing in the same flow.`

```
setOptMode -setupSdfFile filename -holdSdfFile filename
```

```
optDesign -postRoute -si -hold -useSDF
```

### ***Fixing Hold Violations Using External SDF for MMMC Designs***

For MMMC designs, use the [read\\_sdf](#) command to load an SDF file for each view.

- Use the following commands to load separate SDF files for two different views:  
`read_sdf -view viewname1 filename1`  
  
`read_sdf -view viewname2 filename2`
- Enable the external SDF based SI fixing flow:  
`optDesign -postRoute -si -hold -useSDF`

## **Using optDesign to Fix Transition Time Violations with Crosstalk Effects**

You can now fix SI induced maximum transition violations in Encounter by using:

- A transition file generated during noise analysis
- External transition file(s) generated by third-party tools.

### **SI Transition Violation Fixing**

In the default flow, use the -fixDRC parameter of the [setSIMode](#) command to perform maximum transition violation fixing. After setting this parameter to true, run the `optDesign -postRoute -si` command to perform maximum transition violation fixing using transition information obtained from the transition file (`celtic.slew`) generated during noise analysis.

**Note:** The transition values will be used for maximum transition violation fixing only and will not affect the timing calculation in Encounter's timing engine. This is because the transition effect is considered by the internal SI engine during delay pushout calculation.

The following commands support the `-fixDRC` parameter of the [setSIMode](#) command, and report the transition violations in the Timing Summary report.

- [timeDesign](#) -postRoute -si
- [timeDesign](#) -signoff -si
- [timeDesign](#) -reportOnly -si

#### Transition Violation Fixing Using Transition File Generated During Noise Analysis Without Glitch or Noise-On-Delay Fixing

Use the `-drv` parameter of the [optDesign](#) command to perform DRV fixing only:

```
optDesign -postRoute -si -drv
```

In this flow, the software does not perform glitch and noise-on-delay fixing.

- !** The `-hold` parameter cannot be specified when using the `-drv` parameter with the `-si` parameter.

#### Transition Violation Fixing Using External Transition File(s)

Use the [readTransitionFile](#) command to read the external transition file before performing maximum transition violation fixing. The transition file should contain the transition values in the following format:

```
setTranTime [-add] [-maxR slewValue | -maxF slewValue ] {pinName }
```

- !** The transition values specified in the transition file must be in nanoseconds (ns).

Where:

<b>Option</b>	<b>Description</b>
-add	Specifies that the transition value is incremental.
-maxR   -maxF	Specifies the maximum rise or fall transition.
<i>pinName</i>	<p>Specifies the pin name for which the transition value is specified.</p> <p>To specify multiple pin names, use separate <code>setTranTime</code> statements as follows:</p> <pre>setTranTime -add -maxR 3.5 inst/A setTranTime -add -maxR 3.5 pinB</pre> <p><b>Note:</b> Minimum transition values will be ignored while performing transition violation fixing.</p>

- Specify the following command for reading an external transition file for non-MMMC designs:  
`readTransitionFile -file fileName`
- Specify the following command for reading an external transition file for MMC designs:  
`readTransitionFile -view viewName -file fileName`  
**Note:** Ensure that you specify the transition files for all setup views.

After you read in the transition file, specify the following command to fix the maximum transition violations:

```
optDesign -postRoute -si -useTransitionFiles -drv
```

To fix the maximum transition violations for selected nets, use the `-selectedNets` parameter of the [optDesign](#) command.

**Note:** In this flow, the software will exit after the `ecoRoute` command.

- Use the following command to report the maximum transition violations when specifying external transition files:

```
timeDesign -postRoute -si -useTransitionFiles
```

### External Transition and SDF Files Used for Transition Violation and Delay Fixing

To perform maximum transition violation fixing along with delay pushout fixing, you will need to specify the external transition file as well as the external SDF file.

- Use the following commands for non-MMMC designs:

```
setOptMode -setupSdfFile fileName
```

```
setSIMode -acceptableWNS value #This is a mandatory setting.
```

```
readTransitionFile -file fileName
```

```
optDesign -postRoute -si -useSDF -useTransitionFiles
```

**Note:** To perform transition violation fixing using an external transition file, use the -selectedNets parameter of the [optDesign](#) command. This parameter works with the -useTransitionFiles parameter but not with the -useSDF parameter.

- Use the following commands for MMC designs:

```
read_sdf -view viewName1 fileName1
```

```
read_sdf -view viewName2 fileName2
```

```
readTransitionFile -view viewName1 -file fileName1
```

```
readTransitionFile -view viewName2 -file fileName2
```

```
setSIMode -acceptableWNS value #This is a mandatory setting.
```

```
optDesign -postRoute -si -useSDF -useTransitionFiles
```

**Note:** To perform maximum transition violation fixing on selected nets, use the -selectedNets parameter of the [optDesign](#) command. This parameter is valid with the external

transition file flow only.

## Performing XILM-Based SI Analysis and Fixing

The model-based flow in Encounter involves generating timing accurate interface logic models (ILMs) for hierarchical blocks. To perform SI analysis, the model data generation process should account for the characterized noise library of the blocks, the timing window information of non-ILM timing path nets inside the blocks, and cross-coupling information. This data, which is an extension to ILMs, is called eXtended Interface Logic Model (XILM). An XILM is built with:

- Cells and RC network (including cross-coupling data) connected from each I/O pin to and from the first latch or flip-flop.
- eXtended Timing Window Format (XTWF) file that contains timing window and slew information of non-ILM timing paths inside the block, which might be aggressors to the nets on the ILM timing path or the top-level nets.

**Note:** For more information on XILM-based SI analysis, see the [Using Interface Logic Models in Hierarchical Designs](#) chapter of the *Encounter User Guide*.

---

# Power and Rail Analysis

---

- [Overview](#)
- [Early Rail Analysis](#)
  - [Early Rail Analysis Key Features](#)
  - [Prior to Running Early Rail Analysis](#)
  - [Setting up and Running Early Rail Analysis](#)
  - [CPF Support for ERA](#)
  - [Running Early Rail Analysis in Unplaced Mode](#)
  - [Viewing Early Rail Analysis Results](#)
- [Signoff-Rail Analysis](#)
- [Electrostatic Discharge Analysis](#)
- [EDI System and EPS menu differences](#)

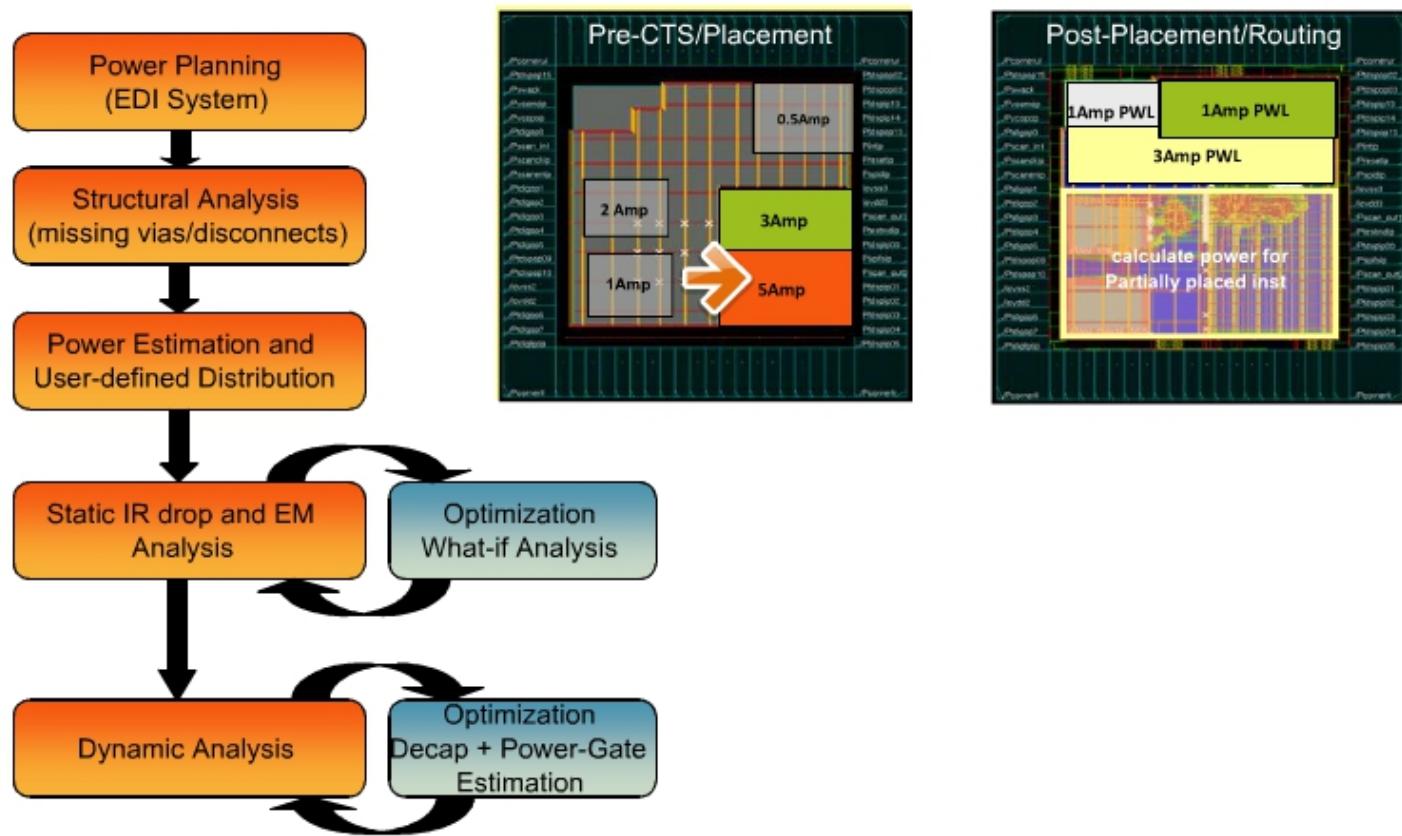
## Overview

Encounter Power System (EPS) sign-off power and rail analysis engines are fully integrated in Encounter Digital Implementation System (EDI System). The TCL and GUI use-models are identical in stand-alone EPS and its integration in EDI System, allowing a smooth transition from early power planning to sign-off power analysis. The power and rail analysis in EDI System is available under the *Power* menu (EPS is under *Power & Rail* menu). These menus are arranged differently between the two products (See [EDI System and EPS menu differences](#) ).

The Early Rail Analysis (ERA) feature is available through EDI System and is not available in EPS. ERA provides quick rail analysis with the same use model as EPS, without requiring an EPS license. ERA is not intended for sign-off; if you want to include power-grid views for signoff, then an EPS license is required. ERA is instead intended for early analysis, and can run on a power-grid floorplan before placement and routing.

## Early Rail Analysis

Early Rail Analysis (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 power planning and ERA flow:



The following steps describe the ERA flow:

- **Structural Analysis:** Check if there are any missing vias or disconnects. Prior to running ERA, you must run the verifyGeometry, verifyConnectivity, and verifyPowerVia commands to ensure that there are no power rail shorts, floating power nets, and missing or open power vias.
- **Power Estimation and User-Defined Distribution:** Identify an area and assign a power value, or assign a power value to placed components, such as RAMs and ROMs.
- **Static IRdrop and EM Analysis:** Run static IRdrop and electromigration analysis. You can perform What-if Analysis to make changes to the design within the environment before implementing them in layout. What-if Analysis allows you to run various scenarios, such as increase or decrease current, to determine what changes you should make to alleviate IRdrop problems.
- **Dynamic Analysis:** Run dynamic IRdrop analysis. You can check whether you need to optimize the power-grid for decaps and power gates.

ERA uses EPS power and rail analysis engines and can be used during power-grid prototyping to analyze power, IRdrop and power-grid integrity. This section includes:

- Early Rail Analysis Key Features

- Prior to Running Early Rail Analysis
- Setting up and Running Early Rail Analysis
- Running Early Rail Analysis in Unplaced Mode
- Viewing Early Rail Analysis Results

## Early Rail Analysis Key Features

- Static and Dynamic Rail analysis during the floorplanning stage using grid based interactive current specification use-model. Dynamic Rail Analysis requires the EPS-XL license. Static Rail Analysis can be run using the EDI license.
- Power and rail analysis during the placement stage using virtual power-grid connectivity to the instance power pin
- Power and rail analysis on a placed and routed database
- No requirement for extraction tech file or power-grid cell libraries. Optionally, power-grid libraries can be used for the placed macros to achieve better accuracy. Usage of power-grid libraries (.cl extension) requires EPS licenses.
- Flexible power-constraints specification including:
  - Total power and current regions
  - Calculating power for placed instances and specify current regions for unplaced regions
  - Instance current and current region files for macros
- Package and pad location optimization based on IRdrop analysis
- Early power-switch analysis to refine power-switch placement
  - Support for static instance power file in the ASCII format
  - Support for power-grid electro-migration analysis
  - Support for power-grid optimization guidance using sensitivity analysis
  - Support for net-based and domain-based rail analysis
  - Decap analysis and optimization during dynamic rail analysis
  - Support for multi-CPU in static 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 of the *Encounter® Power System User Guide* .
  - N-port die-model generation during dynamic rail analysis. The early die-model for the chip can be used with the package to perform resonance and impedance analysis of the

system.

- Support for the peak current mode for the entire design during dynamic rail analysis
- Support for the unplaced flow during static and dynamic analysis
- What-if analysis to guide power-grid optimization
- Support for native power-up analysis

## Prior to Running Early Rail Analysis

Prior to running ERA, it is recommended to check the power rail and vias for problems. This can be done with the [verifyGeometry](#), [verifyConnectivity](#), and [verifyPowerVia](#) commands described in the EDI System Text Command Reference. You can find additional information in the Design Sanity Checks chapter of the Encounter Power System User Guide.

Following is an example of verifyPowerVia:

```
***** Start: VERIFY POWER VIA *****
Start Time: Mon Aug 6 21:41:07 2012

Check all 623 Power/Ground nets
**** 21:41:07 **** Processed 50 nets (Total 623)
**** 21:41:07 **** Processed 100 nets (Total 623)
**** 21:41:07 **** Processed 150 nets (Total 623)
**** 21:41:07 **** Processed 200 nets (Total 623)
**** 21:41:07 **** Processed 250 nets (Total 623)
**** 21:41:07 **** Processed 300 nets (Total 623)
**** 21:41:07 **** Processed 350 nets (Total 623)
**** 21:41:07 **** Processed 400 nets (Total 623)
**** 21:41:07 **** Processed 450 nets (Total 623)
**** 21:41:07 **** Processed 500 nets (Total 623)
**** 21:41:07 **** Processed 550 nets (Total 623)
**** 21:41:07 **** Processed 600 nets (Total 623)
```

Found missing via(s) on net VDD.

Found missing via(s) on net VSS.

Begin Summary

7 Problem(s) Found between Layers: M3 and M4.

7 total info(s) created.

End Summary

End Time: Mon Aug 6 21:41:07 2012

\*\*\*\*\* End: VERIFY POWER VIA \*\*\*\*\*

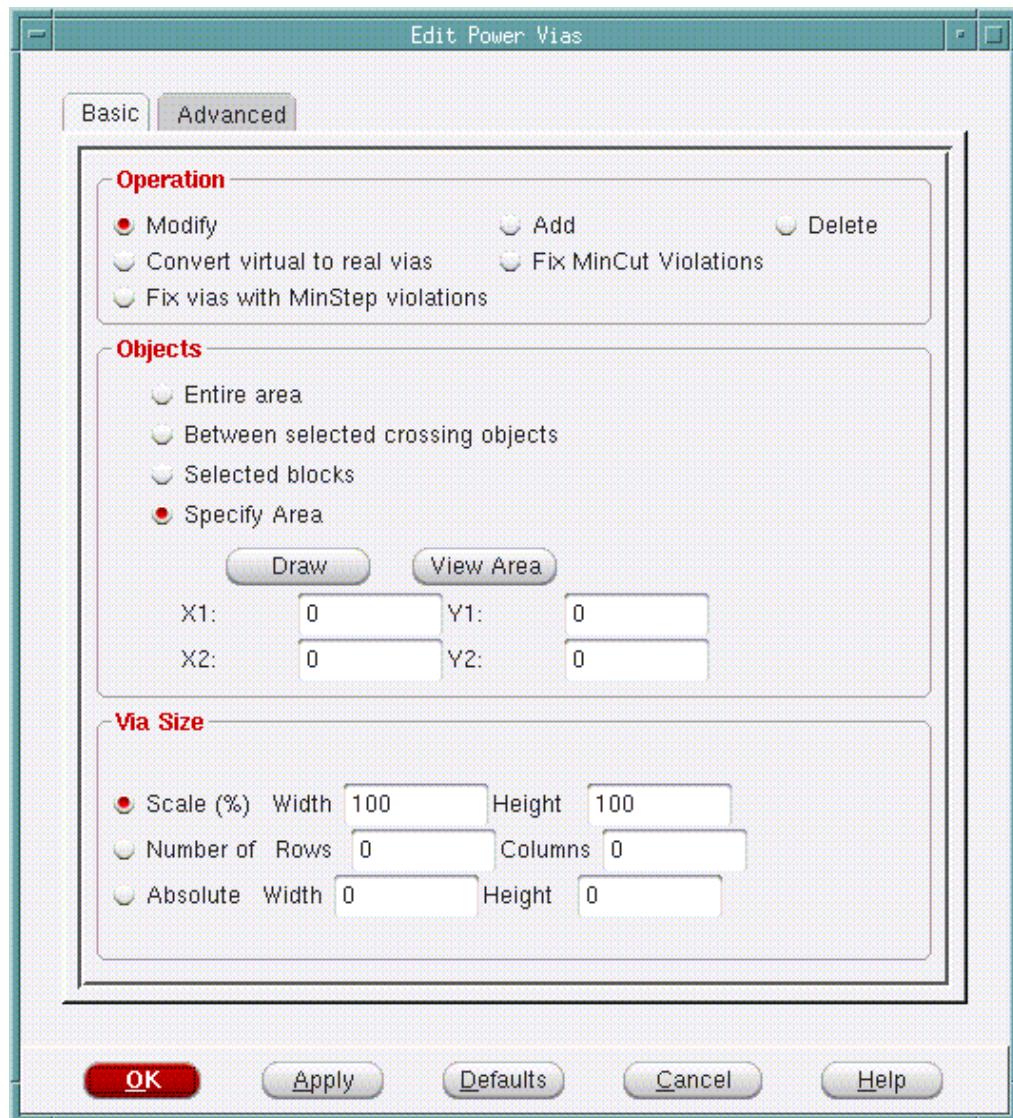
Verification Complete : 7 Viols. 0 Wrngs.

(CPU Time: 0:00:00.6 MEM: 2.020M)

If there are missing power vias, you can use the *Power - Power Planning - Edit Power Vias* form shown in Figure 37-1 to correct the problem. This form can be used for adding, removing, and sizing the power vias.

The design must be loaded into EDI System.

**Figure 37-1 Edit Power Vias**



## Setting up and Running Early Rail Analysis

After the design is loaded, use the following steps to set up and run Early Rail Analysis:

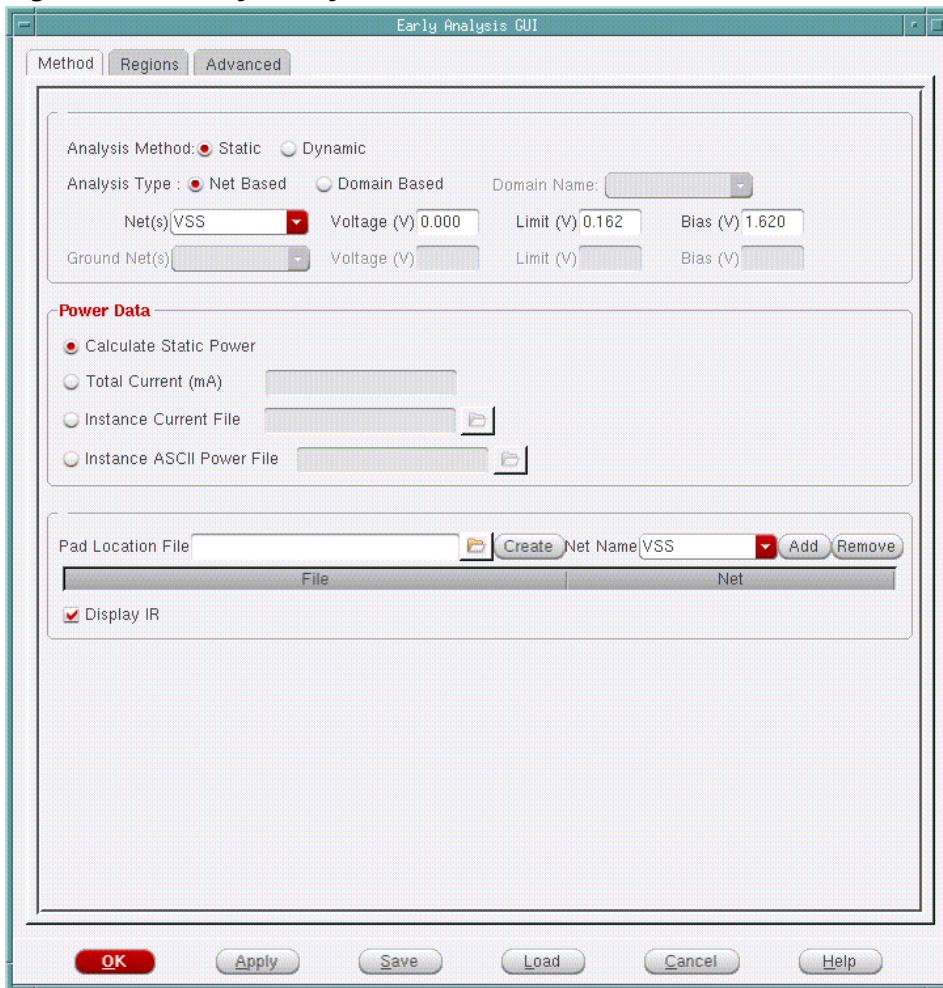
1. Select the *Power -> Rail Analysis -> Early Rail Analysis* menu item.

The form shown in Figure 37-2 appears with the tab set to *Method* by default.

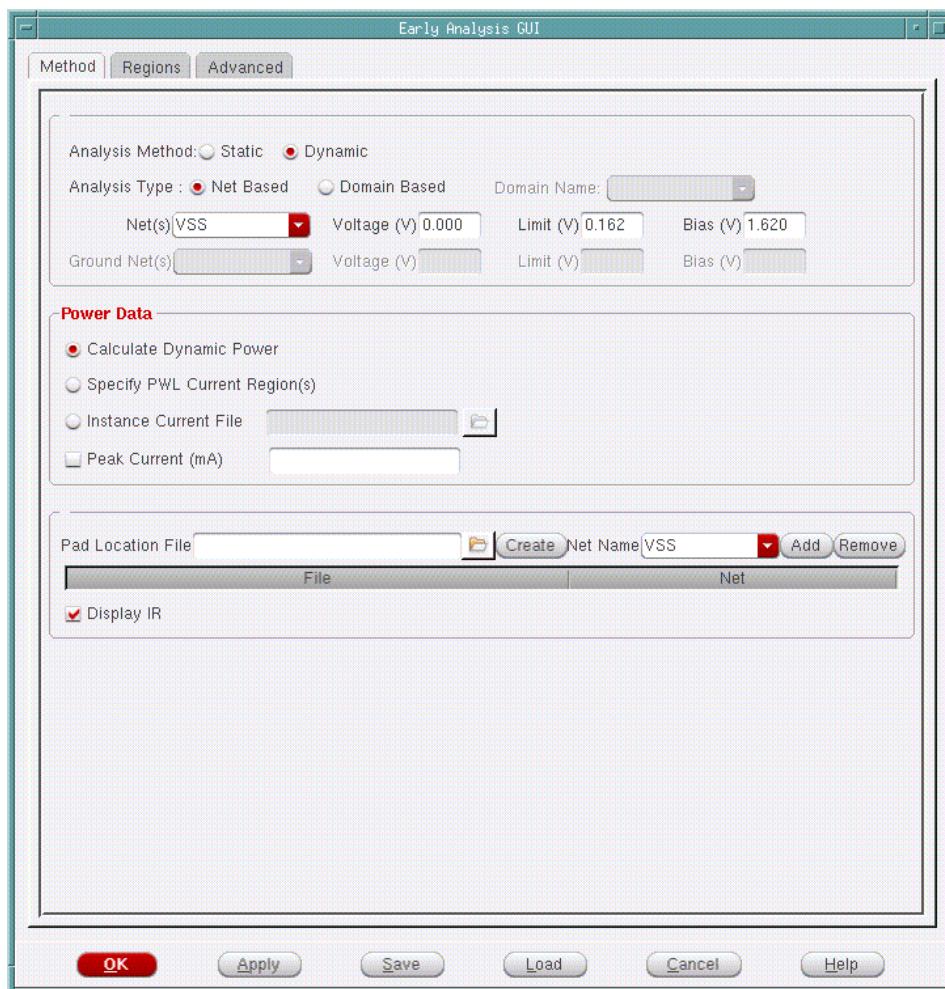
2. Select the *Static* or *Dynamic* method, and specify the *Net Based* or *Domain Based* analysis type.

In domain-based analysis, the power and ground nets in the domain are analyzed together. While in static analysis, the IR drop results do not change whether you choose *Net Based* or *Domain Based* analysis. In dynamic analysis, the domain-based analysis uses coupled capacitance between power and ground net which can give better accuracy as compared to net-based analysis which uses decoupled capacitance.

**Figure 37-2 Early Analysis - Static Method Form**



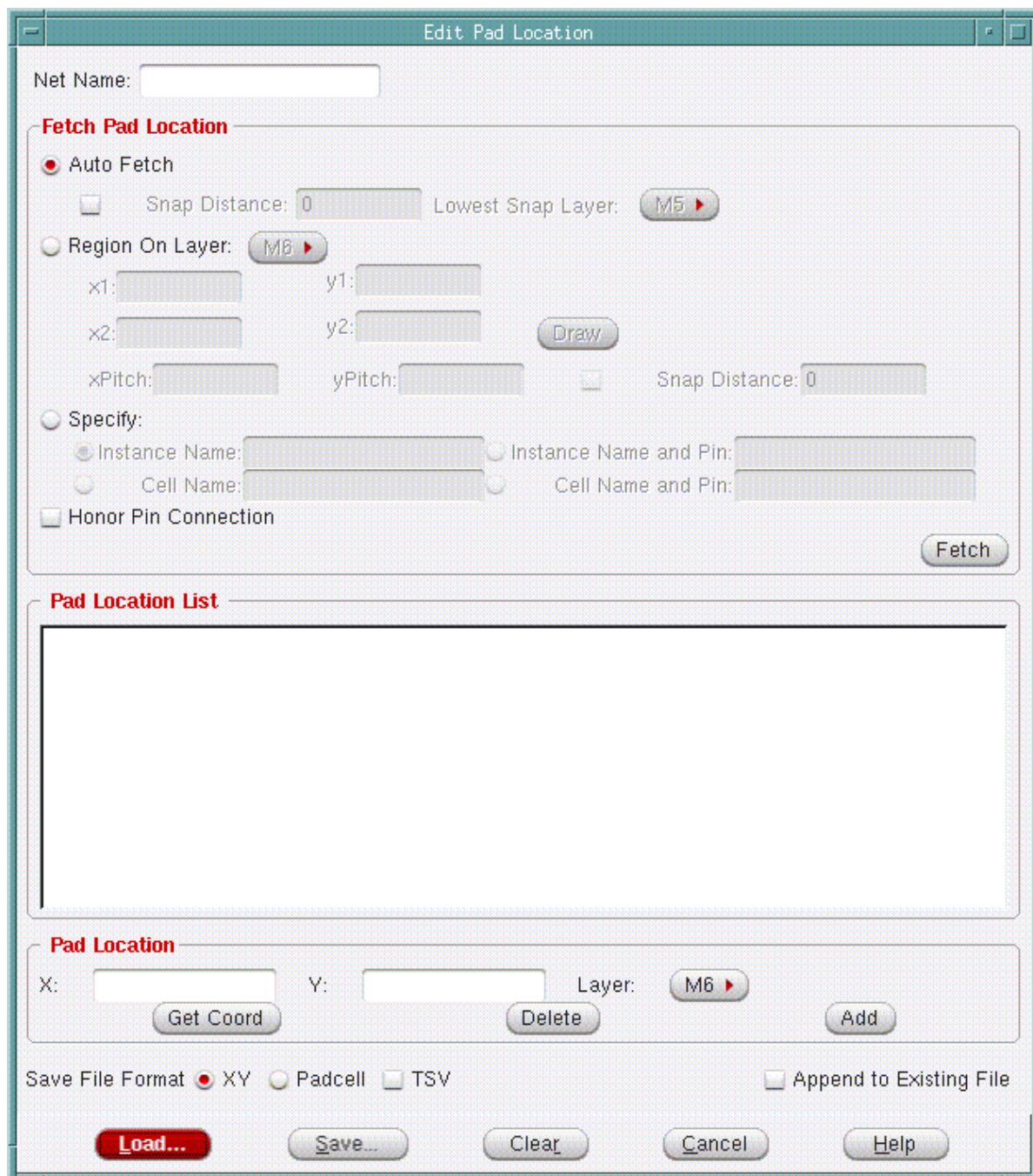
**Figure 37-3 Early Analysis - Dynamic Method Form**



**Note:** The content of the form changes depending on the selection of the analysis method.

3. Next, the locations of the power rail voltage sources (VDD, VSS) need to be specified and placed in a text file. This can be done automatically by clicking the *Create* button. The Edit Pad Location form (Figure 37-4) appears.

**Figure 37-4 Edit Pad Location Form**



You can fetch voltage sources using one of the following three methods:

- Select **Auto Fetch** - fetch all voltage sources of a power or ground net. In this method, specify one of the power or ground net names, such as VSS, in the Net Name field, and click the Fetch button. The VSS Pad Location List will automatically be filled in with location and layer values based on the VSS pins in the DEF. Repeat the process for all of the other power and ground sources that you are interested in analyzing.
- Select **Region** - fetch the voltage sources within the specified region of the specified

layer controlled by the specified xPitch/yPitch. In this method, specify the rectangular region ( $x1\ y1\ x2\ y2$ ) that the current will be distributed within. A grid is created by using the specified xPitch and yPitch within the region. The current will be specified at each of the intersecting pitch lines. Select *Layer* to specify the metal layer that the current sink will be placed on. If you click *Draw*, and then select a box in the main window, the coordinate of this box will be automatically populated in the  $x1\ y1\ x2\ y2$ . Click the *Fetch* button. The *Pad Location List* will automatically be filled in with location and layer values.

 For region-based auto fetch, if the voltage source points that were fetched according to the start point and the specified region pitch (xPitch/yPitch) are not on the stripe, these points will be ignored during analysis. You can use the *Snap Distance* option to snap the voltage sources to a stripe within +/- snap distance/2. If no such stripe is found, this point will not be saved into the pad location file. As a result, the voltage sources are generated only on the stripes and these points will be saved to the pad location file.

- Select **Specify** - fetch voltage source for the specified cell/cell pin/instance/instance pin. Click the *Fetch* button. The *Pad Location List* will automatically be filled in with location and layer values.

You can save the *Pad Location List* to a file. This file can be saved in either the XY (xy coordinates), or *Padcell* (VoltageStorm), or *TSV* format. After specifying the file format, click *Save*.

 Optionally, you can select *Honor Pin Wire Connection* to honor pin wire connection when fetching voltage source location from a power pad. When you select this option, only pins with wire connection are output as voltage sources. This option is disabled for region-based auto fetch.

4. In Figure 37-2, select the *Net Name* that you want to analyze from the pull-down menu.

The *Voltage*, *Limit*, and *Bias* fields will be automatically filled in.

- *Voltage* - the voltage of the power or ground net that you will be analyzing.
- *Limit* - the IR drop constraint for analysis. All power-grid elements operating below this limit (or above if it is a ground net) will be flagged as a violation and can be seen in the violation browser.
- *Bias* - the bias voltage (supply range) for power and ground nets. This voltage is used to compute current based on power specification. For the power net, the net voltage is used by default. For the ground net, the rail analysis does not know how to convert the specified power into current or at what voltage power was computed. The bias voltage will be needed to be specified in this case.

**Note:** Bias is only used when you specify *Instance ASCII Power File*. It is not used if

*Calculate Static Power* is selected.

5. Next, specify the method you will be using for defining Power (Current) information:

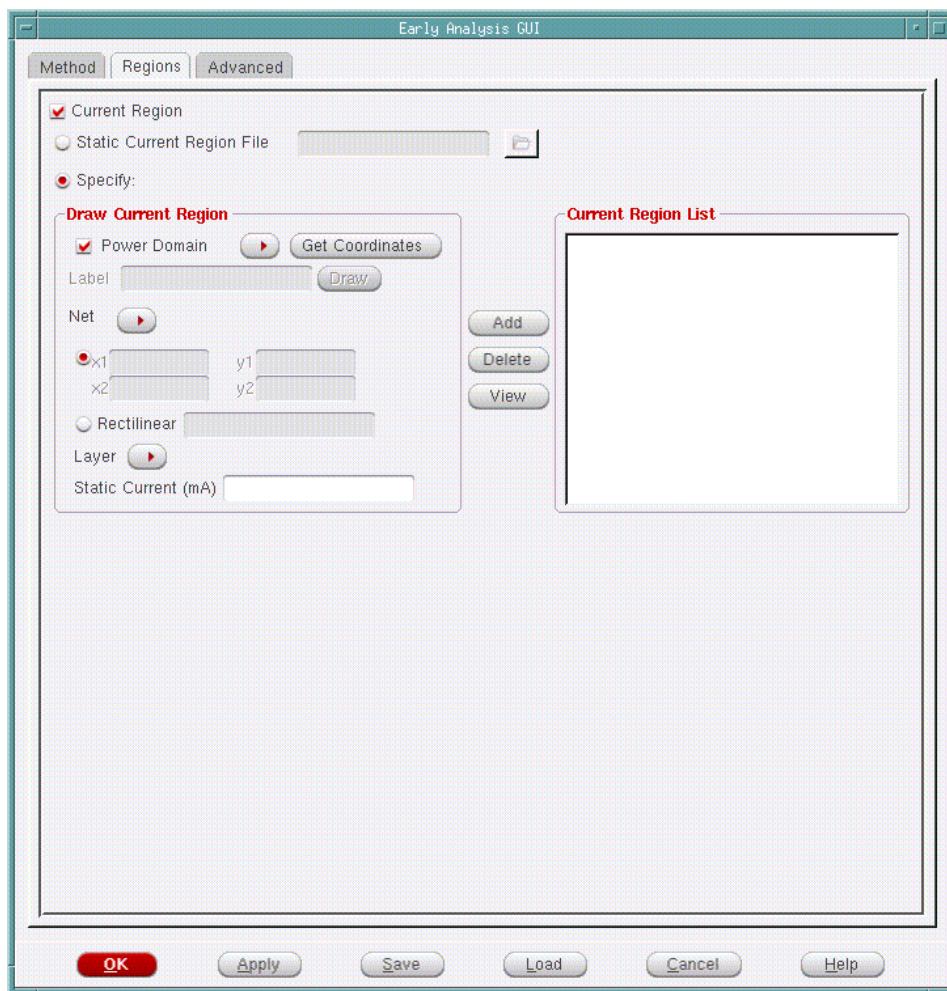
- *Calculate Static Power or Calculate Dynamic Power* - Select this if you want the power calculator to calculate the power for each instance in the design. This requires that your design is placed.
- *Total Current/ Specify PWL Current Regions* - If your design is not placed, you can specify a total static current in mA or PWL dynamic current regions for the design.
- *Instance Current File* - You can also specify an instance current file from a previous power run. The power calculator generates binary static and dynamic current files for each power net with the extension .ptiavg.
- *Instance ASCII Power File* - Optionally, you can also supply an external instance ASCII power file during static rail analysis. This file must have a two-column format with instance name and power in Watt. This option only applies to static rail analysis.
- *Peak Current* - Optionally, you can also supply a single peak current value in mA for the entire design. This option only applies to dynamic rail analysis.

6. Next, specify the *Pad Location File* that was created earlier, select the net name from the pull-down menu, and click *Add*.

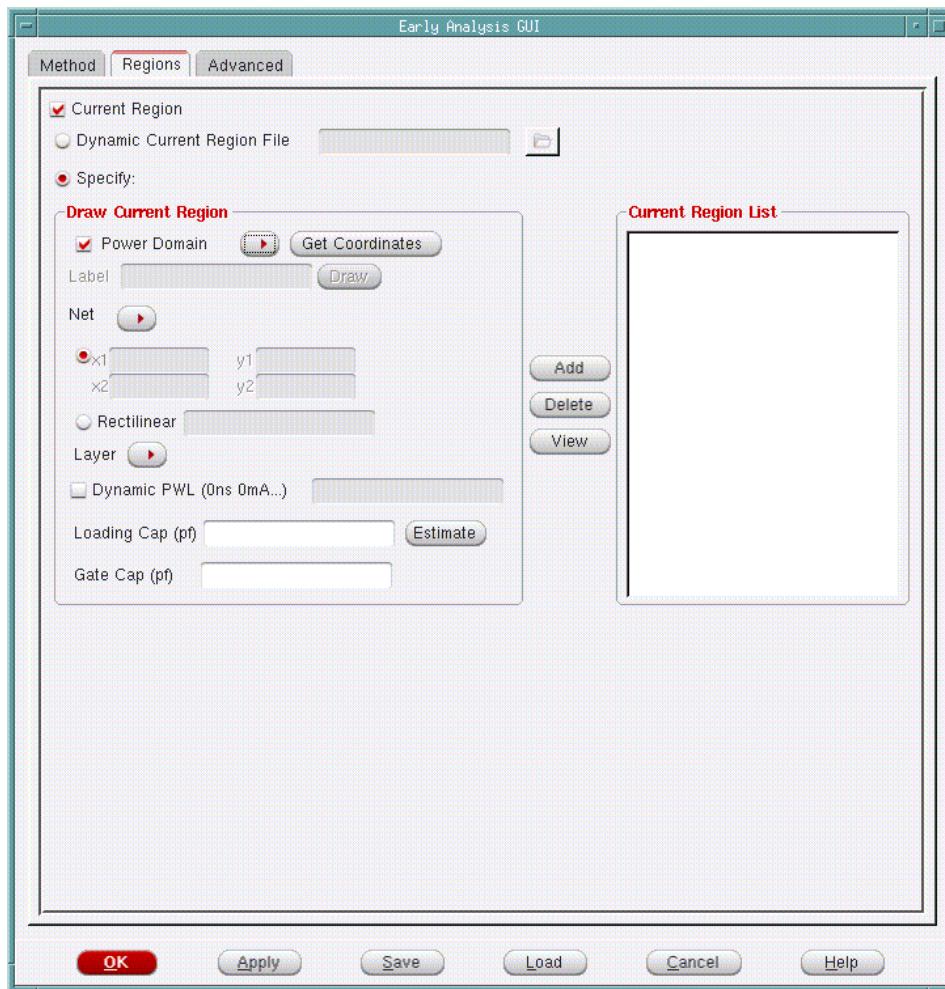
7. If you select *Display IR*, then at the end of early rail analysis, the ir (voltage) results display will automatically be loaded in the layout window.

8. You can select the *Regions* tab as shown in Figure 37-5.

**Figure 37-5 Early Analysis - Static Regions Form**



**Figure 37-6 Early Analysis - Dynamic Regions Form**



9. Selecting *Current Region* field indicates that you want to use current regions instead of calculated power. You can:

- Point to *Static Current Region File* or *Dynamic Current Region File* - The file format is as follows:

```
#####
#Format: LABEL name NET netName AREA x1 y1 x2 y2 LAYER layername <CURRENT
value | PWL (t1 i1 t2 i2 ...) > INTRINSIC_CAP value LOADING_CAP value
#Unit: current mA, cap pf, time ns, coordinate um
#####
```

Following is an example of a dynamic current region file:

```
label1 REGION1 net vdd area 44.932 643.848 613.073 1252.7025 layer M1 pw1 (0
0 1 0 1.9 0 2 5 3 0 4 0 5 0) intrinsic_cap 7.38504 loading_cap 49.2336
```

You can also specify a power domain based current region in the current region file by specifying the power domain name without specifying the boundary box coordinates:

####Format of the current region file:

```
label <power_domain_name> net <net_name> area "" layer
<layer_name> <current value | PWL (t1 i1 t2 i2 ...)>
INTRINSIC_CAP <value> LOADING_CAP <value>
```

- *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, EDI System 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 in EDI System.
- *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 in EDI System, 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.

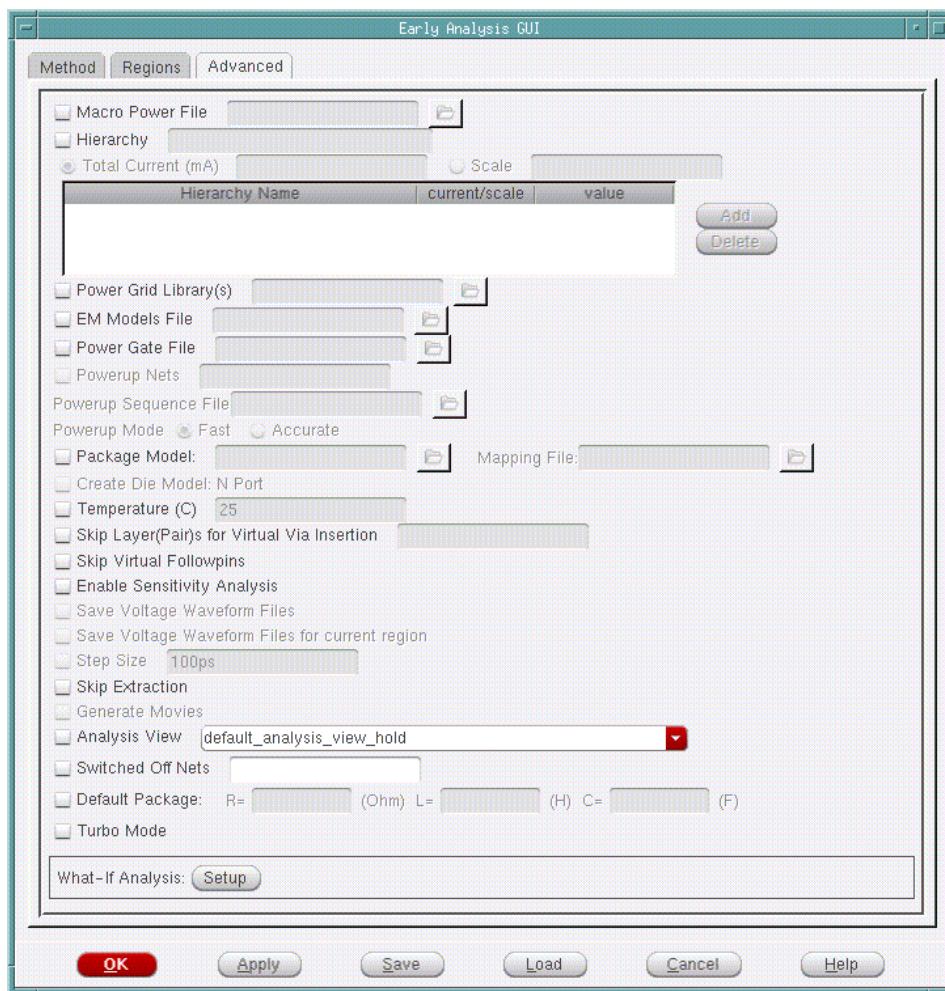
10. Click the *Add* button to add the region to the *Current Region List*.

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.

**Note:** The tool will connect to the region through a "virtual" power connection to the power pin, since there is no power routing.

11. You can specify the advanced early analysis features using the *Advanced* tab (Figure 37-7).

#### **Figure 37-7 Early Rail Analysis - Advanced Form**



12. You can specify one or more of the following advanced options:

- **Macro Power File** - specifies a text file with a list of macros and the power (current) that they consume. This is only available in static rail analysis.  
An example of a macro file is shown below:  

```
RAM1 0 .1
RAM2 0 .2
INVX1 0 .001
```
- **Hierarchy** - specifies power (Currents) or scales power for design hierarchy.
- **Power Grid Library** - specifies a power-grid library for macros to view IRdrop inside macros and its impact on global grid. This is for characterized cells that have been placed.
- **EM Models File** - the electro-migration analysis is only supported in static rail analysis, which includes:
  - Support for LEF layer names in the EM model with/without power-grid libraries
  - Support for technology layer names in the EM model only with power-grid

libraries

For information about EM models, see "[Static IRDrop and EM Analysis Overview](#)" in the *Encounter® Power System User Guide*.

- *Power Gate File* - specifies 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 *Encounter® Power System User Guide*.

- *Powerup Nets* - specifies a list of nets to be powered up during native power-up analysis.
- *Powerup Sequence File* - specifies the power-up sequence file containing the firing time for each power gate instance. The power-up sequence file format is as follows:  

```
<power gate instance name> <pin name> <always-on net> <switched net>
<rise_time> <rise_slew> <fall_time> <fall_slew>
```
- *Powerup Mode* - specifies the accuracy mode for native power-up analysis.
- *Package Model and Mapping File* - The package model is in the SPICE format and contains R,L,C and K parameters for package wires. The mapping file maps package terminals to chip power pad terminals. For information about the file format and analysis, see "[Package Analysis](#)" in the *Encounter® Power System User Guide*.
- *Create Die Model* - If package information is provided during dynamic rail analysis, you can generate the n-port die-model.
- *Temperature* - specifies the analysis temperature.
- *Skip Layer for Virtual Via Insertion* - specifies whether to skip via insertion between stripes and non-stripes, on the specified LEF layer pairs.
- *Skip Virtual Followpins* - specifies whether to skip generation of virtual followpins.
- *Enable Sensitivity Analysis* - highlights power-grid segments that can be optimized to

achieve better overall IRdrop profile for the design. The sensitivity analysis plot is available through the `view_analysis_results` command under plot "rs". The rs analysis plot highlights segments with high sensitivity that should be optimized to fix worst-case IRdrop. This option only applies to static rail analysis.

- *Save Voltage Waveforms Files* - saves instance-based voltage waveform files in the state directory of analyzed net(s) to feed to SPICE critical path analysis and Substrate Noise Analysis. You can select this option only in the dynamic mode.
- *Save Voltage Waveforms Files for current region* - saves voltage waveforms for current regions. You can use this parameter to output the voltage waveforms at the current region "super-nodes" to the `vstorm2.tran.ptiavg` file in the analysis output directory. You can select this option only in the dynamic mode.
- *Step Size* - specifies the transient time step size for a current waveform. You can specify this parameter only in the dynamic mode.
- *Skip Extraction* - skips the steps in constructing an R network (PGDB), while the software continues to generate other files for early rail analysis.
- *Generate Movies* - generates dynamic IRdrop and tap current movies. You can specify this parameter only in the dynamic mode.
- *Analysis View* - if you do not set this option, early rail analysis uses the default analysis view. Only views specified with the `set_analysis_view` and `set_default_view` commands are honored.
- *Default Package* - specifies the default resistance ( $\text{ohm}$ )-inductance ( $\text{Henry}$ )-capacitance ( $\text{Farad}$ ) in ERA to pass the default RLC values to VoltageStorm.
- *What-if Analysis* - performs what-if analysis to guide power-grid optimization. This analysis is used primarily to change electrical parameters of the power-grid to estimate eventual optimization efforts to satisfy IRdrop and ElectroMigration limits. The following power-grid what-if analysis features are available in ERA:

- Scaling Resistance of the Power-Grid
- Scaling Static and Dynamic Currents
- Scaling Capacitance

What-if analysis is also available through the following TCL command parameters:

- `-scale_what_if_resistance fileName`
- `-scale_what_if_capacitance fileName`
- `-scale_what_if_current fileName`

See [`analyze\_early\_rail`](#) command for details on these and other options. For

information about what-if analysis, see "[What-If Rail Analysis](#)" in the *Encounter® Power System User Guide*.

13. Click *OK* or *Apply* and the early rail analysis will be run.

Upon successful completion of the analysis, the Power & Rail Results (Figure 37-8) 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.

## CPF Support for ERA

ERA supports the Common Power Format (CPF) that can be used to ease adoption of low-power and power-efficient designs. You can import CPF-based designs to perform static and dynamic power and IRdrop analysis. In the CPF based ERA flow, a CPF file is used which defines timing analysis views based on a combination of different operating corners, power modes, and timing library set along with capturing the design and technology-related power constraints. When CPF is loaded, you can select a view to drive both power and rail analysis of a power domain in ERA.

The following types of ERA flows support CPF: Domain-based and Net-based.

- When setting up and running net-based ERA using the *Early Rail Analysis - Method* form, you do not need to specify power nets and voltage of those nets as this information is obtained from the loaded CPF. Therefore, the power net and its voltage/limit/bias are displayed automatically in the respective fields.
- In case of domain-based ERA, you can use the *Domain Name* drop-down list to display power domains in the CPF. If you select a domain, the power/ground nets and their voltage/limit/bias in the power domain are displayed automatically in the respective fields.

You must specify an analysis view in the *Early Rail Analysis - Advanced* form to incrementally add the associated switched off nets of the domain that is shut off in the *Switched Off Nets* field. Selection of a view and power domain is essential for extraction of correct nets associated with the domain.

## Running Early Rail Analysis in Unplaced Mode

You can run ERA in unplaced mode, i.e., run the ERA flow without any placed instances. ERA now allows an empty netlist (with no instance inside) with the unplaced flow.

To run ERA in unplaced flow:

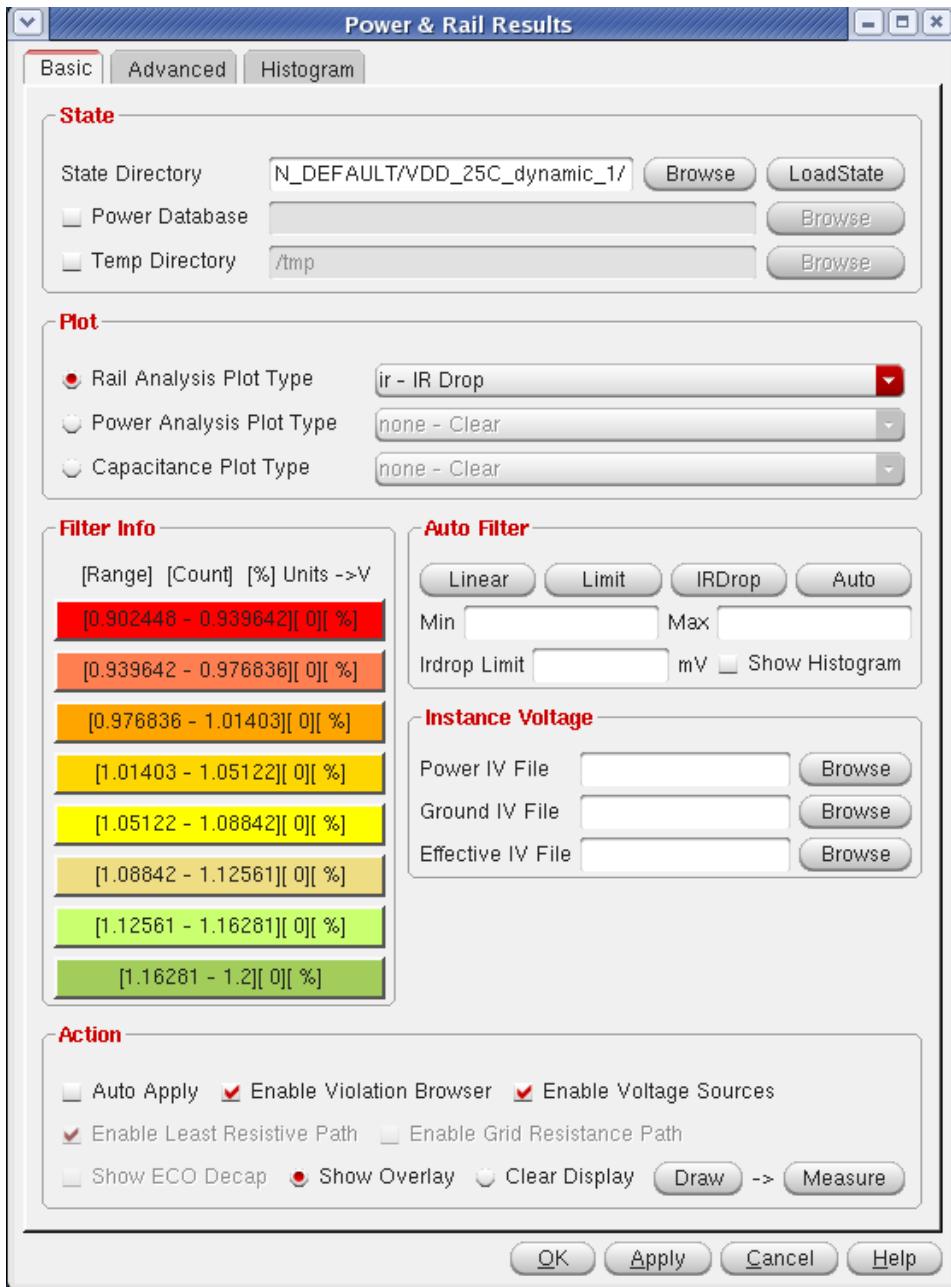
- **static analysis** - specify `-total_current` and `-current_region_file` (specify a current region file that covers the whole design)
- **dynamic analysis** - specify `-current_region_file` (specify a current region file that covers the whole design)

The `-instance_current_file`, `-instance_ascii_power_file`, `- calculate_power`, and `-macro_power_file` parameters are not allowed in an unplaced flow. Also, the pad cell file format is not allowed for the pad location file in an unplaced flow, as it does not have location information for unplaced IO pads.

## **Viewing Early Rail Analysis Results**

Selecting *Power - Report - Power & Rail Result* menu item will bring up the following form:

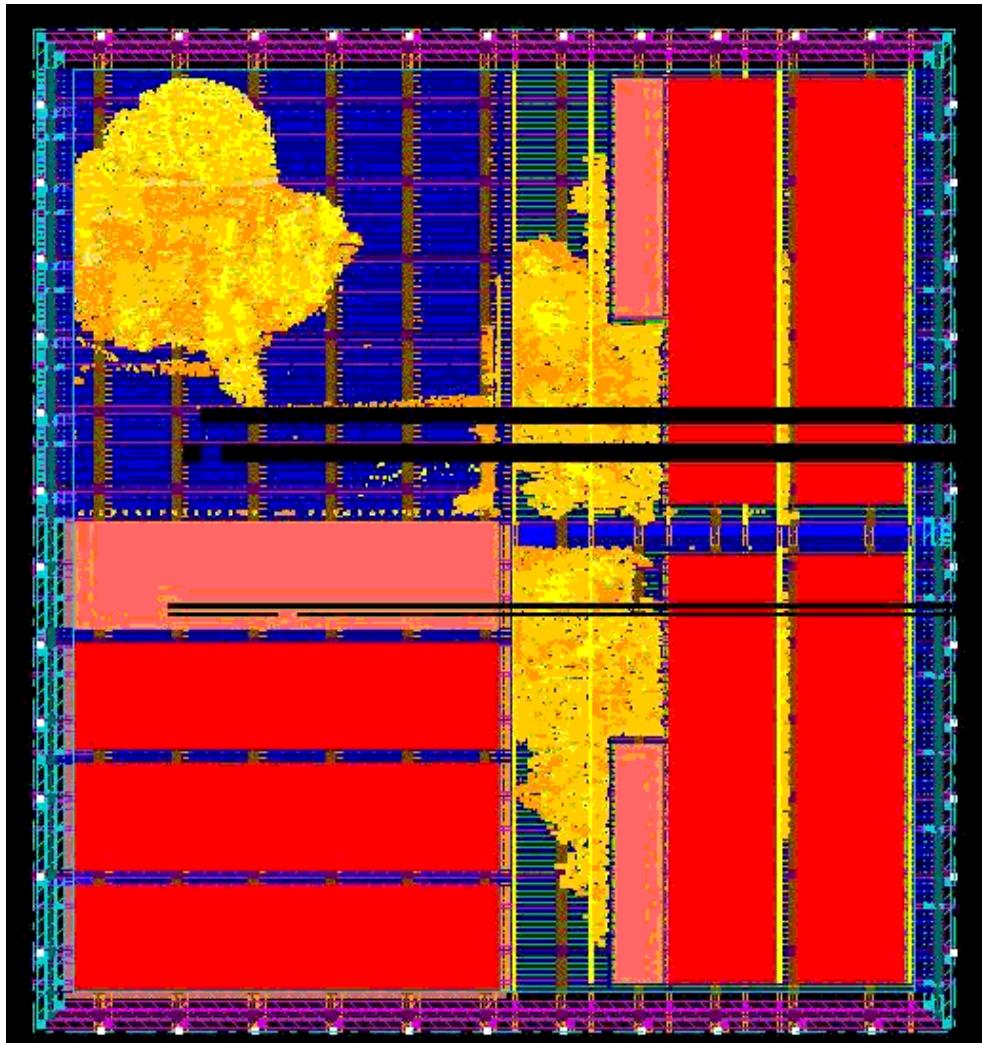
**Figure 37-8 Power & Rail Results**



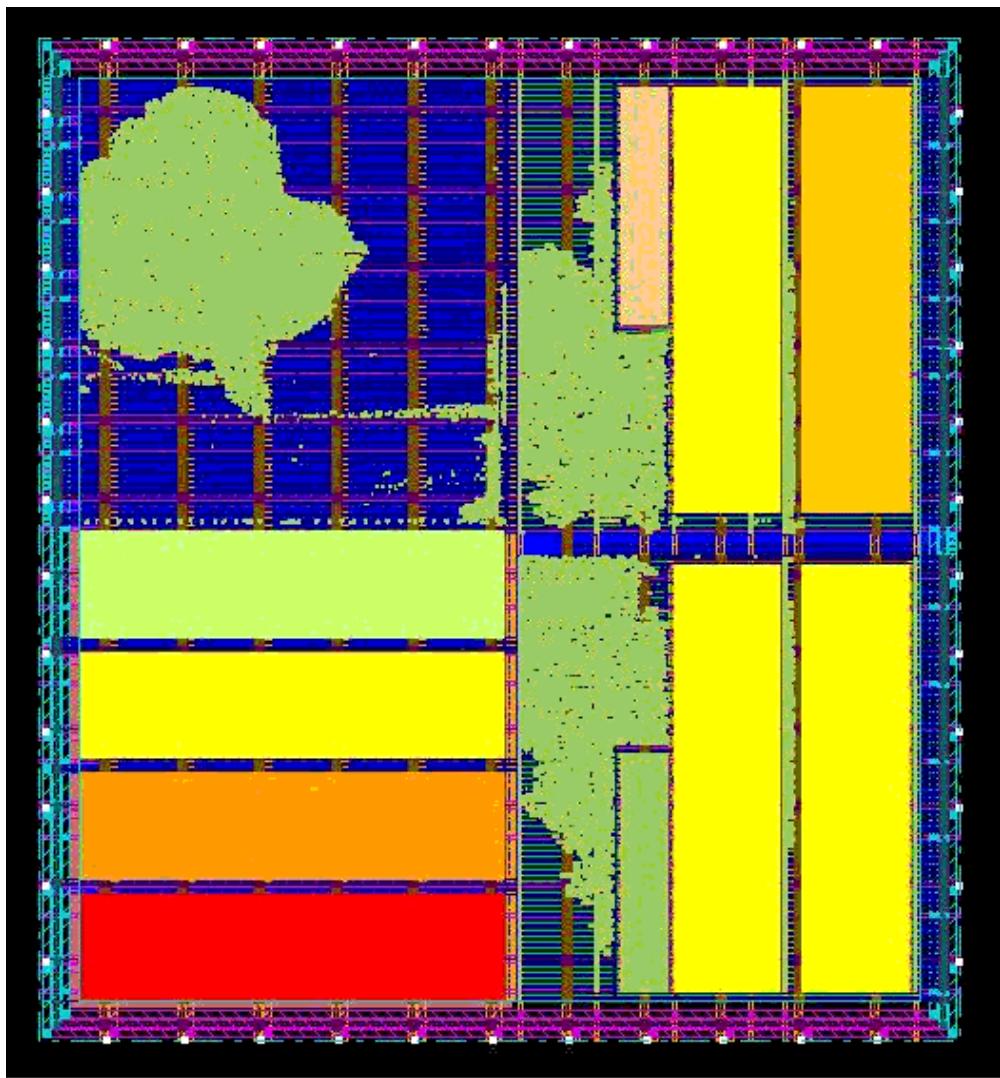
You can use this form to browse to other analysis runs and load them as well to view early analysis results and compare runs. All of the ERA runs will be located in the `FE2VSEarlyRA` directory. You can specify the type of plot (Rail Analysis, Power Analysis, or Capacitance) and then select the specific plot type. An instance power (ip), 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 *Encounter® Power System User Guide*.

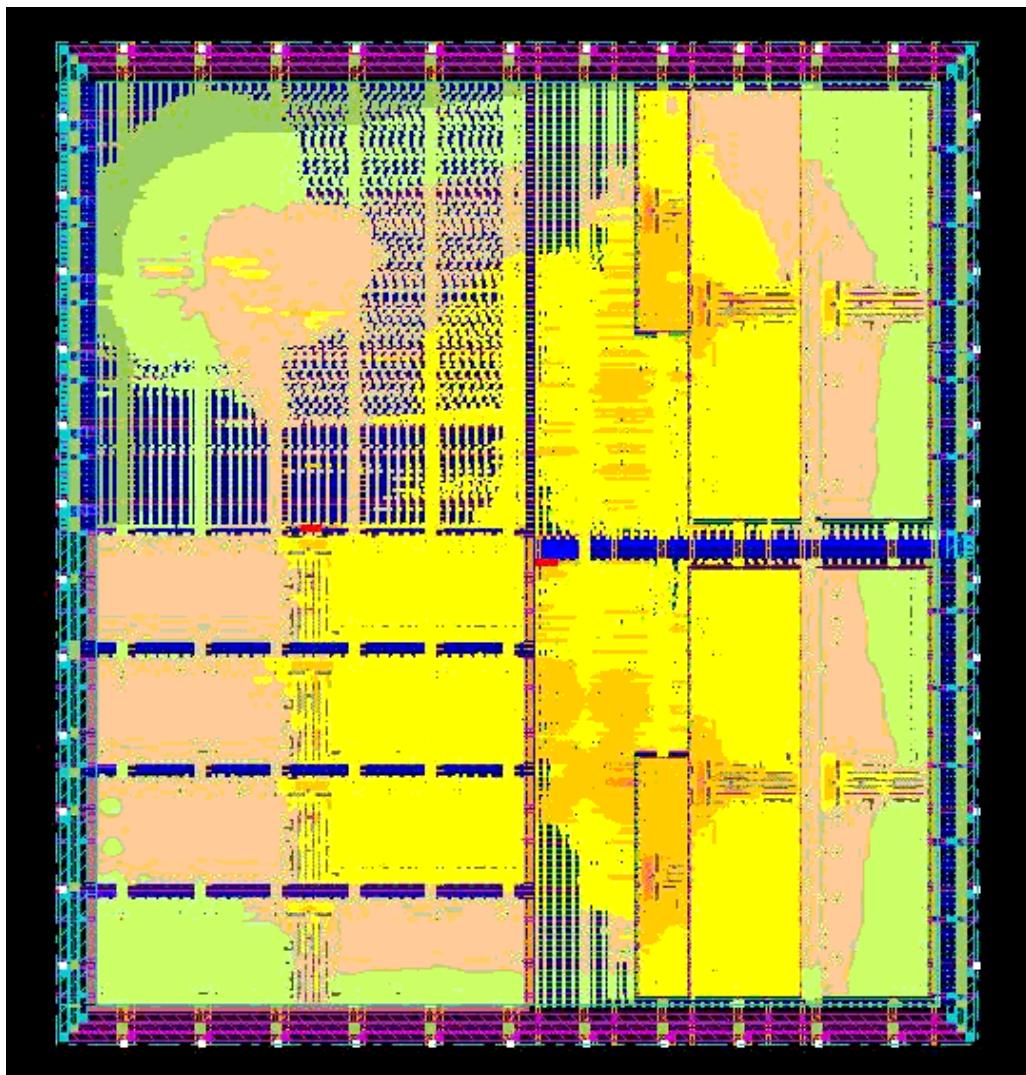
**Figure 37-9 Instance Power Plot**



**Figure 37-10 Load Capacitance Plot**



**Figure 37-11 irDrop Plot**



## Signoff-Rail Analysis

For details on running Signoff Power and Rail Analysis within EDI System, see *Encounter Power System User Guide* chapters 5-12.

## Electrostatic Discharge Analysis

For details on performing Electrostatic Discharge (ESD) analysis within EDI System, see "[Static Power, IRdrop and EM Analysis](#)" in the *Encounter® Power System User Guide*.

- ➊ To invoke this feature from EDS-L/EDS-XL, you need EPS-L/EPS-XL and EPS-AA licenses.

The EDI System Power and Rail Analysis commands and forms are identical to those in Encounter Power System, but the menus are organized differently. The forms are accessed in different places in the pull-down menus for each of the two products.

## EDI System and EPS menu differences

Form	EDI System Menu	EPS Menu
<i>Set Power Analysis Mode</i>	<i>Power - Power Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Run Power Analysis</i>	<i>Power - Power Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Early Rail Analysis</i>	<i>Power - Rail Analysis</i>	not part of EPS
<i>Set Power Library Mode</i>	<i>Power - Rail Analysis - PowerGrid Library</i>	<i>Power &amp; Rail Analysis - PowerGrid Library</i>
<i>Create PowerGrid Library</i>	<i>Power - Rail Analysis - PowerGrid Library</i>	<i>Power &amp; Rail Analysis - PowerGrid Library</i>
<i>Set Rail Analysis Mode</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Run Rail Analysis</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Create Hierarchical View</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>PowerGrid Library Report</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis- Report</i>
<i>Power Report</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis- Report</i>
<i>Power Histograms</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis- Report</i>
<i>Power &amp; Rail Results</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis- Report</i>
<i>Dynamic Movies</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis- Report</i>
<i>Dynamic Waveforms</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis- Report</i>



---

# 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](#)
- [Viewing Violations With the Violation Browser](#)
- [Clearing Violations](#)

## Overview

The verification commands in the Encounter® Digital Implementation System (EDI 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 routingFor more information, see
  - [Verifying Connectivity](#)
  - [verifyConnectivity](#) in the "Verify Commands" chapter of the *EDI System 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 *EDI System 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 *EDI System Text Command Reference*. This command is used for 28nm and above designs.
- [verify\\_drc](#) in the "Verify Commands" chapter of the *EDI System 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 *EDI System 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 *EDI System 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 *EDI System Text Command Reference* .

- Placement:

For more information on the types of violations, see [checkPlace](#) in the "Placement Commands" chapter of the *EDI System 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 EDI System commands, or import markers from another verification tool, such as Assura™ or Calibre, and view the markers with the Violation Browser. The EDI System software saves the markers with the database.

For more information, see [Viewing Violations With the Violation Browser](#) .

## Interrupting Verification

The following verification commands support "Interrupt" (ctrl-c):

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

If you press Ctrl-c while one of the above verification commands is being executed, the verification process stops and asks you for a confirmation. If you choose "N", the commands will continue to execute, and if you choose "Y", the verification process stops and the database is cleaned. You can then change the settings and run the verification process again, as required.

## 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 EDI System 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 *EDI System 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 EDI System session.

### Types of Connectivity Violations Reported

- Antennas (Dangling wires)  
Unconnected wires (dangling wires). For more information, see Types of Antenna Violations Reported.
- 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.

## Results

After verifying connectivity, you can use information in the violation report to repair connectivity violations. You can use the Violation Browser for interactive viewing and highlighting of violation markers.

You can see incremental results in the Violation Browser. For more information, see "Verify Connectivity" in the [Verify Menu](#) chapter of *EDI System Menu Reference*.

## 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 design must be detail routed.

- The design must be loaded into the current EDI System 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 *EDI System 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.

## Before You Begin

- Ensure the following LEF statements are specified:
  - `CLEARANCEMEASURE`
  - `USEMINSPACING` statements for obstructions and pins  
For more information, see the [LEF Syntax](#) chapter in the *LEF/DEF Language Reference*.
- If you plan to run `verifyGeometry` in multiple-CPU processing mode, use the EDI System multiple-CPU commands or select the appropriate options on the Multiple CPU Processing form. For more information, see Verifying Geometry in Multi-Thread Mode.
- Route the design.
- 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
- LEF58\_MANUFACTURINGGRID
- MINSTEP
- MINWIDTH
- OPPOSITEEOLSPACING
- PINMASK
- SPACING
- 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:

- a. On the Verify Geometry - Advanced page, click the *Set Multiple CPU* button to open the Multiple CPU Processing form.
- b. On the Multiple CPU Processing form, specify the number of local CPUs.
- c. Optionally, select *Release License*.
- d. Click *OK* to close the Multiple CPU Processing form and return to the Verify Geometry form.

When you return to the Verify Geometry form, the *Number of Local CPU(s)* option in this form is updated with the value you specified on the Multiple CPU Processing form.

- e. Run Verify Geometry.

#### Related Topics

- [Accelerating the Design Process By Using Multiple-CPU Processing](#)
- [Multiple-CPU Processing Commands](#) chapter in the *EDI System Text Command Reference*
- *Verify Geometry - Advanced* in the [Verify Menu](#) chapter of the *EDI System 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 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 *EDI System Text Command Reference*.

To avoid antenna violations, wires and nets must meet the following conditions:

- Regular wires
  - Must terminate on a pin or the center of a via.
  - If a vertical wire intersects a horizontal wire along its axis, the end of the vertical wire must be covered by the horizontal wire.
  - If the ends of a vertical wire and horizontal wire meet, they must meet at their end points.
- Regular net vias
  - Must be covered by a pin.
  - The center of the via must be the start or end point of a wire.
  - The center point of stacked vias must be coincident.
- Special wires
  - A point that is one-quarter of a wire width from the center of the ending edge of the special wire must be covered by a via, pin, or another wire on the same layer.
- Special net vias
  - The metal rectangle of the special net via must overlap with a special wire or via of the same net.

## Support for Via Rules

`verifyGeometry` uses the rules defined in the `VIARULE` section of the LEF file to check for violations caused by vias.

- The command checks the master via only, and flags violations on only one instance of the via 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]	TH0:	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 Encounter 11.10-b059_1  
# OS: Linux x86_64(Host ID icdcmopt03s)  
# Generated on: Thu Sep 22 16:36:05 2011  
# Design: test  
# Command: verifyWellAntenna -needToProtect decap* -changeToProte...  
#####
```

Encounter WellAntenna Verification Report

Design: test

Protects NeedToProtect Name Class [subclass]

1 0 ZLLLN550V15 CORE

1 0 ZHHHN550V15 CORE

1 0 X0RSN550V15 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.

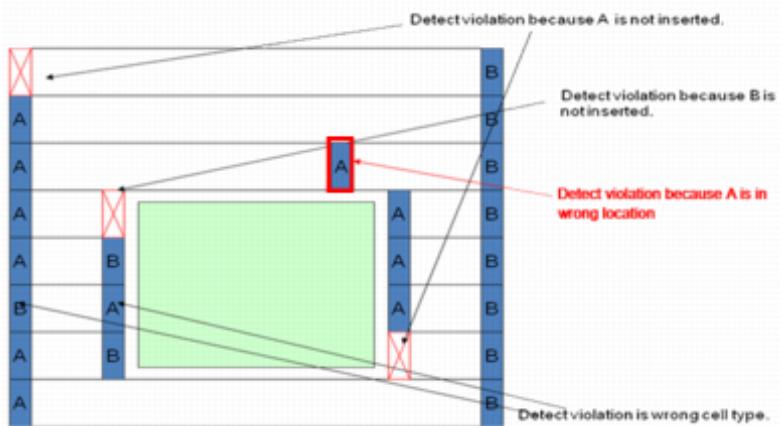
## Verifying End Cap Violations

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:

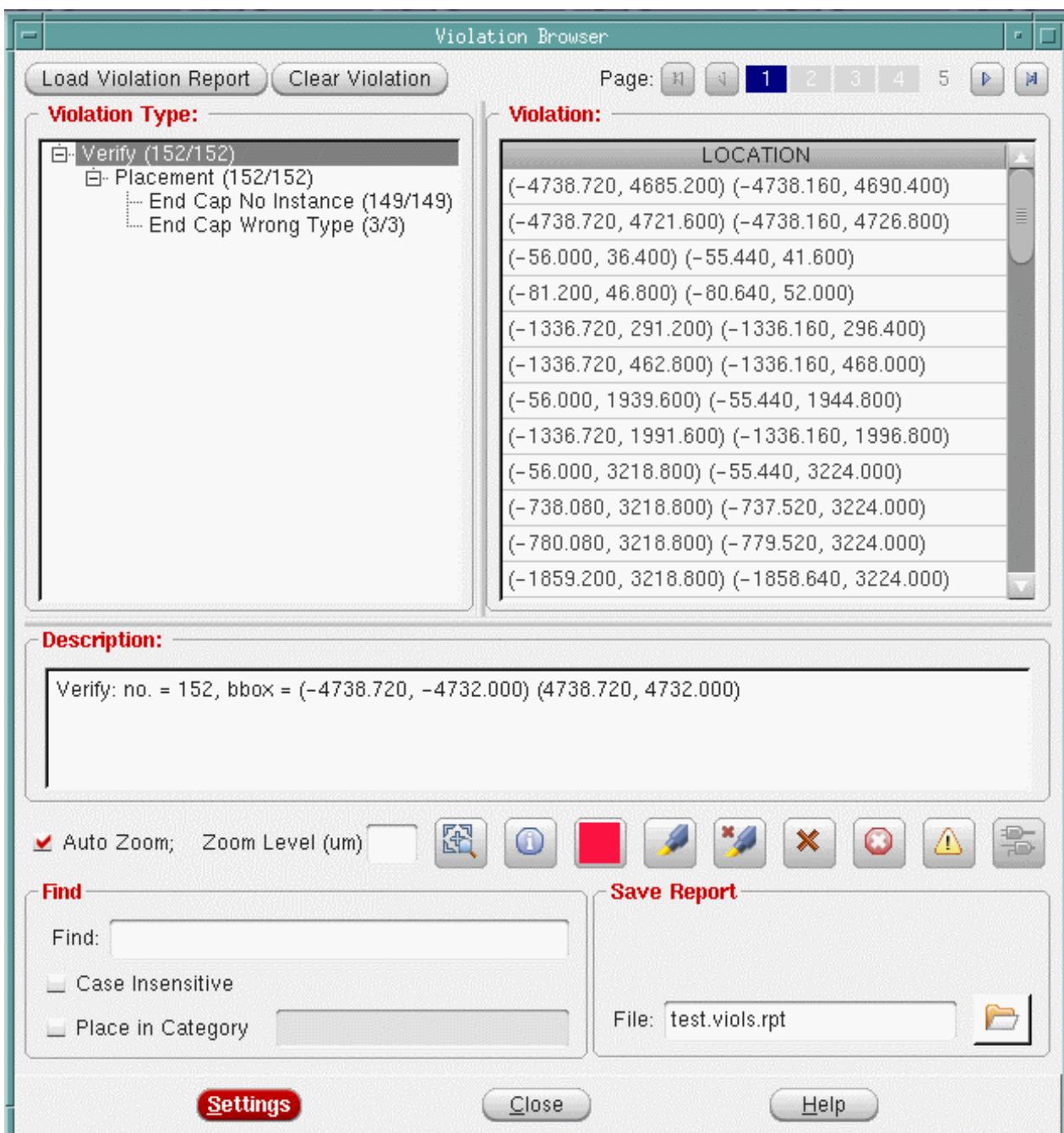
- Specified cap cell is missing. In the diagram, these locations where cap cells were expected but are missing are marked with a red cross enclosed in a rectangle .
- Inserted cap cell is of wrong type, that is cell B is inserted where A is expected and vice versa.
- Cap cell is inserted in wrong location. 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 Encounter 11.10-b059_1
```

```
# OS: Linux x86_64(Host ID icdcmopt03s)
# Generated on: Fri Sep 23 12:07:37 2011
# Design: test
# Command: verifyEndCap
#####
#####
```

### Encounter EndCap Verification Report

Design: test

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 EDI System software checks for maximum floating area violations by default when you run this command. For more information, see [verifyProcessAntenna](#) in the *EDI System 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*.

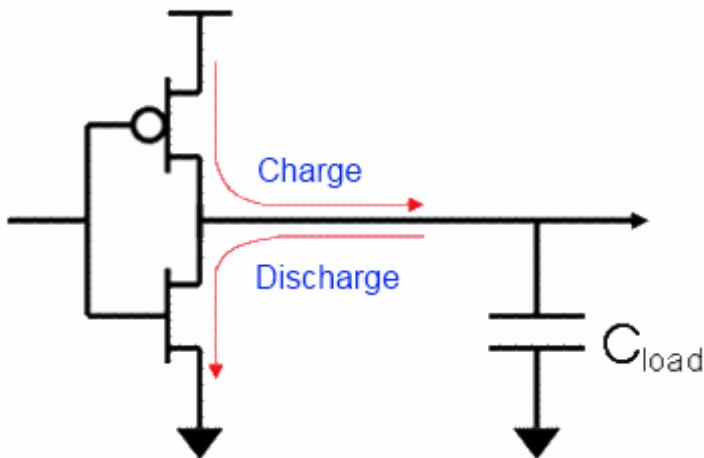
**Note:** To skip maximum floating area violation verification, but run process antenna verification, type the following command:

```
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.

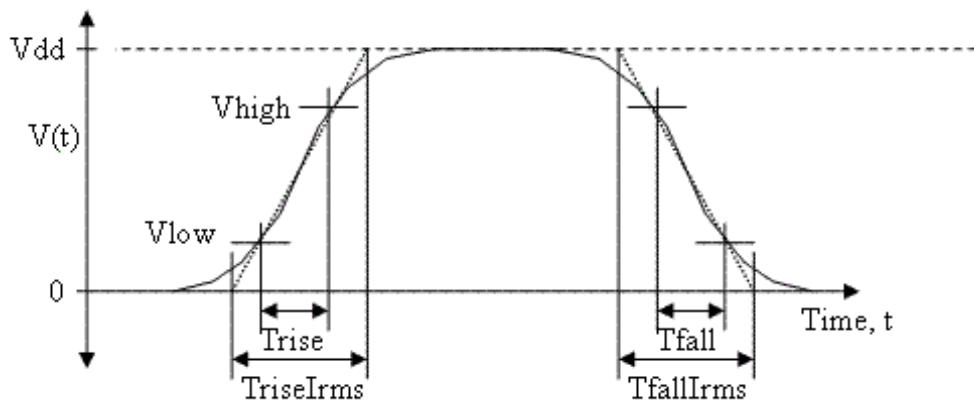
In previous versions, AC current density limits could be defined only in the Technology LEF file and only root-mean-square (RMS) current limit check was supported. From EDI System 11.1 onwards, peak AC current and average AC current limit check are also supported in addition to RMS current limit checks. 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

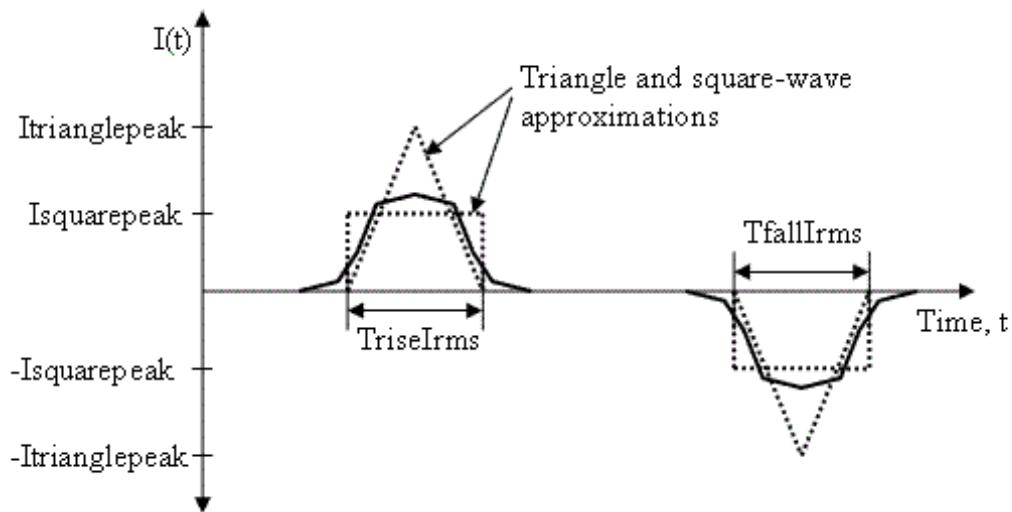
The software estimates the current waveform from the worst-case (fastest) transition time ( $T_{rise}$ ,  $T_{fall}$ ) given by the delay calculator.

The following diagram shows the  $v(t)$  square waveform for a given net:



$T_{rise}$  and  $T_{fall}$  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:



$T_{riseIrms}$  and  $T_{fallIrms}$  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

vlow = 20% of Vdd

vhigh = 80% of Vdd

then

$$TriseIrms = Trise / (.8 - .2) = Trise * 1.67$$

$$TfallIrms = Tfall / (.8 - .2) = Tfall * 1.67$$

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
```

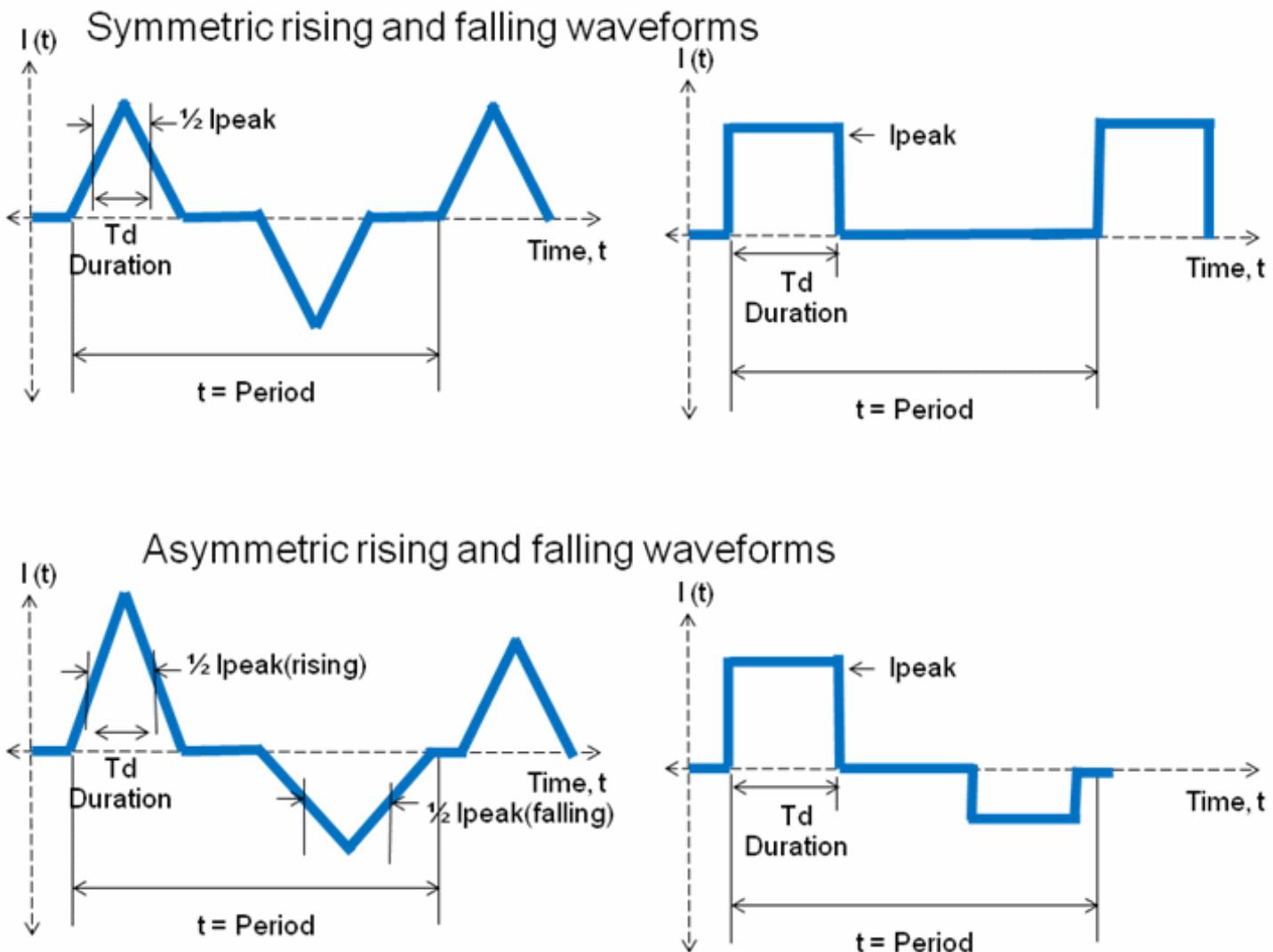
```
setDelayCalMode -engine aae
```

**Note:** If you have characterized ECSM waveforms for library cells using Encounter Library Characterizer,  $I_{rms}$  calculation by delay calculator will be the integration of simulated (characterized) current waveforms, instead of formula-based estimation as described in this section. To achieve more accurate results, it is recommended that you use the ECSM timing library.

## 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:

```
peak_measured = I_peak * sq. root[r]
```



Where,  $r$  is the duty ratio which is defined as pulse duration divided by the period.

$$r = T_d/t$$

The pulse duration  $T_d$  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)

$$\text{Duty ratio } r(\text{rising}) = [T_d(\text{rising}) + (I_{peak(\text{falling})}/I_{peak(\text{rising})})^2 * T_d(\text{falling})] / t$$

## Duty ratio r for I<sub>peak</sub> (falling)

Duty ratio  $r(\text{falling}) = [\text{Td}(\text{falling}) + (\text{I}_{\text{peak}}(\text{rising})/\text{I}_{\text{peak}}(\text{falling}))^2 * \text{Td}(\text{rising})] / t$

Using the above duty ratios for rising and falling peak waveforms,

$$\text{I}_{\text{peak}}(\text{rising}) * \text{sq.root}[r(\text{rising})] == \text{I}_{\text{peak}}(\text{falling}) * \text{sq.root}[r(\text{falling})]$$

Therefore, the software can pick either of these values to compare against the  $I_{\text{peak}}$  limit provided by the foundry. If the value of the measured peak AC current is larger than the  $I_{\text{peak}}$  limit, it will be considered as violation. Currently, the  $I_{\text{peak}}$  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 I<sub>avg</sub> 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,  $I_{\text{avg}}$  is calculated using the following equation:

### Equation1 :

$$I_{\text{avg.}} = \text{abs. max}(\text{rising}, \text{falling}) - EM\_recovery\_factor * \text{abs. min}(\text{rising}, \text{falling})$$

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,  $I_{\text{avg}}$  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,  
 $\text{abs. } I_{\text{avg}}(\text{rising}) = \text{abs. } I_{\text{avg}}(\text{falling})$ .

$$I_{avg(falling)} = \frac{S}{T_{sw}} \int_0^{T_{sw}} I_{(falling)}(t) dt$$

$$I_{avg(rising)} = \frac{S}{T_{sw}} \int_0^{T_{sw}} I_{(rising)}(t) dt$$

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:

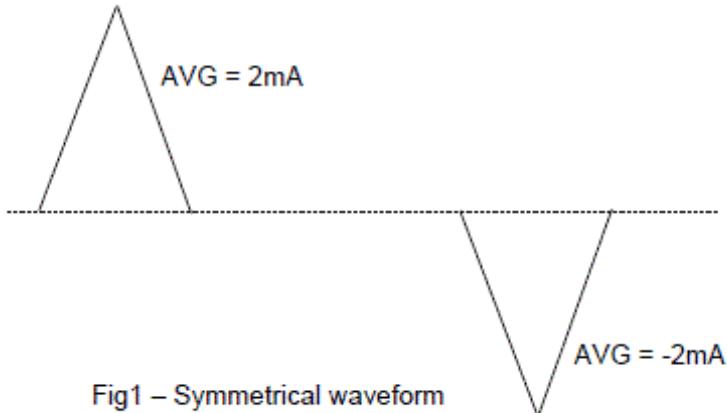


Fig1 – Symmetrical waveform

In this example:

- $I_{avg} (rising) = 2mA$
- $I_{avg} (falling) = -2mA$
- $em\_recovery\_factor = 0.5$

Based on the equation,  $I_{avg}$  calculation is as follows:

$$I_{avg} = 2 - 0.5 \times 2 = 1mA$$

## 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

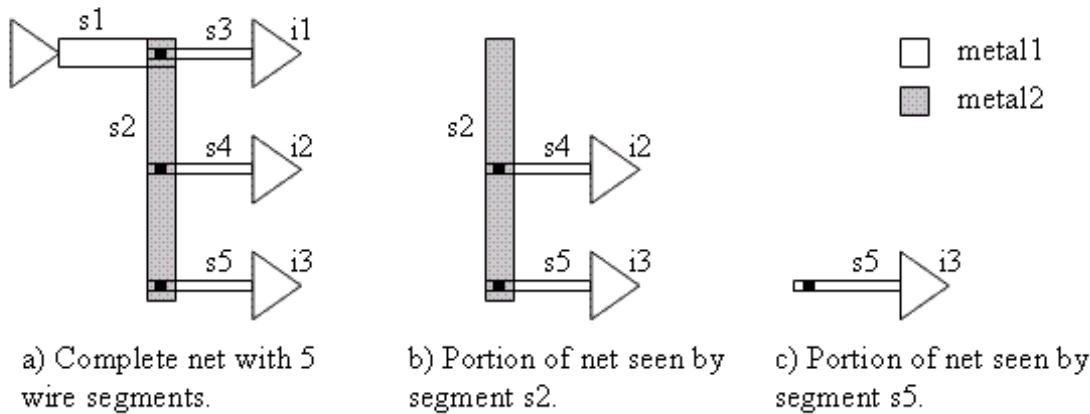
verifyACLimit \
-method avg \
-detailed \
-avgRecovery 0.5 \
-useQRCTech \
-use_db_freq \
```

```
-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)

$I_{rms}$  (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 EDI, 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  $0.15\mu m$  can carry  $10.0 \text{ mA}/\mu m * .15\mu m = 1.5\text{mA}$  of current.

The  $I_{rms}$  limits in the table can also be scaled for other factors like temperature. For example, the  $I$

$I_{rms}$  limits table should be computed for a specific "maximum allowed temperature change" ( $\max T_{change}$ ). The  $I_{rms}$  limits are normally proportional to the square-root of the  $\max T_{change}$ . Therefore, if the table used a  $\max T_{change}$  of 20 degrees, and you want a  $\max T_{change}$  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 verifyACLlimit 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.

verifyACLlimit 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 EDI System using the -qx\_tech\_file parameter of [create\\_rc\\_corner](#) or [update\\_rc\\_corner](#). verifyACLlimit 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 EDI System Text Command Reference.

## 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 EDI System 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 EDI System 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 EDI System 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 *EDI System 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 EDI System software can interpret. For more information, see [loadViolationReport](#) in the *EDI System Text Command Reference* .

- `violationBrowser` (*Tools - Violation Browser*)

This command displays the markers in the Violation Browser. For more information, see [violationBrowser](#) in the *EDI System 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 EDI System 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.

## Clearing Violations

Choose the *Tools - Violation Browser* menu item and then click the *Clear Violation* button to clear the violation markers in your design.

## 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

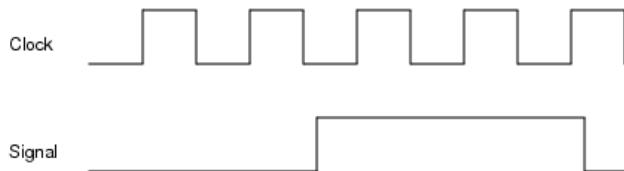
Internal power is calculated by using the internal power tables provided in the .lib, which capture the characterized internal power over a range of input slew rates and external loading. The tables reflect the combination of both the internal switching and internal feedthrough power. Tables are generated 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:

$$\text{inputEnergy} = \frac{1}{2}(\text{port} \rightarrow \text{riseEnergy}() + \text{port} \rightarrow \text{fallEnergy}())$$

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:

$$\text{inputEnergy} = \frac{\sum_i \text{prob}(\text{port} \rightarrow \text{when}(i)) * (\text{port} \rightarrow \text{when}(i) \rightarrow \text{riseEnergy}() + \text{port} \rightarrow \text{when}(i) \rightarrow \text{fallEnergy}())}{2 \sum_i \text{prob}(\text{port} \rightarrow \text{when}(i))}$$

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:

$$\text{inputEnergy} = \frac{\text{prob}(A \& B) * (3.0nJ + 4.1nJ) + \text{prob}(A \& !B) * (3.2nJ + 5.0nJ) + \text{prob}(!A \& !B) * (7.0nJ + 9.0nJ)}{2(\text{prob}(A \& B) + \text{prob}(A \& !B) + \text{prob}(!A \& !B))}$$

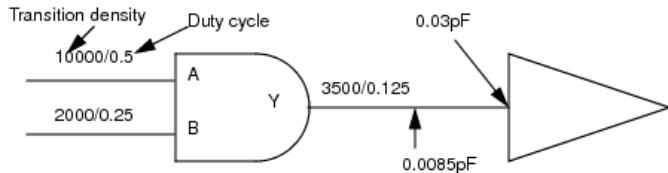
$$= \frac{0.375(7.1nJ) + 0.125 * (8.2nJ) + 0.375(16nJ)}{2(0.375 + 0.125 + 0.375)} = \frac{9.6875}{1.75} = 5.5357nJ$$

#### **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.

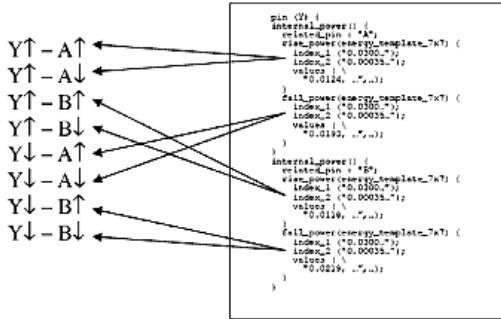
The resulting time should be converted, if necessary, to the time units provided in the .lib:

```
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:

$Y \uparrow - A \uparrow$   
 $Y \uparrow - A \downarrow$   
 $Y \uparrow - B \uparrow$   
 $Y \uparrow - B \downarrow$   
 $Y \downarrow - A \uparrow$   
 $Y \downarrow - A \downarrow$   
 $Y \downarrow - B \uparrow$   
 $Y \downarrow - B \downarrow$

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:

$$E_{Y,rise} = \frac{10000s^{-1} * \frac{1}{2}(0.0134pJ + 0.0134pJ) + 2000s^{-1} * \frac{1}{2}(0.0135pJ + 0.0135pJ)}{10000s^{-1} + 2000s^{-1}}$$

$$= \frac{161pJs^{-1}}{12000s^{-1}} = 0.0134167pJ$$

$$E_{Y,fall} = \frac{10000s^{-1} * \frac{1}{2}(0.0244pJ + 0.0244pJ) + 2000s^{-1} * \frac{1}{2}(0.0267pJ + 0.0267pJ)}{10000s^{-1} + 2000s^{-1}}$$

$$= \frac{297.4pJs^{-1}}{12000s^{-1}} = 0.027833pJ$$

The power is:

$$P = \frac{1}{2}(E_{Y,rise} + E_{Y,fall})D(Y)$$

$$= 0.5(0.0134167pJ + 0.027833pJ)3500s^{-1}$$

$$= 66.85pW$$

#### 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:

$$SwitchingPower = CAV^2 F$$

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:

$$\text{SwitchingPower} = 1/2CV^2 D$$

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.

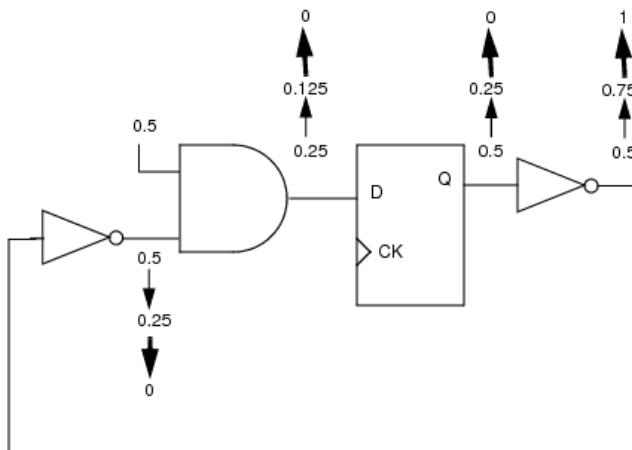
#### *Activity propagation through combinational cells*

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.



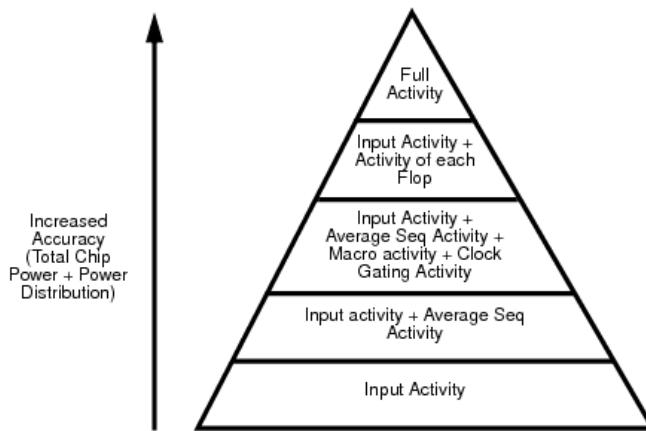
## ***Activity propagation through macros***

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 EDI System 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

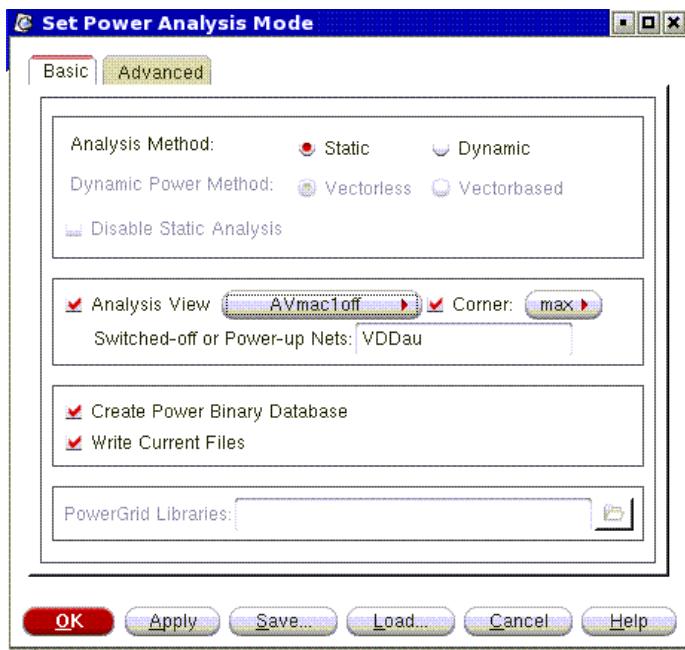
To perform static analysis one must setup the analysis mode and then run power analysis.

### 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 *aDuty* 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

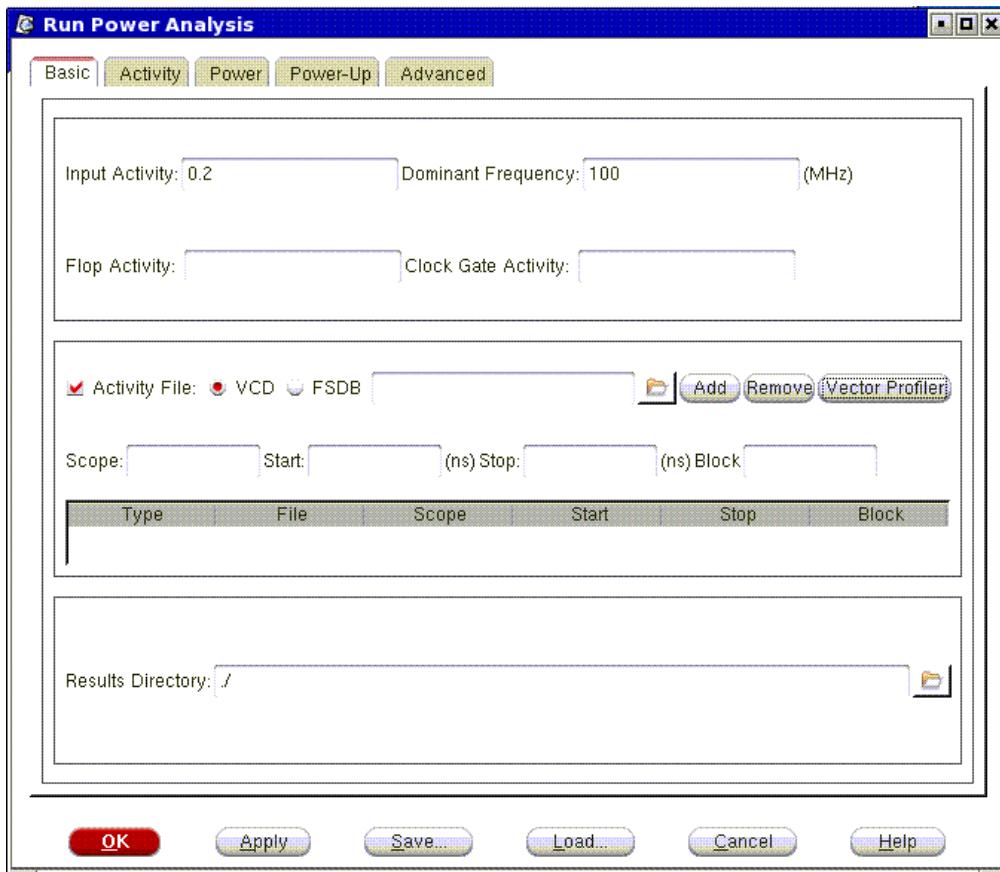


Figure 39-3 Run Power Analysis - Activity form

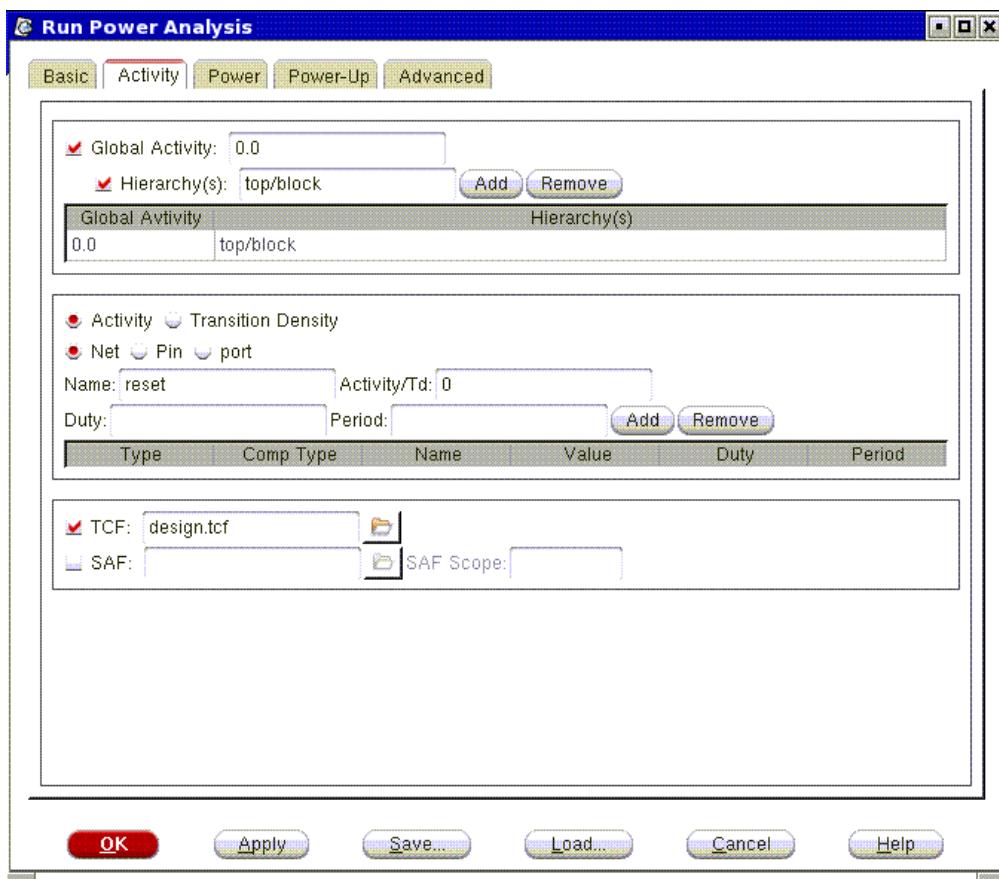
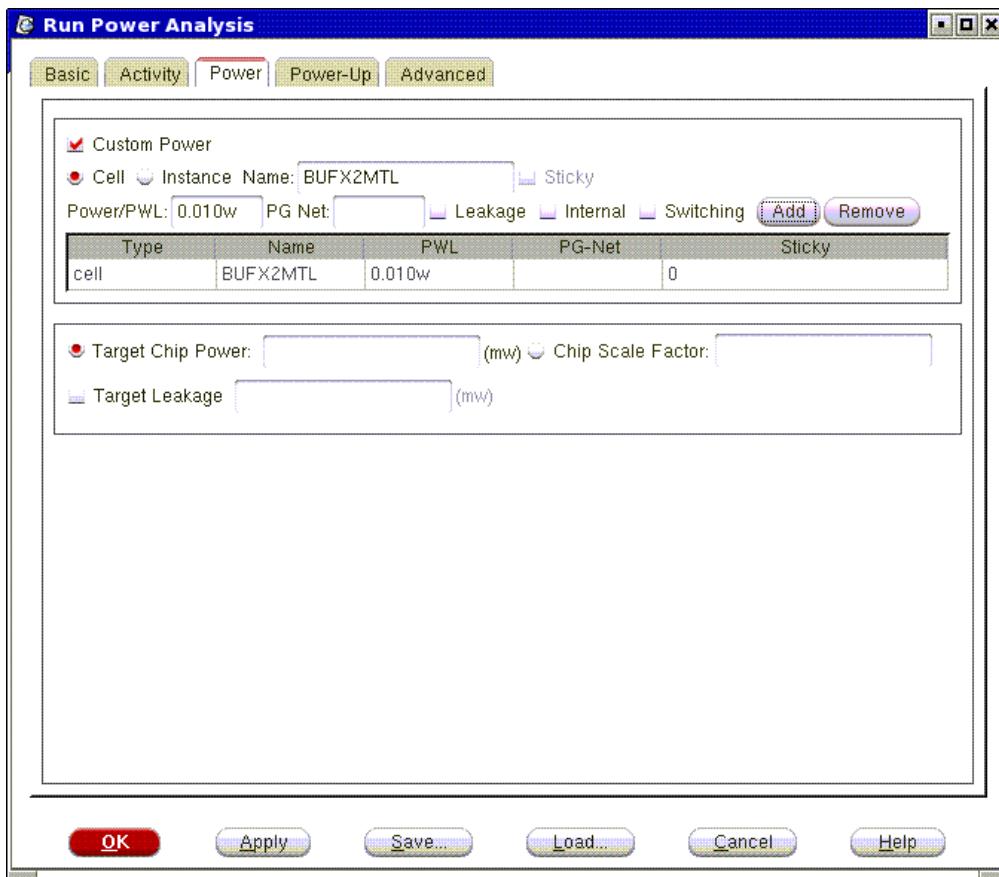


Figure 39-4 Run Power Analysis - Power form



**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 :

- auto\_wire\_load\_selection

- enable\_wire\_load\_model\_support
- report\_wire\_load
- reset\_wire\_load\_mode
- reset\_wire\_load\_model
- reset\_wire\_load\_selection\_group
- set\_wire\_load\_mode
- set\_wire\_load\_model
- set\_wire\_load\_selection\_group

**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 EDI System *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.

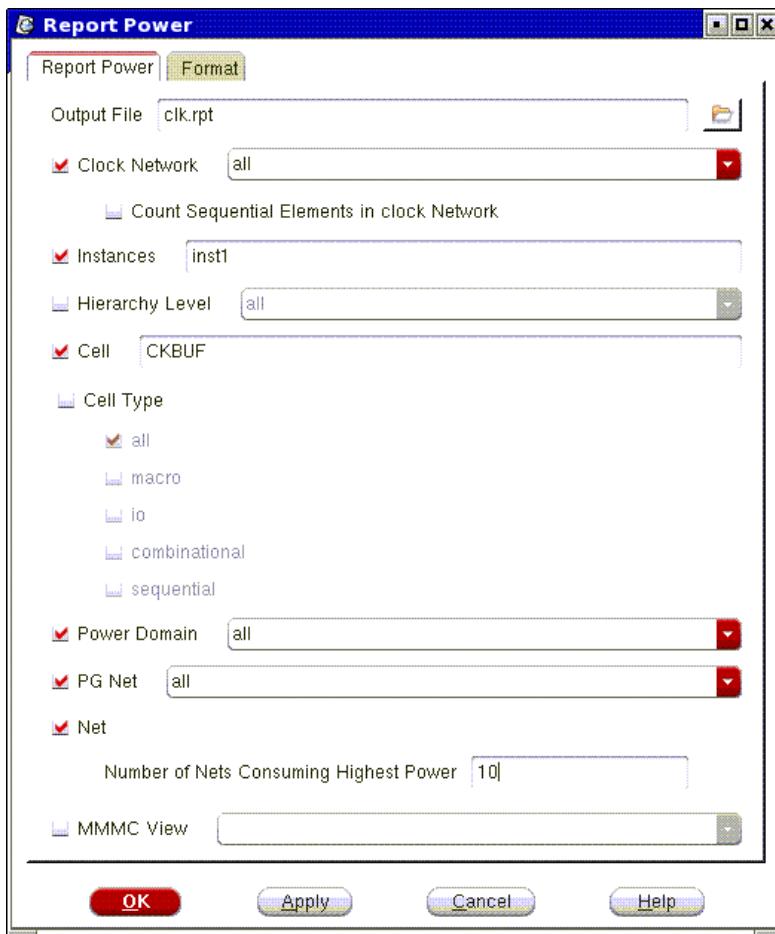
If multiple command types specify the view, the priority is given as follows (from highest to lowest):  
`report_power`, `set_power_analysis_mode`, `set_analysis_mode`

For additional MMMC details see *Performing Multi-Mode Multi-Corner Timing Analysis* in the EDI System 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
```

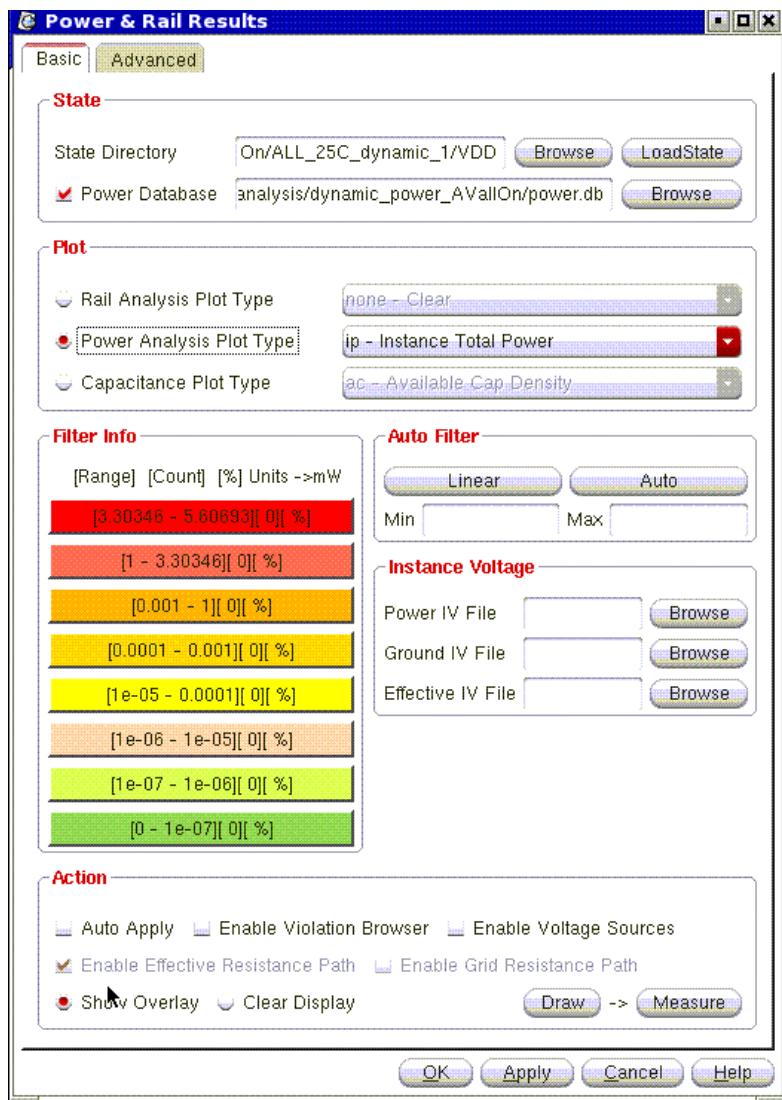
## **Static Power Analysis Plots**

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:

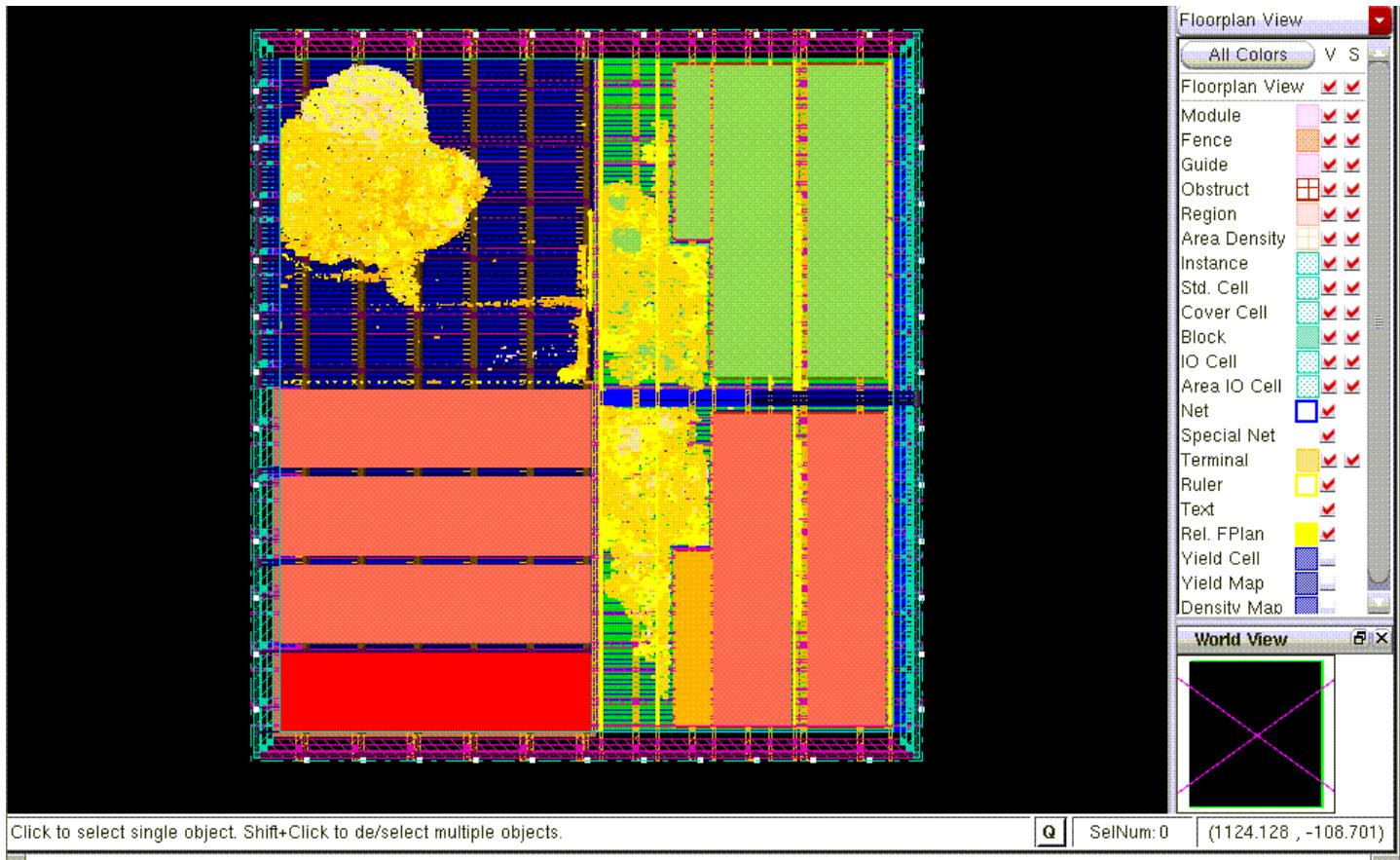
1. Select *Power & Rail - Report - Power & Rail Results* to bring up the form shown in Figure 39-6.
2. If power is calculated during the session, select *Power Analysis Type* radio button and select a power analysis type. See [Viewing and Debugging Static Plots](#) for details of the various plot types 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 EDI System 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 in Encounter Power System. 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*.

# Creating An Initial Floorplan Using Automatic Floorplan Synthesis

---

- [Overview](#)
  - [Automatic Floorplan Synthesis Flow](#)
- [Data Preparation](#)
  - [Selecting Seeds](#)
- [Importing the Design](#)
- [Setting Automatic Floorplan Synthesis Global Parameters](#)
- [Creating an Initial Floorplan](#)
- [Creating Floorplan for Hierarchical Design](#)
  - [Macro placement](#)
  - [Full-chip Floorplan](#)
  - [Power-Domain Aware Floorplan](#)
- [Creating Multiple Alternative Floorplans](#)
- [Analyzing the Floorplan](#)
- [Adjusting Macro Placement](#)
  - [Manual Macro Adjustment](#)
  - [Automatic Floorplan Synthesis Macro Adjustment](#)
- [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.

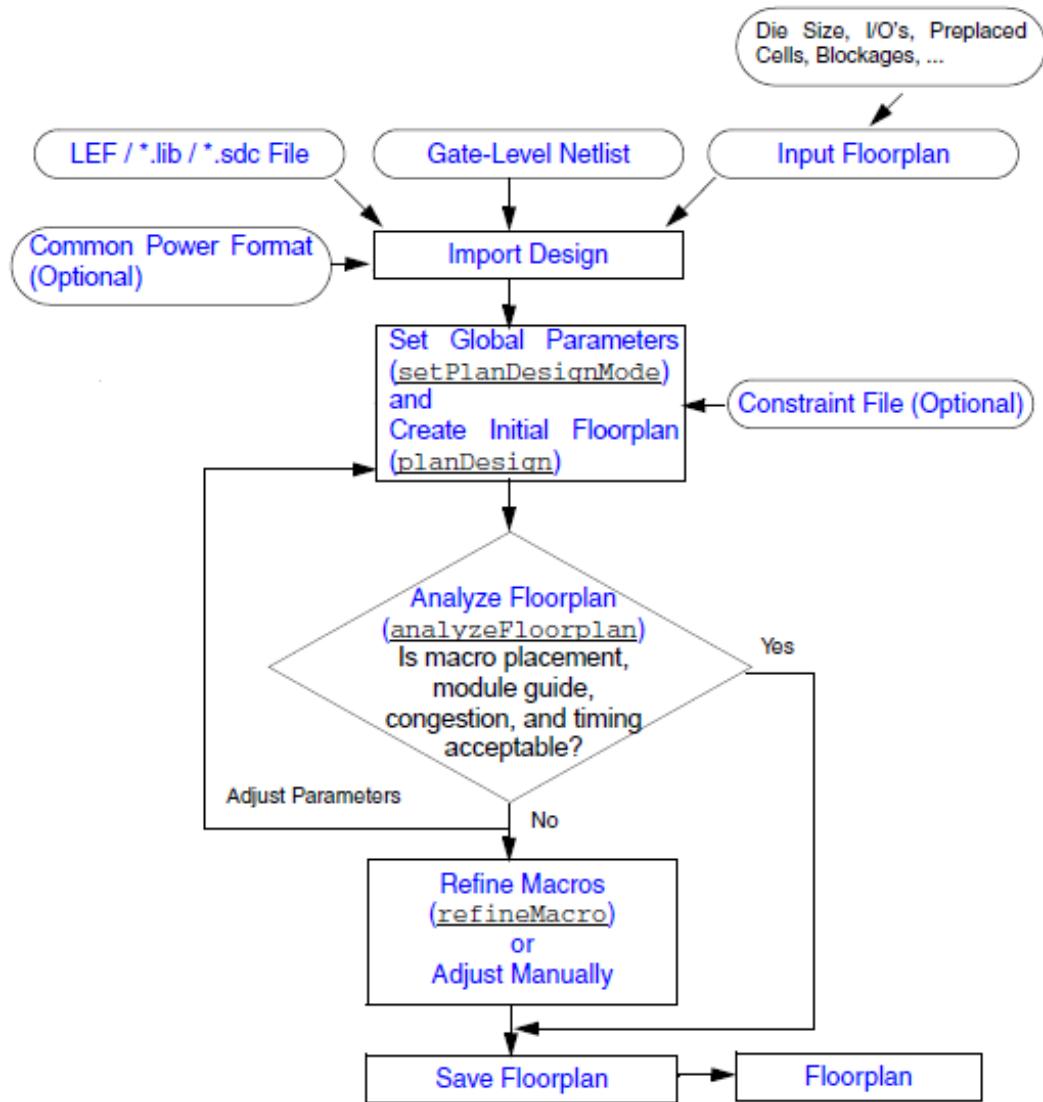
Ideal designs for use with Automatic Floorplan Synthesis are:

- 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

Figure 41-1 shows the typical task flow for using Automatic Floorplan Synthesis to create an initial floorplan:

**Figure 41-1 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:

<code>name= seedname</code>	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 <code>seedname</code> must be a hierarchical module.
<code>util= utilization_value</code>	Specifies the utilization value for the specified module or instance group.
<code>createFence=true</code>	Indicates that a fence should be generated for the hierarchical module.
<code>minWHRatio= value</code>	Specifies the minimum width to height ratio.  <i>Type :float</i>
<code>maxWHRatio= value</code>	Specifies the maximum width to height ratio.  <i>Type :float</i>
<code>minFenceToFenceSpace= value</code>	Specifies the minimum amount of spacing, in microns, allowed between fences.
<code>minFenceToCoreSpace= value</code>	Specifies the minimum amount of spacing, in microns, allowed between a fence and the core boundary.
<code>MinFenceToInsideMacroSpace= value</code>	

	Specifies the minimum amount of spacing, in microns, allowed between the fence boundary and the macros that belong to the fence.
<code>minFenceToOutsideMacroSpace= value</code>	
	Specifies the minimum amount of spacing, in microns, allowed between the fence boundary and the macros that do not belong to the fence.
<code>minInsideFenceMacroToMacroSpace= value</code>	
	Specifies the minimum amount of spacing, in microns, allowed between macros that belong to the same fence.
<code>master= seedname</code>	
	Specifies the name of the master seed. The clone seed inherits the same constrains (except orientation) as the master seed.
<code>cloneOrient={R0 MX MY R180}</code>	
	Specifies the orientation of the clone seed.

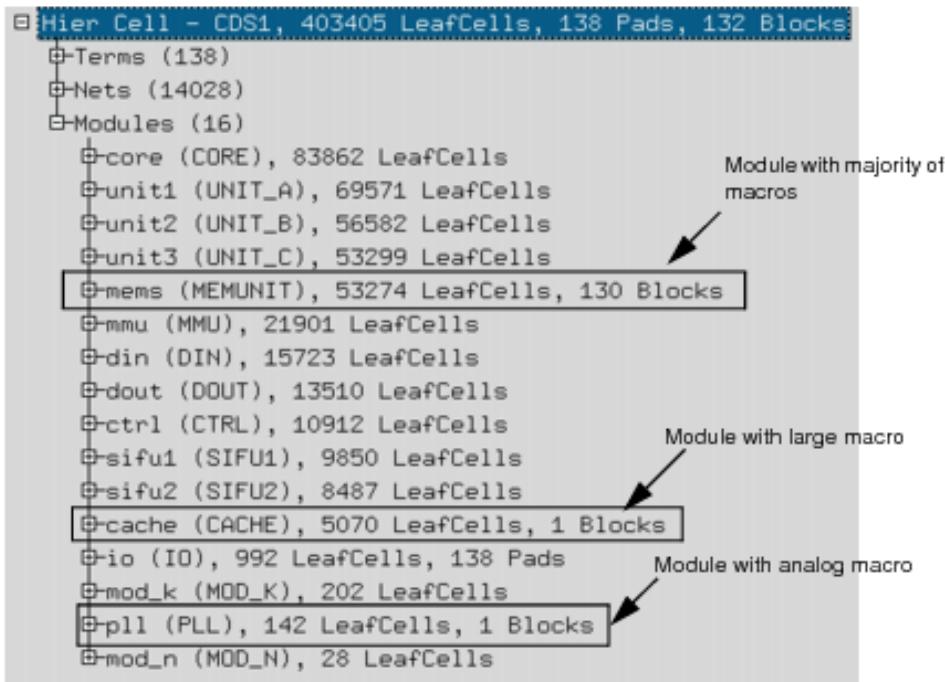
For example:

```
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

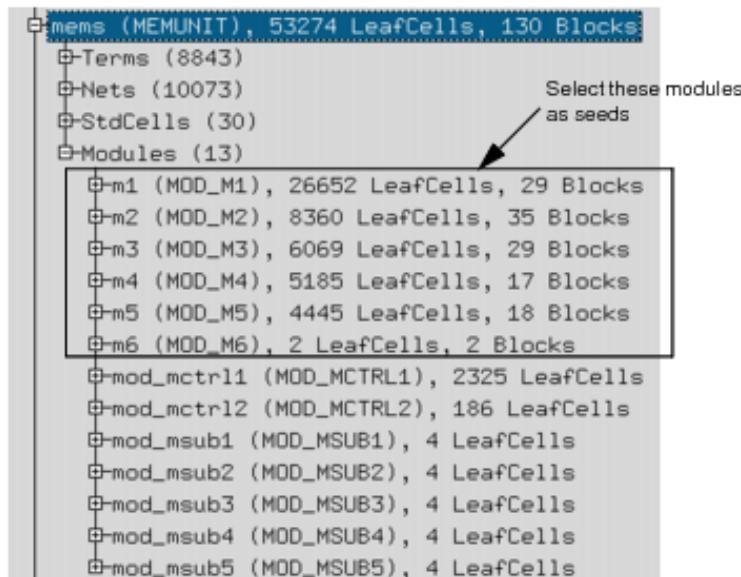
connectivity.

- 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.

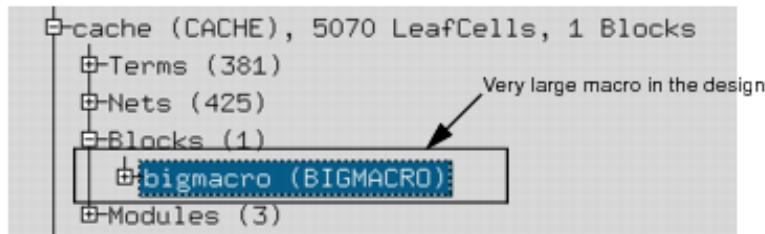
In Figure 41-2, you should select the modules at this level as seeds because it will produce better grouping for macro placement.

**Figure 41-2**



In Figure 41-3, you should select this macro as a seed because it is larger than 1/4 of the core area.

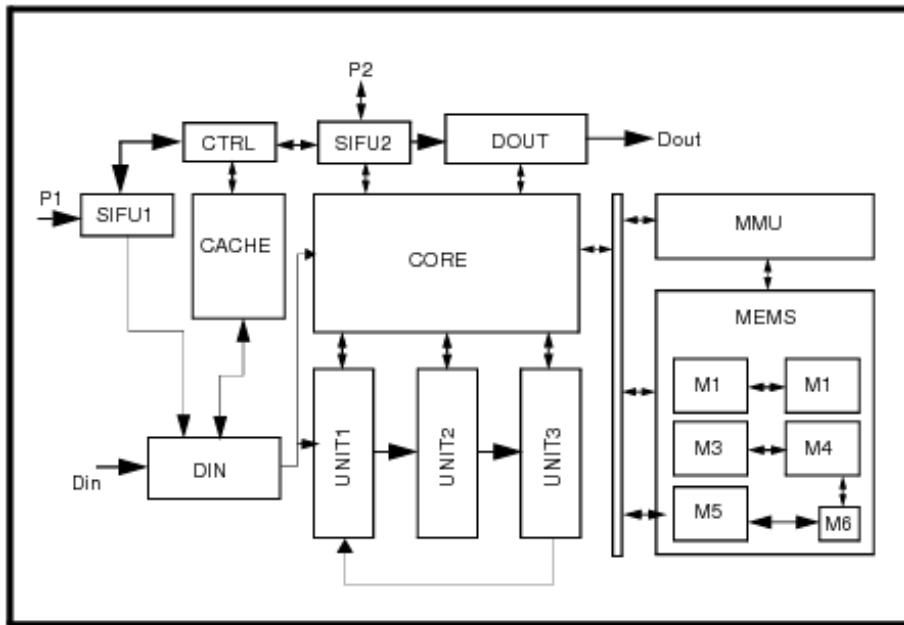
**Figure 41-3 Large Macro**



#### 4. Create a seed section.

For example, Figure 41-4 shows the block diagram for the design.

**Figure 41-4 Design Block Diagram**



Based on this diagram:

1. Examine the module `mems` because it contains the majority of the macros in the design. Select seeds from its sub modules in order to get better macro placement (Figure 41-2 ).
2. Examine the module `cache`, and select the macro `bigmacro` because it is very large (Figure 41-3 ).
3. Do not select the modules named `mod_k` and `mod_n` as seeds because they are so small that they do not show up in the block diagram.
4. Do not select the module named `io` because it includes I/O pad cells that have already been

pre-placed.

The resulting seed section is as follows:

```
core          #Hinst: good for data flow
unit1         #Hinst: good for data flow
unit2         #Hinst: good for data flow
unit3         #Hinst: good for data flow
mmu          #Hinst: good for data flow
din           #Hinst: good for data flow
dout          #Hinst: good for data flow
ctrl          #Hinst: good for data flow
sifu1         #Hinst: good for data flow
sifu2         #Hinst: good for data flow
mems/m1       #Hinst: good for data flow and macro grouping
mems/m2       #Hinst: good for data flow and macro grouping
mems/m3       #Hinst: good for data flow and macro grouping
mems/m4       #Hinst: good for data flow and macro grouping
mems/m5       #Hinst: good for data flow and macro grouping
mems/m6       #Hinst: good for data flow and macro grouping
pll           #Hinst: maintain for analog macro
cache/bigmacro #Inst: maintain for big macro
```

## Importing the Design

1. To import a design, type the following command:

```
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.

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 (Figure 41-5), 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.

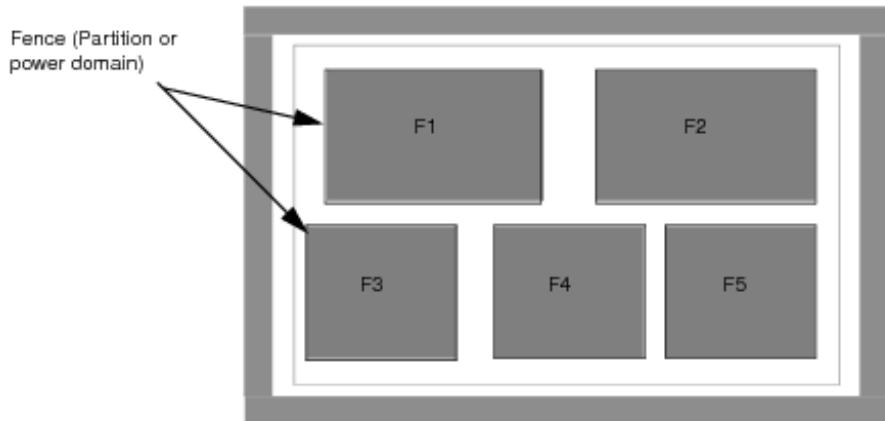
- Power-Domain Aware Floorplan

Reads CPF, creates power domains, and places macros.

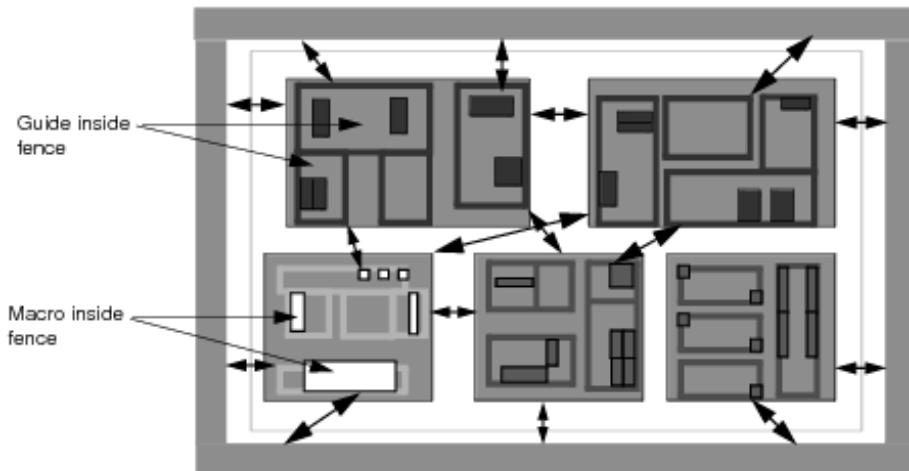
## Macro placement

Automatic Floorplan Synthesis places hard macros, and generates guides for the sub modules within the fence boundary and also on the top level (Figure 41-6), considering the global connectivity.

**Figure 41-5 Hierarchical Top-Level Floorplan**



**Figure 41-6 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.

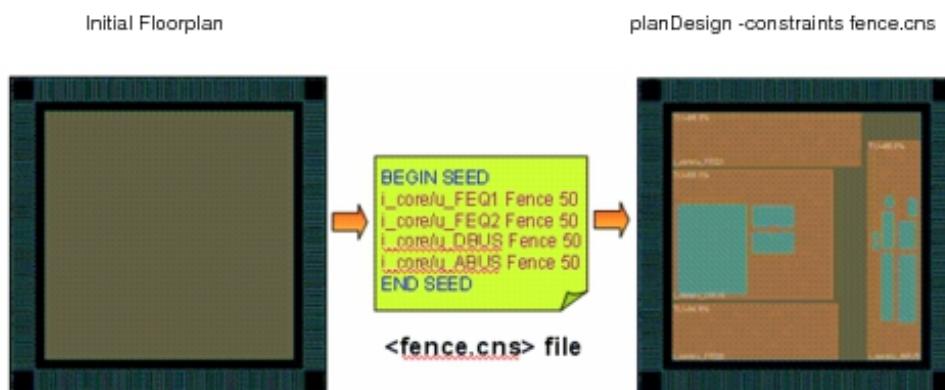
## Full-chip Floorplan

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.

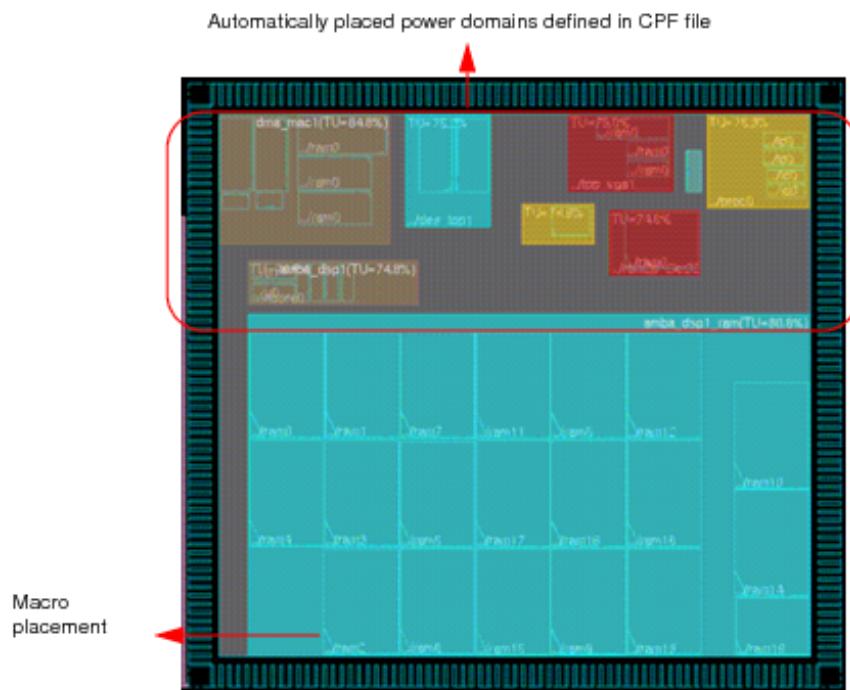
**Note:** The `planDesign` command places power domains inside the core boundary without any overlaps with other flexModels or macros (especially rectilinear macros), thus improving the QoR of

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 *EDI System 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 Encounter® Digital Implementation System (EDI System) 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 EDI System 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 EDI System 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 *EDI System Menu Reference*.

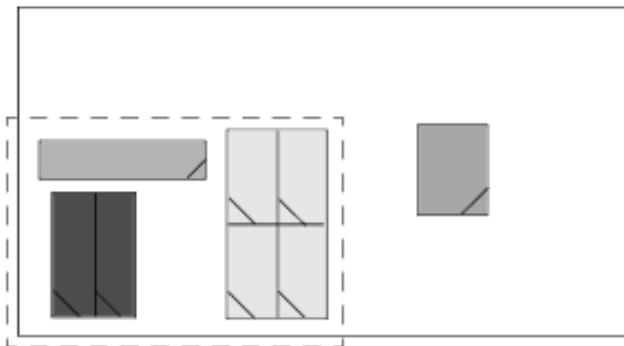
## Automatic Floorplan Synthesis Macro Adjustment

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.

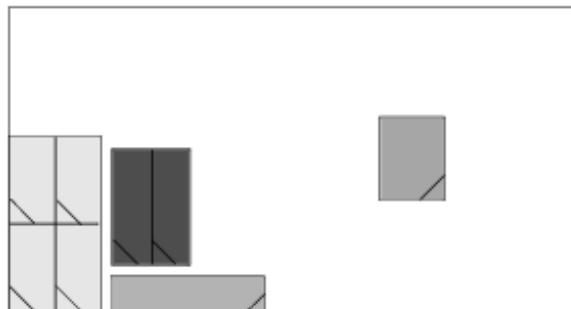
 Macro pack information is stored temporarily in the data base. If you quit the EDI System session, the information is lost



2. Type the following command:

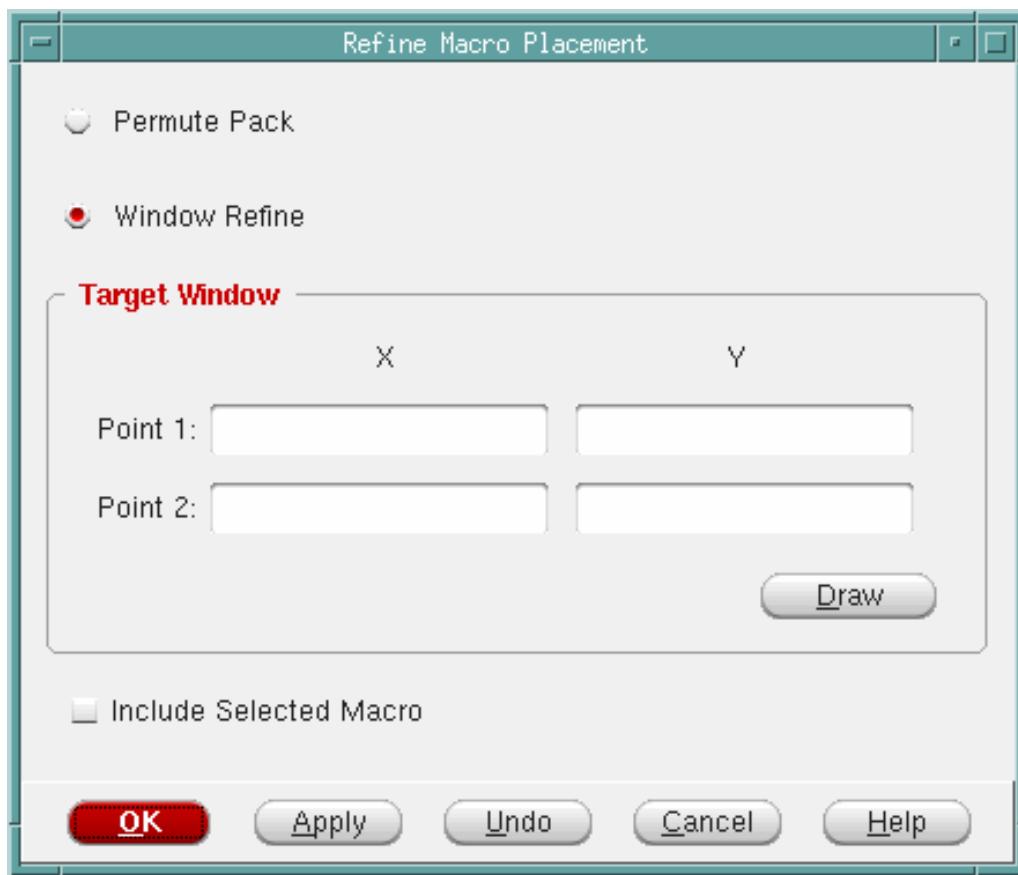
```
refineMacro -permutePack
```

Automatic Floorplan Synthesis adjusts the macros' locations to reduce empty area between them.

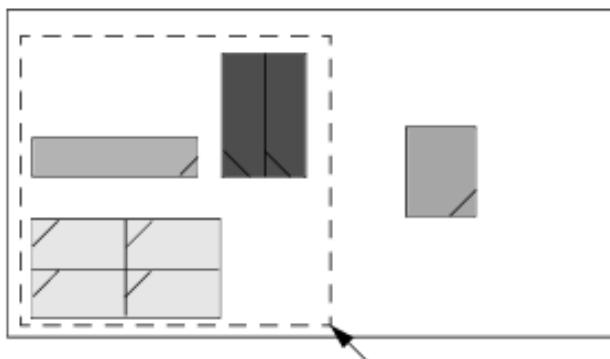


#### Adjusting Macro Placement Within a Specified Area

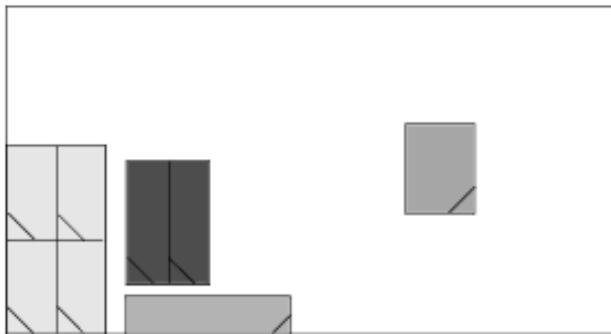
1. Choose *Floorplan - Automatic Floorplan - Refine Macro Placement*.



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.

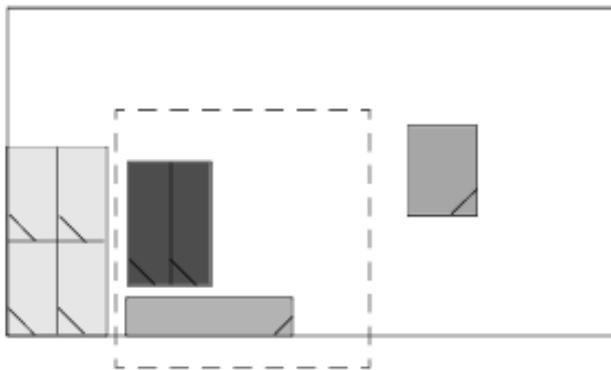


5. Click *Apply* to perform the adjustment.

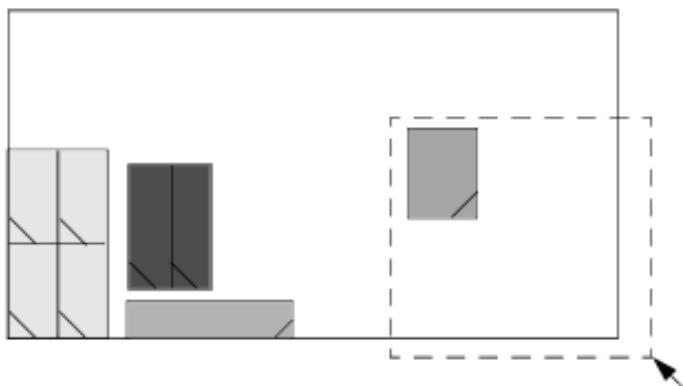


You can also move selected macros to a specified area and adjust them along with any macros already within the area.

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 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
```

---

# 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 QRC TechGen manual.

## Format

Lines in the ICT file are in the following general format:

*command name {argument\_list}*

where *argument\_list* is a list of field-value pairs. The fields in this syntax are separated by white space. ViewICT, IceCaps, and RCgen ignore blank lines.

**Note:** A backslash (\) is generally required for line continuation, but it is not required if you are using braces ({} ) to define a list.

## Data

All data entered into the ICT file should be the actual physical fabrication process information, not the drawn data.

## Comments

A pound-sign character (#) at the beginning of a line indicates text that ViewICT, IceCaps, and RCgen are treated as comments.

## Case Sensitivity

All keywords in the ICT file are case-insensitive. However, the arguments are case-sensitive. Keywords consist of all command and field names.

## Warnings and Errors

The ViewICT utility displays all errors, warnings, and informational messages on screen and writes them in a log file. Warnings and errors include the corresponding line number.

## Invalid Layer Names

The "NX" string is an invalid layer name.

## Commands

This section describes the commands available in the ICT file.

All command fields are enclosed in braces ({}).

### Process

The process command specifies the background dielectric constant. Use it only once in the ICT file.

#### Syntax

```
process name {background_dielectric_constant value }
```

or

```
process name {  
    background_dielectric_constant value  
}
```

This syntax contains the following parameters:

- *name*  
Specifies the name of the process.
- *background\_dielectric\_constant value*  
Specifies the dielectric constant for the region above the top passivation layer or last dielectric layer. This field is required.

#### Example

```
process "Process_Example" {  
    background_dielectric_constant 1.0
```

}

## Well

The `well` command which defines the well layers is an optional command that you can use to differentiate capacitance to a well from capacitance to the substrate.

### Syntax

```
well name { }
```

*Name* specifies the name of the well layer.

Anything placed in the brackets is ignored.

### Example

```
well nwell { }
```

```
well pwell { }
```

## Conductor

The `conductor` command defines conductor layers.

You can specify the height of a conductor layer in three ways:

- Height (absolute)
- Delta height (relative)
- Upto (maximum top down)

You can use more than one of these methods per conductor definition, as long as the numbers are valid.

All measurements are in microns, unless otherwise specified.

### Syntax

```
conductor name {field1 value1 ... fieldN valueN }
```

or

```
conductor name {
```

```
    field1 value1
```

```
    ...
```

```
    fieldN valueN
```

```
}
```

You can specify the *field -value* pairs in any order.

This syntax contains the following parameters:

- *name*  
Specifies the name of the conductor layer.
- *min\_spacing value*  
Specifies the minimum spacing permitted by the technology between two conductors (wires) on a layer.
- *min\_width value*  
Specifies the minimum width of a conductor.
- *height value*  
Specifies the layer's height above the substrate.
- *delta\_height value*  
Specifies the layer's height relative to the top of another layer. This parameter must be used with *delta\_layer*.  
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 *E top***

**wire\_top\_enlargement\_r *E top***

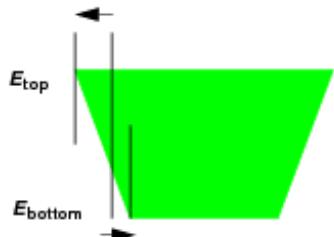
Specifies the enlargement, either positive or negative, of the top edge of the wire. Specify values for both R and C to account for different bias values for R and C. See Figure 42-1.

- **wire\_bottom\_enlargement\_c *E bottom***

**wire\_bottom\_enlargement\_r *E bottom***

Specifies the enlargement, either positive or negative, of the bottom edge of the wire. Specify values for both R and C to account for different bias values for R and C. See Figure 42-1.

**Figure 42-1 Trapezoidal Wire Shape Resulting from Manufacturing Processes**



- **wire\_edge\_enlargement | wire\_edge\_enlargement\_[r|c]**

**wee\_widths *W*1 . . . *W*n**

**wee\_spacings *S*1 . . . *S*m**

`wee_adjustments  $E_{11} \dots E_{1n}$`

.

.

.

`$E_{m1} \dots E_{mn}$`

Models the effect of wire-edge enlargement, if the `wire_edge_enlargement`, `wee_width`, `wee_spacings`, and `wee_adjustments` keywords are specified. The `wee_adjustments` table describes the amount of enlargement applied when certain spacings and widths are observed. For example, the wire is enlarged by  $E_{ij}$  for spacing  $s_i$  and width  $w_j$ . Positive enlargements oversize and negative enlargement undersize the wire. A piecewise constant interpolation is used to obtain enlargements for intermediate spacings and widths. For width/spacings outside of the boundary width/spacing points, the boundary values are used.

`Wire_edge_enlargement_r` and `wire_edge_enlargement_c` can be used if one wants to specify different values for resistance and capacitance.

- `wee_widths  $w_1 \dots w_n$`

Specifies the widths of the wires in the design. Typically, variation is only seen for widths less than 1.5 microns.

- `wee_spacings  $s_1 \dots s_m$`

Specifies the spacings of the wires in the design. Typically, variation is only seen for spacings less than 1.5 microns.

- `wee_adjustments  $E_{11} \dots E_{1n} \dots E_{m1} \dots E_{mn}$`

Specifies the enlargement, either positive or negative, of the wire edge.

See `Wire-Width Values` for information on the wire-width values to use.

#### ***Required Conductor Command Fields***

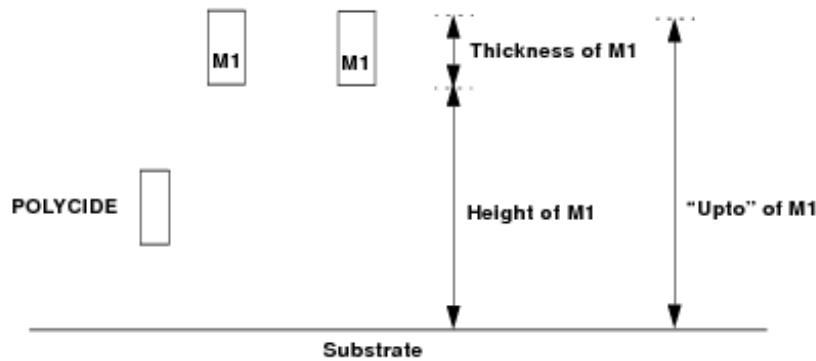
The required fields in this syntax are `min_spacing`, `min_width`, `resistivity`, `gate_forming_layer`,

`min_net_fill_spacing`, `X_fill_fill_spacing`, `Y_fill_fill_spacing`, `unit_fill_region`, and two of the following three parameters:

- `height` (or `delta_height` and `delta_layer`)
- `thickness`
- `upto`

Figure 42-21035744 illustrates these parameters.

## Figure 42-2 Geometric Fields in a Conducting Layer



### Wire-Width Values

You can use the `wire_edge_enlargement` statement with the `wire_top_enlargement` statement or the `wire_bottom_enlargement` statement, or both in the ICT file. If you use the `wire_edge_enlargement` statement with either or both of these statements, the width of the wires defined by `wee_widths` must be biased as follows:

$$\text{drawn\_width} + ((\text{top} + \text{bottom}) / 2)$$

When calculating resistivity as a function of width, you must use the `wire_top_enlargement` and `wire_bottom_enlargement` values to correct the resistance-width pairs. If a table of wire-edge enlargement values is available, the RC extractor uses the wire widths in the table, which always include biasing and wire-edge enlargement. If this table is not available, the resistance is calculated as follows:

$$\rho * L / (\text{drawn\_width} + (\text{top} + \text{bottom}) / 2 + (\text{top} + \text{bottom}) / 2)$$

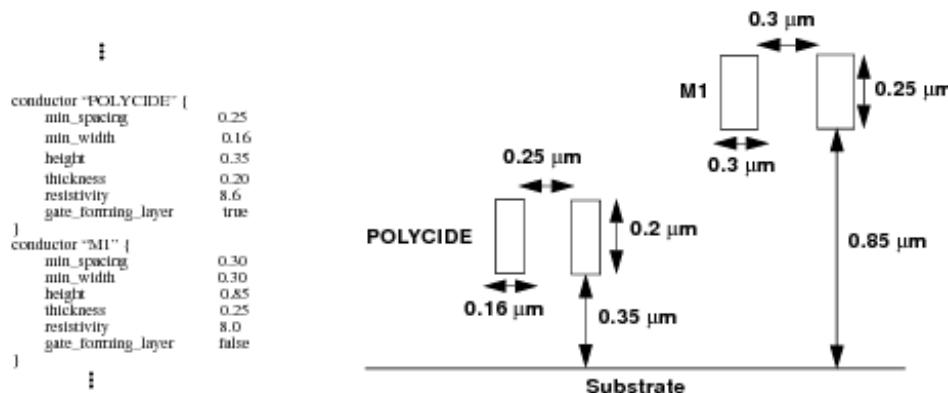
where  $\rho$  is the sheet resistivity.

Wire-width values are used in the following order:

1. Drawn width
2. Biased width
3. Edge-enlarged width
4. Resistivity as a function of width

Figure 42-31035796 illustrates the defining of the conductor layer.

### Figure 42-3 Example Conductor Definition



#### *Example File for Conductor Definition*

```

conductor "POLYCIDE" {

    min_spacing      0.25
    min_width        0.16
    height           0.35
    upto             0.55
    resistivity       8.6
    gate_forming_layer true
}

conductor "M1" {

    min_spacing      0.30

```

```
min_width          0.30
delta_layer        POLYCIDE
delta_height       0.30
thickness          0.25
resistivity         8.0
gate_forming_layer false
wire_top_enlargement 0.01
wire_bottom_enlargement -0.01
wire_edge_enlargement {
    wee_widths      0.18  0.00   0.26   0.30   0.34
    wee_spacings    0.18  0.00   0.26   0.30   0.34   0.38
    wee_adjustments 0.00  0.00  -0.10 -0.10 -0.20
                           0.00  0.00   0.00 -0.10 -0.20
                           0.10  0.00   0.00   0.00 -0.10
                           0.10  0.10   0.00   0.00   0.00
                           0.20  0.20   0.10   0.00   0.00
                           0.30  0.20   0.20   0.10   0.00
}
}
```

## Dielectric

The dielectric command defines dielectric layers.

All measurements are in microns unless otherwise specified.

## Syntax

```
dielectric name {conformal value field1 value1 ... fieldN valueN }
```

or

```
dielectric name {
```

```
    conformal value
```

```
    field1 value1
```

```
    ...
```

```
    fieldN valueN
```

```
}
```

You can specify the *field -value* pairs in any order.

The syntax for planar dielectrics contains the following parameters:

- *name*  
Specifies the name of the dielectric layer.
- *conformal false*  
Specifies that the dielectric is planar. This field is required.
- *height value*  
Specifies the layer's height above the substrate.
- *thickness value*  
Specifies the layer's thickness.
- *dielectric\_constant value*  
Specifies the dielectric constant for this material.
- *delta\_height value*  
Specifies the layer's height relative to the top of another layer.
- *delta\_layer layer\_name*  
Specifies the reference layer for *delta\_height*. It must be a layer that has already been

defined. A reference layer can be a conducting layer or a dielectric layer.

- ***upto value***

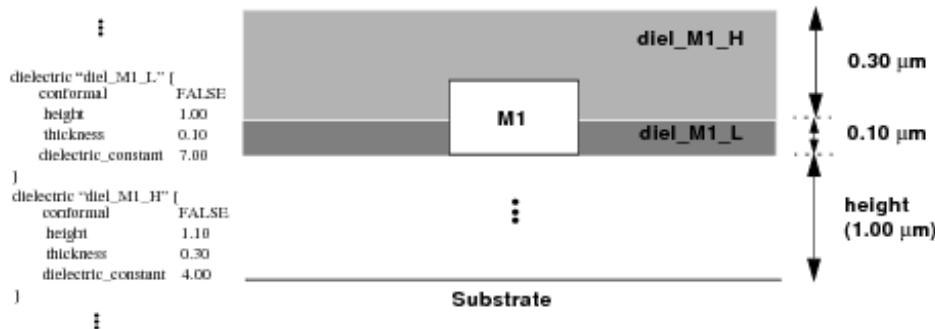
Specifies the layer's top surface height above the substrate. This value is equal to the height plus the thickness. You only need to specify two of the three parameters (*height* (or *{delta\_height, delta\_layer}*), *thickness*, *upto*) to complete the geometrical definition of a dielectric layer.

The required fields in the specification for planar dielectrics are *conformal*, *dielectric\_constant*, and two of the following three parameters:

- *height* (or *{delta\_height and delta\_layer}*)
- *thickness*
- *upto*

Figure 42-41035958 illustrates the planar dielectric syntax.

**Figure 42-4 Planar Dielectric Syntax**



## Passivation

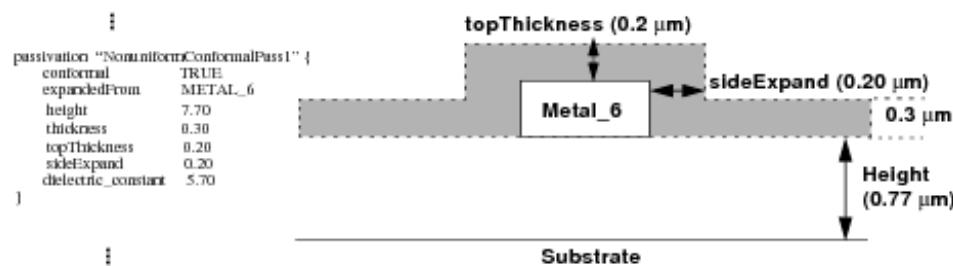
The passivation command defines passivation layers. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

## Syntax

The syntax of this command is the same as that of the *dielectric* command, except that *name* specifies the name of the passivation layer. See *Dielectric* for information on this syntax. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

Figure 42-51036417 illustrates the defining of the passivation layer.

## Figure 42-5 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 layer declarations.

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  
}
```

## ECO Flows

---

This appendix summarizes the variety of Engineering Change Order (ECO) flows possible with Encounter, and outlines the current approach for each flow.

- [Overview](#)
  - [Assumptions](#)
  - [Flows](#)
- [Pre-Mask ECO Changes from a New Verilog File](#)
  - [Preparation](#)
  - 
  - [Steps](#)
  - [Read the new netlist.](#)
- [Pre-Mask ECO Changes from a New DEF File](#)
  - [Preparation](#)
  - [Flow](#)
  - [Steps](#)
- [Pre-Mask ECO Changes from an ECO File](#)
  - [Preparation](#)
  - [Flow](#)
  - [Steps](#)
- [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 Encounter User Guide.

### Assumptions

The ECO flows in this appendix assume the following:

- The old Verilog netlist and the new Verilog netlist have already been uniquified so that no Verilog module is instantiated more than once.
- Your design uses an existing floorplan.

- Your old placement, special routing, and routing information is saved in one of the Encounter formats, DEF, and so forth.

## Flows

This appendix describes various types of ECO flows:

- Pre-Mask ECO Changes from a New Verilog File  
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
8. Add notches (optional)
9. Trim metal fill (optional)

Or,

1. 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 netlist files.

[loadPlan](#) 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.

8. (Optional) Add notches.

```
fillNotch
```

9. (Optional) If the original design contained metal fill, trim the metal fill:

```
trimMetalFill
```

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

Or,

1. Use the [ecoDesign](#) command to perform ECO operations. For example, the following command performs a pre-mask ECO:

```
ecoDesign original.enc.dat top_cell newchip.v
```

2. 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 doing 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 want to make a few changes with known cell placements from a new DEF file.

Examples of this flow include:

- Bringing in an external clock tree after placement
- Bringing in external optimizations such as new buffers or cell resizing
- Bringing in external post-route fixes such as new buffers or cell resizing

## Preparation

Before starting the flow steps, you should have the following files available:

- oldchip.enc
- oldchip.enc.dat/

Create these files by using the [saveDesign](#) command after the previous placement, optimization, and routing steps.

or

- oldchip.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 post-route violations with specific placements required. Any necessary physical cells (+SOURCE DIST) are expected to also be in the new DEF.

You must start with the old Verilog and update the Verilog modules with new ports and nets as required to match the new DEF netlist. You need to make sure the DEF instance names match the expected Verilog names (for example, a new buffer added to the output of instance /i1/i2/i3 should have a name such as /i1/i2/mynewbuf\_i100), otherwise spurious Verilog ports will be created.

## Flow

1. Read the old floorplan/netlist
2. Compare the old netlist to DEF
3. Load the ECO file
4. Write the modified netlist (optional)
5. Read the new placement data

6. Perform incremental or final routing
7. Trim metal fill

## Steps

1. Read the old Verilog netlist, floorplan, and placement information into Encounter by doing one of the following:

[restoreDesign](#) oldchip.enc

or

source ol dchip.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. (Optional) If the original design contained metal fill, trim the metal fill:

[trimMetalFill](#)

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

8. Continue with the normal post-routing flow (analysis, repair, notch-fill, metal-fill, verify, sign-off, and so forth).

## Pre-Mask ECO Changes from an ECO File

In this flow, your design is placed and possibly routed, and you want to make a small number of changes using an ECO file methodology. The changes are done before the masks are made so it is a pre-mask ECO flow, and there is no need to keep the original poly/diffusion patterns.

For example, you might want to apply a small number of late logical changes, but you want to keep as much of the previous placement, optimization, clock tree, and routing to avoid disturbing previous timing/SI optimization and repair.

### Preparation

Before starting the flow steps, you should have the following files available:

- oldchip.enc
- oldchip.enc.dat/

Create these files by using the [saveDesign](#) command after the previous placement, optimization, and routing steps.

or

- oldchip.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

- Read the old netlist
- Load the ECO file
- Write the new netlist (optional)
- Remove filler cells or notch fill (if present)
- Perform incremental placement

- Add filler cells
- Perform incremental or final routing
- Add notches (optional)
- Trim metal fill

## Steps

1. Read the old Verilog netlist, floorplan, and placement information into Encounter.

[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. (Optional) Add notches.

[fillNotch](#)

9. (Optional) If the original design contained metal fill, trim the metal fill:

[trimMetalFill](#)

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

10. Continue with the normal post-routing flow (analysis, repair, add metal fill, notch fill, verify, sign-off, and so forth).

**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 EDI 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.

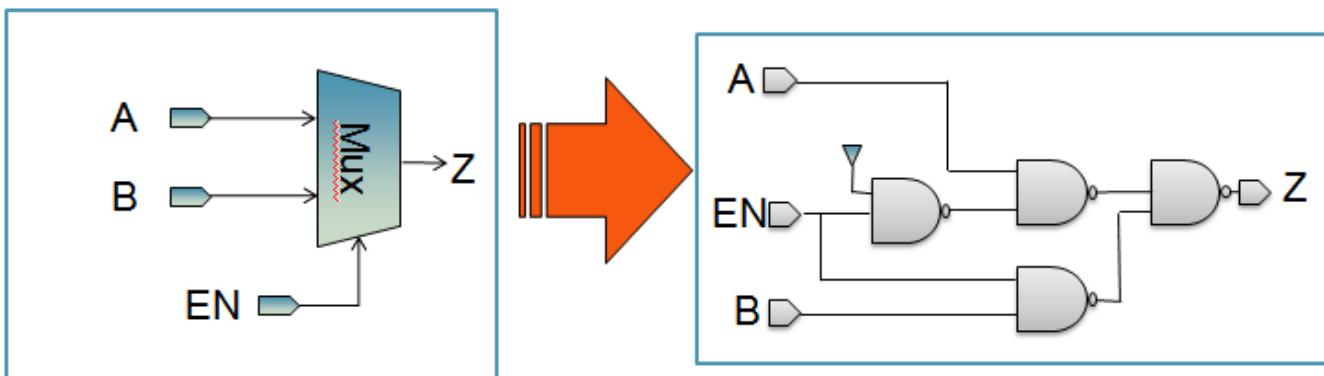
[ecoRemap](#) [-allowConstantTie number] [-useGACells {techSiteName} | -useGAFillerCells {cellName ...}] [-useGASStdCells {cellName ...}] [-rcSpareMapFile fileName]

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`

## Figure A-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`. `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.

#### 10. (Optional) Add notches and metal fill.

[fillNotch](#)

**Note:** Skip this step if you specify freeze metal layers, because this command modifies all layers.

#### 11. (Optional) If the original design contained metal fill, trim the metal fill:

[trimMetalFill](#)

Use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

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.

#### 9. (Optional) Fill notches and add metal fill.

```
fillNotch
```

**Note:** Skip this step if you specify freeze metal layers because this command modifies all layers.

```
trimMetalFill
```

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

## Post-Mask 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 A-2 GA Fillers Insertion for GACells Flow**



Standard cell



Standard cell

#### GA Fillers Insertion for

**Figure A-3**  
**GAFillerCells Flow**

To perform incremental placement:

```
ecoPlace -useSpareCells true (#optional)
```

Unfix all GAFillers in the design

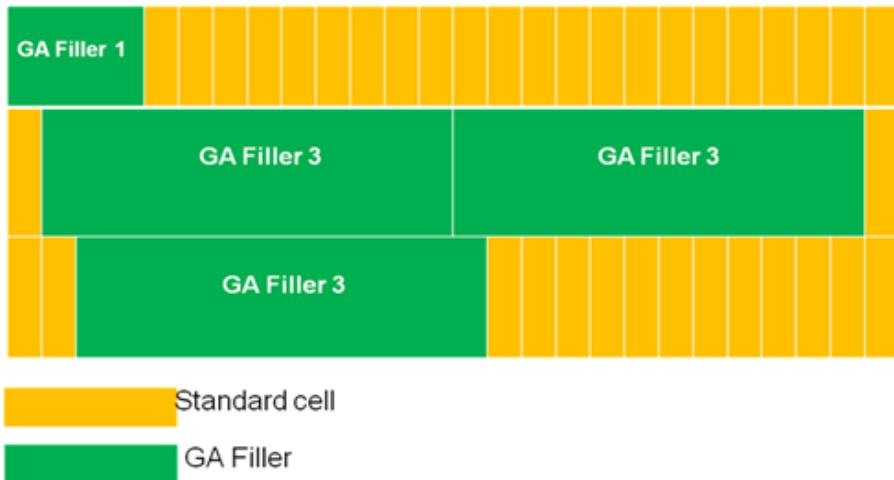
```
ecoPlace -useGAFillerCells {List of GAFillerCells }
```

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 A-3 and A-4 illustrate the replacement of GA Fillers:



**Figure A-4**  
**ecoPlace -useGAFillerCells {GAFILL1 GAFILL2 GAFILL3}**



**Figure A-5**  
**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 Encounter® Digital Implementation System (EDI System) by using the `loadECO` command on the Encounter command line. The following command loads `myDirectivesFile`:

```
loadECO myDirectivesFile
```

 These are ECO directives, not Encounter Tcl commands.

- You can use these directives only in an ECO directives file.
- You cannot use these directives on the Encounter command line.
- You cannot source this file to run the directives.
- You must use the `loadECO` command to read the file.

The names of the directives appear in this appendix in uppercase characters to distinguish them from interactive Encounter commands with the same names; however, the software is case-insensitive.

The ECO File directives do not support Verilog® escape name syntax. For directives that modify or delete existing objects, you cannot specify the name of the object using Verilog escape name syntax.

File format requirements are shown in the section Example ECO File.

The directives are presented in alphabetical order.

- ADDHIERINST
- ADDINST
- ADDMODULEPORT
- ADDNET

- ATTACHMODULEPORT
- ATTACHTERM
- CHANGEINSTNAME
- DELETEBUFFER
- DELETEINST
- DELETEMODULEPORT
- DELETENET
- DETACHMODULEPORT
- DETACHTERM
- INSERTBUFFER
- Example ECO File
- HECO Directives

## 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

*cellName*

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>	Specifies the name of the port to be added.  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.  The new port name can be the same as the existing net name. This port is attached directly to the existing net name if you specify the port direction (scalar port).
input   output   bidi	Specifies whether the port is input, output, or bidirectional.

#### Examples

- The following directive creates an input port p1 on instance i1/i2/i3. The port p1 must be a new port name in instance i1/i2/i3. Instance i1/i2/i3 contains no nets name p1:  
  
ADDMODULEPORT i1/i2/i3 p1 input

- 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.
<code>-moduleBased <i>verilogModule</i></code>	Attaches the terminal to the specified verilog module.
<i>netName</i>	Specifies the name of the net to attach to the terminal. If the net name does not exist, Encounter displays an error message and the directive stops.
<code>-noNewPort</code>	Prohibits Encounter from creating hierarchical ports when it attaches a terminal. If the terminal cannot connect to the net through existing ports, Encounter displays an error message and the directive stops. <i>Default:</i> If you do not specify this parameter, Encounter creates hierarchical ports as needed.
<code>-pin <i>refInstName</i> <i>refPinName</i></code>	Specifies the pin <i>refPinName</i> on instance <i>refInstName</i> connected to the net that Encounter connects to the terminal. You cannot specify the <code>-port</code> parameter if you use this parameter.
<code>-port <i>portName</i></code>	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 <code>-pin</code> parameter if you use this parameter. <i>Default:</i> If you do not specify this parameter, Encounter uses existing ports or creates new hierarchical ports as necessary to connect the terminal to the net.

<i>termName</i>	Specifies the name of the terminal that Encounter connects to the specified net. If the terminal name does not exist, Encounter displays an error message and the directive stops.
-----------------	--

## Examples

- The following directive attaches terminal *in1* of instance *i1/i2/i3* to net *i1/i2/ net26*:

```
ATTACHTERM i1/i2/i3 in1 i1/i2/net26
```

If *i1/i2/i3* does not exist, or if *in1* is not a terminal of *i1/i2/i3*, or if *i1/i2/net26* does not exist, the software displays an error message and the directive stops.

- The following directive attaches terminal *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
```

## 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.

 This directive has been replaced by [ecoDeleteRepeater](#) command.

## Parameters

<i>deleteNetName</i>	Specifies the name of the net connected to the terminal on <i>instName</i> opposite to the terminal that connects to <i>keepNetName</i> . If <i>deleteNetName</i> is not connected to the terminal on <i>instName</i> opposite to the terminal that connects to <i>netName</i> , the software displays an error message and the directive stops. For example, if <i>keepNetName</i> connects to the input of <i>instName</i> , <i>deleteNetName</i> specifies the name of the net connecting to the output. The software uses the <i>deleteNetName</i> for error checking only. When the DELETEBUFFER directive merges the nets, it might detach connections to hierarchical ports, but does not change the direction of hierarchical ports.
----------------------	--

<i>instName</i>	Specifies the name of the buffer instance that the <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 Encounter to delete.

### Parameters

<i>moduleName</i>   `-'	Specifies the module from which you want to delete the port. To specify the top module, enter `-'.
<i>netName</i>	Specifies the name of the net from which you want to delete the port.
<i>portName</i>	Specifies the name of the port to be deleted

### Example

- The following directive deletes port1 from the top-level module:

```
DELETEMODULEPORT - port1
```

## DELETENET

## DELETENET

```
[ -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 Encounter to delete.

### Parameters

<code>-moduleBased <i>verilogModule</i></code>	
	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.
<i>-moduleBased 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.	
- noNewPorts	Specifies that Encounter must not add new ports when inserting the buffer. If you try to insert a buffer that needs a new port, Encounter issues an error message and does not insert the buffer.	
<i>nrBuffer</i>	Specifies the number of buffers attached to the net.	
<i>nrNetTerm</i>	Specifies the original number of terminals attached to the net.	
INST	Specifies a buffer instance. You must specify the <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.
INSTTERM	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.
NETTERM	Specifies the net connection between a driver terminal and the	

	added buffer.	
	<i>instName</i>	Specifies the instance containing the terminal.
	<i>netName</i>	Specifies the net connected to the terminal.
	<i>termName</i>	Specifies the terminal connected to the net.

### Example

- The following directives insert three buffers, b1, b2, and b3, on net0, which originally connects terminal out on instance i0 to receivers i1, i2, and i3. After the three buffers are added, terminal out drives one terminal: b1/in.

```
INSERTBUFFER net0 4 3
```

```
INST b1 buffer 2
INSTTERM in net0
```

```
INSTTERM out net1
INST b2 buffer 2
INSTTERM in net1
INSTTERM out net2
INST b3 buffer 2
INSTTERM in net1
INSTTERM out net3
NETTERM i0 out net0
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, subports, 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
```

# 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. In this release, the Encounter Digital Implementation System (EDI System) is enhanced to support 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.
  - For regular standard cells, the \*WELL layers shapes will be in their own PIN. See Figure 3.

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)**

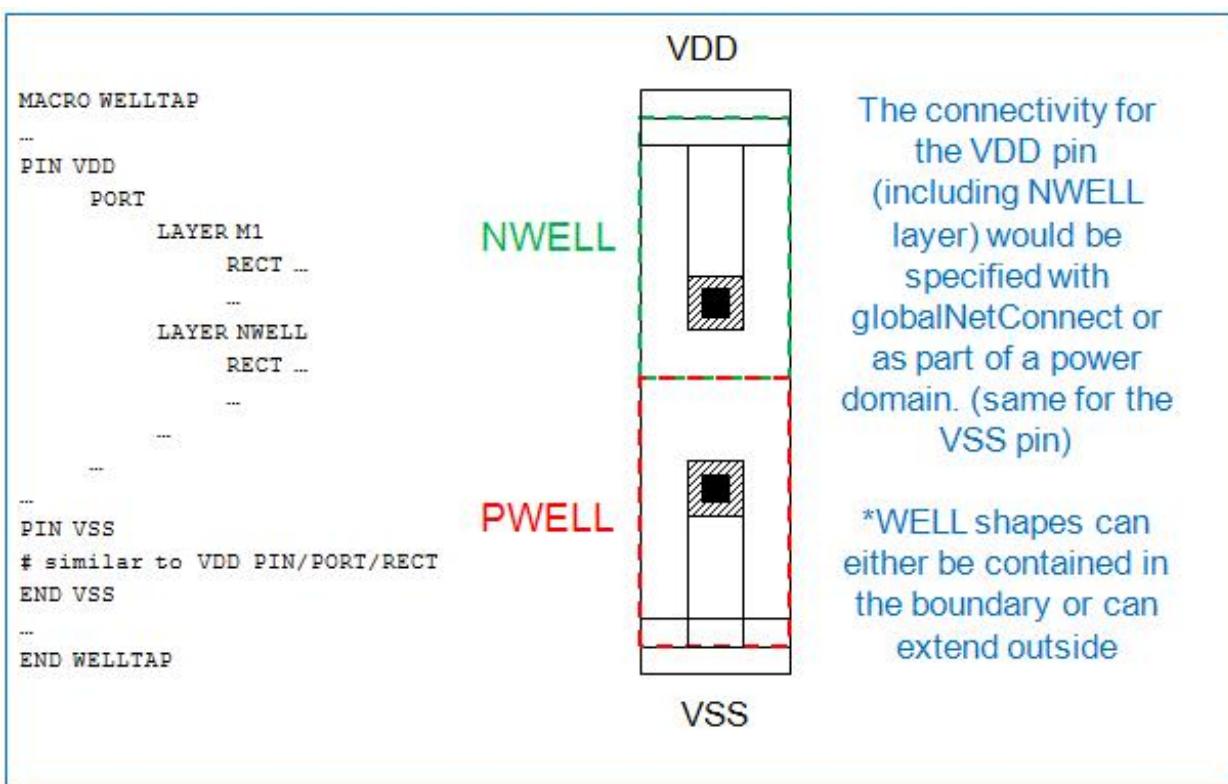
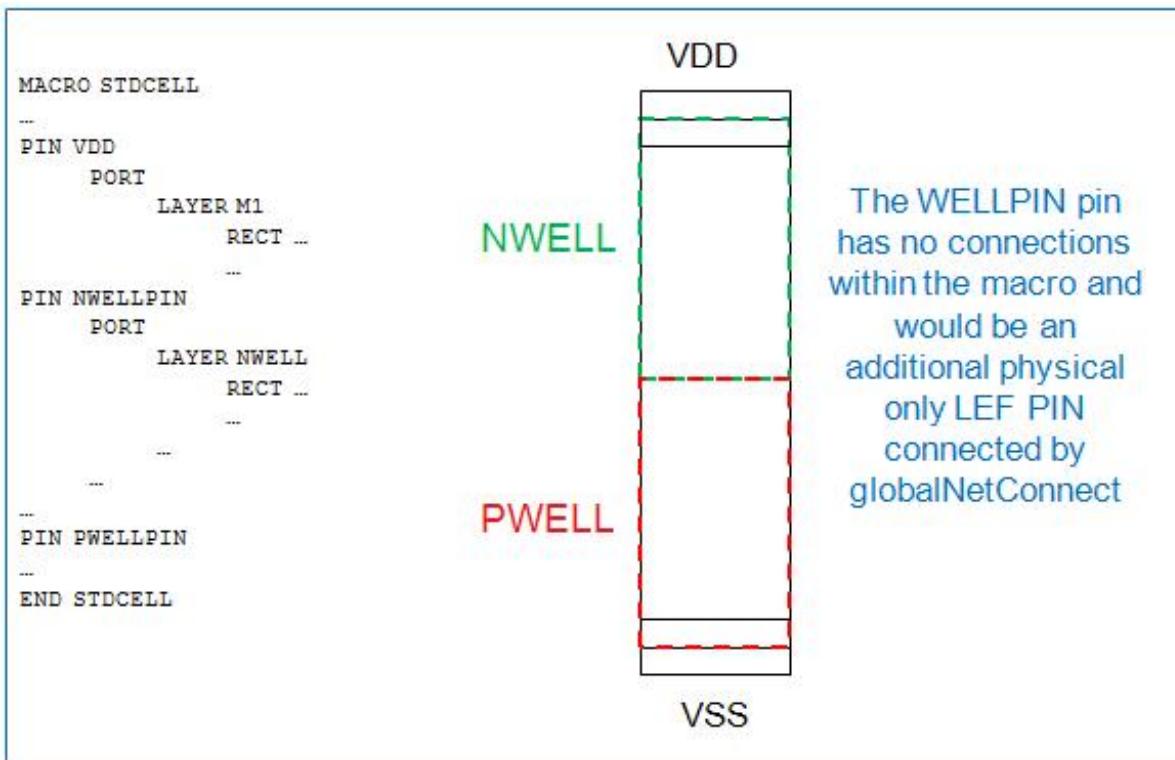


Figure 2: LEF File Example for VBIAS MACRO (CORE)

```
MACRO VBIASCELL
...
PIN VDD
  PORT
    LAYER M1
      RECT ...
    ...
PIN VBIAS
  PORT
    LAYER M1
      RECT ...
    ...
    LAYER NWELL
      RECT ...
    ...
    ...
    ...
PIN PWELLPIN(same as standard cell, unless
there is a n-channel bias in the triple well
case) * Could be part if the VSS pin as well.
END PWELLPIN
END VBIASCELL
```

The connectivity for the VBIAS pin would be specified with globalNetConnect or as part of a power domain

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 *EDI System User Guide* .
- [Connect Global Nets](#) section in the Power Menu chapter in the *EDI System 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 *EDI System 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 *EDI System 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 *EDI System 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 routefollowpin 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)

---

# 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 RouteTypeDef section.
Layer <i>layerName</i>	Specifies the metal layer to be used. The name you specify must be a layer name defined in the LEF file.
Width <i>distance</i>	Specifies the width of the metal layer to be used.
Spacing <i>distance</i>	Specifies the spacing to all other wires in the design. <i>Default:</i> Uses the minimum spacing values specified in the LEF file
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 RouteTypeDef 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.

<i>ClockMesh 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:

<i>Period timeValue</i>	Specifies the clock period (in picoseconds) for the mesh.  <i>Default:</i> 0 (frequency also will default to 0)
<i>MaxPower 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>	<p>Specifies the transition time (in picoseconds) at the root pin.</p> <p><i>Default:</i> 0</p>
MinDelay <i>timeValue</i>	<p>Specifies the minimum phase delay (in picoseconds).</p> <p><b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.</p> <p><i>Default:</i> 0</p>
MaxDelay <i>timeValue</i>	<p>Specifies the maximum phase delay (in picoseconds).</p> <p><b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.</p> <p><i>Default:</i> Largest possible number, based on the machine</p>
MaxSkew <i>timeValue</i>	<p>Specifies the maximum skew between sink pins (in picoseconds).</p> <p><b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.</p> <p><i>Default:</i> 0</p>
MaxBufferTrans <i>timeValue</i>	<p>Specifies the maximum input transition time for buffers (in picoseconds).</p> <p><b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.</p> <p><i>Default:</i> 0</p>
MaxLeafTrans <i>timeValue</i>	<p>Specifies the maximum input transition time for leaf pins (in picoseconds).</p>

**Note:** Currently, the clock mesh tool does not use this value when implementing the mesh.

*Default:* 0

## Tracing and Analysis Scope Section

This section specifies the logical extent of the clock domain, from the root through any mesh drivers to the leaves. Scope is determined by tracing the netlist from the clock root.

The following table describes the entries for the tracing and analysis scope section:

<code>RootPin <i>instanceName</i> / <i>pinName</i></code>	Specifies the complete name of the clock root pin from which the tracing starts.
<code>LeafPin + <i>instanceName</i>/<i>pinName</i> {rising   falling}</code>	Defines the specified input pin as a leaf pin for non-clock type instances. The clock mesh tool will stop tracing at this pin, and skew is analyzed only to this pin. This statement also defines the rising or falling trigger edge for the inputs.  For example:  <code>LeafPin + corem/sync1/reg_3/D rising + corem/sync1/reg_4/D rising</code>
<code>LeafCellPin + <i>cellType</i>/<i>pinName</i> {rising   falling}</code>	Defines the specified input pin as a leaf pin for non-clock type cells. The clock mesh tool will stop tracing at this pin, and skew is analyzed only to this pin. This statement also defines the rising or falling trigger edge for the inputs.  For example:  <code>LeafCellPin</code>

+ AND2X/A rising	
DefaultTrigger {rising   falling}	Specifies the default trigger edge value for leaves that do not have a specified trigger edge value.
AllowGating {true   false}	Specifies whether the clock mesh tool traces through gates. If you specify true, the tool traces until it finds leaf pins. If you specify false, the tool stops at gate inputs.  <i>Default:</i> false

## Mesh Structure Section

This section defines the overall logical attributes of the clock mesh structure.

The following table describes the entries for the mesh structure section:

UseMeshModule [true   false]	Specifies whether the clock mesh tool should create a separate module for the mesh buffers. If you specify true, the tool creates a module at the level of the root net. If you specify false, the tool inserts buffers (flat) at the level of the root net.  <i>Default:</i> true
MeshModule <i>moduleName</i>	Specifies the module name to be used when synthesizing the clock mesh.  <b>Note:</b> The UseMeshModule statement must be set to true in order for this statement to apply.  <i>Default:</i> The clock mesh tool creates a name based on the mesh

	<p>name.</p> <p><code>LoadCell + cellName1 [+ cellName2] ...</code></p>
	<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>
<code>MeshArea l1x l1y urx ury</code>	<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:

<code>GlobalMesh</code>	<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>
<code>MeshDrivePoint {Center   Root   x , y }</code>	<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</li> </ul>

	<p>near the center of the clock mesh structure (center)</p> <ul style="list-style-type: none"> <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 (center). If there is no <code>TopChain</code> section defined, the drive point will be the clock root pin (Root).</li> </ul>	
	Center	Positions the pre-drivers near the center of the clock mesh structure.
	Root	Positions the pre-drivers near the clock root pin.
	$x, y$	Positions the pre-drivers at the specified location.
<b>MeshType {Fishbone   HTreeMesh   MultiSpine}</b>		
	<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>	
<b>PatternTrunkClusterTargetSize <i>n</i></b>		
	<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 : 1</i></p>	

**TrunkOrientation [Horizontal | Vertical]**

Specifies the trunk orientation.

**Note:** You must specify the TrunkOrientation statement if you want to synthesize a clock mesh. If you only want to generate a report, the statement is optional.

**TrunkPlacement {UniformPitch | LoadWeighted}**

Specifies how trunks are placed if pre-defined target locations are not specified using the TargetTrunkLocs statement in the mesh stage section.

UniformPitch

Places the trunks uniformly according to trunk pitch.

LoadWeighted

Places each trunk so that it drives a similar load.

**HTreePattern pattern**

Specifies the order for building horizontal (h) and vertical (v) structures for an HTree + Mesh clock mesh.

You can use h and v in any pattern to specify the order. Specifying an asterisk (\*) causes the tool to repeat alternating structures. For example, specifying h\* is equivalent to specifying hvhv; it is *not* equivalent to specifying hhhh.

**Note:** You only can use this statement when you specify MeshType HTreeMesh.

**TrunkDriveDist {StrictAttach | Uniform | LoadWeighted  
| LoadWeightedMatch}**

Specifies how buffers are placed along driving trunks in the final global mesh stage.

*Default : strictAttach*

	<code>StrictAttach</code>	Places the buffers according to the specified <code>BranchAttachFrequency</code> value.
	<code>Uniform</code>	Places the buffers in an even, uniform distribution along the trunk.  <b>Note:</b> If you specify <code>Uniform</code> , the clock mesh tool ignores the <code>BranchAttachFrequency</code> value.
	<code>LoadWeighted</code>	Places the buffers according to load distribution. The <code>BranchAttachFrequency</code> value limits how closely the buffers can be placed to each other.
	<code>LoadWeightedMatch</code>	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.
<code>PreDriveCTS</code>		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.
<code>Enabled</code> [true   false]		<p>Enables the implementation of the pre-drive structure.</p> <p>When you specify true, the clock mesh tool calls CTS to synthesize the clock tree using the last stage of the global mesh drivers as the leaf pin.</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</p>

pre-drive structure.

*Default:* false

DriveCells + *cellName1* + *cellName2* . . .

Specifies the types of drive cells to use for the pre-drive structure. You must specify at least one cell choice. Multiple drive cells can be specified.

For example:

```
DriveCells
+ CLKBUFX20
+ CLKBUFX16
```

NonDefaultRule *ruleName*

Specifies the LEF NONDEFAULTRULE statement for the router to use when routing the nets in the pre-drive structure.

*Default:* The router uses the default routing rule.

TopPreferlayer *layerName*

Specifies the top preferred metal layer to use for routing the pre-drive structure. The name you specify must be a layer defined in the LEF file.

BottomPrefLayer *layerName*

Specifies the bottom preferred metal layer to use for routing the pre-drive structure. The name you specify must be a layer defined in the LEF file.

DummyBuffer + *cellName1* + *cellName2* . . .

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.

*Default* : true

`optAddBufferLimit n`

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 ratio`

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.

*Default*: 1

`End`

Denotes the end the pre-drive structure definition. The pre-drive structure definition must begin with a `PreDriveCTS` statement, and close with an `End` statement.

`HTreeSplitDrive [true | false]`

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

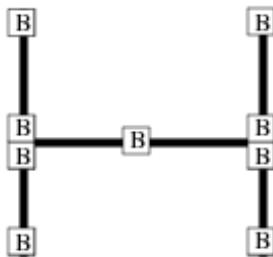
Stage

    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>driveCell1</code>).</p>
<code>NumDriver number</code>	<p>Specifies the number of drivers that should be inserted at the current stage.</p>
<code>X number</code>	<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 <math>X = 1</math>, and the diagram on the right shows four drivers in stage 2, where <math>X=2</math>.</p>
<code>DriveCell cellName</code>	<p>Specifies the type of driver to be used for the stage being defined.</p>

`RouteTypePair type1 type2`

	Specifies the routing types to be used for the stage. Routing types are defined using the <code>RouteTypeDef</code> statement.
<code>NumTrunk number</code>	<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>
<code>TrunkPitch distance</code>	<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>
<code>TrunkAttachFrequency number</code>	<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</p>

trunks is 2, as shown in the diagram on the left (each trunk must have a driver attached to it).

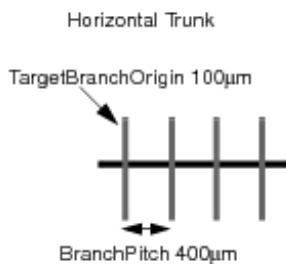
If TrunkAttachmentFrequency is 2, the minimum number of trunks is 3, as shown in the diagram on the right (the middle trunk must not have a driver attached to it). The clock mesh tool generates an error if the specified number of branches does not meet the minimum number of branches.



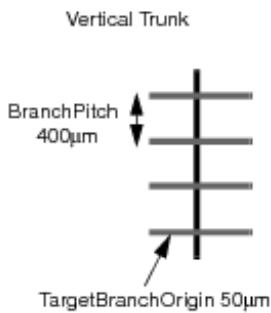
<i>NumBranch</i> <i>number</i>	<p>Specifies the number of branches.</p> <p><i>Default:</i> The clock mesh tools computes a suitable value automatically.</p>
--------------------------------	---

<i>BranchPitch</i> <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 <i>BranchPitch</i> 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>
---	--

<i>TargetBranchOrigin</i> <i>coordinate_in_um</i>	<p>Specifies the absolute location, in microns, of the first branch for a clock mesh.</p> <p><b>Note:</b> The <i>TargetBranchOrigin</i> statement changes the branch origin only. It does not affect the number of branches or the branch pitch.</p> <p>For a horizontal trunk, <i>coordinate_in_um</i> is the x-axis coordinate for the left-most branch, as shown in the following illustration:</p>
---	--



For a vertical trunk, *coordinate\_in\_um* is the y-axis coordinate for the bottom branch, as shown in the following illustration:

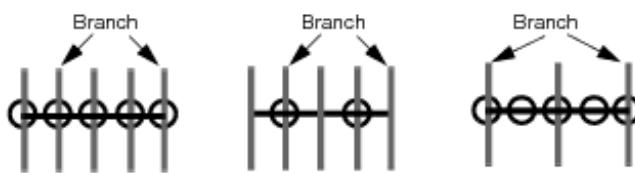


#### BranchAttachFrequency *number*

Controls the frequency with which buffers are attached to branches. You can specify a positive or negative whole number for *number*.

**Note:** The clock mesh tool honors the `BranchAttachFrequency` statement depending on the setting of the `TrunkDriveDist` statement. If you specify `TrunkDriveDist Uniform`, the clock mesh tool ignores the `BranchAttachFrequency` value.

In the following illustration, if `BranchAttachFrequency` is 1, each branch must have a driver attached to it, as shown in the diagram on the left. If `BranchAttachFrequency` is 2, there must be one branch between each driver that does not have a driver attached to it, as shown in the middle diagram. If `BranchAttachFrequency` is -2, there are drivers between the branches that are not attached to branches, as shown in the diagram on the right.



`TargetTrunkLocs + trunkLocation ...`

Positions the trunks at the specified locations.

For example:

`TargetTrunkLocs`

`+ 100um`

`+ 200um`

`+ 300um`

`+ 550um`

You are not required to specify target trunk locations for every stage; however, if you specify a stage, you must list all trunks for that stage.

**Note:** The clock mesh tool considers the `TargetTrunkLocs` statement a soft constraint; it attempts to position the trunks at the specified locations, but might deviate a small distance to find a solution.

`NonDefaultRule ruleName`

Specifies the LEF NONDEFALTRULE statement for the router to use when routing the nets for a particular stage of the global mesh structure.

*Default:* The router uses the default routing rule.

`TopPreferLayer layerName`

Specifies the top preferred metal layer to use for routing for a particular stage of the global mesh structure. The name you specify

	must be a layer defined in the LEF file.
<code>BottomPreferLayer <i>layerName</i></code>	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.
<code>PreferredExtraSpace [ 0 - 3 ]</code>	Specifies the extra space to add around clock wires, when routing the nets in the clockmesh stage.  <i>Default:</i> 0
<code>End</code>	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.
<code>End</code>	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.
<code>ShieldNet <i>netname</i></code>	Specifies the shielding net name for the global mesh.  <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> .

## 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 SpineTemplate section must begin with a SpineTemplate and close with an End statement.
<u>Stage</u>	Denotes the beginning of a stage definition. The Stage section must begin with a Stage and close with an End 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 Stage statement and close with an End statement.
End	Denotes the end of the SpineTemplate definition. The SpineTemplate definition must begin with a SpineTemplate statement, and close with an End 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 Stage section must begin with a Spine and close with an End statement. This section is for specification of the target location of the spine.
<u>Stage</u>	Denotes the beginning of a stage definition. The Stage section must begin with a Stage and close with an End 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 Stage statement, and close with an End statement.
End	Denotes the end of the Spine definition. The spine definition must begin with a SpineTemplate statement, and close with an End 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 stage section must begin with a Stage and close with an End 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 Stage statement, and close with an End 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 stage section must begin with a stage and close with an End 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 : 0</i>
End	Denotes the end of a stage definition. Each stage definition must begin with a Stage statement and close with an End statement.

### ***The Stage Definition for the Spine Section***

The following table describes the Stage definition for the spine section.

Stage	Denotes the beginning of a stage definition. The stage section must begin with a Stage and close with an End statement.  The stage section of a spine section is for specification of parameters such as the number of driver for each spine.
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 stage statement, and close with an End 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 Stage section must begin with a Mesh and close with an End 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.
BranchPitch <i>distanceValue</i>	Specifies the pitch for the branches for a particular

	<p>stage.</p> <p><b>Note:</b> The clock mesh tool considers the BranchPitch statement to be a soft constraint that applies to the final mesh stage. The tool might need to make adjustments to the specified pitch in order to meet physical constraints.</p>
RouteType	Specifies the routing types to be used for the final mesh grid. Routing types are defined using the RouteTypeDef statement. There must be two RouteType specifications for the vertical and horizontal wires of the mesh grid.
End	Denotes the end of a Mesh definition. The mesh definition must begin with a Mesh statement, and close with an End statement.

### 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 : false</i>
MultiPartSpicePartitionLevel <i>num</i>   Global [+ - <i>num</i> ]	
	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 section</i>	
	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 file	
	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: false</i>
DriveCell cellName	Specifies the type of buffer to be used for constructing the top-level chain. Only one cell type can be used for the chain.
NumLevel number	Specifies the number of levels in the chain structure (which is equal to the number of buffers).  You must specify an even number of levels if you use an inverter for the drive cell (DriveCell).

`TargetLocs + driverLocationX driverLocationY ...`

Positions the top chain drivers at the specified locations.

For example:

`TargetLocs`

`+ 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.

**Note:** The clock mesh tool considers the `TargetLocs` statement to be a soft constraint; it attempts to place the drivers at the specified locations, but might deviate a small distance in order to find a solution.

`NonDefaultRule ruleName`

Specifies the LEF `NONDEFAULTRULE` statement for the router to use when routing the nets in the top-level chain.

*Default:* The router uses the default routing rule.

`TopPreferLayer layerName`

Specifies the top preferred metal layer to use for routing the top-level chain buffers. The name you specify must be a layer defined in the LEF file.

`BottomPreferLayer layerName`

Specifies the bottom preferred metal layer to use for routing the top-level chain buffers. The name you specify must be a layer defined in the LEF file.

`PreferredExtraSpace [ 0-3 ]`

Specifies the extra space to add around clock wires, when routing the nets in the top-level chain.

	<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.
ShieldNet <i>netname</i>	<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 synthesizeClockMesh.</p> <p><b>Note:</b> Shield nets using regular wires for shielding top chain and local tree nets are created during routeClockMesh.</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:

LocalTree	Denotes the beginning of the local tree definition. The local tree definition must begin with a LocalTree statement, and close with an End statement.
Enabled [true   false]	<p>Enables the implementation of the local tree.</p> <p><i>Default:</i> false</p>
RootPos {ClusterCenter   OnMesh   NearMeshInCluster}	
	<p>Specifies how to place the buffers.</p> <p><i>Default:</i> clusterCenter</p>

	<b>ClusterCenter</b>	
		Places buffers at the cluster center of gravity.
	<b>OnMesh</b>	Places buffers on or along the nearest mesh trunk or branch. Buffer placement can extend beyond the bounding box of the cluster.
	<b>NearMeshInCluster</b>	
		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.
<b>DriveCells</b> + <i>cellName1</i> + <i>cellName2</i> . . .		<p>Specifies the types of drive cells to use in the local tree. Multiple drive cells can be specified.</p> <p>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>
<b>NonDefaultRule</b> <i>ruleName</i>		<p>Specifies the LEF NONDEFAULTRULE statement for the router to use when routing the local tree.</p> <p><i>Default:</i> The router uses the default routing rule.</p>
<b>TopPreferLayer</b> <i>layerName</i>		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.

<code>BottomPreferLayer <i>layerName</i></code>	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.
<code>PreferredExtraSpace [ 0-3 ]</code>	Specifies the extra space to add around clock wires when routing the local tree.  <i>Default:</i> 0
<code>Cluster</code>	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.
<code>DriveCell <i>cellName</i></code>	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.
<code>TargetLoc <i>driverLocationX</i> <i>driverLocationY</i> ...</code>	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 + instanceName/pinName`

	Specifies the instance pins to use to form a cluster. The set of leaf pins must be consistent (that is, they must all be from the same original domain). Additionally, the same leaf cannot appear in multiple clusters.
End	Denotes the end of the cluster subsection. The cluster subsection must begin with a <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.
<code>ShieldNet netname</code>	<p>Specifies the shielding net name for local tree.</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>

## 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

#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

#Markes the end of the CTS pre-drive structure definition

End
```

```
#Definition for mesh Stage 1
```

```
Stage
```

```
NumDriver 1
```

```
X 4
```

```
DriveCell CLKBUFX20
```

```
RouteTypePair RT1 RT2
```

```
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 Encounter product family, see [Appendix E, "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* \

```

```
-direction down \
-output_power_pin VDD \
-ground VSS \
-valid_location to

#####
##### Always-on High-to-low level shifters
#####

define_level_shifter_cell -cells A0VLH2L* \
-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 \

```

```
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_A0 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_A0 -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:  
  
##### A0 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 "A0"  
#####  
  
create_power_domain -name A0 -default  
  
update_power_domain -name A0 -internal_power_net VDD  
  
  
create_global_connection -domain A0 -net VDD -pins VDD  
create_global_connection -domain A0 -net VSS -pins VSS  
create_global_connection -domain A0 -net VDD_core_SW -pins VDDI  
  
  
#####  
  
### For power domain "CORE"  
#####  
  
create_power_domain -name core -instances CORE_INST \  
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_A0 -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_lvll2h_1p1.lib \
# "
# set lib_ao_bc_extra "\$libdir/technology45_lvll2h_1p3.lib \
# "
# set lib_core_wc_extra "\$libdir/technology45_lvh2l_0p8.lib \
# \$libdir/technology45_headers_0p8.lib \
# \$libdir/technology45_sprg_ao_0p8.lib \
# "
# set lib_core_bc_extra "\$libdir/technology45_lvh2l_1p0.lib \
# \$libdir/technology45_headers_1p0.lib \
# "
```

```
$libdir/technology45_srpq_ao_1p0.lib \
"
set lib_pll_wc "\$libdir/pll_slow.lib \
$libdir/ram_256x16_slow.lib \
$libdir/rom_512x16_slow.lib \
"
set lib_pll_bc "\$libdir/pll_fast.lib \
$libdir/ram_256x16_fast.lib \
$libdir/rom_512x16_fast.lib \
"
#####
#### Define library sets
#####
define_library_set -name ao_wc_0p8 -libraries "$lib_0p8_wc \$lib_ao_wc_extra"
define_library_set -name ao_bc_1p0 -libraries "$lib_1p0_bc \$lib_ao_bc_extra"

define_library_set -name ao_wc_1p1 -libraries "$lib_1p1_wc_base \$lib_ao_wc_extra"
define_library_set -name ao_bc_1p3 -libraries "$lib_1p3_bc_base \$lib_ao_bc_extra"

define_library_set -name core_wc_0p8 -libraries "$lib_0p8_wc \$lib_core_wc_extra"
```

```
define_library_set -name core_bc_1p0 -libraries "$lib_1p0_bc $lib_core_bc_extra"

define_library_set -name pll_wc_1p1 -libraries "$lib_pll_wc"
define_library_set -name pll_bc_1p3 -libraries "$lib_pll_bc"

#####
#### Create operating corners
#####

create_operating_corner -name BC_PVT_A0_L \
-process 1 -temperature 0 -voltage 1.0 \
-library_set ao_bc_1p0

create_operating_corner -name WC_PVT_A0_L \
-process 1 -temperature 125 -voltage 0.8 \
-library_set ao_wc_0p8

create_operating_corner -name BC_PVT_A0_H \
-process 1 -temperature 0 -voltage 1.3 \
-library_set ao_bc_1p3

create_operating_corner -name WC_PVT_A0_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 \
-dominant_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 \
-domin_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 \
-domin_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 \
-domin_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 \
-domin_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 \
-domin_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 \
-domin_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 \
-domin_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_A0

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 A0 to CORE
```

```
create_level_shifter_rule -name LSRULE_H2L -from A0 -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 A0 to CORE
```

```
create_level_shifter_rule -name LSRULE_H2L_A0 -from A0 -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_A0 -cells A0LVLH2L2Y -location to
```

```
#### For signals from PLL to A0
```

```
create_level_shifter_rule -name LSRULE_H2L_PLL -from PLL -to A0
```

```
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		Unsupported
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_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
	-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 Encounter product family, see [Appendix F, "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 \
-input_power_pin VDDL -output_power_pin VDD -ground VSS -valid_location to
```

```
define_power_switch_cell -cells { HEADERHVT1 HEADERHVT2 } \
-stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 -stage_2_enable \
NSLEEPIN2 -stage_2_output NSLEEPOUT2 -type header -power_switchable VDD \
-power TVDD
```

```
define_state_retention_cell -cells { MSSRPG* } -cell_type \
master_slave -clock_pin CP -restore_check !CP -save_function !CP \
-always_on_components { DFF_inst } -power_switchable VDD -power TVDD \
-ground VSS

define_state_retention_cell -cells { BLSPRG* } -cell_type balloon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS
```

```
#-----
# macro models
#-----

#-----
# top design
#-----

set_design top
```

```
create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
```

```
create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-----
# create power domains
#-----

create_power_domain -name PDdefault -default
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifspip \
IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
-external_controlled_shutoff -shutoff_condition io_shutoff_ack
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
IOPADS_INST/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		Unsupported
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_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	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 Encounter product family, see "[Appendix G, 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 { BLSPRG* } -cell_type ballon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS
```

```
#-----
# macro models
#-----

#-----
# top design
#-----

set_design top
```

```
create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_EQ -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
```

```
-average_ir_drop_limit 0
```

```
create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
```

```
-average_ir_drop_limit 0
```

```
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
```

```
-average_ir_drop_limit 0
```

```
create_nominal_condition -name nom_0v81 -voltage 0.81
```

```
create_nominal_condition -name nom_0v72 -voltage 0.72
```

```
#-----
```

```
# create power domains
```

```
#-----
```

```
create_power_domain -name PDdefault -default
```

```
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifspip \
```

```
IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
```

```
-external_controlled_shutoff -shutoff_condition io_shutoff_ack
```

```
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
```

```
IOPADS_INST/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 PDt dsp@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 PDt dsp@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 PDt dsp@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 -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
```

# Cadence-Specific Liberty Extensions



This chapter has some formatting issues after converting to wiki. Pubs R&D is working to fix this.

This appendix describes Cadence-specific extensions to the Synopsys Liberty (.lib) library file.

These extensions allow the storage of the data required by SignalStorm® nanometer delay calculator's effective current source model (ECSM) voltage and current profiles.

This appendix contains information on the following topics:

- [Overview](#)
- [Guidelines For Adding ECSM Extensions](#)
- [Representing ECSM Information in a Library](#)
- [Defining ECSM Extensions in a Library](#)
  - [ecsm\\_waveform Group](#)
  - [ecsm\\_capacitance Group](#)
- [Example](#)

## Overview

SignalStorm nanometer delay calculator uses an advanced cell driver model to represent the effect of non-linear switching waveforms on cell-based interconnect delay calculation and signal integrity. This ECSM model provides an efficient mechanism for storing output current or voltage profiles during active transitions on the circuit.

ECSM models are typically generated by the Encounter Library Characterizer, and stored directly in a SignalStorm database. However, you can also derive ECSM information from a generic characterization flow and store it in a traditional library file using the extensions described in this appendix. These extensions allow the contents of ECSM models to be stored in library files in a way that is compatible with existing data and with other new signal integrity constructs.

**Note:** The presence of ECSM data does not interfere with other uses of the library file.

## Guidelines For Adding ECSM Extensions

- Extensions must be compatible with existing or new library model constructs. They cannot be stored as comments, which could potentially be stripped out by internal or commercial tools that read the library models.
- Extensions must resemble similar library constructs.
- Extensions must use the same context as the equivalent library construct. For example, units must be consistent. If an extension uses a current value and the current unit is defined as 1 milliampere, the values placed in the attribute must be in milliamperes.
- Syntactically correct extensions added to the libraries models must pass through the Synopsys Library Compiler, and should not cause any change in behavior in any tool that uses the compiled models.
- Extensions should be easily extracted by tools that correctly parse library models.
- Storage of the waveforms must be efficient. Appropriate reduction must be performed on the waveforms to a user-specified level of accuracy. The controls provided to adjust the accuracy should enable you to achieve a compromise between the accuracy of the analysis and the size of the model.

## Representing ECSM Information in a Library

ECSM information is stored in the form of output voltage waveforms, which enables the library to include more information about the transition, and to improve the driver model accuracy. Given that the library transition table is based on two types of indexes (input slew rate and output loading capacitance), and the delay model uses a lumped capacitance to represent the observed loading, the following equation computes the output current ( $I_{out}$ ):

$$ECSM(t_1, t_2) = \frac{\int_{t_1}^{t_2} I_{out}(t) dt}{t_2 - t_1} = C_{load} \times \frac{(V_{out}(t_2) - V_{out}(t_1))}{t_2 - t_1}$$

You can use this equation to convert the output voltage waveform to ECSM. To support multiple supply voltage corners, normalize the output voltage numbers from 0.0 to 1.0 of the supply voltage. The effective current should also be normalized to the supply voltage.

Using the output voltage waveform approach, an ECSM extension becomes a table containing voltage over transition times for every input slew and output loading capacitance index combination.

Add this table to the output transition table group using the following format:

```
rise_transition ( template_of_slew_load ) { // fall_transition (template_of_slew_load) {  
  
index_1 : // redefine of slew rate index  
  
index_2 : // redefine of loading index  
  
  
  
ecsm_waveform( name0 ) {  
  
index_1 : "..."; // output voltage sample points  
  
values : "..."; // output voltage sample times  
  
}  
  
ecsm_waveform(name1 ) {  
  
index_1 : "..."; // output voltage sample points  
  
values : "..." ; // output voltage sample times  
  
...  
  
}  
  
}
```

## Defining ECSM Extensions in a Library

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.

1. To include these groups in a library, add the following `define_group` statements to the library.

```
define_group( e fsm _ waveform, rise _ transition);  
  
define_group( e fsm _ waveform, fall _ transition);  
  
define_group( e fsm _ capacitance, rise _ transition);  
  
define_group( e fsm _ capacitance, fall _ transition);  
  
define_group( e fsm _ capacitance, pin);
```

or:

```
define_group( e fsm _ waveform, retain _ rise _ slew); define_group( e fsm _ waveform, retain _ fall _ slew); define_group(  
e fsm _ capacitance, retain _ rise _ slew); define_group( e fsm _ capacitance, retain _ fall _ slew); define_group(  
e fsm _ capacitance, pin);
```

1. Add the following `define` statements to the library to define attributes for the groups:

```
define( index _ 1, e fsm _ waveform, string );  
  
define( values, e fsm _ waveform, string );  
  
define( values, e fsm _ 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( e fsm _ version, library, float );
```

The current version is 1.2.

## **ecsm\_waveform Group**

The `ecsm_waveform` group enables the output voltage waveforms to be specified separately for each index permutation suggested by the slew and load indexes. It also enables each waveform to be sampled at different points so that waveforms that are predominantly linear, can be represented with fewer sample points. The number of `ecsm_waveform` groups must match the number of entries in the `values` attribute of the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

The name of the `ecsm_waveform` group must be an integer enclosed within quotation marks. The minimum value of the integer must be 0, and the maximum value must be a number that is one less than the number of entries in the `values` attribute of the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

Including the `ecsm_waveform` group in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group specifies that all information related to the transition arc also applies to the `ecsm_waveform` group. If not explicitly specified, the lookup template name and the index overrides are inherited from the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. Attributes associated with the `timing` group in which the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group resides are also associated with the `ecsm_waveform` group, including `related_pin`, `timing_sense`, and `timing_type`.

Figure K-1 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" );  ects_m_waveform( "0" ) {  
        index_1 : "0.1, .3, .7,.9";  
  
        values : "1.005n, 1.012n, 1.018n, 1.02n";  
        } ects_m_waveform( "1" ) {  
            index_1 : "0.1, .48, .7,.9";  
  
            values : "1.011n, 1.02n, 1.027n, 1.032n";  
            } ects_m_waveform( "2" ) {  
                index_1 : "0.1, .2, .4, .7,.9";
```

```

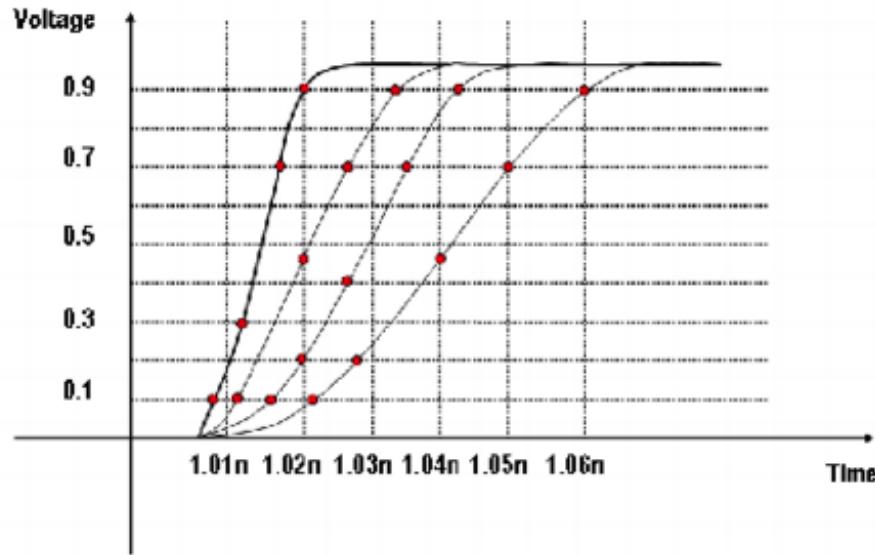
values : "1.015n, 1.02n, 1.028n, 1.035n, 1.07n";
} ecsm_waveform( "3" ) {
index_1 : "0.1, .2, .45, .7,.9";

values : "1.021n, 1.029n, 1.04n, 1.06n, 1.07n";
}

}

```

**Figure K-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" );  
  
    ects_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:

```
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) {  
  
        ectransition(rise) {  
  
            index_1 : "0.1n 0.2n";  
  
            values : "0.01, 0.02";  
  
        }  
  
        ectransition(fall) {  
  
            index_1 : "0.1n 0.2n";  
  
            values : "0.01, 0.02 ";  
  
        }  
  
    }  
  
}
```

Attributes associated with the `timing` group in which the `pin` group resides are also associated with the `ecsm_capacitance` group, including `timing_sense`, and `timing_type`.

### **ecsm\_capacitance Attributes**

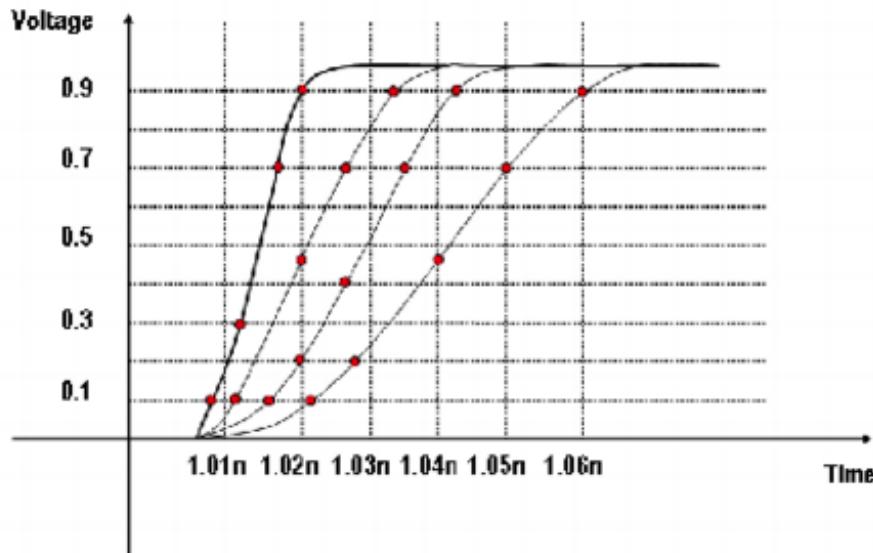
One simple attribute is allowed in an `ecsm_capacitance` group:

- `values`

The values attribute is a comma-separated list of floating-point numbers representing the input capacitance at each specified index point. The values for index\_1 and index\_2 cannot be overridden and are inherited from the rise\_transition, fall\_transition, retain\_rise\_slew or retain\_fall\_slew group. The number of floating point numbers must be equal to the number of entries in the values attribute in the rise\_transition, fall\_transition, retain\_rise\_slew or retain\_fall\_slew group. The capacitive units for this attribute use the same value as the library-level capacitive\_load\_unit attribute.

Figure K-2 shows four sampled output voltage waveforms that begin a transition at 1 nanosecond.

**Figure K-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( ecmw_waveform, rise_transition);
```

```
define_group( ecsm_waveform, fall_transition);
define_group( ecsm_capacitance, rise_transition);
define_group( ecsm_capacitance, fall_transition);
define( ecsm_version, library, float)
define( index_1, ecsm_waveform, string );
define( values, ecsm_waveform, string );
define( index_1, ecsm_capacitance, string );
ecsm_version : 1.2

input_voltage(default) {
    vil : 0.0 ; vih : 1.5 ; vimin : 0.0 ; vimax : 1.5 ;
}

operating_conditions(typ) {
    process : 1.0 ; temperature : 25 ; voltage : 1.5 ;
}

output_voltage(default) {
    vol : 0.0 ; voh : 1.5 ; vomin : 0.0 ; vomax : 1.5 ;
}

lu_table_template(tmg_ntin_oload_5x5) {
    variable_1 : input_net_transition ; variable_2 : total_output_net_capacitance ; index_1("1.0, 2.0, 3.0, 4.0, 5.0");
    index_2("1.0, 2.0, 3.0, 4.0, 5.0");
```

}

```
power_lut_template(pwr_tin_oload_3x3) {  
  
variable_1 : input_transition_time ; variable_2 : total_output_net_capacitance ; index_1("1.0, 2.0, 3.0"); index_2("1.0,  
2.0, 3.0");  
  
}  
  
  
cell(INV1) {  
  
area : 1.0 ; cell_leakage_power : 4.467483e-06 ;  
  
pin(A) { capacitance : 0.00230301522 ; direction : input ; }  
  
  
pin(Z) { direction : output ; function : "IA" ; 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");  
  
}  
} timing0 {  
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, \  
             1444
```

```
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" ;  
  
}  
  
}  
  
}  
  
}
```