Innovus User Guide

Product Version 22.10
September 2022

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

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.

Contents

About This Manual

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

See *Innovus Stylus Common UI User Guide* for the Innovus Stylus user interface.

Audience

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

How This Manual Is Organized

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

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

- Design Implementation Flow
- Hierarchical and Prototyping Flow

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

- Introduction and Setup Guide
- Design Import and Export Capabilities
- Design Planning Capabilities
- Design Implementation Capabilities

- Hierarchical Flow Capabilities
- Prototyping Flow Capabilities
- Analysis Capabilities
- Verification Capabilities
- ECOs and Interactive Design Editing
- Design Methodology for 3D IC with Through Silicon Via
- Syntax and Scripts

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: `set_message –severity info`
`text`	Indicates information for which you must substitute a name or value. In the following example, you must substitute the name of a specific file for *`file_name`*: `compare_release -out_file` *`file_name`*
text	Indicates the following: - Text found in the graphical user interface (GUI), including form names, button labels, and field names - Terms that are new to the manual, are the subject of discussion, or need special emphasis - Titles of manuals
`[ ]`	Indicates optional arguments. In the following example, you can specify none, one, or both of the bracketed arguments: `command [-arg1] [arg2 value]`
`[ | ]`	Indicates an optional choice from a mutually exclusive list. In the following example, you can specify any of the arguments or none of the arguments, but you cannot specify more than one: `command [arg1 | arg2 | arg3 | arg4]`
`{ | }`	Indicates a required choice from a mutually exclusive list. In the following example, you must specify one, and only one, of the arguments: `command {arg1 | arg2 | arg3}`

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

Related Documents

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

Innovus Product Documentation

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

- *Innovus Text Command Reference*
  Describes the Innovus text commands, including syntax and examples.
- *Innovus Menu Reference*
  Provides information specific to the forms and commands available from the Innovus graphical user interface.
- *Innovus Database Access Command Reference*
  Lists all of the Innovus database access commands and provides a brief description of syntax and usage.
- *Innovus Foundation Flow Guide*
  Describes how to use the scripts that represent the recommended implementation flows for digital timing closure with the Innovus software.
- *Mixed Signal Interoperability Guide*
  Describes the digital mixed-signal flow.
- README file
  Contains installation, compatibility, and other prerequisite information, including a list of Cadence Change Requests (CCRs) that were resolved in this release. You can read this file online at downloads.cadence.com.

Stylus Common UI Documentation

- *Innovus Stylus Common UI Migration Guide*
  Provides information on migrating from legacy to the Stylus Common UI version of the Innovus software.
- *Innovus Stylus Common UI User Guide*
  Describes how to install and configure the Innovus Stylus software, and provides strategies for implementing digital integrated circuits.
- *What's New in Innovus Stylus Common UI*
  Provides information about new and changed features in this release of the Innovus family of products.
- *Innovus Stylus Common UI Text Reference Manual*
  Describes the Innovus Stylus Common UI text commands, including syntax and examples.
- *Innovus Stylus Common UI Menu Reference*
  Provides information specific to the forms and commands available from the Innovus Stylus

Common UI graphical user interface.

- *Stylus Common UI Database Object Information*
  Provides information about Stylus Common UI database objects.
- *Innovus Stylus Common UI Mixed Signal (MS) Interoperability Guide*
  Describes the digital mixed-signal flow using Innovus Stylus Common UI.

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

Additional Learning Resources

Online Trainings

Cadence offers several training courses on Innovus. For a visual guide of the courses available in the Digital Design and Signoff space, click the links provided on our Learning Map page.

You can also write to training_enroll@cadence.com.

Videos

The Video Library on Cadence Online Support provides a comprehensive list of videos on various Cadence products. Click the link below to view a list of available Innovus videos:

- Innovus Video Library

1

Introduction and Setup Guide

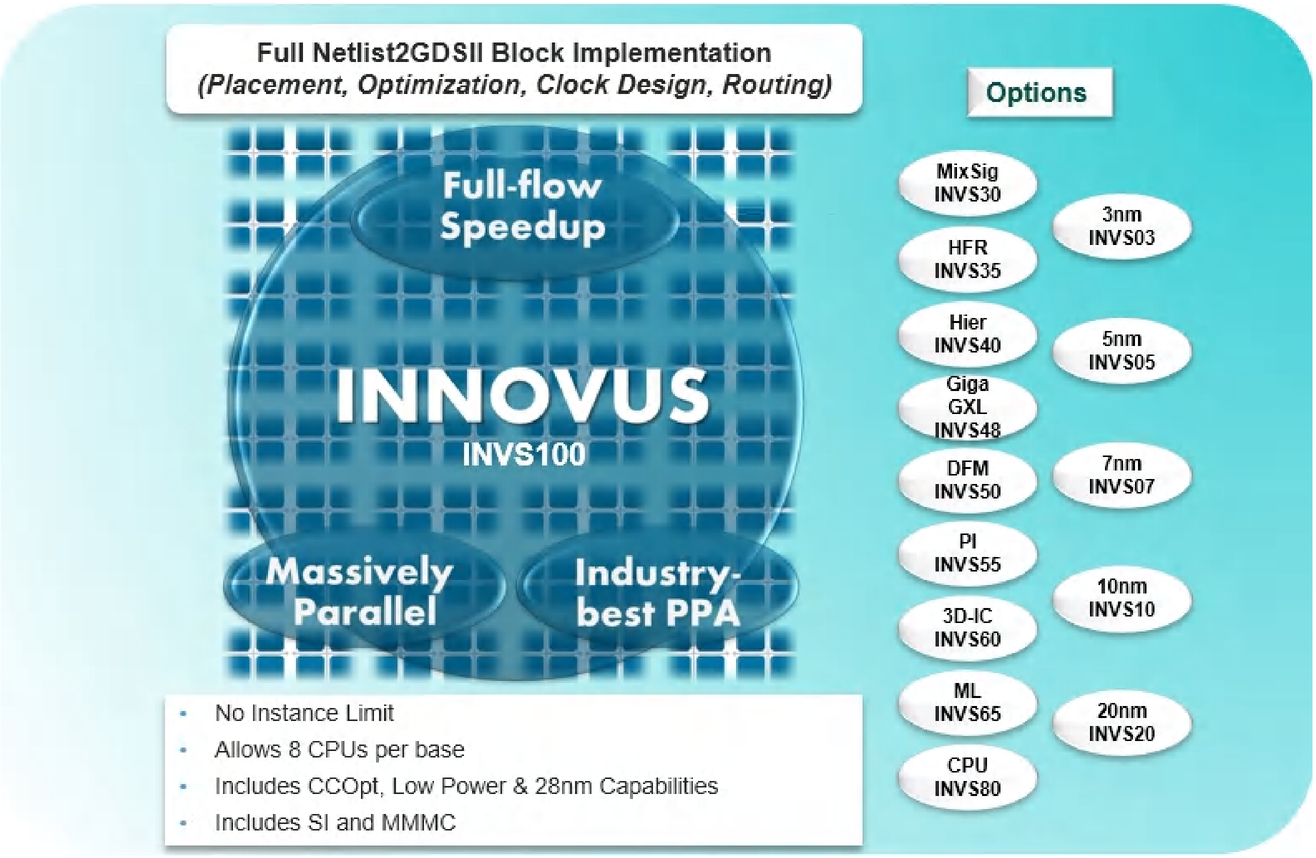
Product and Licensing Information

Product Packages and Options

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

- Innovus Product Packaging
- Innovus Basic Product Packaging
- Virtuoso Digital Implementation Product Packaging
- First Encounter Product Packaging
- Product Options

Innovus Product Packaging

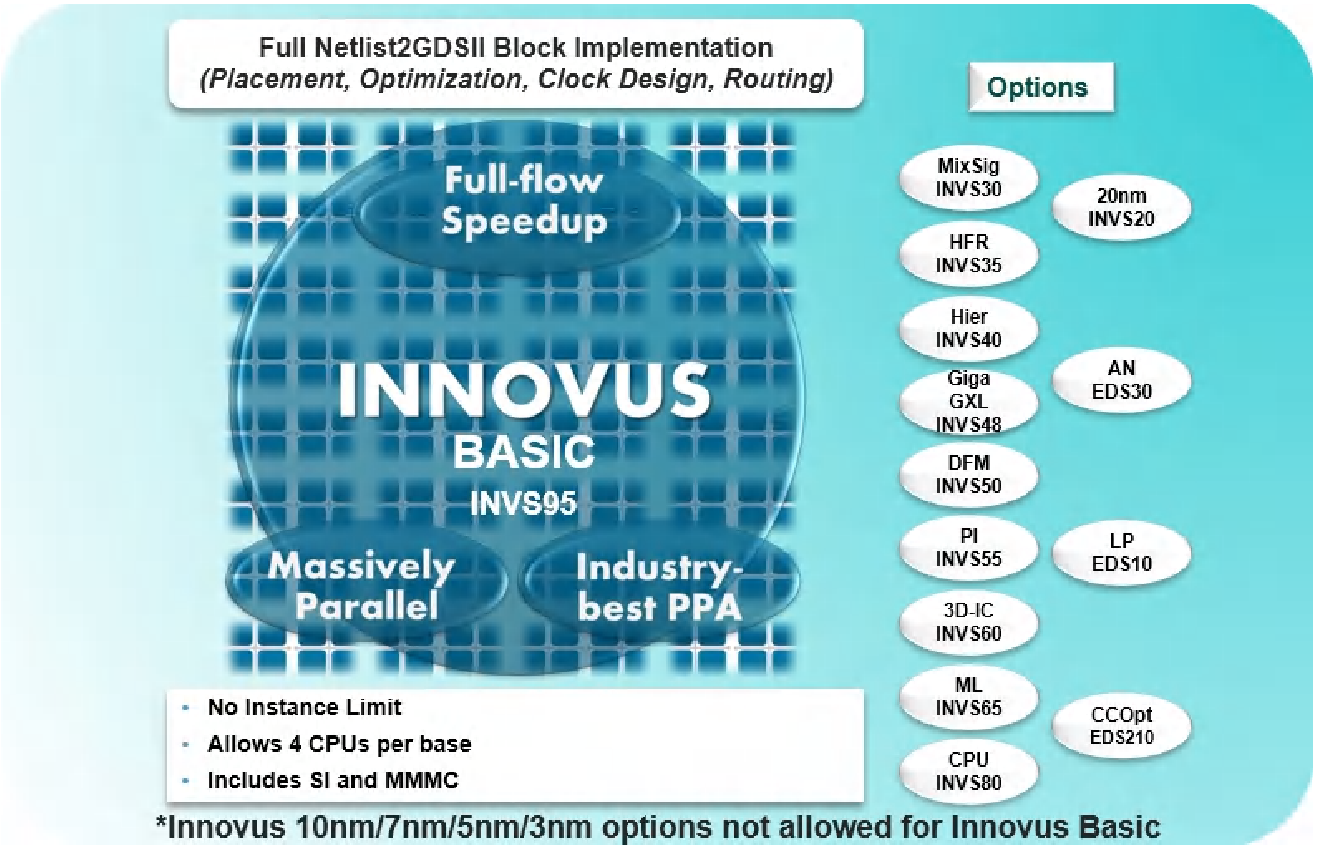


To start the product, type `innovus` on the UNIX command line. For more information on using this command, see "Starting the Software" section in the Getting Started chapter in the *User Guide* and the `innovus` command description in the *Text Command Reference*.

Key Features in INNOVUS (INVS100)

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

Innovus Basic Product Packaging



Full Netlist2GDSII Block Implementation
(Placement, Optimization, Clock Design, Routing)

Full-flow Speedup

INNOVUS
BASIC
INVS95

Massively Parallel

Industry-best PPA

- No Instance Limit
- Allows 4 CPUs per base
- Includes SI and MMMC

Options

MixSig INVS30

20nm INVS20

HFR INVS35

Hier INVS40

AN EDS30

Giga GXL INVS48

DFM INVS50

PI INVS55

LP EDS10

3D-IC INVS60

ML INVS65

CCOpt EDS210

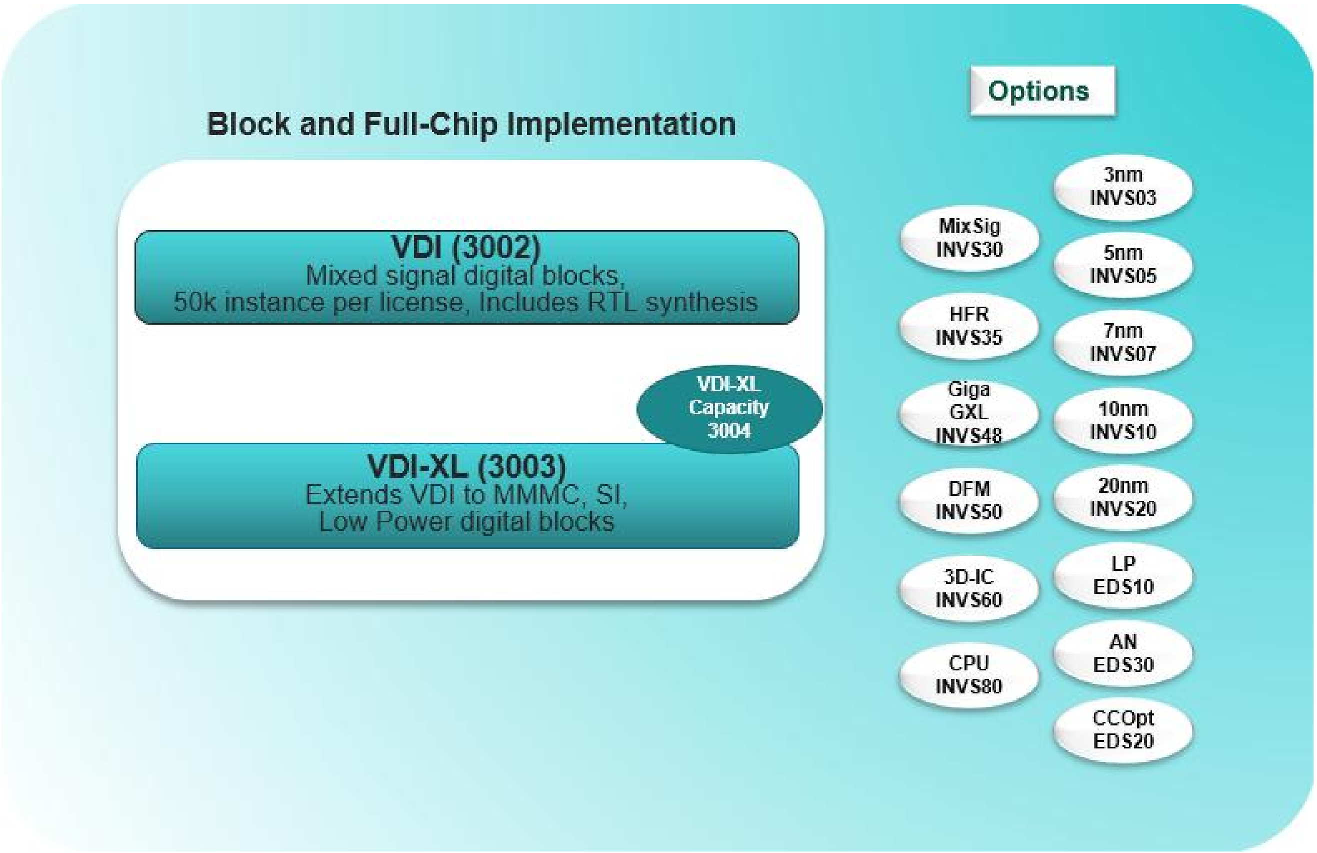
CPU INVS80

*Innovus 10nm/7nm/5nm/3nm options not allowed for Innovus Basic

Key Features in INNOVUS BASIC (INVS95)

- Full Netlist2GDSII Block Implementation (*Placement, Optimization, Clock Design, Routing*)
- Supports parallel multi-threaded and distributed computing architecture with 4 CPUs per base
- Supports process nodes 16 and above (does not support 10nm capabilities)
- New GigaPlace solver-based placement technology that is slack-driven and topology-/pin access-/color-aware, enabling optimal pipeline placement, wirelength, utilization and PPA
- Advanced timing and power-driven optimization that is multi-threaded and layer aware, reducing dynamic and leakage power with optimal performance
- Allows Low Power, Advanced Node and CCOpt capabilities to be added through options
- Unique concurrent clock and datapath optimization that includes automated hybrid H-tree generation, enhancing cross-corner variability and driving maximum performance with reduced power
- Slack-driven routing with track-aware timing optimization that tackles signal integrity and improves post-route QOR
- Full-flow multi-objective technology enables concurrent electrical and physical optimization for best PPA
- Tight integration with signoff Tempus, Quantus and Voltus technologies to accurately model parasitics, timing, signal, and power integrity issues and converge smoothly to signoff
- Mixed-signal design seamless Interoperability flow through integration to Virtuoso® and our custom/analog tools
- Hierarchical model creation (ILM, black box) and top-level assembly and signoff optimization
- Complete flip-chip support, 45 degree routing (Area/peripheral IO support)
- Global Timing Debug with links to Conformal Constraint Designer, Global Clock Debug, Global Power Debug

Virtuoso Digital Implementation Product Packaging



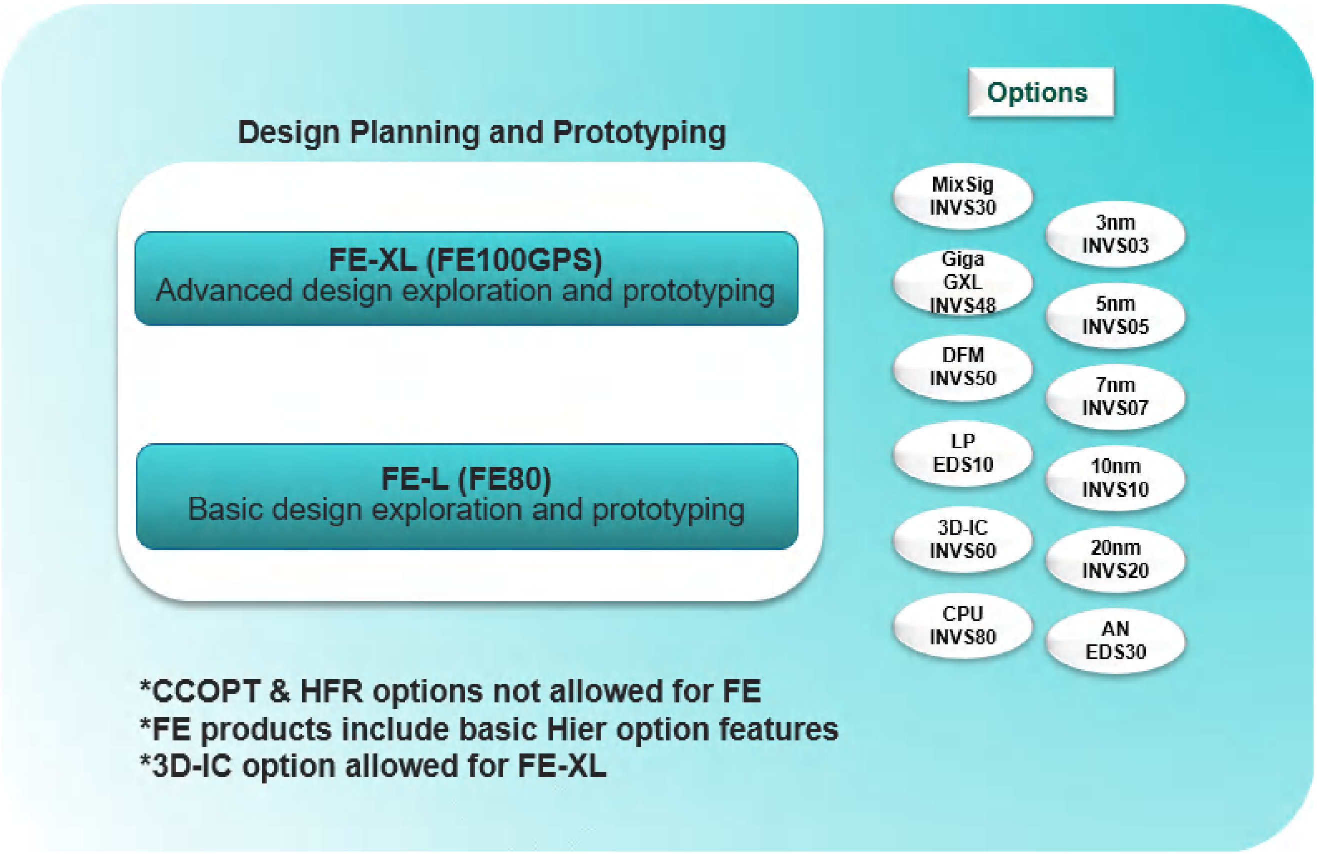
Key Features in VDI (3002)

- Genus Synthesis and N2N (Netlist to Netlist) optimization
    - 50K instances per license (Up to two VDI licenses can be stacked up to 100K instances)
- Block-level floorplanning, wire editing, clock tree synthesis, routing, optimization and design closure
    - 50K instances per license (multiple VDI licenses can be stacked up to 100K instances)
- Hierarchical model creation (ILM, black box)
- Automatic floorplan synthesis, automatic macro placement, wire editing
- SMART routing (Signal integrity, Manufacturing Aware, Routability, and Timing), metal fill, verify DRC/LVS, ECOs
- Multi-Vth optimization, clock gating for low power, early rail analysis using signoff power analysis engine
- GDSII, Oasis, and OpenAccess (OA) support and interoperability
- Implementation timing and delay calculation, Global Timing Debug (GTD) with links to Conformal Constraint Designer (CCD)
    - Signoff timing enabled with VDS-T or Tempus license

Key Features in VDI-XL (3003)

- Capacity limited to 50K instances per license, stackable up to 100K instances with two licenses
- Capacity can be increased to 300K using the VDI-XL Capacity option
- All VDI features
- Top-level assembly and timing analysis
- Multi-mode and multi-corner support
- SI analysis and fixing
    - Signoff timing enabled with VDS-T or Tempus license
- Low power synthesis capabilities from the Genus Low Power option
- Low power implementation capability from the Innovus Low Power option

First Encounter Product Packaging



Design Planning and Prototyping

FE-XL (FE100GPS)
Advanced design exploration and prototyping

FE-L (FE80)
Basic design exploration and prototyping

Options

MixSig INVS30

Giga GXL INVS48

DFM INVS50

LP EDS10

3D-IC INVS60

CPU INVS80

3nm INVS03

5nm INVS05

7nm INVS07

10nm INVS10

20nm INVS20

AN EDS30

*CCOPT & HFR options not allowed for FE
*FE products include basic Hier option features
*3D-IC option allowed for FE-XL

Key Features in FE-L (FE80)

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

Key Features in FE-XL (FE100GPS)

- All FE-L features included
- Netlist-to-netlist optimization and advanced netlist restructuring
- Automatic floorplan synthesis, automatic macro placement
- Placement and optimization
- Clock Tree Synthesis
- Complete MMMC support including MMMC-ILM top-level assembly
- Power domain floorplanning, power driven place, Multi-Vth opt, DVFS, PSO, and Global Power Debug
- Tri-lib support for multi-voltage, multi-temperature delay calc
- Advanced flip-chip support with 45 degree RDL routing (Area/peripheral IO support)
- Timing Aware ECO/spare-cell remapping

Product Options

The product options provide 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. The following product options are available with this release of the software:

3nm	**3nm Option (INVS03)** • Enables 3nm node features • Enables all higher node (5/7/10/14/16/20nm and above) features as well
5nm	**5nm Option (INVS05)** • Enables 5nm node features • Enables all higher node (7/10/14/16/20nm and above) features as well

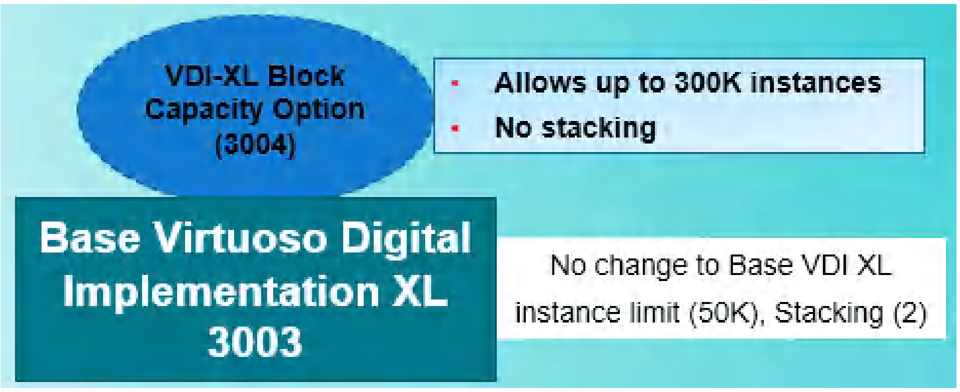
YIELD 7nm	**7nm Option (INVS07)** • Enables 7nm node features • Placement support for new pin access rules • Router support for color dependent “wire cutting” or “metal trim rules” • New metal cutting rules for pin extension, signal, and PG routing • Compact base layer abutment rule support • New PG structure rules with specialized cuts • Layer Additions to DEF, GDS • Enables all higher node (10/14/16/20nm and above) features as well
YIELD 10nm	**10nm Option (INVS10)** • Enables 10nm node features • Supports self-aligned double patterning (SADP) • Supports triple mask/color (TPT) on the first metal layer and cut layers • Enables all higher node (14/16/20nm and above) features as well
YIELD 20nm	**20nm Option (INVS20)** • Enables 20/22/16/14nm node features • FinFET support • Double patterning-correct placement and optimization • NanoRoute double patterning-correct routing for all rules • Rules, colorization for standard cells/hard macros • Real-time colorization for uncolored metal shapes • Implementation-stage DPT conflict/DRC checks • Enables all higher node (28/32nm and above) features as well

MS	**Mixed Signal Option (INVS30)** • Mixed Signal floorplanning and interoperability with Virtuoso, and ability to automatically create detailed abstracts for routing • Enables accurate digital timing analysis using the full timing model • Full automation of digital block implementation using the VDI interface from Virtuoso • Ability to interoperate P-cell submaster geometries in Innovus through P-cell cache • Interoperability of Width Spacing Patterns (WSPs) between Innovus and Virtuoso • Hierarchical propagation of integrated routing constraints through pull and push • Support for creating and modifying Mixed Signal routing constraints • Support for launching Virtuoso Space-Based Router from Innovus • Integrated constraint verification using PVS • Ability to populate Virtuoso layout canvas with interoperability violation markers
HF ROUTE	**High Frequency Route Option (INVS35)** • Enables structured routing for high frequency nets • Parallel and coaxial shielding support • Differential pair routing • Routing with a length and resistance constraints • Bus routing • NDR with combination of constraints

HIER	**Hierarchical Option (INVS40)** • Enables all Hierarchical design capabilities • Partitioning and Budgeting features supported • FlexModel design exploration and prototyping • FlexILM for hierarchical implementation • Partition-in-partition capability • Early floorplanning and exploration using SoC Architecture Information (SAI) • psPM.model creation and usage • No instance limit restrictions
EHFS	**Early Hierarchical Floorplan Synthesis Option (INVS46)** • Functional block modeling by SAI or quick RTL-generated models • Timing-driven module clustering and placement • Hierarchical area compaction and minimization • Shape generation considering embedded macros packing • Channel creation and various placement constraint support • Full-chip congestion-aware early feedthrough estimation • Floorplan geometry checking by UFC (or tCIC)
GigaPlace GXL	**GigaPlace GXL Option (INVS48)** • Allows concurrent standard cell and macro placement for quick floorplan generation • Automatically creates initial floorplan from RTL as part of iSpatial flow • Legalizes and aligns macro cells during placement • Enables automated 3D-IC memory on logic flow

DFM	**DFM Option (INVS50)** • Litho hotspot analysis and hotspot detection • Samsung Process Hotspot Repair (PHR) hotspot detection for 32nm and below • GF DRC and hotspot detection for 40nm and below • User-defined and/or foundry-defined pattern search • Limited model-based CMP analysis and hotspot detection • Used with LPA120 to apply local optimization and guideline-based fixing
PI	**Power Integrity Option (INVS55)** • IR drop and EM aware placement to spread high power density hot spots to reduce IR drop • IR drop fixing with local PG stripe/via addition in hotspots • Power grid optimization to free up routing resources for better PPA
Automotive	**Automotive Option (INVS56)** • Enables the automatic implementation and checking of the safety mechanisms defined in the Unified Safety Format (USF) file • Supports triple modular redundancy (TMR) insertion, placement separation, and clock isolation • Supports dual-core lockstep (DCLS) placement and routing separation and clock isolation
3D-IC	**3D-IC Option (INVS60)** • Stacked die design capabilities • Applies to implementation and signoff
ML-I	**Machine Learning Implementation Option (INVS65)** • Enables Innovus Machine Learning (ML) implementation flow • At least one INVS65 license is required

ML-T	**Machine Learning Training Option (INVS66)** • Enables the generation of custom ML-based training models and testing • At least one INVS66 license is required • Multiple training licenses can be purchased to run training in parallel
CPU	**CPU Accelerator Option (INVS80)** • Multi-CPU acceleration with 8 additional CPUs throughout the flow
LP	**Low Power Option (EDS10)** • Power-intent driven low power methodology with CPF/IEEE1801 specification • End-to-end multi-supply voltage (MSV) support • Power domain-aware automatic floorplan synthesis and routing • Dynamic Voltage Frequency Scaling support • Power shut-off and power switch prototyping • State Retention Power Gating support • Always-on buffer and Dual-flop support • Hierarchical Macro Model support
AN	**Advanced Node Option (EDS30)** • 32/28nm support in routing and verify • Context-driven placement • Structured datapath support • DFM/DFY optimization for wires, cell, vias • Litho-aware routing with prevention and fixing • 1-D routing support • Clock mesh and hybrid implementation

CCOpt	**CCOpt Option (EDS210)** • Simultaneous clock tree synthesis and physical optimization • True useful-skew and Multi-mode multi-corner timing including OCV derates • Worst chain design closure with time borrowing across the delay chain • FlexH driven Hybrid H-tree CTS
VDI CAP	**VDI-XL Block Capacity Option (3004)**  VDI-XL Block Capacity Option (3004) • Allows up to 300K instances • No stacking Base Virtuoso Digital Implementation XL 3003 No change to Base VDI XL instance limit (50K), Stacking (2) License check-out behavior is as follows: • Default ◦ Under 50k instances: vdixl checked out ◦ Between 50 and 100k instances: Second vdixl checked out, as in previous releases. If not found, new vdixl_capacity_opt checked out ◦ Between 100 and 300k instances: One vdixl base plus one vdixl_capacity_opt checked out ◦ Greater than 300k instances: Errors out with appropriate message • Explicit (at startup) ◦ `-lic_startup vdixl -lic_startup_options vdixl_capacity`

Licensing Information

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.
- **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.
- **License Mgr & Daemon** - License Manager (lmgrd) and the Cadence license Daemon (cdslmd) should be at 11.16.4.0 or higher. This can be checked with `lmgrd -v` and `cdslmd -v`. If the daemon versions have not been updated, Innovus may fail to check out a license. In this case, you will need to restart the license server using the latest `cdslmd` and `lmgrd` versions included in the release tree.

For information on startup options, refer to `innovus -help` in the software or the `innovus` command description in the *Text Command Reference*.

For information on the managing licenses for product options, see `setLicenseCheck` in the *Text Command Reference*.

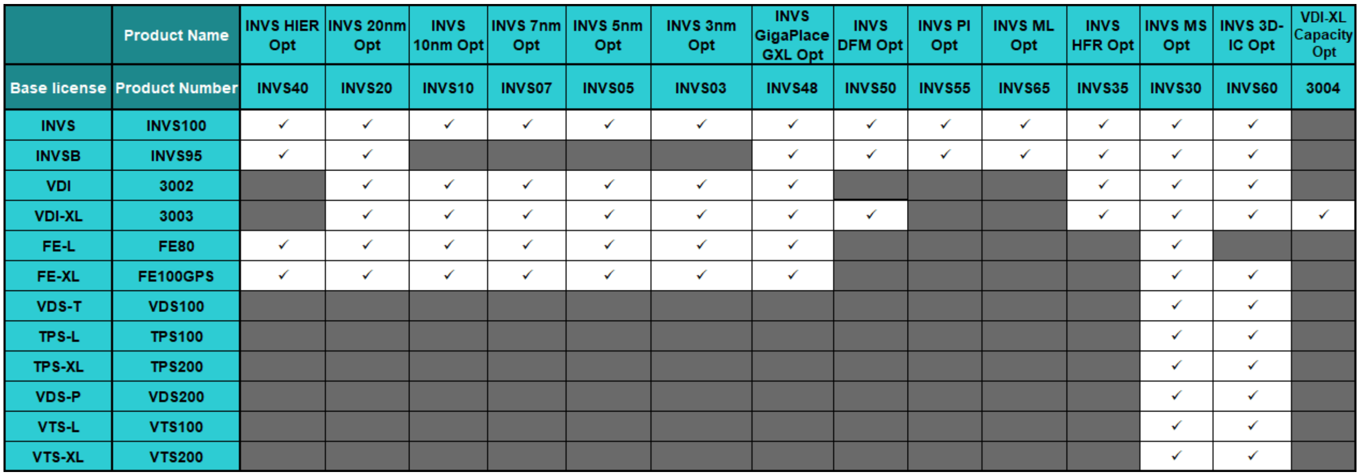
Dynamic Checkout Matrix

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

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

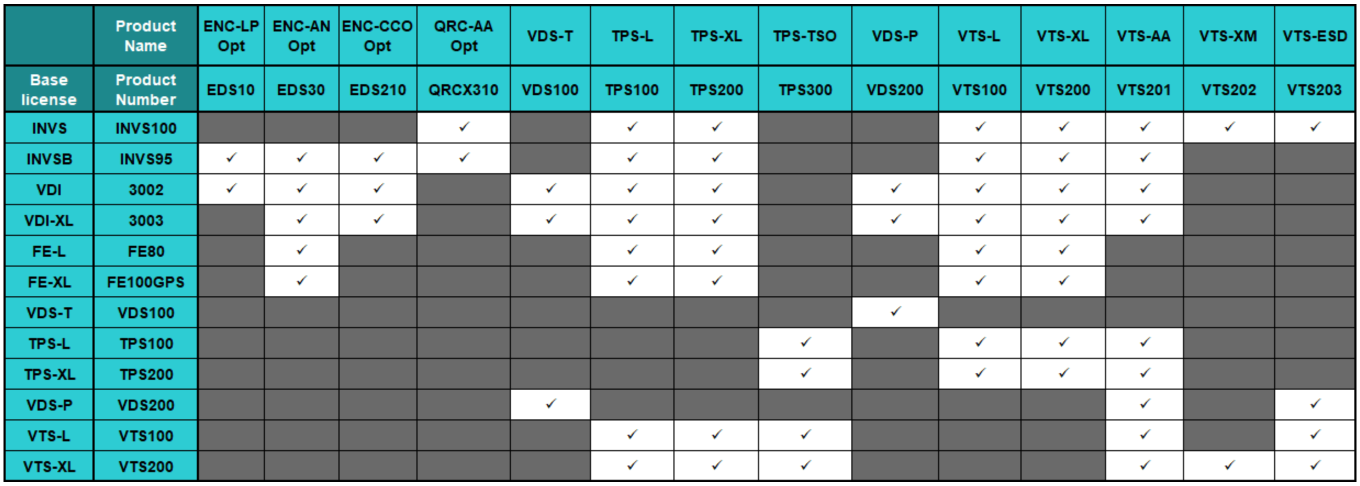
Note: The table has been split into two parts to improve readability.

Dynamic Licensing Matrix - Part 1



	Product Name	INVS HIER Opt	INVS 20nm Opt	INVS 10nm Opt	INVS 7nm Opt	INVS 5nm Opt	INVS 3nm Opt	INVS GigaPlace GXL Opt	INVS DFM Opt	INVS PI Opt	INVS ML Opt	INVS HFR Opt	INVS MS Opt	INVS 3D-IC Opt	VDI-XL Capacity Opt
Base license	Product Number	INVS40	INVS20	INVS10	INVS07	INVS05	INVS03	INVS48	INVS50	INVS55	INVS65	INVS35	INVS30	INVS60	3004
INVS	INVS100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	▨
INVSB	INVS95	✓	✓	▨	▨	▨	▨	✓	✓	✓	✓	✓	✓	✓	▨
VDI	3002	▨	✓	✓	✓	✓	✓	✓	▨	▨	▨	✓	✓	✓	▨
VDI-XL	3003	▨	✓	✓	✓	✓	✓	✓	✓	▨	▨	✓	✓	✓	✓
FE-L	FE80	✓	✓	✓	✓	✓	✓	✓	▨	▨	▨	▨	✓	▨	▨
FE-XL	FE100GPS	✓	✓	✓	✓	✓	✓	✓	▨	▨	▨	▨	✓	✓	▨
VDS-T	VDS100	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	✓	✓	▨
TPS-L	TPS100	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	✓	✓	▨
TPS-XL	TPS200	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	✓	✓	▨
VDS-P	VDS200	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	✓	✓	▨
VTS-L	VTS100	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	✓	✓	▨
VTS-XL	VTS200	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	✓	✓	▨

Dynamic Licensing Matrix - Part 2



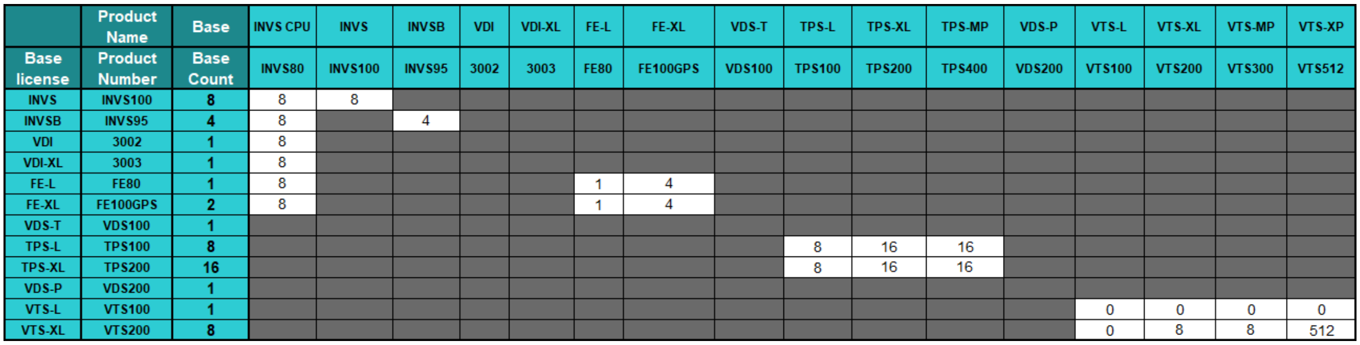
	Product Name	ENC-LP Opt	ENC-AN Opt	ENC-CCO Opt	QRC-AA Opt	VDS-T	TPS-L	TPS-XL	TPS-TSO	VDS-P	VTS-L	VTS-XL	VTS-AA	VTS-XM	VTS-ESD
Base license	Product Number	EDS10	EDS30	EDS210	QRCX310	VDS100	TPS100	TPS200	TPS300	VDS200	VTS100	VTS200	VTS201	VTS202	VTS203
INVS	INVS100				✓		✓	✓			✓	✓	✓	✓	✓
INVSB	INVS95	✓	✓	✓	✓		✓	✓			✓	✓	✓		
VDI	3002	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		
VDI-XL	3003		✓	✓		✓	✓	✓		✓	✓	✓	✓		
FE-L	FE80		✓				✓	✓			✓	✓			
FE-XL	FE100GPS		✓				✓	✓			✓	✓			
VDS-T	VDS100									✓					
TPS-L	TPS100								✓		✓	✓	✓		
TPS-XL	TPS200								✓		✓	✓	✓		
VDS-P	VDS200					✓							✓		✓
VTS-L	VTS100						✓	✓	✓				✓		✓
VTS-XL	VTS200						✓	✓	✓				✓	✓	✓

Multi-CPU Matrix

The Multi-CPU matrix shows the number of CPUs that are enabled by each base/additional license.

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

Multi-CPU Acceleration Matrix



	Product Name	Base	INVS CPU	INVS	INVSB	VDI	VDI-XL	FE-L	FE-XL	VDS-T	TPS-L	TPS-XL	TPS-MP	VDS-P	VTS-L	VTS-XL	VTS-MP	VTS-XP
Base license	Product Number	Base Count	INVS80	INVS100	INVS95	3002	3003	FE80	FE100GPS	VDS100	TPS100	TPS200	TPS400	VDS200	VTS100	VTS200	VTS300	VTS512
INVS	INVS100	8	8	8														
INVSB	INVS95	4	8		4													
VDI	3002	1	8															
VDI-XL	3003	1	8															
FE-L	FE80	1	8					1	4									
FE-XL	FE100GPS	2	8					1	4									
VDS-T	VDS100	1																
TPS-L	TPS100	8									8	16	16					
TPS-XL	TPS200	16									8	16	16					
VDS-P	VDS200	1																
VTS-L	VTS100	1														0	0	0	0
VTS-XL	VTS200	8														0	8	8	512

Optional License Requirement for 3/5/7/10/20/32nm Nodes

The following Innovus product options provide licenses for advanced nodes:

- **INVS03**
  - Needed if the minimum width specified on any routing layer is <= 12nm
  - Is a superset of 5, 7,10, 20, and 32nm node features
- **INVS05**
  - Needed if the minimum width specified on any routing layer is <= 15nm
  - Is a superset of 7,10, 20, and 32nm node features
- **INVS07**
  - Needed if the minimum width specified on any routing layer is <= 20nm
  - Is a superset of 10, 20, and 32nm node features
- **INVS10**
  - Needed if the minimum width specified on any routing layer is <= 26nm
  - Is a superset of 20 and 32nm node features
- **INVS20**
  - Needed if the minimum width specified on any routing layer is <= 40nm
  - Is a superset of 32nm node features
- **EDS30**
  - Needed for the VDI products if the minimum width specified is <= 50nm
  - This is not needed for Innovus products which have the capability built-in

For more information on these products and options, see Innovus Packaging and Licensing.

Getting Started

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 Innovus™ Implementation System licenses, see "About Innovus Licenses" in the Product and Licensing Information chapter.

Setting Up the Run-Time Environment

If *install_dir* is the location of your Innovus installation, you should set up your run-time environment like this:

- Add the *install_dir*/bin directory to your path. The bin directory has links to all the public executables in the install hierarchy.
- If you want the legacy Innovus man pages to be available from the Unix man command, you can add *install_dir*/share/innovus/man to your MANPATH envar.
- If you want the Tcl man pages to be available from the Unix man command, you can add *install_dir*/share/tcltools/man to your MANPATH envar.

For example, you might add this to your startup shell script:

```
set install_dir = /tools/innovus17.1/lnx86

set path = ($install_dir/bin $path)

setenv MANPATH $install_dir/share/innovus/man:$install_dir/share/tcltools/man:$MANPATH
```

Note: When Innovus launches, it automatically adds the legacy man pages and the Tcl man pages to the beginning of the current MANPATH inside Innovus. Therefore, from within Innovus, the man command will see both sets of man pages before any other man pages.

Supported and Compatible Platforms

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

64-Bit Version of Innovus Applications

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

Temporary File Locations

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

By default, tmp_dir is created in ./. If you do not have write permission in ./, the tool will use /tmp.

The name of the tmp_dir directory will look like:

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

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

```
innovus_temp_10233_farm254_bob_nfp9ez
```

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

OpenAccess

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

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

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

Launching the Console

The window (shell tool, xterm, and so on) where you start the Innovus session is called the Innovus console. You enter all Innovus text commands in the console window, and the software displays messages there. You start legacy Innovus from Unix like this:

```
>innovus
```

When a session is active, Innovus shows the Tcl interpreter prompt like this:

```
innovus 1>
```

Innovus currently uses Tcl version 8.6. The current version of the Tcl interpreter is in the $tcl_version variable.

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

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

For a detailed description of the `innovus` command-line options and the initialization files loaded at startup, see `innovus` in the *Text Command Reference*. The initialization files can be used to configure the GUI, load utility Tcl files, or configure Innovus settings.

Alternatively, at the Unix prompt ,you can type:

```
>innovus -help
```

for a summary of the options or

```
>man innovus
```

for the full man page (available if `MANPATH` includes `<`*`install_dir`*`>/share/innovus/man`).

If you type the `innovus` command without parameters, the Innovus software starts in the 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.

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

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

- First Encounter® XL

For an overview of the products and product licensing, see Product and Licensing Information.

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

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

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

```
innovus 3> run_<Tab>
```

```
run_abstract run_decap_eco run_replay run_vsr
```

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

```
innovus 3> run_abstract
```

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

Tab Completing Parameter Names

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

For example,

```
innovus 3> report_timing -check<Tab>
-check_clocks  -check_type
```

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

```
innovus 3> report_timing -check_type
```

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

```
innovus 3> report_timing -<Tab>
```

```
-begin_end_pair    -check_clocks          -check_type
-clock_from        -clock_to              -collection
-debug             -delay_limit           -derate_summary
-early             -edge_from             -edge_to
-fall              -format                -from
-from_fall         -from_rise             -hpin
```

...

Tab Completing Global Variable Names

The `Tab` completion capability is also available for Tcl variable names. The `set_db` command understands them for its first argument.

For example, if you type `set_db lefDefOut<Tab>` the software completes the name:

```
innovus 5> set_db lefDefOutVersion
```

Similarly, the `man` command also takes Tcl variable names as its first argument and allows `Tab` completion.

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

```
innovus 6> puts $delaycal_default<Tab>
```

```
delaycal_default_net_delay delaycal_default_net_load
delaycal_default_net_load_ignore_for_ilm
```

Tab Completing Enum Type Values for Parameters

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

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

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

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

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

Tab Completing Unix file and directory names

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

For example,

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

Command-Line Editing

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

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

> ⓘ Copying and pasting pointers in an active session may cause the tool to crash. To prevent this situation, do not:
>
> - Copy pointers from one command's result and paste to another command as input.
> - Copy a pointer from one session to another.

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

Notes

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

Independent keys	Result

`Home`	Goes to the start of the current line. If already there, goes to the start of the previous line.
`Down`	In a multiline command, moves to the next line.
`End`	Goes to the end of the current line. If already there, goes to the end of the next line.
`Tab`	Completes the command.
`Up`	In a multiline command, moves to the previous line.
Control characters	**Result**
`Ctrl+a`	Goes to the beginning of the line.
`Ctrl+b`	Moves the cursor left by one character.
`Ctrl+c`	Exits from editing mode, returning the console to normal Innovus mode.
`Ctrl+d`	Deletes the next character if the cursor is in the middle of a line. Lists the files in the current directory beginning with the word just before the cursor, if the cursor is at the end of a line.
`Ctrl+e`	Goes to the end of the current line.
`Ctrl+f`	Move the cursor one character to the right.
`Ctrl+h`	Deletes one character before the cursor.
`Ctrl+i`	Completes filename or displays all possible options in the given context.
`Ctrl+j`	Submits the line; Same as `Enter`.
`Ctrl+k`	Deletes characters from the cursor to the end of the line.
`Ctrl+l`	Clears the screen and redisplays the last line.
`Ctrl+m`	Same as `Ctrl+j`.
`Ctrl+n`	Goes to the next line in history; Same as `Ctrl+Down`.
`Ctrl+o`	Accepts the line, moves the history pointer to the next position.

`Ctrl+p`	Goes to the previous line in history; Same as `Ctrl+Up`.
`Ctrl+r`	Searches backword through history for text; must start line if text begins with an Up arrow.
`Ctrl+t`	Transposes characters, that is exchanges the character before the cursor with the character at the cursor, and then moves the cursor one character right.
`Ctrl+u`	Deletes the line.
`Ctrl+w`	Deletes the characters between the cursor and the marked position set by `Esc+space`.
`Ctrl+x`	Moves the cursor to the marked position set by `Esc+space`.
`Ctrl+y`	Pastes yanked string before the cursor.
`Ctrl+z`	Suspends the tool (System hotkey)
`Ctrl+]`	Moves to the next character; Equals to the next input character.
`Ctrl+Down`	Goes to the next line in history; Same as `Ctrl+n`.
`Ctrl+Up`	Goes to the previous line in history; Same as `Ctrl+p`.
`Ctrl+?`	Deletes the character before cursor.
Escape sequences	**Result**
`Esc+Ctrl+h`	Deletes the previous word.
`Esc+Delete`	Deletes the previous word.
`Esc+space`	Marks a position.
`Esc+.`	Inserts the last argument of the last command before the cursor.
`Esc+<`	Displays the first command in history.
`Esc+>`	Displays the last command in history.
`Esc+?`	Displays all possible file names.

`Esc+b`	Moves the cursor to the beginning of the word to the left.
`Esc+d`	Deletes the word to the right of the cursor.
`Esc+f`	Moves cursor to the beginning of the next word.
`Esc+l`	Changes the characters from the cursor to the end of the word to lowercase.
`Esc+u`	Change the characters from the cursor to the end of the word to uppercase.
`Esc+y`	Pastes the yanked string before the cursor.
`Esc+w`	Saves the strings between the marked position (set by `Esc+space`) and the cursor position into the yank buffer.
`Esc+p`	Starts a backward search in history.
`Esc+Up`	Moves the cursor up to the previous line.
`Esc+Down`	Moves the cursor down to the next line.
`Esc+Left`	Same as `Esc+b`.
`Esc+Right`	Same as `Esc+f`.

Notes

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

Setting Preferences

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

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

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

For information on setting design preferences, see "Set Preference" in the View Menu chapter of the *Menu Reference*.

Interrupting the Software

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

Interrupt Behavior When Tool Is Idle

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

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

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

Interrupt Behavior in Interactive Mode

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

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

2) Quit.

3) Finish current command but interrupt Tcl script.

4) Interrupt current command and Tcl script and return to prompt.

5) Suspend current command and return to prompt. Use command 'resume' to continue.

Type a number 1-5 and press ENTER:
```

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

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

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

- `routeDesign`
- `globalDetailRoute`
- `optDesign`
- `verifyConnectivity`
- `verifyPowerVia`
- `verifyMetalDensity`
- `verifyProcessAntenna`
- `verifyACLimit`

Interrupting the Execution of Batch Files

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

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

- To confirm that you want to interrupt script, type `Y`.

  In this case, you can save the design and proceed with the flow.
- To continue running the script, type `N`.

Suspending the Execution of a Script

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

Stopping the Software

Use one of the following methods to stop the software:

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

  ```
  exit
  ```

The Log Files and Controls

Command logging plays a vital role in the debug process. By default, the tool creates a `.log`, `.logv`, and `.cmd` file at startup for the following purposes:

- `.log` - Captures all the output to the xterm.
- `.logv` - Captures all the output to the xterm with detailed information that is not needed normally but can be useful while debugging problems. Every output line has a time-stamp, and complex commands, such as `optDesign`, will write out much more detailed internal information to help debug issues.
- `.cmd` - Captures just the Tcl commands that are executed, without any extra formatting, so that they can be cut-and-pasted for reuse.

The names of the files default to the program name, extension, and an extra number if there is a file name collision. For example, `innovus.log2`, `innovus.logv2`, and `innovus.cmd2`. These names can be overridden, or the files suppressed with the Linux `innovus` command-line options `-log`, `-no_cmd`, and `-no_logv` . Type `man innovus` at the Linux prompt for details.

You can access the log file through the integrated log file viewer. Use one of the following methods to access 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 *Menu Reference*.
- On the text command line, type the following command in the console window where the software is running:

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

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

Accessing Documentation and Help

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

- Launching Cadence Help From the Command Prompt
- Accessing Documentation and Help from the GUI
- Using the man and help Commands on the Command Line

Launching Cadence Help From the Command Prompt

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

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

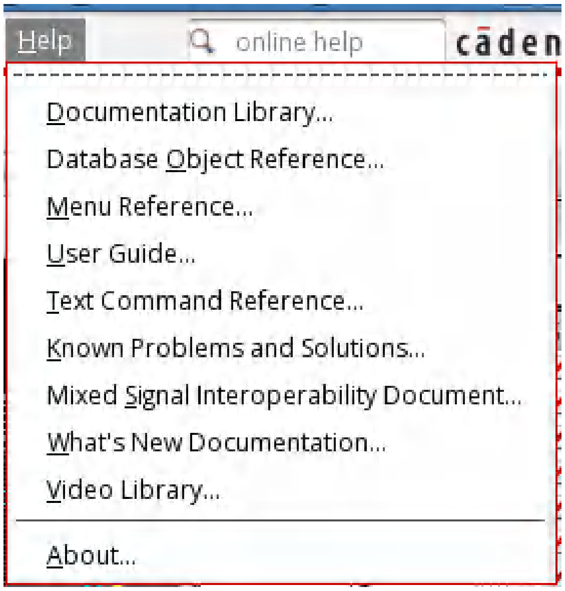
Accessing Documentation and Help from the GUI

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

- Select Help from the Main Menu
- Select Help Button on a Form

Select Help from the Main Menu

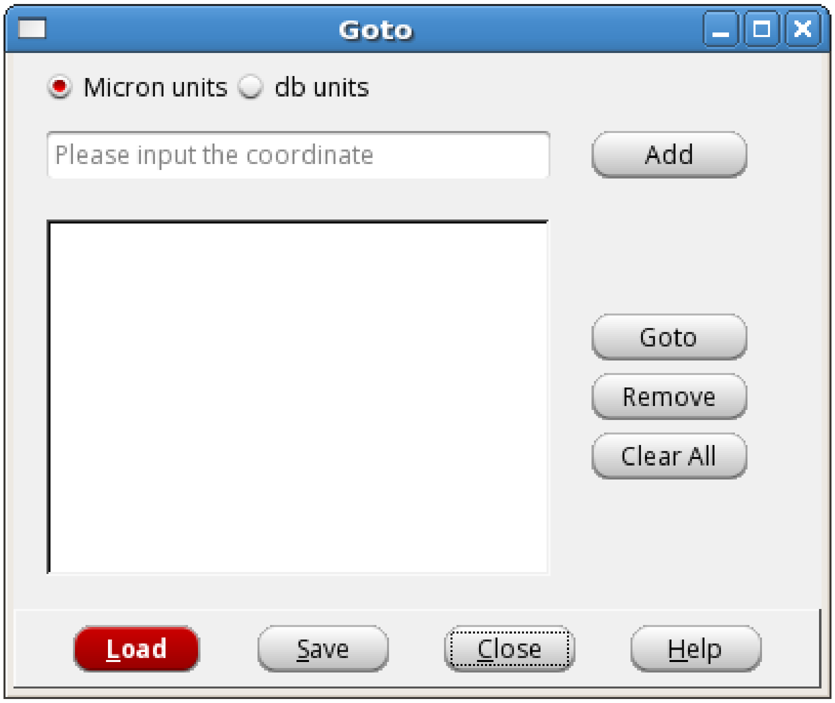
Click *Help* on the main menu and then select *Documentation Library* to open the Cadence Help window. The Cadence Help window provides access to all the documentation shipped with the release.


Help
online help
cāden
Documentation Library...
Database Object Reference...
Menu Reference...
User Guide...
Text Command Reference...
Known Problems and Solutions...
Mixed Signal Interoperability Document...
What's New Documentation...
Video Library...
About...

Alternatively, you can select any of the options in the *Help* menu to open that document directly. For example, select *Text Command Reference* to open the Table of Contents page of the text command reference.

Select Help Button on a Form

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



Clicking the *Help* button on a form opens the *Menu Reference* entry for that form in the Cadence Help window.

Using the man and help Commands on the Command Line

Using the help Command to View the Command Syntax

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

  `help` *`command_name`*

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

  ```
  help getAllLayers
  ```

  The software displays the following text:

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

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

  ```
  help
  ```

Using the man Command to View the Command Description

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

  ```
  man command_name
  ```

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

  ```
  man getAllLayers
  ```

The software displays the following text:

```
Name

       getAllLayers  - Returns a complete list of all layers and floorplan object
       settings

Syntax

       getAllLayers [-help] [type]

Description

       Returns a complete list of all layers and floorplan  object  settings.  If
       you  specify  type,  the  software returns all the layers of the specified
       type. This command can be used at any stage in the design flow.

Parameters

       -help  Prints a brief description that includes type and default  informa-
              tion for each getAllLayers parameter.

              For  a  detailed  description of the command and all of its parame-
              ters, use the man command:

              man getAllLayers

       type   Specifies the type of the layer. Innovus supports six types of lay-
              ers, which can be specified as follows:

              * object:  If  you specify type as object, the software returns all
                object layers, which represent db  objects,  such  as  instances,
                modules, pins, and so on.

              * display:  If  you  specify  type as display, the software returns
                display-only or view-only layers, including flightlines,  rulers,
                and congestion.

              * multi:  If you specify type as multi, the software returns multi-
                ple color layers, such as congestion, maps, and yield map.

              * metal: If  you  specify  type  as  metal,  the  software  returns
                wire/via layers, including metal/via, pin, and blockage.
              * custom:  If you specify type as custom, the software returns cus-
                tom layers, which are used to represent custom objects and  GDSII
                data.
              * internal:  If  you specify type as internal, the software returns
                all internal layers.
Example
       Returns all metal layer names:
       getAllLayers metal

(END)
```

Using the help Command to View Message Summary

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

  ```
  help msg_id
  ```

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

  ```
  help TAMODEL-302
  ```

  The software displays the following text:

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

Using the man Command to View Message Detail

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

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

  ```
  man msg_id
  ```

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

  ```
  man TAMODEL-302
  ```

  The software displays the following text:

  ```
  NAME

         TAMODEL-302 (warning)

  SUMMARY

         Data signal arrives at clock pin '%s'.  This data/clock conflict may be

         due to missing or incomplete clock definitions.  Trigger arcs and check

         arcs associated with '%s' are being removed to prevent data signal from

         propagating to clock paths.

  DESCRIPTION

         Usually data signals  arrive  at  clock  pins  of  sequential  elements

         because  clock  source  is  not  defined  properly.  Please trace clock

         sources backward from the clock pins of  sequential  elements  to  make

         sure  that clock waveforms are associated with clock sources.  This can

         be done by using create_clock or create_generated_clock command.
  ```

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

Customizing the User Interface

Overview

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

- `uiAdd`
- `uiDelete`
- `uiSet`
- `uiGet`
- `uiFind`

For more information on these commands, see the "GUI Commands" chapter of the *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

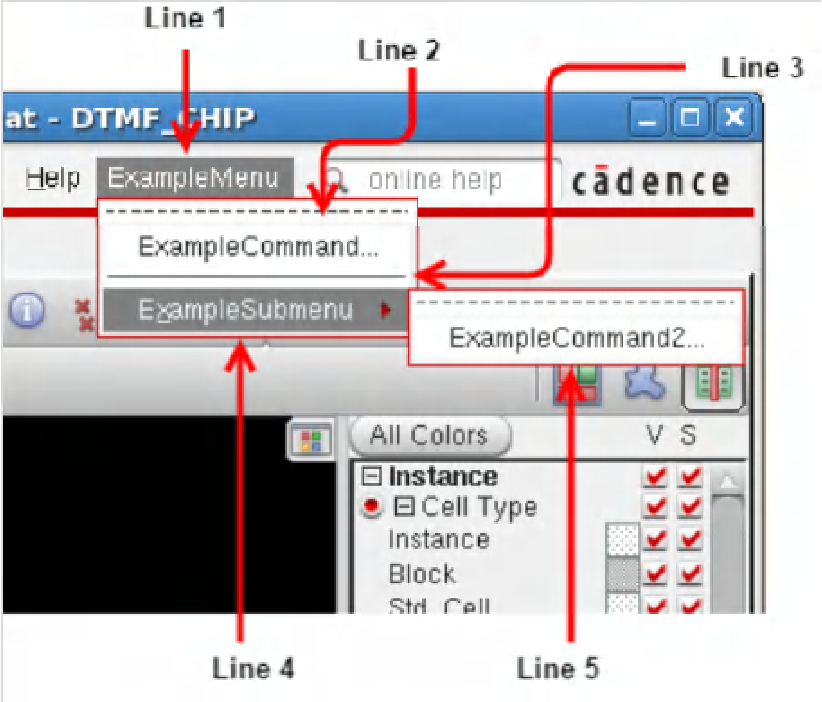
Using the `uiAdd` command, you can create a new menu and add it to the main window menu bar. You can then add menu elements, such as command, submenu, separator, radio button and check box, to the new menu using the same `uiAdd` command.

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 expSubmenu -type submenu -label "ExampleSubmenu" -underline 1 -in expMenu

uiAdd expCommand2 -type command -label "ExampleCommand2..." -command [list
puts "Example Command"] -in expSubmenu
```

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 *Text Command Reference*.

Modifying an Existing Menu

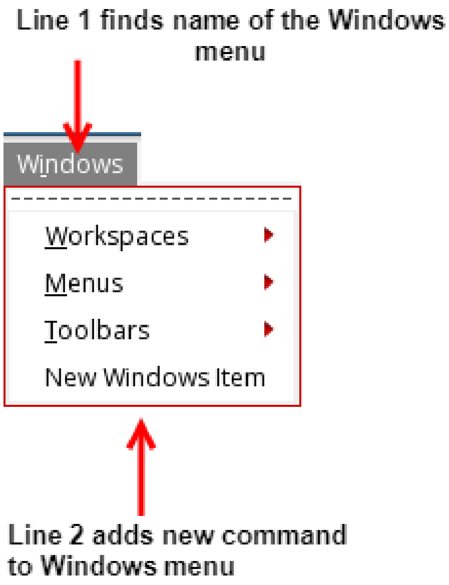
You can also use the `uiAdd` command to add or replace menu elements in an existing menu.

Adding a Menu Element to an Existing Menu

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

```
set wMenu [uiFind main -type menu -label "Windows"]

uiAdd newWindows -type command -label "New Windows Item" -command [list puts
"New Windows Item"] -in $wMenu
```

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


Line 1 finds name of the Windows menu
Windows
Workspaces
Menus
Toolbars
New Windows Item
Line 2 adds new command to Windows menu

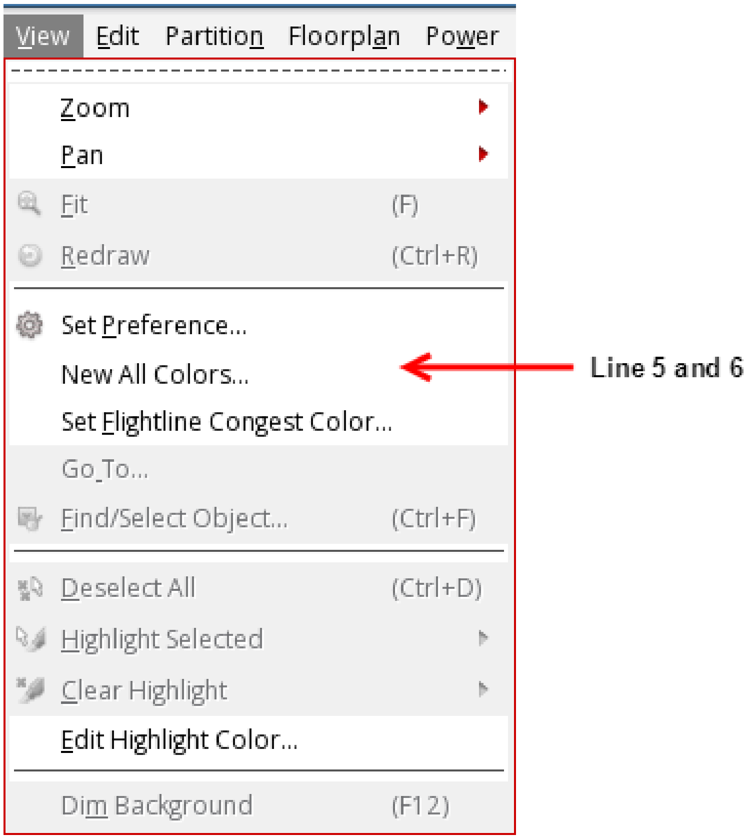
Replacing an Existing Menu Element

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

```
set viewMenu [uiFind -type menu -label "View"]

set oldMenu [uiFind $viewMenu -type command -label "All Colors..."]

set before [uiGet $oldMenu -before]

uiDelete $oldMenu

set newMenu ${oldMenu}_new

uiAdd $newMenu -type command -label "New All Colors..." -before $before -command "puts
{New All Colors}" -in $viewMenu
```

In this script:

- Line 1 finds the name of the *View* menu.
- Line 2 finds the name of *All Colors* menu element in the *View* menu by its label.
- Line 3 finds its neighbor using the `uiGet` command.
- Line 4 deletes the *All Colors* menu element by using the `uiDelete` command.
- Line 5 and 6 create a new menu element labeled *New All Colors* in the same location.



Adding a New Toolbar and Toolbutton

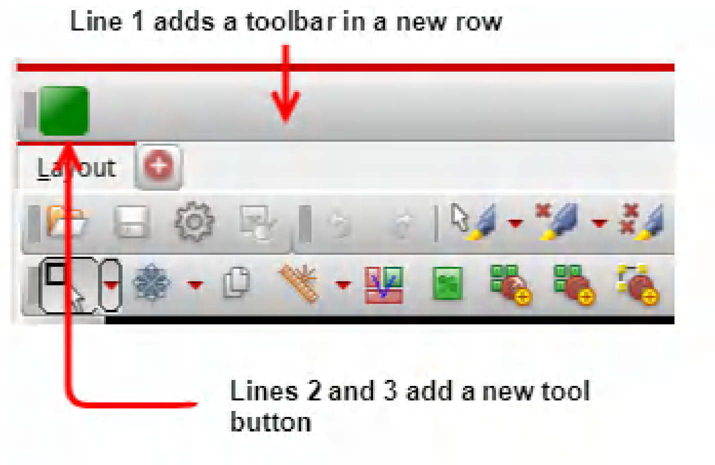
Using the `uiAdd` command, you can add a new toolbar and toolbuttons as shown in the following script:

```
uiAdd expToolbar -type toolbar -in main -label "Example Toolbar" -newline true
```

```
set ICON_DIR "./"

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

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

Note: You must have the `example.png` file in the specified directory for the above code to work correctly.


Line 1 adds a toolbar in a new row
Layout
Lines 2 and 3 add a new tool button

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

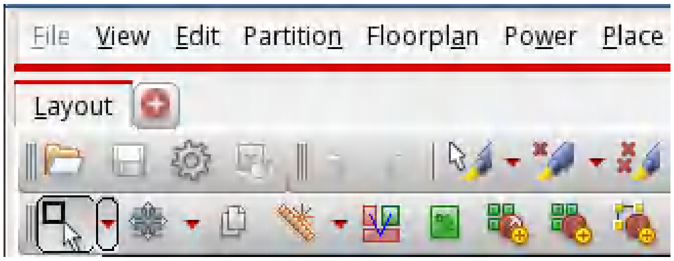
Using the `uiGet`, `uiFind`, and `uiSet` commands in the GUI development kit, you can query and configure various interface elements, including menus, status bar, and the main window.

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
```

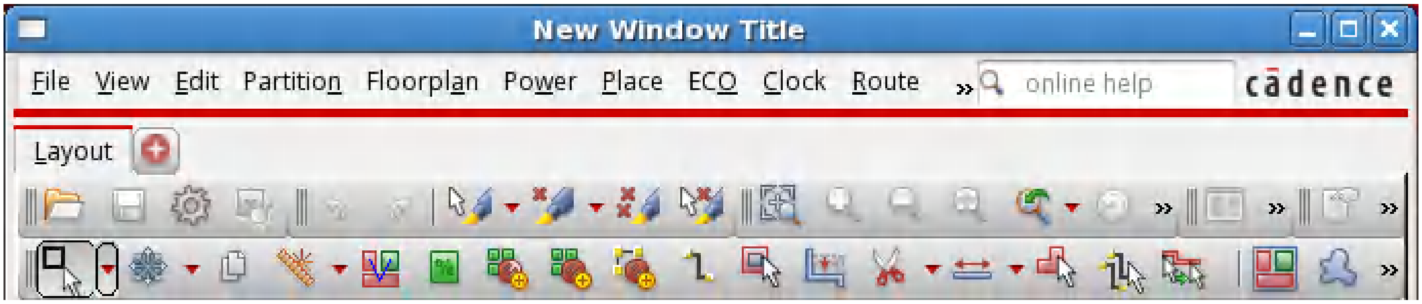
Setting the Main Window's Size and Title

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

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

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

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

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



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

Accelerating the Design Process By Using Multiple-CPU Processing

Overview

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

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

You configure multiple-CPU processing by using the commands described in the Multiple-CPU Processing Commands chapter of the Innovus System Text Command Reference or the "Multiple CPU Processing" form in the *Tools Menu*.

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

Table: Innovus System features that support multiple-CPU processing

Feature	Commands	Details
Capacitance table generation	`generateCapTbl`	See Generating a Capacitance Table.
Global placement	`place_design`	See Running Placement in Multi-CPU Mode in the "Placing the Design" chapter.
Optimization	`optDesign {-preCTS	-postCTS	-postRoute}`	See Distributed Timing Analysis for Hold Fixing in the "Optimizing Timing" chapter.
Clock concurrent Optimization (CCOPT)	`ccopt_design`	See section Clock Tree Synthesis.

Metal fill	`addMetalFill`	See Adding Metal Fill in the Multiple-CPU Processing Mode in the "Optimizing Metal Density" chapter.

NanoRoute router	`globalRoute`
	`detailRoute`
	`routeDesign`
	`ecoRoute`
		• Superthreading is supported for detailed routing only.
		• Superthreading options take precedence over multi-threading options.
		See Accelerating Routing with Multi-Threading and Superthreadingin the "Using the NanoRoute Router" chapter.

TQuantus, IQuantus, and Standalone extraction	`setExtractRCMode` `extractRC`	See Distributed Processing in Extraction.
Signal integrity analysis	`optDesign` `timeDesign`	See Multi-CPU Processing Settings in the "Analyzing and Repairing Crosstalk" chapter. • For backward compatibility, distributed processing options take precedence. • Superthreading options take precedence over multi-threading options.
Verify connectivity	`verifyConnectivity`	See "Verifying Connectivity" in the "Identifying and Viewing Violations" chapter.
Verify DRC	`verify_drc`	See Verifying DRC in the "Identifying and Viewing Violations" chapter.
Verify metal density	`verifyMetalDensity`	See Verifying Metal Density in Multi-Thread Mode in the "Identifying and Viewing Violations" chapter.
Delay calculation	All commands that require timing data and invoke a full delay calculation.	See "Base Delay Analysis" chapter.

Timing Budgeting	`deriveTimingBudget` `saveTimingBudget`	See the 'Support for Distributed Processing in Budgeting' section in the Timing Budgeting chapter.
Power Planning	`addStripe`	See the 'Planning Power' section in the Low Power Design chapter.

Running Distributed Processing

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

- `setDistributeHost`
  Use this command to specify a configuration file for distributed processing or create the configuration for the remote shell, secure shell,RTDA, or load-sharing facility queue to use for distributed processing. If you request more machines than are available, most applications wait until all requested machines are available.
  To display the current setting for `setDistributeHost`, use the `getDistributeHost` command.
- `setMultiCpuUsage`
  Use this command to specify the maximum number of computers to use for processing.
  To display the current setting for `setMultiCpuUsage`, use the `getMultiCpuUsage` command.

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
place_design
```

Running Superthreading

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

- setDistributeHost
- setMultiCpuUsage

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

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

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

Memory and Run Time Control

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

When you run `report_resource -verbose`, the following detailed memory information is displayed:

Current (total cpu=0:00:12.9, real=0:05:48, peak res=275.8M, current mem=383.9M)				
Cpu(s) 2, load average: 4.63				
Mem: 16443800k total, 16378412k used, 65388k free, 105704k buffers				
Swap: 16777208k total, 17460k used, 16759748k free, 12528212k cached				
Memory Detailed Usage:				
	Data Resident Set(DRS)	Private Dirty(DRT)	Virtual Size(VIRT)	Resident Size(RES)
Total current:	383.9M	275.8M	854.1M	358.9M
peak:	383.9M	275.8M	854.1M	358.9M

- `Cpu(s)` is the number of available processors in the machine.
- `Load` average is the system load averages for the past 1 minute.
- `Mem` and `Swap` are the current memory information of the machine.
  The value of `MEM` in the LSF report corresponds to the value of `RES` in the `report_resource` report, and the value of `SWAP` in the LSF report corresponds to the value of `VIRT` in the `report_resource` report.
- `Data Resident Set (DRS)`is the amount of physical memory devoted to other than executable code. "`current mem`" shows this value (Total current DRS) .
- `Private Dirty (DRT)` is the memory which must be written to disk before the corresponding physical memory location can be used for some other virtual page. "`peak res`" shows this value (Total peak DRT). This is the minimum number that you must reserve to run the program.

- `Virtual Size (VIRT)` is the total amount of virtual memory used by the task. It includes the swapped and non-swapped memory.
- `Resident Size (RES)` is the non-swapped physical memory a task has used. The number of "Total Peak RES" is the recommended physical memory to reserve.

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

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

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

The following message is displayed:

<table>
<tr><td colspan="5">Ending "opt_postroute" (total cpu=0:57:18, real=0:33:24, peak res=6493.1M, current mem=5305.0M)</td></tr>
<tr><td colspan="5">Memory Detailed Usage:</td></tr>
<tr><td></td><td>Data Resident Set(DRS)</td><td>Private Dirty(DRT)</td><td>Virtual Size(VIRT)</td><td>Resident Size(RES)</td></tr>
<tr><td>Total current:</td><td>5305.0M</td><td>4012.1M</td><td>5919.2M</td><td>4255.4M</td></tr>
<tr><td>peak:</td><td>10712.8M</td><td>6493.1M</td><td>15090.3M</td><td>7312.5M</td></tr>
<tr><td>Master current:</td><td>5305.0M</td><td>4012.1M</td><td>5919.2M</td><td>4255.4M</td></tr>
<tr><td>peak:</td><td>5565.5M</td><td>4055.1M</td><td>6064.5M</td><td>4298.4M</td></tr>
<tr><td>Task peak:</td><td>748.8M</td><td>368.7M</td><td>1219.4M</td><td>456.4M</td></tr>
</table>

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

Checking the Distributed Computing Environment

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

Setting and Changing the License Check-Out Order

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

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

Limiting the Multi-CPU License Search to Specific Products

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

```
[DEV]innovus 1> getMultiCpuUsage

Total CPU(s) Enabled: 2
Current License(s): 1 Encounter_Digital_Impl_Sys_XL
keepLicense: true
licenseList: enccpu edsl edsxl
```

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

Releasing Licenses Before the Session Ends

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

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

  ```
  setMultiCpuUsage -keepLicense false
  ```

  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.

2

Flows

- Design Implementation Flow
- Foundation Flow
- Hierarchical and Prototyping Flow
- Machine Learning Flow

Design Implementation Flow

Introduction

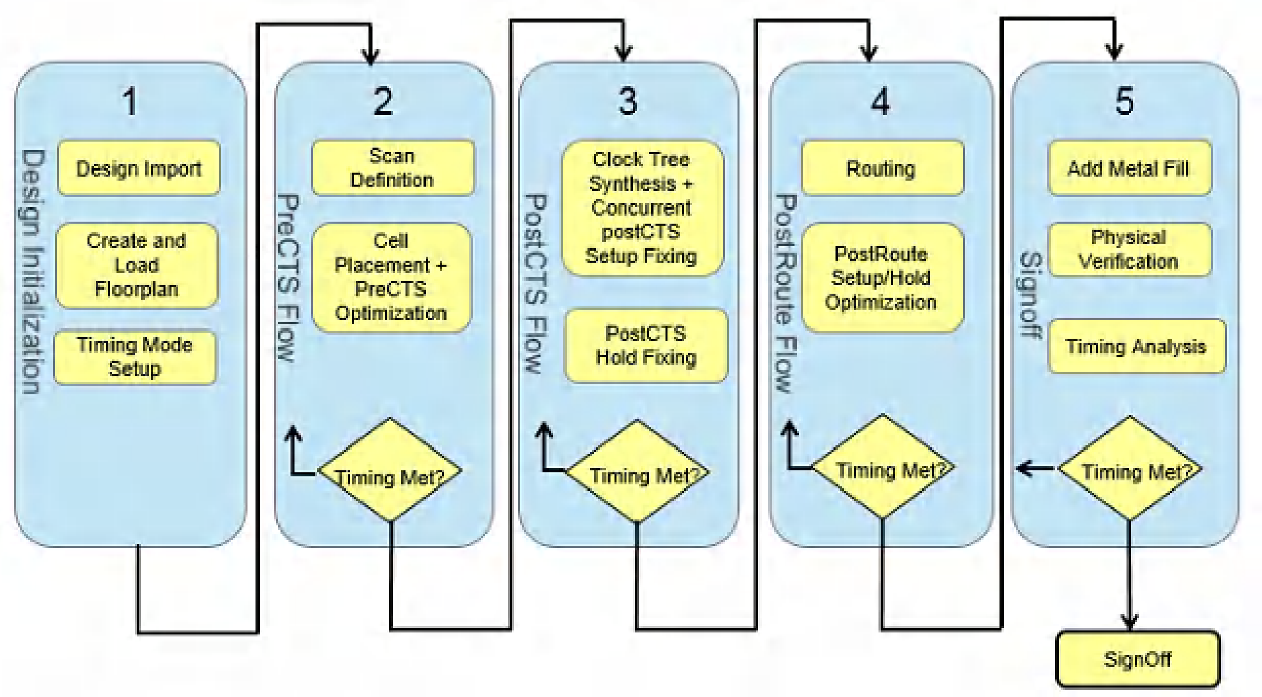
Achieving timing closure on a design is the process of creating a design implementation that is free from logical, physical, and design rule violations and meets or exceeds the timing specifications for the design. For a production chip, all physical effects, such as metal fill and coupling, must be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

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

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

Recommended Timing Closure Flow

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



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

Software

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

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

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

Data Preparation and Validation

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

The goals of data preparation and validation include:

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

Data Preparation

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

Timing Libraries

- Every cell used in the design should be defined in the timing library.
    - If multiple delay corners are being analyzed, then each cell needs to be characterized for each corner.
    - Innovus supports Non-linear Delay Models (NLDM), ECSM, and CCS. ECSM libraries are recommended.
        - CCS/ECSM are less pessimistic than NLDM and, therefore, you can gain about 5% to 10% of the clock period on the slack by using these libraries.

Physical Libraries

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

Verilog Netlist

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

Timing Constraints

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

Setting Preservation Constraints for Design Objects

As part of data preparation, preservation constraints (such as the `dont_touch*` and `dont_use*` attributes) can be set on various `netlist` objects, including `topCell`, `hInst`, `net`, `instTerm`, `libCell`, `vCell`, `hNet`, and `hPin`. Preservation constraints are optimization constraints that are independent of the timing constraints. So, the software does not depend on the SDCs to save, restore, or report these optimization constraints. These constraints are kept as database attributes and can be set and queried, as required.

The user and application-created constraints are stored with different attributes. So, the user constraints are never overwritten. Hence, the original user intent can always be queried. For example, `ccopt_design` will not change an inst `.pStatus`, but instead, it will set `.pStatusCTS`. When there are multiple constraints for a specific attribute, there is an overall effective attribute to get the net result (for example, an inst has `pStatusEffective` and `dontTouchEffective` attributes).

Constraining Design Objects

Like any other attribute, all the preservation attributes can be queried and set through the `dbSet` command, as shown below:

```
dbSet [dbGet top.insts.name i1/i2 -p1].dontTouch true
```

The existing SDC commands can also be used as shown below:

```
set_dont_touch i1/i2 true
```

These commands do not require interactive constraint modes, because they are not associated with constraint modes and are not stored as part of SDCs.

Use one of the following commands to see all the preservation attributes on the various objects:

```
dbSchema * dont*
```

```
dbSchema inst place_status*
```

See the Innovus Database Object Information document for information on Innovus objects and attributes.

Dealing with SDC and Library Constraints

Some users have initial preservation constraints in their SDC and `.lib` files (such as `set_dont_touch` and `set_dont_use`). So, when an SDC or `.lib` file is read for the first time during design initialization using the `read_mmmc` and `init_design` commands, these commands set the corresponding database preservation attributes.

However, later in the flow, these commands are ignored inside the SDC or `.lib` files (like during `restoreDesign`, or if `set_analysis_view` causes new SDC or `.lib` files to be read). This prevents the database attributes from being inadvertently reset back to their initial values after the user changes them during the flow. To change the values, direct commands must be used outside the SDC or `.lib` files (like by directly sourcing a file, or by typing them in).

Extraction

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

Signal Integrity (SI) Libraries

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

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

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

See Configuring the Setup for Multi-Mode Multi-Corner Analysis in the *Innovus User Guide* for information on defining the MMMC environment.

Data Validation

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

Loading the Design

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

```
source design.globals
init_design
```

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

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

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

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

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

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

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

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

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

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

```
Total number of usable buffers: 18
List of unusable buffers:
Total number of unusable buffers: 0
List of usable inverters: CLKINVX2 CLKINVX1 CLKINVX12 CLKINVX16 CLKINVX20
CLKINVX3 CLKINVX4 CLKINVX8 CLKINVXL INVX1 INVX2 INVX12 INVX16 INVX20 INVX3 INVXL
INVX4 INVX8
Total number of usable inverters: 18
List of unusable inverters:
Total number of unusable inverters: 0
List of identified usable delay cells: DLY2X1 DLY1X1 DLY4X1 DLY3X1
Total number of identified usable delay cells: 4
List of identified unusable delay cells:
Total number of identified unusable delay cells: 0 Also, look for cells that do
not have a function defined for them: No function defined for cell 'HOLDX1'. The
cell will only be used for analysis.
```

Checking Timing Constraint Syntax

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

- **Unsupported constraints**
    - The Innovus software may not support the SDC constraints being used with the design, or the constraints may not match the netlist. If the constraints are not supported, they may need to be re-expressed (if possible) in constraints that the Innovus software does support.
- **Ignored timing constraints**
    - Syntax errors can cause the tools to ignore certain constraints resulting in the misinterpretation of important timing considerations. Check for warnings or errors about unaccepted SDC constraints. The following are possible causes for ignored constraints.
        - A design object is not found. If the constraints refer to pins, cells, or nets that are not found in the netlist, then consider the following possible causes:
            - There could be a naming convention problem in the constraint file.
            - The netlist and constraints are out of sync, and a new set of constraints and/or a new netlist needs to be obtained.

    - An incorrect type of object is being passed to a constraint.
    - An option is being used incorrectly or an unknown option is used.
    - Illegal endpoints are used in assertions. Use the primary IOs (top-level ports, CK or D register pin) to define the starting and endpoints of `set_false_path` and `set_multicycle_path`. A combinatorial pin or a Q register pin is not valid.

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

Extraction File Checks

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

Validating Timing Constraints

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

To validate timing constraints, use the following command:

```
timeDesign -prePlace -outDir
```

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

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

Checking Logically Equivalent Cells Available for Optimization

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

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

Checking for Missing or Inconsistent Library and Design Data

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

Data Preparation and Validation for Low Power Designs

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

- If you are defining the MMMC setup in the CPF, make sure the CPF points to the proper libraries and constraints.
  Note: Delay corner names are 'CPF-generated', so keep that in mind when attaching RC corners and derating timing.
- For low power designs utilizing power shutdown, ensure that an always-on buffer is available

and usable. It is recommended to use `check_design -type` {`power_intent`} to validate the current design setup.

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

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

  ```
  setOptMode -opt_power_effort {low | high} -opt_leakage_to_dynamic_ratio 1.0
  ```

  Note: The power effort selected has an impact of the power-driven timing optimization, the calls to leakage reclaim, and the steps within preCTS optimization.

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

  ```
  setOptMode -opt_power_effort {low | high} -opt_leakage_to_dynamic_ratio 0.0
  ```

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

For more information on leakage, dynamic, and combined power optimization, see "Optimizing Power During optDesign" section of the Optimizing Timing chapter.

Several steps need to be performed to ensure that power optimization gives the best results and these should always be undertaken before starting all leakage and dynamic power optimization.

It is important to specify the correct leakage and dynamic view for optimization. The optimal view for leakage is the one with higher temperature corners (85/125 degrees) and typical libraries. The optimal view for dynamic power is dependent both on the design and on your inputs. For specifying the power view, consider the following:

- If the leakage and dynamic view is to be the same, then run the following command:

  ```
  set_power_analysis_mode -leakage_power_view power_view_name -
  dynamic_power_view power_view_name
  ```

  You can still use the `-analysis_view` *`power_view_name`* parameter but this parameter will be made obsolete in a future release, so it is not recommended.

- If the leakage and dynamic view is to be different, then run the following command:

  ```
  set_power_analysis_mode -leakage_power_view leakage_view_name -
  dynamic_power_view dynamic_view_name
  ```

If the view is not an active view, it will be automatically handled by the optimization code. However, the `report_power` command does not support non-active views. So, for this command, you will need to add the view to the active views using the `set_analysis_view` command and then call the `report_power -view power_view_name` command. Also, in terms of leakage, if the view is not active then the optimization will be forced to set the `-state_dependent_leakage` parameter of the `set_power_analysis_mode` command to `false`.

Note: If you want to have state-dependent leakage (`-state_dependent_leakage true`) optimization, then the view needs to be made part of the active view list. Also, it is important to ensure that the specified views used are always well defined from both a power and timing point of view to get the optimal QOR.

For dynamic power optimization, it is also recommended that you provide an activity file. This can be done by using the following command:

```
read_activity_file -format {VCD | TCF | SAIF | FSDB | PHY | SHM} file_name
```

In the absence of a switching file, it is recommended you use the following command:

```
set_default_switching_activity -input_activity 0.2 -seq_activity 0.2
```

This will ensure both predictability and consistency throughout the flow.

Flow Preparation

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

Setting the Design Mode

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

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

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

- The `setDesignMode -flowEffort` command is used to force every super command, such as `place_opt_design`, `optDesign`, `routeDesign` and so on to use their extreme-effort settings and the additional non-default options. In this mode, the target is to achieve the best possible timing/yield at the expense of some increase in the CPU runtime.
  - `express`: Configures the flow to give the best turnaround time with good WNS/area correlation as compared to the standard flow. The flow is appropriate for prototyping.
  - `standard`: Configures the flow to give the best overall combination of quality of results and full-flow turnaround time. This flow is appropriate for the majority of designs.
  - `extreme`: Configures the flow to give the best quality of results at some cost in turnaround time. This flow is appropriate for designs where timing closure is challenging.
- The following example sets the process to 45nm and effort level to extreme:

  ```
  setDesignMode -process 45 -flowEffort extreme
  ```

⚠ The `extreme` flow is part of a limited-access feature in this release. It is enabled by a variable specified using 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.

Extraction

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

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

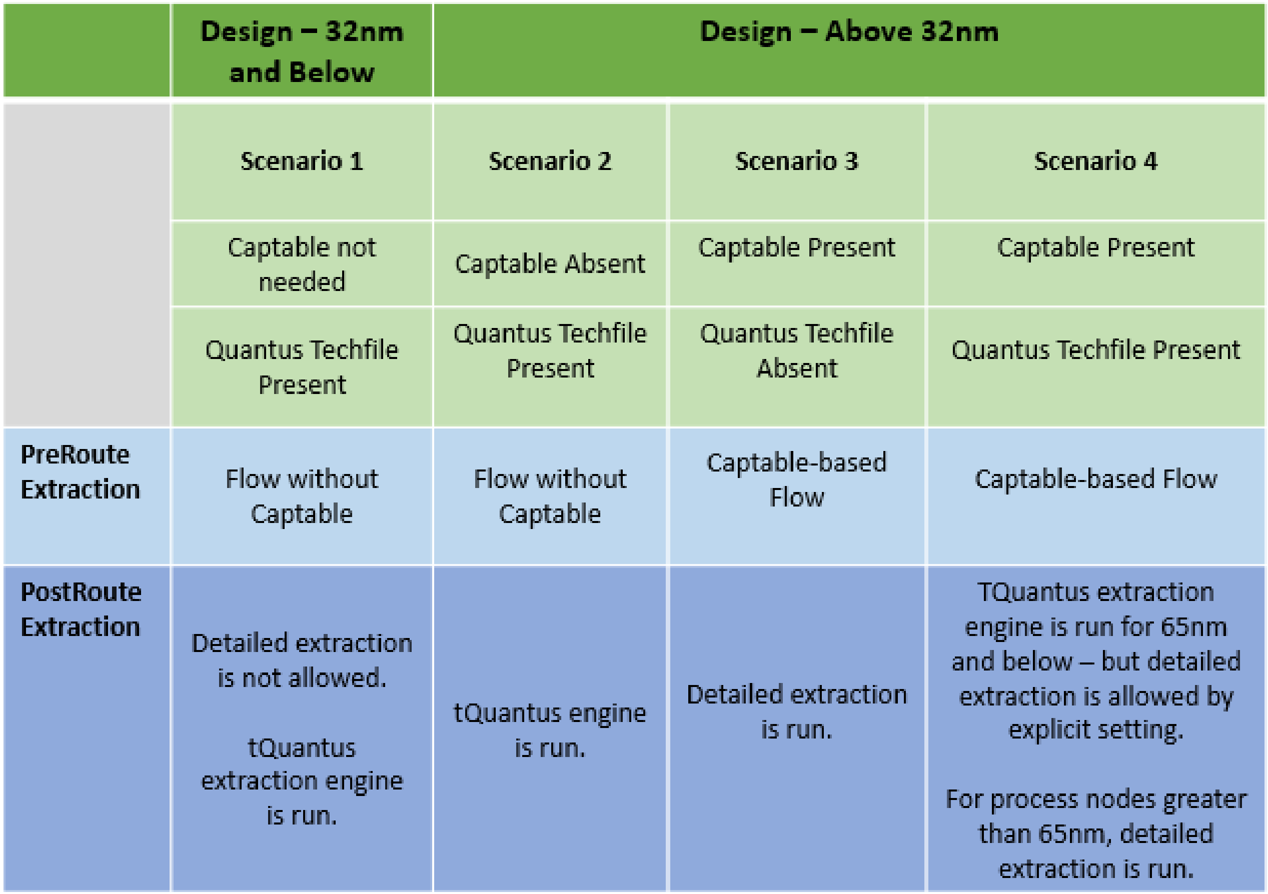
- Use the `-engine preRoute` option when the design has not yet been detail routed by NanoRoute. When the `-engine preRoute` option is set, RC extraction is done by the fast density measurements of the surrounding wires; coupling is not reported.
- Use the `-engine postRoute` option after the design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

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

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

The default value for the `-effortLevel` parameter depends on the value of `setDesignMode` settings.

The table below shows how extraction is run based on the process and whether a Quantus tech file and/or captable is provided.



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

Timing Analysis

Timing analysis is typically run after each step in the timing closure flow using the `timeDesign` command. If timing violations exist, we recommend that you use the Global Timing Debug (GTD) GUI to analyze and debug the results. The Global Timing Debug (GTD) is an invaluable tool that provides forms and graphs to help you view timing problems.

Note: To learn more about GTD, see the Global Timing Debug Rapid Adoption Kit (RAK). This RAK provides a lab and includes instructions to demonstrate the features of GTD.

Pre-Placement Optimization

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

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

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

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

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

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

Floorplanning and Initial Placement

The goals of floorplanning and initial placement include the following:

- Creating prototypes using multiple iterations with a focus on routability
- Moving toward timing-driven placement as routability stabilizes
- Adding power routing once timing and congestion converge

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

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

- Run the initial placement without any regions and guides. It is best to get a baseline placement without constraining the placer.

Early on, use the `setPlaceMode -place_design_floorplan_mode true` command to run placement in prototyping mode for a faster turnaround.

- Prototype placement does not produce legal placement so make sure you run placement with the `setPlaceMode -place_design_floorplan_mode false` command as you converge on a floorplan.
- Use the Prototyping Foundation Flow if your design contains a large number of hard macros. The Prototyping Foundation Flow utilizes Flex Models that can improve the run time by 20X, while still providing accurate area, timing and congestion analysis. The Prototyping Foundation Flow RAK provides a lab, instructions, and other information to demonstrate its features.
- Analyze the placement for timing and routability issues, and make the necessary adjustments.
- Employ module guides, placement blockages, and other techniques to refine the floorplan. The placement engine automatically detects low-utilized designs and turns on the options required to achieve an optimal placement.

Ensuring Routability

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

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

- Choose an appropriate floorplan style.
    - If possible, review a data flow diagram or a high-level design description from the chip designer to determine an appropriate floorplan style.
    - Assess different floorplan styles such as hard macro placement in periphery, island, or doughnut (periphery and island). Keep the macro depth at 1 to 2 for best CTS,

optimization, and Design-for-test (DFT) results. If possible, consider different aspect ratios to accommodate a shallower macro depth. Consider using the Prototyping Foundation Flow and relative floorplanning constraints to simplify floorplan iterations.

- Preplace I/Os and macros.
    - Review hard macro connectivity and placement based on the minimum distance from a hard macro to its target connectivity.
    - Preplace high-speed and analog cores based on their special requirements for noise isolation and power domains.
- Review I/O placement to identify I/O anchors and the associated logic.
    - Verify that logic blocks and hard macros that communicate with I/O buffers are properly placed and have optimal orientation for routability. Push down into module guides to further assess the quality of the floorplan and resulting placement.
- Allow enough space between preplaced blocks.
    - Allow space between I/Os and peripheral macros for critical logic such as JTAG, PCI, or Power management logic. Use the specifyJtag and placeJtag commands prior to placing blocks.
    - Use block halos, placement obstructions or fences around blocks prior to optimization or Clock Tree Synthesis (CTS). Placement generally does not do a good job of placing cells between macros. Reduce or remove halos and obstructions after placement to make sufficient space available around macros for optimization, CTS, DRV, or SI fixing to add buffers. Placement blockages of type Soft or Partial are useful tools to control cell placements between blocks.
    - Place other cells such as endcaps, welltaps, and decaps prior to placement, as required.
- Use module guides carefully.
    - Place module guides, regions, or fences only when greater control is required. Be careful not to place too many module constraints early in the floorplanning process because it is time consuming and greatly constrains the placement. Module guides should be used for floorplan refinement or hierarchical partitioning.
    - Review the placement of module guides related to the datapath and control logic relative to the associated hard macros. Datapath logic can be a source of congestion problems due to poor aspect ratios, high fanout, and large amounts of shifting. Consider

    tuning the locations of these module guides and lowering the density to reduce congestion.
    - Review the placement of module guides related to memories. These modules can typically have higher densities due to the inclusion of the memories.
- Reorder scan chains
    - When evaluating congestion, make sure that scan chains are reordered to eliminate "false" hot spots in the design. Failing to reorder the scan chain can cause a routable floorplan to appear unroutable. By default, `place_design` performs scan tracing and scan reordering based on global or user-specified scan reorder settings and specified or imported scan chain information. If scan chain information is missing, no reordering is performed.

Validating the Floorplan

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

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

Make sure to consider the following when finalizing the floorplan:

- The power grid should be defined:
    - Global net connections are properly defined using the `globalNetConnect` command.
    - Nets requiring optimization should be defined as signal (regular) nets. Optimization treats nets in the SPECIALNETS as dont_touch.
    - PINS marked with + SPECIAL cannot be optimized.
    - Followpin routing should align to rows/cells with the correct orientation (VDD pin to VDD followpin).
- Gaps between standard cells and blocks should be covered with soft or hard blockages. Placement cannot place cells in the area of soft blockages but optimization and CTS can.
- All blocks should be marked fixed.
- Tracks should match IO pins and placement grid (rows). Use `add_tracks` to update the tracks.

GigaPlace

As the routability of the floorplan stabilizes, you should shift your focus to placement and preCTS timing closure based on ideal clocks including:

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

Preplace optimization, Slack driven placement and interleaving optimization are enabled by default:

```
place_opt_design
```

GigaPlace will remove buffer trees, followed by slack-driven global placement, and detail placement. The new congestion-driven algorithm is faster and more stable in maintaining congestion anywhere in the flow. The wire length model is the same between global and detail placement to ensure that wirelength and congestion are maintained throughout the Innovus flow.

PreCTS optimization is also called by GigaPlace to do more interleaving between placement and congestion. It leads to better timing and congestion QOR because placement is more aware of timing and congestion critical areas. By default, preCTS timing optimization begins with the first phase in which following transforms are called: netlist simplification, high fanout net buffering (including high fanout net), multidriver buffering, DRV fixing at high level, and global optimization. After this stage, the core optimization starts for all the negative end points. While working on WNS and TNS, the software controls timing convergence by updating the design state, placement and routing, incrementally. Adaptively calling area reclaim allows you to control utilization at the preCTS stage. It also takes advantage of the upper layer characteristics by using layer assignment transform on the optimization of critical nets.

For more information on optimizing the design, refer to the Optimizing Timing chapter in the *Innovus User Guide*.

`place_opt_design` can be used to replace the old flow `place_design` + `optDesign -preCTS`.

Additionally, `place_opt_design -incremental` can be used to perform additional timing optimization after the initial `place_opt_design` call. This is the preferred incremental optimization strategy and replaces the `optDesign -incremental` approach.

Placement Analysis

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

- Review the overflow values in the log file. Typically they should be below 1% but will depend on the design and technology.

  - The following option increases the numerical iterations and makes the instance bloating more aggressive. It also automatically enables the `congRepair` command.

    ```
    setPlaceMode -place_global_cong_effort high
    ```

- Turn on the congestion map and analyze any hot spots. Although the global congestion may be low, a hot spot can make the design impossible to route.
  - Use partial placement blockages to reduce the density in specified areas.
- There is a standalone command called `congRepair` that can be called in any part of the preRoute flow to attempt to relieve congestion. The command uses `globalRoute` + incremental placement. This can have a significantly detrimental effect on timing and often requires additional `optDesign` calls (and may not converge). So, use the `congRepair` command with caution.
- Clocks are consuming more routing resources because they are routed with wider rules, greater spacing and shielding. So it is important to model this early on by reading in the clock routing constraints prior to placement. The CTS specification file and the CCOpt property should contain the routing constraints including non-default rules (NDRs), spacing, and shield.

Guidelines for PreCTS Optimization

Before starting `place_opt_design`, perform the following sanity checks for preCTS optimization:

- Review `checkDesign -all` results.
- Check that the SDC is clean.
- Check that the timing is met in zero wireload using `timeDesign -prePlace.` Violating paths in this mode are likely to be failing significantly more when actual wireloading has been applied.
- Check that the NDRs are good and well selected for NDR aware optimization – too many NDRs will slow down optimization.
- Check the don’t use report.
- Activate all required views.
- Adjust settings depending on specific scenarios: high performance (timing/power), high routing congestion, high utilization, mixture.

While performing timing optimization, check the following:

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

PreCTS optDesign Command Sequences

You can use standalone `place_opt_design` for preCTS optimization in the following ways if needed:

Note: You can use any of these features separately or in combination. Use the `setOptMode` command to control optimization behavior.

- To optimize a design after you have already run `place_opt_design`, use the following command:

  ```
  place_opt_design -incremental
  ```

- To perform rapid timing optimization for design prototyping, use the following commands:

  ```
  setDesignMode -flowEffort express
  place_opt_design
  ```

Checking and Debugging Timing Optimization Results

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

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

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

- Graphically check the critical paths, not just the first one.
- Investigate cell/net delay to identify bad buffering, bad sizing, and weak cell usage.
- Investigate routing:
  - Scenic routing on standard cells area – rework on placement.
  - Check for bad routing over macros – add soft placement blockage or modify floorplan.

  - Critical paths go through congested area. Try using cell padding (using the `specifyCellPad` or `specifyInstPad` commands) or partial placement blockages (also known as density screens) to reduce the congestion – check global density and hot spot.
  - Critical path(s) go through routing congested area – review floorplan / macro placement / narrow channels.
  - Check if the worst path is going through deep logic level – If yes, see if the initial netlist is to be further improved during synthesis.

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

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

  ```
  reportIgnoredNets -outfile fileName
  ```

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

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

- If similar paths are not meeting timing, create a custom path group for these paths and optimize them separately. See the "Path Group Optimization" section for details.
- If critical paths go through congested area, then try using cell padding (running the `specifyCellPad` or `specifyInstPad` commands) or partial placement blockages (also known as density screens) to reduce the congestion.
- Useful skew is often required on hard to close timing designs.Useful skew is now integrated into all design steps from preCTS to postRoute, as appropriate. Useful skew in all flow steps can be disabled using the following command:

  ```
  setOptMode -opt_skew false
  ```

- If clock gating checks are on the critical path, you can create separate path groups for the clock gating cells and over-constrain them. Following is a sample script to do this. Set the `$clkgate_target_slack` to your desired value. Use caution when over-constraining designs as it increases the runtime:

# Set target overshoot slack for clock gating elements preCTS (in nanoseconds):

```
set clkgate_target_slack 0.15
```

# Create separate reg2reg and clkgate groups

```
group_path -name reg2reg -from all_registers -to [filter_collection
[all_registers] "is_integrated_clock_gating_cell != true"]

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

setPathGroupOptions reg2reg -effortLevel high

setPathGroupOptions clkgate -effortLevel high -targetSlack $clkgate_target_slack
```

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

  ```
  setDontUse cellName(s)
  ```

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

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

  ```
  setOptMode -opt_setup_target_slack slack -opt_drv_margin value
  ```

Path Group Optimization

You can focus timing optimization on specific paths using path groups. If path groups are not defined, `place_opt_design` will temporarily generate two high-effort `path_groups` (reg2reg and reg2cgate).

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

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

```
group_path -name path_group_name -from from_list -to to_list -through through_list

setPathGroupOptions...

place_opt_design
```

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

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

Clock Tree Synthesis

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

For details on the capabilities and configuration of CCOpt-CTS and CCOpt, an overview of CTS engines in Innovus, and an introduction to the concepts and terminology used, see the Clock Tree Synthesis chapter in the *Innovus User Guide*.

Configuring CCOpt-CTS or CCOpt

For complete command examples, see the Flow and Quick Start section in the Clock Tree Synthesis chapter.

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

- Configure non-default routing rules (NDRs) and route types using the following commands:

  `create_route_type`

  `set_ccopt_property route_type`

- Set a target maximum transition time, and for CCOpt-CTS a target skew, using `set_ccopt_property target_max_trans` and `set_ccopt_property target_skew` commands.
- Configure which library cells CTS should use by setting the `buffer_cells`, `inverter_cells`, `clock_gating_cells`, and `use_inverters` properties.
- Create a clock tree specification from active timing constraints using the `create_ccopt_clock_tree_spec` command.

For more recommendations on settings, see the Configuration and Method section in the Clock Tree Synthesis chapter.

Running CCOpt-CTS or CCOpt

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

```
ccopt_design -cts
```

To run CCOpt to perform CTS with Clock Concurrent Optimization, use the same command but without the `-cts` parameter. An example is provided below.

```
ccopt_design
```

CCOpt and CCOpt-CTS automatically do the following:

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

Reporting after CCOpt-CTS or CCOpt

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

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

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

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

Visualization of Clock Trees after CCOpt-CTS or CCOpt

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

PostCTS Optimization

The goals of postCTS optimization include:

- Fixing remaining design rule violations (DRVs)
- Optimizing remaining setup violations
- Optimizing hold timing violations

PostCTS SDC Constraints

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

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

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

Remove or change the SDC constraints that are not valid postCTS like clock uncertainty or clock latency.
You may need to adjust the clock latencies on the IOs so that IO timing does not become the critical path by
adjusting the virtual clock source latencies.
Note: CCOpt and CCOpt-CTS automatically adjust source latencies. If FE-CTS is used, you can adjust IO constraints directly.
	`clock_uncertainty`
	`clock_latency`
	`set_clock_latency -source xxx` *`clock`*
	`set_input_delay or set_output_delay`
	OR
	`update_io_latency`

Adjust derating applied to each delay corner.	

Clock routing will use a different scale factor than the signal nets.	**Signal nets:**
	`create_rc_corner/update_rc_corner -preRoute_cap`
	Clock nets:
	`create_rc_corner/update_rc_corner -postRoute_clkcap` (if defined) or
	`-postRoute_cap.`

Run timing analysis to check the timing.	`timeDesign -postCTS -outDir ctsTimingReports`

Note: If the timing is not similar to preCTS timing results, check the following:

- Was CTS able to achieve the skew constraints?
- Are these skew constraints within the `set_clock_uncertainty` set during preCTS timing?
- Was the clock uncertainty adjusted after CTS to only model jitter?

PostCTS Setup Optimization Command Sequences

For flows that deploy full CCOpt, and not just CCOpt-CTS, that additional postCTS setup optimization is not normally required. CCOpt is discussed further in the Clock Tree Synthesis chapter.

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

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

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

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

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

- To optimize timing after the clock tree has been built, use the following commands:

  ```
  optDesign -postCTS -outDir postctsOptTimingReports
  ```

- PostCTS incremental optimization takes advantage of useful skew by default starting from Innovus 16.1 release.

Hold Optimization

At this point, run timing analysis to report hold violations:

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

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

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

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

Following are recommendations to achieve closure for hold timing:

- Ensure that the hold timing uncertainty is realistic
    - Too large a value can cause the insertion of thousands more buffers
- Ensure that delay cells are allowed to be used and avoid providing very weak buffers for hold fixing because they are more sensitive to routing detour and signal integrity
- For Multi-Vth design, ensure that you run leakage optimization (through `optDesign` or `optPower`) before running hold fixing since leakage reduction improves hold timing.
- Add cell padding during placement to leave more space for hold fixing.
    - Cell padding must be removed before postCTS hold fixing
- Ensure that timing constraints are correctly in sync between setup and hold (especially multicycle paths)
- Ensure that the skew in hold is also good.
    - A good clock tree for hold is as important as for setup
- For a design with many hold violated paths, it is highly recommended to run hold fixing at the postCTS stage already (although there is no need to achieve 0ns slack at this stage). Then, use postRoute hold fixing to fix the remaining violations. You can use a negative hold target slack to focus hold fixing on the paths with large violations and fix the remaining hold violations after routing. For example, the following sets a hold target slack of `-200ps`:

  ```
  setOptMode -opt_hold_target_slack -0.2
  ```

  Note: Additionally, `optDesign -hold` can be run with the `-holdVioData` *`fileName`* option to print detailed information about the reasons for remaining violations.
- By default, hold fixing can degrade setup TNS (but not Setup WNS). This can be changed through the following:

  ```
  setOptMode -opt_hold_allow_setup_tns_degradation true | false
  ```

- To exclude some path_groups from hold fixing, apply the following:

```
setOptMode -opt_hold_ignore_path_groups {groupA groupB...}
```

- In the `innovus.logv` file hold fixing will print a detailed report where each net that is not buffered will be sorted through several categories.
  - This will allow the user to identify what are the most critical issues to resolve.
  - It will help the user in understanding why a given net was not buffered.

  Below is an example:

```
======================================================================
                Reasons for remaining hold violations
 ======================================================================

 *info: Total 1 net(s) have violated hold timing slacks.

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

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

```
setOptMode -opt_hold_allow_overlap true
```

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

Detailed Routing

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

- Routing the design without DRC or LVS violations.
- Routing the design without degrading timing or creating signal integrity violations.

- Using DFM techniques such as multi-cut via insertion, wire widening, and wire spacing to optimize the yield.

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

Routing Command Sequence

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

```
routeDesign
```

If you have timing information loaded, the `routeDesign` command automatically sets `setNanoRouteMode -route_with_timing_driven true`. The `routeDesign` command automatically unfix the clock nets (`setNanoRouteMode -route_fix_clock_nets false`) so it can ECO route the clock nets after postCTS optimization and resolve DRC violations.

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

```
setNanoRouteMode -route_with_timing_driven false
setNanoRouteMode -route_detail_post_route_spread_wire true
routeDesign -wireOpt
setNanoRouteMode -route_detail_post_route_spread_wire false
```

Double cut via effort is set with `setNanoRouteMode -route_detail_use_multi_cut_via_effort {low | medium | high}`

Improving Timing during Routing

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

- Check with your library provided or foundry for the latest technology LEF to use.
- Check the definition of tracks in the DEF file. If the tracks are poorly defined, regenerate tracks with the `add_tracks` command.
- If there is local or global congestion, return to placement, optimization and postCTS optimization to optimize further until congestion is resolved.
- Make sure the top max routing later is set appropriately.

- Check the NonDefaultRules (NDRs) and shielding and layer constraints.

PostRoute Extraction

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

- Specify the engine to postRoute:

  ```
  setExtractRCMode -engine postRoute
  ```

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

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

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

Checking Timing

Use the following commands to do a postRoute timing check on non-SI timing to compare with the preRoute timing:

```
setDelayCalMode -SIAware false

timeDesign -postRoute -outDir postrouteTimingReports

timeDesign -postRoute -hold -outDir postrouteTimingReports
```

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

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

  Note: Once the scaling factors are determined, specify them in your MMMC setup using the `create_rc_corner` or `update_rc_corner` commands.
- Is the routing topology similar between postCTS and postRoute wires? Compare the same paths between postCTS and postRoute databases and for large differences in loads

PostRoute Optimization

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

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

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

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

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

- An increase or decrease in incremental delay on a net due to the coupling of its neighbors and their switching activity.

- Glitches (voltage spikes) that can be caused in one signal route by the switching of a neighbor resulting in a logic malfunction.

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

Data Preparation for SI Analysis

SI optimization requires the following preparation:

- Make sure ECSM/CCS noise models or cdB libraries are provided for each cell for each delay corner.
- You must be in the on-chip variation (OCV) mode to see simultaneous clock pushout/pullin and enable CPPR (Clock Path Pessimism Removal). Enable this feature using the following settings:

  ```
  setAnalysisMode -analysisType onChipVariation -cppr both
  ```
- Enable SI CPPR using `set_global timing_enable_si_cppr true`. (This is the default setting.)

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

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

  ```
  setNanoRouteMode -route_with_timing_driven true
  ```

  ```
  setNanoRouteMode -route_with_si_driven true
  ```
- Fix transition time violations. This is automatically done as one of the first steps when the `optDesign -postRoute` command is run.
  - Slow transitions introduce a larger delay penalty or incremental delay and can more readily be victim to faster switching SI aggressors coupled to the net.

PostRoute Optimization Command Sequences

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

```
optDesign -postRoute
```

```
optDesign -postRoute -hold
```

Note: PostRoute setup and hold optimization can be combined to reduce runtime by using the following command:

```
optDesign -postRoute -setup -hold
```

Analysis and Debug of PostRoute Optimization Results

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

- For multi-VT designs, you can perform a LEF-Safe Optimization with only cell swapping by setting the following:
  ```
  setOptMode -opt_allow_only_cell_swapping true
  optDesign -postRoute
  ```
  - Doing so after a normal `optDesign -postRoute` may help close timing if the design is congested and those final few paths were detoured by NanoRoute's ECO routing.
  - Doing so prior to a normal `optDesign -postRoute` may speed up closure in terms of turnaround time.
  - Only works for multi-VT libraries
  - Does not have much impact when design has already mainly LVT cells.
- Make sure your extraction filters correlate to your signoff extraction
- When using IQuantus the filters typically can be set to the exact signoff values. Make sure to use `setExtractRCMode -capFilterMode relAndCoup`

Below are suggestions to help achieve SI Closure:

- SI prevention in the preRoute flow for data path can be achieved by adding more pessimism to force optimization to work harder. This can be done by increasing the clock uncertainty.
  - Choose reasonable values to avoid over fixing and increasing area/power too much
- You can apply a targeted approach for cases with the timing paths with very large depth (> 40)
  - Sum of small SI push-out on long paths leads to large timing penalty.
  - Solution is to add a pessimism only on nets that are part of the large depth path.

- Check the global max transition and ensure that wire spreading is enabled during detailed routing.

Optimizing With Third-Party SPEF

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

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

```
spefIn rc_corner1.spef -rc_corner rc_corner1

spefIn rc_corner2.spef -rc_corner rc_corner2

...

optDesign -postRoute [-hold] -outDir spefFlowTimingReports
```

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

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

Chip Finishing

Once postRoute optimization is performed, you may need to add filler cells and metal fill shapes to meet the DRC rules. The filler cells are recommended to be inserted before `routeDesign`, with proper `setFillerMode` settings. The postRoute optimization can automatically delete the fillers before optimizing, restore the deleted fillers back, and insert fillers again before eco route. In the ECO stage, `deleteFiller` can be used to free the placement space. Filler cells are used to fill the gaps between standard cells in the standard cell rows to fill in the device layers like pwells or nwells, and to meet any implant layer width or spacing DRC rules. The normal cells have implant layers but they may not meet the width and spacing rules of the implant layers without abutting filler cells that use the correct implant layers. An example of the command to insert filler cells is provided below:

```
addFiller -cell filler_cell_list -prefix FILLER
```

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

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

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

```
[PROPERTY LEF58_FILLTOFILLSPACING "FILLTOFILLSPACING spacing ;" ;]

[FILLACTIVESPACING spacing ;]

[MINIMUMDENSITY minDensity ;]

[MAXIMUMDENSITY maxDensity ;]

[DENSITYCHECKWINDOW windowLength windowWidth ;]

[DENSITYCHECKSTEP stepValue ;]
```

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

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

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

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

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

addMetalFill
```

Use the commands listed below to insert sign-off metal fill in the design. This flow calls the Cadence Pegasus application or Cadence PVS application depending on which product you are using for sign-off metal-fill insertion. This is the recommended flow:

If using Pegasus:

```
streamOut -mapFile gds_map -outputMacros -units unit
```

```
run_pegasus_metal_fill -ruleFile PEGASUS_RULE_DECK -defMapFile  -gdsFile -cell
gds_top_cell
```

If using PVS:

```
streamOut -mapFile gds_map -outputMacros -units unit
```

```
run_pvs_metal_fill -ruleFile PVS_RULE_DECK -defMapFile  -gdsFile -cell gds_top_cell
```

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

```
setMetalFill -windowStep x_step y_step -windowSize x y
```

```
trimMetalFillNearNet -slackThreshold $slack1 -
spacing value -spacingAbove value -spacingBelow value -minTrimDensity value
```

Timing Sign Off

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

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

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

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

These commands perform the following operations:

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

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

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

Final Timing Analysis and Optimization using Tempus/Quantus

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

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

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

Additional Resources

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

- *Innovus Foundation Flows User Guide*
- *How to Generate Scaling Factors for RC Correlation*

Foundation Flow

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

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

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

Related Information

Design Implementation Flow

Hierarchical and Prototyping Flow

- Hierarchical and Prototyping Flow Overview
- Top-down and Bottom-up Hierarchical Methodologies
  - Top-down Methodology
  - Bottom-up Methodology
- Hierarchical Floorplan Considerations
  - Hierarchical Methodologies
- Hierarchical Partitioning Flow and Capabilities
  - Hierarchical Partitioning
- Chip Planning
  - FlexModel
  - Timing Net Delay Model with Pico-second Per Micron (psPM)

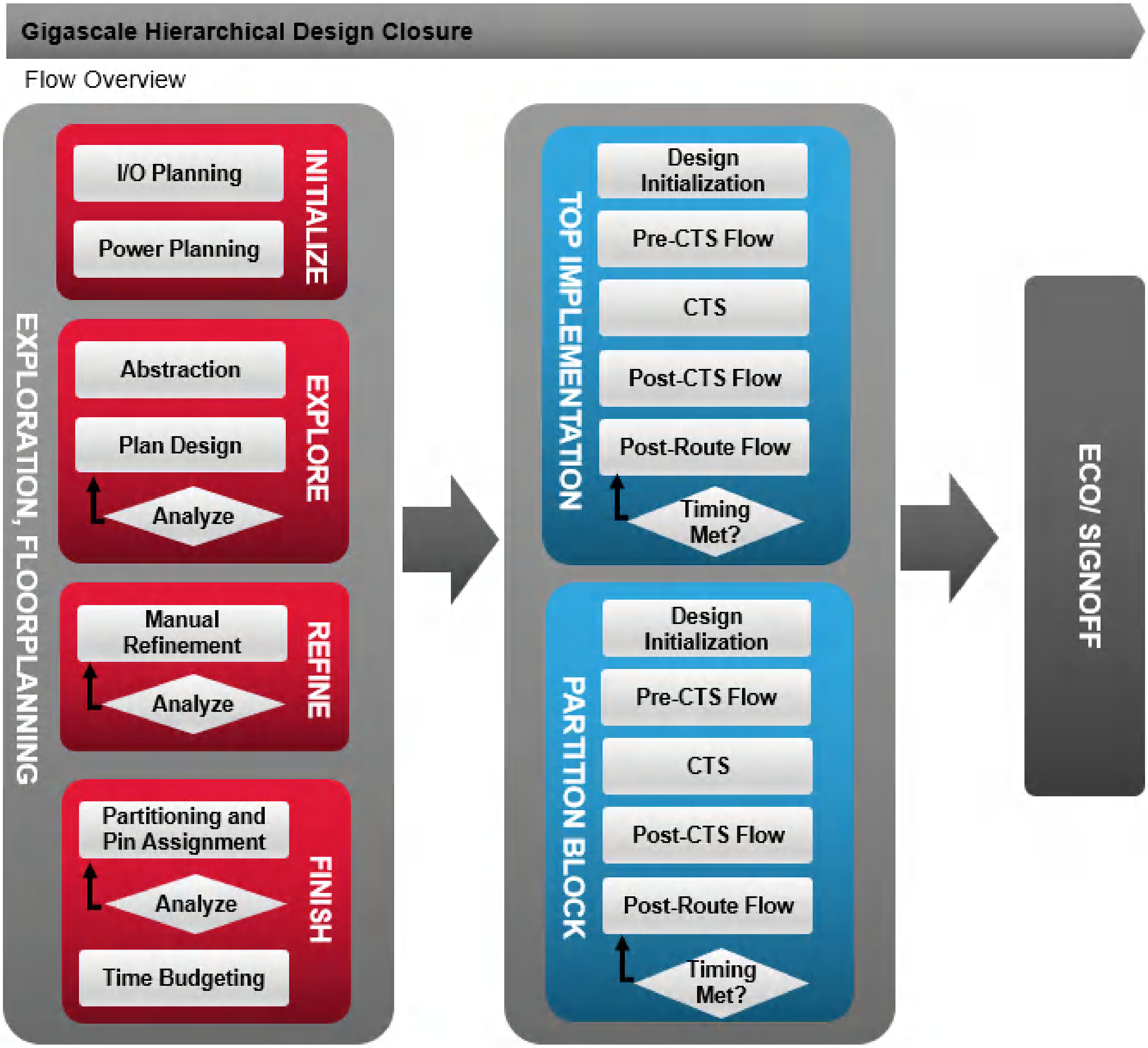
Hierarchical and Prototyping Flow Overview

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, as the design size grows beyond a few million instances, Flat design flow becomes unavailable due to huge memory requirements and excessive run time. Hierarchical Flow capabilities are essential to divide and conquer the design process – where the design can be divided into manageable partitions, and each partition can be independently assigned to different design groups to be developed in parallel.

However, Hierarchical Flow presents new challenges – such as increased complexity of Prototyping, Planning and Partitioning the design, Hierarchical timing closure at the top level, and managing the late ECOs. Design planning and hierarchical timing closure contribute significantly to Hierarchical design turnaround time. These challenges need advanced planning and modeling capabilities.

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

- **Chip Planning**
  Breaks down a design into block-level designs to be implemented separately.
- **Implementation**
  This stage consists of two sub-stages: block implementation for a block-level design and top-level implementation for a design based on block-level design abstracts and timing models.
- **Chip Assembly**
  Connects all block-level designs into the final chip.



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

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

Top-down and Bottom-up Hierarchical Methodologies

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

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

Top-down Methodology

Chip Planning

Chip planning involves optimal planning of resources, pins, feedthrough paths and timing, and aims to deliver better performance, area, timing and power planning. The Chip Planning section describes the most common flow for chip planning which includes specifying partitions and blackboxes.

> ⚠ **Note:** For information on Chip Planning with the Integrated Hierarchical Database (iHDB) flow, refer to the "Chip Planning" section of the Partitioning the Design chapter.
> To know more about the Stylus Hierarchical Database Flow, see Stylus Hierarchical Database Flow.

Top and Block Implementation

After the chip planning, the next stage is to implement the individual blocks. Each block is implemented using the constraints for timing, size, and pin assignment determined during the planning stage. Block implementation should be done at a block directory that is generated by the `savePartition` step.

⚠ **Note:** For the Integrated Hierarchical Database (iHDB) flow, block implementation must be done with the pnr model that was generated by the `savePartition` step. To restore a block or top-level design, `restore_module_model -cell` *`cell_name`* `-tag` *`tagName`* should be used.

Note: At the completion of this step, all needed models such as 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 such as .lib and ILM, and FlexILM (Refer to the "Top-level Timing Closure Methodologies" section for more information), 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:** For the Integrated Hierarchical Database (iHDB) flow, chip assembly is done using `set_module_model` and `commit_module_model` commands.

Note: For the top-down approach in the iHDB flow, see the "Chip Assembly for iHDB Flow" section of the Partitioning the Design chapter. It contains information on how to bring together all the top-level and block-level netlists and routing information in the iHDB flow.

To know more about the Integrated Hierarchical Database Flow, see Integrated Hierarchical Database Flow.

Bottom-up Methodology

Implementation

Implementation in this section means completing place and route, including all I/Os, clock trees and power.

Block Implementation

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

Top-level Implementation

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

Chip Assembly

For the bottom-up approach, see the "Chip Assembly " section of the Partitioning the Design chapter for information on how to bring together all the top-level and block-level netlists and routing information.

Hierarchical Floorplan Considerations

- Prototype the design to plan up-front as much as possible
- Pick a partition size that is optimal for a flat block implementation
- Match logical and physical hierarchy
- Find better partitioning in terms of:

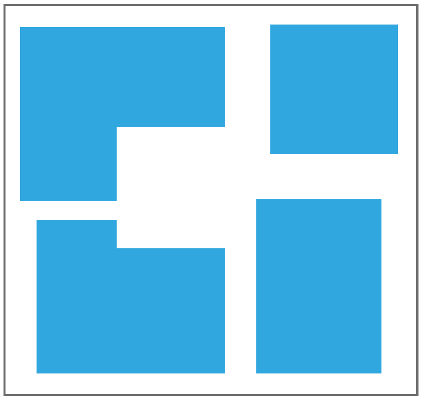
  - Minimal top-level timing paths or logics
  - Simple partition interface timing
  - Minimal pin count
- Create a hierarchical Verilog module wrapper if feedthrough insertion is needed
  Note: The hierarchical Verilog module wrapper is required if you do not want to change the original block netlist due to a requirement to change test vectors for simulation or to avoid running into LEC problems later on if using third party LEC tools that do not support feedthroughs.
- Do register-bounded on the partition interface
  - Interface latches are not good for modeling, for example ILM

Hierarchical Methodologies

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

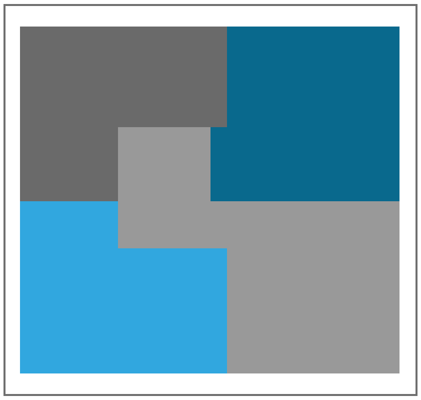
Channel-based Methodology

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



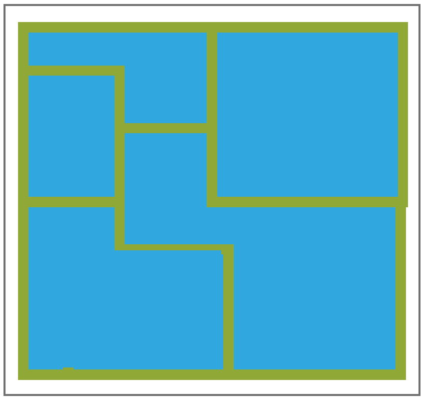
Channel-less Methodology

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



Narrow Channel Methodology

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



Hierarchical Partitioning Flow and Capabilities

The fundamental Innovus capabilities for Hierarchical Flow are given below. For more information on these capabilities, see Partitioning the Design chapter of the *User Guide*.

Hierarchical Partitioning

Capabilities

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

Pin Assignment

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

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

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

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

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

Feedthrough Insertion

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

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

Timing Budgeting

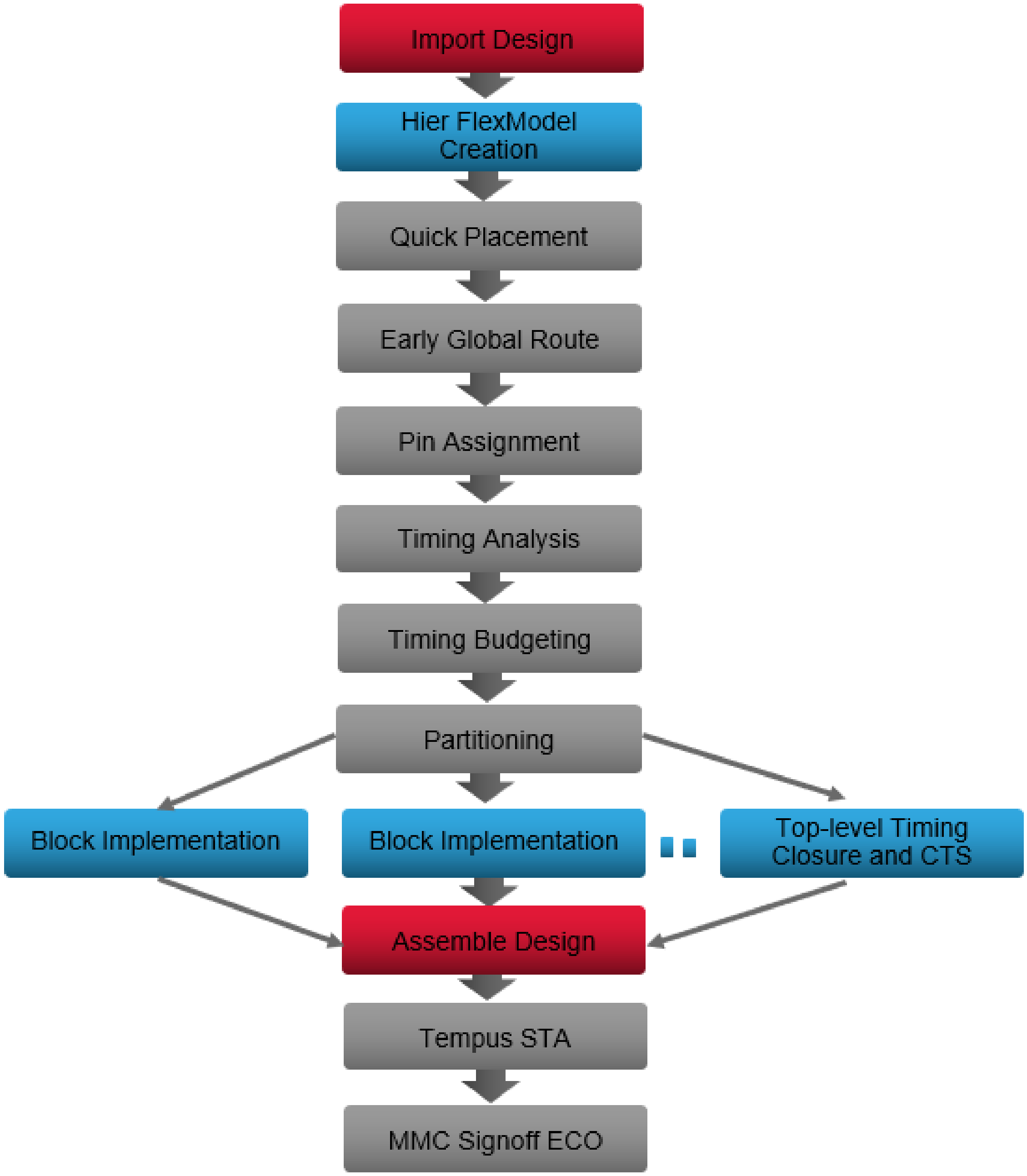
In hierarchical design flows, chip-level timing constraints must be mapped correctly to corresponding block-level constraints while partitioning the design. Innovus automatically apportions budgets to blocks using a path-based method. Since the budgeting is run on an early-stage non-optimized design,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

Assembling Partitions

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

Hierarchical Partitioning Flow

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



⚠ **Note:** For more information on these flow steps, see the Partitioning the Design chapter of the *User Guide*. To know more about the Stylus Hierarchical Database Flow the, see Stylus Hierarchical Database Flow.

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

Chip Planning

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

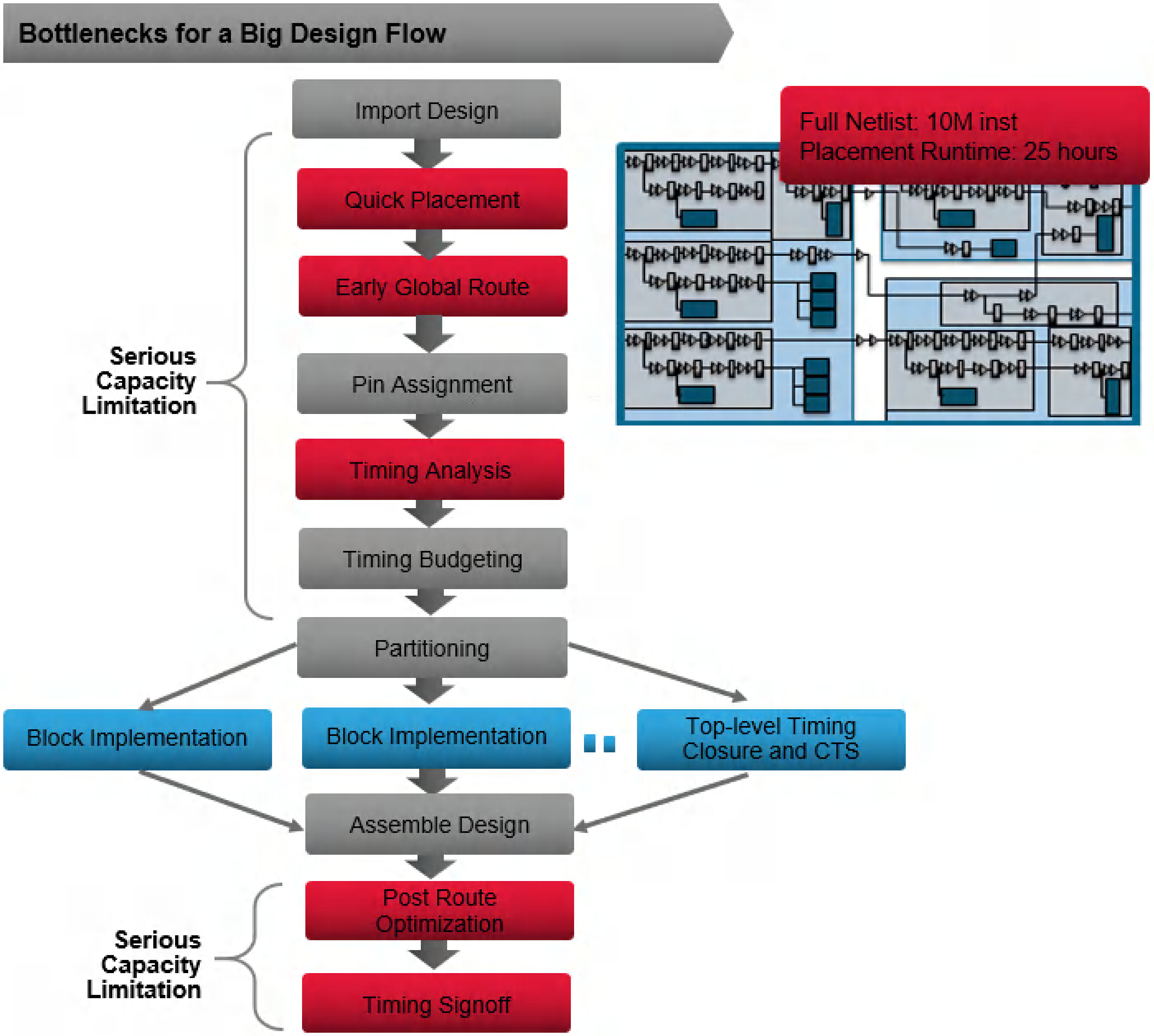
- FlexModel
- Timing Net Delay Model with Pico-second Per Micron (psPM)
- Prototyping Flow

⚠ **Note:** FlexModel and psPM generation do not support the Integrated Hierarchical Database (iHDB) flow yet.

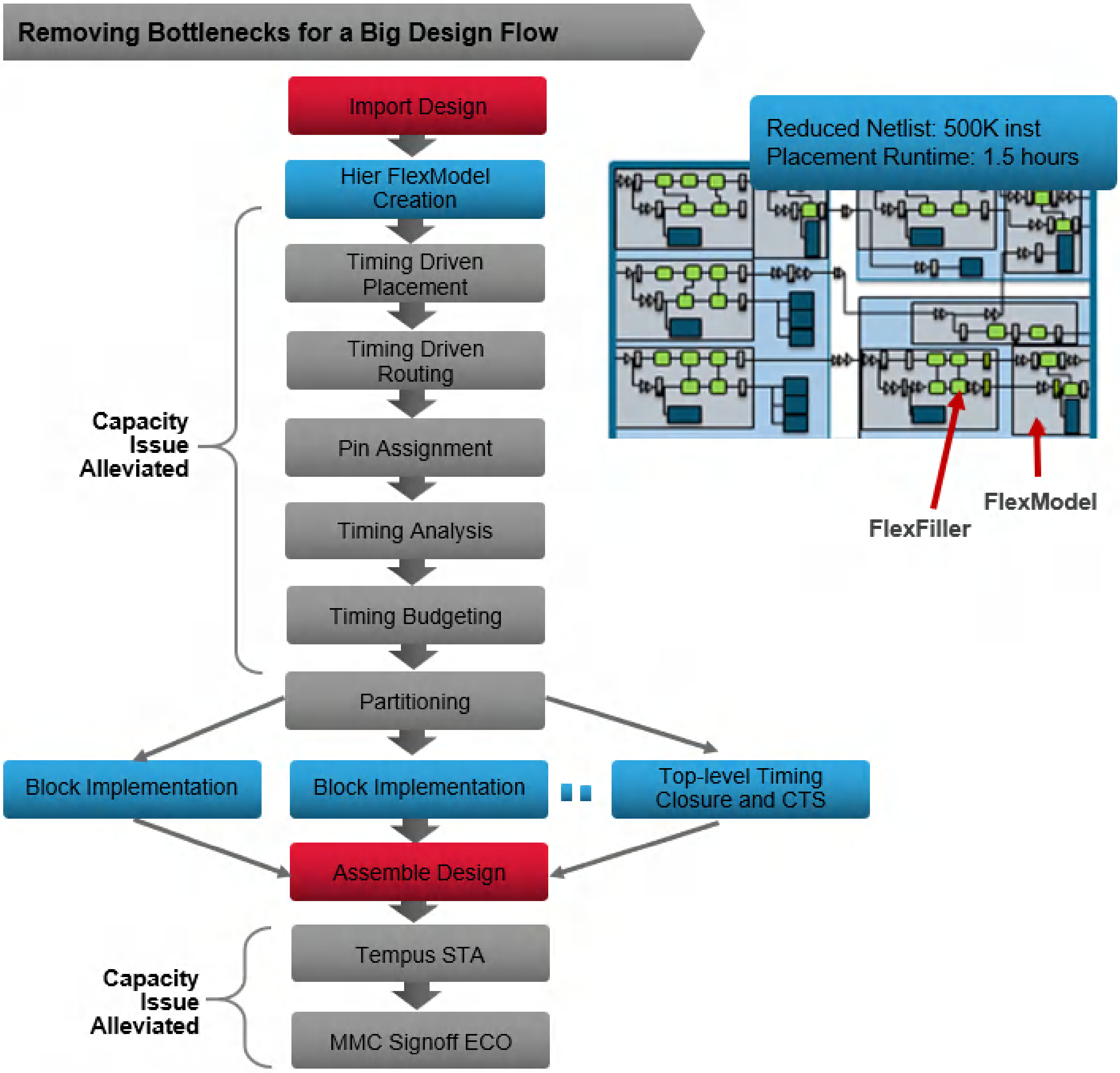
FlexModel

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

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



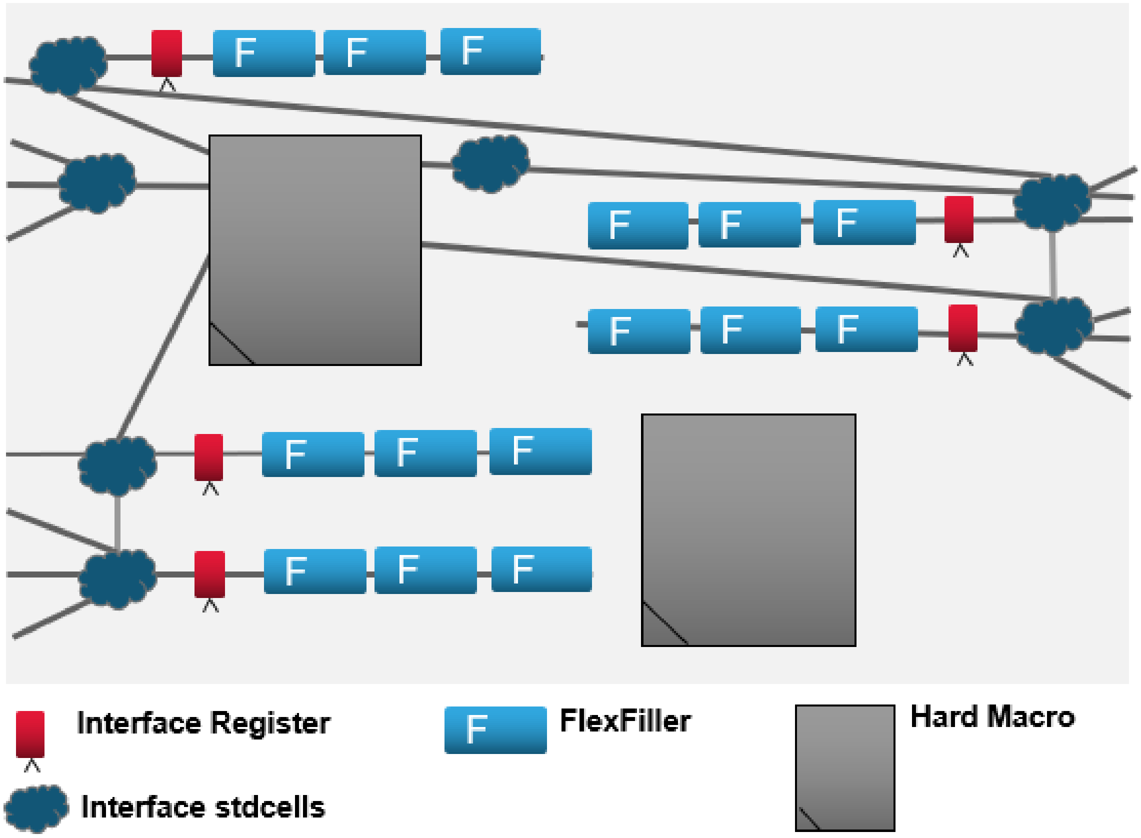
To address these challenges, FlexModel can be used to fasten the overall process. The FlexModel flow requires an additional FlexModel generation step after restoring the design. Other than that, the flow is almost the same as the normal hierarchical flow.



FlexModel - Introduction

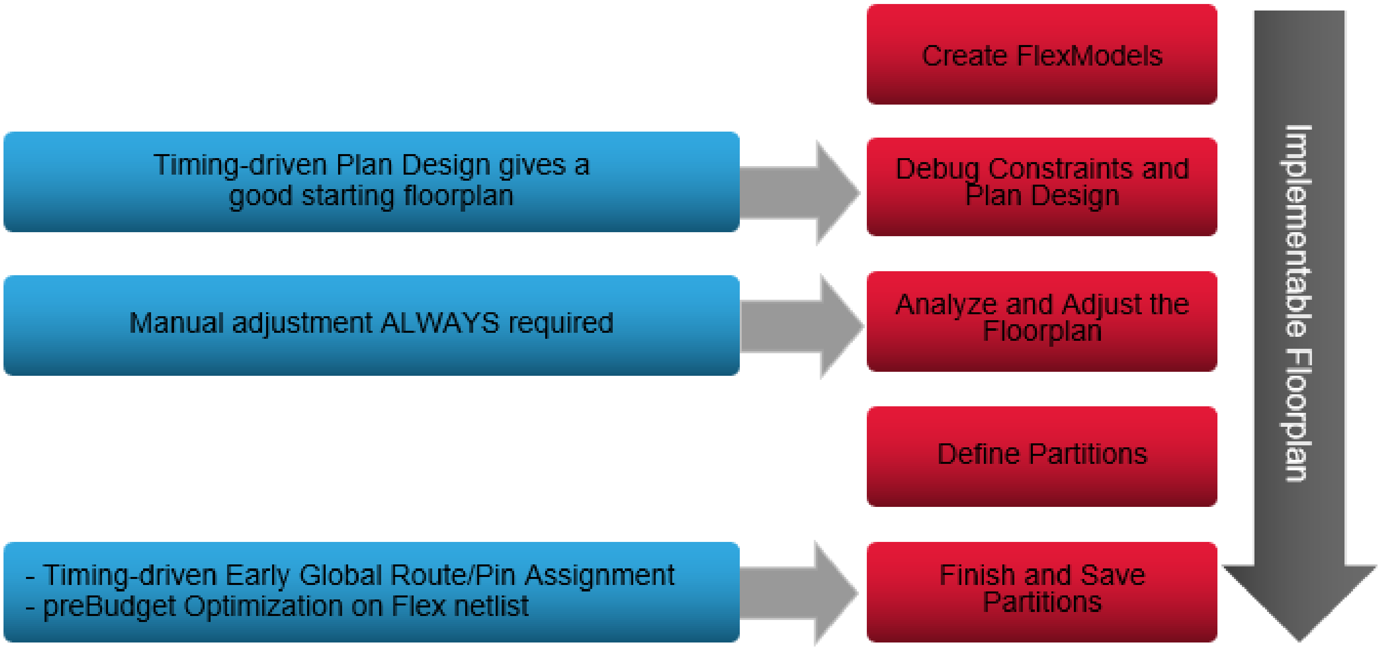
A FlexModel can be a Verilog module or an instance group. It contains macros, interface standard cells, and FlexFillers. FlexFillers fill in the space for the internal register-to-register logic. They do not have timing models associated with them, and connect such that the placer will place them close together in one group. This helps in accurate timing and area estimation. A FlexModel netlist is usually one-tenth the number of instances of its full netlist. It is used during early design planning to reduce the run time and memory while accurately modeling the timing and area.

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



FlexModel Prototyping Flow Stages

Following are the prototyping flow stages:


Create FlexModels
Timing-driven Plan Design gives a good starting floorplan
Debug Constraints and Plan Design
Manual adjustment ALWAYS required
Analyze and Adjust the Floorplan
Define Partitions
- Timing-driven Early Global Route/Pin Assignment
- preBudget Optimization on Flex netlist
Finish and Save Partitions
Implementable Floorplan

- Create FlexModels
  - Identify models
  - Create models
  - Create timing net delay model: Pico-second-Per-Micron model (psPM)
- Debug Constraints and Plan Design
  - Debug constraints
  - Generate a good initial floorplan that can be used as starting point for manual modification
- Analyze and Adjust the Floorplan
  - Manually move FlexModels to shorten timing paths
  - Re-analyze floorplan
- Define Partitions
  - Define partitions based on FlexModel placement

    - Optionally re-place macros within partition fences
    - Optionally manually re-shape and position FlexModel regions within partition boundaries
    - Generate partition fences
- Finish and Save Partitions
    - Placement and early global route
    - Commit/Save partitions
    - Pin assignment, feedthrough insertion, budgeting
    - Power, bus and pipeline planning

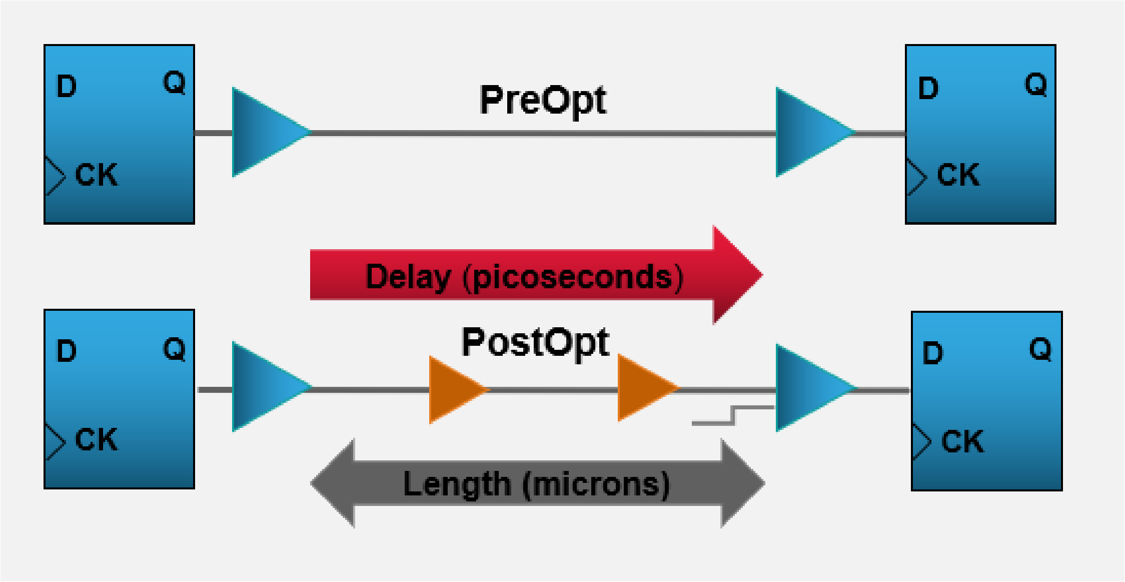
For more information on creation of FlexModel, model generation, see the Prototyping Methodologies chapter of the *User Guide*.

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

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

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

With prototype timer that uses psPM models, timing can be accurately estimated with a quick TAT. Since the same gigaOpt engine (`optDesign`) is used for characterizing and generating psPM models, timing results with psPM models are correlated well with `optDesign -preCTS` results (~10%)



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

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

Prototyping Flow

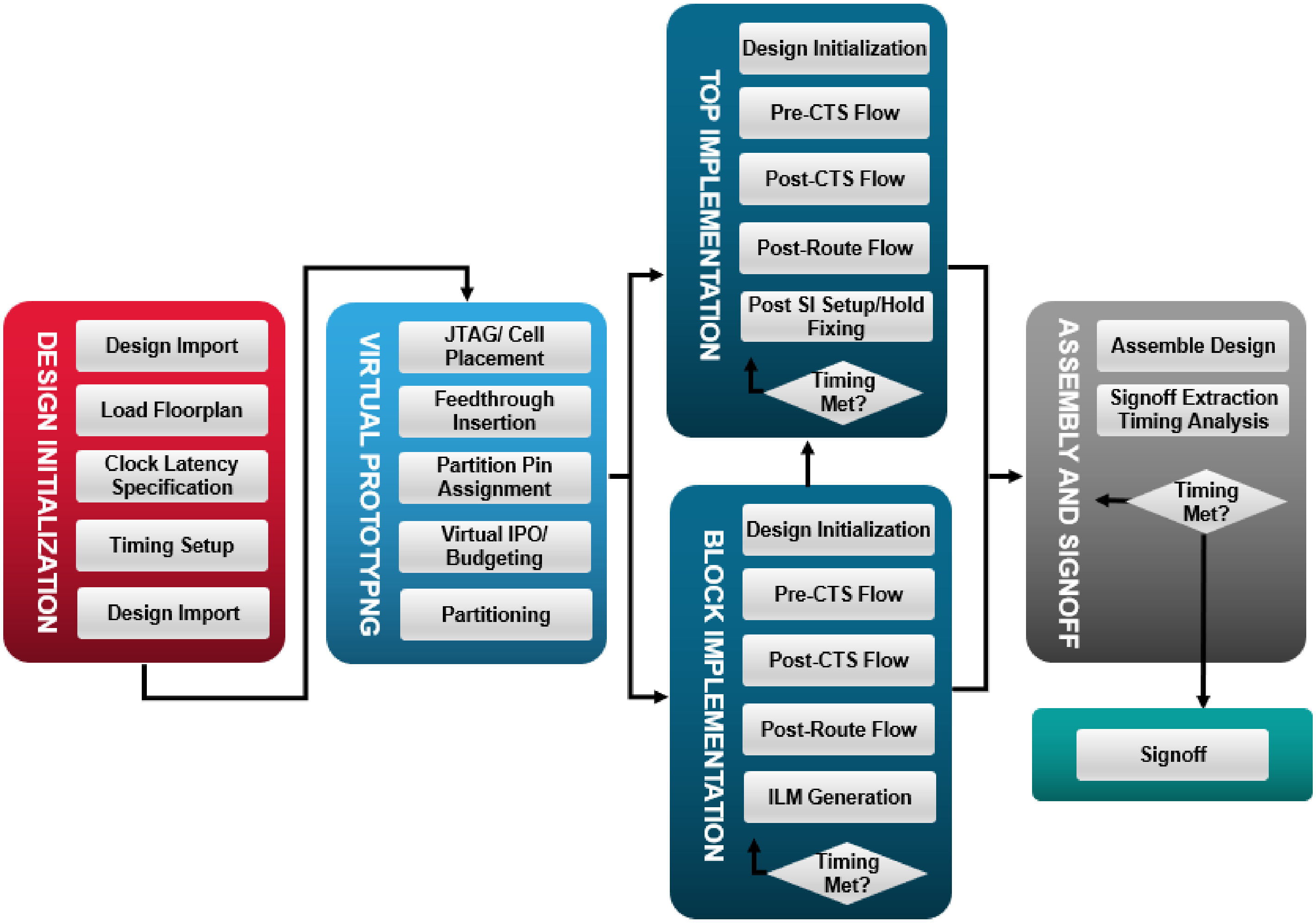
Overview

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

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

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

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

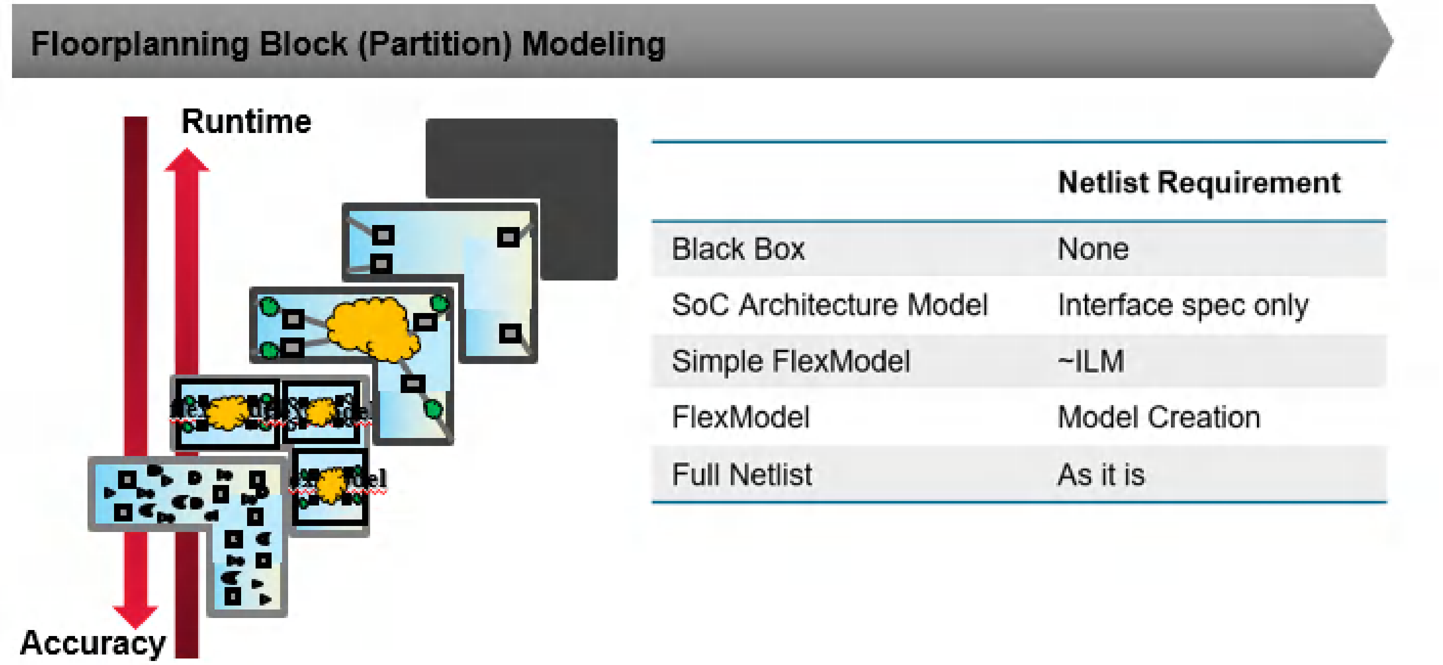


Prototyping Methodologies

Innovus prototyping flow supports different abstract models such as:

- Black box
- Soc Architecture Information (SAI)

- FlexModel



Floorplanning Block (Partition) Modeling

Runtime

Accuracy

	Netlist Requirement
Black Box	None
SoC Architecture Model	Interface spec only
Simple FlexModel	~ILM
FlexModel	Model Creation
Full Netlist	As it is

As shown in the above diagram, there is a tradeoff between accuracy and run time. As the details in the models increase, the accuracy improves. However, it comes at the cost of increased run time and memory. For example, usage of the BlackBox model during prototyping enables quicker run time but lesser accuracy. Next in line is the SoC Architecture Model, where some minimum details are covered till the boundary flops. It provides more accuracy in terms of Quality of results, but with a marginal increase in run time. FlexModels improves the modeling accuracy even further, leading to better accuracy but with slight increase in runtime.

Note: The netlist of a FlexModel is similar to an ILM netlist where interface boundary logics are kept and internal logics between registers are removed.

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

BlackBox

Normally a blackbox is a module with content that is not well defined. However, a well-defined module can also be defined as a blackbox. A blackbox is similar to a hard block, but like a fence, a blackbox can be resized, reshaped, and have pins assigned. After a blackbox has its pins assigned and is partitioned, it behaves like a hard block. The blackbox feature can be used only with a partitioned design. After the netlist has been loaded, you can further specify which modules or cells will be regarded as blackboxes, or modify the existing blackbox sizes. A blackbox size can be specified in terms of an estimated area (an actual value or an area value in terms of gate count), or a fixed block width and height.

Blackbox-based Flow

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

- Import the design.
  Note: Users can import their blocks even if they are partially defined, or undefined, and then convert them into Blackboxes to continue with prototyping flow. During importing, the Innovus software by default would keep the empty modules.
  Specify the blackboxes or load a floorplan file with blackbox information.
- Floorplan the design.
- (Optional) Save the design, which saves the blackbox information.
- Run placement.
- (Optional) Run `earlyGlobalRoute`.

Saving Blackbox

Proceed with the normal hierarchical flow for the design. To save blackbox information, use the `saveDesign` command or the *File - Save Design* menu command.

Reshaping Blackbox

During `proto_design`, 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.

Removing Blackbox Specifications

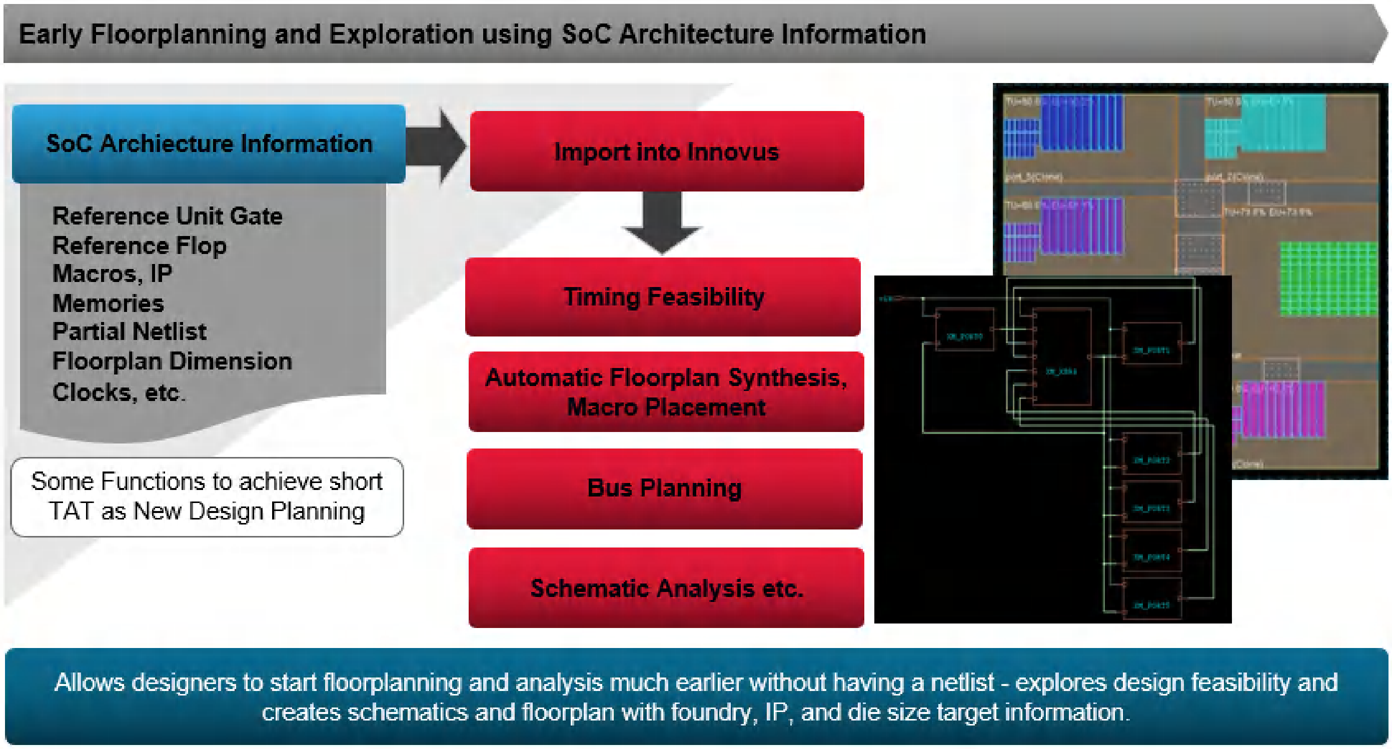
A blackbox can be unspecified by using the `unspecifyBlackBox` command. If the blackbox is an empty module in the netlist, then you can also convert it to a partition fence using the `convertBlackBoxToFence` command.

Soc Architecture Info (SAI)

Soc Architecture Information (SAI) is new methodology for prototyping that is available from the 14.1 version of the software onwards. SAI allows design exploration floorplan creation and analysis at a much earlier stage in the design flow. It is very useful in very early chip size study, hierarchical floorplan analysis. Using this methodology you can begin design feasibility without a complete netlist and get to enable Innovus floorplanning.

Soc Architecture Info (SAI)

Soc Architecture Information (SAI) is new methodology for prototyping that is available from the 14.1 version of the software onwards. SAI allows design exploration floorplan creation and analysis at a much earlier stage in the design flow. It is very useful in very early chip size study, hierarchical floorplan analysis. Using this methodology you can begin design feasibility without a complete netlist and get to enable Innovus floorplanning.



Early Floorplanning and Exploration using SoC Architecture Information

SoC Archiecture Information

Reference Unit Gate
Reference Flop
Macros, IP
Memories
Partial Netlist
Floorplan Dimension
Clocks, etc.

Some Functions to achieve short TAT as New Design Planning

Import into Innovus

Timing Feasibility

Automatic Floorplan Synthesis, Macro Placement

Bus Planning

Schematic Analysis etc.

Allows designers to start floorplanning and analysis much earlier without having a netlist - explores design feasibility and creates schematics and floorplan with foundry, IP, and die size target information.

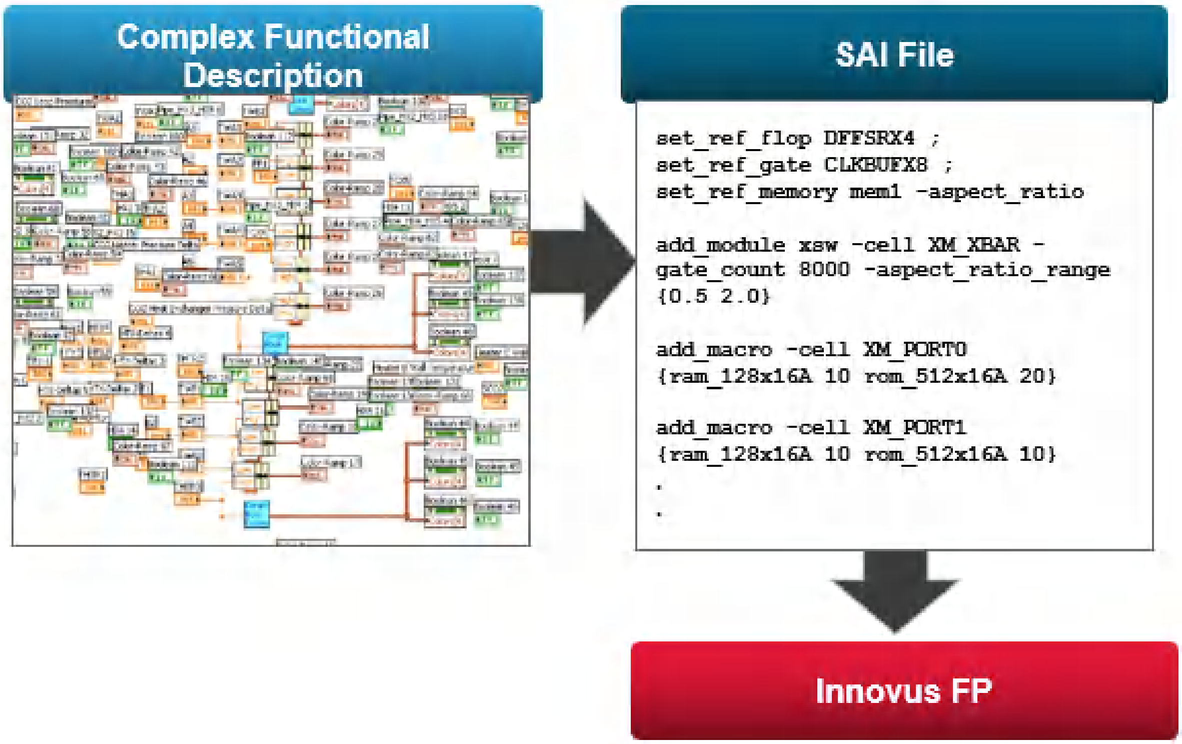
SoC Architecture Info (SAI) File

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

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

- Clocks
- Floorplan dimension

SAI architecture as seen can contain the details, such as reference gate, IP information, reference flop, partitions, partial netlist, clock, and floorplan information.
Basically one can build upon the design by providing this content information using the SAI commands.



```
set_ref_flop DFFSRX4 ;
set_ref_gate CLKBUFX8 ;
set_ref_memory mem1 -aspect_ratio

add_module xsw -cell XM_XBAR -
gate_count 8000 -aspect_ratio_range
{0.5 2.0}

add_macro -cell XM_PORT0
{ram_128x16A 10 rom_512x16A 20}

add_macro -cell XM_PORT1
{ram_128x16A 10 rom_512x16A 10}
.
.
```

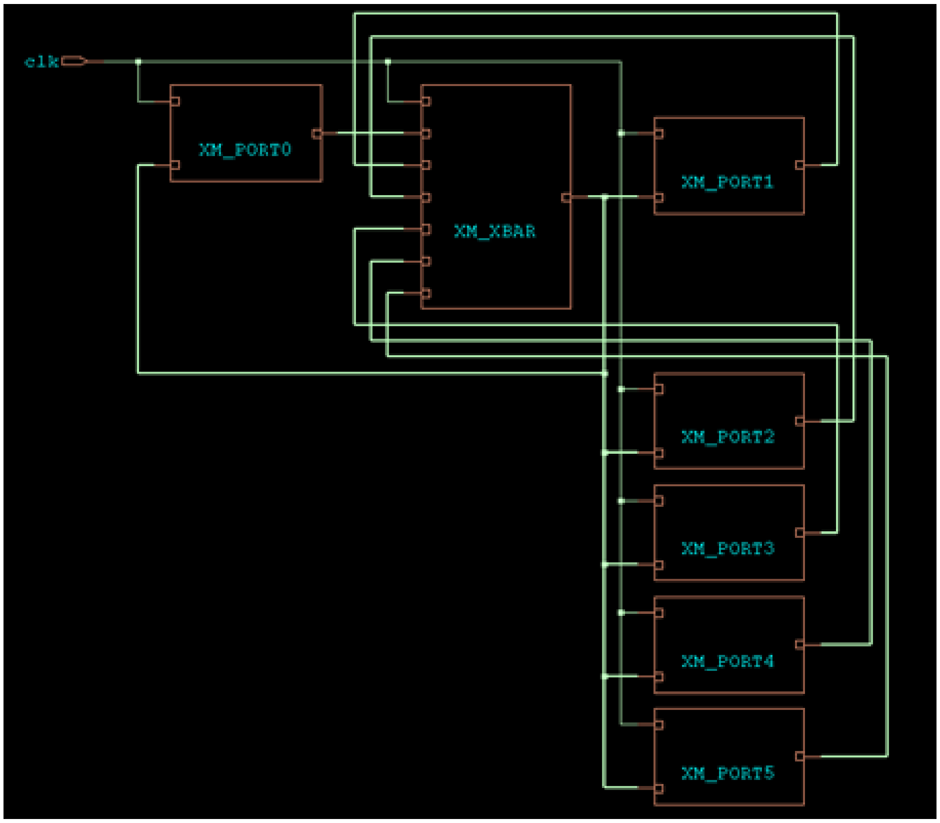
Through SAI, you can create an ideal mechanism for communicating between front-end and back-end designers.

Netlist Creation

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

Innovus Schematics Display

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



Automatic Netlist Creation Flow with SoC Architecture Info(SAI)

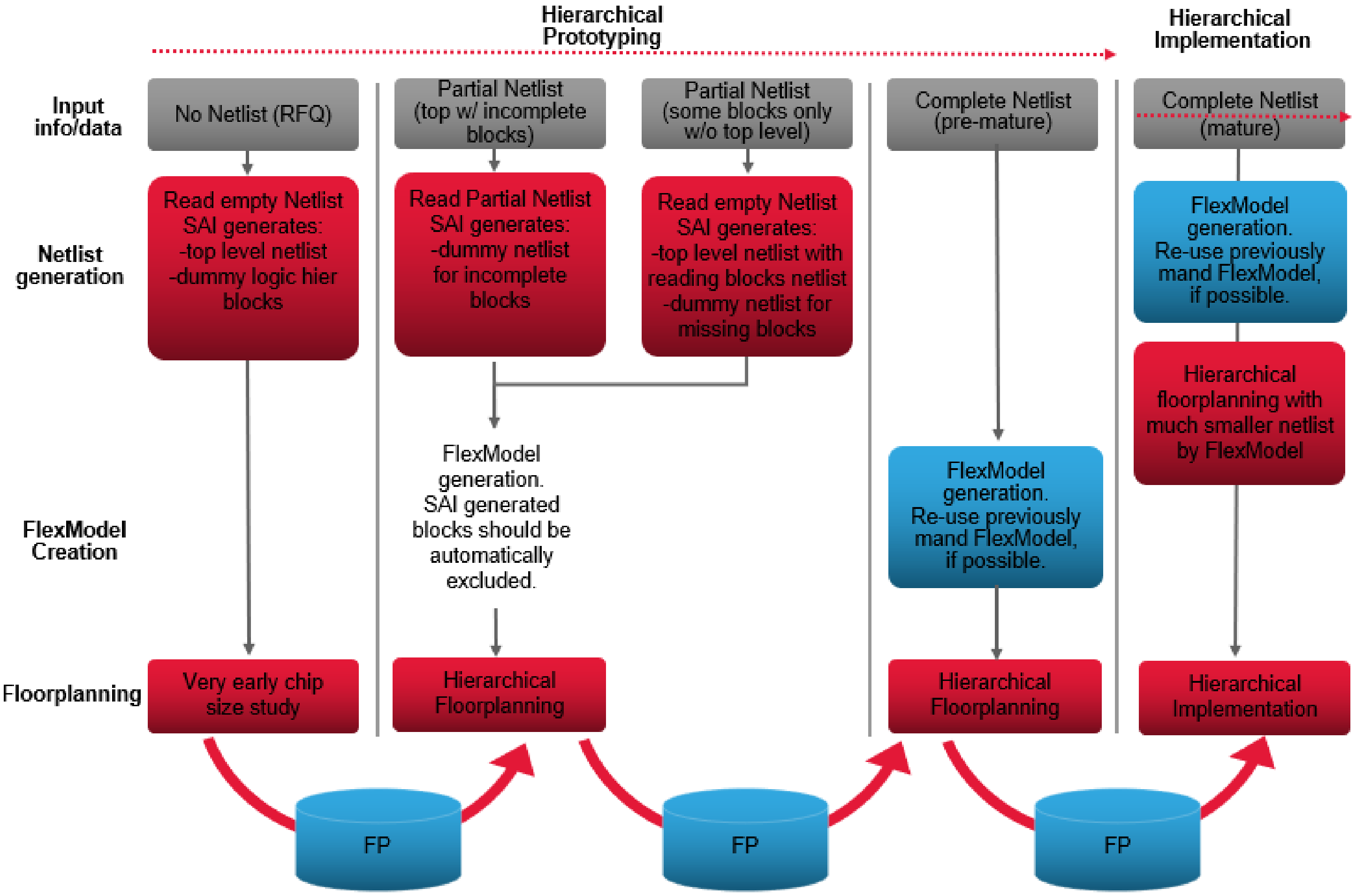
Based on specified SAI architecture information, Innovus creates a netlist to enable Innovus floorplanning as follows:

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

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

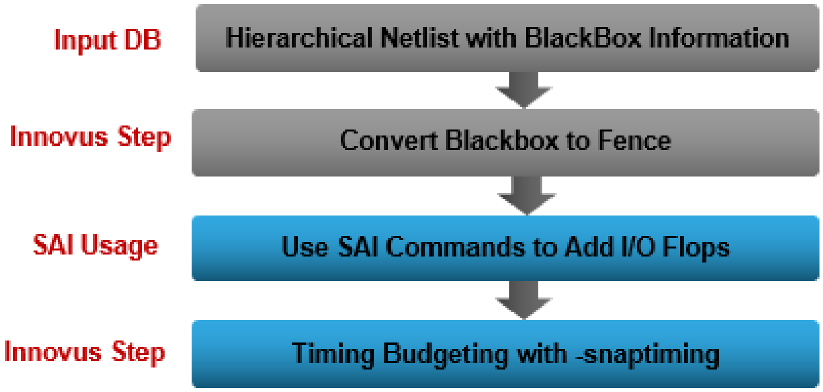
Possible Application of SAI/FlexModel Flows

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


Hierarchical Prototyping
Hierarchical Implementation
Input info/data
Netlist generation
FlexModel Creation
Floorplanning
No Netlist (RFQ)
Partial Netlist (top w/ incomplete blocks)
Partial Netlist (some blocks only w/o top level)
Complete Netlist (pre-mature)
Complete Netlist (mature)
Read empty Netlist SAI generates: -top level netlist -dummy logic hier blocks
Read Partial Netlist SAI generates: -dummy netlist for incomplete blocks
Read empty Netlist SAI generates: -top level netlist with reading blocks netlist -dummy netlist for missing blocks
FlexModel generation. Re-use previously mand FlexModel, if possible.
Hierarchical floorplanning with much smaller netlist by FlexModel
FlexModel generation. SAI generated blocks should be automatically excluded.
FlexModel generation. Re-use previously mand FlexModel, if possible.
Very early chip size study
Hierarchical Floorplanning
Hierarchical Floorplanning
Hierarchical Implementation
FP
FP
FP

SAI based Black-Box Time Budgeting Flow

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


Input DB
Hierarchical Netlist with BlackBox Information
Innovus Step
Convert Blackbox to Fence
SAI Usage
Use SAI Commands to Add I/O Flops
Innovus Step
Timing Budgeting with -snaptiming

FlexModel

For more information, refer to the FlexModel section.

Supporting Giga-Scale Designs in Planning stage

As the design size grows to 10s or 100s of millions of instances, serious capacity and run time limitations start occurring in various parts of the Hierarchical Planning stages. Addressing these limitations need advanced modeling techniques and capabilities in the flow. FlexModels provide a very light-weight abstraction for the partitions that can help reduce the design netlist up to 95%, leading to higher capacity and faster runtimes through the Planning stages. Besides using FlexModel technology, Active-logic Reduction Technology can be used to reduce timing graph.

Active-logic Reduction Technology

Active-logic Reduction Technology (ART) is a technique that is used to activate a certain portion of logic in a design and masking the other logic, while maintaining full physical design database in memory. In ART, an active logic view contains only the active portion of the logic. ART can be applied to any timing-related command, such as timing budgeting or timing optimization to reduce run time and memory usage. In timing operations, an active logic view contains only the set of timing paths exposed to the specific operation. When applied to timing optimization, active logic views enable cross-hierarchical optimization while preserving the full hierarchical view of the design after optimization is complete.

Top-level Timing Closure

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

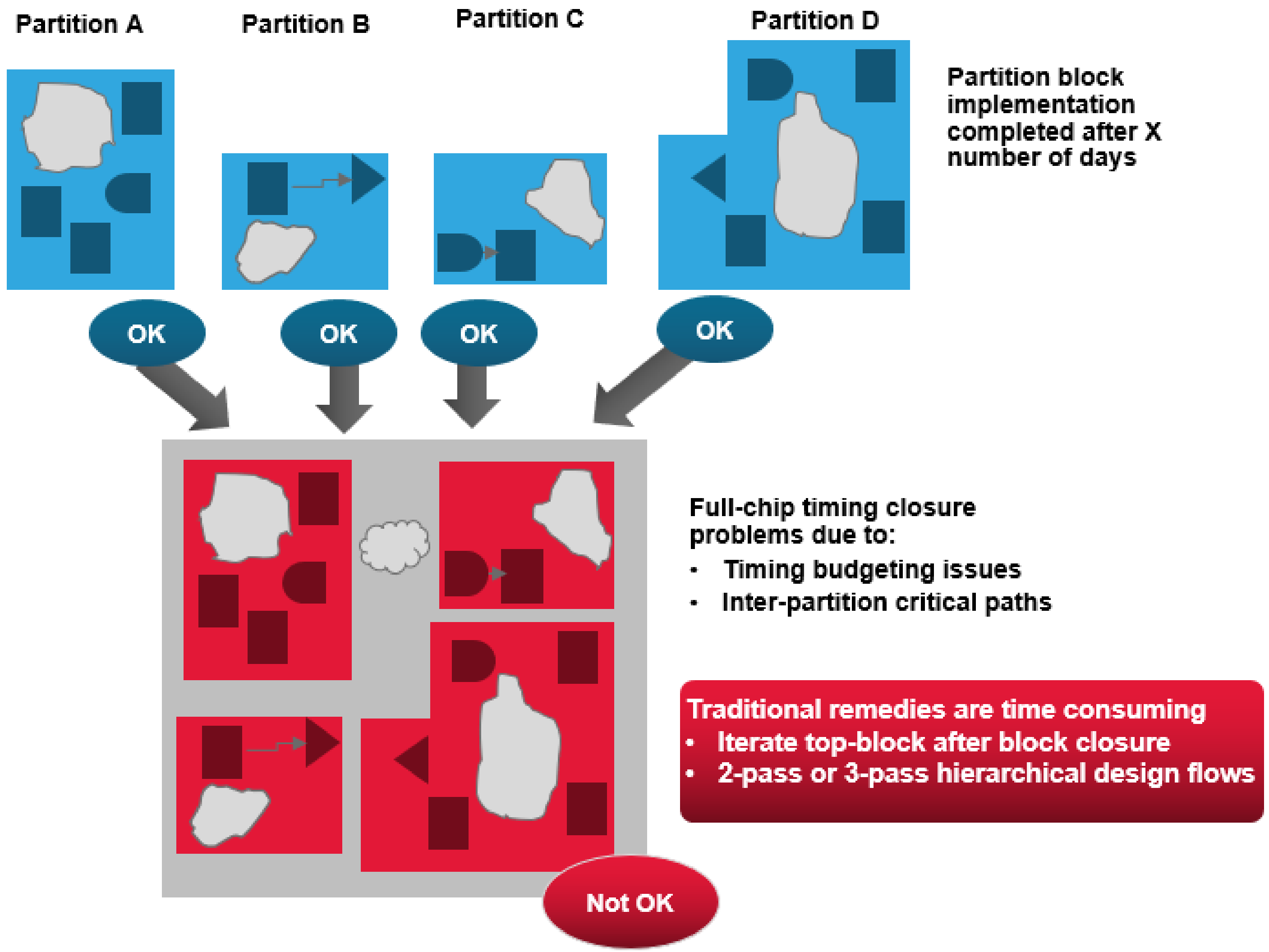
Using Interface Logic Models (ILM)

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

The advantages of using ILMs are as follows:

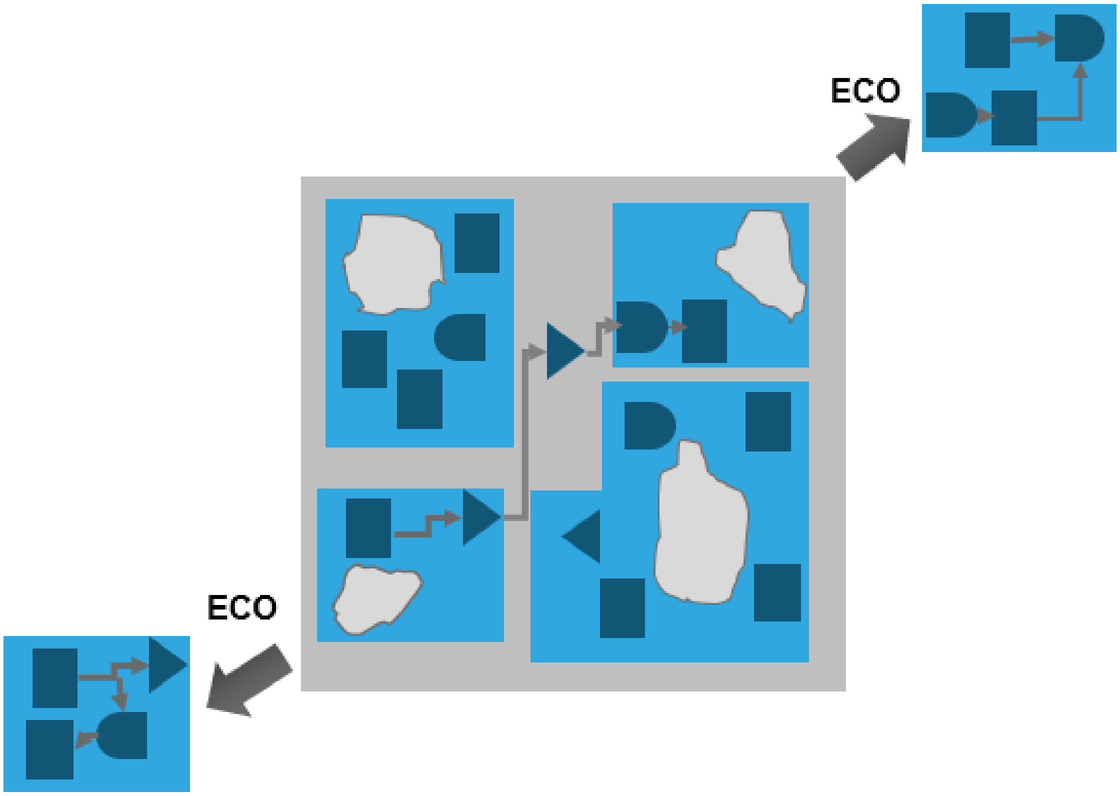
- More accurate analysis than a blackbox flow
- More SI aware than combined .lib or .cdb approach
- Can model clock generator inside block
- More accurate timing and SI reduces the number of design iterations to close timing and SI
- No need to characterize blocks
- Works on a actual design data
- Can be used in the initial prototyping stage for very big designs, when loading full design data is not feasible
- Allows you to modify only top-level data
- Fully preserves implemented partitions
- Uses the original constraint file for top-level analysis
- No abstraction for timing exceptions

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


Partition A
Partition B
Partition C
Partition D
Partition block implementation completed after X number of days
OK
OK
OK
OK
Full-chip timing closure problems due to:
• Timing budgeting issues
• Inter-partition critical paths
Traditional remedies are time consuming
• Iterate top-block after block closure
• 2-pass or 3-pass hierarchical design flows
Not OK

Using Flexible Interface Logic Models (FlexILM)

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



⚠ **Note:** For information on how to create a FlexILM model using the Integrated Hierarchical Database (iHDB) flow, see Top-level Timing Closure Methodologies for iHDB Flow.

⚠ **Note:** For the Integrated Hierarchical Database (iHDB) flow, chip assembly is done using `set_module_model` and `commit_module_model` commands.

Note: For information on how to bring together all the top-level and block-level netlists and routing information in the iHDB flow, see "Chip Assembly for iHDB Flow"" section of the Partitioning the Design chapter.

To know more about the Integrated Hierarchical Database Flow, see Integrated Hierarchical Database Flow.

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.

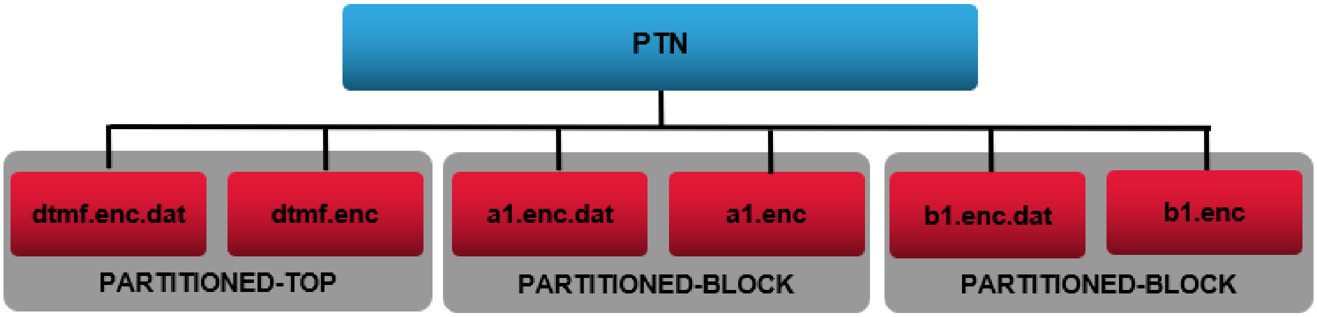
> ⚠ **Note:** For the Integrated Hierarchical Database (iHDB) flow, chip assembly is done using `set_module_model` and `commit_module_model` commands.
>
> **Note:** For information on how to bring together all the top-level and block-level netlists and routing information in the iHDB flow, see "Chip Assembly for iHDB Flow"" section of the Partitioning the Design chapter.
>
> To know more about the Integrated Hierarchical Database Flow, see Integrated Hierarchical Database Flow.

Before using the `assembleDesign` command, for each design, save the top-level and the block-level designs using the `saveDesign -def` command. Designs should be saved with a def file so that it can be used for assembling back the designs using the DEF merge capability for a fast turn around time.

Note: In case of Tempus ECO flow usage, the `checkPlace` command must be run after the `assembleDesign` command and all violations must be fixed in order get the best possible QOR with the Tempus physical ECO feature.

As an example, consider a design called dtmf that has two partitions: a1 and b1. After running the `partition` command, the partition directories are saved under the PTN directory. You would, therefore, implement the following:

- The design files are a1.enc and a1.enc.dat for a1 block and b1.enc and b1.enc.dat for b1 block. The following figure shows the directory structure:



Chip assembly also supports the DEF merge capability. You can use the direct block DEF transform-and-merge approach during `assembleDesign`. The following are the advantages of the DEF merge capability:

- No partition LEF files are required
- Lesser peak memory requirement

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

- Concatenates the Verilog netlist files from the partitions back to the top level
  Note: The partition netlists and top-level netlist are changed from the time the save partition step was performed.
- Merges the design data with the original top design level. By default, data from DEF files is used. However, you can use the `-fe` parameter to specify that Innovus data should be used. You can also use data in the OpenAccess database format.
- Rows at top-level design will be cut, and the rows at block-level design will be brought back
- Preserves scan chain information at partition block-level design, therefore minimizing the floorplan data loss during partition and assemble design cycle. The start and stop scan chain points at partition block I/O pins are adjusted back to instances that connect to scan chain points. Top-level scan chains are not connected to block-level scan chains.

Note: You must run the `assembleDesign` command from the directory that contains the full chip-level floorplan for the top-down hierarchical flow.

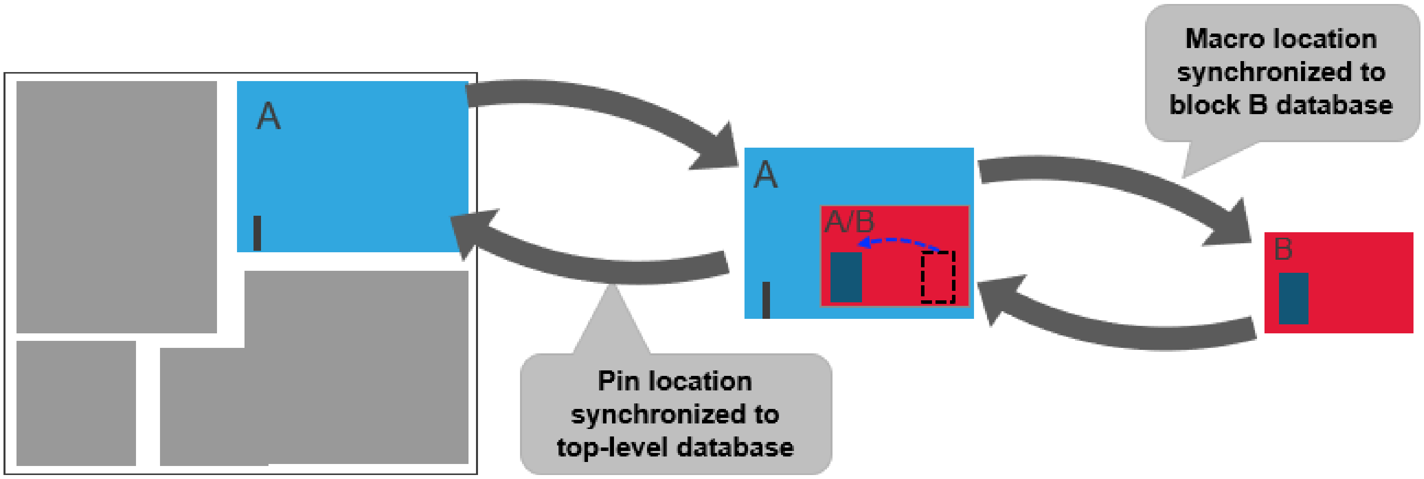
Integrated Hierarchical Database

Overview

The Integrated Hierarchical Database (iHDB) flow capability enables you to easily navigate through the hierarchical database and selectively open specific blocks in a specific block model to save memory. This feature helps to traverse through multiple levels of hierarchy easily. It automatically synchronizes the database changes from top-level to block level or vice versa.

With just a GUI click you can open a block-level database in an independent session. You can use this session to:

- Modify the database. For example, change the floorplan, update pin locations, change power routing, or run the `place_opt_design` command.
- Save the changes and create block models
- Synchronize the changes back to the top-level design such that it includes the updated changes.
  Note: You can also generate the ECO diff report when changes are synchronized from the top-level design to a block design.


Macro location synchronized to block B database
A
A
A/B
B
Pin location synchronized to top-level database

The Integrated Hierarchical Database flow provides:

- Provides a bird's-eye view through hierarchies. From the top-level design, you can view lower level hierarchies.
- Enables you to do DRC/Timing debugging across the boundary and multi-levels of logical hierarchy.

- Supports different models (such as ILM, FlexILM, LEF, etm, pnr DB, customized model) at the block-level design.
- Enables you to set or configure block models at the top-level design. It supports mix-and-match model types and selective model view/tags. Additionally, it provides flexibility in saving and restoring the hierarchical database configuration.
- Supports global block database repository management as well as version control (SVN). It also includes the utility to pack the block database repository and re-link libraries for portability.

Note: The Integrated Hierarchical Database (iHDB) flow works with the "base" Innovus Digital Implementation System license (the base license is the license used to invoke the software). However, you do require an additional Hierarchical license to use the `commit_module_model` command for the pnr and FlexILM model types.

Integrated Hierarchical Database Flow: Examples

Partitioning and Creating Initial iHDB

The following is a sample of a Tcl script for partitioning and creating the initial Integrated Hierarchical Database:

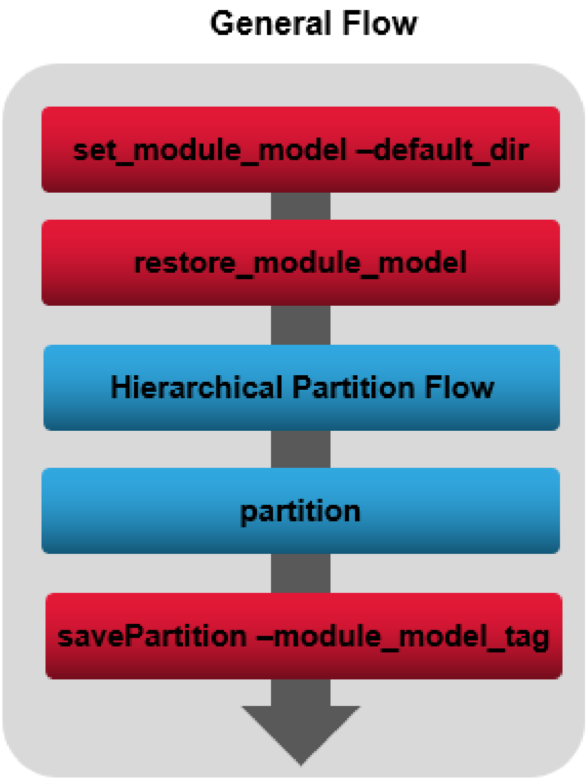
```
# Define module models repository
set_module_model -default_dir /myproject/DATA
```

Note: If `set_module_model -default_dir` is not specified, the Innovus system does not set any directory by default. Later on, when you try to run the `restore_module_model` command, it displays an error message that the `-default_dir` parameter is not set.

```
restore_module_model myTopCell -tag myTagName
```

```
# Place design
# Feedthrough and pin assignment
deriveTimingBudget -cycleRatio
partition
savePartition -module_model_tag mytagName
```

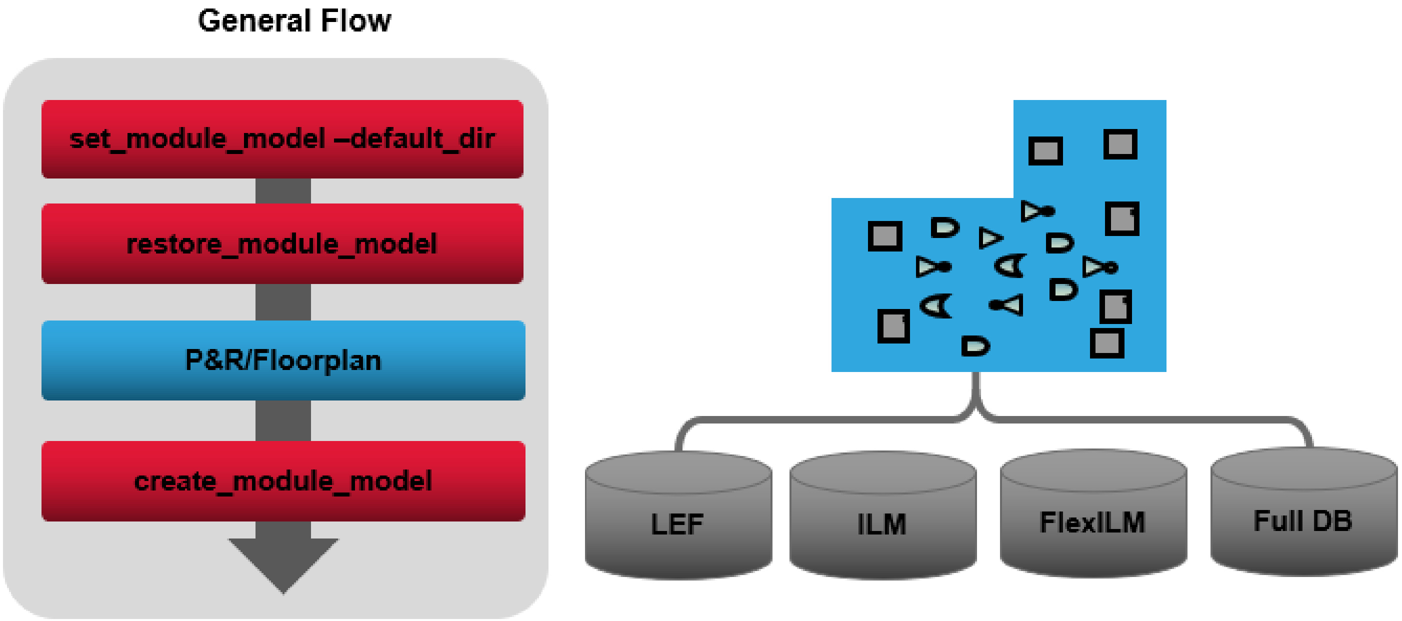
```
# It is mandatory to specify the tag name here.
```


General Flow
set_module_model –default_dir
restore_module_model
Hierarchical Partition Flow
partition
savePartition –module_model_tag

Block Model Generation

The following is a sample of a Tcl script for block model generation in the Integrated Hierarchical Database (iHDB) flow:

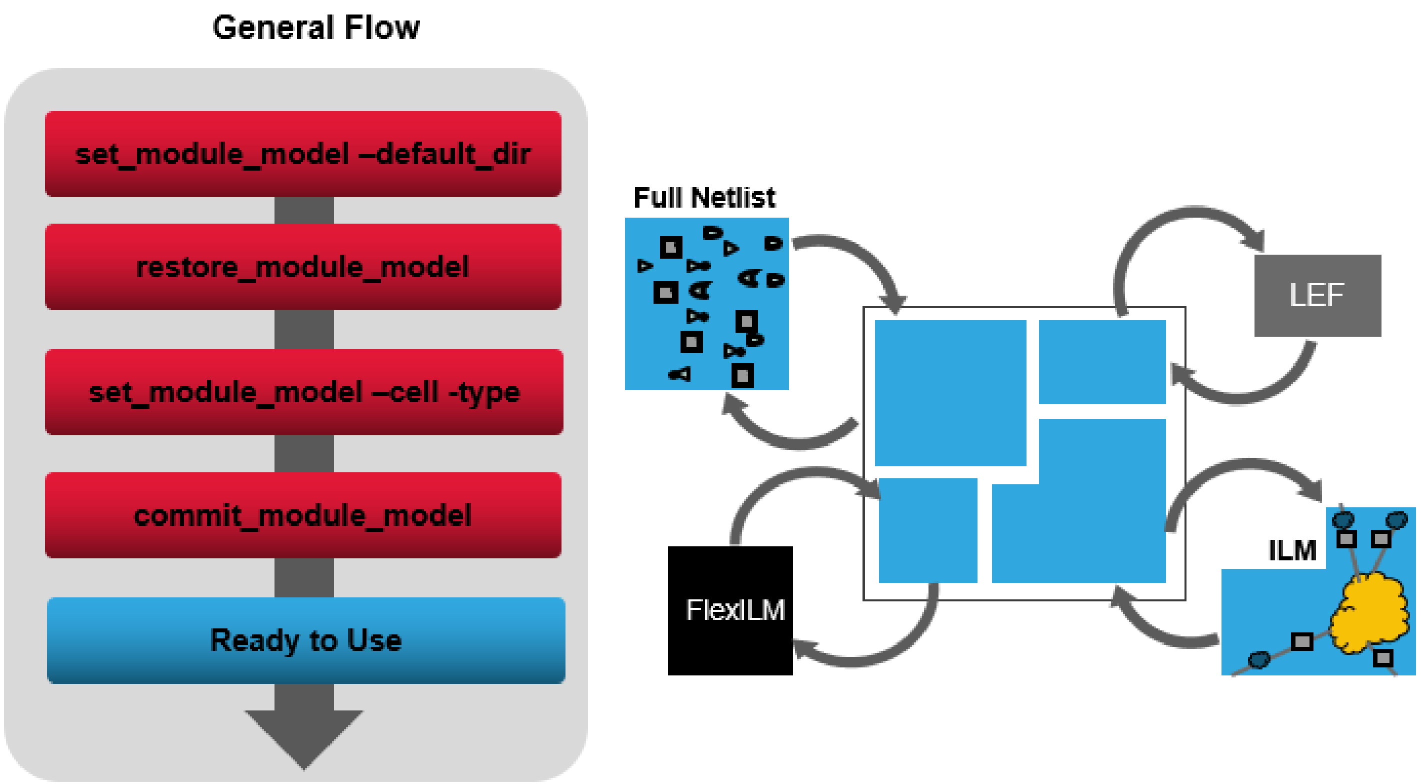
```
# Define module models repository

set_module_model -default_dir /myproject/DATA

set_module_model -default_options {stripes_pins -pg_pin_layers {M2 M3 M4}} -type lef

restore_module_model blockCellName -tag init

# Place and optimize the design

# Insert clock tree

# Route the design

create_module_model -tag preCTS; # This will create pnr and lef models.

create_module_model -tag preCTS -type ilm
```



Top-level Instantiation

The following is a sample of a Tcl script for top-level instantiation in the Integrated Hierarchical Database (iHDB) flow:

```
set_module_model -default_dir /myproject/DATA

restore_module_model myTopDesign -tag init

set_module_model -cells * -type lef
# Latest tag will be used for all cells since -tag option is not specified

set_module_model -cells ptn_wrapper -tag preCTS -type flexIlm
set_module_model -cells ram_128x16_test -tag latest -type pnr

commit_module_model
set_module_model -cells tdsp_core -type ilm -update
commit_module_model

report_module_model
```



Integrated Hierarchical Database Repository Management

The Integrated Hierarchical Database flow supports global database repository management as well as simple version control. To manage the database:

- Use the `set_module_model -default_dir` command to specify the default location where the module models will be loaded or saved.
- Create or update the module models using the following commands:
  - `savePartition -module_model_tag`
    With the `-module_model_tag` parameter, the `savePartition` command saves top and block level pnr module models into the Stylus global hierarchical database repository. These block-level and top-level pnr models can be used for implementing the block-level and top-level designs, respectively.
  - `create_module_model`
    This command creates and saves the specified module models into the global database repository.
- Restore top cell (block db) database in a consistent way using the `restore_module_model` command.

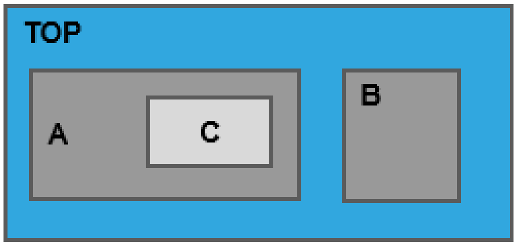
The logical structure of the global database repository has the default directory `(set_module_model -default_dir`) on the top. All the cells or module models are configured under the default directory. Each cell name is associated with a user-defined tag name that can be used as simple version control. These tags names can be a design stage/status name such as ‘prects’, ‘cts’, etc. It can be any name except 'latest'.

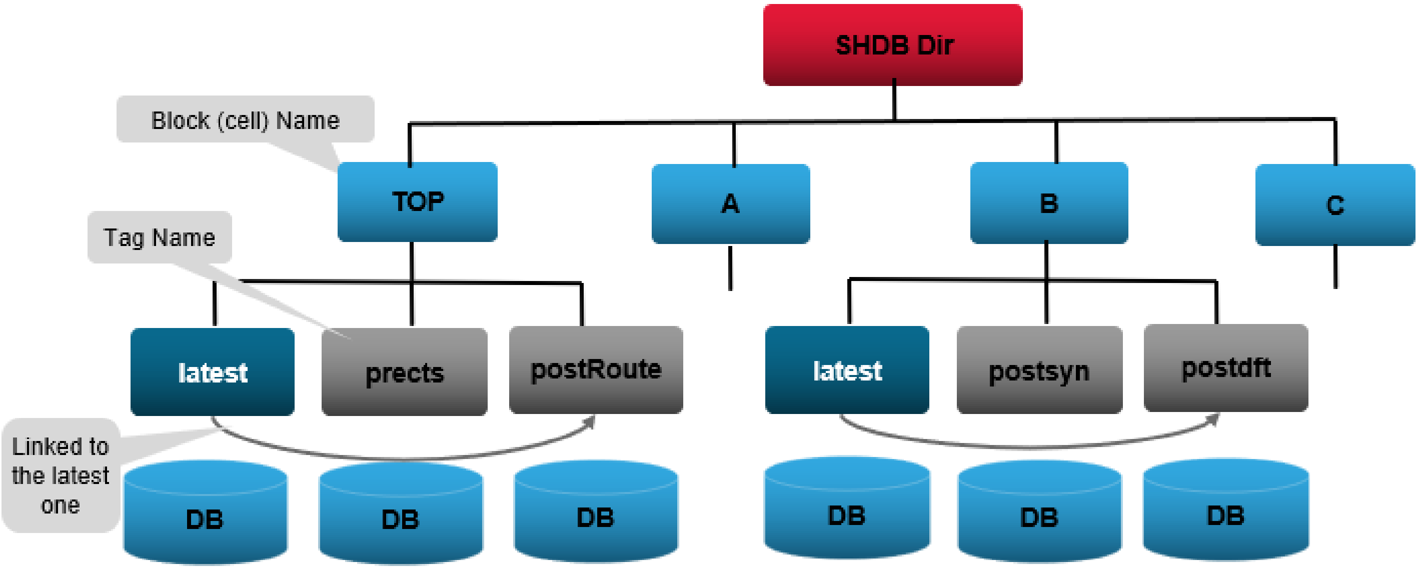
Note: The 'latest' tag will be used if this a tag name is not specified. Latest tag will point to the latest saved tag directory.

Example:

The following is the example of the disk directory structure:

```
set_module_model -default_dir /myproject/DATA
```





Note: Cadence recommends you to specify all the hierarchical DB default settings in an Innovus environment file (.encrc) at the current run directory so that you do not need to specify them in two run scripts.

For example:

```
setMultiCpuUsage -localCpu 4
```

```
set_module_model -default_dir /myproject/DATA
```

Creating a Module Model

To create a module model:, do the following:

1. Use the `set_module_model -default_dir` command to specify the default central directory name where models data will be saved/loaded.
   For example, `set_module_model -default_dir /myproject/DATA`
2. Use the `restore_module_model` command to restore a block-level design from a default module model repository.
   For example, the following command restores the myTopCell database and uses the "latest" tag.
   ```
   restore_module_model myTopCell -tag latest
   ```
3. Implement the block-level design.
4. Use the `create_module_model` command to create a module model and save different types of models. It honors the `set_module_model -default_options` command settings.

Setting and Commit a Module Model

A module model type can be set or configured at a top-level design using the `set_module_model` command. It supports:

- Mix and match models types
- Selective view or tag information
- Wildcard for cell names where Innovus will automatically set all models at top-level based on their existence in the global DB repository.

Once the specified type of module models are specified, the `commit_module_model` command should be run to load the specified models.

Example:

```
# Latest tag will be used for all cells since -tag option is not specified

set_module_model -cells * -type lef

set_module_model -cells ptn_wrapper -tag preCTS -type flexIlm
set_module_model -cells ram_128x16_test -tag latest -type pnr
```

```
commit_module_model
set_module_model -cells tdsp_core -type ilm -update
commit_module_model
```

Updating a Module Model

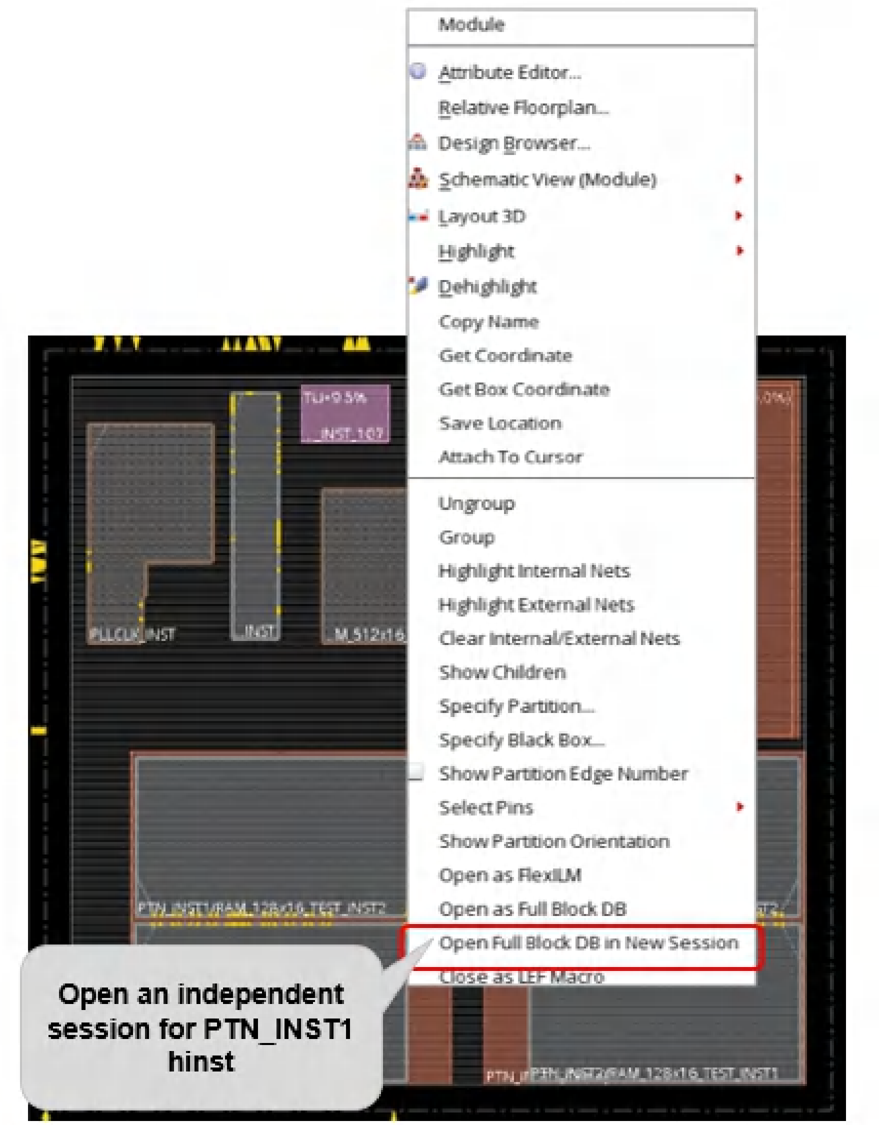
The Integrated Hierarchical Database provides the ability to reflect the floorplan changes from the top to the block database and vise-versa using the `update_module_model` command.

- Floorplan changes made at the top-level can be updated to a block-design in an independent session.
- Changes made at block-level design can be updated to top-level design once exiting an independent session to go back to top-level design. By default, an updated LEF of the same tag will be brought back to top-level design. However, the LEF model will not be updated if you do not re-generate the updated models at block-level design using the `create_module_model` command.

The `update_module_model` command can invoke an Innovus independent session with in interactive or batch mode.

Interactive Mode

For interactive mode, move the cursor on top of a model and click on '*Open as Full Block DB in New Session*' option from the context pop-up menu to open an independent session in GUI.



Open an independent session for PTN_INST1 hinst

Batch Mode

For batch mode, the `-plugin_tcl` parameter should be specified with the list of TCL commands that will be executed at block level design. Once the block design has been re-implemented with the new floorplan change, updated models should be re-generated.

Example: Open an Innovus independent session with the ptn_wrapper block-level design (pnr) to do ECO route and generate update module models (pnr/LEF/ILM/FlexILM):

```
update_module_model -cell ptn_wrapper -plugin_tcl {
ecoRoute
create_module_model
create_module_model -type ilm
create_module_model -type flexilm
}
```

Importing/Converting an Existing DB(*.enc.dat) to a Hierarchical Module Model

To convert an existing Innovus database into a hierarchical module model, do the following:

1. Specify the default central directory name where the module models data will be saved/loaded. For example:

   ```
   set_module_model -default_dir /myproject/DATA
   ```

2. Restore the existing database from the default module model repository.

   ```
   restoreDesign ./design/dtmf_recvr_core.enc.dat dtmf_recvr_core
   ```

3. Create a module model. By default, it will create pnr and LEF models.

   ```
   create_module_model -tag start
   ```

Validating Budgeting Results

When you generate the timing budgets for partitions using the `deriveTimingBudget` command, the `checkPartitionSdc` command can be used to validate the results. The validation ensures that the timing data of the partitioned block is the same as that of the chip. This saves the ECO iteration for the chips and partitioned blocks, and thereby reduces the overall design cycle time. You can use `checkPartitionSdc -module_model_tag` command to specify the module model tag for the cell on which the SDC comparison is to be done.

For example, the following commands validates the timing budgeting results for the specified module model:

```
set_module_model -default_dir /myproject/DATA
checkPartitionSdc -chip scripts/chip.tcl -cells $block -module_model_tag init -no_top
```

Validating an ILM Model

You can use the `check_module_model` command to check and validate an ILM model to see if current reduction ratio will cause:

- Missing constraints at the hinst terminal
- Slack difference

The `check_module_model` command compares/checks the target timing result with the reference timing result and generates a summary and detailed report.

Note: If a specified cell has more than one instantiated modules, all hinst terminals are checked and reported.

Usage Model:

- Generate a reference ILM model with the `createInterfaceLogic -keepAll` option and a target ILM model with user-specified settings.
  Note: The reference ILM model should be specified with the same user-specified settings as the target ILM model.
- Restore the top-level design and set/commit target ILM model(s).
- Time the design.
- Call `check_module_model`

Example:

```
restore_module_model  top -tag init

set_module_model -cell A -type ilm -tag pre_cts
set_module_model -cell B -type ilm -tag pre_cts
commit_module_model -mmmc_file full_chip_view_definition.tcl

timeDesign -preCTS
check_module_model -cell A -ref_tag ilm_reference -type ilm -out_file
check_module_model.rpt
```

Working with Nested ILMs in the iHDB Flow

ILM models (donut models) of all levels of hierarchy can be specified at top-level design for top-level timing closure. The following is a sample of a Tcl script for specifying nested ILMs in the iHDB flow:

1. Specify the default central directory name where the module models data will be saved/loaded. For example:

   ```
   set_module_model -default_dir ./DATA2
   ```

2. Restore the top-level design:

   ```
   restore_module_model dtmf_recvr_core -tag init
   ```

3. Specify all ILMs included nested ones:

   ```
   set_module_model -cell tdsp_core -tag my_preCts -type ilm
   set_module_model -cell ptn_wrapper -tag my_preCts -type ilm
   set_module_model -cell ram_128x16_test -tag my_preCts -type ilm
   ```

4. Commit flexILM

   ```
   commit_module_model
   ```

5. Execute the pre-CTS flow with both placement and pre-CTS optimization.

   ```
   place_opt_design
   ```

6. Create a module model.

   ```
   create_module_model -tag top_pod
   ```

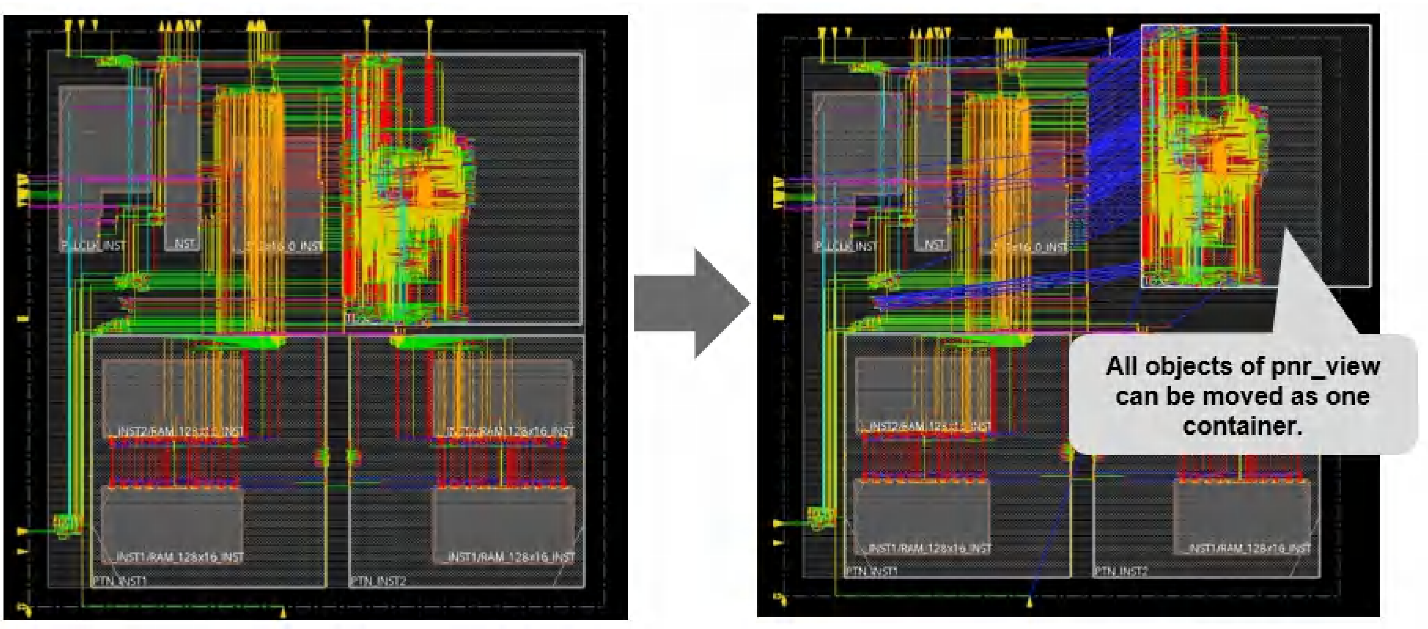
Controlling the Physical View Visibility of ILMs

While configuring ILM type of module models (`set_module_model -type ilm`), you can use the `pnr_view` add_ons model to see the complete physical view of an ILM including routing, cells, etc.

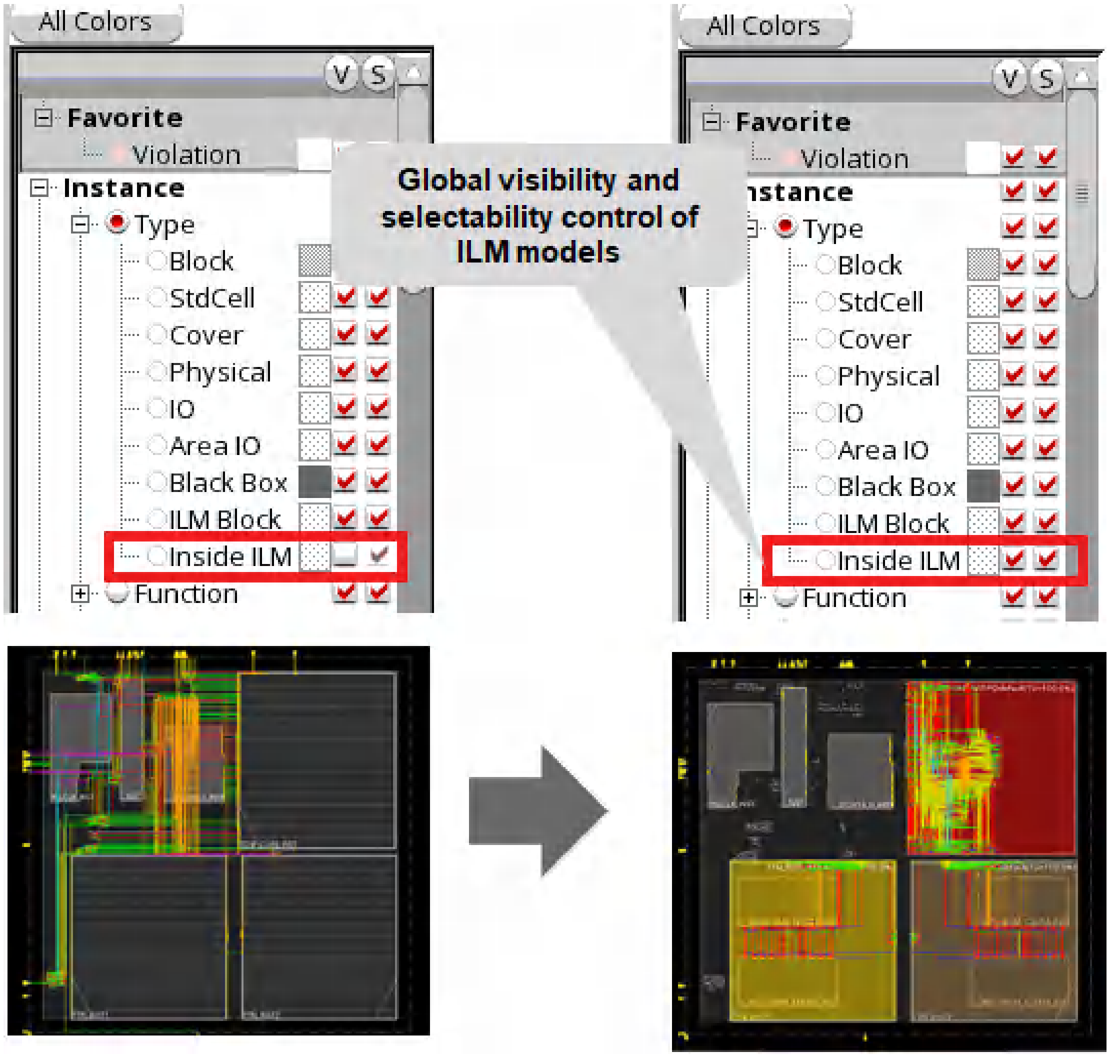
For example:

```
set_module_model -default_dir ./DATA -cell tdsp_core -type ilm -tag my_tag -add_ons
{pnr_view}
```

Note: A pnr_view add_on model is an independent container. When you move a pnr_view, all object that belong to it will also move.

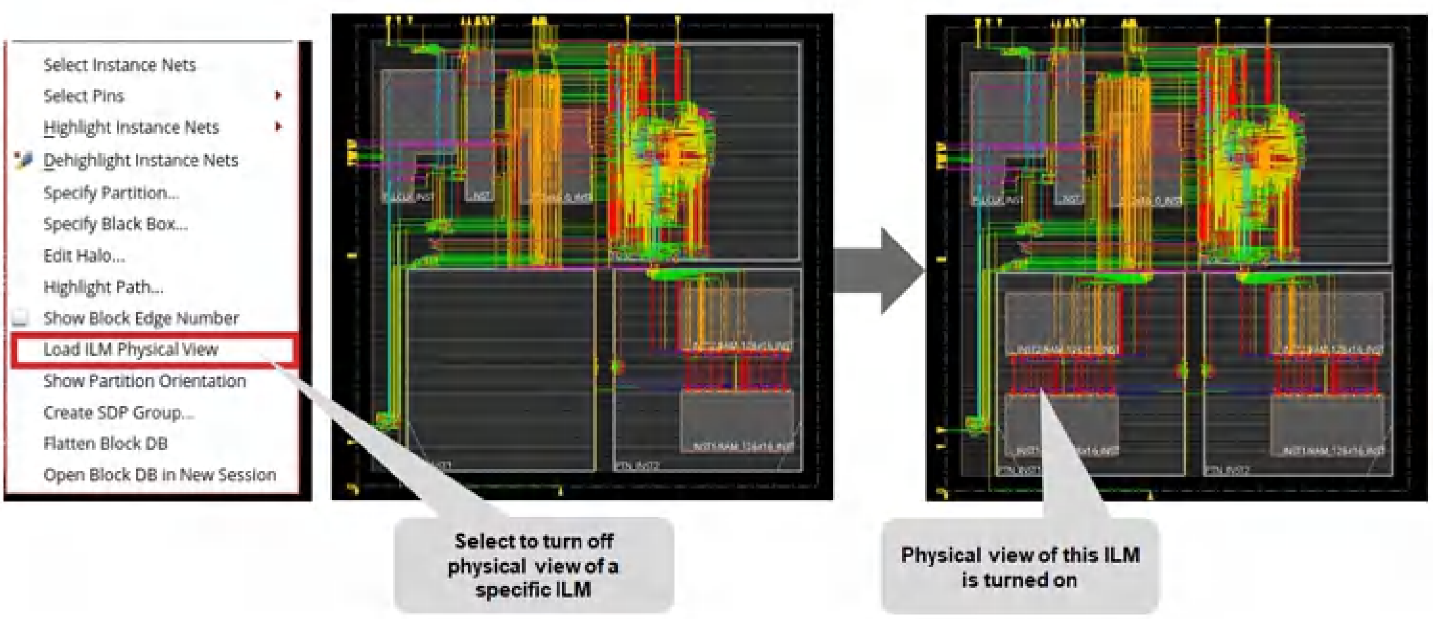


The *Inside ILM* checkbox in the *All Colors* panel is the global control for all ILM displays. It controls the visibility and selectability of ILM models. When this option is enabled, the details of the ILM are displayed but if this option is disabled, an ILM block is displayed instead. While configuring ILM type of module models (`set_module_model -type ilm`), you can use the `pnr_view` add_ons model to see the complete physical view of an ILM including routing, cells, etc.



Global visibility and selectability control of ILM models

You can turn on or off the visibility and selectability of a specific ILM model by using the *Load ILM Physical View* from the right-click context menu.



Select to turn off physical view of a specific ILM

Physical view of this ILM is turned on

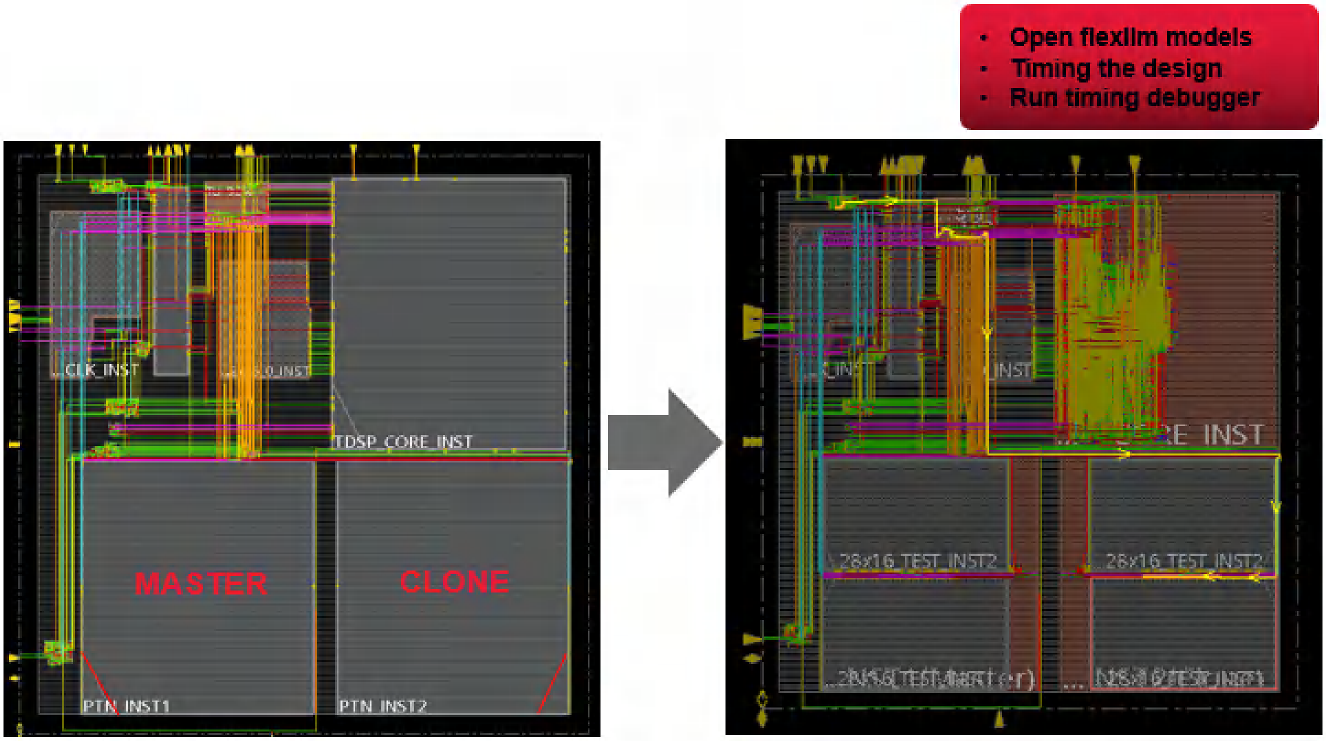
The following table summarizes the updated ILM view settings:

Inside ILM **checkbox in** ***Color Panel***	***Load ILM Physical View*** **checkbox in a specific ILM context menu**	**Show/Hide Object in ILM block**
On	On	Pnr physical view of the block is displayed. **Note:** If the pnr physical view is not available in memory then it is dynamically loaded.
On	Off	ILM global view is displayed.
Off	On	Hidden
Off	Off	Hidden

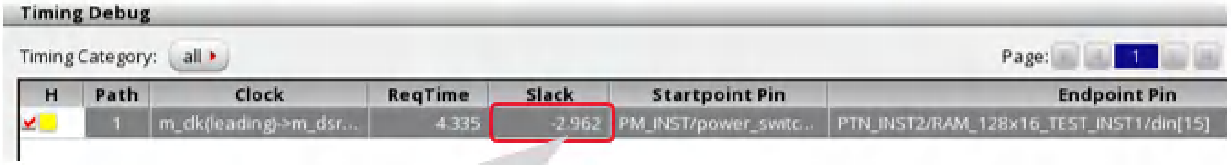
Example of Using Hierarchical DB for Timing Debugging and Floorplan Editing

1. Consider a design where timing is not met. The root cause of timing violation may be due to bad pin location.

```
restore_module_model -tag init
set_module_model -cell * -tag post_route -type flexilm
commit_module_model -mmmc_file chip.enc.dat/viewdefinition.tcl
```



- Open flexilm models
- Timing the design
- Run timing debugger



Timing Debug

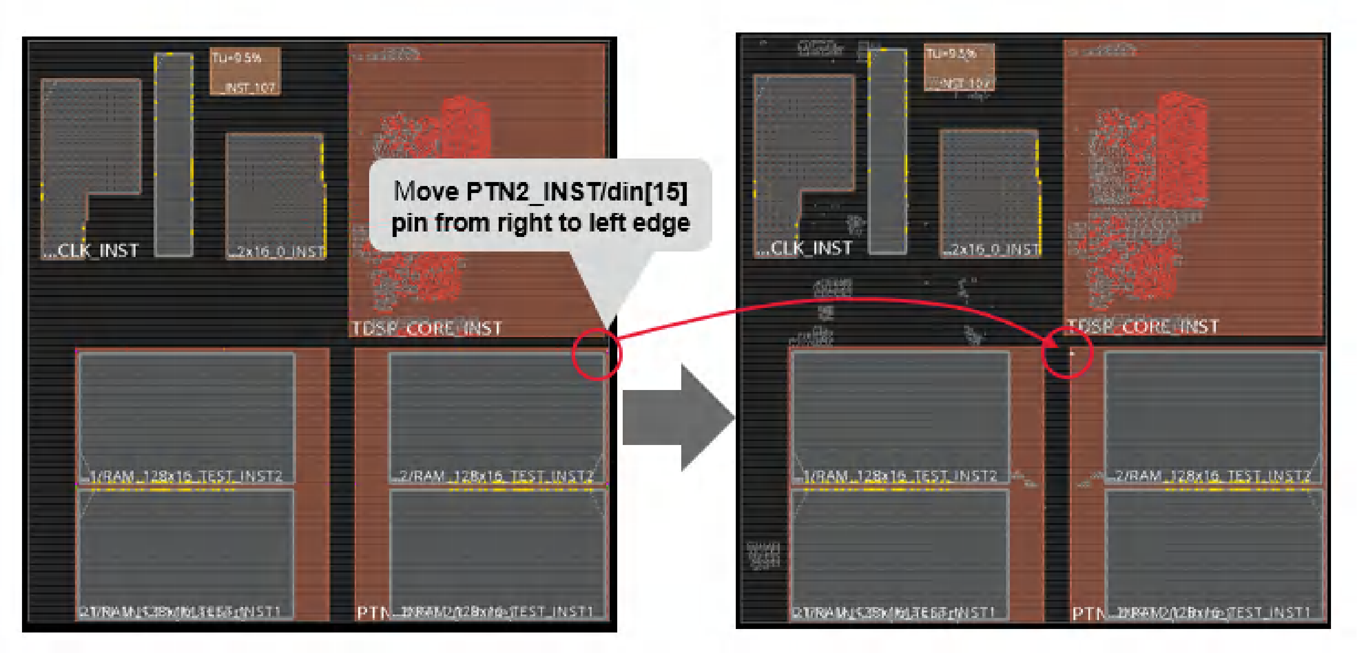
Timing Category: all

Page: 1

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
✓🟨	1	m_clk(leading)->m_dsr...	4.335	-2.962	PM_INST/power_switc...	PTN_INST2/RAM_128x16_TEST_INST1/din[15]

Timing slack of critical path through partition pin PTN_INST2/din[15] is -2.962

2. Adjust the pin location of a block at the top-level database.

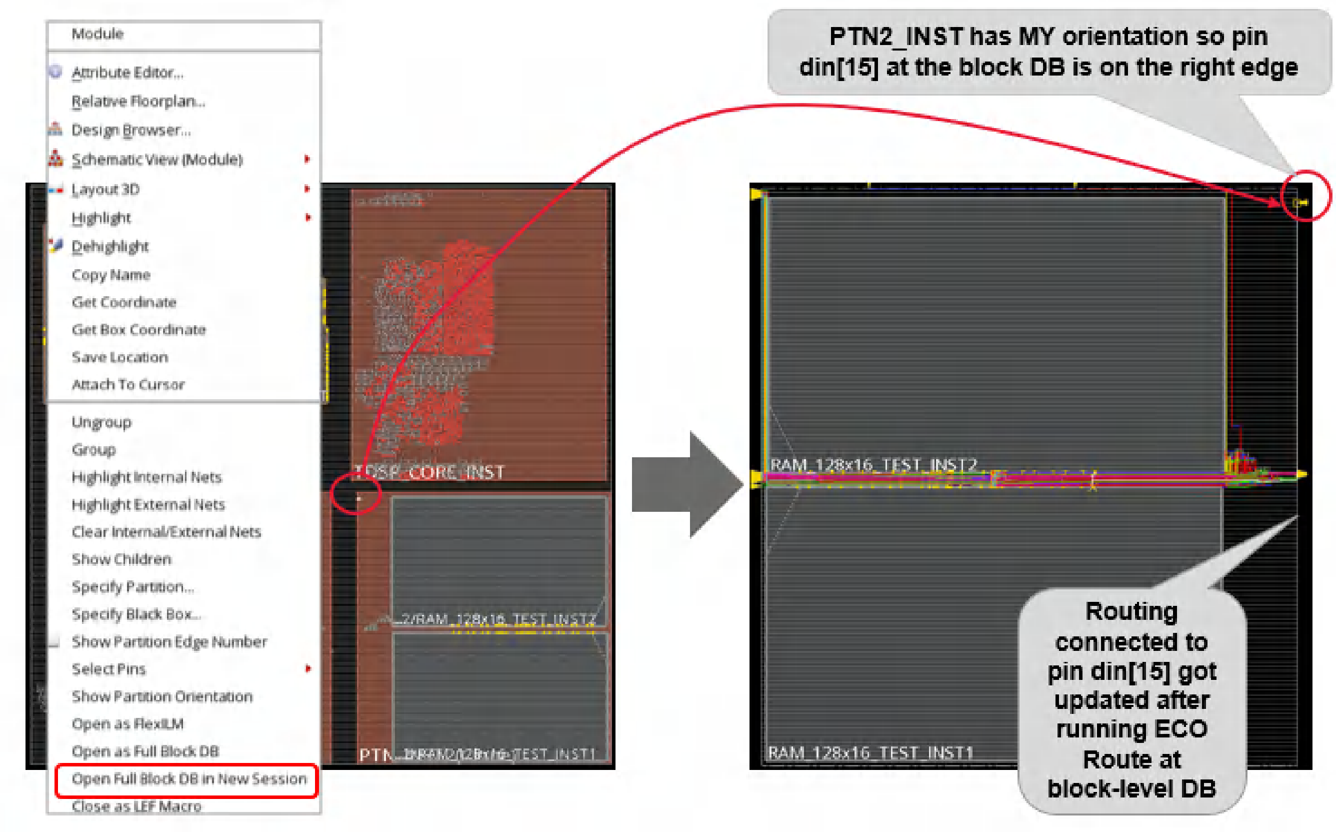


Move PTN2_INST/din[15] pin from right to left edge

...CLK_INST
...2x16_0_INST
TDSP_CORE_INST
...1/RAM_128x16_TEST_INST2
...2/RAM_128x16_TEST_INST2
...CLK_INST
...2x16_0_INST
TDSP_CORE_INST
...1/RAM_128x16_TEST_INST2
...2/RAM_128x16_TEST_INST2

3. Open the block level database of the modified block module in an independent session to update the block design with pin location change using `update_module_model` command. Modified pin location is automatically updated at block-level design.
   For example, the following commands can be used at block-level design to run ECO route based on the new pin location and re-generate all models(LEF/ILM/flexIlm/Full DB):

```
update_module_model -cell ptn_wrapper -plugin_tcl {

ecoRoute

create_module_model

create_module_model -type ilm

create_module_model -type flexilm

}
```



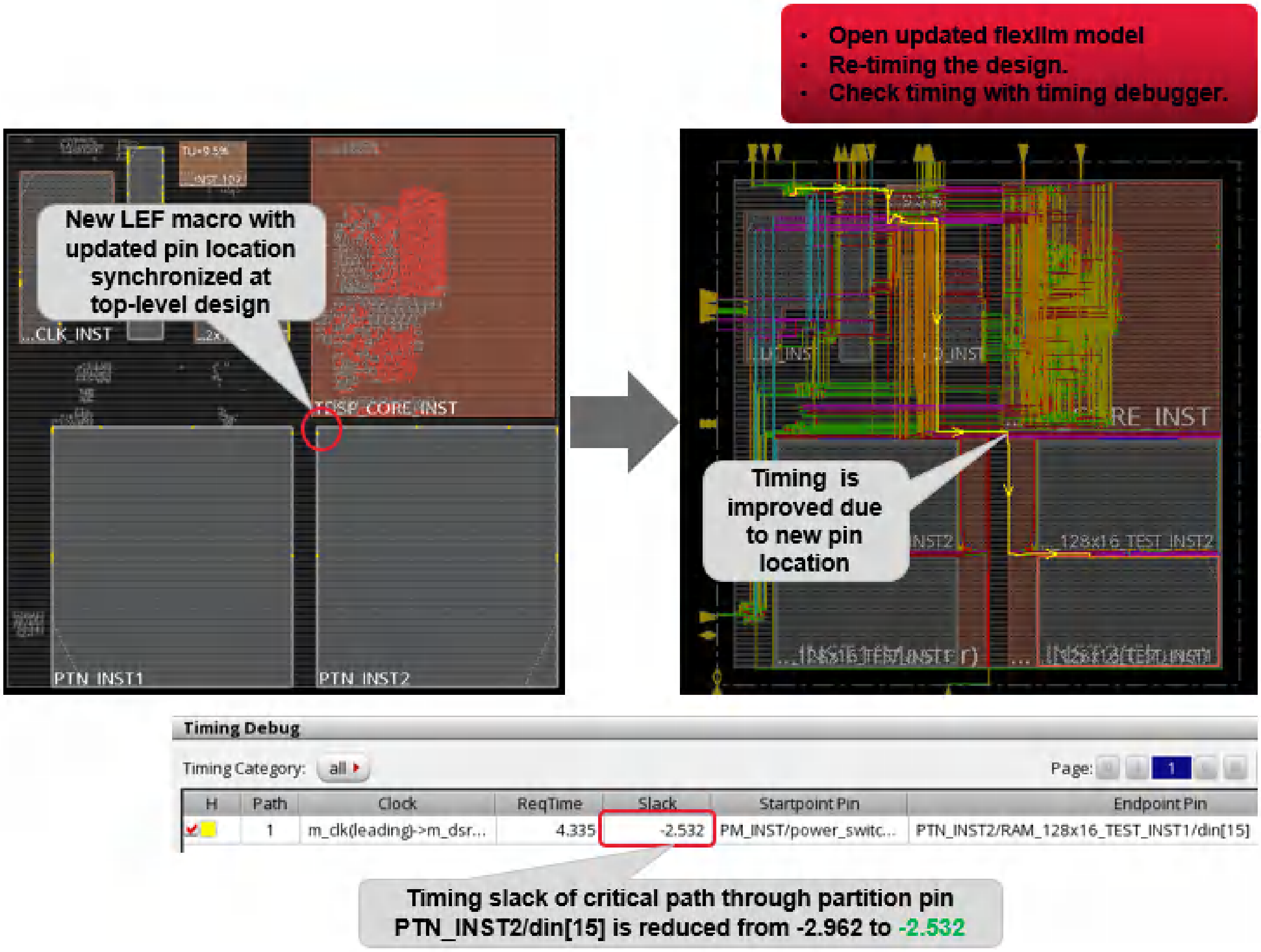
PTN2_INST has MY orientation so pin din[15] at the block DB is on the right edge

Routing connected to pin din[15] got updated after running ECO Route at block-level DB

4. Once the independent session is exits, the main Innovus session will be resumed and the new re-generated model of the same is automatically updated at the top-level design.

```
set_module_model -cell ptn_wrapper -type lef
commit_module_model
earlyGlobalRoute
set_module_model -cell * -tag latest -type flexilm
commit_module_model -mmmc_file chip.enc.dat/viewdefinition.tcl
```

You can check the timing again and see the difference.



- Open updated flexilm model
- Re-timing the design.
- Check timing with timing debugger.

New LEF macro with updated pin location synchronized at top-level design

Timing is improved due to new pin location

Timing Debug

Timing Category: all

Page: 1

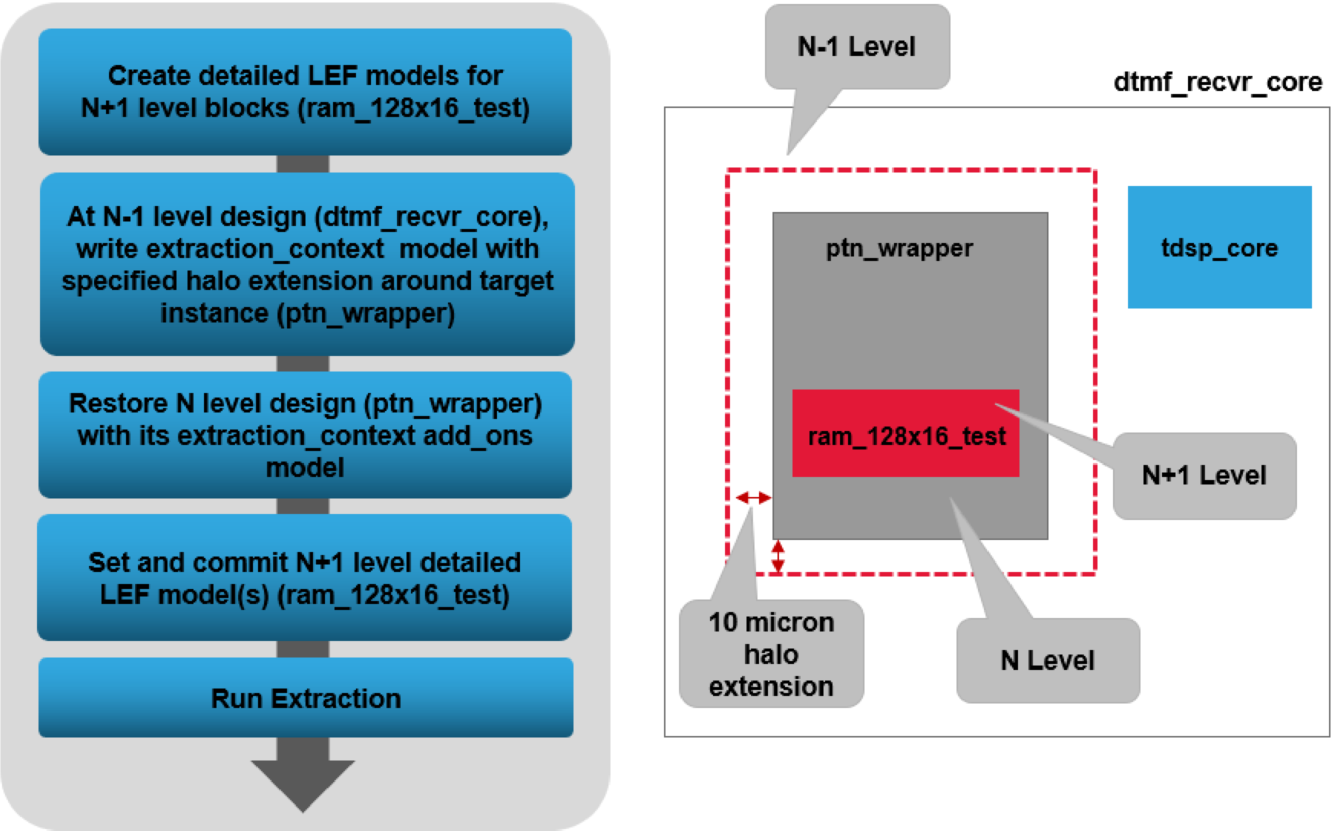
H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
✓	1	m_clk(leading)->m_dsr...	4.335	-2.532	PM_INST/power_switc...	PTN_INST2/RAM_128x16_TEST_INST1/din[15]

Timing slack of critical path through partition pin PTN_INST2/din[15] is reduced from -2.962 to -2.532

Hierarchical Extraction

The iHDB flow supports hierarchical extraction where the tool can generate parasitic information of a target instance that includes the routing surrounding the target block at its parent level design and detailed extracted LEF(s) of its lower-level hierarchy block(s).

Following is the use model for the high level hierarchical extraction flow for the target instance ptn_wrapper (N level) that is instantiated at dtmf_revcr_core design (N-1 level) and it has ram_128X1_test as the lower level hierarchy (N+1 level):


Create detailed LEF models for N+1 level blocks (ram_128x16_test)
At N-1 level design (dtmf_recvr_core), write extraction_context model with specified halo extension around target instance (ptn_wrapper)
Restore N level design (ptn_wrapper) with its extraction_context add_ons model
Set and commit N+1 level detailed LEF model(s) (ram_128x16_test)
Run Extraction
N-1 Level
dtmf_recvr_core
ptn_wrapper
tdsp_core
ram_128x16_test
N+1 Level
10 micron halo extension
N Level

Session 1: Lower-level (N+1)

- Create detailed LEF models for N+1 level blocks (ram_128x16_test).

```
restore_module_model  ram_128x16_test  -tag init

set_module_model -default_options {-cutObsToExposeRouting 20} -type lef

create_module_model -type lef -tag my_extract_v1
```

Session 2: Upper-level (N-1)

- At N-1 level design (dtmf_recvr_core), create extraction_context _model that has routing information within specified halo extension around target instance (ptn_wrapper)

```
restore_module_model dtmf_recver_core -tag init
set_module_model -cell ptn_wrapper -tag my_extract_v1 -type lef
write_module_model_context -type extraction_context -cell ptn_wrapper -
type_specific_options {-extension 50}  -tag my_extract_v1
```

Session 3: Target-level (N)

- Restore the N level design (ptn_wrapper) with its extraction_context add_ons model.

  ```
  restore_module_model ptn_wrapper -tag my_extract_v1 -add_ons {extraction_context}
  ```

- Set and commit the N+1 level detailed LEF model(s) (ram_128x16_test)

  ```
  set_module_model -cell ram128x16_test -type lef -tag my_extract_v1
  commit_module_model
  ```

- Run Extraction

  ```
  rcOut -spef …
  ```

Check-in Version Control (SVN)

Innovus supports a simple way for database version control using tags. Additionally, the Integrated Hierarchical Database is compatible with the Apache subversion (SVN) that can be used for controlling the database versions and revisions. For database version control in the Integrated Hierarchical Database (iHDB) flow:

- A central repository is set-up for checking-out data to the local working directory before invoking Innovus.
- Updated block data models are checked into the central repository after generating models for block-level designs.
- Data is checked-in and checked-out in a sequential order to avoid data locking issues.

Machine Learning Flow

Overview

Although crucial for final power, performance, and area (PPA), preRoute and postRoute delay correlation is difficult to achieve due to three major reasons:

- Different routing topology between Early Global Route (EGR) and NanoRoute
- Different RC engine at the preRoute and postRoute stages
- Exclusion of coupling delay in the preRoute stage

To achieve a better convergence for final timing closure, users typically tune preRoute and postRoute delay/timing correlation either by using different preRoute and postRoute settings, such as clock uncertainty, timing derate, and RC Factor, or by applying some delay/RC settings to introduce extra pessimism or optimism in the preRoute stage. These methods work on all the paths and timing arcs arbitrarily. As a result, preRoute optimization either addresses some wrong paths (pessimism) or misses some real critical paths (optimism), which would need to be fixed later in the postRoute stage.

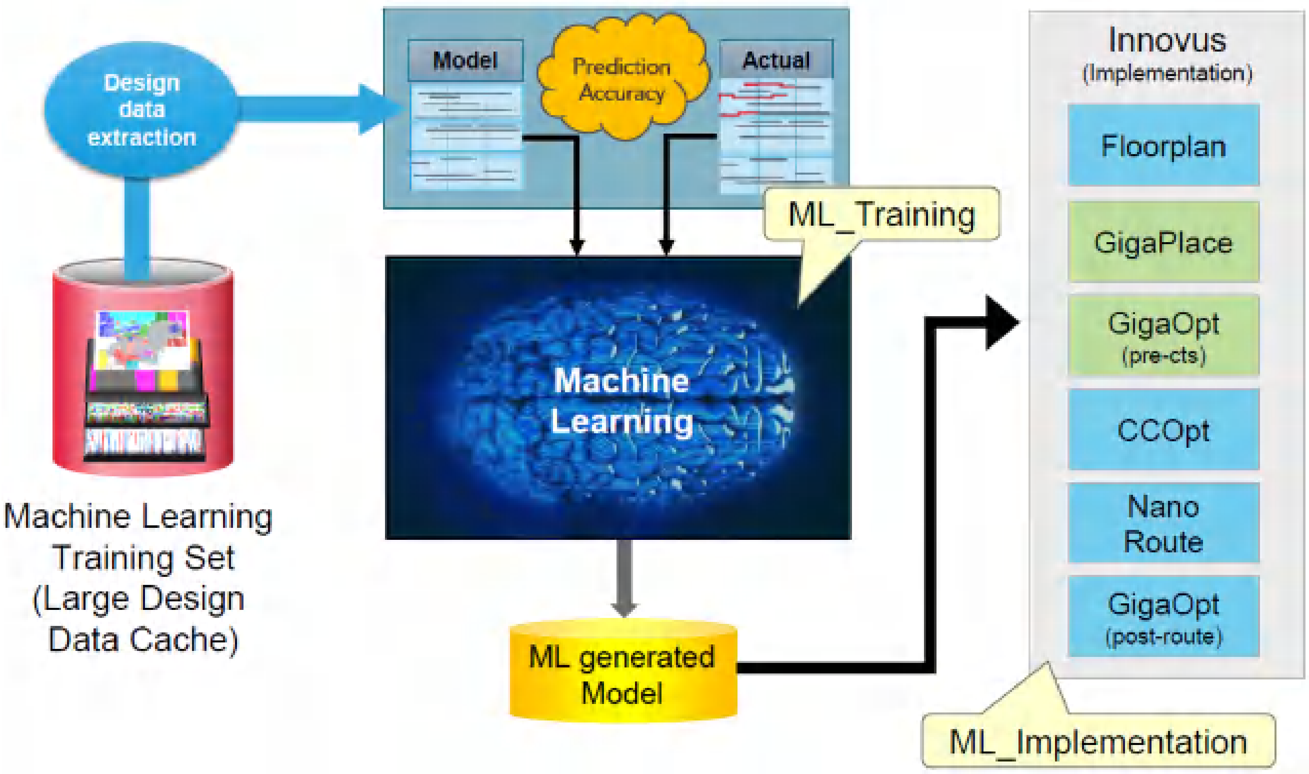
In such cases, the machine learning (ML) based preRoute opt flow provides the ideal solution for improving PPA. In this flow, the basic idea is to learn postRoute delay from the routed database by a supervised learning method. The ML-predicted delays are then fully integrated in preRoute optimization to guide opt transforms. The advantage of this method is that ML-predicted delays are calculated for every timing arc (net or cell) and are adjusted dynamically throughout the preRoute optimization stage.

With the ML-based preRoute opt flow, better preRoute and postRoute delay/timing correlation are shown in most designs, helping you achieve better final PPA and shorter Time-to-Market (TTM).

Innovus Machine Learning Options

The ML capability comprises two basic parts:

1. Training: The training part of the flow is used to generate ML-based specific models: Multiple training runs may be required to generate the ML models. The training program can be done outside Innovus. In addition, correlation tests can also be done through a testing program outside Innovus.
2. Implementation: In the implementation part of the flow, the ML-based training models are used to run the Innovus implementation flow.

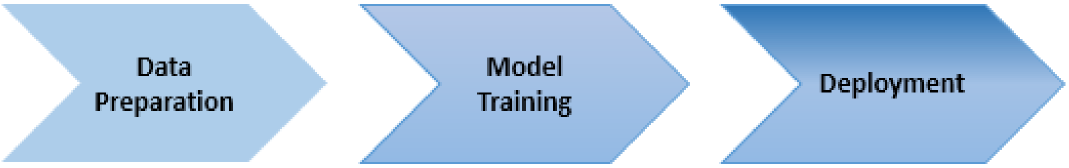


The INVS100 base product supports the ML-based flow, which can be used during placement and optimization to improve PPA of the final design implementation. Innovus provides the following two license options for the ML-based flow:

Name	Short Name	Product Number	Command Line Name	Description

Innovus Machine Learning Training Option	Innovus ML Training Opt	INVS66	`dlTrain` and `dlTest` *External binaries, part of the Innovus installation directory	• Enables the generation of custom ML-based training models and testing • At least one INVS66 license is required • Multiple training licenses can be purchased to run training in parallel
Innovus Machine Learning Implementation Option	Innovus ML Impl Opt	INVS65	`invs_ml`	• Enables Innovus ML implementation flow • At least one INVS65 license is required

The Machine Learning Flow



Data Preparation

If you are running the CCOpt flow, run the ML-based flow from a placed database:

- Load the placed db.

- Run the following command:

  `setMachineLearningMode -training net_cell_delay`

- Run from `ccopt_design` to `routeDesign`.

Do not modify the original flow, except for adding `setMachineLearningMode`.

The data file is dumped out after `routeDesign`. The data file is named as `<`*`design_name`*`>_postroute.db`. If you rename it, be sure to retain the `postroute.db` postfix in the name.

In the Data Preparation step, data files are generated for the machine learning model training. The current recommended flow is to run `routeDesign` from a pure CTS database without doing any optimization, which is referred as the noOpt flow.

To simplify usage, the flow is customized to use the machine learning mode.
Add `setMachineLearningMode -training <`*`delay_type`*`>` and run your original flow. Do not make any other modifications in the flow script.

The purpose of the Data Preparation flow is to generate data for model training. The databases saved in the flow are not usable for further implementation because all optimizations are skipped in this flow.

Model Training

The Model Training step of the ML-based flow is executed using the `dlTrain` external binary, which is part of the Innovus installation directory. As mentioned previously, the training program can be done outside Innovus.

ⓘ Training Command

Use the following command for launching model training:

```
<innovus_installation>/bin/dlTrain -mode train -type <training_type> -datadir
<data_dir> -outdir <model_dir> -pybin <python_bin_dir> -log <log_file_name>
```

Here:

`<data_dir>` is the directory containing the dumped out data, `<design_name>_postroute.db`, from the Data Preparation step.

`<model_dir>` is the directory containing the output model files (`.pb`) and training log files. The model file (`.pb`) naming conventions are as follows:

- net_slew_model: `xxx_slew1_net_xxx.pb`
- net_delay_model: `xxx_net_xxx.pb`
- cell_delay_model: `xxx_slew1_cell_xxx.pb`

`<python_bin_dir>` is the directory containing the python binary.

Note: Model training needs some extra packages that are not installed with Innovus. Contact your Cadence representative for installation of these additional packages.

Deployment

In the Deployment step of the ML-based flow, the ML-predicted delays from the previous step are fully integrated in preRoute optimization to guide opt transforms. By setting `setMachineLearningMode -deployment`, postCTS optimization and TDGR previous layer assignment will use ML-predicted delays.

The flow is as follows:

- Load the placed db.
- `setMachineLearningMode -deployment {models}`
- Run from `ccopt_design` to `postRoute`.

ⓘ Detail Settings

- Net model only:

  ```
  setMachineLearningMode -deployment {net_delay <net_delay_model>}
  ```

  Or

- All net/cell models:

  ```
  setMachineLearningMode -deployment {{net_delay net_delay_slew_model}
  {cell_delay cell_delay_model}} -slew true
  ```

Correlation Test

After completing the training ML delay model, you may want to check the accuracy of the trained ML models by using a correlation test. In this test, the errors between ML delays and postRoute delays (golden) are calculated and compared with the errors between preRoute delays and postRoute delays (golden).

Test Command:

```
<innovus_installation>/bin/dlTest -preroute_db <preroute_db_name> -route_db
<route_db_name> -model <model_directory>
```

The delay correlation test will report delay error statistics by view, including ME (mean error), MAE (mean absolute error), STD (error standard deviation) of PREROUTE vs POSTROUTE and ML vs POSTROUTE. ML model accuracy is guaranteed by smaller ML vs POSTROUTE error (MAE/STD). Scatter plots and histograms are also generated.

Delay Correlation Test Result

View: <*view1_name*>

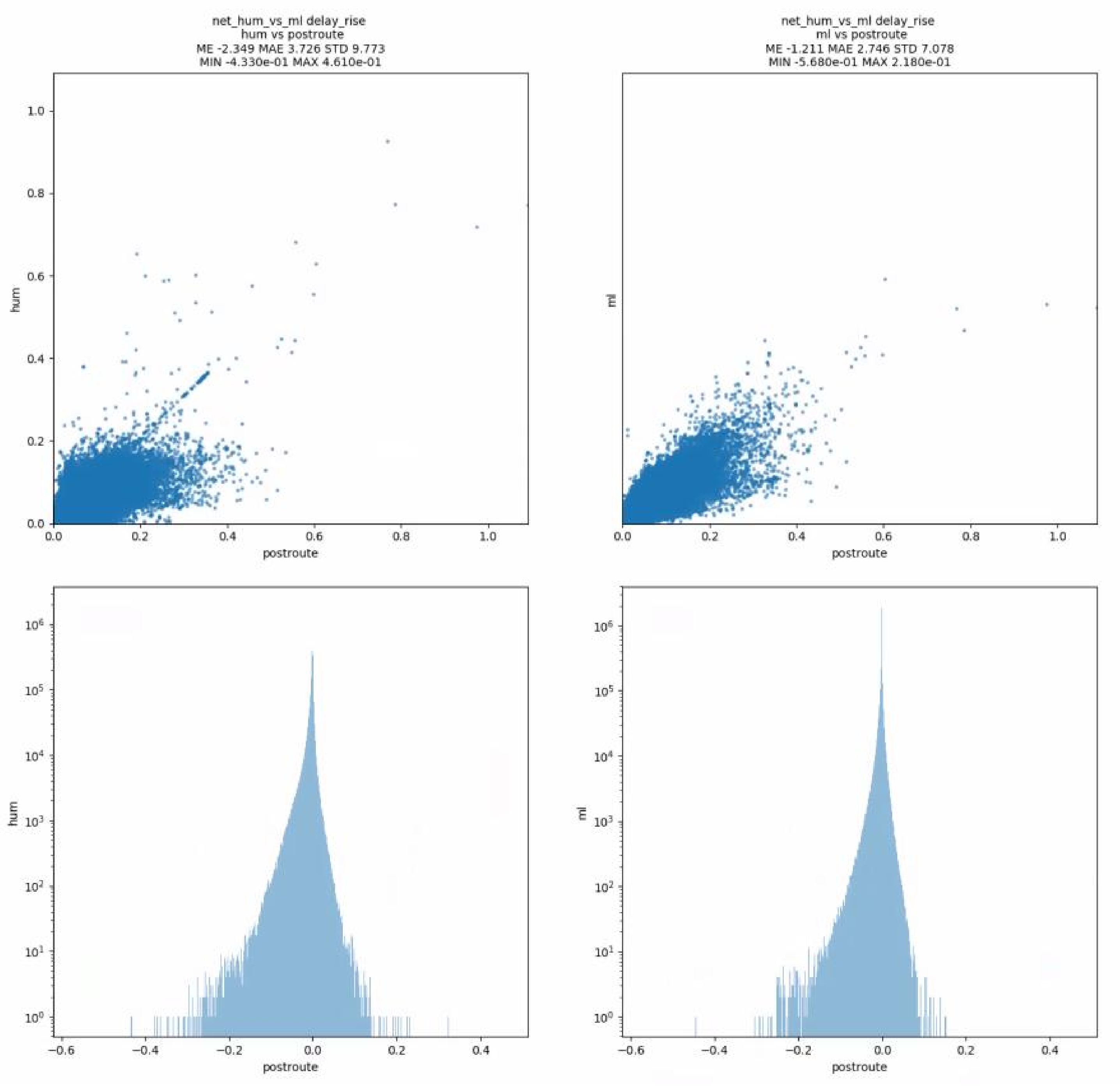
```
net_preroute_vs_ml delay_rise PREROUTE vs POSTROUTE: ME -3.106 MAE 4.450 STD
12.103 MIN -6.060e-01 MAX 4.820e-01 GOLDEN 10.999
net_preroute_vs_ml delay_fall PREROUTE vs POSTROUTE: ME -2.869 MAE 4.257 STD
11.478 MIN -5.440e-01 MAX 4.720e-01 GOLDEN 10.855
net_preroute_vs_ml delay_rise ML vs POSTROUTE: ME -1.925 MAE 3.379 STD 9.215 MIN
-6.030e-01 MAX 2.180e-01 GOLDEN 10.999
net_preroute_vs_ml delay_fall ML vs POSTROUTE: ME -1.625 MAE 3.169 STD 8.508 MIN
-5.240e-01 MAX 2.240e-01 GOLDEN 10.855
```

View: <*view2_name*>

```
net_preroute_vs_ml delay_rise PREROUTE vs POSTROUTE: ME -2.349 MAE 3.726 STD
9.773 MIN -4.330e-01 MAX 4.610e-01 GOLDEN 10.333
net_preroute_vs_ml delay_fall PREROUTE vs POSTROUTE ME -2.316 MAE 3.724 STD 9.736
MIN -4.300e-01 MAX 4.530e-01 GOLDEN 10.402
net_preroute_vs_ml delay_rise ML vs POSTROUTE: ME -1.211 MAE 2.746 STD 7.078 MIN
-5.680e-01 MAX 2.180e-01 GOLDEN 10.333
net_preroute_vs_ml delay_fall ML vs POSTROUTE: ME -1.109 MAE 2.710 STD 6.936 MIN
-4.950e-01 MAX 2.240e-01 GOLDEN 10.402
```

View: <*view3_name*>

```
net_preroute_vs_ml delay_rise PREROUTE vs POSTROUTE: ME -1.766 MAE 3.345 STD
8.747 MIN -5.460e-01 MAX 4.680e-01 GOLDEN 10.740
net_preroute_vs_ml delay_fall PREROUTE vs POSTROUTE: ME -1.961 MAE 3.544 STD
9.069 MIN -5.490e-01 MAX 4.620e-01 GOLDEN 11.109
net_preroute_vs_ml delay_rise ML vs POSTROUTE: ME -0.998 MAE 2.431 STD 6.273 MIN
-6.240e-01 MAX 2.410e-01 GOLDEN 10.740
net_preroute_vs_ml delay_fall ML vs POSTROUTE: ME -1.014 MAE 2.552 STD 6.448 MIN
-5.950e-01 MAX 2.480e-01 GOLDEN 11.109
```



Typically, correlation should be tested on two databases, before and after routing, with the same netlist.

Usually, the delay correlation is tested on databases saved in the Data Preparation flow (noOpt flow), in which trackOpt is turned off internally. The databases saved before and after `routeDesign` can be used directly for correlation test because they have the same netlist.

However, if you want to test the ML model correlation in databases with different netlists rather than that in the noOpt flow (for example, using the database saved in the refOpt flow, which is the database after running `routeDesign -trackOpt` in the baseline run), you need to specify only `-route_db` in the test command, and the preroute_db can be generated from `dlTest` itself by calling `earlyGlobalRoute` on the specified route_db to make sure that you are performing correlation test on pre-route and post-route databases with the same netlist.

3

Design Import and Export Capabilities

Data Preparation

Generating a Technology File

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

Creating Technology Information Using LEF

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

Creating Technology Information Using OpenAccess

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

Preparing Physical Libraries

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

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

Using LEF to Create Physical Libraries

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

- Use the Abstract Generator.
  For more information, see the *Cadence Abstract Generator User Guide*.
- Create LEF `MACRO`s manually.
  For more information, see the *LEF/DEF Language Reference.*

Creating OpenAccess Physical Libraries

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

Unsupported LEF and DEF Syntax

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

Unsupported LEF 5.7 Syntax

The software parses but ignores the following LEF 5.7 syntax:

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

Nondefault Rule	[DIAGWIDTH *diagWidth* ;] [HARDSPACING ;] [USEVIARULE *viaRuleName* ;]
Via Rule Generate	[DEFAULT]

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

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

Unsupported DEF 5.7 Syntax

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

DEF Statement	Unsupported Syntax
Blockages	[+ SLOTS]
Groups	[+ PROPERTY { *propName propValue* }...]
Extensions	All BEGINEXT syntax
History	All HISTORY syntax

Nets	[+ SYNTHESIZED] [+ VPIN *vpinName* [LAYER *layerName*] *pt pt* [PLACED *pt orient* | FIXED *pt orient* | COVER *pt orient*]] [+ SUBNET *subNetName* [({ *compName pinName* | PIN *pinName* | VPIN *vpinName* })] [NONDEFAULTRULE *ruleName*]] **Note:** SUBNET NONDEFAULTRULE is ignored; routing uses rule for NET. [+ USE {RESET | SCAN | TIEOFF}] **Note:** Supports ANALOG, CLOCK, GROUND, POWER, and SIGNAL. [+ PATTERN {STEINER | WIREDLOGIC} [+ ESTCAP *wireCapacitance*] [+ SOURCE {DIST | NETLIST | TEST | USER}
Pins	[+ USE {TIEOFF | SCAN | RESET} **Note:** Supports SIGNAL, POWER, GROUND, ANALOG, and CLOCK. [+ DIRECTION FEEDTHRU] [+ NETEXPR " *netExprPropName defaultNetName* "] [+ SUPPLYSENSITIVITY *powerPinName*] [+ GROUNDSENSITIVITY *groundPinName*]
Pin Properties	All PINPROPERTIES syntax
Property Definitions	The object types: GROUP, REGION, and ROW
Regions	[+ PROPERTY { *propName propVal* }...]
Rows	[+ PROPERTY { *propName propVal* }...]
Slots	All SLOTS syntax

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

Styles	All `STYLES` syntax

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

DEF Statement	**Unsupported Syntax**

Nets
(Regular Wiring Statement)
	`[` *`orient`* `]`
	`[STYLE` *`styleNum`* `]`

Generating the I/O Assignment File

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

Innovus to determine the locations automatically.

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

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

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

This section discusses the following topics:

- Creating an I/O Assignment File
- Creating a Rule-Based I/O Assignment File
- I/O Pad and Pin Assignment Examples
- Performing Area I/O Placement

Creating an I/O Assignment File

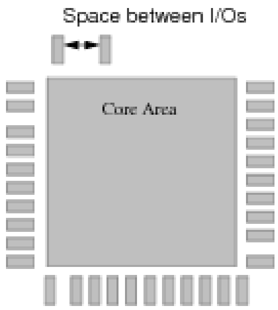
You manually create an I/O assignment file using the following template:

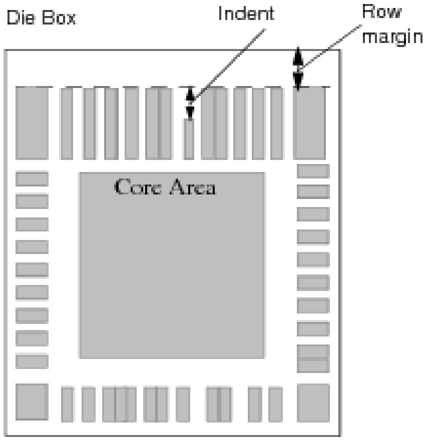
```
( globals

   version = 3

   space = 0

   io_order = default

)


( row_margin

  ( top

  ( io_row ring_number = 1 margin = 0)

  ( io_row ring_number = 2 margin = 630)

  ( io_row ring_number = 3 margin = 830)
```

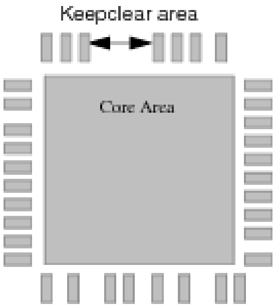
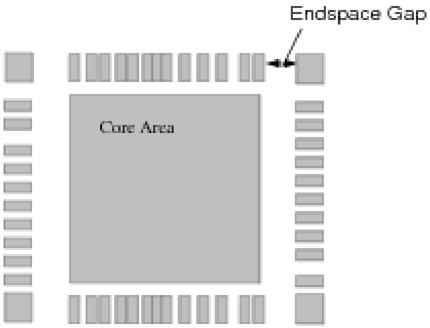
```
 )
  ( bottom
  ( io_row ring_number = 1 margin = 0)
  ( io_row ring_number = 4 margin = 0)
  ( io_row ring_number = 5 margin = 0)
 )
  ( left
  ( io_row ring_number = 1 margin = 0)
  ( io_row ring_number = 6 margin = 200)
  ( io_row ring_number = 7 margin = 0)
)
  ( right
  ( io_row ring_number = 1 margin = 0)
  ( io_row ring_number = 8 margin = 200)
)
  )

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

    ( left
      ( locals ring_number = 6)
( inst name="ins_5"   offset = 5896.2 )
( inst name="ins_7"  space = 5)
( inst name="ins_9"  place_status = placed)

( inst name="ins_10" orientation = R0)

( inst name="ins_11" )

( inst name="ins_12" )

( inst name="ins_14" space = 0)

   ( locals ring_number = 7)

( inst name="ins_6"   offset = 5826.4 )

( inst name="ins_8" )

( inst name="ins_13" )

( inst name="ins_15" )

( inst name="ins_17" )

( inst name="ins_19" )

( inst name="ins_21" )

( inst name="ins_23" )

( inst name="ins_25" )


( bottomleft

      (locals ring_number = 1)

      ( inst name="ins_1")

   )


 ( bottom

     ( locals ring_number = 4    )

( inst name="ins_167" offset = 3946.8)

( inst name="ins_168" )

( inst name="ins_169"  space = 5)

( inst name="ins_170" )

( inst name="ins_171" )
( inst name="ins_172" space = 0)
( inst name="ins_173" )
( inst name="ins_174" )
( inst name="ins_175" )
( inst name="ins_176" )
      ( locals ring_number = 5)
( inst name="ins_261"   offset = 11812.3 )
( inst name="ins_262" )
( inst name="ins_263" )
( inst name="ins_264" )
( inst name="ins_265" )
( inst name="ins_266" )
( inst name="ins_267" )

( bottomright
       (locals ring_number = 1)
( inst name="ins_2" orientation=R0 )
    )
( right
      ( locals ring_number = 8    )
( inst name="ins_313"   offset = 200 )
( inst name="ins_315" )
( inst name="ins_316" )
( inst name="ins_318" )
( inst name="ins_320" )
( inst name="ins_322" )
( inst name="ins_324" )
( inst name="ins_326" )
( inst name="ins_328" )

( inst name="ins_330" )

( inst name="ins_332" )

( inst name="ins_334" )

( inst name="ins_336" )

( inst name="ins_337" )

( inst name="ins_338" )

( inst name="ins_339" )

( inst name="ins_341" )

( inst name="ins_343" )

( inst name="ins_344" )

( topright

       (locals ring_number = 1)

( inst name="ins_3")

    )

( top

      ( locals ring_number = 2    )

 ( inst name="ins_610"   offset = 100 )


      ( locals ring_number = 3)

( inst name="ins_611"   offset = 200 )

( inst name="ins_612" )

( inst name="ins_614" )

( inst name="ins_616" )

( inst name="ins_617" )

( inst name="ins_619" )

)

)
```

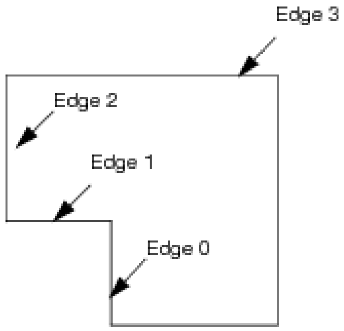
The following entries are included in the template:

`globals`		
	`version = 3`	Specifies the beginning of a new I/O format.
	`io_order`	Specifies the order of the I/O pads and pins. This can be: • clockwise • counterclockwise • default **Note:** The default I/O order for a vertical edge is from the bottom to the top, and for a horizontal edge, it is from the left to the right.
	`total_edge`	Specifies the number of edges for the rectilinear block design. The edges are numbered starting from 0. For example, if the *`total_edge`* is 4, then the edges are numbered as edge 0, edge 1, edge 2, and edge 3. **Note:** You must verify that the total number of edges that you specify matches with the value in the destination design.
	`space`	Specifies the global I/O pin spacing, in µmeters.
`iopad locals`		
	`space`	Specifies the local I/O pad spacing, in µmeters. **Note:** This space setting is honored by the first cell on one edge, when xy or offset is not specified.
	`ring_number`	Specifies the ring number in which the I/O pad is placed.
	`row_name`	Specifies the I/O *`row`* name.
`iopad instance`		
	`name`	Specifies the name of the I/O instance.
	`x, y`	Specifies the absolute `x,y` location of the I/O pad instance, starting from the lower left corner. **Note:** Specifying x,y location for sides and edges of I/O pads is not supported in the I/O file.

	skip	Specifies the distance, in µmeters, of the I/O pad from the previously defined I/O pad. The value that you specify here is valid only for this cell.
	space	Specifies the spacing, in µmeters, between the pad being defined and the previously defined pad. The value that you specify here, overrides the global space setting. 
	offset	Specifies the distance in microns from the IO ring edge to the pad edge based on io_order constraint. The value that you specify here is valid only for this cell.
	Note: For one I/O pad, you can specify only one of the following parameters: • skip • space • offset If you specify all the three parameters, only the last parameter that you define, is considered for I/O pad placement.	

	indent	Specifies the offset, in µmeters, from the row margin.  However, for designs with single I/O ring, row margin is 0. Hence, indent is the offset of the I/O pad from the die boundary.
	orientation	Specifies the orientation of the I/O.
	place_status	Specifies the placement status of the I/O pad instance. This can be: • placed • covered • fixed *Default* : fixed.

	keepclear	Specifies an area on the chip where you cannot place pins or pads. Specify a range between *begin* and *end* , in µmeters, on the chip side in which pins and pads cannot be placed. **Note:** You must define pad cells in the order in which they appear in the design. 
	cell	Specifies the physical I/O cell.
	endspace gap	Specifies the space, in µmeters, between the corner pad and the last I/O pad for the specified side of the design. 
`iopin locals`		

	side	Specifies the side of the I/O pin. This can be: • top | north • left | west • right | east • bottom | south
	edge num = 0	Specifies the edge number of the I/O pin, with `edge num = 0` starting from the left side of the lowest y coordinate and the left most corner, in the clockwise direction. 
	space	Specifies the spacing, in µmeters, between the previously defined pin and the pin being defined. The value that you specify here, sets the global space setting.
`iopin`		
	pin name	Specifies the name of a pin. Specify I/Os as pins for block designs.
	layer	Specifies the metal layer on which the pin must be placed.
	width	Specifies the width of the pin in µmeters. It is the length of the edge that is centered at the reference point.
	depth	Specifies the length of the pin in µmeters.
	up	Specifies the details of internal I/O pins.
	x, y	Specifies the absolute `x,y` location of the internal I/O pin. **Note:** The I/O file supports specifying `xy` location for internal I/O pins only.

	#	Specifies the incremented I/O pin edge number.

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

Row Margin		
	side	Specifies the side of the I/O row margin. This can be: • top • north • left • west • right • east • bottom • south
	ring_number	Specifies the I/O ring number on which the I/O rows are placed, with ring 1 being the outer most ring.
	margin	Specifies the distance, in microns, from the die boundary edge to the I/O row edge.

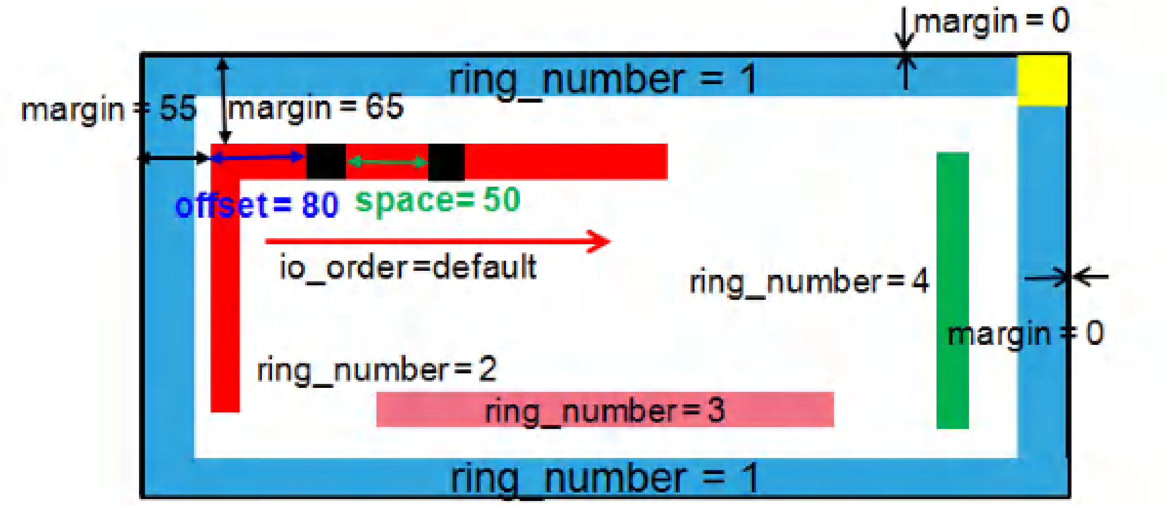
Note: You can use the Edit I/O Ring form to specify I/O pad rings and row margins for multiple rows. Alternatively, to achieve the same using text commands, you must first use the `setIoRowMargin` command to set the distance from the die boundary edge to start of each row and then use the `placePadIO`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

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

Example for margin/offset/space usage:

```
(globals
    version = 3

       space= <value>
     io_order = default

)
```

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

)

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

   (bottomleft
    )
    (bottom
    )
    (bottomright
    )
    (right
    )
)
```



Specifying Area I/O Information

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

- Bump
  A bump is a piece of metal that works as a bonding pad to the package. When defining a bump, you must specify its master bump cell and its physical location. You can generate one bump or mutiple bumps of the same bump cell type.
  - To define signal bumps, use the following syntax:

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

    For example:

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

  - To define power bumps, use the following syntax:

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

For example:

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

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

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

  For example:

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

Example of an Area I/O file

```
(globals
version = 3
io_order = default
)
(iopad
(top
(inst name="IOPADS_INST/Pibiasip" offset=35.2800 place_status=placed )
(inst name="IOPADS_INST/Ppllrstip" offset=108.8050 place_status=placed )

)
(left
(inst name="IOPADS_INST/Ptdspip03" offset=35.2800 place_status=placed )
(inst name="IOPADS_INST/Ptdspip04" offset=106.8500 place_status=placed )

)
(bottom
(inst name="IOPADS_INST/Pscanout1op" offset=35.2800 place_status=placed )
(inst name="IOPADS_INST/Pvcopop" offset=108.8050 place_status=placed )

)
(right
(inst name="IOPADS_INST/Ptdspop04" offset=35.2800 place_status=placed )
(inst name="IOPADS_INST/Ptdspop05" offset=112.3550 place_status=placed )

)
(bumps

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

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

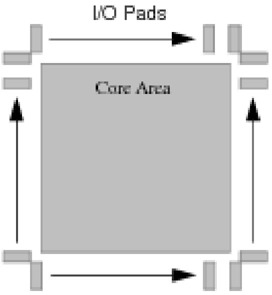
Creating a Rule-Based I/O Assignment File

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

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

I/O Pad and Pin Assignment Examples

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

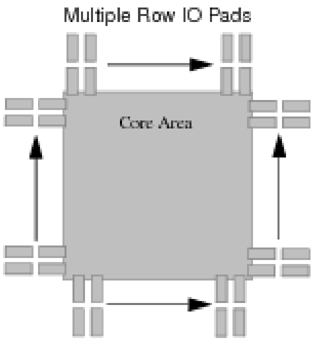


```
version = 3
```

```
io_order = clockwise

total_edge = 4

space = 1.06

(inst

     name = IOPADS_INST/pad1 W

     offset = 235.0000

     orientation = R0

     place_status = fixed

  )

  (inst

     name = IOPADS_INST/pad2 W

     offset = 296.1250

     orientation = R0

     place_status = fixed

  )
```

Assigning Pads for Multiple Rows

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

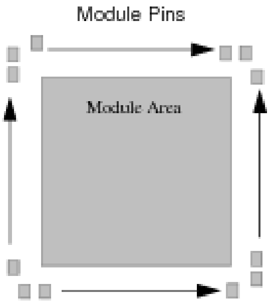


```
version = 3

io_order = clockwise
```

```
total_edge = 4

space = 1.06


iopad

(topright

      (locals

          ring_number = 1

      )

      ( instname = IOPADS_INST/pad1 W

      offset = 235.0000

      )

      (locals

      ring_number = 2

      )

      ( instaname = IOPADS_INST/pad2 W

      offset = 296.1250

      )

)
```

Assigning Module Pins

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



```
version = 3

(iopin

     (top | north | edge num = 0

         (locals

         space = 1.2

         )

         (pin name = address[14] N

         layer = 3

         width = 0.28

         depth = 0.28

         offset = 19.4700

         place_status = fixed

         )

         (pin name = address[14] N

         layer = 4

         width = 0.38

         depth = 0.38

         offset = 39.2700

         place_status = fixed

         )

     )
```

)

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, CLASS PAD or CLASS BLOCK with CLASS BUMP` in a LEF file. See the "*Data preparation*" section in the Flip Chip Methodologies chapter.

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

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

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

MACRO INBUF

     CLASS PAD AREAIO ;

     FOREIGN INBUF 0.00 0.00 ;
```

```
    ORIGIN 0 0 ;

    SIZE 210 BY 100.8 ;

    SYMMETRY X Y R90 ;

    SITE 10 ;

    PIN PAD

        DIRECTION INPUT ;

        USE SIGNAL ;

        PORT ;

        LAYER M6 ;

           RECT 95.0 40.0 115.0 60.0 ;

        END

    END PAD
```

Note: The bump status can be saved in the `DEF` file only if the bump status is `FIXED` or `COVER`. See Defining BUMP CELL Placement Status.

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

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

```
MACRO PVDD1DGZ

  CLASS PAD AREAIO ;

    FOREIGN PVDD1DGZ 0.000 0.000 ;

  ORIGIN 0.000 0.000 ;

  SIZE 40.000 BY 35.280 ;

  SYMMETRY x y r90 ;

  SITE IO1 ;
```

```
    PIN VDD
         DIRECTION OUTPUT ;
         USE POWER ;
         PORT
         CLASS BUMP ;
         LAYER METAL8 ;
            RECT 5.0 25.0 15.0 35.0 ;
         END
     END VDD
  END PVDD1DGZ
```

For more information, see "Macro Pin Statement" in the LEF/DEF Language Reference and the "CLASS BUMP Attribute" section in the Flip Chip Methodologies chapter.

Defining BUMP CELL in LEF

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

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

```
          END
     END PAD
END BUMPCELL
```

Defining BUMP CELL Placement Status

You can define the bump cell placement status, `FIXED | COVER` for a bump object in the design, in a DEF/IN file or using the Attribute Editor in Innovus. The bump placement status, `FIXED` or `COVER`, could be saved to DEF file.

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

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

```
Bump: Bump_83_2_8 BUMPCELL 255.720 855.720 refclk -fixed  -placeStatus placed

Bump: Bump_82_1_8 BUMPCELL 155.720 855.720 pllrst -fixed  -placeStatus cover

Bump: Bump_81_0_8 BUMPCELL 55.720 855.720 ibias -fixed  -placeStatus fixed
```

Importing LEF Files

To import the LEF files, use the following procedure:

1. Select *File - Import Design*.
   The Design Import form appears.
2. On the Design page, enter the names of the Verilog files, and choose a top cell assignment option.
3. In the LEF Files field, type the LEF file names to import, and include the file that contains the `CLASS PAD AREAIO` statement. Or, you can click on the … icon to the right of the field to select files.
4. Click *OK*.
   The Design Import form closes and Innovus imports the data.

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

1. Select *File - Load - Floorplan* or run the `loadFPlan` text command.
2. Select *File - Load - I/O File* or run the `loadIoFile` text command.

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

```
IOFile:  iofile_name
```

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

For more information on the I/O assignment file, see "Creating an I/O Assignment File".

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

1. Select *File - Save - Floorplan*. Fill out the form and click *Save*.
   As an alternative, you can specify the text command.
2. Select *File - Save - I/O File*. Fill out the form and click *Save*.
   As an alternative, you can specify the 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 text command.
  Specify the 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 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 command.

Preparing Timing Libraries

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

Encrypting Libraries

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

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

Parameters

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

Preparing Timing Constraints

To import timing constraints, use the `write_script` or `write_sdc` command from within Genus. 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 inside Genus, you can use the following command:

```
write_sdc
```

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

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

Preparing Capacitance Tables

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

For more information on preparing capacitance tables, see chapter RC Extraction.

Preparing Data for Delay Calculation

If you want to use the SignalStorm® nanometer delay calculator, see chapter Base Delay Analysis for information about preparing ECSM libraries.

Preparing Data for Crosstalk Analysis

For information on preparing data for crosstalk analysis, see chapter Analyzing and Repairing Crosstalk. For more information on preparing cdB noise libraries using the `make_cdB` utility, see the "*make_cdB Noise Characterizer User Guide*."

Checking Designs

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

To perform these checks, use the following commands:

`checkDesign`

`checkNetlist`

`check_timing`

`check_design`

You can check for the following data:

- Physical library
- Timing library
- Netlist

- I/Os
- Tie-high and tie-low pins
- Power and ground pins

Cadence recommends that you check libraries and data as follows:

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

Preparing Data in the Timing Closure Design Flow

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

Converting iPRT Format to LEF

The `iprt2lef` translator converts DRC rules, place-and-route technology data, and RCX data from iDRC, iPRT and iRCX format to the technology LEF format.

For more information about this translator, refer to the *iPRT to LEF Translator Application Note* on Cadence Online Support.

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

Importing and Exporting Designs

Overview

The Innovus® Implementation System (Innovus) software provides the following options for saving, restoring, importing, and exporting design data:

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

Verifying Data before Importing a Design

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

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

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

```
getCheckMode
```

Preparing the Design Netlist

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

- To ensure that the design is uniquified automatically after the top cell is flattened, set the following global variable to `1`:
  init_design_uniquify

The init_design Import Flow

All designs are saved in Innovus using the `init_design` import model. In this design import model, all analyses are configured the same way using the multi-mode/multi-corner (MMMC) style of configuration, and the configurations are used directly for initialization.This section introduces the basics of the `init_design`-based data flow. This section has the following subsections:

- init_design Simple Data Flow
- Supported init_design Invocation Methods

init_design Simple Data Flow

In the `init_design`-based data flow:

- Global variables that store data explicitly required for the initialization process are prefixed with `init_`.
- All `init_*` global variables have `help`, can be queried, and are stored by `saveDesign` in the `.globals` file.
- Since MMMC syntax can be used to configure one mode or corner as well as many, `init_design` relies on a valid MMMC specification to provide the necessary timing, SI, constraint, and extraction related data for the system

Note: The Innovus `init_design` style configuration cannot be restored by 10.x or earlier releases of the software directly.

`init_*` style variables are used to store design-level and physical data. For example, the `init_mmmc_file` variable is the pointer to the file containing the MMMC configuration. In addition, the `init_cpf_file` provides a pointer to the design's Common Power Format (CPF) file. This is significant for initialization since an MMMC configuration can be derived from CPF. So while a valid

MMMC configuration must be available for `init_design`, it is not required that it come specifically from the `init_mmmc_file` pointer.

The Tcl global variables used by `init_design` are:

- init_abstract_view
- init_cpf_file
- init_design_netlisttype
- init_design_settop
- init_gnd_net
- init_import_mode
- init_io_file
- init_layout_view
- init_lef_file
- init_mmmc_file
- init_oa_default_rule
- init_oa_design_cell
- init_oa_design_lib
- init_oa_design_view
- init_oa_ref_lib
- init_oa_search_lib
- init_oa_special_rule
- init_pwr_net
- init_top_cell
- init_verilog

For more information on the init_* globals, see the Import and Export Global Variables section in the *Innovus Text Command Reference*. Several possible `init_design` scenarios are discussed in a later section of this chapter.

Supported init_design Invocation Methods

You have seen how to get all the data required to bring up an Innovus session with `init_design`. Let us now look at different examples of actually invoking the `init_design` command:

- Using a Pointer to an MMMC Configuration File
- Using a Pointer to a CPF File
- Using init_design with an Inline MMMC Script
- Using Physical-Only Flow

Using a Pointer to an MMMC Configuration File

One of the most common ways of invoking `init_design` is to first use initialization variables to define where to find the key pieces of data. The `init_mmmc_file` variable is used to point to a functioning MMMC configuration. Here, functioning is defined as follows:

- The MMMC configuration must include a `set_analysis_view` command and be complete and correct enough to initialize the specified `-setup` view
- At a minimum, timing library information is required.

The following example uses a pointer to an MMMC configuration file before invoking `init_design`:

```
set init_verilog "top.v"

set init_top_cell "top"

set init_mmmc_file "viewDefinition.tcl"

init_design
```

Instead of having the init globals asserted one-by-one, you can also source the file containing the variable settings and then initialize the design as follows:

```
source test.globals

init_design
```

Note: Here, it is assumed that `test.globals` is configured in MMMC mode.

Using a Pointer to a CPF File

In the following example, a CPF file is used in place of an explicit `viewDefinition.tcl` file. The MMMC configuration is derived from the CPF:

```
set init_verilog "top.v"

set init_top_cell "top"

set init_cpf_file "top.cpf"

init_design
```

Here:

- The CPF must be a MMMC style-CPF, which means it must contain at least one analysis view definition.
- The design is initialized based on the default power domain's library information.

Using init_design with an Inline MMMC Script

If you have a script which is creating the MMMC configuration on-the-fly rather than having a pointer to static file, you can still use the `init_design` flow successfully. However, there is a circular dependency problem that needs to be resolved. `set_analysis_view` cannot be issued until the design has been initialized by `init_design`, but init_design requires a complete MMMC configuration including the requisite -setup and -hold view information. The solution is to use the `-setup` and `-hold` options of the init_design command itself, instead of using `set_analysis_view` in this scenario.

```
set init_verilog "top.v"

set init_top_cell "top"

create_delay_corner -name my_delay_corner_max
             -library_set my_max_library_set
             -rc_corner   my_rc_corner_worst

create_delay_corner -name my_delay_corner_min
             -library_set my_min_library_set
             -rc_corner   my_rc_corner_worst

create_analysis_view -name my_analysis_view_setup
          -constraint_mode my_constraint_mode
          -delay_corner    my_delay_corner_max
```

```
create_analysis_view -name my_analysis_view_hold
          -constraint_mode my_constraint_mode
          -delay_corner    my_delay_corner_min

     init_design -setup my_analysis_view_setup
                -hold  my_analysis_view_hold
```

Using Physical-Only Flow

You can also run `init_design` in the absence of an MMMC configuration. This initializes the system into physical-only mode. No access to the timing part of the system is provided under this mode. To reinitialize, you would need to exit the software or run the `freeDesign` command.

Importing Designs using the GUI

Before you import a design, you must first prepare the data. For more information, see the Data Preparation chapter in the *Innovus User Guide*.

Importing an OpenAccess Design

To import an OpenAccess design, complete the following steps:

- Select *File - Import Design.*
- Select *OA*.
- Specify the following information:
  - *Library*
    Specifies the OpenAccess database library.
  - *Cell*
    Specifies the OpenAccess database cell.
  - *View*
    Specifies the OpenAccess database view.
- Specify the following OpenAccess technology and physical library information:
  - *OA Reference Libraries*
    Specifies the OpenAccess reference libraries to import. The first OpenAccess reference

library listed in this field must contain the technology information for the leaf cells. Each reference library is processed using the abstract view name list (*Abstract View Names*).

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

  - *OA Abstract View Names*
    Specifies the OpenAccess view names that the software should examine to find the equivalent LEF `MACRO` information (for example, `PINS`, `OBS`, `FOREIGN`).
  - *OA Layout View Names*
    Specifies the layout view names (separated by spaces) to import.
- Click *Save* or *OK*.
  - *Save* saves your settings to a configuration file. The design is not imported.
  - *OK* uses the current settings to import the design. The configuration file is not updated.

Importing a Design with LEF and Verilog

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

- Select *File - Import Design.*
- Specify the gate-level Verilog netlist files to import in the *Files* text field.
- Select one of the following options to specify the top cell:
  - *Auto Assign*
    Automatically extracts the top cell name from the netlist, provided the netlist contains only one design.
  - *By User*
    (Default) Specifies the name of the top cell when a netlist contains more than one design (more than one top design name). The top cell name specified is the design the software imports and processes.
- Specify the LEF files to import. You must specify the technology LEF file first, then specify the

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

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

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

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

Loading a Previously Saved Global Variables File

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

- Select *File - Import Design*.
- Click *Load*. The Load Global Variables form is displayed.
- Select the directory of the file you want to load.
- Select *Global Variable files (*.globals)* in the *Files of type* field.
- Specify a file name or click on the filename in the window. The filename suffix is `.global`.
- Click *Open*. The Load Global Variables form closes. The global variable file is loaded.
- In the Design Import form, continue to specify data you want to import into the design.
- Click *Save* or *OK*.
  - *Save* saves your settings to the global file. The design is not imported.
  - *OK* saves your settings to the global file and starts the design import process. This might take several minutes to complete, depending on the size of your design. When the design is loaded, the *Design Import* form closes and the design displays in the Innovus main window.

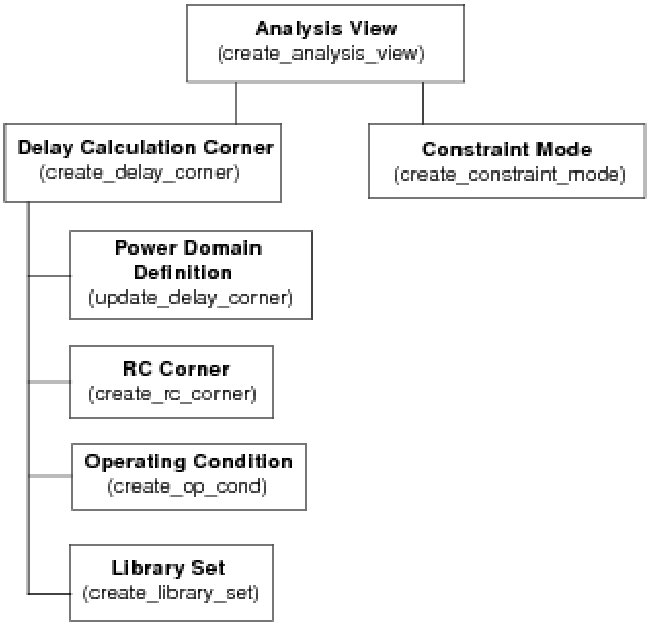
Handling Verilog Assigns

Verilog assign statements may be added, removed, or replaced with buffers automatically by Innovus. All Innovus applications, including GigaOpt, CTS, CCopt, Place, Route, Hierarchy/ILM Flow, MSV, and manual ECO, can handle verilog assign nets natively.

Configuring the Setup for Multi-Mode Multi-Corner Analysis

Multi-mode multi-corner analysis uses a tiered approach to assemble the information necessary for timing analysis and optimization. Each top-level definition (called an analysis view) is composed of a delay calculation corner and a constraint mode. The active analysis views defined in the software represent the different design variations that will be analyzed.

Figure 6-1 Hierarchical Analysis View Configuration



Creating Library Sets

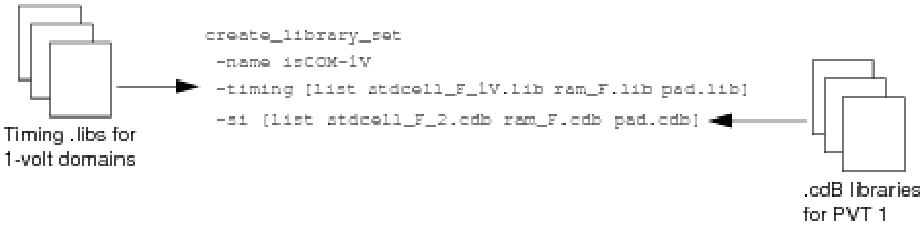
Complex designs can require the specification of multiple library files to define all of the standard cells, memory devices, pads, and so forth, included in the design. Different library sets can be defined to provide uniquely characterized libraries for each delay corner or power domain.

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

The same library set can be referenced multiple times by different delay calculation corners.
To create a library set, use the following command:

```
create_library_set
```

The following figure shows the creation of a library set that associates timing libraries and cdB libraries with a nominal voltage of 1 volt with the library name `IsCOM-1V`:



```
create_library_set
 -name isCOM-1V
 -timing [list stdcell_F_1V.lib ram_F.lib pad.lib]
 -si [list stdcell_F_2.cdb ram_F.cdb pad.cdb]
```

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 - MMMC Browser, 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 Edi*t Operating Condition* form.

ⓘ You can edit a virtual operating condition before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

- Choose *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 - MMMC Browser, 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-0ff 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 - MMMC Browser, and double click on the name of the RC corner object you want to edit.

ⓘ You can use the update_rc_corner command or the Edit RC Corner form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

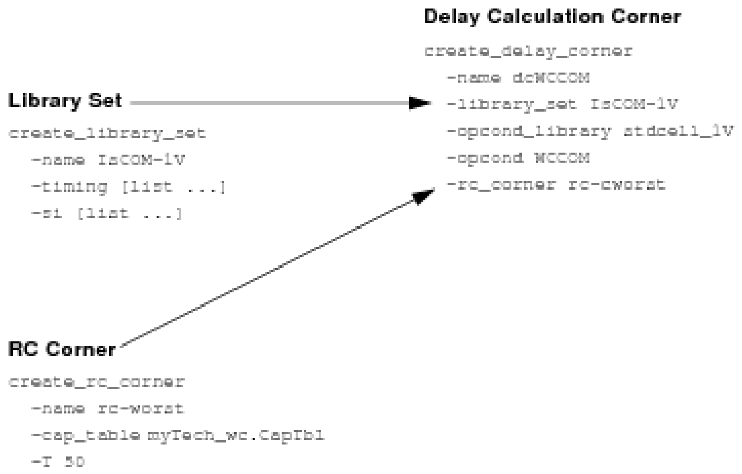
Creating Delay Calculation Corner Objects

A delay calculation corner provides all of the information necessary to control delay calculation for a specific analysis view. Each corner contains information on the libraries to use, the operating conditions with which the libraries should be accessed, and the RC extraction parameters to use for calculating parasitic data. Delay corner objects can be shared by multiple top-level analysis views. To create a delay calculation corner, use the following command:

```
create_delay_corner
```

- Use separate delay calculation corners to define major PVT operating points (for example, Best-Case and Worst-Case).
- 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:



```
create_delay_corner
  -name dcWCCOM
  -library_set lsCOM-1V
  -opcond_library stdcell_1V
  -opcond WCCOM
  -rc_corner rc-cworst
```

```
create_library_set
  -name lsCOM-1V
  -timing [list ...]
  -si [list ...]
```

```
create_rc_corner
  -name rc-worst
  -cap_table myTech_wc.CapTbl
  -T 50
```

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 Edi*t 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 - MMMC Browser, 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* - *MMMC Browser*, 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.

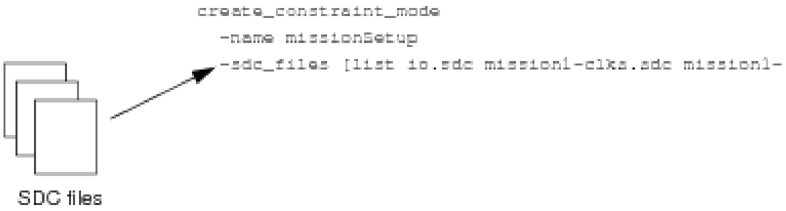
> ⓘ You can use the update_delay_corner command or the Edit Power Domain form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, 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`:



```
create_constraint_mode
  -name missionSetup
  -sdc_files [list io.sdc mission1-clks.sdc mission1-
```

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 - MMMC Browser, and double click on the name of the constraint mode object you want to edit.

> ⓘ You can use the update_constraint_mode command or the Edit Constraint Mode form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

Entering Constraints Interactively

You can use the `set_interactive_constraint_modes` command to update assertions for a multi-mode multi-corner constraint mode object, and have those changes take immediate effect.

Specifying `set_interactive_constraint_modes` puts the software into interactive constraint entry mode for the specified constraint mode objects. Any constraints that you specify after this command will take effect immediately on all active analysis views that are associated with these constraint modes. By default, no constraint modes are considered active.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all views in the current session that are associated with the constraint mode `func1`:

```
set_interactive_constraint_modes func1

set_propagated_clock [all_clocks]
```

The software stays in interactive mode until you exit it by specifying the `set_interactive_constraint_modes` command with an empty list:

```
set_interactive_constraint_modes { }
```

All new assertions are saved in the SDC file for the specified constraint mode when you save the design (`saveDesign`).

The `all_constraint_modes` command can be used to generate a list of constraint modes as the argument for this command.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all active analysis views in the current session.

```
set_interactive_constraint_modes [all_constraint_modes -active]

set_propagated_clock [all_clocks]
```

Use the `get_interactive_constraint_modes` command to return a list of the constraint mode objects in interactive constraint entry mode.

Note: Interactive constraint mode only works when the software is in multi-mode multi-corner timing analysis mode. In min/max analysis mode, constraints are always accepted interactively.

Constraint Support in Multi-Mode and Multi-Mode Multi-Corner Analysis

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

- `create_clock`
- `create_generated_clock`
- `set_annotated_check`
- `set_annotated_delay`
- `set_annotated_transition`
- `set_case_analysis`
- `set_clock_gating_check`
- `set_clock_groups`
- `set_clock_latency`
- `set_clock_sense`
- `set_clock_transition`
- `set_clock_uncertainty`
- `set_disable_timing`
- `set_drive`
- `set_driving_cell`

- set_false_path
- set_fanout_load
- set_input_delay
- set_input_transition
- set_load
- set_max_delay
- set_max_time_borrow
- set_max_transition
- set_min_delay
- set_min_pulse_width
- set_multicycle_path
- set_output_delay
- set_propagated_clock
- set_resistance

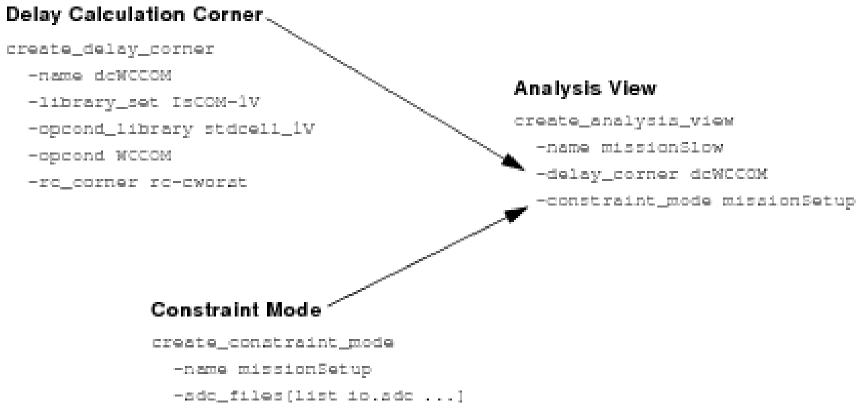
Note: Path groups defined with group_path are considered to be global definitions across all analysis views.

Creating Analysis Views

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:



```
create_delay_corner
  -name dcWCCOM
  -library_set IsCOM-1V
  -opcond_library stdcell_1V
  -opcond WCCOM
  -rc_corner rc-cworst
```

```
create_analysis_view
  -name missionSlow
  -delay_corner dcWCCOM
  -constraint_mode missionSetup
```

```
create_constraint_mode
  -name missionSetup
  -sdc_files[list io.sdc ...]
```

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 - MMMC Browser*, and double click on the name of the analysis view you want to edit.

> ⓘ You can use the update_analysis_view command or the Edit Analysis View form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

Setting Active Analysis Views

After creating analysis views, you must set which views the software should use for setup and hold analysis and optimization. These "active" views represent the different design variations that will be analyzed. Active views can be changed throughout the flow to utilize different subsets of views. Innovus applications can handle the views concurrently or sequentially, depending on their individual capabilities. Libraries and data are loaded into the system, as required to support the selected set of active views.
To set active analysis views, use the following command:

```
set_analysis_view
```

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

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

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

Guidelines for Setting Active Analysis Views

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

Changing the Default Active Analysis View

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

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

```
set_default_view -setup mission2Slow
```

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

Checking the Multi-Mode Multi-Corner Configuration

Use the following command to generate a hierarchical report of your current MMMC configuration:

```
report_analysis_views
```

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

You can also use the Innovus GUI to review the MMMC configuration:

- Select *File -> Import Design -> Create Analysis Configuration*

  or

- Select *Timing -> MMMC Browser*.

For more information, see the File Menu chapter in the *Innovus Menu Reference Guide*.

Changing How the MMMC Browser Displays Configuration Information

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

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

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

- Choose *File - Import Design*. Click the *Create Analysis Configuration* button under *Analysis*

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 - MMMC Browser*, 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.

In legacy environment, during `restoreDesign`, if the memory has:

- `init_mmmc_version 1`, you can use `saveDesign` to save a legacy DB, and you can also use `saveDesign -mmmc2` to save a Stylus DB . In this scenario, the tool does not support `eval_common_ui {read_db}` flow.
- `init_mmmc_version 2`. you can use `saveDesign` to save a Stylus DB only. In this scenario, the tool supports `eval_common_ui {write_db}` to save a Stylus design.

Updated SDC files for each mode are saved to the save directory, if ECO changes were made that affect pins that have constraint assertions.

Saving Designs

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

- Use the text command as follows:

  ```
  saveDesign sessionName
  ```

  or

- In the Innovus GUI, click the *Save Design* command on the File Menu and then click the *Innovus* option button in the *Save Design* form.

The design files you save depend on the work completed during an Innovus session.

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

Saving an OpenAccess Design

A Verilog based design that was loaded from *File - Import Design* can only be saved into OpenAccess if it uses OpenAccess reference libraries. It cannot be saved in OpenAccess if LEF files were used.

Transferring OpenAccess Data between Innovus and Virtuoso for ECO

- From an Innovus session, save the OpenAccess design.

  ```
  saveDesign sessionName -cellview {lib cell view}
  ```
- Exit the Innovus session.
- Open the OpenAccess database in Virtuoso XL (VXL) and edit the design.
  Note: You must use VXL rather than the Virtuoso Layout Editor.
- Save the design.
- Exit the VXL tool.
- Start an Innovus session.
- Restore the OpenAccess design.

  ```
  restoreDesign sessionName topCell -cellview {lib cell view}
  ```

Loading and Saving Design Data

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

Loading a Partition

To load a partition, you can use the *Load - Partition* command from the File Menu. Before you load a partition, perform the following tasks:

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

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

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

Loading Floorplan Data

To load floorplan data, use the following command from the *File* menu:

Load > Floorplan

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

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

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

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

Placement File Requirement

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

Loading an I/O Assignment File

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

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 Innovus software treats the following items as floorplan data:

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

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

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

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

Saving and Restoring Timing Graph

You can use the Timing Graph save and restore to get identical initial QOR upon restoring the DB using the standalone optimization and reporting commands. It provides CPU saving as you do not need to redo RC extraction, AAE DelayCal, and Timing Graph rebuild. You need to perform these only if there is a change in timer settings or extraction settings, which will lead to a different timing graph. With Timing Graph save and restore incremental optimizations, timing debug becomes faster.

You will get maximum benefit with Timing Graph save/restore in the following cases:

- Incremental optimizations preroute and postroute

For example:

- `place_opt_design -incremental`
- `place_opt_design -incremental_timing`
- `optDesign -postRoute -incr`

- Debugging the timing at various stages with save/restore
- Postroute and preroute setup and hold optimizations

You can trigger the flow using the following command:

```
saveDesign -timingGraph
```

- The `saveDesign -timingGraph` command currently saves RC, timing graph, and AAE DelayCal information. You do not need to specify the `-rc` option with `-timingGraph`.
  Note that -timingGraph is not on by default.
- After restoring the design, you can use any reporting command such as `report_timing`, `timeDesign`, or any optimization command such as `optDesign` without the need to do RC extraction, delay calculation, or timing computation.
- This applies to preroute and postroute, and only to the same Innovus build. If a different build is used then the Timing Graph will not load.
- Commands must not force timing recomputation when it is not needed.
  Examples:

1. Consider `timeDesign` in the following sequence of commands:
   a. `timeDesign -preCTS`
   b. `saveDesign -timingGraph preCtsOptDB.enc`
   c. `restoreDesign preCtsOptDB.enc`
   d. `timeDesign -preCTS`
      The `timeDesign -preCTS` command in step d will not rebuild Timing Graph. It will simply regenerate the reports and output a timing summary.

2. Consider `optDesign` in the following sequence of commands:
   a. `optDesign -preCTS`

b. `saveDesign -timingGraph preCtsOptDB.enc`

c. `restoreDesign preCtsOptDB.enc`

d. `optDesign -preCTS -incr`

   The `optDesign` command in step d does not rebuild Timing Graph to generate the Initial Summary. you should simply use the Timing Graph stored and continue with the optimization.

Converting an Innovus Database to GDSII Stream or OASIS Format

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

- For GDSII Stream format:
  - `setStreamOutMode`
  - `streamOut`

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

    Note: You can also use the following GUI forms:

    - *Mode Setup - StreamOut* from the Tools Menu.
    - *Save - GDS/OASIS* from the File Menu.
- For OASIS format:
  - `setStreamOutMode`
  - `streamOut`
  - **Note:** You can also use the following GUI forms:
    - *Mode Setup - OasisOut* from the Tools Menu.
    - *Save - GDS/OASIS* from the File Menu.

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

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

Related Topics

For more information, see Merging GDSII Stream or OASIS Files.

Creating Cells and Instances

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

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

Renaming LEF Vias

To force the `streamOut` command to give unique names to LEF vias, type the following command before running the `streamOut` command:

- `setStreamOutMode -SEvianames true`

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

`topSructureName``_VIA` *`index`*

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

For more information, see `setStreamOutMode` in the *Innovus Text Command Reference*.

Merging GDSII Stream or OASIS Files

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

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

Merging Files Using the Command Line

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

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

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

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

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

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

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

Merge Examples

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

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

Case 1

Most cases are similar to the following:

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

The design instantiates cells `X` and `Y`.

- For examples of cases where hierarchical cells are involved and the contents of a hierarchical cell is different from another cell with the same name, see **Case 2**.

Example 1

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

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

Example 2

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

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

Example 3

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

- `GDS1` processed: `X` and `COMMON` are copied from `GDS1`.
- `GDS2` processed: `Y` is copied from `GDS2`, `COMMON` is copied from `GDS2` but renamed `COMMON_GDS2` due to uniquification, reference from `Y` to `COMMON` is changed to `COMMON_GDS2`.

Example 4

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

- `GDS2` processed: `Y` and `COMMON` are copied from `GDS2`.
- `GDS1` processed: `X` is copied from `GDS2`, `COMMON` is copied from `GDS2` but renamed to `COMMON_GDS1` due to uniquification, reference from `X` to `COMMON` is changed to `COMMON_GDS1`.

Results

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

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

Case 2

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

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

The `Y` cells in the files contain different information.

The design instantiates cells `X` and `Y`.

Example 5

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

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

Example 6

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

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

Example 7

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

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

Example 8

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

- `GDS2` processed: `Y` from `GDS2` is copied from `GDS2`.
- `GDS1` processed: `X` is copied from `GDS1`, `Y` is copied from `GDS1` but renamed to `Y_GDS1` due to uniquification, reference from `X` to `Y` changed to `Y_GDS1`.

Results

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

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

For more information, see the following commands:

- `streamOut`

Merging GDS/OASIS Files Using the GUI

Use the GDS/OASIS Export form.

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

For more information, see *Save* - *GDS/OASIS* in the File Menu chapter of the *Innovus Menu Reference*.

About the GDSII Stream or OASIS Map File

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

Map Files

Flat Map File Format

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

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

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

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

```
#This comment is the first argument on a line#
METAL1    NET           1        0
METAL1    SPNET         999      0
  #This comment is preceded by white space#
METAL1    PIN           1000     0
     #This comment is preceded by a tab#
METAL1    LEFPIN        2000     0
METAL1    FILL          3000     0
METAL1    VIA           4000     0 #This comment is at the end of a line#
METAL1    VIAFILL       5000     0
METAL1    LEFOBS        10000    0
NAME      METAL1/NET    20000    0
```

Map File Columns

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

	MAXVOLTAGE	Adds max voltage label for each net according to its voltage. Labels will be used for checking of voltage-related DRC rules in VDR (Voltage Dependent Rules) flow. Sample syntax: `MAXVOLTAGE M1 123 1`
	MINVOLTAGE	Adds min voltage label for each net according to its voltage. Labels will be used for checking of voltage-related DRC rules in VDR (Voltage Dependent Rules) flow. Sample syntax: `MINVOLTAGE M1 456 1`
	NAME	Specifies a text label for the layer name and associated object type. If you do not want to output text labels, remove the `NAME` lines from the file. There is no limit on the length of a structure (cell) name. Because some GDS/OASIS readers have a 32-character limit, the Innovus software issues a warning message when a structure name is longer than 32 characters. If *`layerObjName`* is `NAME`, *`layerObjType`* can be a composite layer name`/`object type (`LEFPIN`, `NET`, `PIN`, or `SPNET`), or `COMP`. • `LEFPIN` places the label on the LEF `MACRO PIN` shape. (Applies only when the `-outPutMacros` parameter is specified. For more information, see `streamOut`.) • `NET` places the label on the `NET`. • `PIN` places the label on the `PIN` or I/O abstract pad. • `SPNET` places the label on the `SPECIALNET`. • `COMP` places the label on the placed DEF component.

layerObjType	Specifies an object type. You can specify a subtype for some *layerObjTypes*. For more information, see Specifying Object Subtypes.	
	ALL	• In routing layers, ALL is equivalent to NET, SPNET, VIA, PIN, LEFPIN, FILL, FILLOPC, LEFOBS, VIAFILL, and VIAFILLOPC. • In cut layers, ALL is equivalent to VIA, VIAFILL, and VIAFILLOPC.
	BLOCKAGE	Equivalent to DEF BLOCKAGES without + FILLS.
	BLOCKAGEFILL	Equivalent to DEF BLOCKAGES with + FILLS.
	CUSTOM	Applies to text labels created via the add_gui_text command. The add_gui_shape command also applies shapes on a CUSTOM layer.
	FILL	Equivalent to DEF FILLS without + OPC or DEF SPECIALNETS with + SHAPE FILLWIRE. You can separate FILL into floating and connected fill by specifying the FLOATINGsubtype. For more information, see "Fill Subtype" in the *Specifying Object Subtypes* section of this chapter.
	FILLOPC	Equivalent to DEF FILLS with + OPC or DEF SPECIALNETS + SHAPE FILLWIREOPC. You can separate FILLOPC into floating and connected fill by specifying the FLOATING subtype. For more information, see "Fill Subtype" in the *Specifying Object Subtypes* section of this chapter. **Note:** DEF 5.6 does not support this object type.

	LEFOBS	Equivalent to LEF OBS. (Applies only when the -outPutMacros parameter is specified. For more information, see streamOut.)
	LEFPIN	Equivalent to LEF PIN. (Applies only when the -outPutMacros parameter is specified. For more information, see streamOut.)
	NET	Equivalent to DEF NETS wiring. For more information, see "Net Name Subtype" in the *Specifying Object Subtypes* section of this chapter.
	PIN	Equivalent to DEF PINS.
	SHORT	Applies to trim-metal layer. It is used to control whether a short metal shape will be generated.
	SPNET	Equivalent to DEF SPECIALNETS without + SHAPE FILLWIRE or + SHAPE FILLWIREOPC. For more information, see "Net Name Subtype" in the *Specifying Object Subtypes* section of this chapter.
	TEXT	Applies to text labels created via the add_text command.
	TRIM	Indicates that this is the trim-metal.
	VIA	For via master creation for regular vias. For more information, see "Net Name Subtype" in the *Specifying Object Subtypes* 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 *Specifying Object Subtypes* section of this chapter.

	`VIAFILLOPC`	You can separate `VIAFILLOPC` into floating and connected fill by specifying the `FLOATING` subtype. For more information, see For more information, see "Fill Subtype" in the *Specifying Object Subtypes* section of this chapter. **Note:** DEF 5.6 does not support this object type."Fill Syntax"
`layerNumber`	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.	
`dataType`	Specifies the GDSII Stream/OASIS single data types, comma separated list of data types (for example, 1, 2, 4), or a range of data types (for example, 1-4). The data type must be an integer between 0 and 65535.	

See the DEF Syntax chapter in the *LEF/DEF Language Reference* for more information on the object types.

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

Specifying Object Subtypes

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

You can specify the following subtypes:

- Blockage name
  For more information, see "BLOCKAGE name Subtype".

- 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".

BLOCKAGE name Subtype

Use the following syntax to specify blockage name:

```
layerObjName layerObjType[:NAME:<name>] layerNumber dataType
```

:name is optional. It specifies blockage name. Use this syntax for BLOCKAGE and BLOCKAGEFILL shapes.

In the map file, different name blockage shapes can be output to a different `layerNumber/dataType`, 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 fill blockage named as critical_1 on *METAL1* is output to *`layerNumber 8 dataType 0`* and blockage named as *`normal_1`* to *`layerNumber 8 dataType 51`*, the map file would have the following statements:

```
METAL1 BLOCKAGEFILL:NAME:critical_1 8 0
METAL1 BLOCKAGE:NAME:normal_1 8 51
```

Fill Subtype

Use the following syntax to specify metal and via fill:

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

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

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

For example, to divide the output of metal fill shapes, so that non-floating fill on *METAL1* is output to `layerNumber 8 dataType 0` and floating fill to `layerNumber 8 dataType 51`, the map file would have the following statements:

```
METAL1 FILL 8 0
METAL1 FILL:FLOATING 8 51
```

To output the connected metal fill shapes on *METAL1* to `layerNumber 8 dataType 0` and floating fill to both `layerNumber 8 dataType 0` and to `layerNumber 8 dataType 51`, the map file would have the following statements:

```
METAL1 FILL 8 0
METAL1 FILL:FLOATING 8 0,51
```

Net Name Subtype

Innovus supports the following net name subtypes:

- `SPNET`

  Usage: `SPNET:NETNAME:$netname`

- `NET`

  Usage: `NET:NETNAME:$netname`

- `VIA`

  Usage: `VIA:NETNAME:$netname`

- `FILL`

  Usage: `FILL:NETNAME:$netname`

- `FILLOPC`

  Usage: `FILLOPC:NETNAME:$netname`

- `FILLDRC`

  Usage: `FILLDRC:NETNAME:$netname`

- `VIAFILL`

  Usage: `VIAFILL:NETNAME:$netname`

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
```

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:

```
Metal2 NET:clk1 32 10

Metal2 NET:clk1 132 0
```

Voltage Subtype

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

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

For example, to output nets on LEF layer *METAL1* with a minimum voltage of 1.8 to *layerNumber* 31 *dataType* 3, use the following syntax:

```
METAL1 NET:VOLTAGE:1.8 31 3
```

To output nets on LEF layer *METAL1* with a minimum voltage of 1.8 and a maximum voltage of 2.499 to *layerNumber* 31 *dataType* 3, use the following syntax:

```
METAL1 NET:VOLTAGE:1.8:2.499 31 3
```

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

```
METAL1 NET:VDD 31 4
METAL1 NET:VOLTAGE:1.8:2.499 31 1
```

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

```
METAL1 NET 31 0
METAL1 NET:VOLTAGE:1.8:2.499 31 0,1
```

SIZE Subtype

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

The map file syntax is as follows:

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

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

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

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

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

```
VIA12 VIA:SIZE:0.1X0.2 41 1
```

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

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

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

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

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

MASK Subtype

Use the following syntax to specify `MASK` attribute for designs using processes that require multiple masks per layer:

```
layerObjName layerObjType[:MASK:#] layerNumber dataType
```

Example

```
Met1 NET,SPNET,PIN 31 0 # collector for "gray/uncolored" data

Met1 NET,SPNET,PIN:MASK:1 31 1 # output for "MASK1" wiring shapes
```

```
Met1 NET,SPNET,PIN:MASK:2 31 2 # output for "MASK2" wiring shapes

Met1 VIA 31 0 # collector for "gray/uncolored" data

Met1 VIA:MASK:1 31 1 # output for "MASK1" via shapes

Met1 VIA:MASK:2 31 2 # output for "MASK2" via shapes

Via12 VIA 41 0 # no support for cut layer coloring in Phase 1

Met2 VIA 32 0 # collector for "gray/uncolored" data

Met2 VIA:MASK:1 32 1 # output for "MASK1" via shapes

Met2 VIA:MASK:2 32 2 # output for "MASK2" via shapes

...
```

…

SHAPE Subtype

You can use SHAPE to streamOut SPNET per shape. The SHAPE name supports:

- None
- RING
- STRIPE
- FOLLOWPIN
- IOWIRE
- COREWIRE
- BLOCKWIRE
- PADRING
- BLOCKRING

Note: Shape names are not case sensitive

The syntax for using SHAPE Subtype is:

```
layerObjName layerObjType:SHAPE:shape_name layerNumber dataType
```

Example

In the following example, all the SPNET objects (other than STRIPE) are streamed out to layer 31:0. STRIPE objects on SPNET are streamed out to both layers 31:0 and 31:1.

```
METAL1 SPNET 31 0
```

```
METAL1 SPNET:SHAPE:STRIPE 31 0,1
```

Note: From the 14.1 release, a cut layer in the map file (for a via) can have MASK, SIZE, and VOLTAGE at the same time. For example, you can specify the size of cuts and the mask attribute for design as follows:

```
layer VIA:SIZE:value1xvalue2:MASK:maskvalue layerNumber dataType
```

Example:

```
Via1 VIA:SIZE:0.24x0.24:MASK:2 196 2
```

Hierarchy Map File Format

Hierarchy map file format is supported by default in Innovus. This means that the tool can recognize the map file format itself, parse it automatically. You are not required to use any additional setting.

The `streamOut` command creates a hierarchy format map if `-mapFile` option is not used.

Rules

The rules for the hierarchy map file format are given below:

- File has version number
- Use # for comment
- Root hierarchy is layer or design, keywords is LAYER, DESIGN, and VERSION.
- Hierarchy structure, use indent and ‘-‘ char to indicate
- Attribute starts with ‘.’,
- AND keyword indicates the and logic
- Use ‘,’ to separate multiple objects, like layers or objects
- Uses “()” for range

Keywords

Keyword	**Meaning**
VERSION	File version number
LAYER	LEF layer or user-defined layer setting is under this hierarchy node
DESIGN	Design global setting is under this hierarchy node
LABEL	NET, SPNET, PIN, LEFPIN name text
NETNAME	Net name for NET, SPNET, VIA object type
FILLS	Routing BLOCKAGE with fills attribute setting
AND	Attributes are in and logic

Supported Attributes

Hierarchy map file format supports the following attribute types:

MASK	Mask shift setting on layer shape object
VOLTAGE	Voltage range on regular/special net and pin
SIZE	Via size
NETNAME	Net name for specified object about net special net and via
LABEL	Name label for net, special net, pin and LEF pin
NAME	Blockage name
MAXVOLTAGE	Max voltage text for pin
MINVOLTAGE	Min voltage text for pin
FILLS	Blockage attributes
FLOATING	Floating metal fill on _FILLS_RESERVED net

The following table lists the hierarchy map file formats objects and the supported attributes for each object:

OBJECT	**Sub Object support**	**Description**
NET	MASK, VOLTAGE, NETNAME, LABEL, MAXVOLTAGE, MINVOLTAGE	Regular wires on net
SPNET	MASK, VOLTAGE, NETNAME, LABEL, MAXVOLTAGE, MINVOLTAGE	Special wires on special net
PIN	MASK, VOLTAGE, LABEL	Top cell design pin and physical pin
LEFPIN	MASK, LABEL	Cell pin
VIA	MASK, VOLTAGE, SIZE, NETNAME	Via cell top/cut/bottom layer shape
FILL	MASK, FLOATING	Metal fill special wire on special net
FILLOPC	MASK, FLOATING	Metal fill special wire with OPC attribute on special net
FILLDRC	MASK, FLOATING	Metal fill special wire with DRC attribute on special net
VIAFILL	MASK, FLOATING, SIZE	Metal fill special via on special net
VIAFILLOPC	MASK, FLOATING, SIZE	Metal fill special via with OPC attribute on special net
VIAFILLDRC	MASK, FLOATING, SIZE	Metal fill special via with DRC attribute on special net
SPENTNOFILLDRC		Special wires on special net without DRC fill
TRIM	MASK	Ttrimmetal on regular and special net patch wire

SHORT	MASK	Trimmetal short on regular and special net patch wire
BLOCKAGE	MASK, FILLS, NAME	Routing blockage
CUSTOM		GUI object
TEXT		OA text object
COMP	LABEL	Instance box, instance name
DIEAREA		Design die area box
LEFPVERLAP		Cell LEF overlap layer box

Example of Hierarchy Syntax Format

The following example of hierarchy syntax format shows all type combinations:

```
# layer map file in hierarchy format
VERSION 1.0


# FE layer setting
LAYER METAL1,METAL2
  - NET 100 0
     - .MASK 1  100 1
     - .MASK 2  100 2
      - .FIXEDMASK 100 3
      - .VOLGATE (0.001 3.338) 100 4
     - .NETNAME spi_clk 100 5
     - .LABEL 110 6
  - SPNET 110 0
      - .MASK 1  110 1
      - .MASK 2  110 2

      - .VOLGATE (0.001 3.338) 110 3

      - .NETNAME VSS,VDD 110 3

      - .LABEL 110 12

  - PIN,LEFPIN 10 0

      - .MASK 1 10 1

      - .MASK 2 10 2

  - PIN

      - .VOLTAGE (0.001 3.339) 10 3

  - VIA 120 0

     - .MASK 1  120 1

     - .MASK 2  120 2

     - .VOLTAGE (0.003 3.337) 120 3

      - .SIZE 24x24 120 4

     - .NETNAME VSS,VDD 120 5

  - BLOCKAGE 200 0

     - .FILLS 200 1

     - .NAME bkg_tx 200 3


LAYER METAL3

  - NET 130 0

     - .MASK 1 130 1

     - .MASK 1 AND .VOLTAGE (1.5 3.33) 130 100

     - .MASK 1 AND .VOLTAGE (1.5 3.33) AND .NAME spi_clk 130 110

     - .MAXVOLTAGE 130 120

     - .MINVOLTAGE 130 130


LAYER METAL4
```

```
  - FILL,FILLOPC,FILLDRC 400 0

     - .FLOATING 400 1

  - VIAFILL,VIAFILLOPC,VIAFILLDRC 500 0

     - .FLOATING 500 0


# customer layer setting

LAYER CUST_M3

  - CUSTOM 200 0


# text setting

LAYER text

  - TEXT 300 0


# design setting

DESIGN

  - DIEAREA 140 0

  - COMP 150 0

     - .LABEL   150 1

            - LEFOVERLAP 160 0
```

The following example shows the syntax format map file with contents,

```
###########################################################

# file version number                                     #

###########################################################

Version: 1.0


# For trimmetal object
```

```
LAYER CUSTM1
  - TRIM 1 0


LAYER CUTM2,CUTM3
  - TRIM 2 0


LAYER CUTM3
- TRIM 3 0


# For FE layer object
LAYER M0
  - NET 4 0
     - .LABEL 14 0
  - SPNET 5 0
     - .LABEL 15 0
  - PIN 6 0
     - .LABEL 16 0
  - LEFPIN 7 0
     - .LABEL 17 0
  - FILL 8 0
  - FILLOPC 9 0
  - VIA 10 0
  - VIAFILL 11 0
  - VIAFILLOPC 12 0
  - LEFOBS 13 0


LAYER V0
  - PIN 18 0
  - LEFPIN 19 0

  - FILL 20 0

  - FILLOPC 21 0

  - VIA 22 0

  - VIAFILL 23 0

  - VIAFILLOPC 24 0


LAYER M1

  - NET 25 0

     - .LABEL 35 0

  - SPNET 26 0

     - .LABEL 36 0

  - PIN 27 0

     - .LABEL 37 0

  - LEFPIN 28 0

     - .LABEL 38 0

  - FILL 29 0

  - FILLOPC 30 0

  - VIA 31 0

  - VIAFILL 32 0

  - VIAFILLOPC 33 0

  - LEFOBS 34 0


LAYER V1

  - PIN 39 0

  - LEFPIN 40 0

  - FILL 41 0

  - FILLOPC 42 0
  - VIA 43 0

  - VIAFILL 44 0

  - VIAFILLOPC 45 0


LAYER M1

  - NET 46 0

     - .LABEL 56 0

  - SPNET 47 0

     - .LABEL 57 0

  - PIN 48 0

     - .LABEL 58 0

  - LEFPIN 49 0

     - .LABEL 59 0

  - FILL 50 0

  - FILLOPC 51 0

  - VIA 52 0

  - VIAFILL 53 0

  - VIAFILLOPC 54 0

  - LEFOBS 55 0


# For design setting

DESIGN

  - COMP 313 0

     - .LABEL 312 0

  - DIEAREA 314 0

  - LEFOVERLAP 315 0
```

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 Innovus Session

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

Type	Replace	Update	How
ILM	Y	Y	`specifyIlm` `unspecifyIlm`
LEF	N	Y	`loadLefFile -incremental`
Quantus Tech File	Y	N	`update_rc_corner -qx_tech_file`
Timing Libraries	N	Y	`update_library_set`

Timing Constraints	Y	Y	`update_constraint_mode`
			`set_interactive_constraint_modes`
			`source`

I/O Assignment File	Y	N	`loadIoFile`
Partition File	Y	N	`specifyPartition`
Floorplan File	Y	N	`loadFPlan`
Routing File	Y	N	`restoreRoute`
Special Route File	Y	Y	Use `loadSpecialRoute` to replace
DEF	Y	Y	`defIn` (use the `-scanChain` option to update scan chains)
PDEF	Y	Y	`pdefIn`

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

SKILL to TCL Mapping

The following table shows the mapping of Virtuoso SKILL functions to Innovus TCL functions while using the `setOaxMode` `-bindkeyFile` parameter.

Virtuoso Key (Default)	**SKILL Function**	**Innovus Key (Default)**	**Innovus Command**
`Shift-k`	`leHiClearRuler()`	`K`	`cleanRuler`
`Shift-m`	`leHiMerge()`	`M`	`mergeWire`
`Shift-q`	leEditDesignProperties()	`Q`	`summaryReport`
`Shift-r`	`leHiReShape()`	`R`	`resizeMode`
`Shift-s`	`leHiSearch()`	`S`	`getWireInfo`
`Shift-u`	`hiRedo()`	`U`	`redo`

`Shift`
`<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

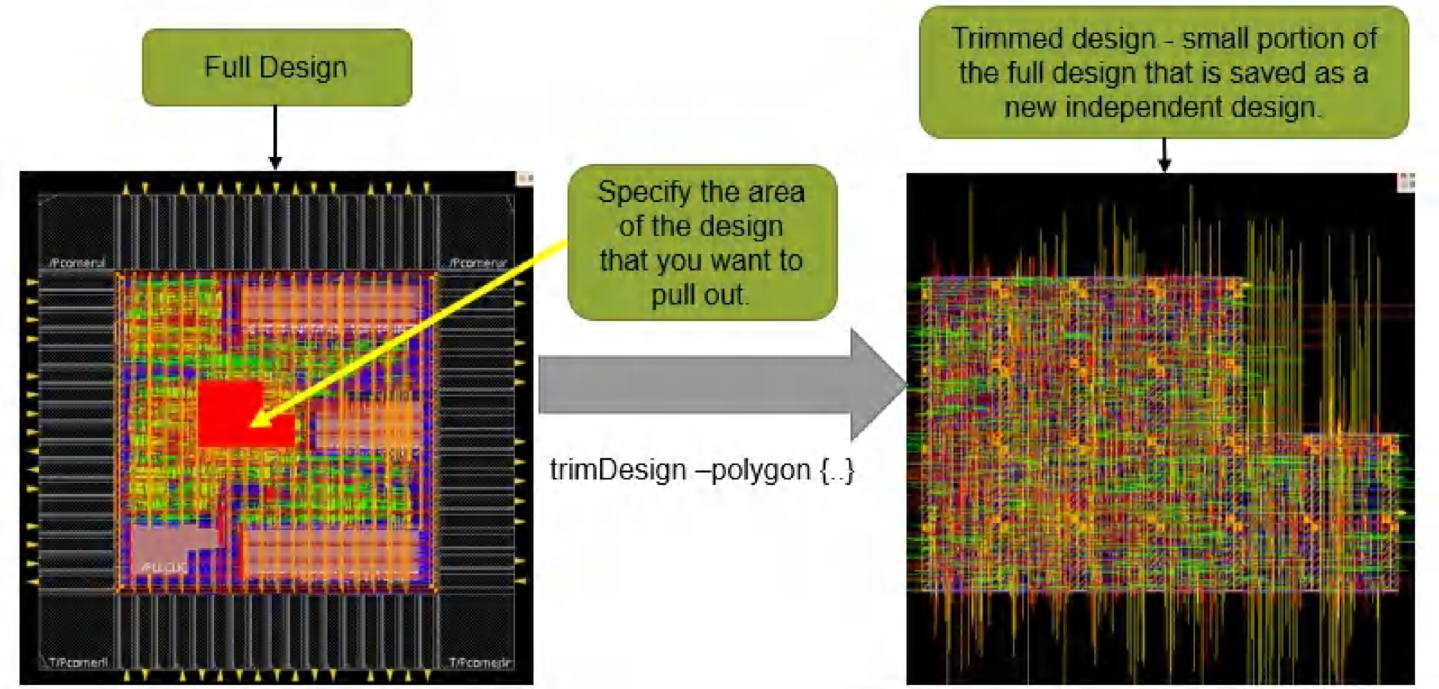
`Ctrl-d`	`geDeselectAllFig()`	`Ctrl-d`	`deselectAll`
`Ctrl-n`	`leSetFormSnapMode (\\\"90XFirst\\\")`	`Ctrl-n`	`snapFloorplan`
`Ctrl-r`	`hiRedraw()`	`Ctrl-r`	`redraw`
`Ctrl-s`	`leHiSplit()`	`Ctrl-s`	`splitWire`

Note: If the `setOaxMode -bindkeyFile` parameter is used, then the Virtuoso Key column applies to Innovus for all of the equivalent commands in the mapping.

Trimming the Design

Sometimes instead of sharing the whole design (containing third-party IP data), you may want to only share a small portion of the design and avoid sharing the whole third-party IP information. This is especially helpful in a scenario where you want to analyze a part of the design to investigate a localized issue or try out a different setting. The small trimmed version of the design can then be treated as a small independent design for designers to work on.

With the `trimDesign` command, you can create a small trimmed down portion of a design. This trimmed version of the design is an independent design that can be used for investigation and analysis. The `trimDesign` command enables you to specify the coordinates of a box to create a small portion of the design. While the trimmed design completely preserves the physical data, it does not preserve the timing and the power domain data. Working on a small portion of the design offers a faster turnaround time and a lesser memory footprint.


Full Design
Specify the area of the design that you want to pull out.
Trimmed design - small portion of the full design that is saved as a new independent design.
trimDesign –polygon {..}
/Pcornerul
/Pcornerur
/PLLCLK
T/Pcornerll
T/Pcornerlr

Related Information

- Advantages of Working on a Trimmed Design
- Use Model of Working on Trimmed Designs
- How Design Objects are Handled in the Trimmed Design
- Encrypting the Names of Instances and Nets

Advantages of Working on a Trimmed Design

The advantages of working on a trimmed design are:

- Helps you to focus on a smaller database.
- The trimmed design inherits the physical design data elements like regular/special wires, instances, macros, rows, sites, tracks, colors, routing/placement blockages, attributes, properties of the original full design. This ensures a design environment similar to the original full design and similar behavior by the tool.
- All the violations (`checkPlace` and `verify_drc`) present in the original design are inherited in the trimmed version of the design. This makes the `trimDesign` capability a useful feature for debugging smaller sections of the design, thereby boosting quality.
- The names of design objects are encrypted in the trimmed design to maintain data confidentiality. The `trimDesign` command enables you to customize the encryption procedure that is used to achieve this. The `trimDesign` process does not save any binary data, instead it saves the DEF, Verilog, LEF information in a .txt file format.

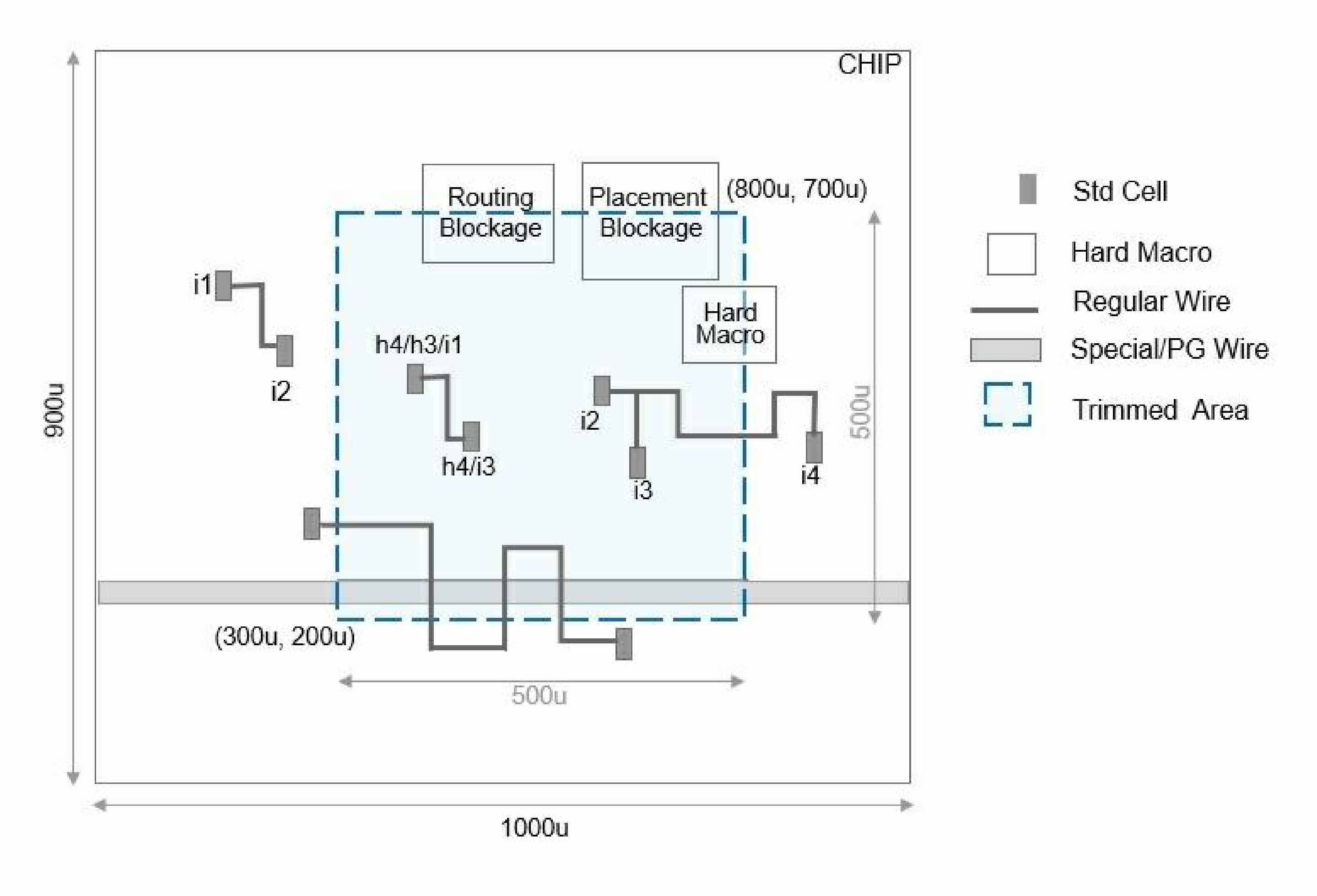
Related Information

- Trimming the Design
- Use Model of Working on Trimmed Designs
- How Design Objects are Handled in the Trimmed Design
- Encrypting the Names of Instances and Nets

Use Model of Working on Trimmed Designs

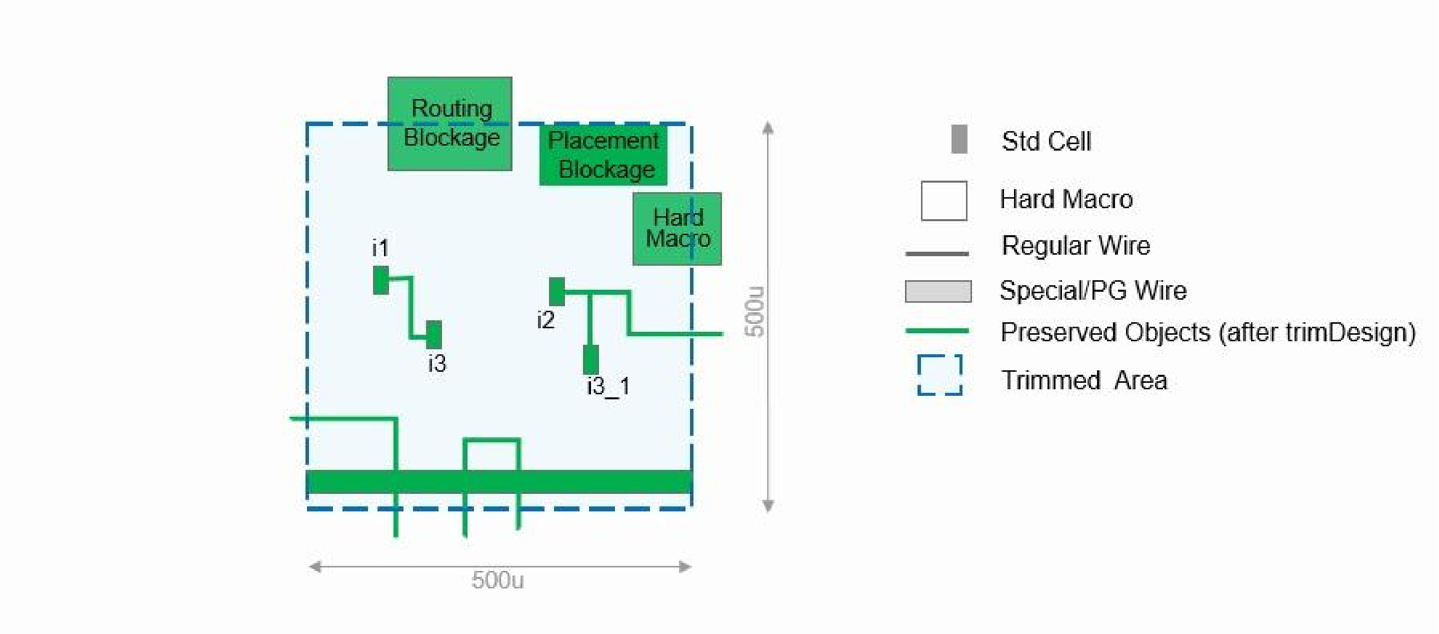
With `trimDesign`, you can create a design which is physically equivalent to a small portion of a big design. It will preserve all physical data so much so that `checkPlace` and `verify_drc` will yield similar results on this small design, versus the same area in the big design. Thus, you can then share this small design independently for any investigation related to checking the placement and DRC violations.

Note: The LEF technology file must be shared with the trimmed design as it is required for checking the placement and verifying the drc.



In the following example, a trimmed design, `trim_db`, is created with the specified coordinates and is saved in the DB file:

```
trimDesign -dir DB -name trim_db -area{300.0 200.0 800.0 700.0}
```



With `trimDesign`, you can create a design which is physically equivalent to a small portion of a big design. It will preserve all physical data so much so that `checkPlace` and `verify_drc` will yield similar results on this small design, versus the same area in the big design. Thus, you can then share this small design independently for any investigation related to checking the placement and DRC violations.

To use a selected routing blockage as the trim area, use the following command:

```
trimDesign -selectedRouteBlk
```

The `trimDesign` command does not include any timing related data (viewDefinition.tcl, SDC, power domains), however, it does include the physical data (.fp, .place, .route). Since timing and power data is not included, the trimmed design is better suited for physical data analysis rather than optimization, timing, power analysis.

- The `checkPlace` command reports all violations that were there on instances in the trim box/area in the original full design. However, since hierarchical fences are not brought back, the trimmed design may not have the Region/Fence and Not-of-Fence Violations.
- The `verify_drc` command reports all violations that were there on nets in the trim box/area in the original full design. However, since only the net wires crossing box boundary are preserved, the trimmed design may have some extra violations.

Related Information

- Trimming the Design
- Advantages of Working on a Trimmed Design
- How Design Objects are Handled in the Trimmed Design
- Encrypting the Names of Instances and Nets

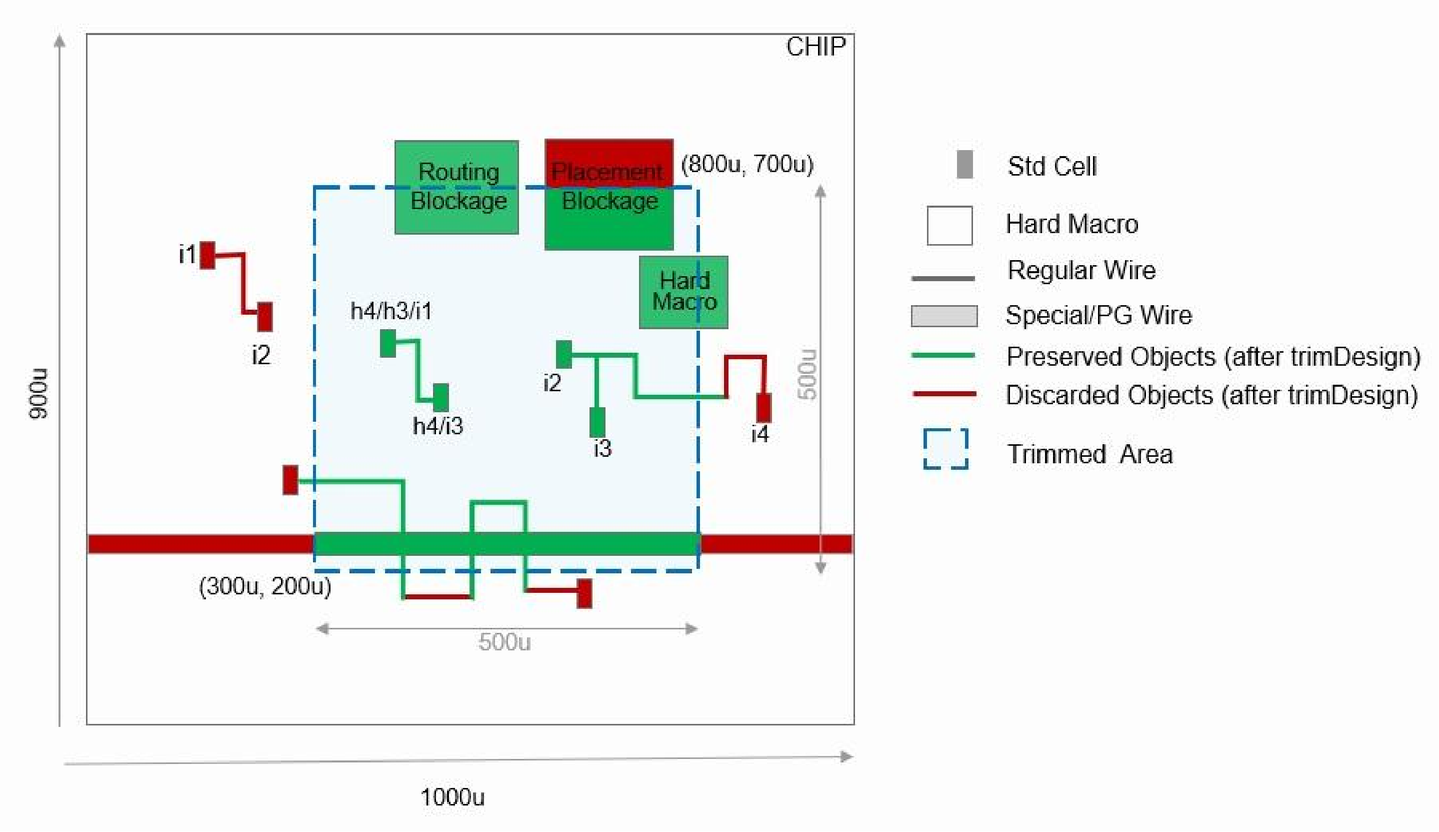
How Design Objects are Handled in the Trimmed Design

With the `trimDesign` command you can specify the coordinates of a box to create a small portion of the design as an independent design. This trimmed down version of the design can be then used for investigation and analysis. The dimensions of the trimmed design are the same as the dimensions of the trim box/area specified with the `trimDesign` command.

For example, the following command will create a trimmed area of 500uX500u:

```
trimDesign -dir DB -name trim_db -area {300.0 200.0 800.0 700.0}
```

Note: During `trimDesign` the specified trim box/area is snapped to the instance grid. All instances in the box lose their original logic hierarchy and appear flat in the new db. This process is irreversible, and it is used only for debugging purposes. You should not attempt to assemble back the trimmed design to the original full design.



The trimmed design:

- Retains the attributes and properties of all the standard cells and macros that are completely inside the trim box/area. All hard macros that overlap with the trim box/area are also preserved. It does not include any standard cells or macros that are completely outside the specified trim box/area.
- Preserves all overlapping rows, sites, and colors.
- Flattens all instances and net names. All instances in the trim box/areal lose their original logic hierarchy and appear 'flat' in the trimmed database. Any naming conflicts arising due to flattening are automatically resolved. For example, h1/h2/i1 in the full design will appear as i1 in the trimmed design. Also, h1/h4/i1 may appear as i1_uniquify1, to distinguish it from i1.
  Note: To resolve naming conflicts due to the loss of hierarchy, additional name uniquification is done. A mapping of original nets/instances names vs their trimmed design modified names, is written in a map file for book keeping purpose.
- Maintains the design parameters such as instance location/orientation, regular/special wire and its connectivity relative to the original full design.
- Retains the attributes and properties of all the nets having some instances inside the trim box/area. It also preserves (pushes down) the nets that are not connected to any instance in

the trim box/area, but have some wires overlapping with trim box/area. Nets having all instances and wires outside of trim box/area are discarded.

- Creates ports for all wires crossing the boundary of the trimmed design.
- Preserves the tracks, attributes and colors of all wires that are completely inside the trim box/area. All wires that overlap with the trim box/area are also preserved. It does not include any wire that is completely outside the specified trim box/area.
- Preserves the Power/Ground/Special wires that overlap with the boundary of the trim box by cutting them against the boundary. It creates additional PG ports at cross points of these wires.
- Discards fences, regions, guides, instance groups, and power domains.
- Saves the `.globals` file(s) that have the .lef list copied from original full design’s .globals file(s).

Related Information

- Trimming the Design
- Advantages of Working on a Trimmed Design
- Use Model of Working on Trimmed Designs
- Encrypting the Names of Instances and Nets

Encrypting the Names of Instances and Nets

The names of design objects are encrypted in the trimmed design to maintain data confidentiality. Using the `-encryptName` parameter of the `trimDesign` command you can specify the procedure that is used to encrypt the names of instances and nets in the trimmed design. This feature is useful to maintain data confidentiality.

Note: The `trimDesign` process does not save any binary data, instead it saves the DEF, Verilog, LEF information in a .txt file format.

For example, the following command uses a *get_new_name* procedure to encrypt the names on the instances and nets in the trimmed design *trim_db*:

```
trimDesign -dir DB -name trim_db -area {300.0 200.0 800.0 700.0} -encryptName
gen_new_name
```

The following is format of the encryption procedure that is used by the `trimDesign`.

You can customize this procedure:

```
proc gen_new_name {name} {

  global n

  if {![info exists n]} {

    set n 0

  }

  set new_name en_name__$n

  incr n

  return $new_name

}
```

Related Information

- Trimming the Design
- Advantages of Working on a Trimmed Design
- Use Model of Working on Trimmed Designs
- How Design Objects are Handled in the Trimmed Design

4

Design Planning Capabilities

Floorplanning the Design

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 early global routing, can be an iterative design process.

The Innovus Implementation System (Innovus) software provides a rich set of commands and GUI functions to floorplan your design interactively. There are also commands for creating an initial floorplan automatically, or, resize a finished floorplan while keeping relative placement of objects.

- For information on floorplan commands, see the *Floorplan Commands* chapter, in the *Innovus Text Command Reference*.
- For information on floorplan GUI, see the *Floorplan Menu* chapter, in the *Menu Reference*.

Innovus includes several keyboard shortcuts for use with the floorplanning feature. Make sure you type the bindkey while the main Innovus window is active and the cursor is in the design display area. The *Binding Key* form contains a complete list of bindkeys. To display this form, select *View - Set Preference* from the Innovus menu, then click the *Binding Key* button on the *Design* tab of the *Preferences* form, or use the default `b` binding key.

Common Floorplanning Sequence

Floorplanning usually starts by preplacing blocks, modules, and submodules according to the prepared floorplan. All other modules or blocks not in the prepared floorplan are left outside the chip area. The following steps describe the most common sequence for floorplanning:

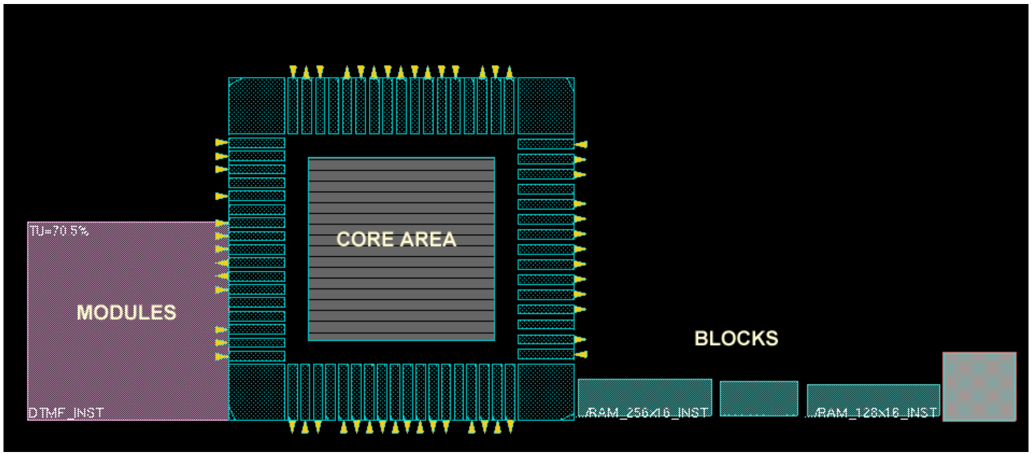
1. Importing the design. For more information, see Importing and Exporting Designs.
2. Studying the design's connectivity.
3. Performing the minimum amount of floorplanning based on the chip design floorplan, or do no floorplanning at all.
4. In some cases, no floorplanning is required. For example, 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 Early Global Route to view placement and routing congestion.

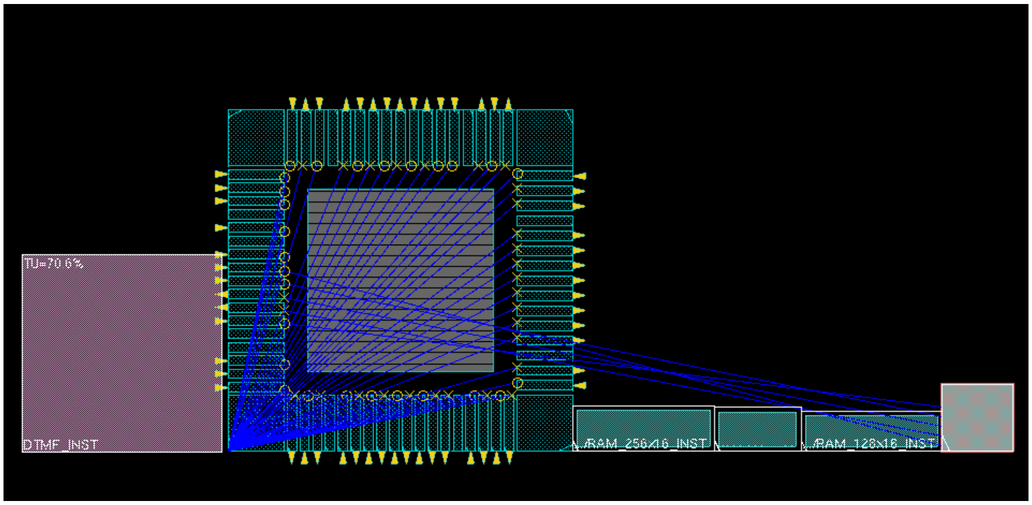
   Optionally, running the `resizeFloorplan` command to enlarge or shrink the die after placement and routing.
6. In this case, floorplanning is done to detail the pre-placement of all blocks, most likely done by a back-end designer to gauge the feasibility of a prepared floorplan.
7. The placer places all remaining blocks that were not preplaced in the floorplan, and also recognizes the floorplan object, such as power and ground routes.
8. If you are at the design's top-level in the display area and want to generate a guide for a submodule, ungroup the top module until you have reached the submodule.
9. Using the full chip placement to refine block (hard macro and blackbox) locations. (Optional) Based on the full chip placement results - placement density and routing congestion, running resize floorplan to enlarge or shrink the die.
10. View the placements of blocks to determine if you need to change the alignment or orientations.
11. Looking for congestion in modules and change heavily congested modules' placement density to a lower percentage.
12. (Optional) If you made any changes in step 5, or especially step 6, rerun placement.

Viewing the Floorplan

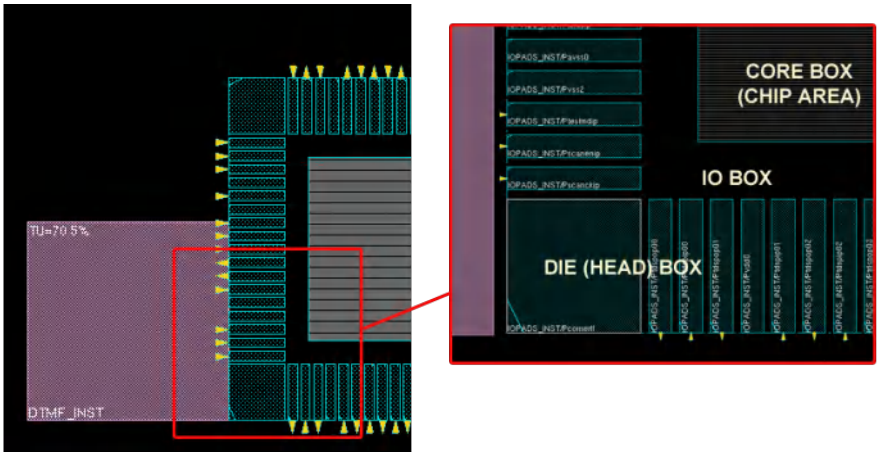
In the design display area, the objects to the left of the core area are the top-level modules, which can be moved and reshaped. The objects to the right of the chip area are the blocks, which can be moved but not reshaped.



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 *View - Set Preference* from the Innovus menu, then click the *Flightline* tab on the *Preferences* form in the GUI. You can change the die or core size; the margins between the core box and I/O pad instances; and the individual die (head), I/O, or core box sizes. These boxes are shown in the following figure.



You can move module or instance groups outside the core area.

Note: Descendant macros and standard cells can move with their ancestor modules when the module is moved in and out of the core area.

Module Constraint Types

The entire design size is initially calculated during design import, and each module size is calculated. The size of the modules is determined by either the core utilization or the core width and height specifications. The imported design modules can have one of the following constraint types:

None

The module is not pre-placed in the core design area. The contents of the module are placed without any constraints.

Guide

The module is preplaced in the core design area. A module guide represents the logical module structure of the netlist. The purpose of a module guide is to guide placement to place the cells of the module in the vicinity of the guide's location. The preplaced guide is a soft constraint, which is discussed later in this section. After the design is imported, but before floorplanning, you can locate module guides on the left side of the core area, which appear as pink objects (by default) in the Floorplan view.
When a module is preplaced in the core design area, it snaps to a standard cell row in the vertical direction and to a metal 2 pitch in the horizontal direction (the default). This default can be changed to snap to the manufacture grid (in the *Preferences* form's *Floorplan* page).

Fence

The module is a hard constraint in the core design area. After specifying a hierarchical instance as a partition, the constraint type status of a module guide is automatically changed to a fence. The physical outline of a fence module is rigid, and the design for the module is self-contained within the rigid outline. Only child instances must be contained within the partition physical outline; non-child blocks or modules that do not belong to the partition are excluded, and should not be pre-placed within another partition. This restriction is a hard restriction for third party back-end tools where the placement file for a partition does not match the partition netlist.

To create a fence for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the `createFence` command or select *Fence* from the Attribute Editor's *Constraint Type* pulldown menu.

Note: Fence groups can potentially cause overlaps that cannot be corrected because the Innovus software cannot move the cells out of the group.

Region

This constraint is the same as a fence constraint except that instances from other modules can be placed within its physical outline by placement. A module guide is changed to a status of Region when preplaced in the core design area.

To create a region for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the `createRegion` command or select *Region* from the Attribute Editor's *Constraint Type* pulldown menu.

Note: Region groups can potentially cause overlaps that cannot be corrected because the Innovus software cannot move the cells out of the group.

Soft Guide/Cluster

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, use the `createSoftGuide` command or select *SoftGuide* from the Attribute Editor's *Constraint Type* pulldown menu.

Note: Soft guides are also called clusters.

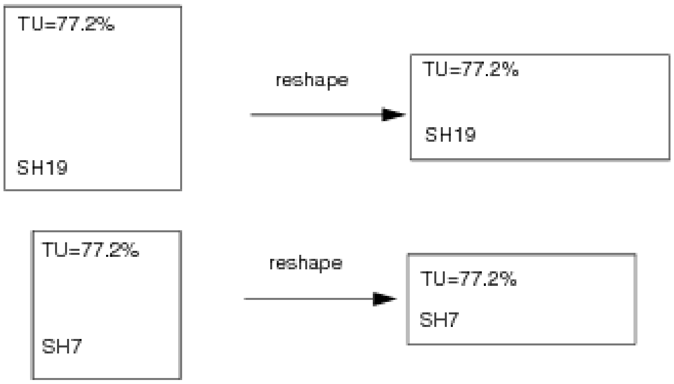
Note: You can use the `setPlaceMode -place_global_soft_guide_strength` command to instruct the placement engine to place those modules which have been specified as soft guide in the floorplan file closer. The degree of closeness depends on the strength given. In case, there are fixed/preplaced instances, the module placed gets affected by those fixed instances and therefore placed close to them.

Note: Now the module constraints, guide, fence, region, and soft guide, are allowed to be out of core area but must be in the die area.

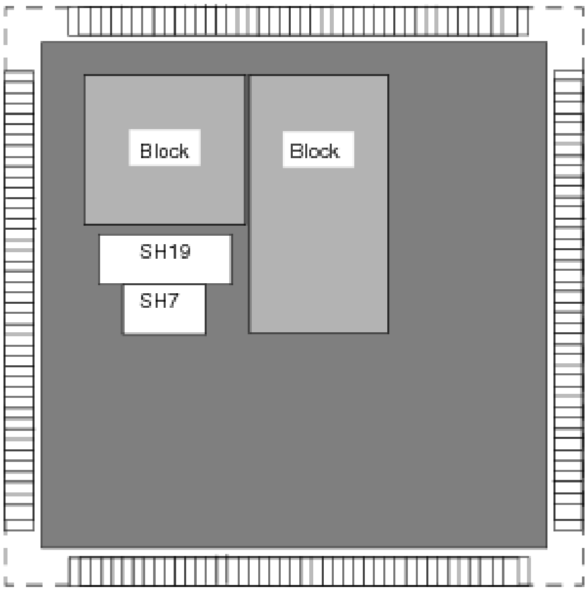
Target Utilization Display

Module constraints display a target utilization (TU=%) value to represent their physical design size. This is an estimation of module utilization for the given size of the module where only standard cell and hard macro areas are considered; floorplan constraints, such as placement blockages, are not considered. This value is calculated by the standard cells area plus the hard macros area, divided by the module area. The initial TU values are calculated during design import.

The TU percentage helps judge the physical size of a module guide to customize the shape of the module in the floorplan. For example, modules `SH19` and `SH7` have a TU values of 77.2%. If the modules are reshaped with the same area, they retain their TU values, as shown in the following figure:



You can place them in the core area so they are preplaced close to one another, as shown in the following figure:



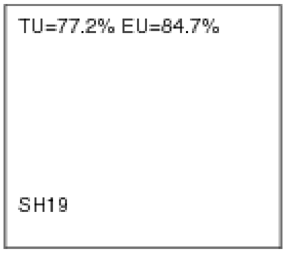
The position of the module guide is a placement constraint, and the final definition of the module is determined by several factors. The most important factor – the highest priority of constraint – is the connectivity between itself and other modules. Other floorplan constraints, such as neighboring preplaced module guides, preplaced blocks, placement blockages, and routing blockages, are also considered, but at a lower priority than connectivity.

Note: You can use a stronger constraint for keeping modules `SH19` and `SH7` close together using the Group Instances form, and even a stronger constraint by saving the regrouped netlist.

Unlike module guides, the position of fences and regions is a hard placement constraint and are not moved by the same factors.

Effective Utilization Display

For fences and regions, you can display the effective utilization (EU=%) value. The EU value takes into account the actual cells and hard macros in the fence or region, placement or routing blockages, partition cuts, and other floorplan constraints. It is a good practice to update the EU value before running placement. Click the *Query Area Density* 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.


TU=77.2% EU=84.7%

Note: The displayed EU values are not automatically updated. You must click the *Query Area Density* toolbar widget each time you want to display the updated EU value. This calculation could be time consuming, especially for larger designs.

Note: If the EU value is at or exceeds 100% for a fence or region, placement changes the fence or region to a guide. To avoid this, before you run placement, make sure to check and update the EU value, if necessary.

Calculating Density

When specifying the floorplan, you can determine the core and module sizes by total density or standard cell density using the *Core Utilization* or *Cell Utilization* options, respectively, in the *Specify Floorplan* form. Core Utilization determines the initial size of the core area and the initial size of the pink module guides off to the left of the die area. The total density is calculated as follows:

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

In determining the size of the core area and module guides, standard cells and hard macros are treated the same. However, you can determine how densely objects can be packed by weighing the standard cell density separately from the hard macro density. The standard cell density is calculated as follows:

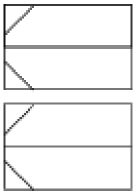
```
Core Size = (standard cell area / cell utilization) + macro area + halo
```

The size of the core is smaller once you specified your floorplan by using *Cell Utilization*.

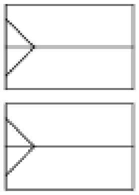
Standard Row Spacing

To configure the rows, use the `setFPlanRowSpacingAndType` command, the `createRow` command or the options from the *Standard Cells Rows* panel of the *Specify Floorplan* form.

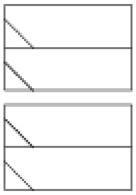
The following row configurations are supported:



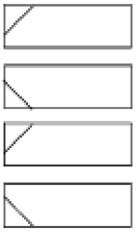
Bottom R0 and flip/abut



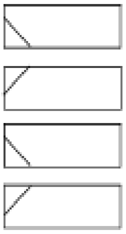
Bottom MX and flip/abut



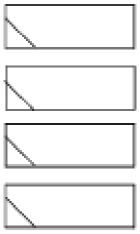
Bottom R0 and abut



Bottom R0 and flip



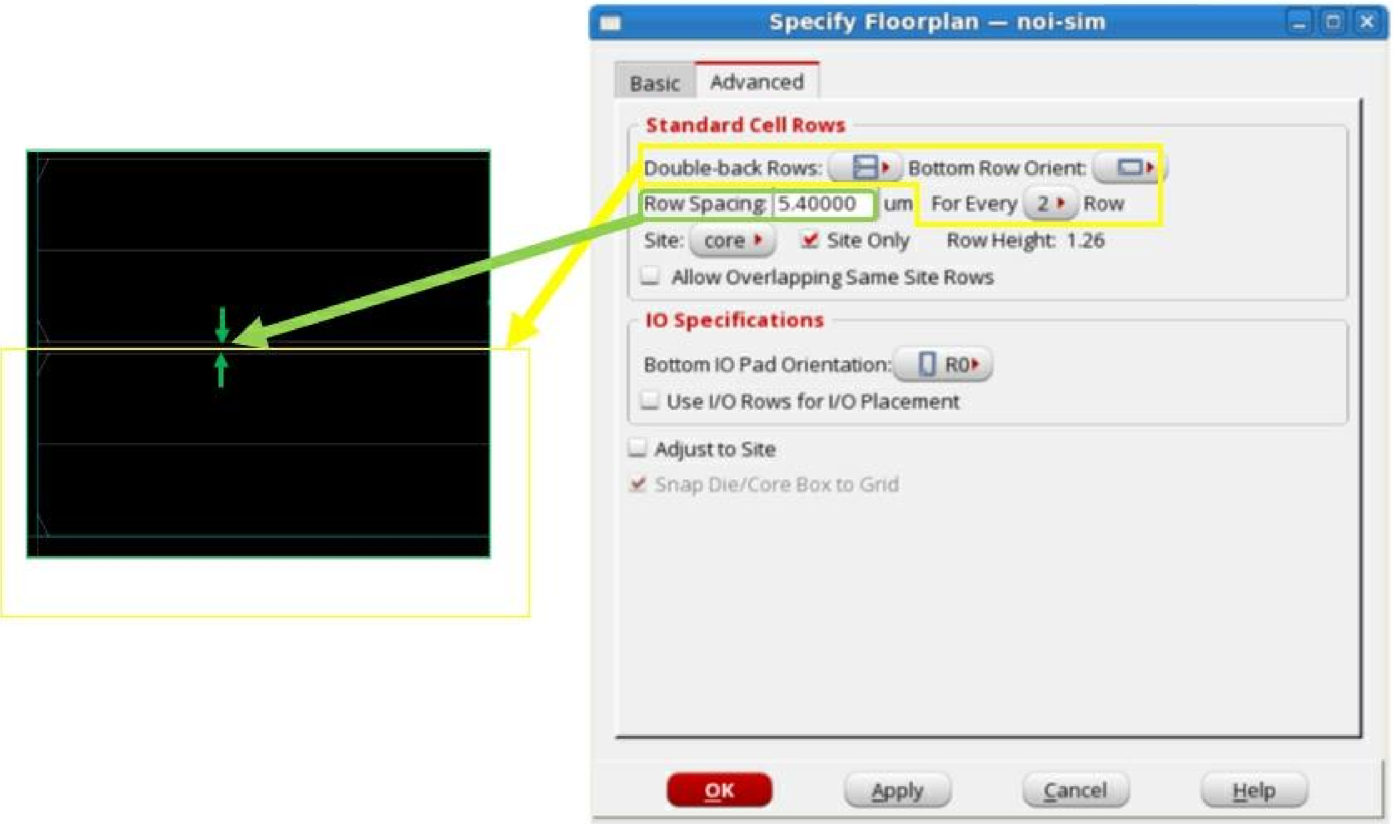
Bottom MX and flip



Bottom R0

Example 1: Bottom R0 and flip/abut

```
createRow -limitInCore -site CORE -spacing 5.4
```



Standard Cell Rows

Double-back Rows: Bottom Row Orient:

Row Spacing: 5.40000 um For Every 2 Row

Site: core ☑ Site Only Row Height: 1.26

□ Allow Overlapping Same Site Rows

IO Specifications

Bottom IO Pad Orientation: R0

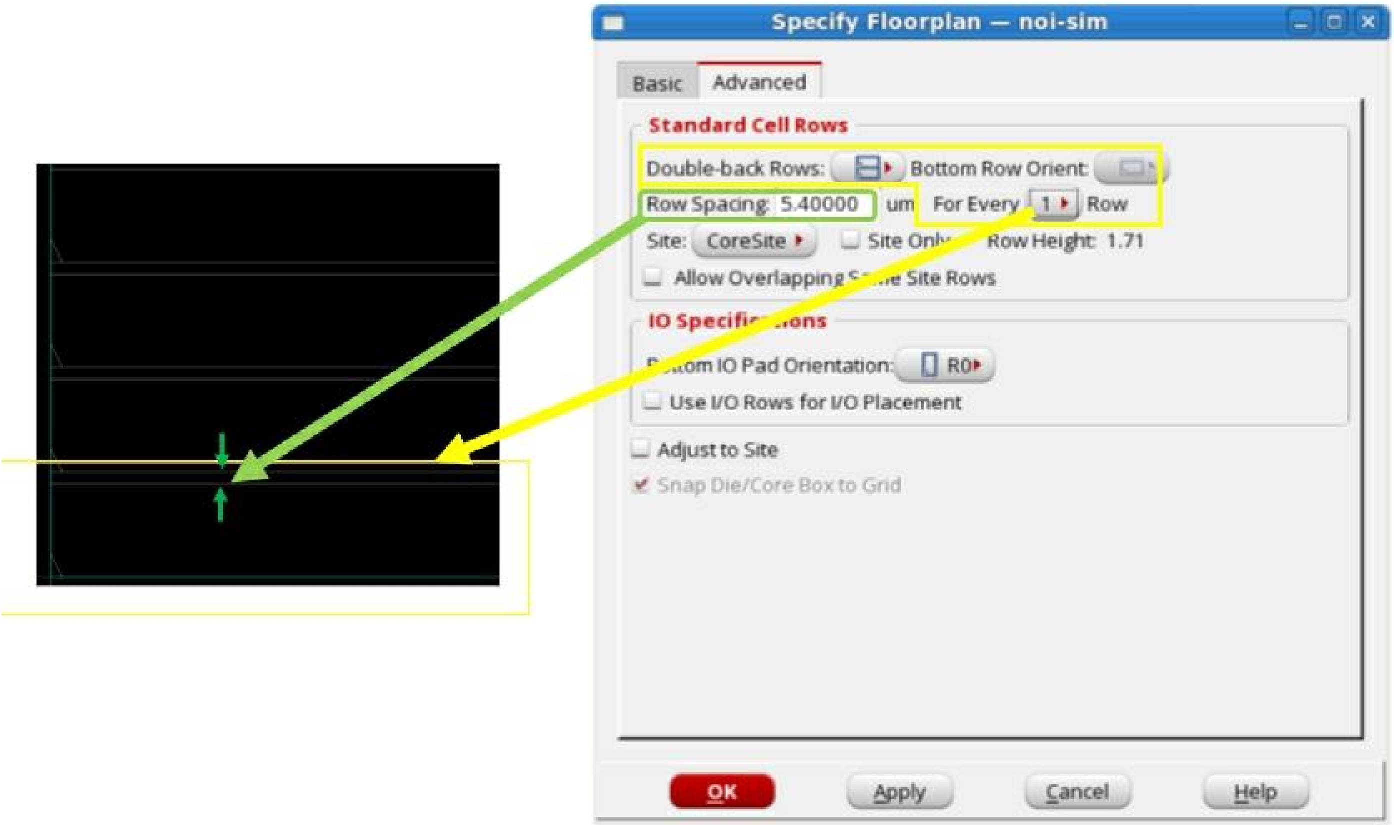
□ Use I/O Rows for I/O Placement

□ Adjust to Site

☑ Snap Die/Core Box to Grid

Example 2: Bottom R0

```
createRow -limitInCore -noAbut -noFlip -site CORE -spacing 5.4
```



Standard Cell Rows

Double-back Rows: Bottom Row Orient:

Row Spacing: 5.40000 um For Every 1 Row

Site: CoreSite □ Site Only Row Height: 1.71

□ Allow Overlapping Same Site Rows

IO Specifications

Bottom IO Pad Orientation: R0

□ Use I/O Rows for I/O Placement

□ Adjust to Site

☑ Snap Die/Core Box to Grid

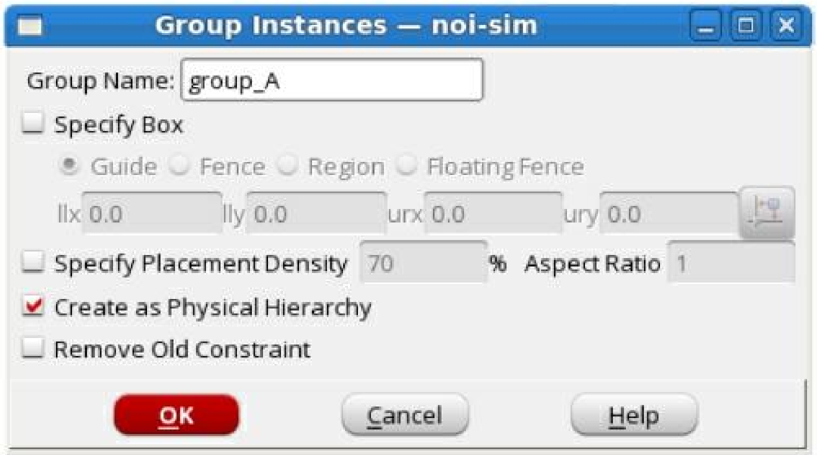
Grouping Instances

Instance groups are used to create a new logical hierarchy unit, and normally all power domains will associate with an instance group which always have a constraint fence. You can use the `createInstGroup` command or the *Group Instances* form to create a new instance group, even outside the core boundary.

Note: The constraint fence means standard cell belonging to this group cannot be placed outside and only standard cell belong to this group can be placed inside. Cadence recommends that this physical-logical coherence should not be violated. The coherence requirement does not only apply for instance groups, but also pertains to all hinsts and instance groups as long as they have constraint fence. In other words, if fence A belongs to fence B in logical, then fence A must be placed inside fence B.

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.

```
createInstGroup group_A -isPhyHier
addInstToInstGroup group_A B
addInstToInstGroup group_A F
createLogicHierarchy -commit -cell aa -newHinst group_A -objects {B F}
```



Group Instances — noi-sim

Group Name: group_A

□ Specify Box

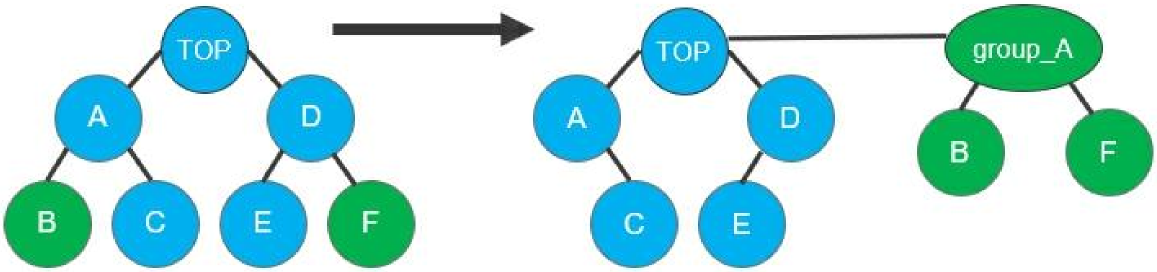
● Guide ○ Fence ○ Region ○ Floating Fence

llx 0.0 lly 0.0 urx 0.0 ury 0.0

□ Specify Placement Density 70 % Aspect Ratio 1

☑ Create as Physical Hierarchy

□ Remove Old Constraint



To delete an instance from an instance group use the `deleteInstFromInstGroup` command. Alternatively, 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 use the `addInstToInstGroup` command. Alternatively, 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.

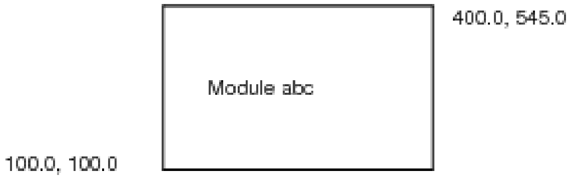
Defining the Bounding Box

During floorplanning, you can use the `setObjFPlanBox` command to define a bounding box of a specified object, and the `setObjFPlanBoxList` command to define rectilinear shape of an object, which is comprised of two or more boxes. This section provides graphical information to illustrate some of the command examples in the Floorplan Commands chapter of the *Innovus Text Command Reference*.

setObjFPlanBox

The following command specifies a bounding box for Module `abc` at a lower left x coordinate of 100.0, a lower left y coordinate of 100.0, and upper right x coordinate of 400.0, and an upper right y coordinate of 545.0:

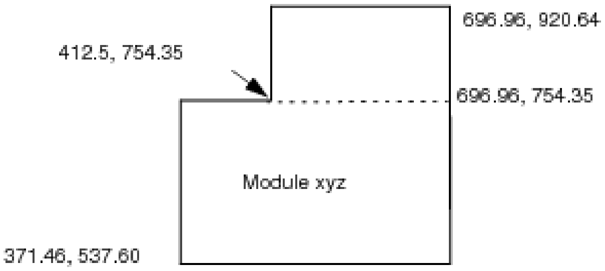
```
setObjFPlanBox Module abc 100.0 100.0 400.0 545.0
```



setObjFPlanBoxList

The following command defines a rectilinear boundary for Module `xyz`. The rectilinear boundary is made up of two bounding boxes: (371.46, 537.60) (696.96, 754.35), and (412.5, 754.32) (696.96, 920.64):

```
setObjFPlanBoxList Module xyz 371.46 537.60 696.96 754.35 412.5 754.35 696.96 920.64
```



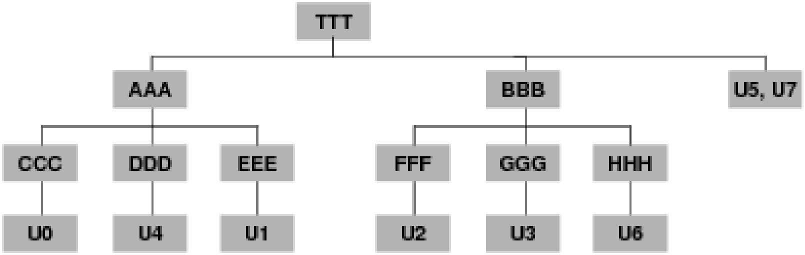
Adding Logical Hierarchy Without Creating Additional Hierarchy

The Innovus software enables you to add logical hierarchy without creating additional hierarchy. For example:

```
createInstGroup /TTT -isPhyHier
addInstToInstGroup /TTT U5
addInstToInstGroup /TTT U7
createLogicHierarchy -commit -cell aa -newHinst TTT -objects {U5 U7}
```

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:



Logical Hierarchy Manipulation

In addition to Adding Logical Hierarchy Without Creating Additional Hierarchy, you can also manipulate the logical hierarchy as follows:

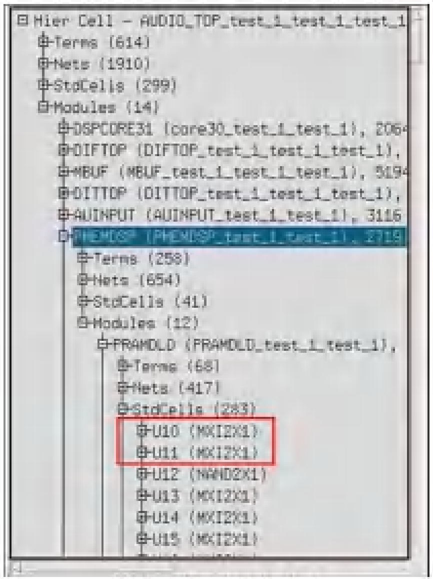
- Moving Instances to a New Top Module
- Moving Instances to an Existing Module

Moving Instances to a New Top Module

To move an instance to a new top module named `TOP101`, you can do the following:

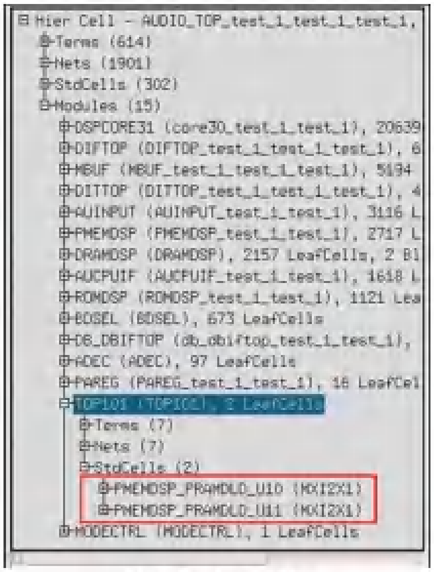
```
createInstGroup TOP101 -isPhyHier
addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U10
addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U11
```

createLogicHierarchy -commit -cell aa -newHinst TOP101 -objects {PMEMDSP/PRAMDLD/U10 PMEMDSP/PRAMDLD/U11}



```
⊟ Hier Cell - AUDIO_TOP_test_1_test_1_test_1
  ⊞-Terms (614)
  ⊞-Nets (1910)
  ⊞-StdCells (299)
  ⊟-Modules (14)
     ⊞-DSPCORE31 (core30_test_1_test_1), 2064
     ⊞-DIFTOP (DIFTOP_test_1_test_1_test_1),
     ⊞-MBUF (MBUF_test_1_test_1_test_1), 5194
     ⊞-DITTOP (DITTOP_test_1_test_1_test_1),
     ⊞-AUINPUT (AUINPUT_test_1_test_1), 3116
     ⊟-PMEMDSP (PMEMDSP_test_1_test_1), 2719
        ⊞-Terms (258)
        ⊞-Nets (654)
        ⊞-StdCells (41)
        ⊟-Modules (12)
           ⊟-PRAMDLD (PRAMDLD_test_1_test_1),
              ⊞-Terms (68)
              ⊞-Nets (417)
              ⊟-StdCells (283)
                 ⊞-U10 (MXI2X1)
                 ⊞-U11 (MXI2X1)
                 ⊞-U12 (NAND2X1)
                 ⊞-U13 (MXI2X1)
                 ⊞-U14 (MXI2X1)
                 ⊞-U15 (MXI2X1)
```

Before Restructuring



```
⊟ Hier Cell - AUDIO_TOP_test_1_test_1_test_1,
  ⊞-Terms (614)
  ⊞-Nets (1901)
  ⊞-StdCells (302)
  ⊟-Modules (15)
     ⊞-DSPCORE31 (core30_test_1_test_1), 20639
     ⊞-DIFTOP (DIFTOP_test_1_test_1_test_1), 6
     ⊞-MBUF (MBUF_test_1_test_1_test_1), 5194
     ⊞-DITTOP (DITTOP_test_1_test_1_test_1), 4
     ⊞-AUINPUT (AUINPUT_test_1_test_1), 3116 L
     ⊞-PMEMDSP (PMEMDSP_test_1_test_1), 2717 L
     ⊞-DRAMDSP (DRAMDSP), 2157 LeafCells, 2 Bl
     ⊞-AUCPUIF (AUCPUIF_test_1_test_1), 1618 L
     ⊞-ROMDSP (ROMDSP_test_1_test_1), 1121 Lea
     ⊞-BDSEL (BDSEL), 673 LeafCells
     ⊞-DB_DBIFTOP (db_dbiftop_test_1_test_1),
     ⊞-ADEC (ADEC), 97 LeafCells
     ⊞-PAREG (PAREG_test_1_test_1), 16 LeafCel
     ⊟-TOP101 (TOP101), 2 LeafCells
        ⊞-Terms (7)
        ⊞-Nets (7)
        ⊟-StdCells (2)
           ⊞-PMEMDSP_PRAMDLD_U10 (MXI2X1)
           ⊞-PMEMDSP_PRAMDLD_U11 (MXI2X1)
     ⊞-MODECTRL (MODECTRL), 1 LeafCells
```

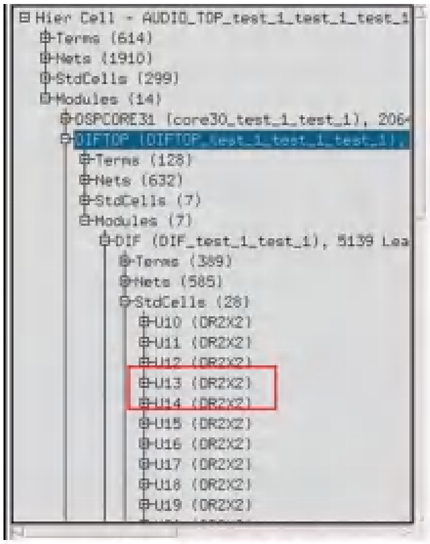
After Restructuring

Moving Instances to an Existing Module

To move an instance to an existing module named `DRAMDSP/DRAMDLD`, you can do the following:

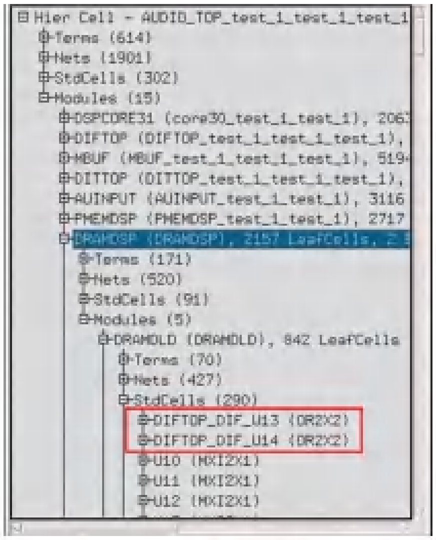
```
createInstGroup /DRAMDSP/DRAMDLD -isPhyHier
addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U13
addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U14
createLogicHierarchy -commit -cell aa -newHinst /DRAMDSP/DRAMDLD -objects
{DIFTOP/DIF/U13  DIFTOP/DIF/U14}
```

Note: The leading/(slash) is required for an existing module.



```
⊟ Hier Cell - AUDIO_TOP_test_1_test_1_test_1
  ⊞-Terms (614)
  ⊞-Nets (1910)
  ⊞-StdCells (299)
  ⊟-Modules (14)
     ⊞-DSPCORE31 (core30_test_1_test_1), 206
     ⊟-DIFTOP (DIFTOP_test_1_test_1_test_1),
        ⊞-Terms (128)
        ⊞-Nets (632)
        ⊞-StdCells (7)
        ⊟-Modules (7)
           ⊟-DIF (DIF_test_1_test_1), 5139 Lea
              ⊞-Terms (389)
              ⊞-Nets (585)
              ⊟-StdCells (28)
                 ⊞-U10 (OR2X2)
                 ⊞-U11 (OR2X2)
                 ⊞-U12 (OR2X2)
                 ⊞-U13 (OR2X2)
                 ⊞-U14 (OR2X2)
                 ⊞-U15 (OR2X2)
                 ⊞-U16 (OR2X2)
                 ⊞-U17 (OR2X2)
                 ⊞-U18 (OR2X2)
                 ⊞-U19 (OR2X2)
```

Before Restructuring



```
⊟ Hier Cell - AUDIO_TOP_test_1_test_1_test_1
  ⊞-Terms (614)
  ⊞-Nets (1901)
  ⊞-StdCells (302)
  ⊟-Modules (15)
     ⊞-DSPCORE31 (core30_test_1_test_1), 206
     ⊞-DIFTOP (DIFTOP_test_1_test_1_test_1),
     ⊞-MBUF (MBUF_test_1_test_1_test_1), 519
     ⊞-DITTOP (DITTOP_test_1_test_1_test_1),
     ⊞-AUINPUT (AUINPUT_test_1_test_1), 3116
     ⊞-PMEMDSP (PMEMDSP_test_1_test_1), 2717
     ⊟-DRAMDSP (DRAMDSP), 2157 LeafCells, 2
        ⊞-Terms (171)
        ⊞-Nets (520)
        ⊞-StdCells (91)
        ⊟-Modules (5)
           ⊟-DRAMDLD (DRAMDLD), 842 LeafCells
              ⊞-Terms (70)
              ⊞-Nets (427)
              ⊟-StdCells (290)
                 ⊞-DIFTOP_DIF_U13 (OR2X2)
                 ⊞-DIFTOP_DIF_U14 (OR2X2)
                 ⊞-U10 (MXI2X1)
                 ⊞-U11 (MXI2X1)
                 ⊞-U12 (MXI2X1)
```

After Restructuring

Creating and Editing Rows

You can create and edit rows in different regions. The following table lists the commands that you can use to create and cut rows.

`createRow`	Creates rows for the specified site. The row boundary can be defined by core area or the area that you specify. This command supports the creation of overlapping rows. This command can create only horizontal rows. By default, the rows are flipped and abutted.
`cutRow`	Cuts site rows that intersect with the specified area or object.
`deleteRow`	Deletes the specified row(s).
`stretchRows`	Stretches selected rows. For example, you can specify that the left edge of all selected rows should be aligned to the left-most edge among all selected rows.

You can also use the following forms available in the *Row* submenu on the *Floorplan* menu:

- *Create Core Rows* (*Floorplan - Row - Create Core Row*)
- *Cut Core Rows* (*Floorplan - Row - Cut Core Row*)
- *Stretch Core Rows* (*Floorplan - Row - Stretch Core Row*)

For more information, see the "Row" section of the *Floorplan Menu* chapter in the *Menu Reference.*

Using Vertical Rows

Note: Support for vertical rows is a beta feature. Usage and support of this beta feature are subject to prior agreement with Cadence. Contact your Cadence representative if you have any questions.

In addition to horizontal rows, Innovus also supports vertical rows. You can import designs with vertical rows and output design data containing the layout of vertical rows. Vertical rows appear vertically in the display. You can select and query vertical rows and delete them with the delete key. The commands that snap rows to row grid also support vertical rows. These commands are `snapFPlan`, `create_relative_floorplan`, and the interactive `move` command.

Horizontal and vertical rows can co-exist, but at different layers of hierarchy. You cannot create horizontal and vertical rows together on the same level of hierarchy.

The global variable `fp_vertical_row` is used to specify whether the rows are horizontal or vertical. The variable can be set to 0 (for horizontal rows) or 1 (for vertical rows) as follows:

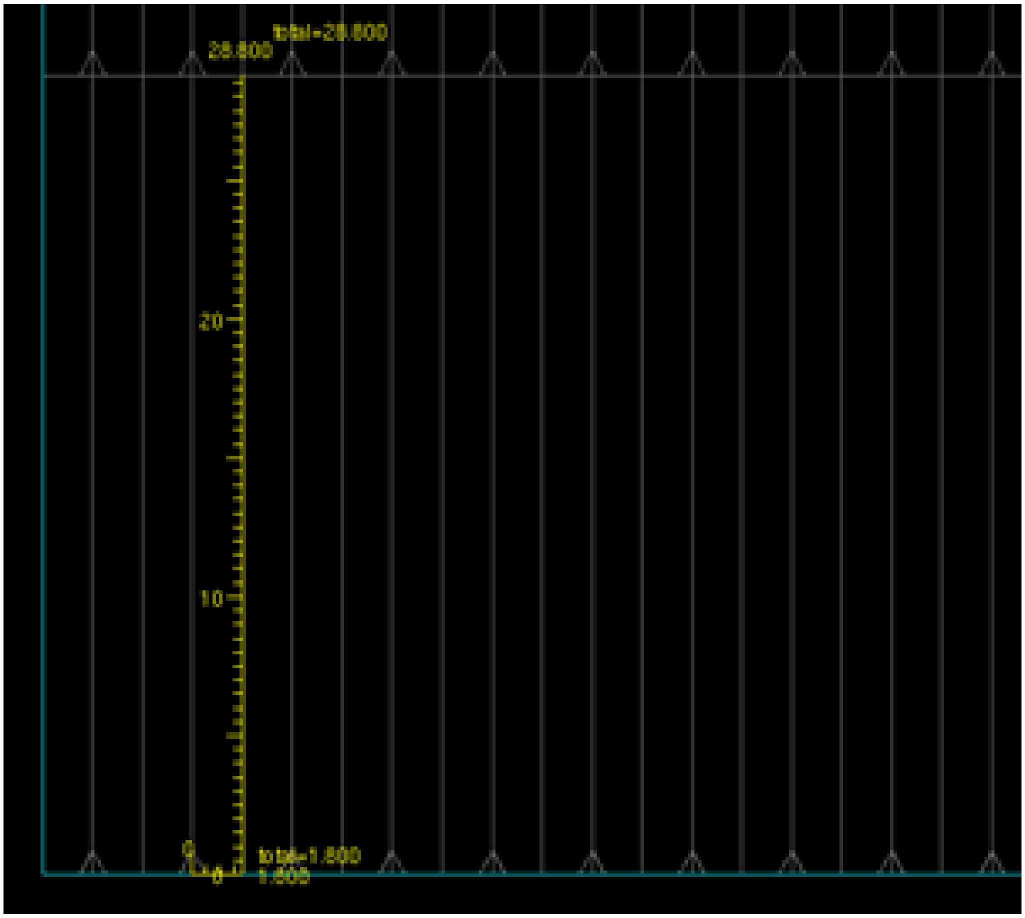
```
set fp_vertical_row {0|1}
```

The Innovus software generates vertical rows, provided the design data or the library meets the following prerequisites:

- Each SITE is stacked one above the other, when rows are created.
- MY and R0 row orientations are supported.
- The cells are stacked vertically, when they are placed and their power rails run vertically.
- The preferred direction of Metal1 is vertical.

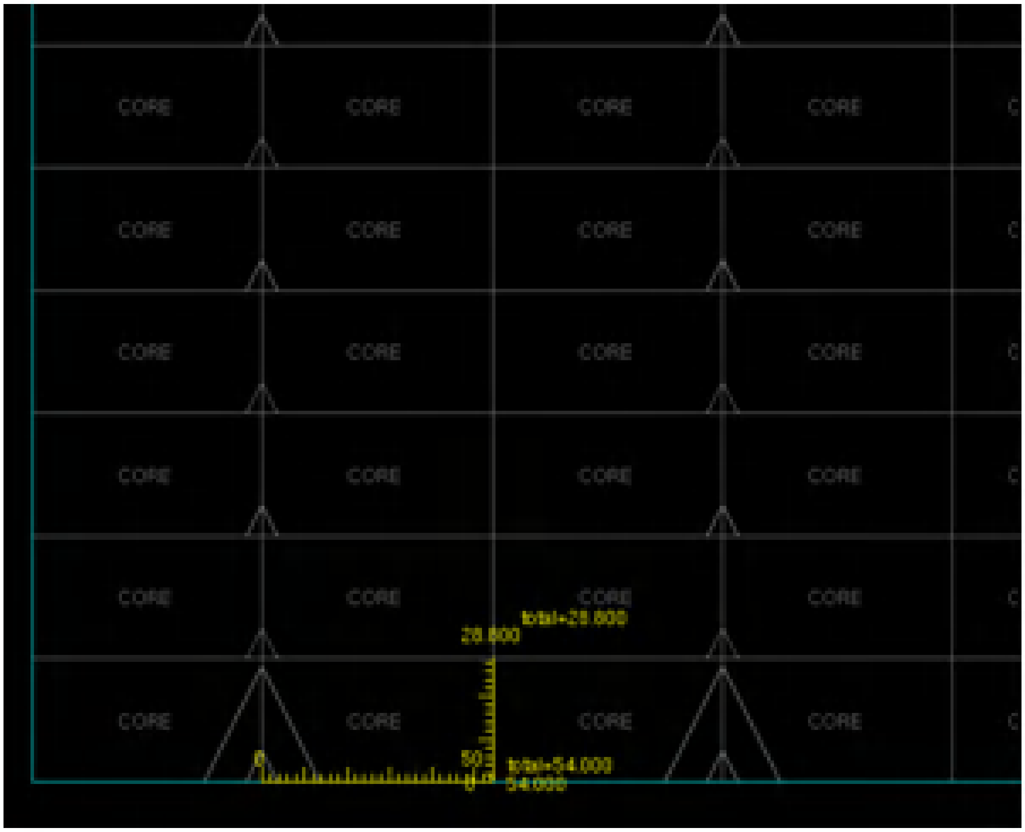
Example 1:

```
SITE CORE
CLASS CORE;
SIZE  1.800 BY 28.800;
END CORE
```



Example 2:

```
SITE CORE
CLASS CORE;
SIZE 54.000 BY 28.800;
END CORE
```



Limitations

The following limitations apply to vertical rows:

- Vertical rows cannot be created inside power domains.
- Non-integer and multiple integer vertical rows are not supported.
- Vertical rows cannot be created or edited directly. That is, the `createRow`, `cutRow`, and `stretchRows` commands are not supported for vertical rows.
  Note: You can, however, select rows to query, and delete rows with the delete key or `deleteRow` command.

Using Multiple-height Rows

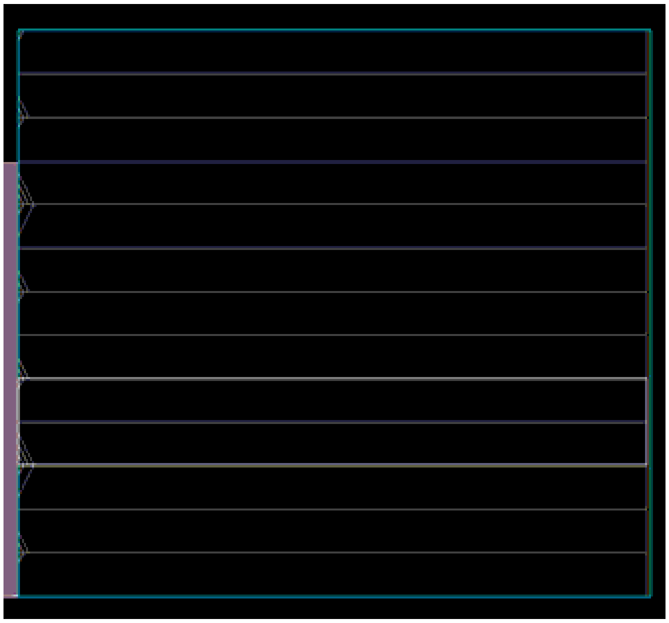
In many cases, designs contain cells with different standard cell heights. For example, a design might utilize multiple standard cell libraries-possibly from different foundries or library vendors-which might have different standard cell heights. Standard cell designers create multiple-height standard cells for improving performance. Also, in a design with multiple power domains, standard cells with different voltages will probably have different footprints and different heights.

The Innovus software supports multiple-height standard cells by supporting:

- A combination of integer multiple-height rows
- A combination of non-integer multiple-height rows

Using Integer Multiple-height Rows

The Innovus software automatically generates integer multiple-height rows overlapping the single-height core rows provided the design data or the library meets the following prerequisites:



- The LEF file contains integer multiple-height SITE definitions and MACROS that use the SITE.
- The netlist includes at least one instantiation of such an integer multiple-height cell.

After you import the design or specify the floorplan, the core area is automatically populated with default rows and multiple-height rows are automatically generated.

Here is an extract from a sample LEF file that contains integer multiple-height SITE definitions and a MACRO that uses a SITE:

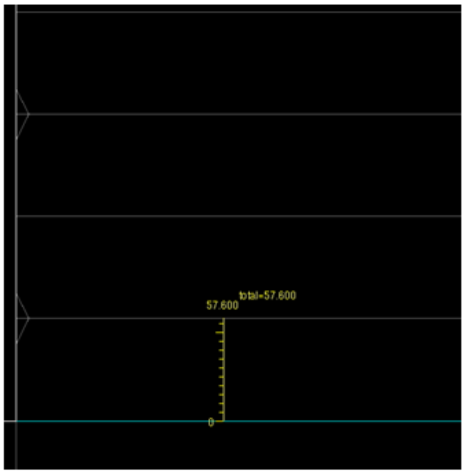
```
SITE coreSite
   SYMMETRY X Y;
   CLASS CORE;
   SIZE 0.660 BY 5.040;
END coreSite

SITE doubleHeightSite
   SYMMETRY X Y;
   CLASS CORE;
   SIZE 0.660 BY 10.080;
END doubleHeightSite

MACRO DFFX64
   CLASS CORE;
   FOREIGN DFFX64 0 0;
   ORIGIN 0 0;
SIZE 21.12 BY 10.080;
SYMMETRY X Y;
SITE doubleHeightSite;
...
END DFFX64
```

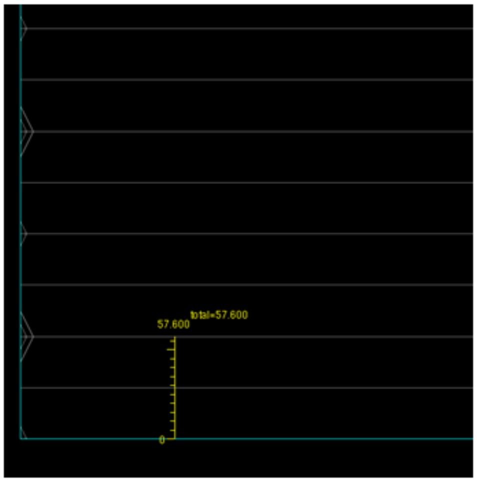
When you create integer multiple-height rows, the rows are automatically aligned with the single-height row. You cannot create unaligned integer multiple-height rows.

For information on creating and editing rows, see Creating and Editing Rows

The following left figure illustrates a double-height row and the right figure illustrates a double-height row flipped to align with the orientation of the single row.

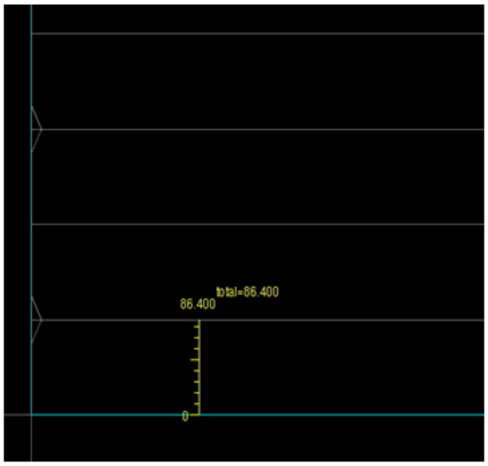
```
SITE CORE
     CLASS CORE;
     SIZE 1.800 BY 28.800;
END CORE
SITE CORE_double
     CLASS CORE;
     SIZE 1.800 BY 57.600;
END CORE_double
```

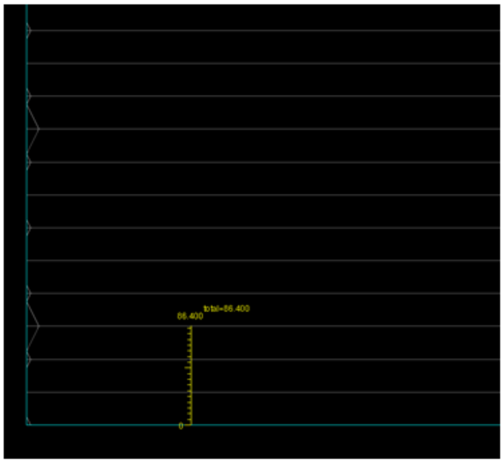




The following left figure illustrates a triple-height row and the right figure illustrates a triple-height row flipped to align with the orientation of the single row.

```
SITE CORE
     CLASS CORE;
     SIZE 1.800 BY 28.800;
END CORE
SITE CORE_triple
     CLASS CORE;
     SIZE 1.800 BY 86.4;
END CORE_triple
```





Using Non-Integer Multiple-height Rows

You can also use non-integer multiple-height (NIMH) rows in your designs.

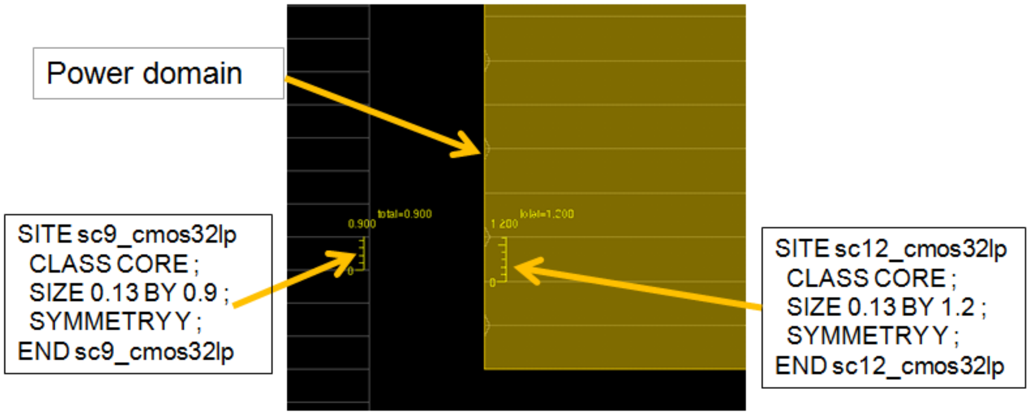
While creating NIMH rows, ensure the following:

- NIMH rows must be created *only* in those areas that have power domains associated with them.
- Any newly created NIMH rows in an area must be an integer multiple of any existing rows in the area.

Each hierarchical instance to be declared as a power domain can only have one type of NIMH standard cells. In other words, NIMH rows inside any particular power domain must have the same height. Multiple types of NIMH standard cells require multiple power domains to be created. For example, if you want to use standard cells with heights that are respectively 2.5, 3.25, and 4.1 times the height of a standard single-height cell, you should create three power domains, with each power domain containing one type of NIMH row.

When you create a power domain, Innovus automatically detects the site that is common to all the cells and creates the rows inside the power domain.

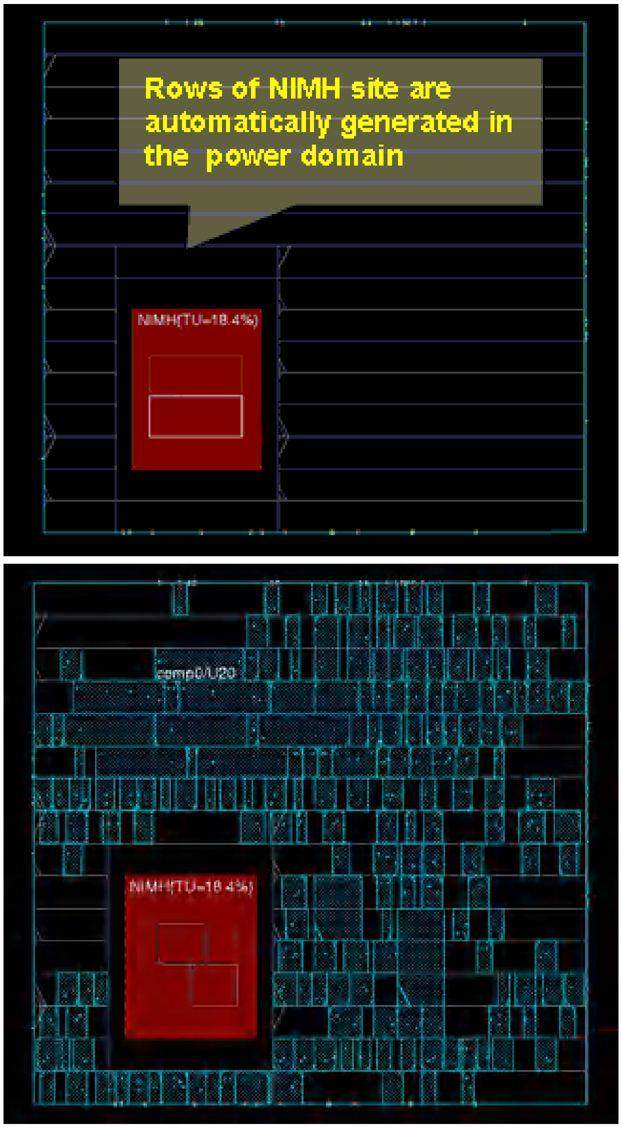
The following diagram illustrates how rows are automatically generated within the power domain for standard cells of the hierarchical instance.



```
SITE sc9_cmos32lp
  CLASS CORE ;
  SIZE 0.13 BY 0.9 ;
  SYMMETRY Y ;
END sc9_cmos32lp
```

```
SITE sc12_cmos32lp
  CLASS CORE ;
  SIZE 0.13 BY 1.2 ;
  SYMMETRY Y ;
END sc12_cmos32lp
```

The following diagram illustrates how the standard cells of the hierarchical instance are all placed on the row inside the power domain.

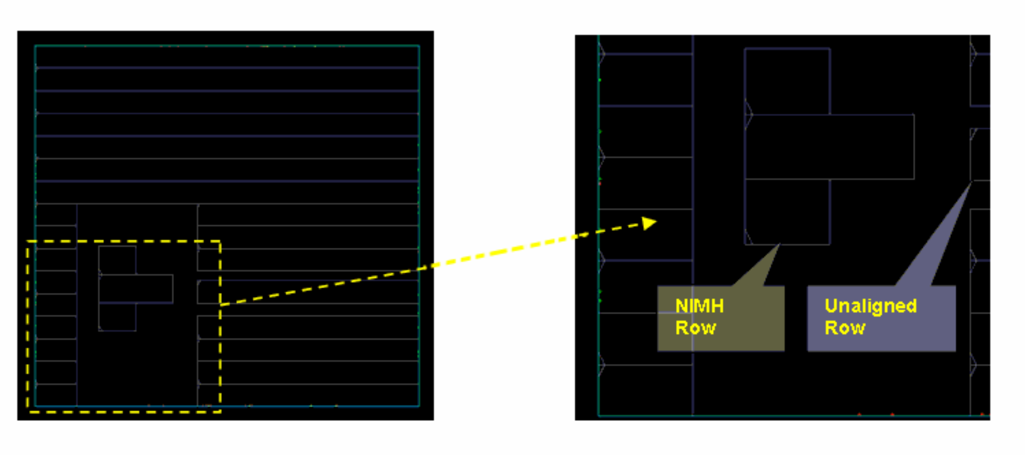


Rows of NIMH site are automatically generated in the power domain

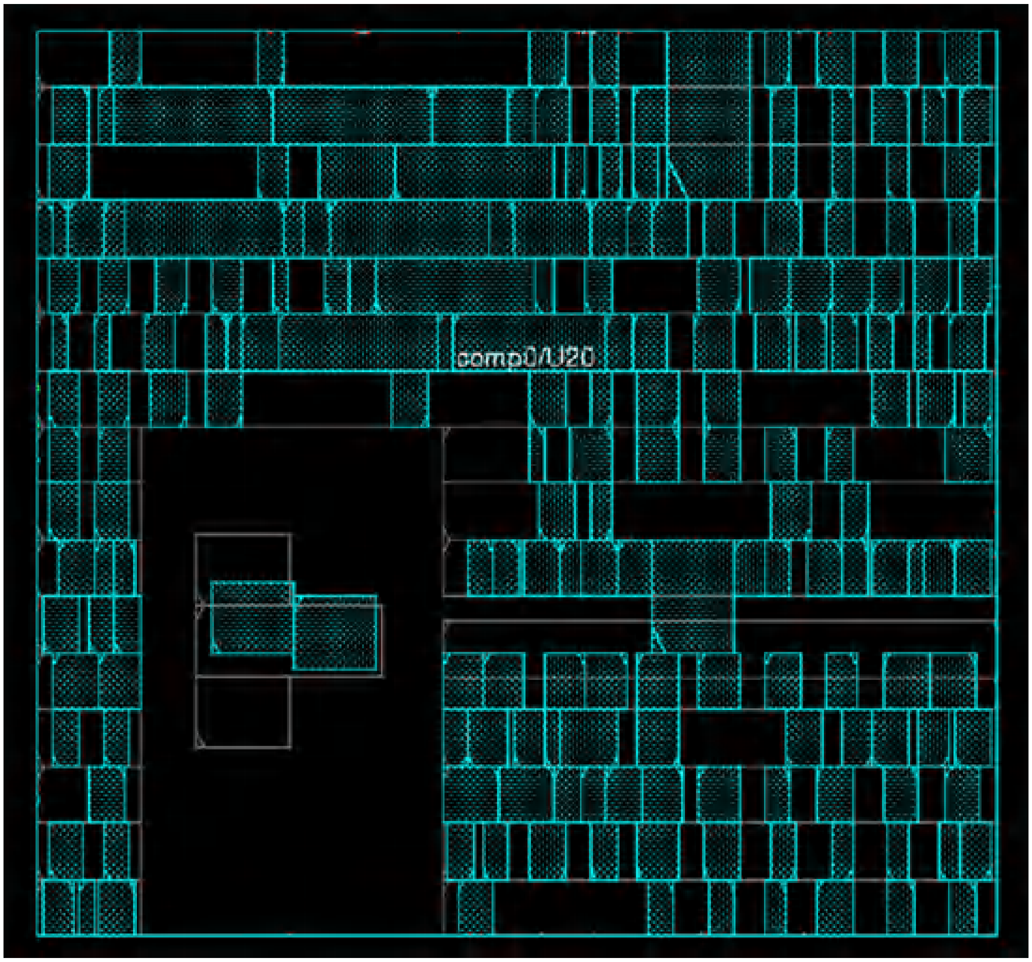
The modules and/or instance groups can be moved outside the core boundary but within the die. As new rows are not created automatically in this area, you can use the `createRow` command to create new rows. Manual editing of rows might not be preserved by `floorPlan`, `resizeFloorplan`, and `initCoreRow` commands. Innovus displays a warning message if you try to move a power domain outside the core boundary, and snaps the power domain inside the core.

Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows

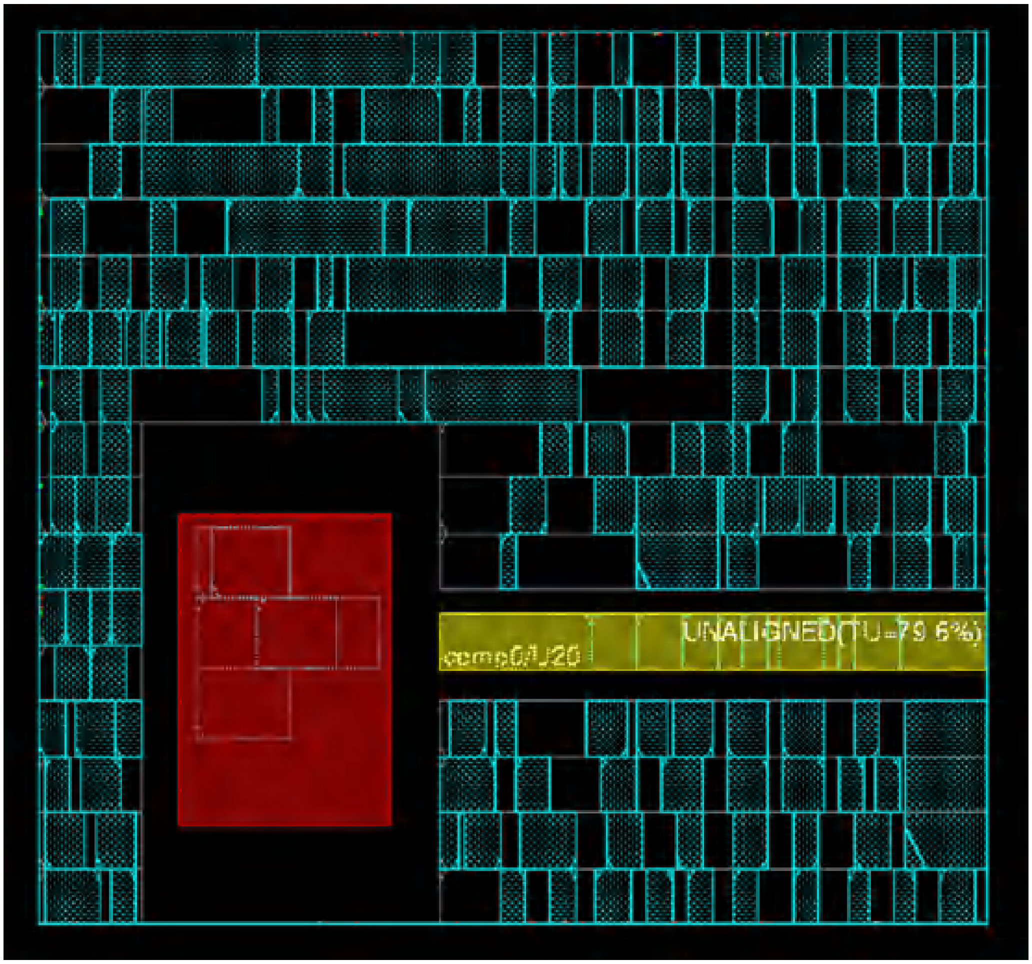
In case of integer multiple-height rows, as long as the rows are overlapping the single-height rows, standard cells will by default be legally placed on their corresponding site or row definition. However, if you import a user-defined plan through a DEF file that contains NIMH rows and unaligned rows, you need to define power domain(s) for each of these disjoint special row style. Otherwise, these rows might not be correctly placed.
The following figure illustrates a user-defined floorplan brought in through a DEF file.



The following figure illustrates how placement without power domain association results in illegal placement.



The following figure illustrates how placement with the correct power domain association results in legal placement.



By default, when a power domain is moved or (re)located in the main core row area, rows are initialized. To keep the rows brought in by the DEF file, you should pre-place and pre-size the power domains that cover the NIMH rows and the unaligned rows.

Merging Hierarchical Floorplans from Partitions

While flattening partitions with the `flattenPartition` command, you can bring back row information, including NIMH rows and unaligned rows, from an existing floorplan. You can then run placement and routing to further improve the design performance. Use the `flattenPartition` command can preserve the row information by default. The `flattenPartition` command also supports rotated partitions.

Power domains are automatically created and associated with each of the partition that have NIMH or unaligned rows. These automatically created power domain have the following characteristics:

- The `minGap` value is the same as the placement halo defined for the partition at the full-chip level.
- The timing library is the same as that specified at the full-chip level.
- The global net connection is the same as that for the full-chip level, which is the same as the partition floorplan global net connection.
- The value of the `RouteSearchExt` field is set to the default value of 0.0.
- The core-to-edge distances are not preserved as an attribute of the power domain, but are preserved in the merged floorplan.
- Row parameters are not translated or detected.
- The power domain is set to `alwaysOn`.

Use Model

The recommended use model for bringing back non-integer multiple-height rows or unaligned rows is as follows:

Top-down flow

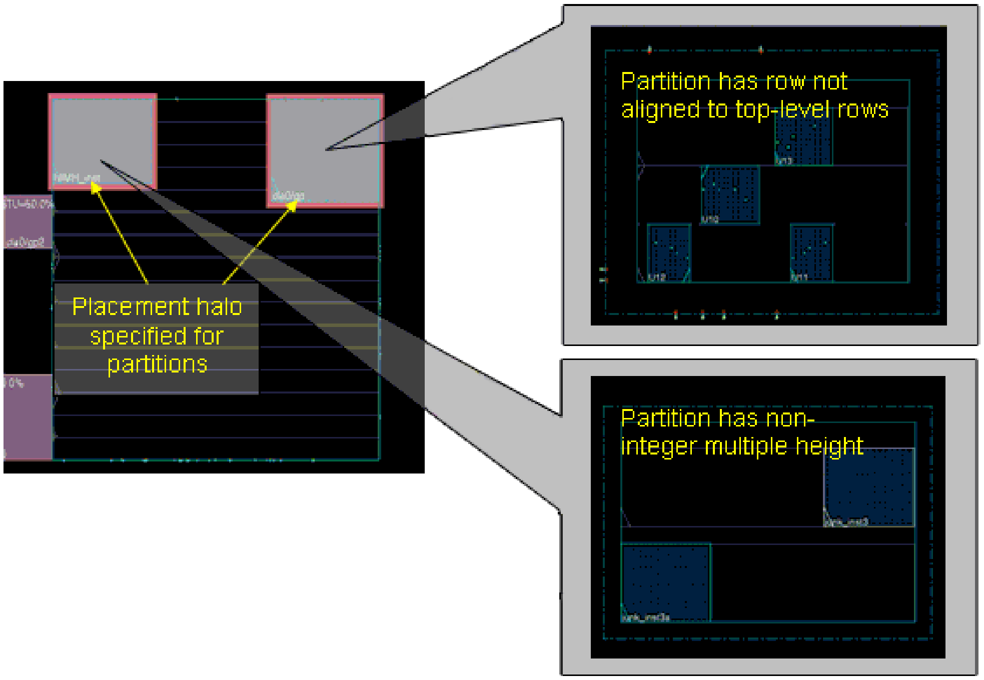
1. Create power domains at full-chip level design.
2. Specify the same hierarchical instance for power domain as defined for the partition.

Bottom-up flow

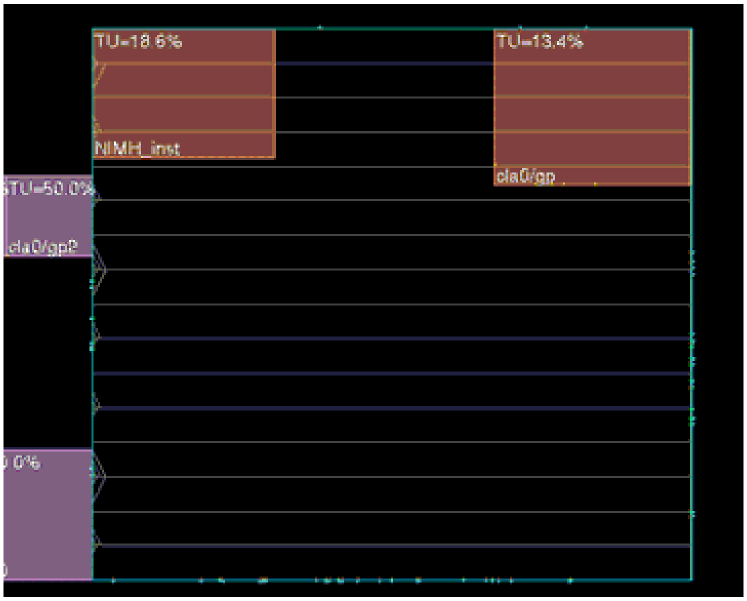
1. Create power domains at top-level design.
2. Create one PD for each of the partitions.
3. Assign instance blocks as a member of the created power domain.

For more information on the `flattenPartition` command, see the *Text Command Reference*.

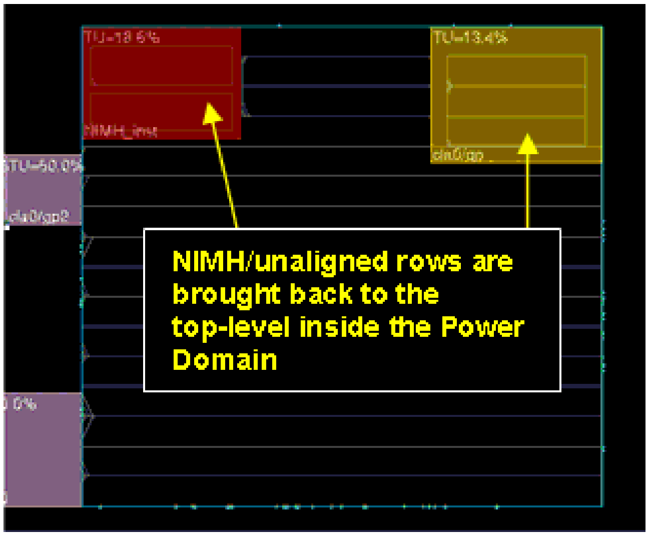
The following figure shows a full-chip view with the partition halos specified.


NIMH_inst
cla0/gp
TU=50.0%
cla0/gp2
Placement halo specified for partitions
Partition has row not aligned to top-level rows
Partition has non-integer multiple height

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.


TU=18.6%
NIMH_inst
TU=13.4%
cla0/gp
STU=50.0%
cla0/gp2
0.0%
NIMH/unaligned rows are brought back to the top-level inside the Power Domain

Performing I/O Row Based Pad Placement

In many cases, designs contain multiple-height I/O pads or asymmetric I/O rings, for example, a design might have a single I/O ring on one side and double rings or no rings on the other side, or no rings on part of a certain side. For such designs, the Innovus software enables you to create, edit, save, and restore I/O rows and perform pad placement based on the I/O rows. You can create I/O rows anywhere in the die – within the core or in the periphery and use the I/O row flow for both, pad and area I/Os.

Prerequisites

1. LEF technology file should contain I/O SITE definition.
   Before you begin the I/O row flow in Innovus, you must first define I/O SITE for each type of I/O cell in the LEF I/O macro (LEF technology file).
2. Each I/O cell LEF must have correct CLASS and SITE type specified.
   Consider the following examples that define LEF I/O SITE and CLASS PAD MACRO in the I/O assignment file, each I/O CLASS PAD macro is referenced with the I/O SITE:

   Example 1:
   The I/O SITE IOPFC is referenced from the I/O CLASS PAD MACRO pnl_qdr_vp:

```
SITE IOPFCSYMMETRY 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 *Innovus User Guide*.

3. Each I/O cell LEF must have correct CLASS and SITE type specified. If the design contains multiple I/O SITES, the gap between the core boundary and the die boundary must be greater than the biggest I/O SITE; Otherwise, the Innovus software issues a warning and no I/O rows are created. Innovus does not automatically expand the core or die boundary to accommodate

all the I/O pads.

Enabling the I/O Row Flow in Innovus

To start using the I/O row flow in Innovus, you must enable the I/O row flow by doing one of the following:

- Specify `setUserDataValue conf_use_io_row_flow 1` in the Innovus global variable file.
- Select *Use I/O Row for I/O Placement* check box in the *Floorplan* - *Specify Floorplan - Advanced* GUI form.
- Set the `setIoFlowFlag` command to 1.

To stop using the I/O row flow in Innovus, you must disable it by setting the `setIoFlowFlag` command to 0.

Note: By default, the I/O row flow is Off. Use the `getIoFlowFlag` command to check the current flow. A value of "0" means traditional I/O flow where as a value of 1 means I/O row flow.

Use Models

Starting a new design

1. Import the design in Innovus
2. Set the I/O row flow by selecting the *Use I/O Row for I/O Placement* check box in the *Specify Floorplan - Advanced* 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 *Floorplan - Row* form or using the text commands.

The text commands that can be used for creating and editing I/O rows are described in the following table.

Commands	Usage
`createIoRow`	Creates an I/O row.
`flipOrRotateObject`	Flips or rotates the selected objects.
`stretchRows`	Stretches the selected I/O rows.
`deleteRow`	Deletes the selected I/O row. You can also delete the row by selecting the row and pressing the `Del` key on the keyboard.
`setIoRowMargin`	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.
`snapFPlanIO`	Snaps the I/O pads onto the correct side of the I/O rows if the pads are not already on the rows.
`spaceIoInst`	Spaces the selected I/O pads on the I/O rows, horizontally or vertically by a specified distance value.

Optionally, you can specify the I/O row constraints in the I/O assignment file. For more information, see "Generating the I/O Assignment File" in the Data Preparation chapter of the *Innovus User Guide*.

Note: To add I/O filler cells to the new I/O rows, use the `addIoRowFiller` command. You can delete the filler cells using `deleteIoRowFiller` command.

After designing the rows, save the design in a floorplan file (*.fp) using
the `saveDesign` or `saveFPlan` command.

Reading an old design

If you already have a design with placed pads, created using an earlier version of the software (v6.2 and above) and you want the design to be read into current version of Innovus to use the new I/O row flow:

1. Restore the design with placed pads, using the `restoreDesign` command or load the floorplan information from the file, using the *Load FPlan File* form or the `loadFPlan` command. Optionally, load the I/O constraint file using the `loadIoFile` command.
2. Turn on the I/O row flow by setting the I/O flow flag (`setIoFlowFlag`) to 1.
3. Create the initial I/O rows using the `createIoRow -deriveByCells` command. This command

creates I/O rows based on the existing pad placement. Once you have rows and pads placed in the design, you can continue to create more rows or edit the rows.
Use the `changeIoConstraints` command to change the constraints of the I/O row read from the I/O constraints file.You can also use the *Attribute Editor* to change the I/O pad location or pad orientation from the GUI.

4. After editing the I/O rows and the I/O row constraints, run the `snapFPlanIO` command to snap the I/O pads and area I/O's onto the legal sites/rows.
5. Save the design in a floorplan file (*.fp) using the `saveDesign` or `saveFPlan` command.

Resizing Rectilinear block-level floorplan

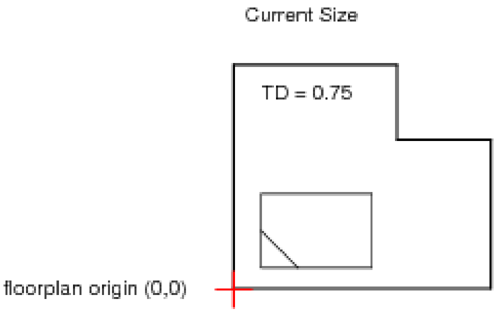
Given an initial rectilinear block-level floorplan, Innovus automatically resizes its bounding box by enlarging or shrinking the edges of the box proportionally in the X and Y directions, ensuring that the specified target utilization is met. During this process, to retain the original shape of the rectilinear block-level floorplan, you must specify the `-keepShape` parameter in the `floorPlan` command. Consider analog designs where you have digital blocks that need to be fit into the analog chip and the shape of the block is already pre-defined. In such mixed-signal designs, you can retain the block shape during resizing, and also meet the specified target utilization value by shrinking or expanding the floorplan using the `floorPlan -keepShape` *`util`*, where `util` is the target core utilization value.

Use Models

1. Import a DEF file or a floorplan file (*.fp) that contains a rectilinear block-level floorplan boundary or you cut one corner of a rectangular block. This results in the core utilization missing the target value.
2. Run the `floorPlan -keepShape util` command to automatically shrink or expand the block-level floorplan boundary, proportionally, in the X and Y directions, trying to meet the specified target core utilization.
3. Use the `checkFPlan -reportUtil` command to report the final core utilization.

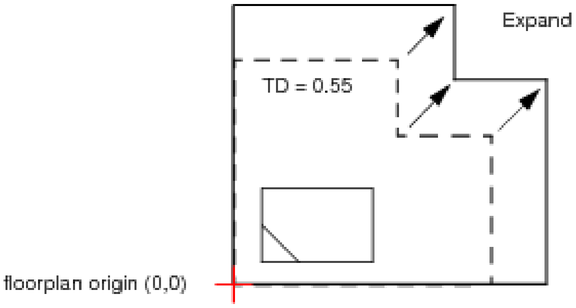
Example:

Consider the following example of a rectilinear block-level floorplan whose current target core utilization value is 0.75.



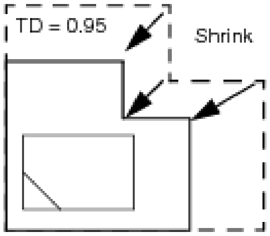
To meet a target core utilization of 0.55, expand the rectilinear block-level floorplan.

```
floorplan -keepShape 0.55
```



To meet a target core utilization of 0.95, shrink the rectilinear block-level floorplan.

```
floorplan -keepShape 0.95
```



In both the cases, the shape of the block-level floorplan is retained, and the required target core utilization is met.

For I/O pins, prior to resize, Innovus saves the I/O file sequence internally and loads the file back after resize. The side and sequence of the I/O pin remains the same as in the old block, but the pins get distributed evenly. To redistribute the I/O pins, you must edit the pins manually in the resize block.

Assumptions

The automatic resizing of rectilinear block-level floorplan is based on the following assumptions:

1. The rectilinear block-level floorplan are L-shaped.
2. The floorplan origin, (0,0) remains unchanged during the resize.
3. The instances inside the block move proportionally or stay fixed during the resize.

Results

The results of automatic resizing of rectilinear block-level floorplan are as follows:

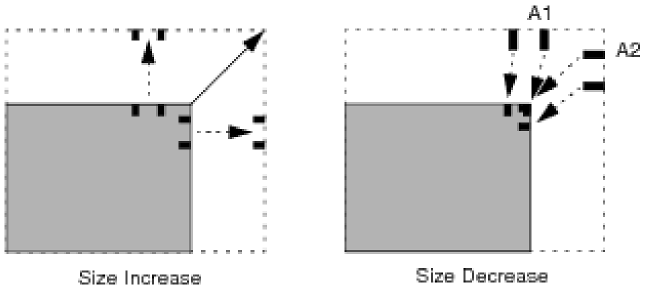
1. The shape of the block-level floorplan is preserved during the resize.
2. Pre-placed macros are adjusted to the new size as much as possible.
3. Pre-routed wires are removed after the resize.
4. The core rows and block pins are automatically adjusted after the resize.

Editing Pins

This section describes how you can move and manipulate pins in your design. For information on blackbox and partition pins, see the Assigning Pins section in the Partitioning the Design chapter of the *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 Metal2 is changed to Metal1 when moving pins from top to left. Moving pins from top to bottom does not change the layer. To move a selected pin or group of pins in the design display area from one edge to another edge (including rectilinear edges) on a module, complete the following steps:

Note: For pin groups, this will preserve the relative position between pins.

1. Click the *Move/Resize/Reshape* widget.
2. Select (left-click) the pin in the design display area.
   For a group of pins, press the `Shift` key to highlight each pin.
3. Left click on the pin(s) and move them to the new location.

Note: To zoom out on the design display area while dragging the pins, press the `Shift-Z` key combination.

You can use the `moveGroupPins` command for moving the pin(s) to the new location.

Swapping Pins

You can swap pins using the `swapPins` command or the *Swap Pins* option in the design display area by completing the following steps:

1. Select two pins of the same block.
2. With the cursor over one of the selected pins, right-click the mouse to bring up the context menu.
3. Select *Swap Instances*.

Using the Pin Editor

You can use the *Pin Editor* to display and edit pins and pin groups. To open the *Pin Editor*, choose *Edit - Pin Editor*. For information in the fields and options, see *Pin Editor* in the *Edit Menu* chapter of the *Menu Reference.*

Here are the main features of the *Pin Editor*:

- Works for all type of pins such as partition pins, blackbox pins, and I/O pins
- When moving pins, associated pin geometry is moved or updated accordingly

- Can be used to move a single pin and/or a group of pins
- Supports pin editing by various criteria such as location, layer, side/edge, and so on
- Provides pin spreading capabilities. For more information, see Using the Pin-Spreading Feature
- Provides pin snapping capabilities such as to manufacturing grid, user grid, and layer track.
- Supports non-preferred routing layers for all supported snapping grids.
- Honors pin constraints at partition-level and pin-level, as well as constraints defined through the GUI.
- Supports pre-assigned pins.
- Supports rectilinear edges (multiple edges per side).

The `editPin` command provides the equivalent functionality of the *Pin Editor.*

The following sections describes some of the features that you can use with the *Pin Editor*.

Using the Pin-Spreading Feature

The *Pin Editor* includes a utility to spread pins along the edges of a block. There are four different methods of spreading pins:

- Use a pin as the starting point (anchor) and provide a pin spacing distance.
- Use the center of a side or edge as the starting point and provide a pin spacing distance.
- Space the pins evenly along the side or edge, using the ends of the side or edge as the starting and ending points. The Pin Editor calculates the pin spacing distance.
- Space the pins evenly using explicit starting and ending points on the side or edge. The Pin Editor calculates the pin spacing distance.

Basic Concepts for Pin Spreading

Two basic concepts underlie the pin-spreading functionality of the Pin Editor:

- Pin ordering affects the starting point for pin spreading. Use the Pin Editor's Group Bus, Reverse Order, or Reorder Pin List functions to specify the first pin in a group. The coordinates of the first pin in a group provide the starting point from which to spread pins.

- Pin spacing distances can be expressed in either positive or negative values:
  - Positive spacing values spread pins to the right along a horizontal block edge, or up along a vertical block edge.
  - Negative spacing values spread pins to the left along a horizontal block edge, or down along a vertical block edge.

Note: You cannot specify pin spacing distances with spacing methods that rely on the Pin Editor to determine the spacing.

Pin Spreading Methods Supported by the Pin Editor

The following sections provide details on the four pin-spreading methods supported by the Pin Editor.

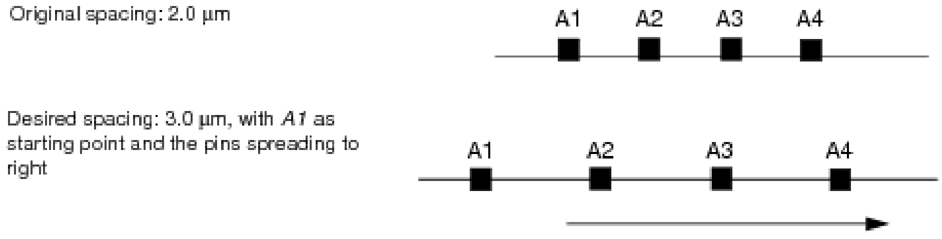
- Using a Pin as the Starting Point for Spreading Pins
- Using the Center of a Side or Edge as the Starting Point for Spreading Pins
- Spacing Pins Evenly Along an Edge or Side
- Spacing Pins Evenly Using Explicit Starting and Ending Points

Using a Pin as the Starting Point for Spreading Pins

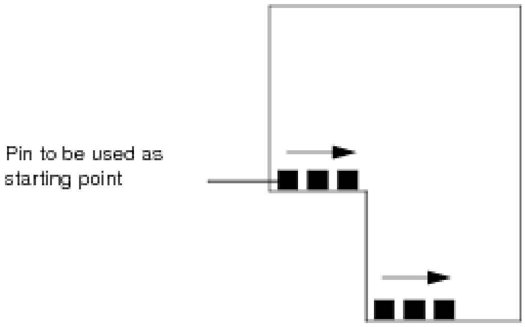
For this method, you select a group of pins and sort them in the desired order. The first pin in the list serves as the starting point (anchor) for spreading the other pins in the group. You must also provide the pin spacing distance if you are spreading more than one pin. Assume that your design contains four pins (*A1, A2, A3*, and *A4*) that are currently spaced 2.0 µm apart. You want to spread the pins to the right with 3.0 µm spacing, using A1 as the starting point. To do this you must

1. Sort the pins so that A1 is the first pin in the list. The coordinates of A1 appear in Starting X/Y.
2. Select *Spread - From Starting Point* on the *Pin Editor* form. Specify a positive spacing value: 3.0.

The following figure illustrates this situation:


Original spacing: 2.0 µm
A1
A2
A3
A4
Desired spacing: 3.0 µm, with *A1* as starting point and the pins spreading to right
A1
A2
A3
A4

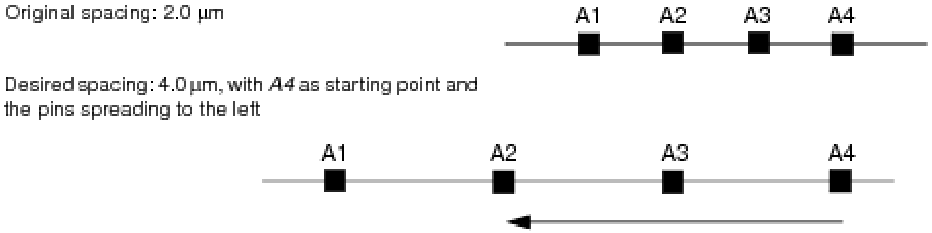
The following figure shows how pins are spread from a pin as starting point in case of rectilinear partitions for a side with multiple edges. In this case, the specified side is bottom, and the pins are therefore spread along the edges of the bottom side.



Now assume that you want to spread the pins to the left with 4.0 µm spacing, using *A4* as the starting point. To do this you must:

1. Sort the pins so that A4 is the first pin in the list. The coordinates of A4 appear in the *Starting* field of the *Pin Editor* form.
2. Specify a negative spacing value: -4.0.

The following figure illustrates this situation:


Original spacing: 2.0 µm
A1
A2
A3
A4
Desired spacing: 4.0 µm, with *A4* as starting point and the pins spreading to the left
A1
A2
A3
A4

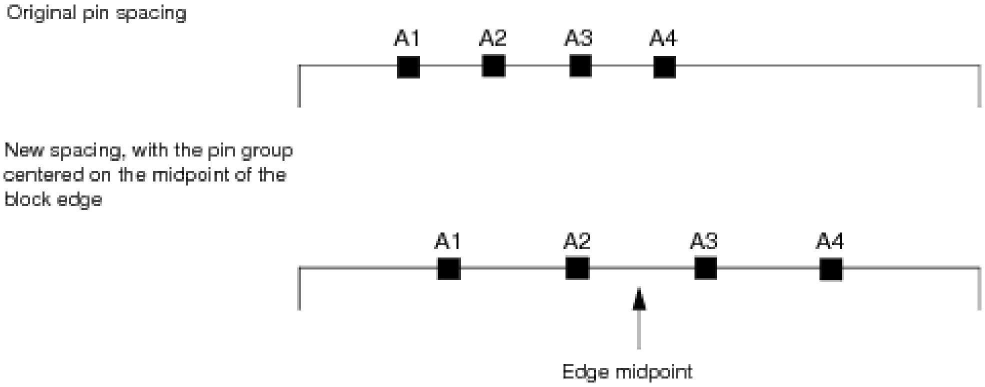
Using the Center of a Side or Edge as the Starting Point for Spreading Pins

For this method, you select a group of pins and sort them in the desired order. You must also provide the pin spacing distance.

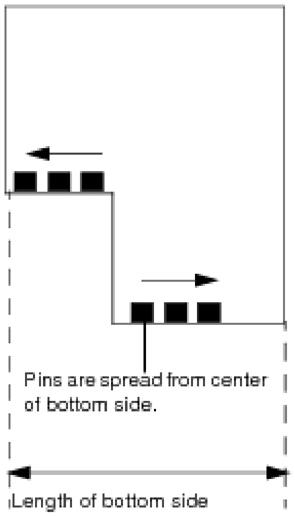
Assume that your design contains four pins (A1, A2, A3, and A4). You want to define new spacing and then group the pins so that the group is centered on the midpoint of the block edge. To do this you must

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
A1
A2
A3
A4
New spacing, with the pin group centered on the midpoint of the block edge
A1
A2
A3
A4
Edge midpoint

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

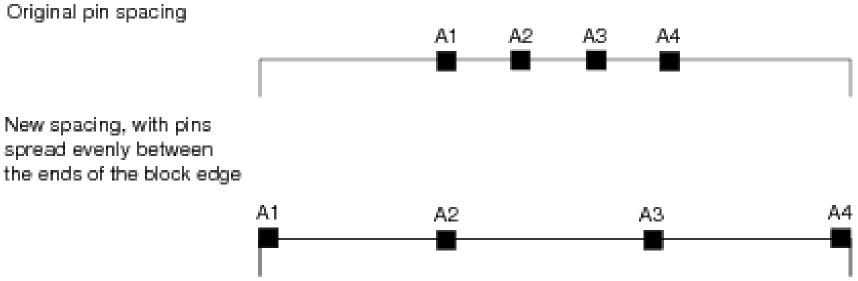
For this method, you select a group of pins and sort them in the desired order. You do not specify a pin spacing distance because the *Pin Editor* calculates the appropriate distance, based on the length of the block edge or side, and spaces the pins evenly along the block edge or side.

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

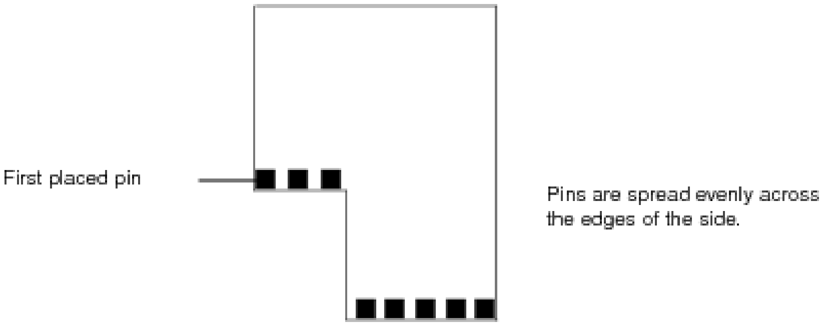
1. Sort the pins in the desired order (optional).

2. Select *Spread - Along Entire Edge* on the *Pin Editor* form.

The following figure illustrates this situation:


Original pin spacing
A1
A2
A3
A4
New spacing, with pins spread evenly between the ends of the block edge
A1
A2
A3
A4

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.



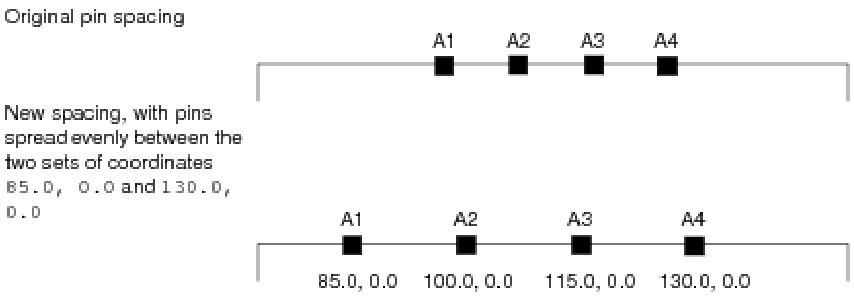
Spacing Pins Evenly Using Explicit Starting and Ending Points

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.

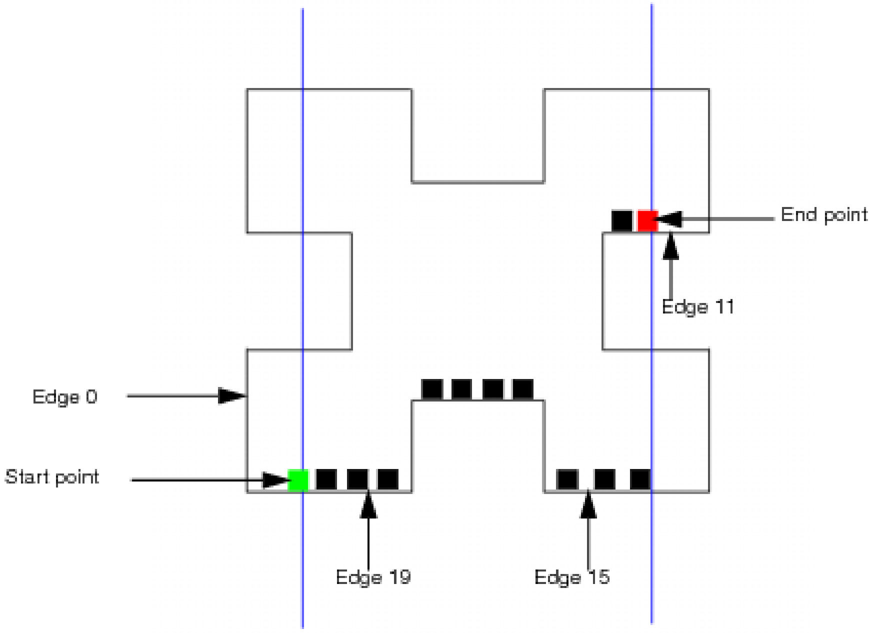
Assume that one edge of your design contains four pins (A1, A2, A3, and A4). You want to spread the pins evenly along the block edge between two sets of coordinates. To do this you must

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:

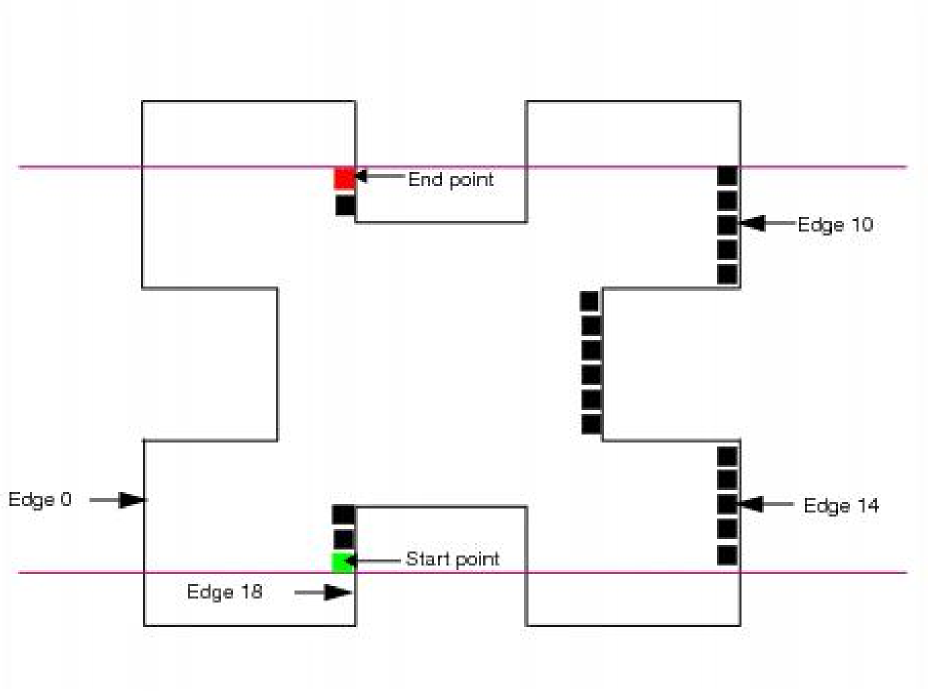

Original pin spacing
A1
A2
A3
A4
New spacing, with pins spread evenly between the two sets of coordinates 85.0, 0.0 and 130.0, 0.0
A1
A2
A3
A4
85.0, 0.0
100.0, 0.0
115.0, 0.0
130.0, 0.0

The following two figures shows how pins are spread along a side in case of rectilinear partitions where a side with multiple edges has been specified.
In the following case, the specified side is bottom, and the pins are, therefore, spread along the edges of the bottom side.



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

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



Pins spread from edge 18 to edge 6
(18, 14, 12, 10, 6)

Spreading Floating Pins

Floating pins are spread across all edges over the whole partition. The number of pins on each edge depends on the pin density. An edge with higher pin density will have lesser pins.

The following considerations are taken into account while spreading the floating pins:

- If some bits of buses have connections and the rest are floating, then the floating pins are kept together as a bus.
- If a bus is floating then it is placed together. If not then its pins are placed on the same edge.
- If some bits of a pin group have connections and the rest are floating, then the floating pins are not spread and the order of the pin group is maintained.
- In the master and clone scenario, both pair of nets (for pin connecting master and pin connected clone) should be floating to consider it as a floating pin pair.
- In the nested scenario, floating pins are not aligned across hierarchies.

Running Relative Floorplanning

This section describes how to use the *Floorplan* menu's *Relative Floorplan* form to capture and define the placement relationship of floorplan objects independently from the actual coordinates in a floorplan. The Relative Floorplan program provides a more flexible way to place objects, such as modules, blocks, groups, blockages, pin guides, pre-routed wires, and power domains. Block I/O pins can be used as reference objects but they cannot be relative objects. You can capture and define the placement relationship of floorplan objects independently from the actual coordinates in a floorplan. You can also resize a module or blackbox based on other floorplan objects, while maintaining its current area based on a specified width and height, a given dimension (width or height), and a target utilization value. You can also specify a wire's start or end point relative to the reference object's reference corner, or specify a wire's start or end point directly.

Before relative floorplanning, the design must be loaded into the current Innovus session.

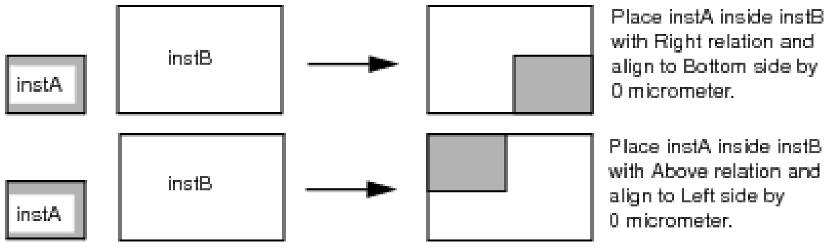
Orientation Key

The following table is a key to the orientation of placed objects:

Value	Definition
`R0`	No rotation
`MX`	Mirror through X axis
`MY`	Mirror through Y axis
`R180`	Rotate counter-clockwise 180 degrees
`MX90`	Mirror through X axis and rotate counter-clockwise 90 degrees
`R90`	Rotate counter-clockwise 90 degrees
`R270`	Rotate counter-clockwise 270 degrees
`MY90`	Mirror through Y axis and rotate counter-clockwise 90 degrees

Instance Place Example

The following figure shows an example of instance placement


instA
instB
Place instA inside instB with Right relation and align to Bottom side by 0 micrometer.
instA
instB
Place instA inside instB with Above relation and align to Left side by 0 micrometer.

Saving and Restoring Relative Floorplan

The Innovus software automatically saves all the executed menu and text commands for the relative floorplanning actions in the innovus.cmd file. To save all the relative floorplan commands that were executed during a session, click the *Save* button on the *Relative Floorplan* form. This saves a script that can be used later for updating or adjusting an existing floorplan based on the new blocks' size and position.

Saving and Loading Floorplan Data

You can save and load floorplan data at any time during a session.

- To save the floorplan information to a file, use the *Save FPlan File* form or the `saveFPlan` command.
- To load the floorplan information from a file, use the *Load FPlan File* form or the `loadFPlan` command.

Note: You can save and load floorplan data in the XML format.

The `saveFPlan` command saves net attributes including non-default attributes of a net, like weight, bottom preferred routing layer, or detour. This capability is especially useful in the prototyping flow where you need to save out the state of the floorplan, and timing-driven `proto_design` sets net weights to guide `proto_design` to a timing driven result. For large designs, as the designer iterates through different versions of the floorplan, they often check-point by doing a `saveFPlan`. A saved .fp file is also excellent for two people working separately on the same floorplan, so that .fp can be passed on to another designer who has a slightly different netlist. By saving the net attributes the `saveFPlan` and `loadFPlan` commands are able to bring back the entire prototyping/floorplanning picture.

Specific Floorplan Section TCL Export/Import

You can use the `writeFPlanScript` command to write out specified floorplan sections and source the output file after init_design. With this command Innovus writes a file that you can examine, edit, and then source to selectively (or completely) define your floorplan. The `writeFPlanScript` command writes out a file with TCL commands to recreate the floorplan. It can optionally write out specific floorplan sections listed with option `-sections`.

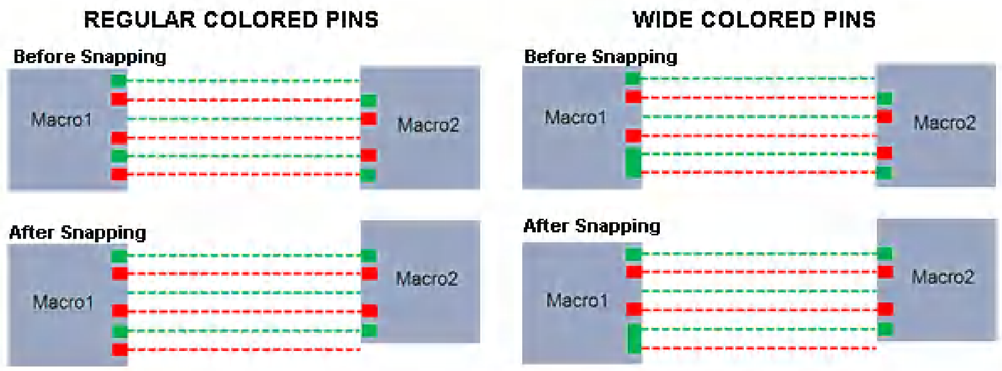
The benefit of this command is selective export/import of concise TCL for better command discovery and easy partial export/import of a portion of a floorplan between two designs. The human readable TCL floorplan file format completely captures the floorplan specification and is cut and paste source-able.

Snapping the Floorplan

The snap floorplan feature enables you to snap objects to the grid. You can use the `snapFPlan` command or alternatively use the *Snap Loaded Floorplan* form to align objects to the grid. Additionally, you can use the `snapFPlanIO` command to snaps I/O cells to a user-defined grid.

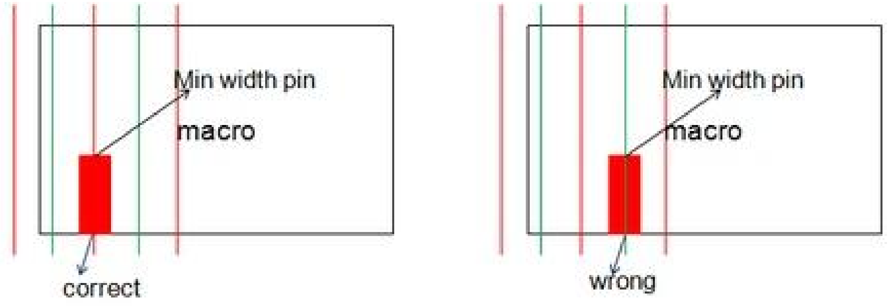
In floorplan flow, after CCE (`colorizeGeometry`) assigns color to macro pins and other objects. Using the `snapFPlan -macroPin` command you can snap macro pins to the correct color track. The `snapFPlan` command only snaps signal pins and places a macro with macro pins on color track. Innovus snaps the port on the lowest metal layer which is close to macro origin to the color track.

In addition to the macro pin color-aware snapping, Innovus also checks that the pins have default width and it uses the first default wide pin to snap. The non-default-width pin shapes are ignored while snapping on colored tracks and only the pins with default width are snapped.

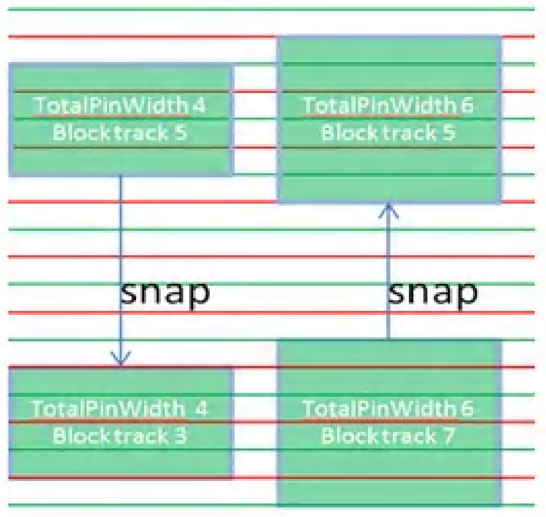


The `checkFPlan` command verifies if the macro pins are snapped to the correct color track. It reports a violation if the macro placer cannot find legal location or when macro pins have color violation.

- For min width pins, the `checkFPlan` command checks whether the pin center is snapped to the routing track with the same color.



- For fat pins, the `checkFPlan` command checks whether the pin center is snapped to the routing track and decides if the pin center should snap to the same or opposite colored routing track.



Innovus chooses the result which saves maximum routing track. It reports an error if a pin is not snapped to any track. It displays a warning if the fat pin snap is not correct. It gives details of:

- The violated fat pin layer and shape.
- The number of tracks the pin has occupied.
- How to modify the LEF file to save routing track.

Resizing the Floorplan

The resize floorplan feature enables you to resize a floorplan while maintaining the relative locations of the floorplan objects.

Normally, floorplan resizing is done

- to reduce the die size on a finished floorplan.
- to expand or shrink the die during floorplan creation, based on the full chip placement results.

You can set the global parameters for the `resizeFloorplan` command by using the `setResizeFPlanMode` command. The parameters that you specify with the `setResizeFPlanMode` command are then used automatically whenever you run the `resizeFloorplan` command to resize the floorplan. Alternatively, you can use the use the *Floorplan - Resize Floorplan* form in the Innovus GUI to perform the resize functions in the design.

Note: You can use the `getResizeFPlanMode` command to view the information about a specified `setResizeFPlanMode` parameter in the Innovus log file and in the Innovus console.

Resize Floorplan Options

The space among floorplan objects can be resized in three ways:

Proportional Spacing

Distributes the space among floorplan objects proportionally (`setResizeFPlanMode -proportional`). It can shrink or expand the space in both, X and/or Y directions. However, you cannot adjust pre-routed wires using proportional spacing.

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.

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`).

For information on resizing a floorplan, see the Resize Floorplan section in the*Floorplan Menu* chapter of the *Menu Reference*.

Setting Resize Lines

For performing shift-based resizing, you can specify one or multiple resize lines. You can use the `setResizeLine` command to set the the resize lines for shift-based floorplan resize option of the `resizeFloorplan` command. The resize lines can be non-continuous though they must be orthogonal. If resize lines are specified, the floorplan will be resized between the area specified by the resize lines (for shift-based floorplan resize). To shrink or expand the floorplan in the horizontal direction, specify a vertical resize line. To shrink or expand the floorplan in the vertical direction, specify a horizontal resize line.

Each resize line can have multiple segments (for example, horizontal, vertical, and again horizontal) but it can only be applied to one direction. If you want to shrink or expand the floorplan in both directions, specify resize lines for each direction. You can create multiple resize lines. The resize lines are removed once the `resizeFloorplan` command is run.

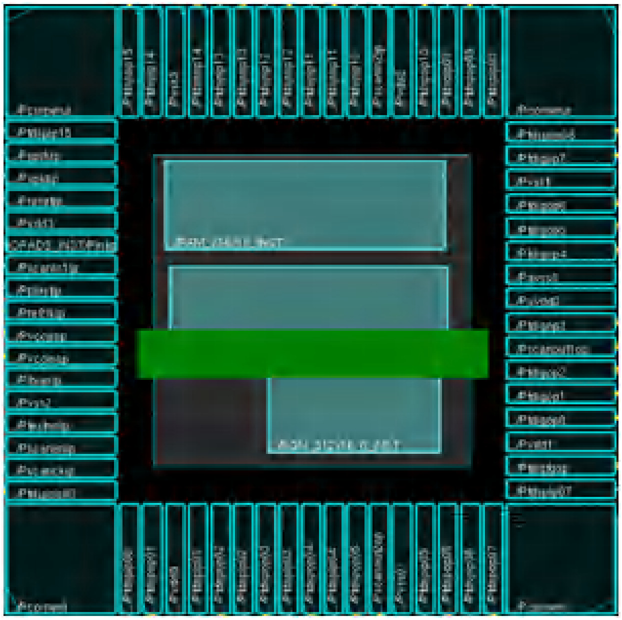
Note: You can use the `addSizeBlockage` command to add a size blockage object that controls the behavior of the resize line under a covered area. The object area prevents floorplan objects in this area from being resized and maintains the alignment and minimum space between these objects during floorplan resize.

Specifying Resize Directions

Resizing can be done in X and Y directions. Positive values mean expanding the space and negative values indicate shrinking. However, Innovus does not support a scenario where resizing line by expanding and shrinking, both occur on the same direction.

For example, the following command specifies a resize line in horizontal direction with a resize width of 100 microns:

```
setResizeLine -direction H -width 100 (279.2845 555.443) (1030.896 555.443)
```



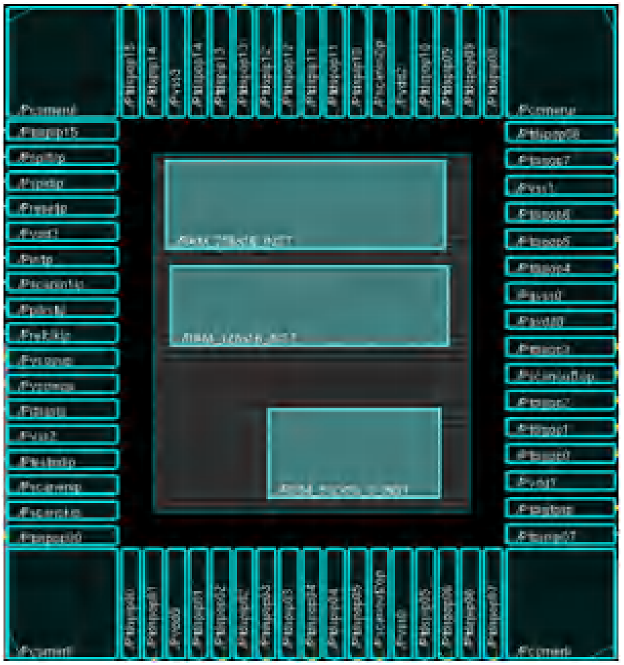
The following command again resizes the floorplan in the same X direction with a negative value of -200 microns:

```
resizeFloorplan -xSize -200
```

Innovus displays a warning message in such a situation.

The following command resizes the floorplan in the same Y direction with the same positive value of +100 microns:

```
resizeFloorplan -ySize +100
```



Snapping Resize Values

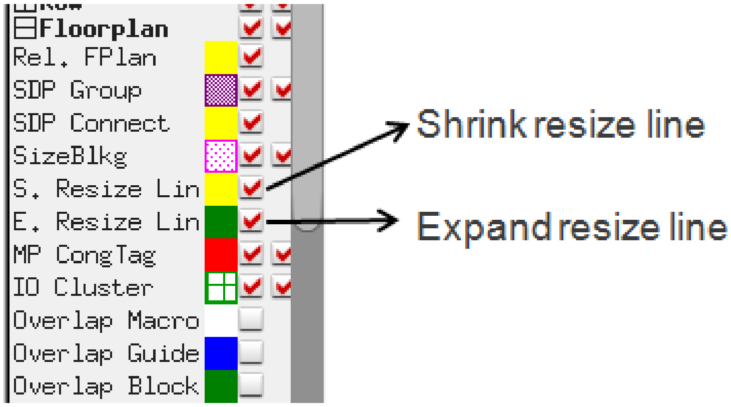
The resize values (shrink/expand) can be snapped to a multiple integer of the metal layer pitch.

Note: Specify the `setResizeFPlanMode -snapToTrack` option to snap resize values.

For example, if the horizontal metal pitch is 1.5 microns and you want to shrink the floorplan by 8 microns in y direction, the actual shrink value is 7.5 microns, the nearest multiple integer of the metal pitch.

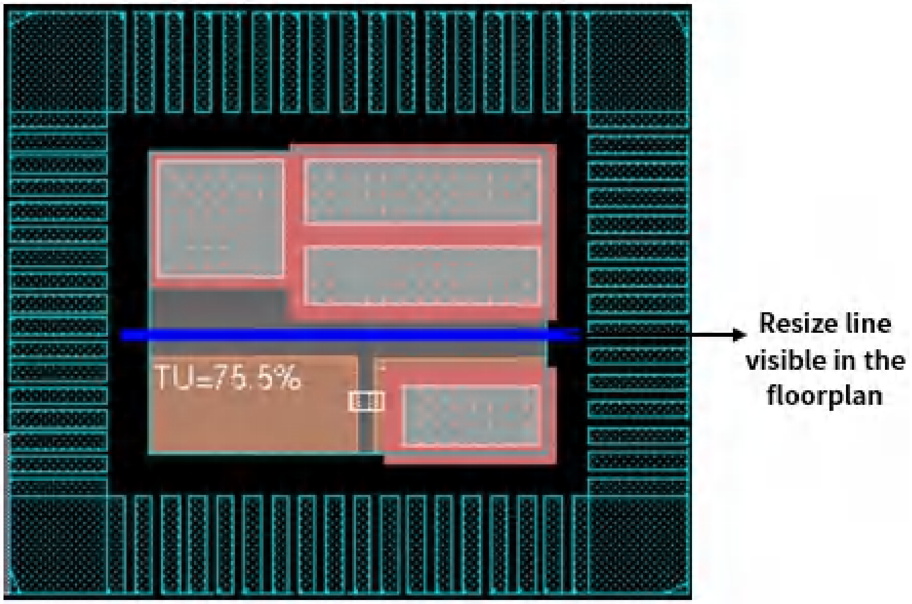
Viewing Resize Lines using Color Preferences

Once you specify the resize lines to perform shift-based resizing, you can display the resize lines in Innovus by setting the *Resize Line* option in the *Floorplan*.



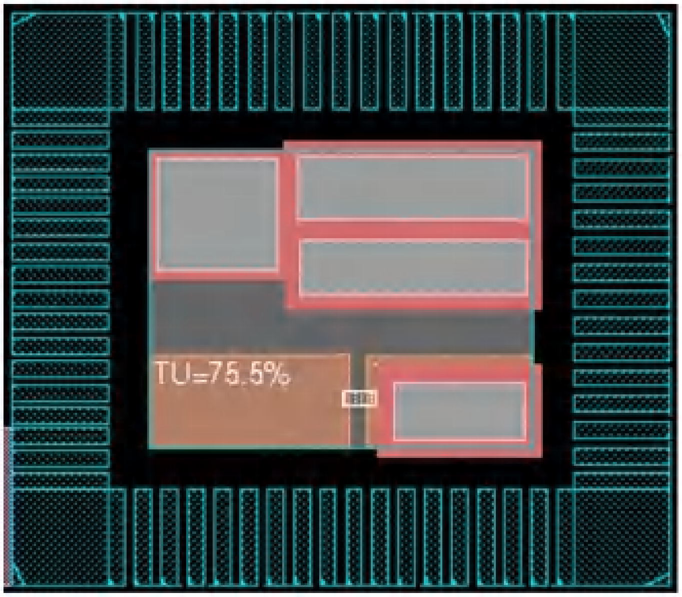
However, the resize lines disappear once you resize the floorplan.

Example: Setting a resize line before resizing the floorplan



Resize line visible in the floorplan

Example: The resize line disappears after resizing the floorplan in shift based mode

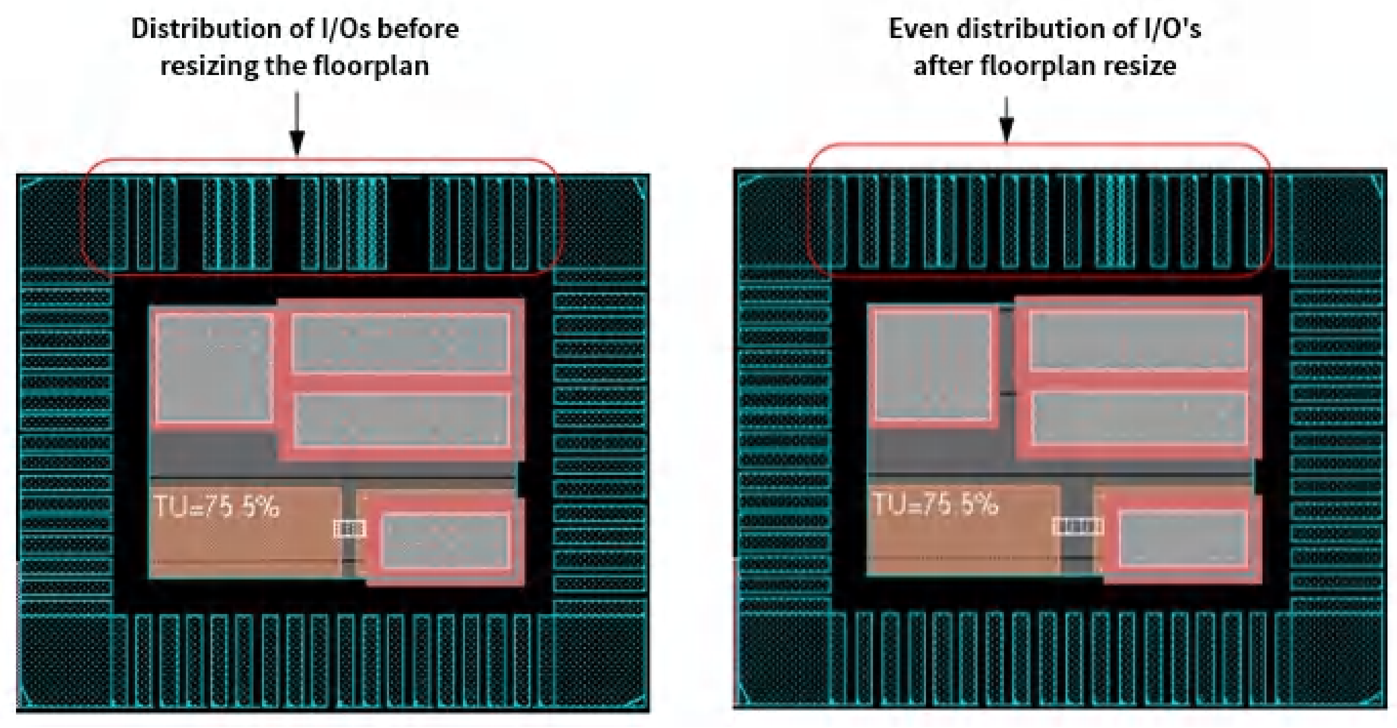


Note: During resizing, the target size may not be achievable. You have to force resize to meet the target size as much as possible, using the `resizeFloorplan -forceResize` option.

Distributing I/Os using Resize Floorplan

You can use the `setResizeFPlanMode -ioproportional` parameter 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.


Distribution of I/Os before resizing the floorplan
TU=75.5%
Even distribution of I/O's after floorplan resize
TU=75.5%

Resizing Floorplan Bounding Box in GUI

You can create a rectilinear floorplan by dragging edges of the bounding box. You can drag and edit all edges of a floorplan including cutouts without moving the other objects. In case of new violations, such as placed objects out of core or I/O pins out of core or routing grid, Innovus will display a violation and suggest a solution to user.

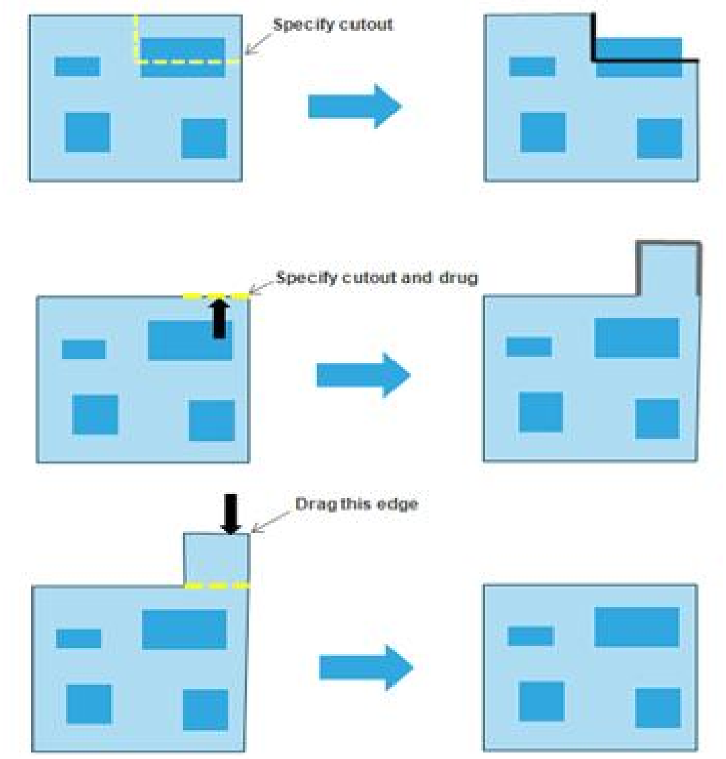
Selecting and Dragging an Edge

You can change the top-cell bounding box and cutting rectangle floorplan to rectilinear shape. IO box /die box/core box will be automatically updated after floorplan reshaping. Innovus cuts an edge at the point you specify.

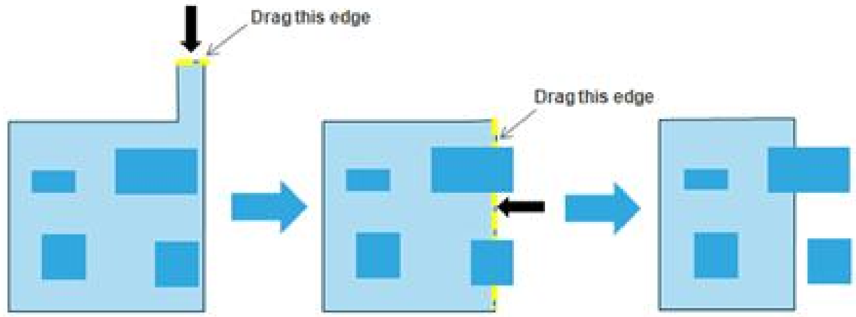
1. Innovus allows you to create/manipulate cutouts. You can cut rectangle floorplan to rectilinear shape and Innovus will automatically cut an edge at the point specified by you. After cutting,

the edge will be divided into two segments and the new segment is editable. You can drag this segment to shrink or expand freely.

2. If you move the selected edge, you can make existing cutout to disappear or “reverse” edge direction.
3. You can create cutouts and then specify the length of each new edge, the cutouts may disappear you merge two edges as one.



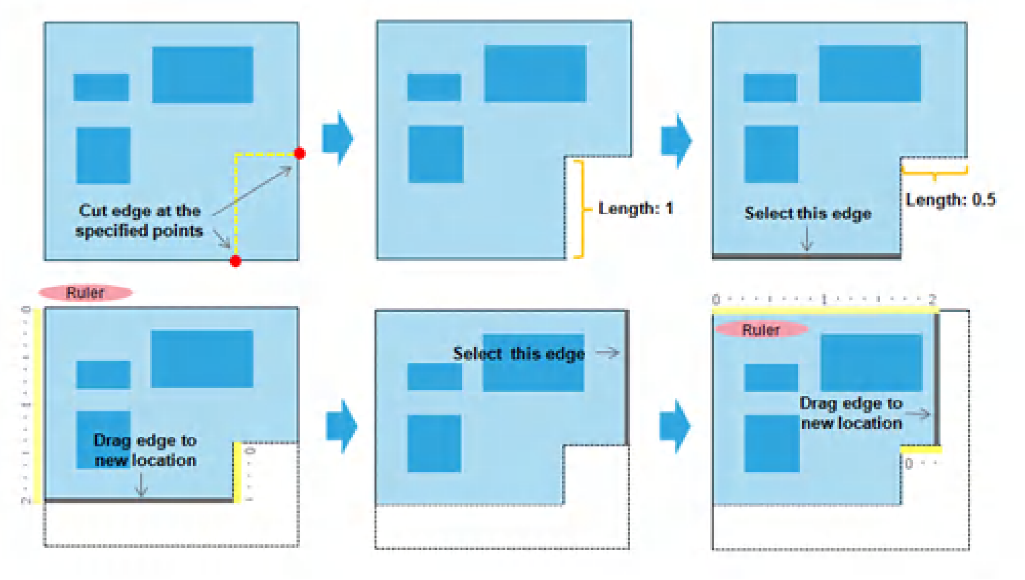
You can drag any edge of the floorplan boundary and edit each edge to reshape the rectilinear floorplan.



Moving Edge and Setting Length of the Specified Edge

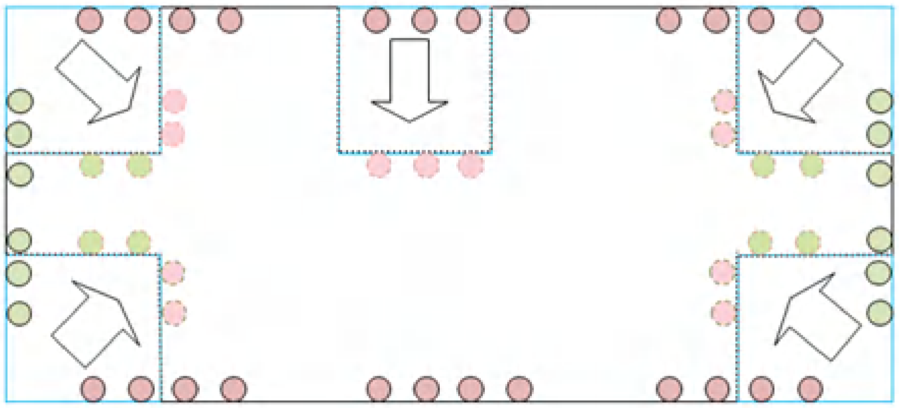
Innovus displays the length of the selected edge in the ruler when you drag the edge to reshape. This allows you to change the top-cell bounding box and cutting rectangle floorplan to rectilinear shape. To set the length for an edge:

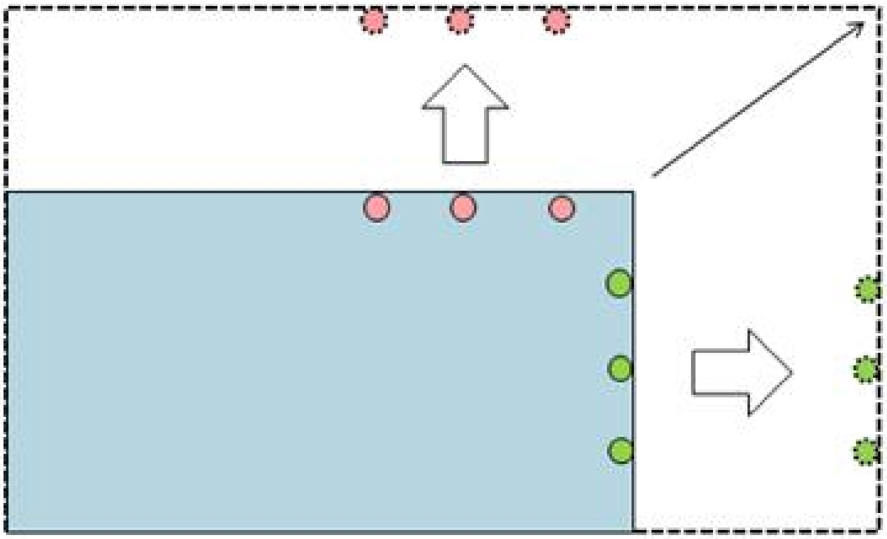
1. Specify a point at the boundary edge, and cut the edge to make a cutout.
2. The length of the new edge (vertical) in the figure below is 1.
3. You can select an edge and drag the edge to a new location. The length of the new edge (horizontal) in the figure below is 0.5.
4. Using the ruler, you can drag the specified edge and set its length according to the ruler.
5. You can select another edge to edit and using the ruler, you can drag the specified edge and set its length.


Cut edge at the specified points
Length: 1
Select this edge
Length: 0.5
Ruler
Drag edge to new location
Select this edge
Ruler
Drag edge to new location

Movement of I/O Pins with Edge

When you move an edge while reshaping a floorplan, I/O pins are moved with the edge. Innovus keeps the same position of the pin (like a rectilinear cut) on the new position of the edge (snapping pin). Also, the side and the sequence of the I/O pin remains the same as in the old floorplan, while the pins get distributed according to the new position of the edge.

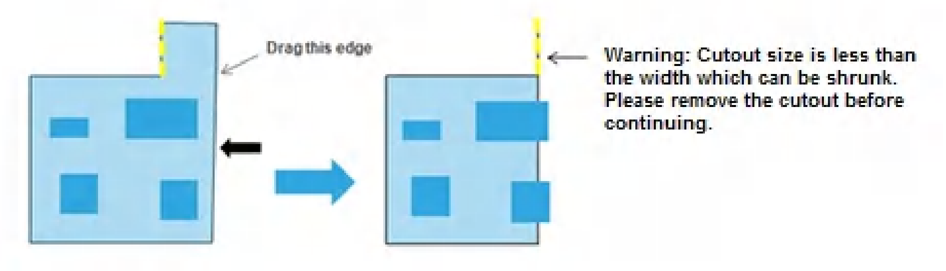




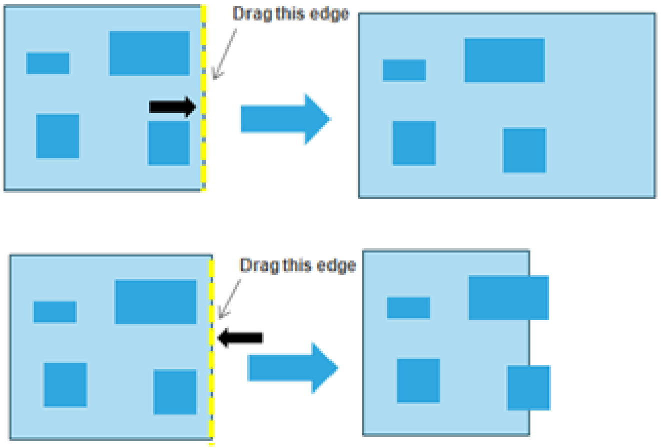
Innovus does not honor pin constraints. The I/O pins which cannot find a legal location, after floorplan is reshaped, are unplaced.

Automatic Checking

Innovus stops resizing when the cutout size gets lower than the predefined threshold.


Drag this edge
Warning: Cutout size is less than the width which can be shrunk. Please remove the cutout before continuing.

When you drag an edge of a floorplan, all objects remain untouched. You can shrink or expand the edge. However, Innovus creates violations if the object is outside the floorplan boundary after resizing. It issues a warning message to let you fix the violation. You can run `checkFPlan` before/after and capture the delta.



Undo/redo Capability

You can always reverse your changes after reshaping a floorplan.

Checking the Floorplan

After creating the floorplan, you can check the floorplan to identify problems before a design is passed to downstream tools. You can check the bus guide, routing grid, placement, pins, power domains, partition clone alignment, feedthrough buffer insertion and narrow channel etc.

checkFPlan

Use the `checkFPlan` command to check the quality of the floorplan to detect potential problems before the design is passed on to other tools. This highlights the cells in the design display area with violation markers, where applicable. Use this command after specifying the floorplan data or running an initial floorplan. Run the `checkFPlan` command on your final floorplan file. Alternatively, you can use the *Check Floorplan* form.

checkDesign

Use the `checkDesign -floorplan` command to check the floorplan, generate error or warning messages, and report the following information:

- Off-grid horizontal and vertical tracks
- Instances not snapped to row site
- Unplaced I/O pins
- Off Grid Power/Ground Pre-routes
- Instances not on the manufacturing grid
- Preroutes not on the manufacturing grid (error)
- Regular preroutes not on tracks

check_design

Use the `check_design` command to check the preconditions for major flow steps before they are run. It provides complete, actionable information about design problems that result in low quality or incorrect results when you start the flow. You can use the `check_design -type place` command option to check for macros that are not fixed prior to placement. The `check_design` command displays a message and reports the macros that are not fixed. You must resolve the issue by updating the `place_status` to fixed before running placement.

checkFPlanSpace

Use the `checkFPlanSpace` command to check the floorplan for spacing rule violations and save the violation information in a report. A marker is displayed at each violation spot. Alternatively, use the *Check Floorplan Space* form to check the floorplan for spacing rule violations.

adjustFPlanChannel

Use the `adjustFPlanChannel` command to automatically adjust the channel width between objects to prevent potential routing congestion. You can use this command after importing the design, loading a completed floorplan, and then running the `place_design` and `earlyGlobalRoute` commands.

Finishing the Floorplan

Finishing the floorplan involves performing advanced placement-related refinements to a floorplan, to produce a more polished floorplan. The `finishFloorplan` command, can be used on any floorplan, including third-party generated floorplans. You can use the `setFinishFPlanMode` command to set the active objects and specify the channel direction for the `finishFloorplan` command to use. The `setFinishFPlanMode` command affects the behavior the `finishFloorplan` command.

Note: You can use the `getResizeFPlanMode` command to return the current settings for the `setFinishFPlanMode` command.

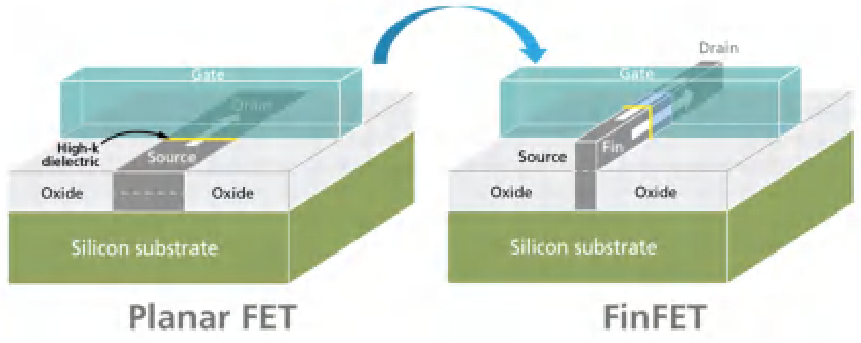
The following commands specify the region layer name list for creating a region layer for the specified DRC region object:

```
setFinishFPlanMode -drcRegionObj macro
finishFloorplan -drcRegionLayer R1
```

FinFET Technology

FinFETs are 3D structures that rise above the planar substrate, giving them more volume than a planar gate for the same planar area. Given the excellent control of the conducting channel by the gate, which wraps around the channel, very little current is allowed to leak through the body when the device is in the off state. This allows the use of lower threshold voltages, which results in optimal switching speeds and power.

In a FinFET, the FET gate wraps around three sides of the diffusion fin as shown below. This forms conducting channels on three sides of the vertical fin structure. This approach provides much more control over channel current compared to planar transistors. Multiple fins can be used to provide more current.



FinFETs also promise to alleviate problematic performance versus power tradeoffs. Designers can run the transistors faster and use the same amount of power, compared to the planar equivalent, or run them at the same performance using less power. This enables design teams to balance throughput, performance and power to match the needs of each application.

FinFET Support in Innovus

Ideally origin of all placeable objects (standard cells, I/o pads and blocks) must be aligned to FinFET girds to be manufacturable for the FinFET technology. FinFET library rules are used to define FinPITCH in the technology LEF file to support the FinFET grid. The FinFET pitch is used to define the legal Y/X values of the placement grid in a design. Innovus honors the FinFET grid and automatically checks for the FinFET definition. If it detects a FinFET definition it enables support and sets the placement grid as FinFET grid by default. It also displays a message informing you about the pitch, offset, and the direction values.

Defining FinFET Rule using LEF 5.8

FinFET pitch can be defined in LEF 5.8 to support FinFET grid. User can define fin pitch, offset and the direction of fin grid in PROPERTYDEFINITIONS of technology LEF file. The FinFET pitch is used to define legal Y (or X) values of placement grid in a design, while the other placement grid is typically derived from Metal1 grid. The FinFET grid must be a multiple of manufacture grid and one FinFET grid per design.

You can define a FinFET rule using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LEF58_FINFET STRING
"FINFET
     PITCH pitch [OFFSET offset] {HORIZONTAL|VERTICAL}
     ;" ;
```

```
END PROPERTYDEFINITIONS
```

```
Where:
FINFET
  PITCH pitch [OFFSET offset] {HORIZONTAL | VERTICAL}
```

Specifies the FinFET pitch to be `pitch`, which starts at the `offset`, if specified, or zero from the origin of the design.

The HORIZONTAL/VERTICAL keywords mean that the FinFET pitch is used to define the legal Y/X values of the placement grid that all cells, blocks, and IOs must align to (specifically, the origin of every cell must be aligned to the legal Y/X values), in order to guarantee all the FinFETs inside are aligned properly. The X/Y value of the placement grid is derived from the standard cell site width and only applies to standard cells. The cell row height should typically be multiple of the FinFET pitch.

Type: Floats, specified in microns

Note: [OFFSET offset] is not supported yet.

Finfet Rule Examples

The following example indicates that the FinFET y pitch is 0.108 µm:

```
PROPERTYDEFINITIONS
    LIBRARY LEF58_FINFET STRING "
    FINFET
    PITCH 0.108 HORIZONTAL ; ";
END PROPERTYDEFINITIONS
```

Snapping Physical and Floorplan Objects to FinFET Grid

With the introduction of FinFET grid, all of the placeable objects such as I/O pads, macros and standard cells (cell rows) must be placed on the FinFET grid defined in technology LEF file. The height of all placeable objects must be a multiple of FinFET grid and the origin of these objects must be snapped to the FinFET grid. If FinFET PITCH is defined in technology LEF file, Innovus sets the placement grid as FinFET grid by default and checks for any violations.

Note: The *FinFET* Grid options in available on the *Layer Control* bar. You can use it to control the visibility of the FinFET grid. Additionally, you can specify the color preference for the FinFET grid using the *FinFET Grid* option on the *Color Preferences* form.

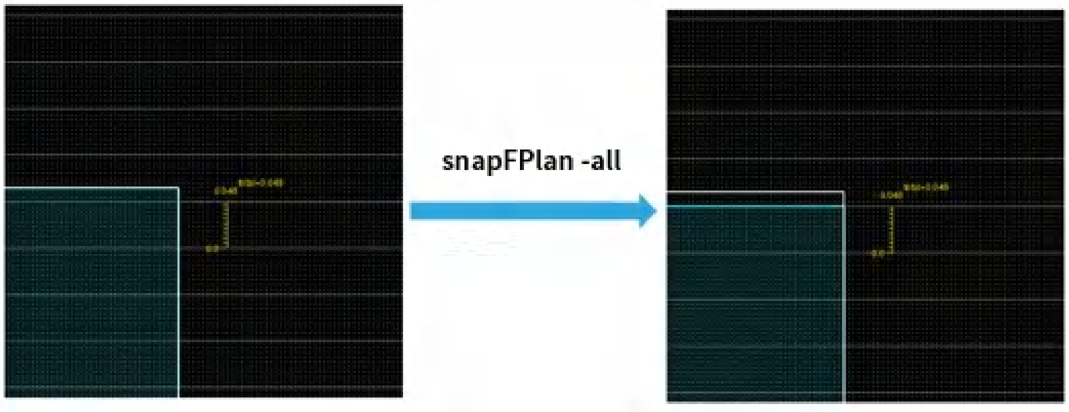
The following floorplan commands support this functionality:

snapFPlan

The `snapFPlan` command snaps blocks, I/O pads, and standard cells to FinFET grid. It honors user specified snapping rule for x direction. For y direction, it honors the FinFET grid (when the FinFET grid is horizontal in regular horizontal row design).

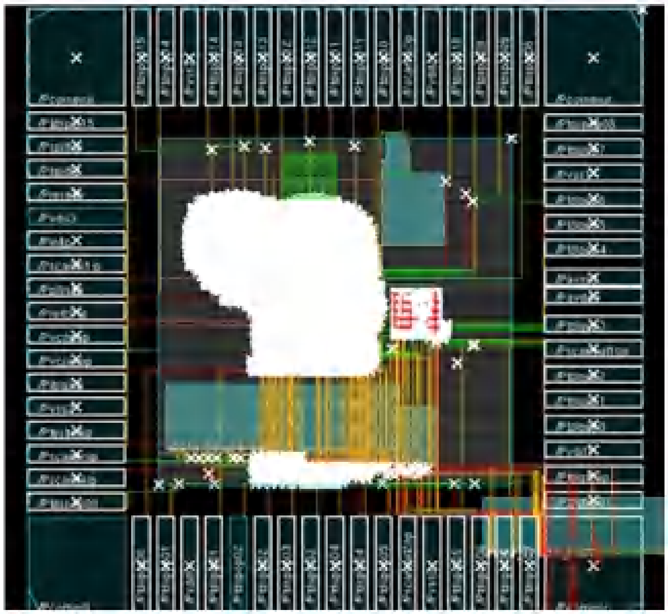
The SITE definition is as follows:

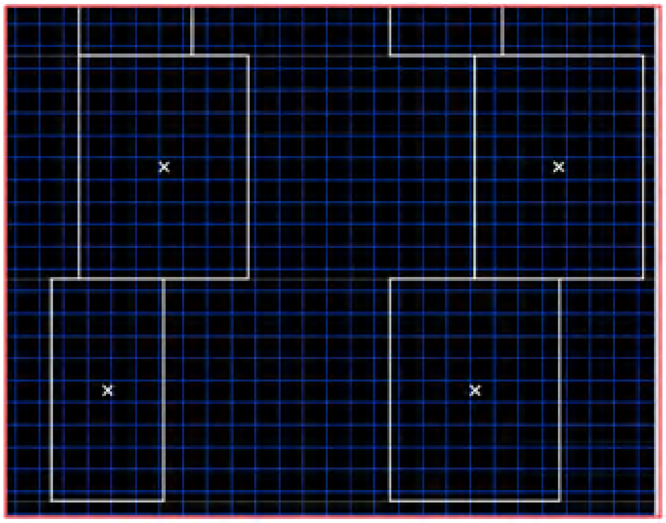
```
LIBRARY LEF58_FINFET STRING "
        FINFET
        PITCH 0.048 HORIZONTAL ; " ;
END PROPERTYDEFINITIONS
```



checkFPlan

The `checkFPlan` command checks that all supported objects (die box, core box, standard cell rows, I/O rows, blocks, IO pads, and standard cells) are aligned with the FinFET grid and reports DRC violations. All supported objects which are not aligned to FinFET grid are marked in the GUI and reported in the Violation Browser.





checkDesign

The `checkDesign -floorplan` command checks the floorplan and reports any objects not aligned with the FinFET grid correctly.

Note: In a top-down flow, for any partition shape (example, fence and power domain), the height of die box/core box must be a multiple of FinFET grid and all the corners of partition shape must be snapped to the grids.

Unified Floorplan Constraints

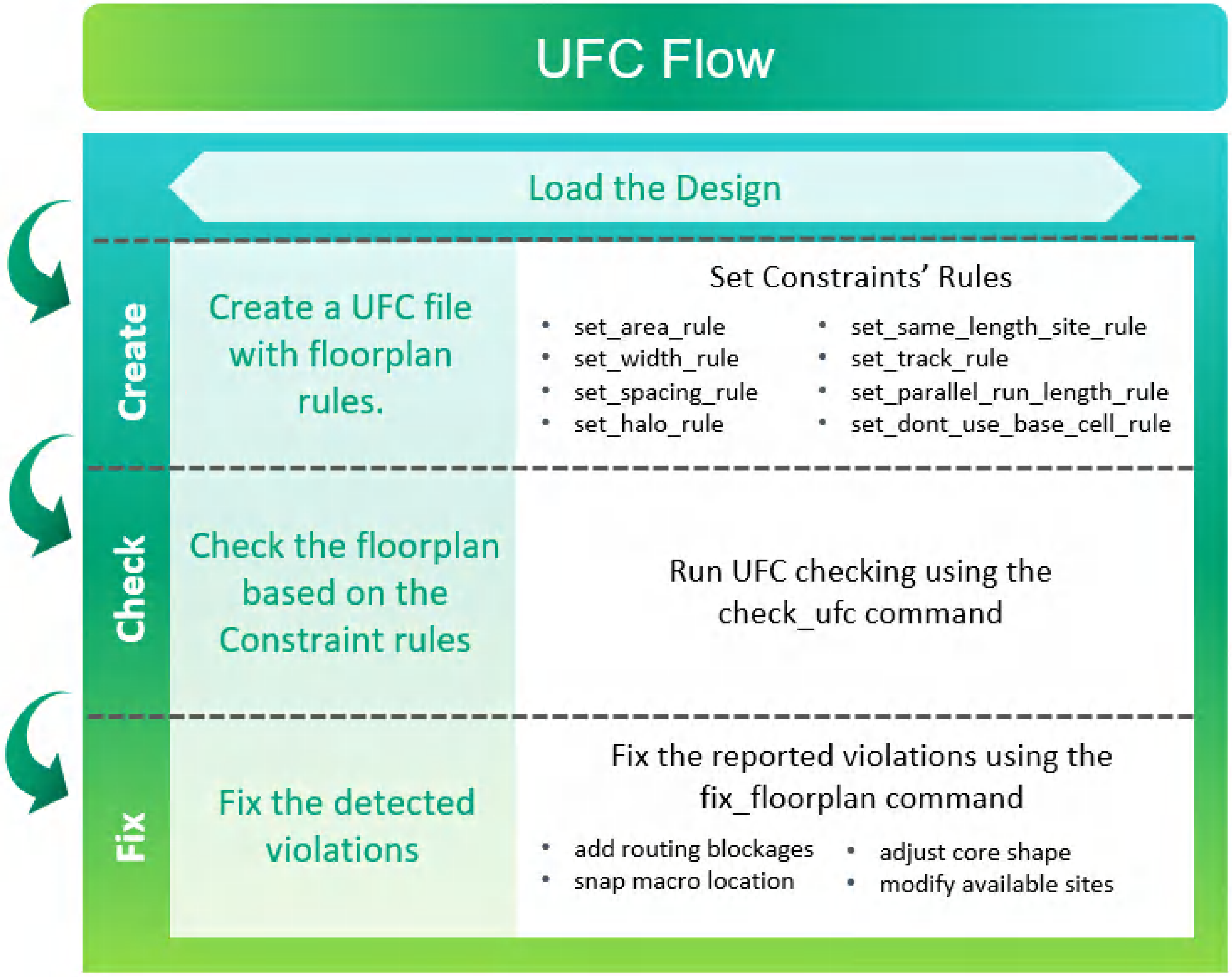
The Unified Floorplan Constraints (UFC) methodology is a powerful Innovus capability that can define several rules on floorplan objects, check specified rules, and report the violations with rule name as well as severity. The violations can be shown in the violation browser. UFC also provides the ability to automatically fix specific violations. It can add routing blockages, snap macro location, adjust core shape to fix corresponding violations. In addition, UFC also provides the ability to exclude rules from checking for specified objects.

The UFC capability provides a method for you to easily detect floorplan geometries in the early design stage so that they can be fixed accordingly. This helps to save the design runtime.

Recommended UFC Flow

The recommended UFC flow involves the following steps:

- Loading the Design.
- Creating a UFC file with floorplan rules.
- Checking UFC rules.
- Fixing reported violations.


UFC Flow
Load the Design
Create
Create a UFC file with floorplan rules.
Set Constraints' Rules
set_area_rule
set_width_rule
set_spacing_rule
set_halo_rule
set_same_length_site_rule
set_track_rule
set_parallel_run_length_rule
set_dont_use_base_cell_rule
Check
Check the floorplan based on the Constraint rules
Run UFC checking using the check_ufc command
Fix
Fix the detected violations
Fix the reported violations using the fix_floorplan command
add routing blockages
snap macro location
adjust core shape
modify available sites

Creating a UFC file with Floorplan Rules

A UFC file defines several rules set on floorplan objects using the supported UFC commands. A UFC file is saved with the .ufc extension. You require the following to create a UFC file:

- UFC Rules
  These are rules that can be applied on floorplan objects. You can use the UFC commands for setting the UFC rules with different rule names.
  Note: For more information on the UFC commands that you can use to define UFC rules in a

UFC file, see "Supported UFC Commands" section of the Syntax and Scripts chapter of the *Innovus User Guide*.

- Supported UFC Objects
  The UFC rules are applied on supported objects that can be used as TCL commands. These objects return shapes that are used by the UFC commands for checking rules.
- Supported UFC Dimension Functions
  The UFC rules are applied on the supported objects using specified dimension functions such as max, min and so on.

After creating the UFC file with floorplan rules, you can use the `check_ufc` command to check the specified rules and report the constraint violations. You can then use the `fix_floorplan` command to fix the violations.

UFC Rules

The UFC commands allow you to set the following rules:

Rule Name	Description	UFC Command to Use
Area rule	Specifies the area constraints	`set_area_rule`
Width rule	Specifies the width constraints. These constraints include simple width, jog width, edge width between incorners, width between two concave corners.	`set_width_rule`

Spacing and enclosure rule	Specifies the spacing constraints. These constraints include horizontal, vertical, orthogonal spacing with minimum parallel run length, and any-angle spacing. This rule can honor cell orientation.	`set_spacing_rule`
Merge and reshape spacing rule	Specifies spacing between merged and reshaped objects. This rule is used to check the spacing between merged objects.	`set_merge_and_reshape_spacing_rule`
Halo rule	Specifies the enclosure of inner and outer edge by side. The sides can be all, vertical, horizontal, top, bottom, left, right. It also has the capability to check if one object always surrounded by another specified object. This rule can honor cell orientation.	`set_halo_rule`
Same length site rule	Specifies the min number of continuous poly stripes.	`set_same_length_site_rule`
Routing track rule	Specifies the allowed location of tracks with specified layer, mask, and direction.	`set_track_rule`

Parallel run length rule	Specifies the constraints between objects with parallel run length or project length.	`set_parallel_run_length_rule`
White_area extension rule	Specifies the object extension to calculate white_area object. The default white_area is the core area except available_sites and macros.	`set_white_area_extension_rule`
Reshape available sites rule	Recalculates the available_sites or available_sites_no_abut by skipping the specified cells. It takes the row sites under the cells into account.	`set_reshape_available_sites_rule`
Dont use libcell rule	Detects if the specified libcell is used in the design.	`set_dont_use_base_cell_rule`
Exclude rule	Specifies that the rules should excluded.	`exclude_rule`

Note: For more information on the UFC commands that you can use to define UFC rules in a UFC file, see "Supported UFC Commands" section of the Syntax and Scripts chapter of the *Innovus User Guide*.

Supported UFC Objects

The UFC rules use supported objects that can be used as TCL commands. These objects return shapes that are used by the UFC commands for checking rules.

The following are the objects supported by the UFC rules. These can be used as TCL commands:

Supported Object	Corresponding TCL command	Description

design_boundary	`dbGet top.fplan`
	`get_db designs .boundary`
		Design window

design_shape	`dbGet top.fplan.boxes`	Design shape
core_boundary	`dbGet top.fplan.coreBox`	Region to place site/row.
core_shape		Core shape
available_sites	`get_db rows .rect`	Sites able to place standard cells. Row site areas will be merged if abutted.
available_sites_no_abut	`get_db rows .rect`	Sites able to place standard cells. Row site area will not be merged if abutted.
insts	`get_db insts`	Instances.
phys_insts	`get_db phys_inst`	Physical instances
base_cells	`get_db insts -if {.name == <base_cell_name_pattern>}`	Specify the instance by their base cell name.
macros	`get_db insts -if {.base_cell.class == block*}`	Hard macros.

place_blockages	`get_db place_blockages`	Placement blockages.
route_blockages	`get_db route_blockages`	Blockage preventing specified routing layer existed in covered area.
markers	`get_db markers`	OBS layer and route blockage on this layer
white_area		Die area except available_sites and macros
ios	`get_db io_constraints`	IO pads

The UFC methodology supports using commands with conditional expression to flit objects.

```
insts |base_cells | macros | available_sites | design_boundary | place_blockages |
route_blockages | …
```

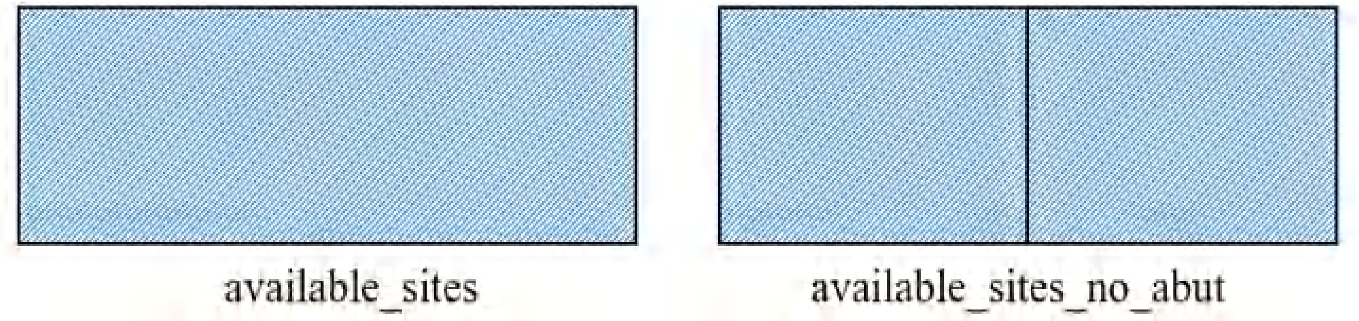
```
-if {expression of <attribute> in DB Tcl format}
```

For example,

- `macros -if {.name==MEM256x*}`
- `insts -if {.base_cell.class==block*}`
- `route_blockages -if {[regexp Metal(2|4|6|8) .layer.name]}`
- `available_sites {.width >= 20.0}`

Note:

- If there are two individual row site areas but they abut, the available_sites object takes them as one row site area and the available_sites_no_abut object takes them as two row site area. The available_sites object will merge abutted row site areas together. The following diagram shows the difference between available_sites and available_sites_no abut.



- The rules specified on the base cells should be able to override the same rules on macros or insts.

  For example, consider two rules R0 and R1 were,

  - Rule R0 defines macro to macro spacing should ≥ 10
  - Rule R1 defines cell A to macro spacing should ≥ 5

  When spacing from cell A (macro) to another macro is 6. R0 should not flag cell A.
- The dimension constraints set using the supported UFC objects are handled using the supported dimension functions. For more information, see Supported UFC Dimension Functions

Supported UFC Dimension Functions

The UFC rules are applied on the supported objects using specified dimension functions such as max, min and so on. The UFC checker supports the following dimension functions for checking dimensions:

Supported Function	Example	Description	Handler Convention

`Minimum`	`{.width > 10.0}`	Measured dimension	**Area:** .area
`Maximum`	`{.width < 20.0}`		
`Grid and offset`	`{on_grid (.width – $offset,$pitch)}`	Grid and offset denote the dimension must be: `$pitch*N + $offset` where, N can be any integer and N≥0	
`Forbidden list`	`{!.width in {10.0 20.0}}`	TCL list of forbidden dimensions	
`Forbidden zone`	`{!(71.0 <= .width && .width <=102.0)}`	TCL list of {min max} pair of forbidden zone	
`White list`	`{.width in {11.0 21.0}}`	TCL list of white dimensions.	
`White zone`	`{.width < 12.0 || (71.0 <= .width && .width <= 102.0} || 131.0 <= .width)`	TCL list of {min max} pair of white zone	

Note:

- For different rules, use different handlers. For example, `.width` refers to width, `.spacing` refers to spacing, etc.
- All values must be specified as float/real numbers.
- Multiple constraint functions should be applied at the same time.
- The `White list` and `Forbidden list` functions are mutually exclusive. They cannot be applied together. Similarly, the `White zone` and `Forbidden zone` functions are also mutually exclusive. However, the `White list` function can be used with the `Forbidden zone` function.

Sample UFC File

The following is a sample UFC file that defines several rules on floorplan objects. A UFC file can be used to check rules and detect floorplan geometry violation in the early design stages.

### DTMF.ufc

#set_area_rule

```
set_area_rule -name CORE.A.1 -obj [available_sites] {.area >= 8000} -severity 
GUIDELINE\
     -description {The area of available sites should be greater than 8000.}
```

#set_width_rule

```
set_width_rule -name CORE.W.1 -obj [available_sites] -type simple \
       -direction horizontal {on_grid(.width-0.66,1.32)} -severity ERROR-01\
       -description {The simple width of available sites in horizontal direction must 
be 1.32*n + 0.66.}
```

#set_spacing_rule

```
set_spacing_rule -name CORE.SP.1 -obj [available_sites] -ref [available_sites] -
direction vertical \
       -parallel_run_length -2.0 {on_grid(.spacing-0.66,1.32)} -severity ERROR-02\
       -description {The horizontal spacing between sites area must be 1.32*n + 0.66, 
if their PRL is greater than -2.0}
```

#set_enclosure_rule

```
set_spacing_rule -enclosure -name CORE.EN.1 -obj [design_boundary] -ref 
[available_sites] \
      -direction horizontal {on_grid(.spacing-0.66,1.32)} -severity ERROR-02\
      -description {The available sites to design boundary spacing in horizontal 
direction must be 1.32*n + 0.66.}
```

```
set_spacing_rule -enclosure -name MACRO.EN.1 -obj [design_boundary] -ref [macros] \
      -direction vertical {.spacing >=80.0} -severity ERROR-02\
      -description {The macro to design boundary spacing in vertical direction must be
greater than 80.0.}
```

#set_halo_rule

```
set_halo_rule -name MACRO.HALO.M4 -obj [route_blockages -if {.layer.name == Metal4}] -
ref [base_cells \
    -if {.name == ram*}] -type inner -side all -detect_ref_enclosed {.halo <= 10.0} -
severity ERROR-03 \
    -description {The inner distance of M4 routing halo on Metal4 enclosure ram in all
direction must be smaller than 10.0. }
```

#set_same_length_site_rule

```
set_same_length_site_rule -name CORE.SL.1 -min_site 20 -site_extension 0.5 -severity
ERROR-04 \
      -description {The same length site must be greater than 20 with 0.5 site
extension.}
```

Checking UFC rules

Checking the UFC rules involves checking the floorplan against the user specified UFC rules defined in the UFC file and reporting violations. You can use the `check_ufc` command to check the UFC rules. The reported violations can be seen in the *Violation Browser*.

Example:

The following command checks the floorplan based on the *DMTB_ENC.ufc* file that contains the user specified UFC rules:

```
check_ufc DMTB_ENC.ufc
```

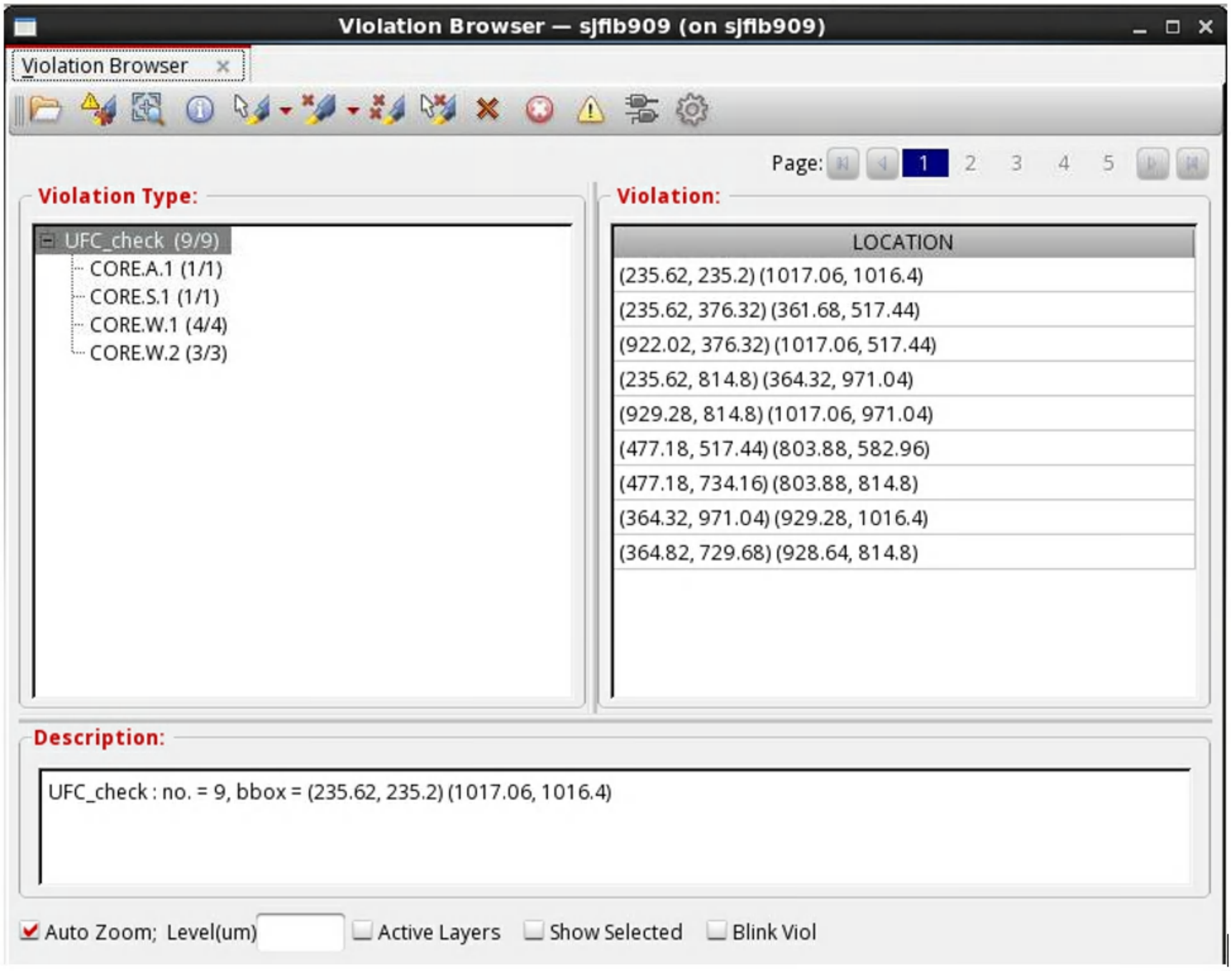
The input *DMTB_ENC.ufc* file:

```
set_area_rule -name CORE.A.1 -obj [available_sites] {.area < 500} \
  -description {The area of available sites should be lesser than 500}
set_width_rule -name CORE.W.1 -obj [available_sites] -direction horizontal -type simple
{.width > 200} \
  -description {The horizontal simple width of available sites must be larger than 200}
set_width_rule -name CORE.W.2 -obj [available_sites] -direction vertical -type simple
{.width > 100} \
  -description {The vertical simple width of available sites must be larger than 100}
set_spacing_rule -name CORE.S.1 -obj [base_cells -if {.name ==ram_256x16A}] -ref
[base_cells -if {.name == rom_512x16A}] -direction vertical {on_grid (.spacing-
0.1,0.2)} \
  -description {The vertical spacing between macro ram_256x16A and rom_512x16A must be
0.2*n + 0.1}
set_spacing_rule -name CORE.S.2 -obj [base_cells -if {.name ==rom_512x16A}] -ref
[base_cells -if {.name == ram_128x16A}] -direction vertical {.spacing > 10} \
  -description {The vertical spacing between macro rom_512x16A and ram_128x16A must
larger than 10}
```

The report summary of UFC checking after running the `check_ufc` command:

```
--------------------------Summary of UFC Checking ---------------------------------------
--
RuleName Count  Description
              ViolatedObjects
........ .....  ............................
             ...........................
CORE.A.1   1    The area of available sites should be lesser than 500
             available_sites
CORE.S.1   1    The vertical spacing between macro ram_256x16A and rom_512x16A must be
0.2*n + 0.1   DTMF_INST/RAM_256x16_TEST_INST/RAM_256x16_INST
CORE.W.1   4    The horizontal simple width of available sites must be larger than 200
             available_sites
CORE.W.2   3    The vertical simple width of available sites must be larger than 100
             available_sites
```

The violations can be seen in the *Violation Browser:*



UFC_check (9/9)
- CORE.A.1 (1/1)
- CORE.S.1 (1/1)
- CORE.W.1 (4/4)
- CORE.W.2 (3/3)

LOCATION
(235.62, 235.2) (1017.06, 1016.4)
(235.62, 376.32) (361.68, 517.44)
(922.02, 376.32) (1017.06, 517.44)
(235.62, 814.8) (364.32, 971.04)
(929.28, 814.8) (1017.06, 971.04)
(477.18, 517.44) (803.88, 582.96)
(477.18, 734.16) (803.88, 814.8)
(364.32, 971.04) (929.28, 1016.4)
(364.82, 729.68) (928.64, 814.8)

UFC_check : no. = 9, bbox = (235.62, 235.2) (1017.06, 1016.4)

Fixing reported violations

You can the `fix_floorplan` command to fix the detected violations as part of the UFC methodology. It fixes violations by:

- Adding routing blockages around macros
- Snapping macro locations
- Re-shaping core shapes

The UFC methodology also provides the capability to automatically fix the violations with imported UFC rules.

Note: The `fix_floorplan` command tries to fix all the detected floorplan violations by default, however, it currently does not support fixing die size and core size violations. If a floorplan has rule conflicts, then the results after the fix may not be perfect. Cadence recommends you ensure that the rules used are not conflicting and that the die or core size satisfy the rules. Cadence also recommends you contact your Cadence representative if your constraints file comes from a third party, but you want to use the `fix_floorplan` command to fix the violations.

Example:

The following command fixes the ufc violations based on the rules defined in the *DMTB_ENC.ufc* file and writes the report to the *fix_cmds.tcl* file.

```
fix_floorplan -type ufc -file DMTB_ENC.ufc -report_file fix_cmds.tcl
```

Fixing violations by adding routing blockages

The detected UFC violations for spacing and halo rules can be fixed by adding routing blockage. The spacing rule uses the `set_spacing_rule -enclosure` and `-detect_ref_enclosed` options to support the addition of routing blockages while the halo rule uses the `set_halo_rule -detect_ref_enclosed` option.

Note: The enclosed/surrounded object is assumed to rectangular and the routing blockages are added by legal value. The object is not removed or reshaped.

- **For spacing rules**
  For spacing rule violations, with the `set_spacing_rule -enclosure` and `-detect_ref_enclosed` options defined, the `fix_floorplan` command creates the routing blockages by minimum legal value. The created blockages follow the contouring of macros.
  - If no value is specified, set enclosure as 0.
  - If only `{.spacing >= $min_value}` rule exists, align enclosure with `$min_value`.
  - If only `{on_grid(.spacing - $offset, $pitch)}` rule exists, align enclosure with `$offset`.
  - If both `{.spacing >= $min_value}` and `{on_grid(.spacing-$offset,$pitch)}` exist, then
  - If `$min_value <= $offset`, set enclosure as $offset.
  - If `$min_value > $offset`, search min integer `N` to satisfy `$pitch*N+$offset >=`

`$min_value`, and then set enclosure as `$pitch*N+$offset`.

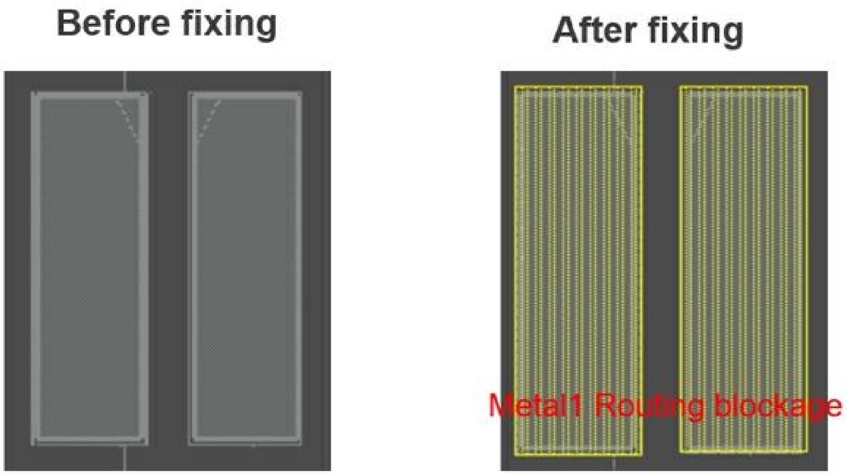
- If only white list or white zone exist, use the min value inside white list or white zone as inner distance.

Example:

When the rules are set as following:

```
set_spacing_rule -enclosure -name A -obj [route_blockages -if{.layer.name ==
Metal1}]\
   -ref [macros] -direction vertical -detect_ref_enclosed {.s >= 2.0}
```

```
set_spacing_rule -enclosure -name B -obj [route_blockages -if{.layer.name ==
Metal1}]\
   -ref [macros] -direction horizontal -detect_ref_enclosed {.s >= 2.0}
```



- **For halo rules**

  For halo rule violations, with the `set_halo_rule-detect_ref_enclosed` option defined, the `fix_floorplan` command creates the routing blockages. The created blockages follow the contouring of macros.

  - The inner distance of the created blockage should use maximum legal value.
  - The outer distance of the created blockage should use minimum legal value.

Fixing violations by snapping macro locations

The detected UFC violations for can be fixed by automatically snapping macro locations when the following rule exists:

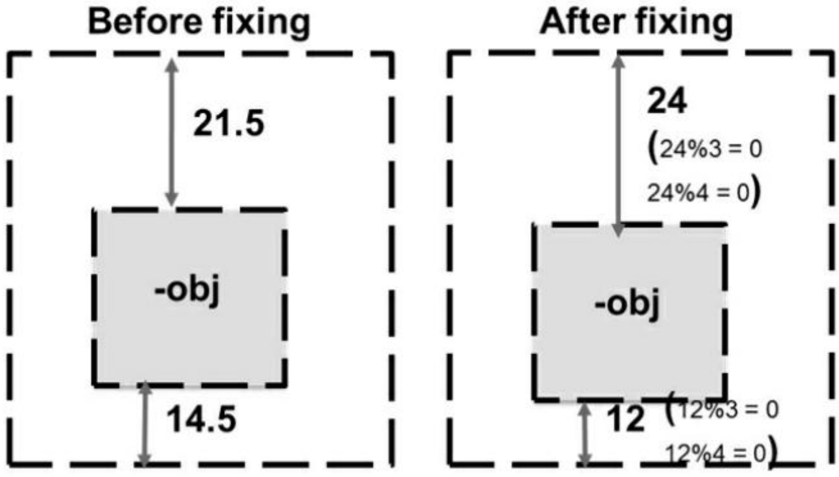
```
set_spacing_rule -obj [macros] {on_grid (.spacing-$offset,$pitch) }
```

Note: When multiple rules exist, the result of snapping must meet all rules. If the results so not meet all the rules at the same time, an error is reported and snapping of macro locations is aborted.

Example:

When the rules are set as following:

```
set_spacing_rule -enclosure -name A -obj [design_boundary] -ref [macros] \
-direction vertical -detect_ref_enclosed {on_grid (.s-0.0,3.0)}

set_spacing_rule -enclosure -name B -obj [design_boundary] -ref [macros] \
-direction vertical -detect_ref_enclosed {on_grid (.s-0.0,4.0)}
```



Fixing violations by re-shaping available sites

The detected UFC violations for can be fixed by adjusting the shape of available sites when the following rule existed and was violated:

- `set_width_rule` on available_sites
- `set_spacing_rule` for design boundary enclosure available_sites
- `set_spacing_rule` between available_sites and macros

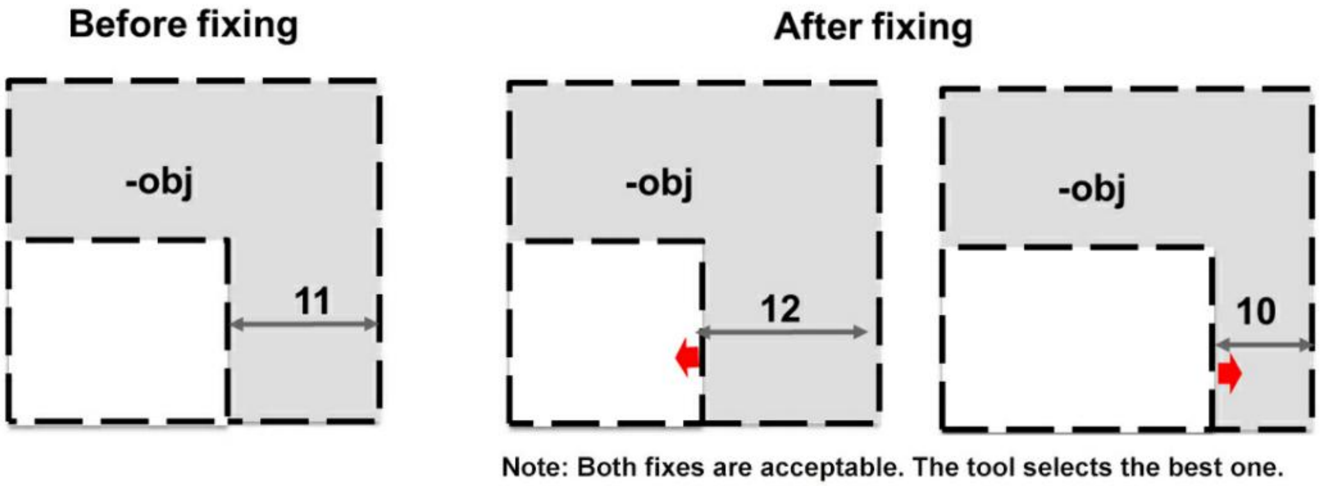
Note: If fixing is not available, then the location of violation is flagged.

Note: Even is all of dimensions white_list/white_zone/on_grid/min/max exist in the available_sites rule, only on_grid/min/max are taken into account.

Example:

When the rules are set as following:

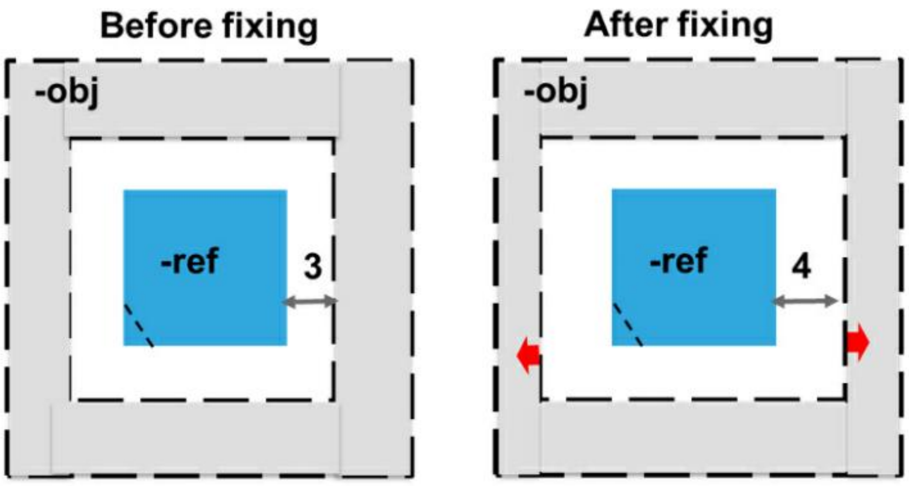
```
set_width_rule -name A -obj [available_sites] -width_type simple -direction horizontal
{on_grid (.w-6.0, 2.0.s >= 2.0)}
```


Before fixing
After fixing
-obj
11
-obj
12
-obj
10
Note: Both fixes are acceptable. The tool selects the best one.

Example:

When the rules are set as following:

```
set_width_rule -name A -obj [available_sites] -ref[macros] -direction horizontal
{on_grid (s >= 4.0)}
```


Before fixing
-obj
-ref
3
After fixing
-obj
-ref
4

Using Structured Data Paths

Introduction to Structured Data Paths

Innovus™ Implementation System provides the Structured Data Path (SDP) capability, which allows you to specify data path information to get better performance, power, and area. You can specify data path information by either importing an SDP relative placement file or sourcing an SDP TCL script. Correct SDP placement ensures uniform routing.

SDP capability should be used when:

- Design is data path intensive. That is, the design contains standard cell columns and rows that require alignment.
- Performance increases are required.
- Time to market does not allow for full custom flow.

SDP is a semi-custom methodology that requires manual intervention so you need to have detailed design knowledge in order to get better speed, power, and area. The Innovus tool makes it easy for you to try different SDP experiments and evaluate their impact on congestion, timing, and power. However, the tool still relies on the relative placement information you specify for placing and handling SDP elements.

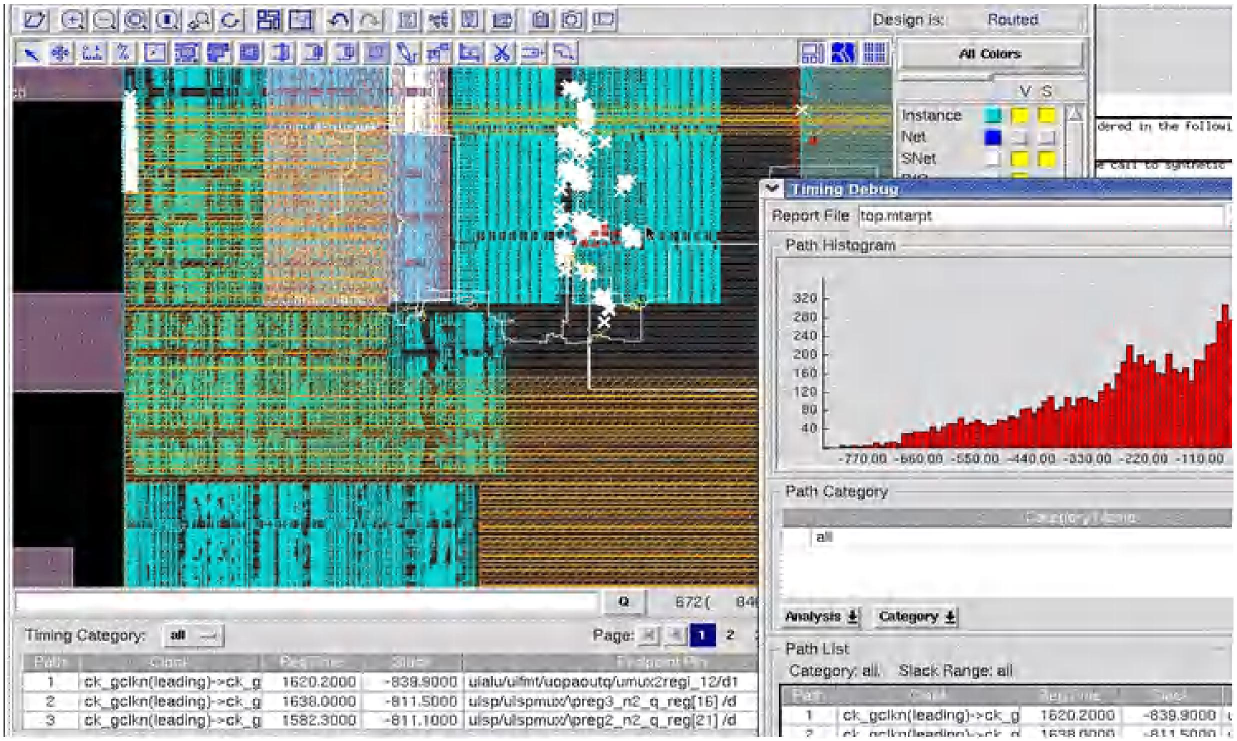
Furthermore, currently Innovus does not identify SDP elements automatically. You must script them based on naming conventions and detailed design knowledge.

Related Information

- Benefits of Using SDP
- General SDP 2G Flow
- SDP Placement Flow
- Implementing SDP Capability

Benefits of Using SDP

SDP provides a uniform environment for data path and control logic for placement, routing, and timing analysis.



Report File top.mtarpt

Path Category: all

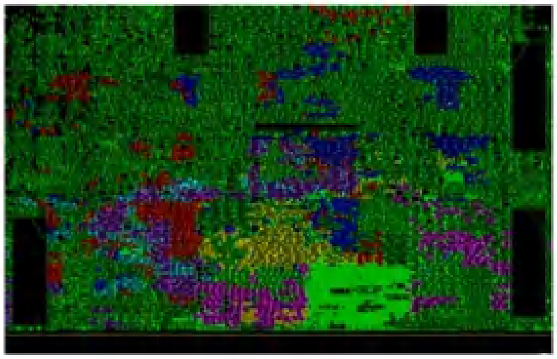
Path List
Category: all. Slack Range: all

Timing Category: all

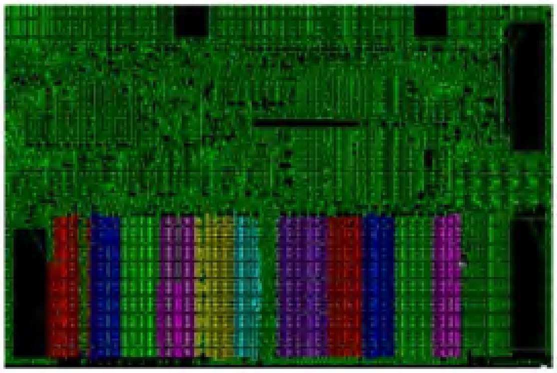
Path	Clock	Req Time	Slack	Endpoint Pin
1	ck_gclkn(leading)->ck_g	1620.2000	-839.9000	ui/alu/ui/fmt/uopaoutq/umux2regi_12/d1
2	ck_gclkn(leading)->ck_g	1638.0000	-811.5000	uisp/uispmux/\preg3_n2_q_reg[16] /d
3	ck_gclkn(leading)->ck_g	1582.3000	-811.1000	uisp/uispmux/\preg2_n2_q_reg[21] /d

Path	Clock	Req Time	Slack
1	ck_gclkn(leading)->ck_g	1620.2000	-839.9000
2	ck_gclkn(leading)->ck_g	1638.0000	-811.5000

SDP cells can be placed concurrently with other standard cells to get the optimal placement

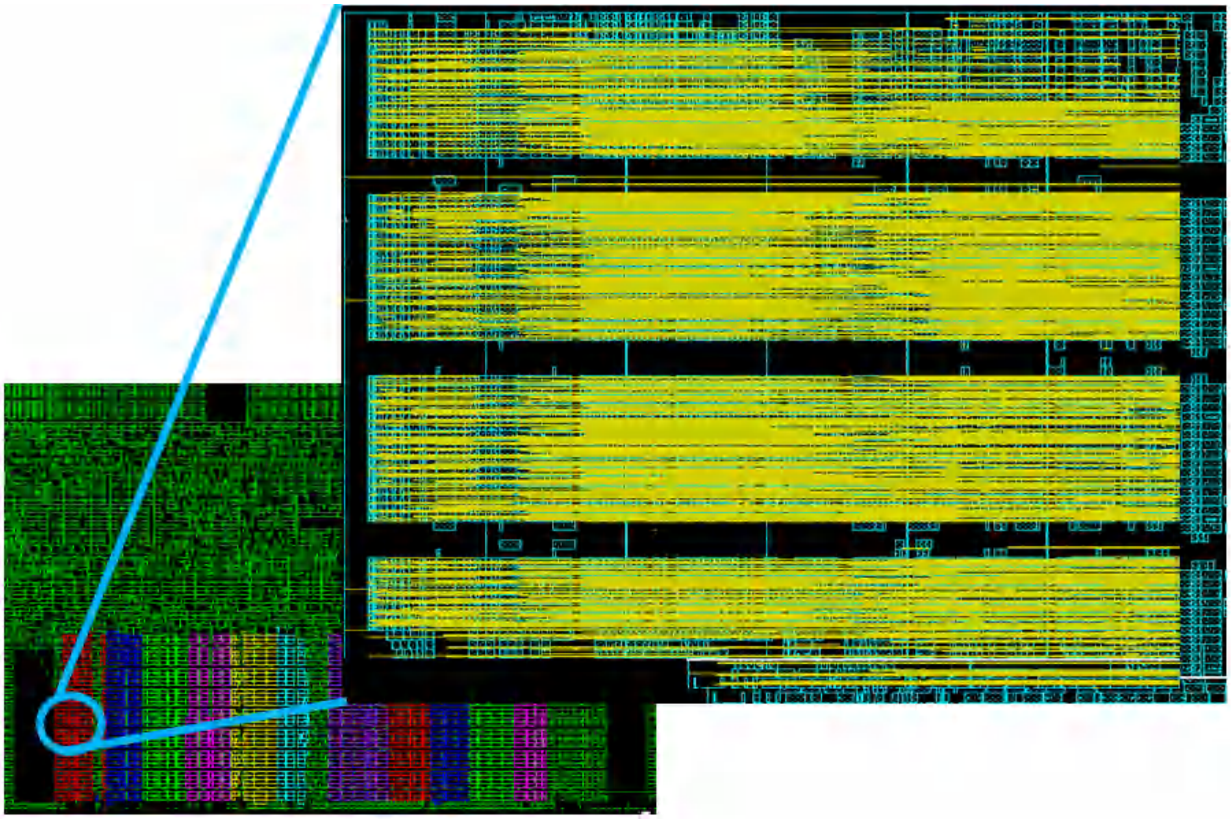


Traditional Placement



SDP Placement

The main advantage of this SDP placement is that it ensures uniform routing.



Related Information

- General SDP 2G Flow
- SDP Placement Flow
- Implementing SDP Capability

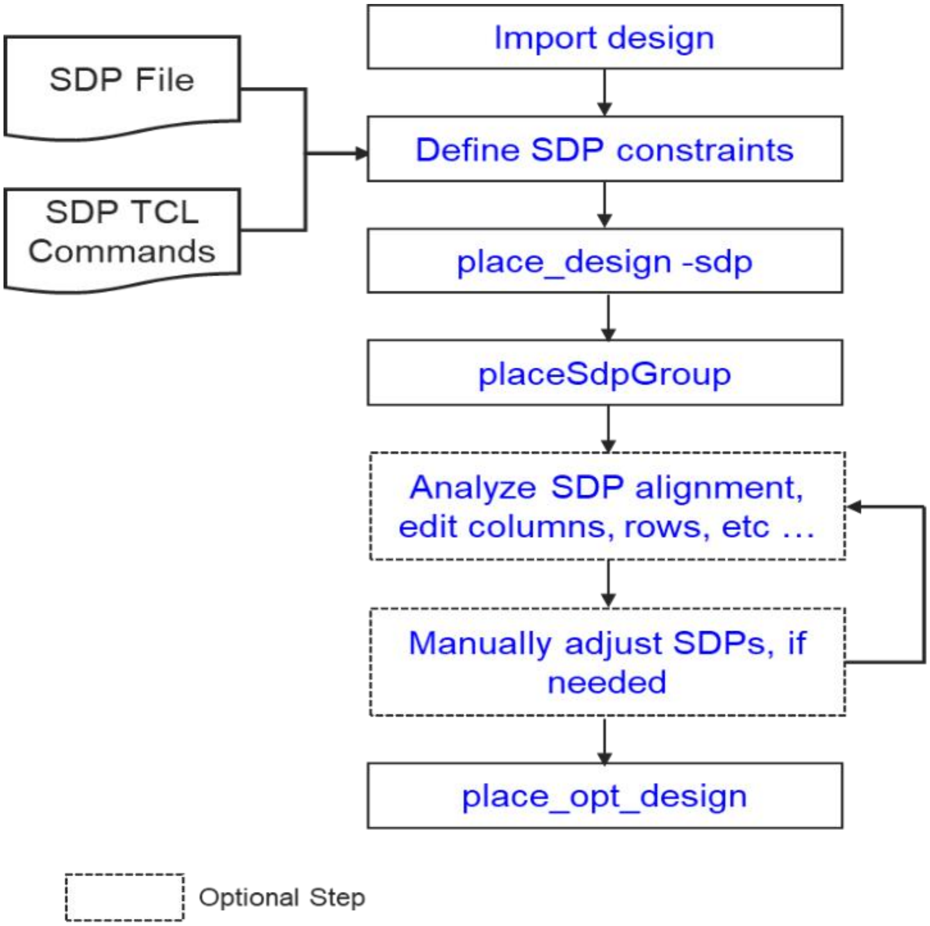
General SDP 2G Flow

The default SDP capability enables you to specify data path information to get better performance, power and area. The Innovus 21.1 release introduces the SDP 2nd Generation (2G) flow, which provides better handling of large scale of SDPs and more robust SDP legalization mechanism with less runtime, as compared to the default SDP flow. The 2G flow is especially useful for low advanced node designs. The 2G flow includes enhancements such as EEQ cell swapping, soft alignment, and capabilities for honoring the `dont_use` attribute and base cell padding.

From the 22.10 release, the `2G` flow is used by default during SDP placement.

Note: The old SDP flow has now been retired and is no longer supported.

The general SDP 2G flow is as follows:



As shown in the above 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`, `setSdpMode`, and so on to define relative SDP placement. All SDP TCL commands can be used during the "Analyze SDP alignment, edit columns, rows, etc." flow step.

After defining SDP constraints:

1. Run `place_design -sdp` to place SDP groups globally where the tool will find suitable positions for SDP groups.
2. Next, run `placeSdpGroup` to detail place SDP groups such that all instances in SDP groups are placed legally.

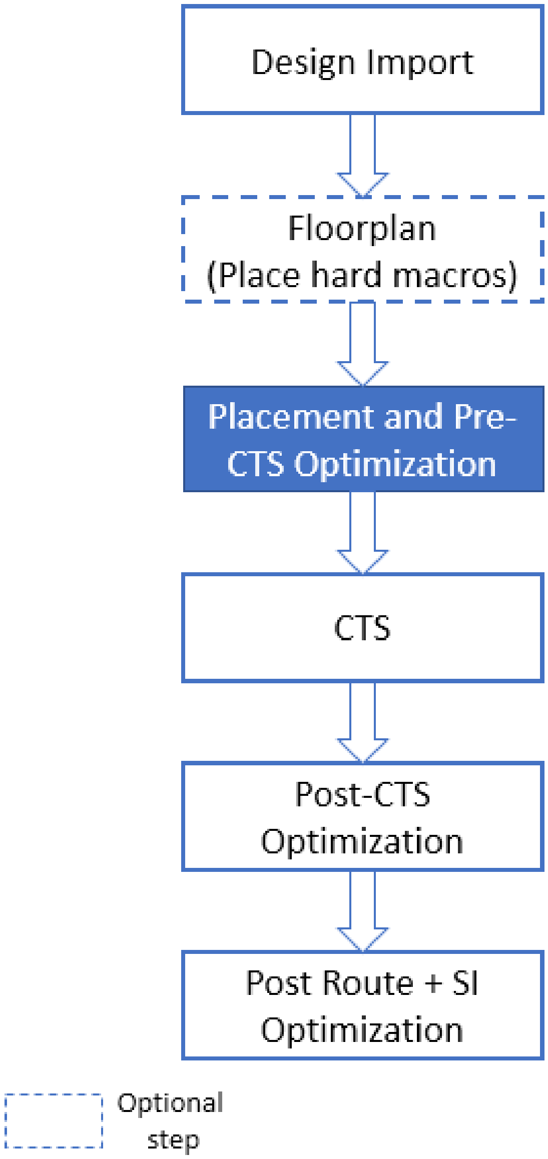
If some SDP groups could not be placed legally after `placeSdpGroup`, you should modify SDP groups manually based on SDP analysis. If you do not want to modify the failed SDP groups manually, the tool can also generate `instGroups` for such failed SDP groups.

Once SDP and standard cells have been placed, you can follow the normal flow.

Related Information

- SDP Placement Flow
- Implementing SDP Capability

SDP Placement Flow



SDP placement is part of the *Placement and Pre-CTS Optimization* step. After importing the design and hard macros placement, follow the general SDP 2G flow to place SDP groups. Note that in the 2G flow, SDP instances are set to fixed *before* `place_opt_design`. After the SDP and standard cells have been placed, you can follow the normal flow.

Related Information

- General SDP 2G Flow
- Implementing SDP Capability

Implementing SDP Capability

The SDP capability can be implemented using the SDP TCL commands, SDP browser, or SDP relative placement file.

- Using the SDP TCL Commands
- Using the SDP Browser
- Using the SDP Relative Placement File

Using the SDP TCL Commands

Innovus provides a number of TCL commands for defining and manipulating data path structures and groups.

For example, you can use the `createSdpGroup` command to create an SDP group based on existing cell placement as follows:

```
createSdpGroup -selected -name newSdp
```

You can then use the `addSdpGroupMember` command to add an object or member to the new SDP group. The added object can be an instance or another SDP group. Use the `-before` or `-after` parameter of the command to control whether the new group member is added before or after the specified reference object. Note that the `addSdpGroupMember` command enables you to create a structured data path using TCL commands. You do not need to learn SDP relative placement format for creating a data path.

You can use the `detachSdpGroup -detach` command to detach an SDP object or group from its parent if you want to regroup SDPs.

Note: When regrouping SDPs, Innovus only allows you to add a row to a column or vice versa. The tool will report an error if you try to add a row to a row SDP or a column to a column SDP.

For more information, see the Structured Data Path Commands chapter of the *Innovus Text*

Command Reference.

In addition to TCL commands, Innovus also enables you to access SDP attributes through database access commands. For example:

- To determine the SDP group to which an instance belongs, use:
  `dbGet top.insts.sdp`
- To determine the parent SDP group to which an SDP object belongs, use:
  `dbGet top.fplan.topSdps.sdps.parent`
- To determine all tree instances in an SDP object, use:
  `dbGet top.fplan.topSdps.sdps.treeInsts`

For example, consider the following SDP group definition:

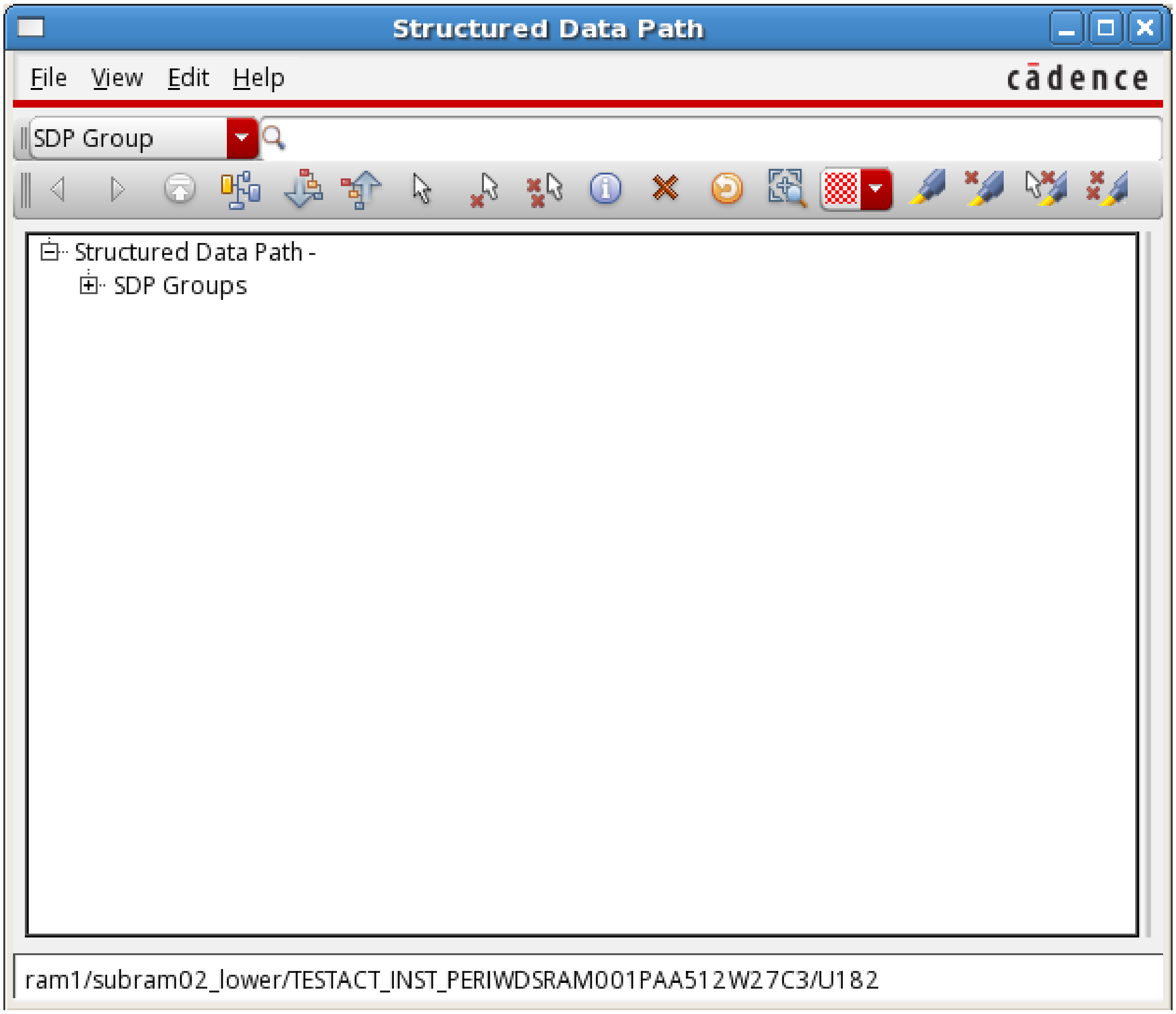
```
datapath DP0 {

   origin  4.446   3.3

    row  DP0_row0 {

        justifyBy SW

        column DP0_row0_col0 {

            justifyBy SW

            inst dp0_cpc

            inst dp0_lat1

        }

    }

}
```

Here:

- `set inst [dbGet -p1 top.insts.name dp0_cpc]`
  `[dbGet $inst.sdp.name]`
  Returns `DP0_row0_col0`
- `[dbGet $inst.sdp.parent.name]`
  Returns `DP0_row0`

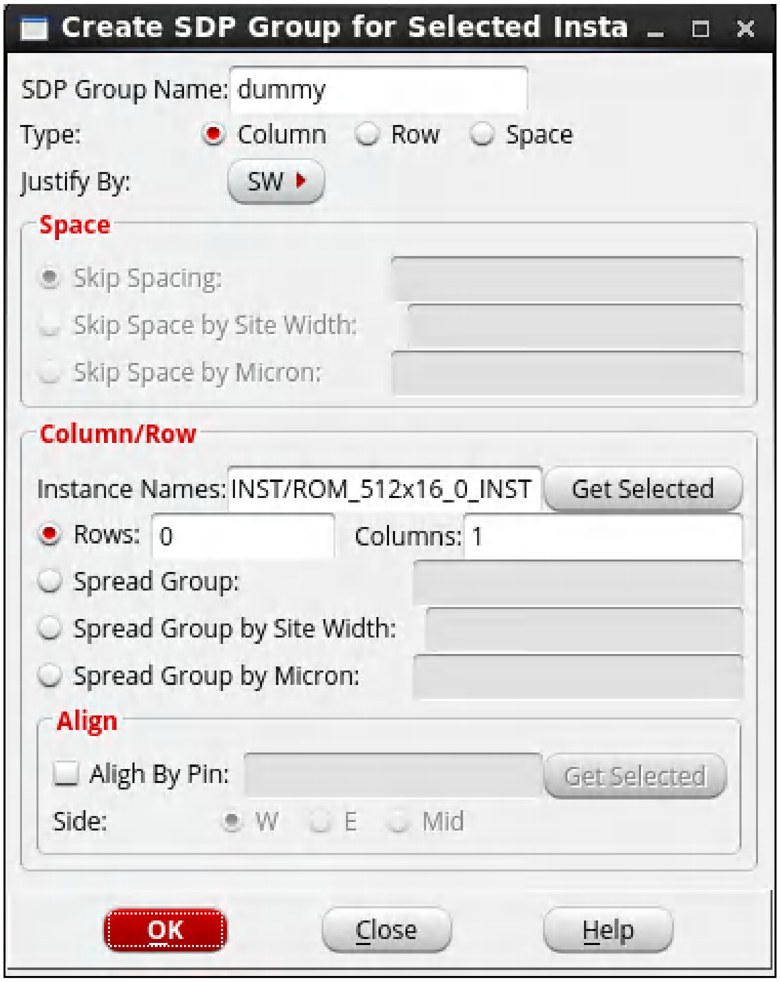
Using the SDP Browser

Innovus also provides the Structured Data Path browser (SDP browser) to help you manipulate SDP groups and/or elements and define data path and control logic using the graphical interface. To launch the browser, select *Floorplan - Structured Data Path* from the menu bar.



ram1/subram02_lower/TESTACT_INST_PERIWDSRAM001PAA512W27C3/U182

You can easily create a SDP group based on the existing physical locations of the selected cells in the GUI. To do so, select the required instances in the GUI, right-click to open the context menu, and then choose the *Create SDP Group* option from the context menu. This opens the Create SDP Group for Selected Instances form.



Create SDP Group for Selected Insta

SDP Group Name: dummy

Type: ● Column ○ Row ○ Space

Justify By: SW

Space

● Skip Spacing:

○ Skip Space by Site Width:

○ Skip Space by Micron:

Column/Row

Instance Names: INST/ROM_512x16_0_INST

● Rows: 0 Columns: 1

○ Spread Group:

○ Spread Group by Site Width:

○ Spread Group by Micron:

Align

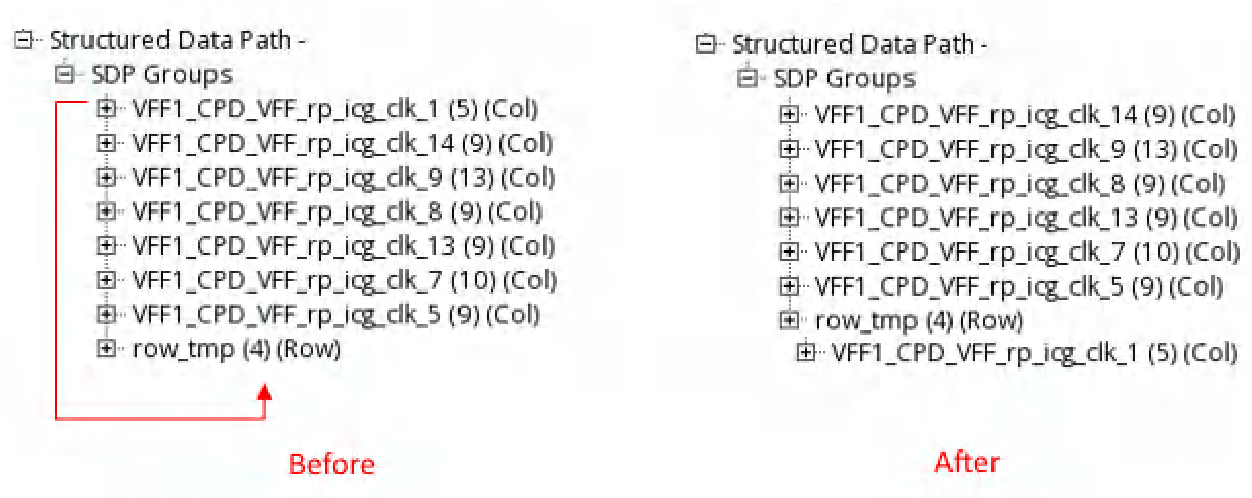
□ Aligh By Pin:

Side: ● W ○ E ○ Mid

Note: All selected cells must already have been placed.

The SDP browser also enables you to move around SDP rows and columns easily, simply by dragging and dropping items in the tree view.

For instance, to move a column under a row, click the column to select it. Then, keeping the right mouse button pressed, drag the column to the new position under the required row.



```
⊟ Structured Data Path -
   ⊟ SDP Groups
      ⊞ VFF1_CPD_VFF_rp_icg_clk_1 (5) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_14 (9) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_9 (13) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_8 (9) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_13 (9) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_7 (10) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_5 (9) (Col)
      ⊞ row_tmp (4) (Row)
```

Before

```
⊟ Structured Data Path -
   ⊟ SDP Groups
      ⊞ VFF1_CPD_VFF_rp_icg_clk_14 (9) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_9 (13) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_8 (9) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_13 (9) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_7 (10) (Col)
      ⊞ VFF1_CPD_VFF_rp_icg_clk_5 (9) (Col)
      ⊞ row_tmp (4) (Row)
        ⊞ VFF1_CPD_VFF_rp_icg_clk_1 (5) (Col)
```

After

Note: You cannot drag a column under another column or a row under another row.

For more information on using the SDP browser and related forms, see the "Structured Data Path" section in the Floorplan Menu chapter of the *Innovus Menu Reference*.

Using the SDP Relative Placement File

You can bring SDP information into the Innovus environment through an SDP relative placement file. The SDP file:

- Specifies the 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

You can define an SDP group file in two ways. The first method is to select instances in the GUI and create an SDP group for the selected instances. The second way is to write the SDP file directly using TCL.

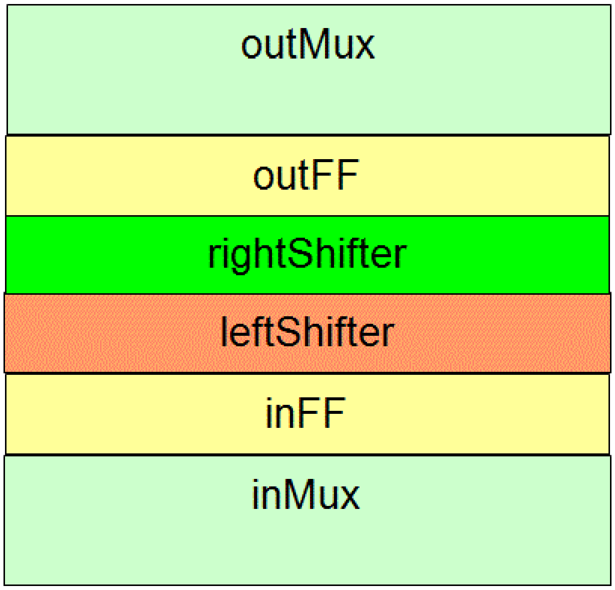
- Here's an example for defining an SDP group in a file named `sdp.tcl`:

```
set insts {DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/acc_reg_31
           inst DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/acc_reg_30
           inst DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/acc_reg_29
           inst DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/acc_reg_28}
createSdpGroup -name col2 -inst $insts -justifyBy SW -type col
setSdpGroupAttribute -name col2 -origin {400 400}
```

  You can then source the `sdp.tcl` file to read in the data paths defined in the file.

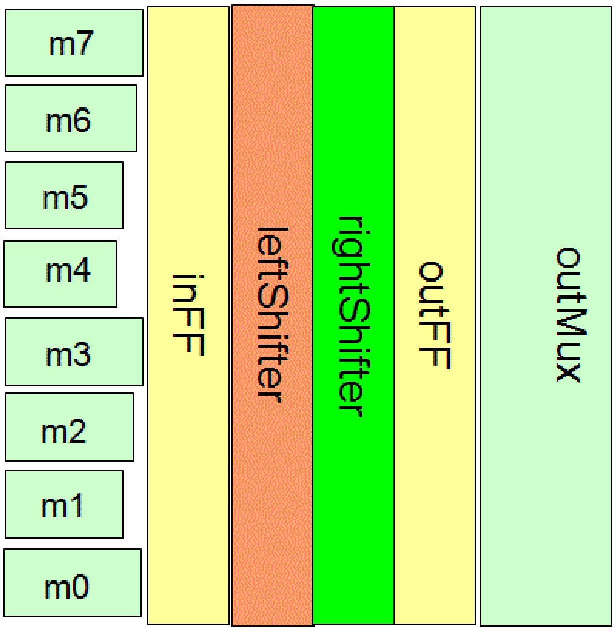
- Following is an example of an SDP file for creating a column of rows:

```
datapath sdp {

column adder {

    justifyBy SW

    row inMux { … }

        row inFF { … }

        row leftShifter { … }

        row rightShifter { … }

        row outFF { … }

        row outMux { … }

}

}
```



- SDP also supports hierarchical construction (SDP within SDP, rows within a column and vice versa), as shown below.

```
datapath sdp_row {

row adder {

    justifyBy SW

    column inMux {

             inst m0

            inst m1

            inst m2

            inst m3

            inst m4

            inst m5

            inst m6

            inst m7 }

        column inFF { … }

        column leftShifter { … }

        column rightShifter { … }

        column outFF { … }

        column outMux { … }

}

}
```



SDP File Format

The SDP relative placement file has the following format:

```
alias var1 var2

datapath name {

    hierPath name

    origin x y

    row name {
            [ orient R0 | MX | MY | R180 | MX90 | R90 | R270 | MY90]
            [ justifyBy NW | SW | SE | NE | W | E | N | S ]
            [ flip X | Y | XY ]
            [ skipSpace value [siteWidth siteName | micron ]
               | inst instanceName [orient R0|R90|..] [justifyBy ...] [flip X|Y|XY]
               | column name { ... } ]...
            [ spreadGroup value instanceName [sideWidth siteName | micron ] ]
            }
    }
datapath name {
    hierPath name
    origin x y
    column name {
           [ orient R0 | MX | MY | R180 | MX90 | R90 | R270 | MY90]
           [ justifyBy NW | SW | SE | NE | W | E | N | S ]
           [ flip X | Y | XY ]
           [ skipSpace value [siteWidth siteName | micron ]
               | inst instanceName [orient R0|R90|..] [justifyBy ...] [flip X|Y|XY]
               | row name { ... } ]...
           [ spreadGroup value instanceName [sideWidth siteName | micron ] ]

           }
    }
 # is used for comment
```

# Keywords like `row`, `column` can be redefined using the `alias` command.

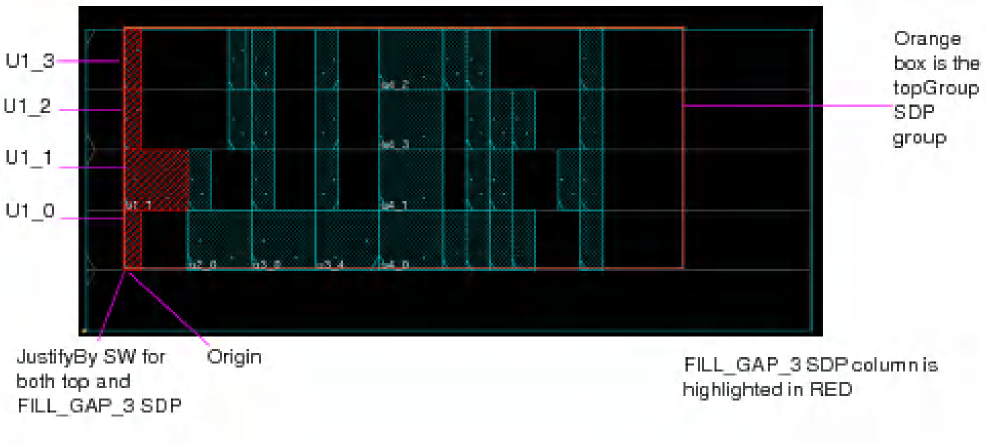
The format uses the following keywords:

- `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.

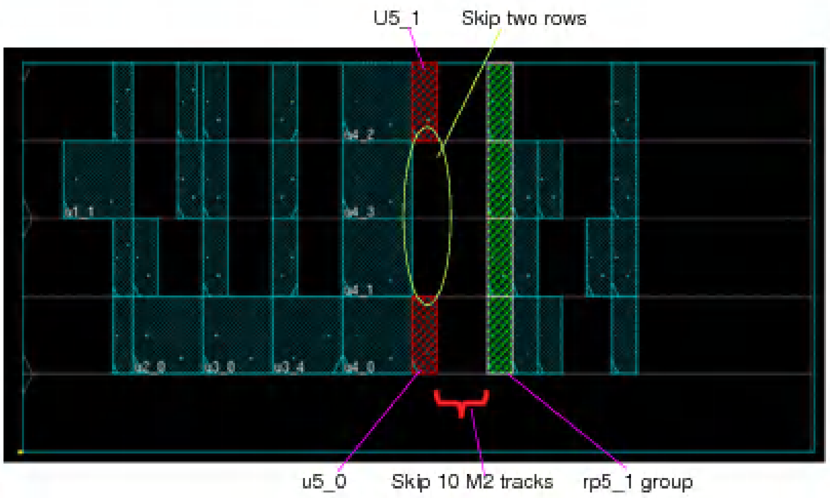

U1_3
U1_2
U1_1
U1_0
u4_2
u4_3
u4_1
u2_0
u4_0
Orange box is the topGroup SDP group
JustifyBy SW for both top and FILL_GAP_3 SDP
Origin
FILL_GAP_3 SDP column is highlighted in RED

- `flip`: Flips the SDP group or an SDP instance element in vertical, horizontal, or both directions. Possible values can be `X`, `Y`, or `XY`.
- `column`: Specifies the name of an SDP column group. Name should be unique within same data path group or across different data path group. If the name is not unique, the tool automatically adds an index to the specified name to make it unique.
- `skipSpace`: Specifies a space value to be skipped. By default if the `skipSpace` value is defined in a column, then this value is for row skipping and represents the number of skipped rows. If the `skipSpace` value is defined in a row, then this value is for column skipping and represents the number of M2 tracks (pitch of first vertical layer). `skipSpace` value can also be specified in micron units or as number of widths of a specified row site. However, row site width should be specified only for an SDP row, and not for an SDP column.
  Following is the example of `skipSpace`:

```
...
 column FILL_GAP_9 {
   justifyBy SW
   row main_5 {
              justifyBy SW
              column rp5_0 {
                       justifyBy SW
                       inst u5_0
                        skipSpace 2 # Skip 2 rows
                        inst u5_1
                        }
```

```
          skipspace 10 # skip 10 M2 tracks
          column rp5_1 {
              justifyBy SW
              inst u5_<2:5>
              }
            }
    }
```

...



- `siteWidth`: Specifies the name of the tech site. Use the number of widths of the specified site as skip unit in horizontal direction.
- `micron`: Specifies that the specified `skipSpace` *value* is to be interpreted as distance in microns (um).
- `inst`: Specifies one or more instance names. Use this keyword if you want to create the SDP group from specified instances.
- `spreadGroup`: Inserts space between group of instances. If `spreadGroup` is specified in a column, spacing will be the number of added rows. If `spreadGroup` is specified in a row, spacing will be the number of placement grids added between the columns in that row. `spreadGroup` value can also be specified in micron or number of widths of a specified site.

In the following example, `spreadGroup` is defined inside a column SDP so that a row will be skipped between specified instances:

```
column sdp_grp1 {
   justifyBy SW

   spreadGroup 1 reqPath/reg_busBit<0:17>

  }
```

This is equivalent to:

```
column sdp_grp1 {

    justifyBy SW

    reqPath/reg_busBit0

    skipSpace 1

    skipSpace 1

    ...

   }

  ...
```

Reusing SDP Instantiations

The SDP file format supports 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

      }
```

Here:

- *normal_dataPath_syntax*: Specifies a row or column data path:

```
row  name  {

      [ orient R0|R90 |... ]

      [ justifyBy NW|SW|SE|NE|W|E|N|S ]

      [ 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 ]
        [ 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] ]
        }
```

After a macro definition is specified, you can instantiate or use the macro definition in a data path specification by using the `use` keyword:

```
use SDP_macro_name row_or_column_name [hierPathName]
```

Here:

- *`SDP_macro_name`*: Specifies the name of a data path macro definition.
- *`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.
- *`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.

From the 18.1 release, the `flip` attribute is supported in SDP instantiations. Using this attribute, you can flip the instances in a child SDP as compared to the parent SDP group. See the second **`use`** statement in `entry2` in the example below.

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 flip X
            }
        }
    }

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
                    }
                }
            }
        }
    }
```

Related Information

- Setting SDP Options
- SDP Online Editing
- Checking SDP Placement

Setting SDP Options

The `setSdpMode` command allows you to set SDP related sticky options before using `place_design -sdp` and `placeSdpGroup`. You can use the `setSdpMode` command to:

- Disable the extension of the core boundary
  If you set `setSdpMode -disable_extended_core` 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 fixed objects

  You can use `setSdpMode -pre_fixed_cells_blockage_direction` to specify spacing constraints between fixed objects/placement blockages and SDP groups to keep SDPs at a suitable distance from them.
- Honor the `dont_use` attribute
  Electrically equivalent (EEQ) cells that have the same functionality and size can be used to replace failed SDP cells. By default, during EEQ cell swapping, the tool can use EEQ cells that have the `dont_use` attribute defined. The `setSdpMode -honor_dont_use` option controls whether or not to honor this attribute. You can set it to `true` if you do not want EEQ cells with the `dont_use` attribute defined to be used for EEQ swapping.
- Control the mechanism for legalizing SDP groups
  In the 2G flow, the tool allows you to control the mechanism for legalizing SDP groups by setting the legalization effort using the `setSdpMode -legalization_effort` parameter. The higher the setting for this parameter, the more the iterations done to place and align the SDP groups leading to more runtime.
- Generate a detailed SDP placement report
  If you specify `setSdpMode -place_report` *`file_name`*, the software generates a detailed SDP

placement report on running `placeSdpGroup`.

The detailed SDP placement report contains the following information:

- Standard output file header
- Summary report at the end of the report file such as:
- Total number of SDPs in the design
- Total number of placed SDPs
- Total number of unplaced SDPs
- Number of overlapped SDPs.

Following is an example of an SDP placement report file:

```
############################################################
# Generated By: Cadence Innovus 18.10-b011_1
# Generated on: Mon May 7 10:00:29 2018
# HostName: noi-leenap1
# Design: DTMF_CHIP
# Command: placeSdpGroup
############################################################

SDP Name Location
========================================================
group1, (349.800, 649.040)
group2, (700.260, 649.040)


Total Number of SDPs: 2
Total Number of placed SDPs: 2
Total Number of Unplaced SDPs: 0
```

Note: You can use the `getSdpMode` command to retrieve the current values of `setSdpMode` command.

Related Information

- SDP Online Editing
- Checking SDP Placement

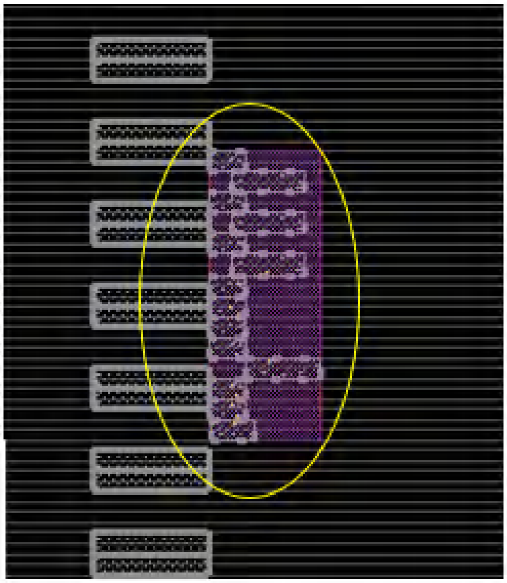
SDP Online Editing

You can move an SDP to a preferred location from within the GUI. The SDP legalizer legalizes the SDP placement, taking into account the following order for moving objects away from the preferred location:

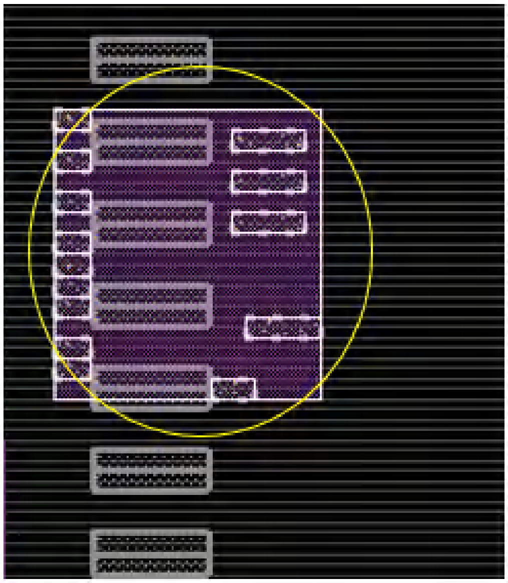
- Fixed instances < SDP < Soft fixed instances < Placed instances

In other words:

- Fixed instances stay in the preferred location. If an SDP is moved to overlap with fixed instances at the preferred location, the SDP is either moved next to it or interleaved with these instances.

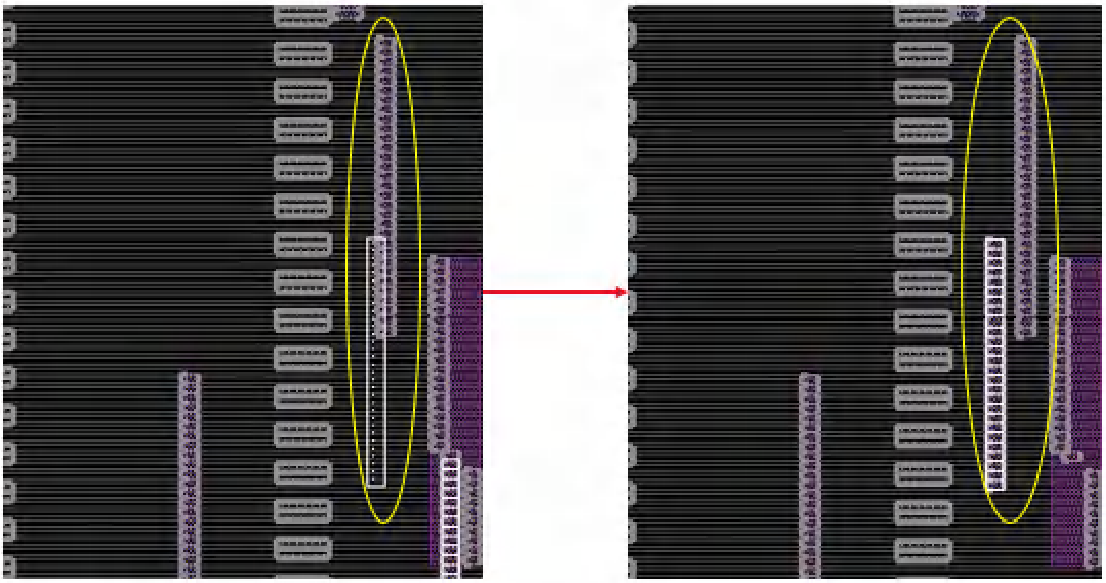


SDP moved next to fixed instances



SDP interleaved with fixed instances

- Soft fixed instances are moved away.
- If there are other SDP groups in the preferred location, the existing SDP groups at the location may be moved away.



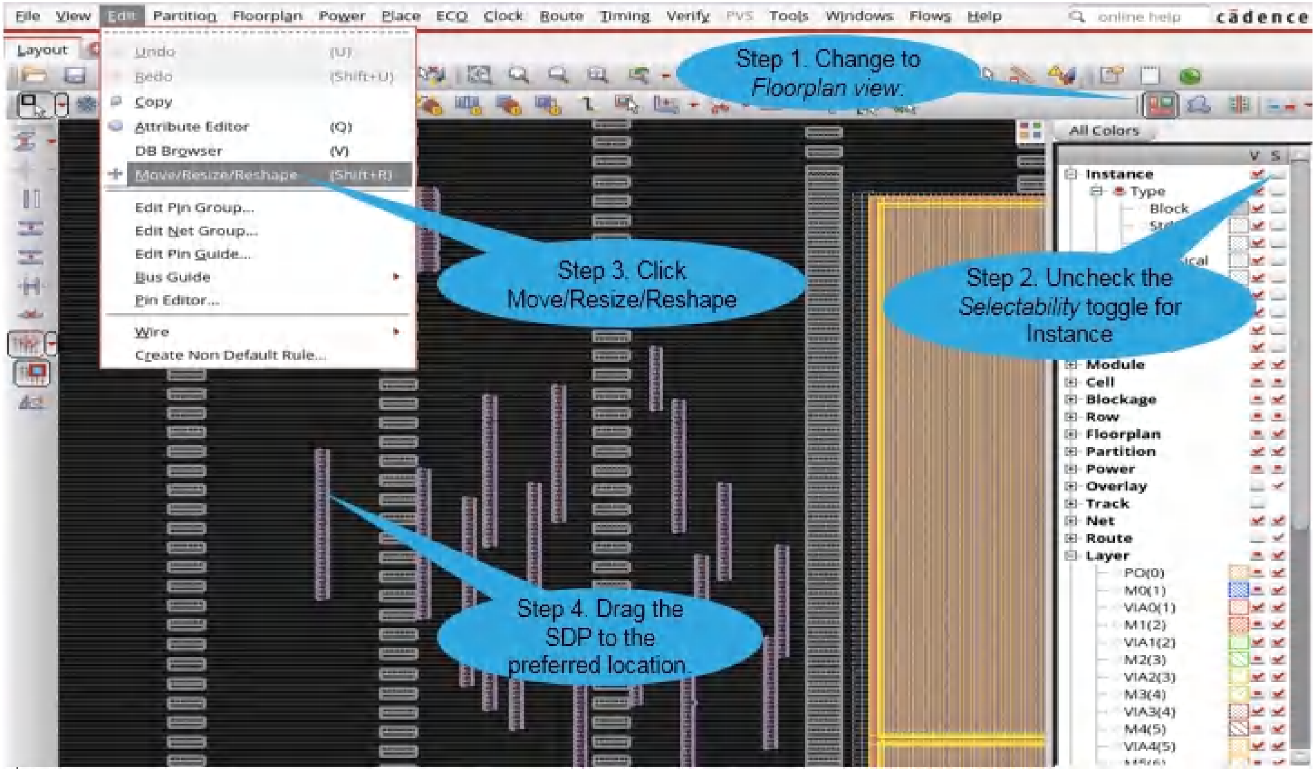
SDP moved to overlap

Existing SDP moved

You need to set the following modes in advance for SDP online editing:

- `setSdpMode -legalization {AUTO | SW}`
- `setSdpMode -pre_fixed_cells_blockage_direction Y`

To move an SDP to a preferred location, follow the steps given below:



Step 1. Change to *Floorplan view.*

Step 2. Uncheck the *Selectability* toggle for Instance

Step 3. Click Move/Resize/Reshape

Step 4. Drag the SDP to the preferred location.

Note: If there are no legal placement locations for all SDP instances in the new location, some SDP instances may not be aligned due to placement constraints.

Related Information

- Setting SDP Options
- Converting Failed SDPs
- Checking SDP Placement

Converting Failed SDPs

In the 2G flow, the tool supports the conversion of failed top SDP groups to instGroups by using the `placeSdpGroup -create_inst_group` parameter. This option works only on top SDPs. This means the tool will convert the top SDPs to instGroups, irrespective of whether it was the top SDPs or child SDPs that failed.

The instances in the instance groups are soft_fixed.

Related Information

- SDP Online Editing
- Checking SDP Placement

Checking SDP Placement

After you perform any manual editing and/or optimization step, you may want to check for any resulting SDP violations before you move on to the next step in your flow. Innovus provides the `checkSdpGroup` 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 `checkSdpGroup`, you can check whether you need to resolve SDP overlapping or re-run SDP placement before moving to the next step in the flow. `checkSdpGroup` checks the following:

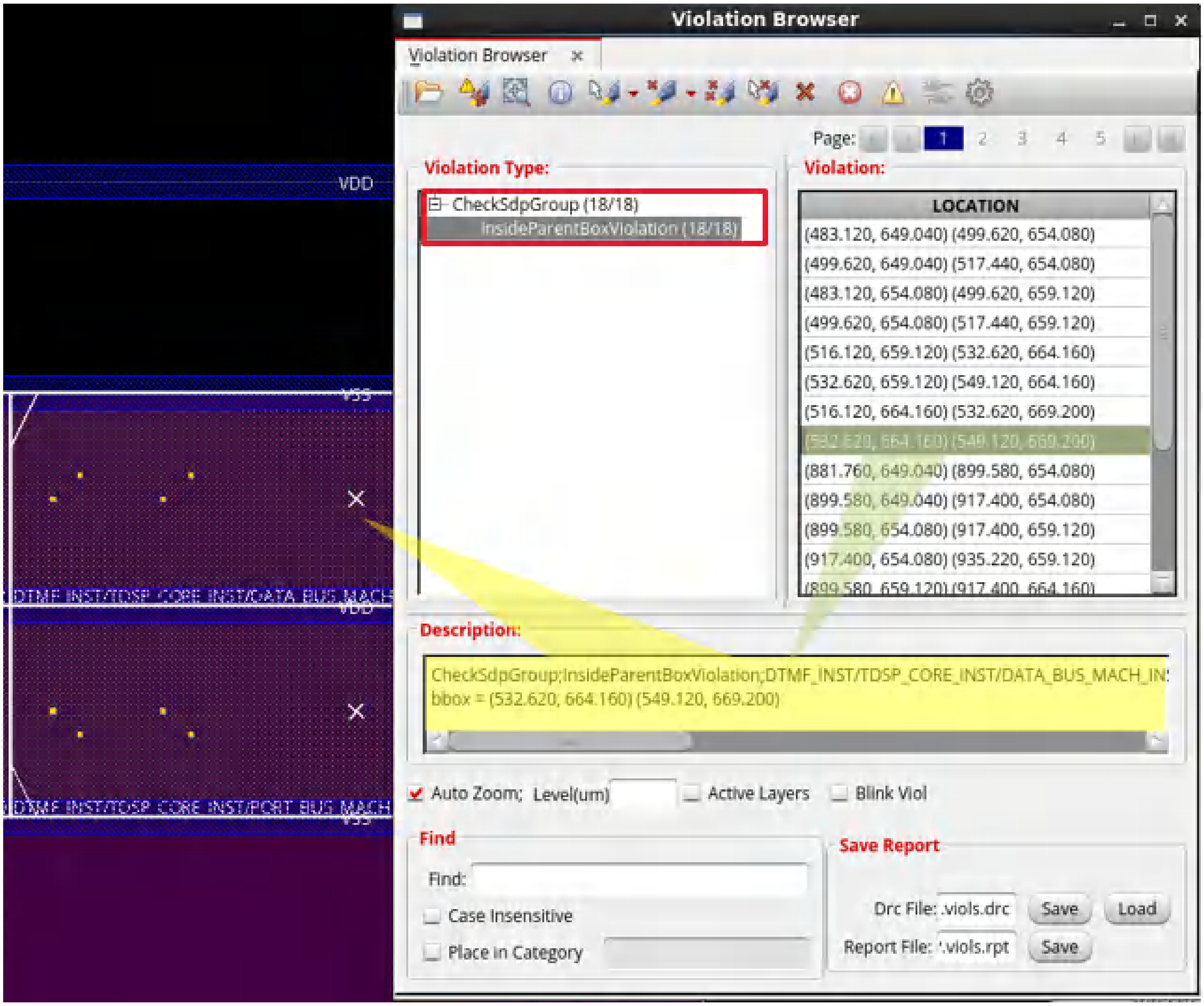
- SDP group/instance orientation
- SDP group/instance `justifyBy` constraint
- Alignment by pin name constraint
- SDP group/instance flip constraint
- Skip space constraint
- SDP group/instance overlapping

Using the `-file` option of `checkSdpGroup`, 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`.

You can view the `checkSdpGroup` violations in the Violations Browser (*Tools -> Violation Browser*).


VDD
VSS
DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH
VDD
DTMF_INST/TDSP_CORE_INST/PORT_BUS_MACH
VSS
Violation Browser
Violation Browser ×
Page: 1 2 3 4 5

Violation Type:

- ⊟ CheckSdpGroup (18/18)
  - InsideParentBoxViolation (18/18)

Violation:

LOCATION
(483.120, 649.040) (499.620, 654.080)
(499.620, 649.040) (517.440, 654.080)
(483.120, 654.080) (499.620, 659.120)
(499.620, 654.080) (517.440, 659.120)
(516.120, 659.120) (532.620, 664.160)
(532.620, 659.120) (549.120, 664.160)
(516.120, 664.160) (532.620, 669.200)
(532.620, 664.160) (549.120, 669.200)
(881.760, 649.040) (899.580, 654.080)
(899.580, 649.040) (917.400, 654.080)
(899.580, 654.080) (917.400, 659.120)
(917.400, 654.080) (935.220, 659.120)
(899.580, 659.120) (917.400, 664.160)

Description:

CheckSdpGroup;InsideParentBoxViolation;DTMF_INST/TDSP_CORE_INST/DATA_BUS_MACH_IN:
bbox = (532.620, 664.160) (549.120, 669.200)

☑ Auto Zoom; Level(um) ☐ Active Layers ☐ Blink Viol

Find

Find:
☐ Case Insensitive
☐ Place in Category

Save Report

Drc File: .viols.drc Save Load
Report File: '.viols.rpt Save

Related Information

- Setting SDP Options
- SDP Online Editing
- Converting Failed SDPs

Bus Planning

Overview

The Bus Planning feature in the Innovus software enables you to plan and create bus guides which are used to guide the path of busses for floorplanning, partition pin optimization, feedthrough insertion, congestion prediction in early global routing, and final routing using the NanoRoute router.

Most designs need bus planning for estimating the design size and routing channel widths. Without bus guides, the routers do not route all the bus bits together on the desired path. Routing the bus bits outside the desired path can have high cost implications. Hence it is very important to accurately plan the bus guide layouts.

Bus planning is critical in the prototyping stage of the hierarchical flow. Use the bus planning capability to guide the path of bus routing for feedthrough insertion, partition pin optimization, and congestion prediction. If you are in the implementation stage, use bus planning to guide the path of busses for detailed routing.

Bus Planning Flow in Innovus

For hierarchical designs, you create bus guides before or after assigning the partition/black box pins. For flat or top-level designs, you create bus guides before routing. Normally, you create bus guides before pin assignment.

The following steps describe the bus planning flow in Innovus:

1. Importing the design

   Import the design into the Innovus environment.

2. Floorplanning the design

   If the design is a partition design, then specify partitions. For more information, see "*Specifying Partitions and Blackboxes*" section in the Partitioning the Design chapter of the *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 `proto_design` to place them automatically. Further, adjust the floorplan if needed.

3. Defining net groups

   Group the bus bit nets together as net groups using `createNetGroup` and/or `addNetToNetGroup` commands.

4. Creating bus guides

   Create bus guides associated with the net groups, to guide routing for all the nets of the specified net group. Bus guides can be created using the Edit Bus Guide form, which can be accessed from the Edit Menu and/or the `createBusGuide` command. See Creating a Bus Guide.

5. Placing the design

   Place the standard cells. If you do not want the Innovus placer `(place_design)` 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 `earlyGlobalRoute` to route the design.

7. (Optional) Inserting feedthrough buffers

   Feedthrough can be inserted based on routing or placement. If `earlyGlobalRoute` 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 *User Guide.*

8. Assigning pins

   Assign pins using the `assignPtnPin` command.

9. Committing partition

   Commit partitions using the `partition` command.

10. Saving Partition

    Save the partition information using the `savePartition` command.

11. Running NanoRoute at the top-level design
    Perform detailed routing using the NanoRoute router at the top-level design.

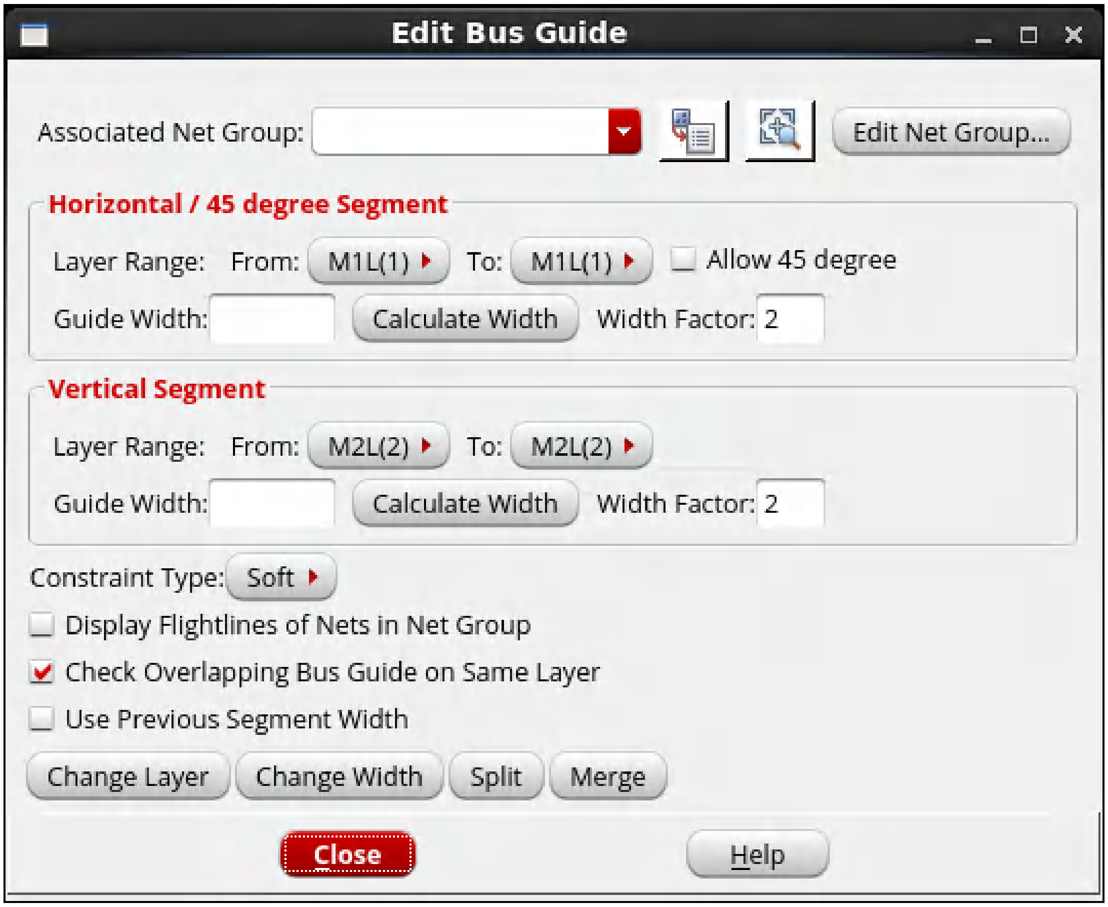
Creating a Bus Guide

A bus guide consists of one or more overlapping segments. It must always be associated with a net group. So, before creating a bus guide you must define a net group. Remember that a net group can either be assigned to a bus guide or a pin guide, but not to both. For each bus guide segment that you create, you must specify a layer or a layer range.

You can create a bus guide Using the Edit Bus Guide GUI and/or Using Text Commands.

Using the Edit Bus Guide GUI

The bus guide editor in Innovus allows you to create bus guides before or after assigning the bus pins. Using the *Edit Bus Guide* form, you can edit the bus guide properties and interactively create the bus guide for a specific net group.



Edit Bus Guide

Associated Net Group:

Horizontal / 45 degree Segment

Layer Range: From: M1L(1) To: M1L(1) □ Allow 45 degree

Guide Width: Width Factor: 2

Vertical Segment

Layer Range: From: M2L(2) To: M2L(2)

Guide Width: Width Factor: 2

Constraint Type: Soft

□ Display Flightlines of Nets in Net Group

☑ Check Overlapping Bus Guide on Same Layer

□ Use Previous Segment Width

You can specify the net group associated with the bus guide you are creating in *Associated Net Group* drop-down list. This is an editable drop-down list, which lists all net groups by default. You can either select the required net group from the list or type the name of the net group in the box. As you start typing the name of the net group, the drop-down list changes to show only the net group names that match the pattern.

In addition to the net group, you can specify the layer or layer range on which the bus guide is to be created and the width of the bus guide segment.

By default, the bus guide editor derives the default minimum guide width required to hold all the nets assigned to the bus guide. If the bus guide connects to placed pins on block edges, the bus guide editor automatically adjusts the width of the guide segment to cover all the pins of nets in the net group. The bus guide editor provides options to enable overlapping check for bus guides created on a specific layer and display flight lines of nets in the net group, when creating the bus guides.

By default, the bus guide is created as a soft constraint. If you specify *Constraint Type* as *Hard*, the tool honors the bus routing guide when you run Early Global Route (eGR), NanoRoute (NR), or NanoRoute High Frequency Router (NRHF). If you specify the *Constraint Type* as *Soft*, the bus guide only guides the route path and the tool can route the net out of the bus guide during eGR, NR, or NRHF routing.

The Edit Bus Guide form can also be used for net group to bus guide cross-probing:

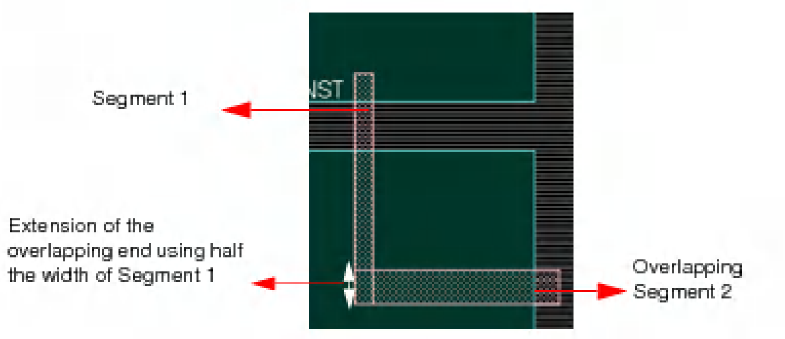
- Finding the net associated with the bus guide selected in the floorplan: Use the *Get Selected* ( ) button to find the name of the net group associated with the bus guide segment selected in the floorplan. When you click the *Get Selected* button, the *Associated Net Group* drop-down list jumps to the net group of the selected bus guide segment.
- Finding the bus guide of a specific net group: Use the new *Zoom Selected* ( ) button to zoom to the bus guide of the net group selected in the *Associated Net Group* drop-down list. When you click the *Zoom Selected* button, the view window will zoom to fit the bus guide of the net group.

For more information on the *Edit Bus Guide* form, see the Edit Menu chapter of the *Menu Reference*.

Drawing a Bus Guide

To draw a bus guide in Innovus, you must first click the *Edit Bus Guide* icon in the toolbar.

Once you are in the bus planning mode, you can draw the bus guide segment by clicking the left mouse button and dragging it along the points of center line for the guide segment. To end a bus guide segment, double-click the left mouse button. By default, the bus guide extends half width for the overlapping end of the created segment. However, if the guide segment overlaps with another segment that has bigger or smaller width, the bus guide editor uses half the width of the other segment for the extension of the overlapping end.


Segment 1
INST
Extension of the overlapping end using half the width of Segment 1
Overlapping Segment 2

Note: All the segments of the bus guide should overlap to ensure continuity; Otherwise, the router (nanoroute) may create routing problems or may take longer time to run.

You can specify a new segment connected to an existing segment as shown in the following image where segment 4 overlaps with segment 1:

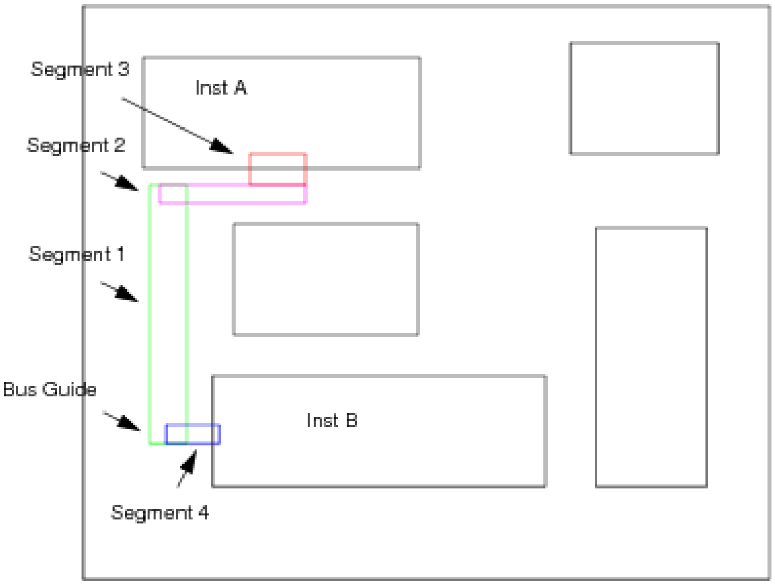


Figure 1

You can also draw a bus guide segment that connects to the placed pins of the associated net group.

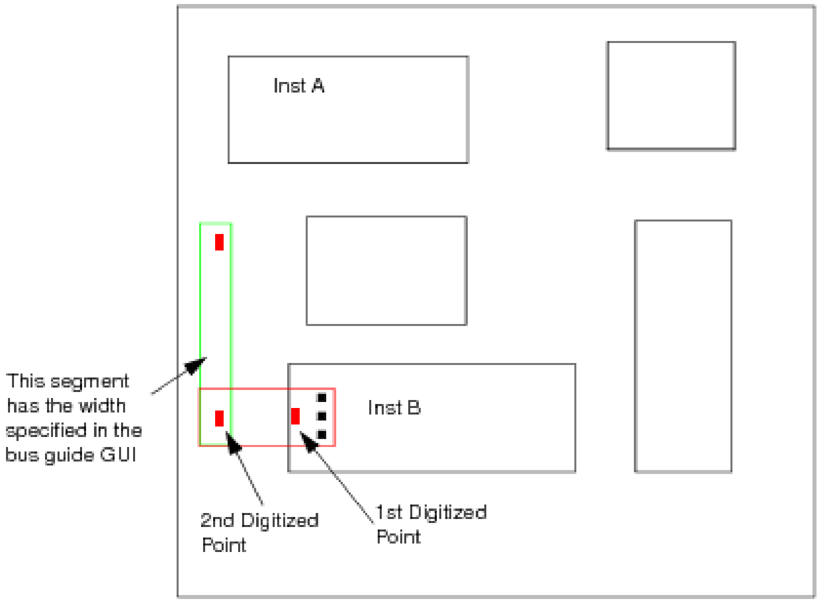


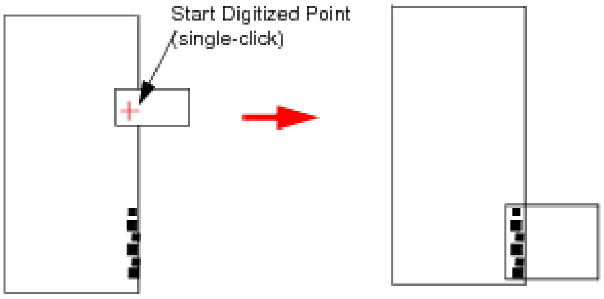
Figure 2

If you click on the partition boundary side where the pins are placed, the bus guide editor automatically snaps to these pins. If the width value specified in the bus guide editor is smaller than the width required to fully cover all these pins, the bus guide editor derives new width for the guide segment such that all the associated physical pin geometries are covered. If the width value is bigger than the width that needs to cover all pins, the editor will use the current width value without adjusting it.

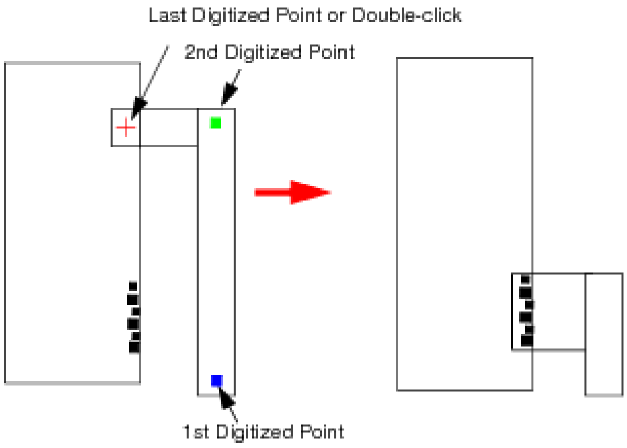
In Figure 2, the width of the segment defined by the first and the second digitized points is derived based on the placed pin information such that the segment width can fully cover the all the pins. The width of the next segment (defined by second and third points) is the width that is specified in the bus guide editor.

The snapping of bus guides to pins (partition or black box pins) occur at the start or at the end of the bus guide, when you double-click to end the bus guide.

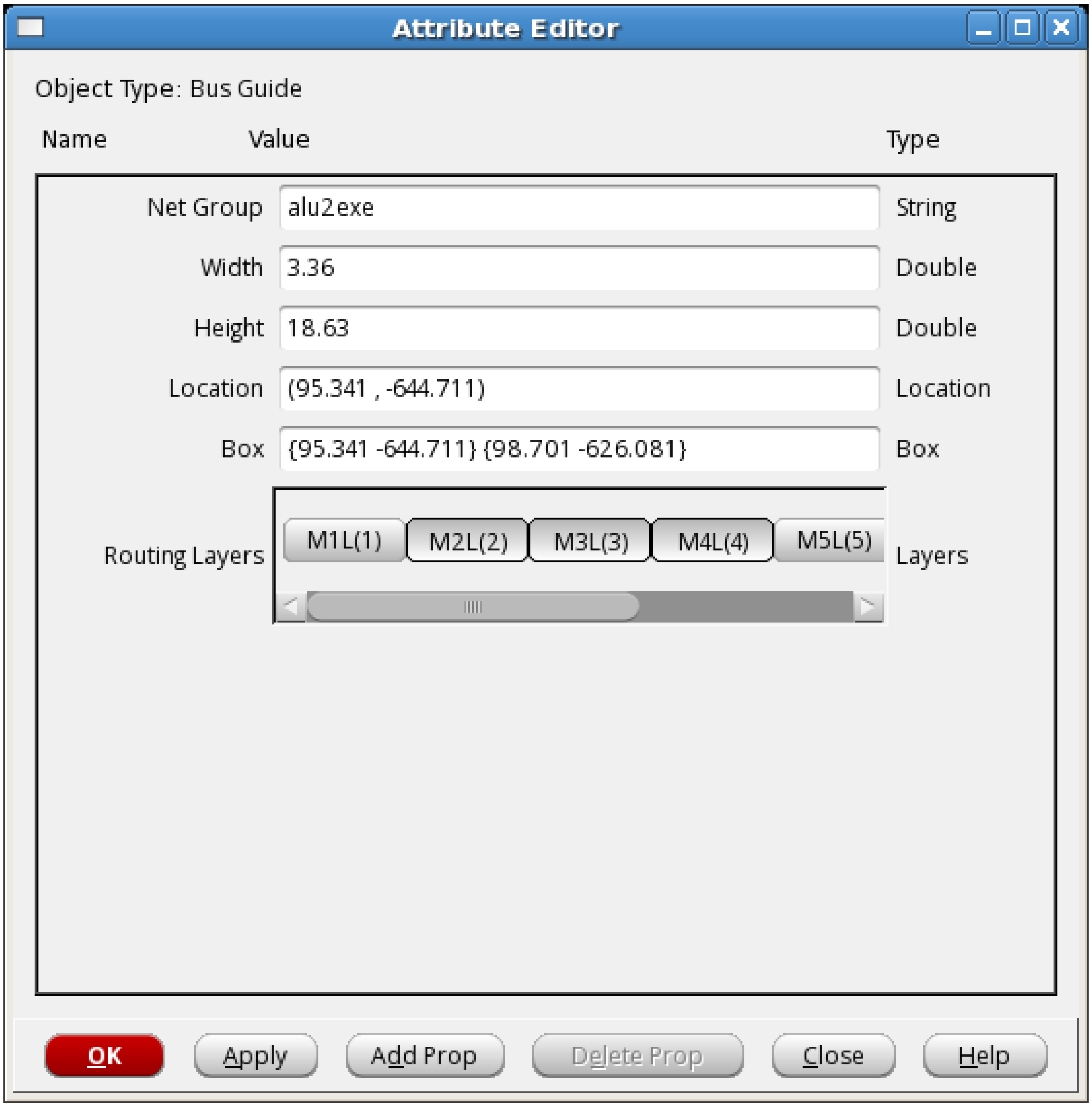
The following example illustrates the snapping behavior at the starting digitized point. The snapping occurs before you specify the second point:



The following example illustrates the snapping behavior at the end of a bus guide.



To view the attributes of a bus guide that you created, double-click the bus guide segment to display the Attribute Editor form as shown in the following example:



Object Type: Bus Guide

Name	Value	Type
Net Group	alu2exe	String
Width	3.36	Double
Height	18.63	Double
Location	(95.341 , -644.711)	Location
Box	{95.341 -644.711} {98.701 -626.081}	Box
Routing Layers	M1L(1) M2L(2) M3L(3) M4L(4) M5L(5)	Layers

A bus guide gets deleted when you delete its associated net group.

Using Text Commands

You can create and edit bus guides using the following text commands:

Commands	Use
`createBusGuide`	To create a bus guide segment

`deleteBusGuide`	To delete a bus guide **Note:** You can also delete a bus guide segment by selecting the segment and pressing the `Del` key on the keyboard.
`deselectBusGuide`	To deselect a bus guide segment
`editCutWire`	To cut a bus guide segment
`editMerge`	To merge two bus guide segments
`editSplit`	To split a bus guide segment
`selectBusGuide` `selectBusGuideSegment`	To select a bus guide segment. You can also use `selectBusGuideSegment` to select a bus guide segment with its specified bounding box.
`update_bus_guide`	To modify the layer or width information of an existing bus guide

For more information on the commands, see the "Bus Plan Commands " chapter in the *Text Command Reference*.

The following example describes the steps to create bus guides using text commands.

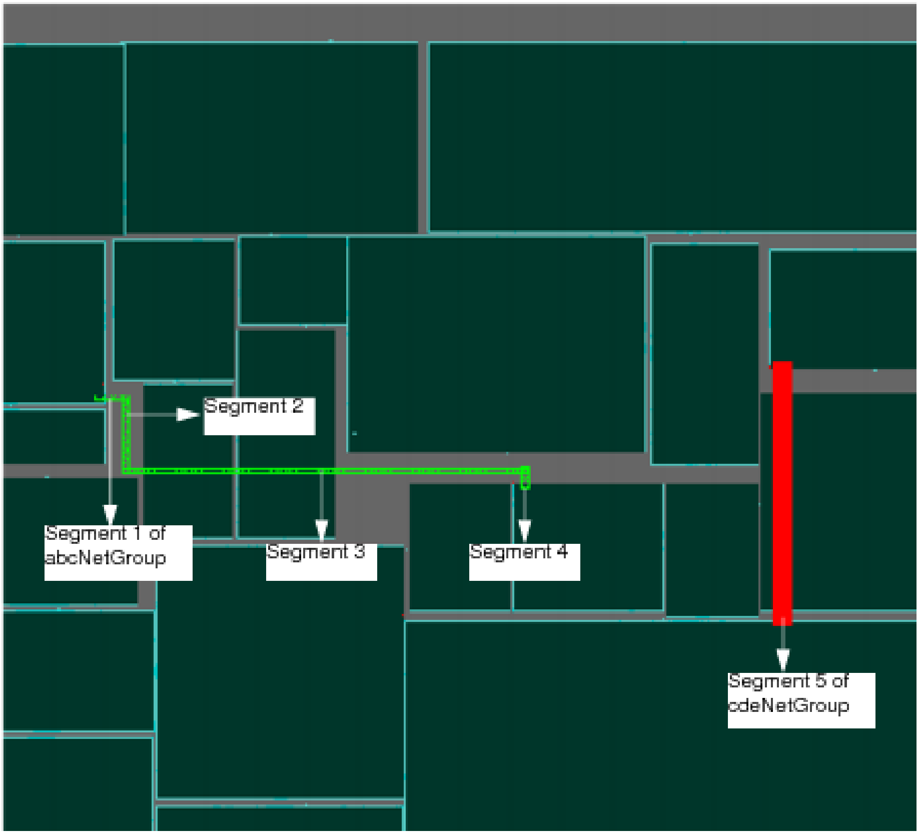
Example

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 Early Global Router

earlyGlobalRoute


#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**:

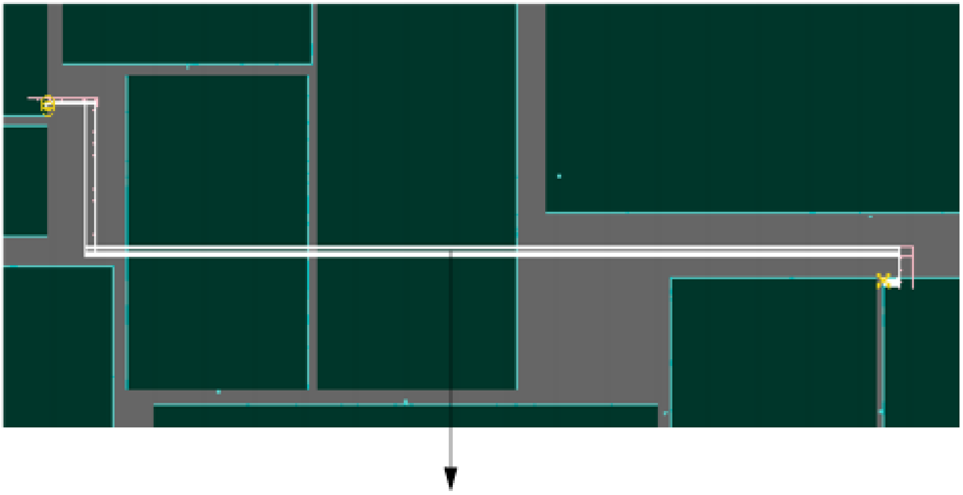
After running `createBusGuide` to create 5 segments



Segments 1, 2, 3, and 4 belong to `abcNetGroup`
Segment 5 belongs to `cdeNetGroup`

The following figure displays the routing of the bus `abcBusNet[0...31]`, routed within the bus guide area:

After running `placeDesign` and `earlyGlobalRoute`



All the 32-bus bits of `abcBusNet` group are routed within the bus guide area.

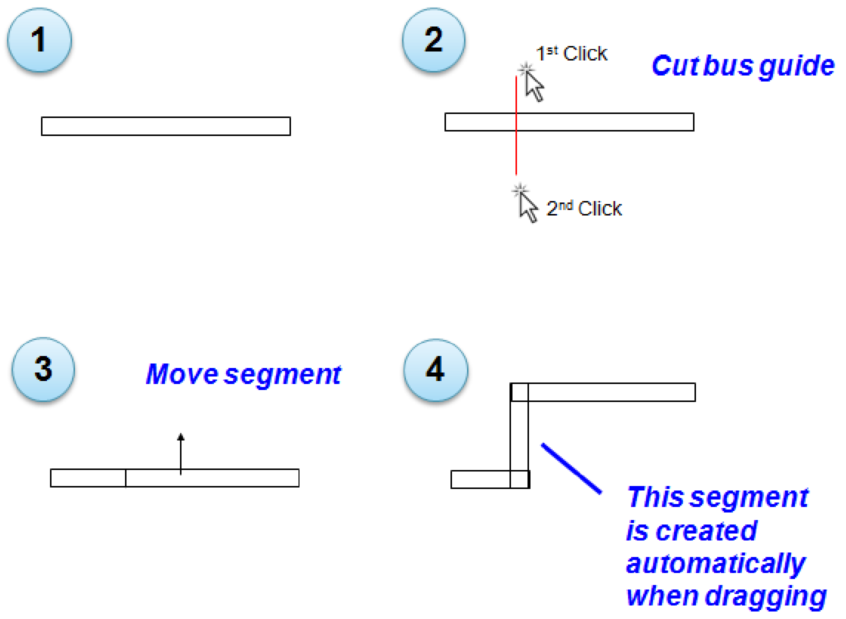
Moving and Stretching a Bus Guide

You can use the *Move Wire* () widget to move a bus guide segment, in the same way as you move a wire. The bus guide connection is maintained while moving a segment. Similarly, you can use the Stretch Wire () widget to stretch a bus guide. When you do so, it auto snaps to connect fully with other segments in the bus guide.

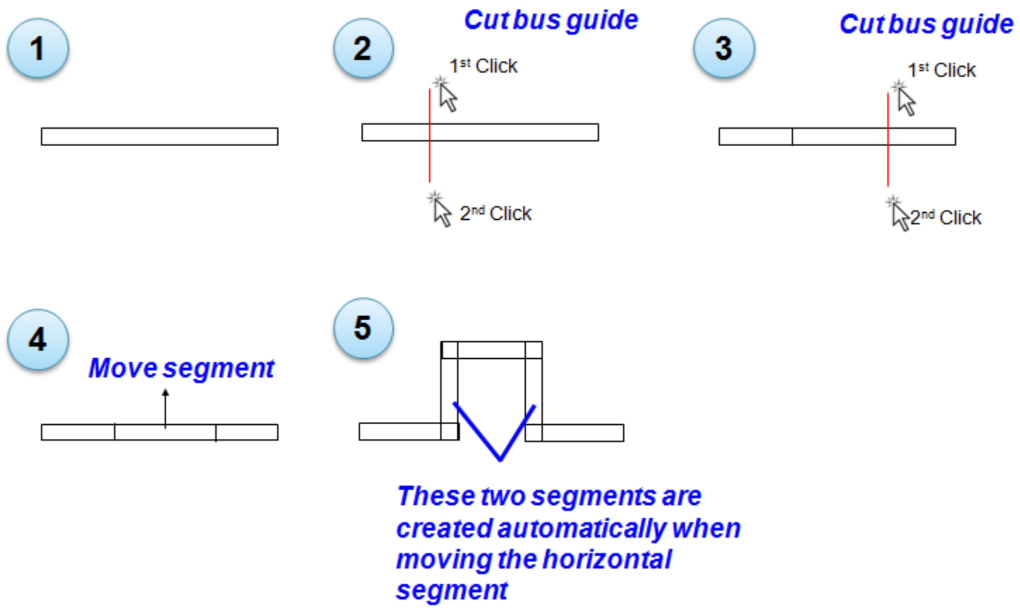
Cutting, Splitting, and Merging Bus Guides

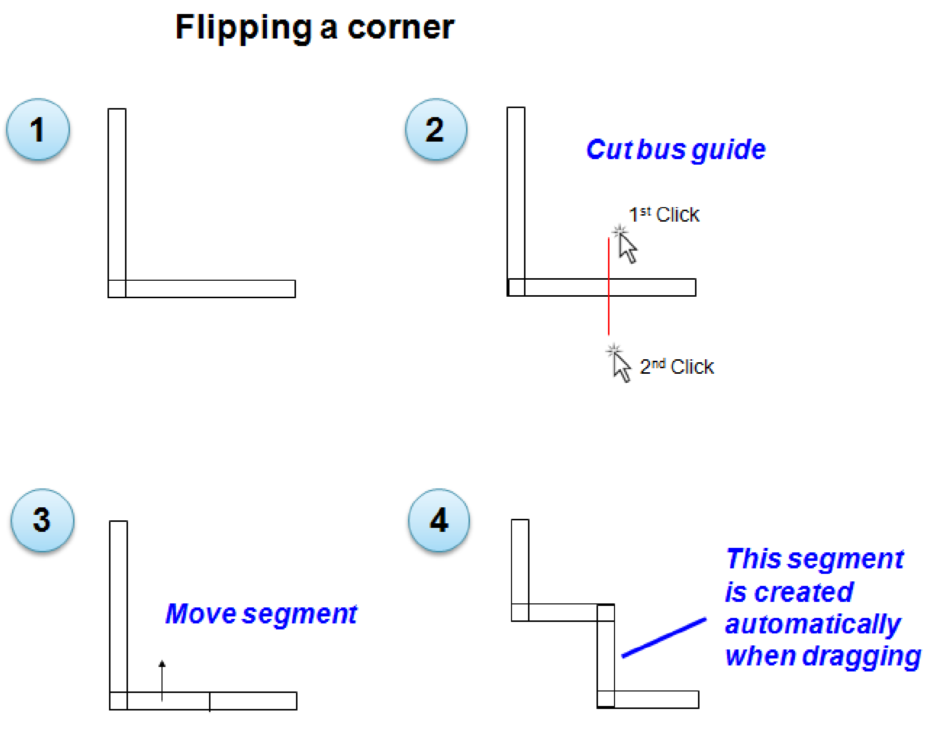
While editing bus guides, you might sometimes need to flip a corner of a bus guide or make a jog or detour on the bus guide. Jogs and detours are needed when the bus guide runs into a obstruction or you want to make a bus guide longer. To make jogs and detours, you would need to adjust bus guides by cutting and merging bus guide segments. Cutting and merging bus guides is possible through the `editCutWire`, `editSplit` , and `editMerge` wire edit commands. These commands support cutting, splitting, and merging of bus guides by default, in addition to wires. You can also use the *Split Selected Bus Guides* and *Merge Selected Bus Guides* buttons on the Edit Bus Guide form to split and merge bus guides.

Creating a jog



Creating a detour





Customizing the Bus Guide Display

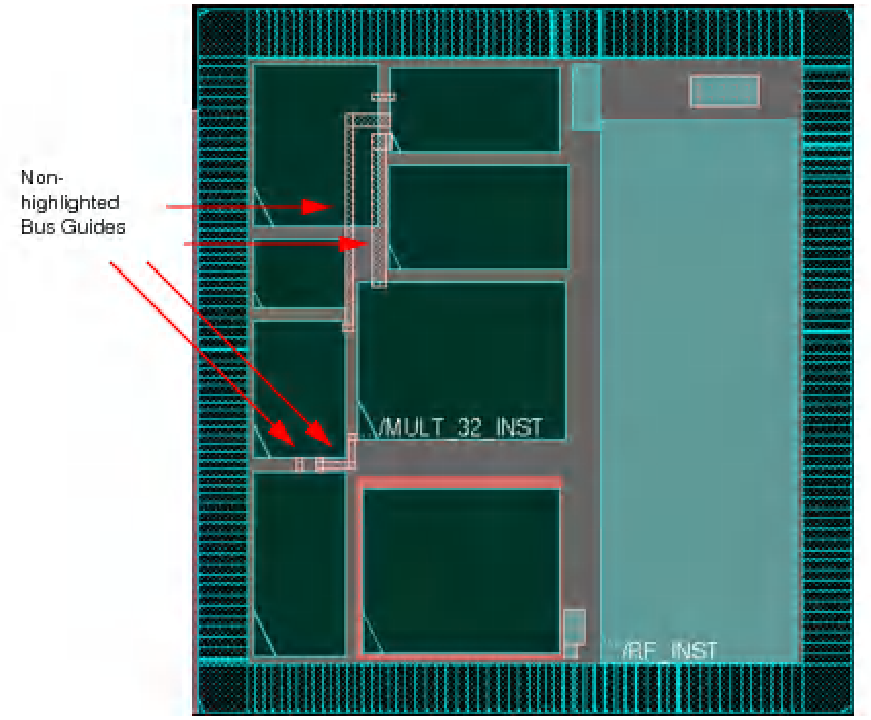
You can specify multiple colors for bus guide objects in the design, using the *Bus Guide Color Selection* form. (*Color Preferences* -- *Objects* -- *Bus Guide* -- *Bus Guide Color Selection*)

Highlighting and Dehighlighting the Bus Guide

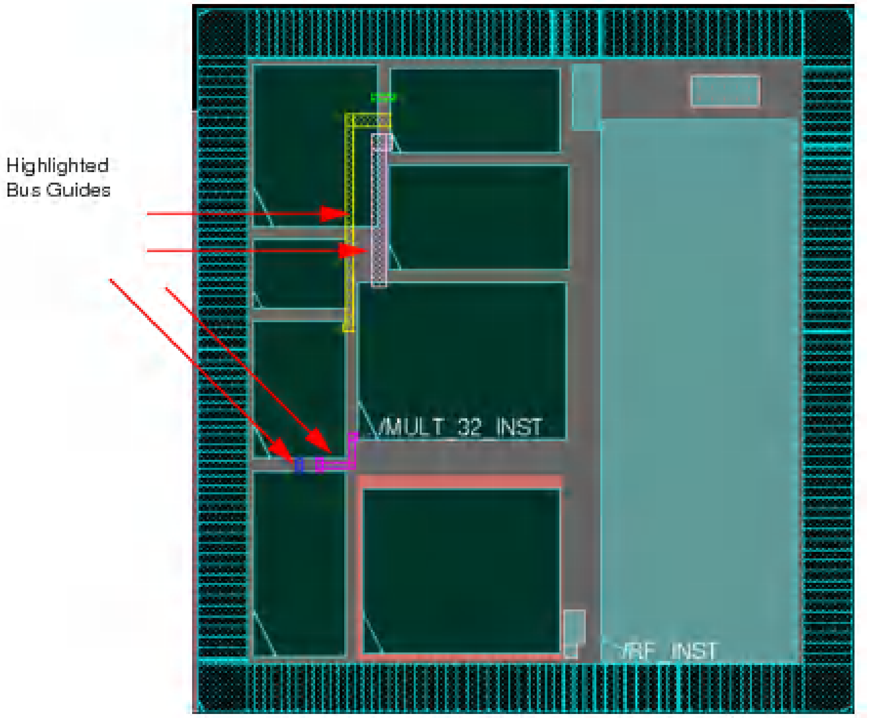
After specifying colors for bus guides, you can highlight the bus guides in the design using the *Edit -- Bus Guide* -- *Color* menu command.

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 run 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.

Limitations of Bus Planning

- Feedthrough insertion does not honor bus planning. Once you insert feedthroughs in the design, the existing bus guides will no longer be valid.
- The software currently does not provide checks to detect the following:
  - Overlapping bus guide segments on different layers
  - Complete bus guide coverage from source to sink
  - Complete coverage of placed pins
  - Enough room for routing.

Power Planning and Routing

- Power Planning:
  - Use the `addRing` command to generate rings with the CORERING and BLOCKRING shapes.
  - Use the `addStripe` command to generate stripes with the STRIPE shape. Use the `editPowerVia` command after the `addStripe` command to drop vias within macro power pins during stripe creation.
- Power Routing: Use the `sroute` command to connect to the following patterns:
  - `padPin`, `blockPin`, and `floatingStripe`: Connected nets are generated with the IOWIRE, BLOCKWIRE, and STRIPE shapes.
  - Standard cell pins and secondary power pins: Connected nets are generated with the FOLLOWPIN (in row area) and COREWIRE shapes (outside the row for further connection with other shapes).
  - `padPin` on left and right side of pad cells to generate `padring`: Connected nets are generated with the PADRING shape.
- VIAGEN Engine: Use it for special via insertion. The VIAGEN Engine:
  - Is called with the `addRing`, `addStripe`, and `sroute` commands to generate special vias.
  - Controls the behavior of special via editing command, `editPowerVia`.

Use Model of Power Planning and Routing Commands

The following table presents the use model of the power planning and routing commands:

Action Command	Mode Setup Command(s)
`addRing`	`setAddRingMode setViaGenMode`
`addStripe`	`setAddStripeMode setViaGenMode`
`sroute`	`setSrouteMode setViaGenMode`
`editPowerVia`	`setViaGenMode`

Each action command has one or two mode setup command(s) to control their characteristics. For non-default characteristics, you need to update/change the value of the mode setup command(s) before applying the action command. For example:

setViaGenMode -viarule_preference {VIAGEN67 VIAGEN56} -optimize_cross_via 1

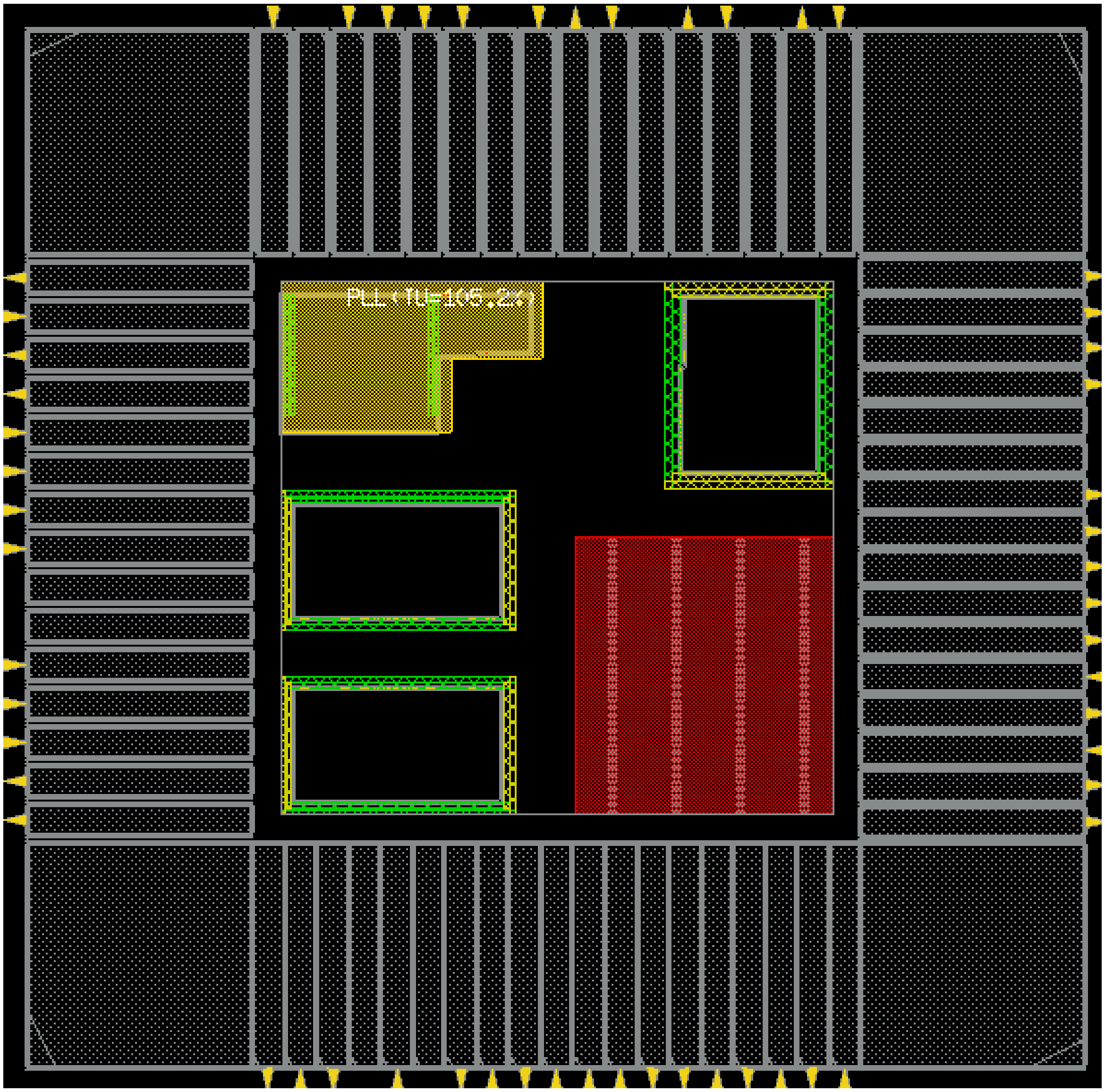
```
setAddStripeMode -stacked_via_bottom_layer M5 -stacked_via_top_layer M7
```

```
addStripe -nets VDD -width 0.144 -spacing 0.441 -set_to_set_distance 2.34 -layer M6 -
direction horizontal
```

Power Planning and Routing Flow

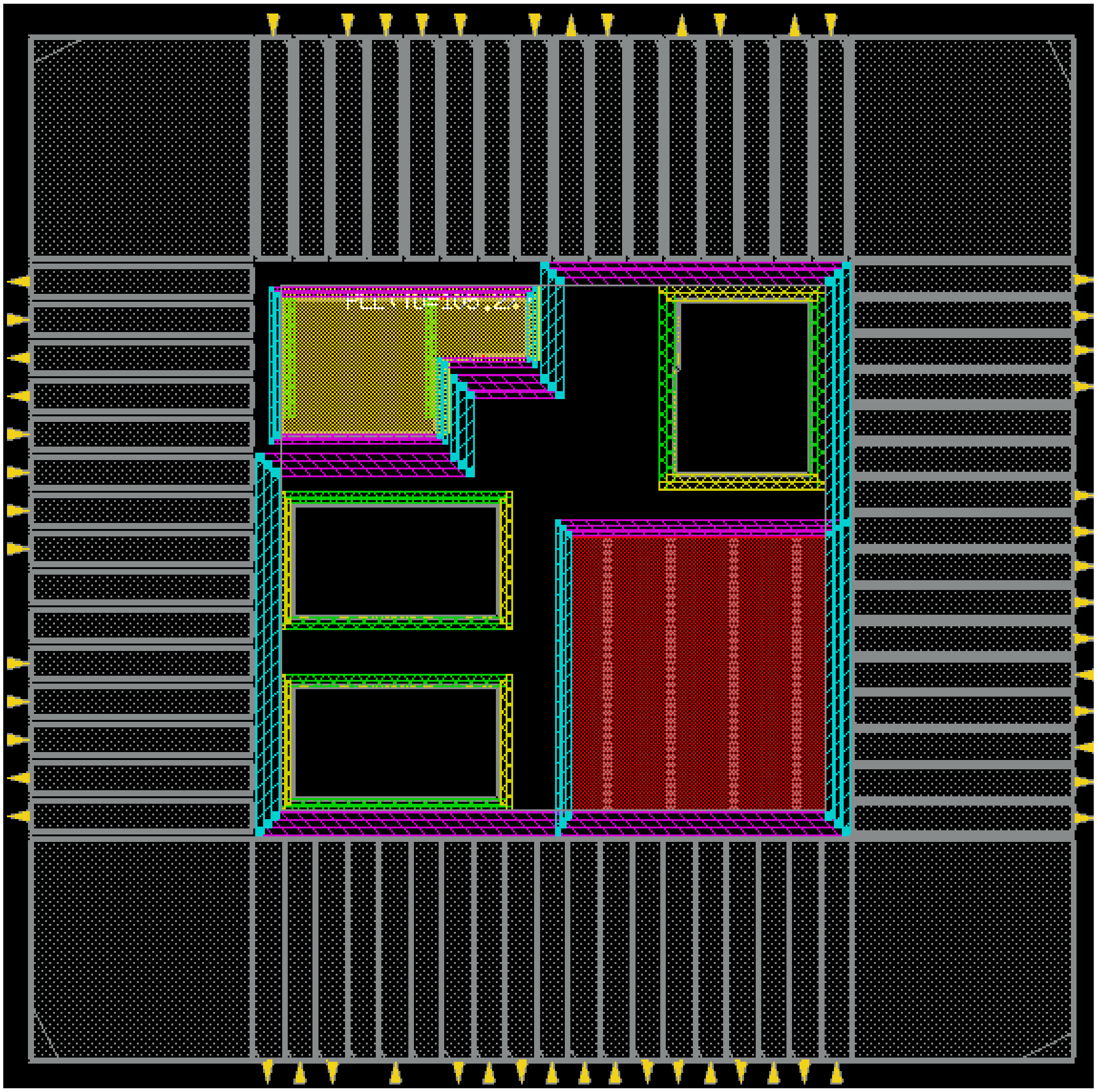
This section presents a real design as an example.

- A floorplan design is shown below, where all PG pins or tie-hi/tie-low pins have been connected with a global power/ground net:



- An example of adding rings is presented below. In the figure below:
  - Core rings are added around the core area (excluding the PLL area) for the `vdd_lp_s`, `vss`, and `vdd` nets.

- Block rings are added around the block PLL for the `Avss` and `Avdd` nets.
- Domain rings are added around non-default domain TDSP for the `vdd_lp_s`, `vss`, and `vdd` nets.



You need to run the following commands for this example:

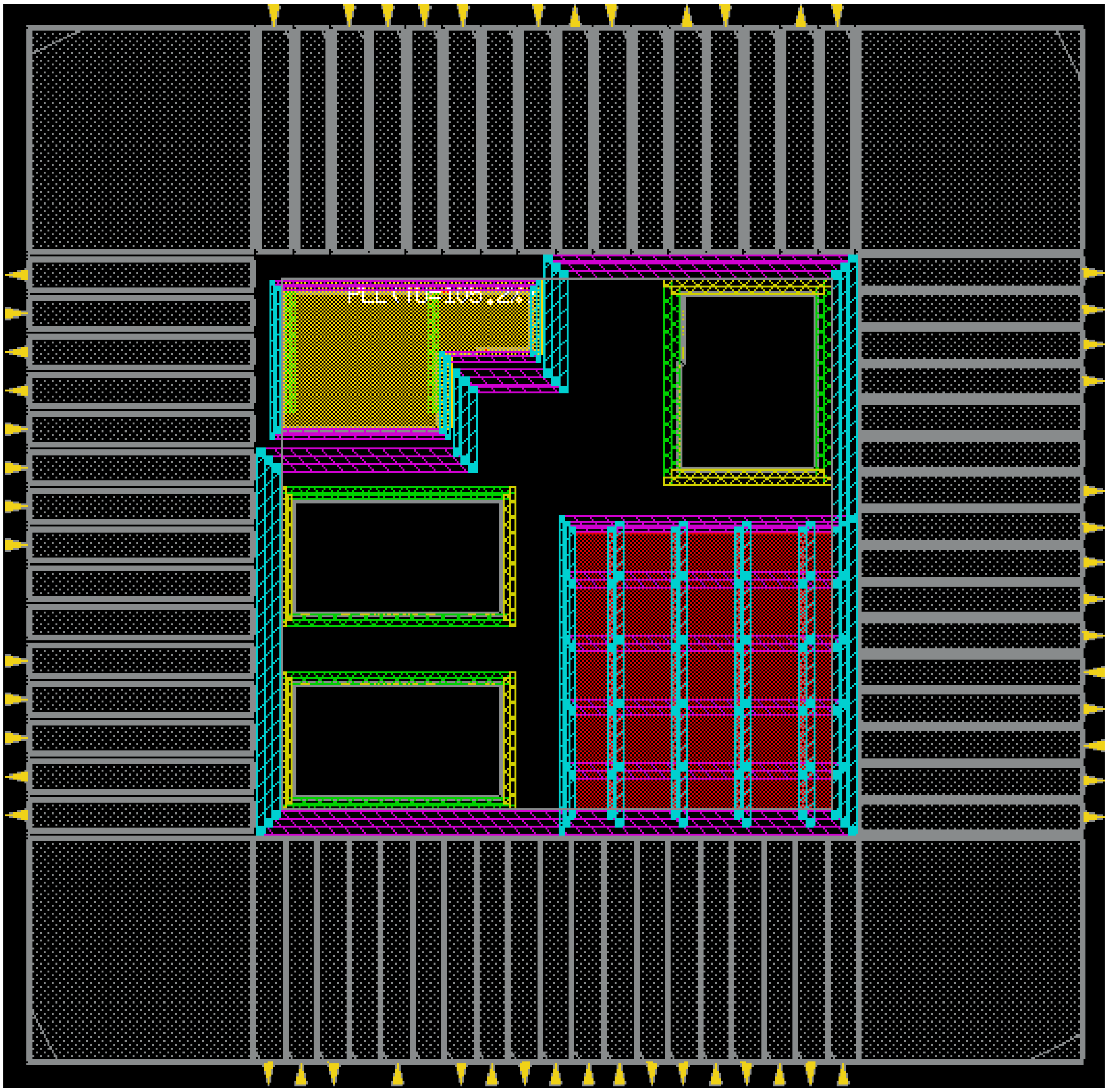
```
selectObject Group PLL
```

```
addRing -type core_rings -nets {vdd_lp_s vss vdd} -layer {top METAL7 bottom
METAL7 left METAL8 right METAL8} -offset 1 -width 8 -spacing 1.0 -
exclude_selected 1
```

```
deselectAll
```

```
selectInst DTMF_INST/PLLCLK_INST

addRing -type block_rings -nets {Avss Avdd} -around selected -layer {top
METAL7 bottom METAL7 left METAL8 right METAL8} -width 5 -spacing 1 -offset 1

deselectAll

selectObject Group TDSPCore

addRing -type block_rings -nets {vdd_lp_s vss vdd} -around power_domain -
layer {top METAL7 bottom METAL7 left METAL8 right METAL8} -width 5 -spacing
1 -offset 1

deselectAll
```

- Examples of adding stripes are the following:
    - In the figure below, domain stripes are added around the stripes over switch cell pins for each different layer:



You need to run the following commands for this example:

```
selectObject Group TDSPCore
```

```
setAddStripeMode -skip_via_on_pin {}
```

```
addStripe -over_pins 1 -nets vdd_lp_s -over_power_domain 1 -layer METAL4 -
width 8 -master HEAD16DM -pin_layer METAL2
```

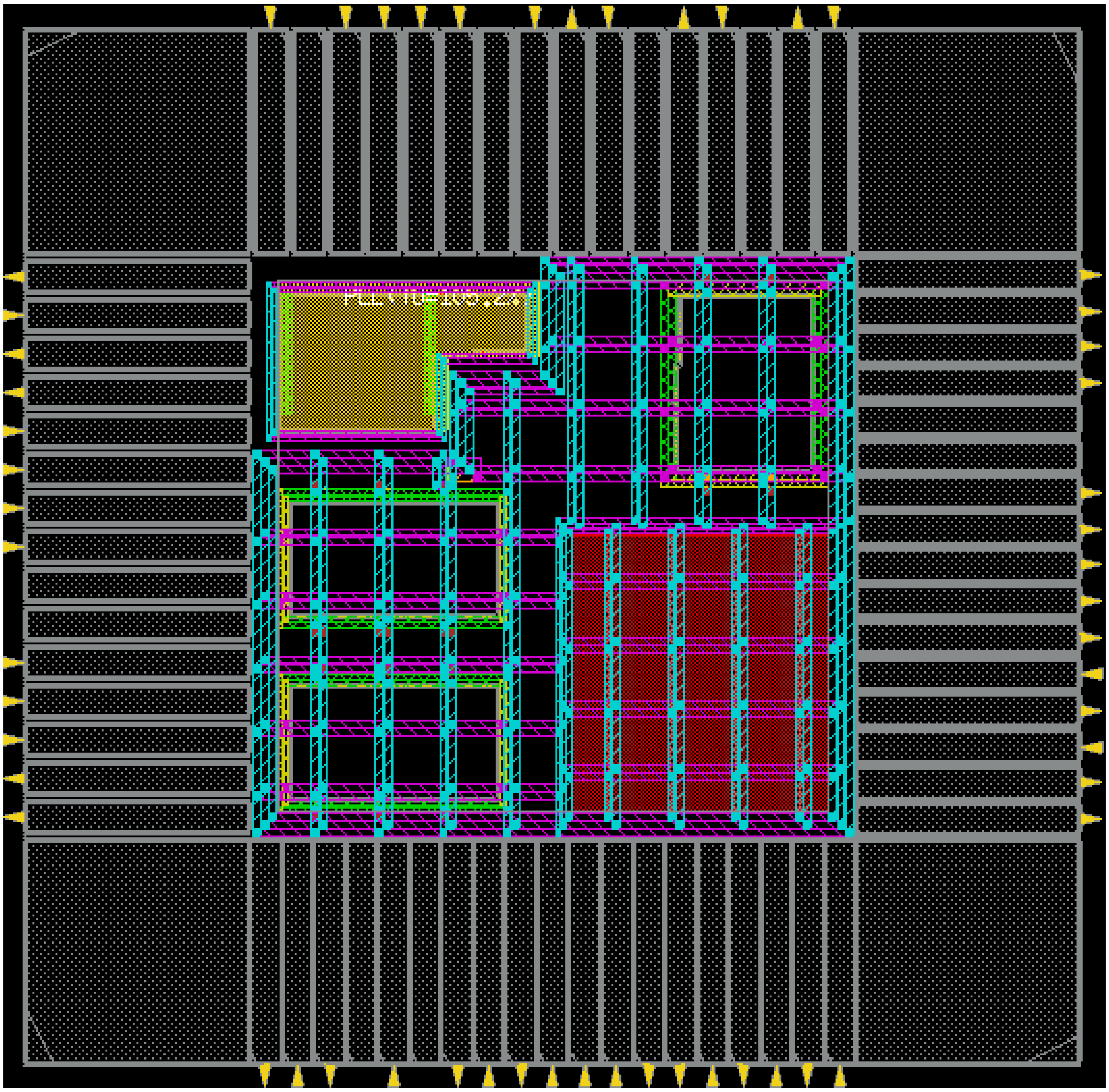
```
addStripe -over_power_domain 1 -nets vss -direction vertical -layer METAL4 -
width 8 -set_to_set_distance 70 -start_from left -start_offset 43.46 -
spacing 1
```

```
addStripe -over_power_domain 1 -nets {vdd_lp_s vss} -direction horizontal -
layer METAL7 -width 8 -set_to_set_distance 70 -start_from bottom -
```

```
start_offset 34.46 -spacing 1
```

```
addStripe -nets {vdd_lp_s vss} -over_power_domain 1 -layer METAL8 -width 8 -
set_to_set_distance 70 -start_from left -start_offset 34.46 -spacing 1
```

- In the figure below, stripes are added in the core area for each different layer:



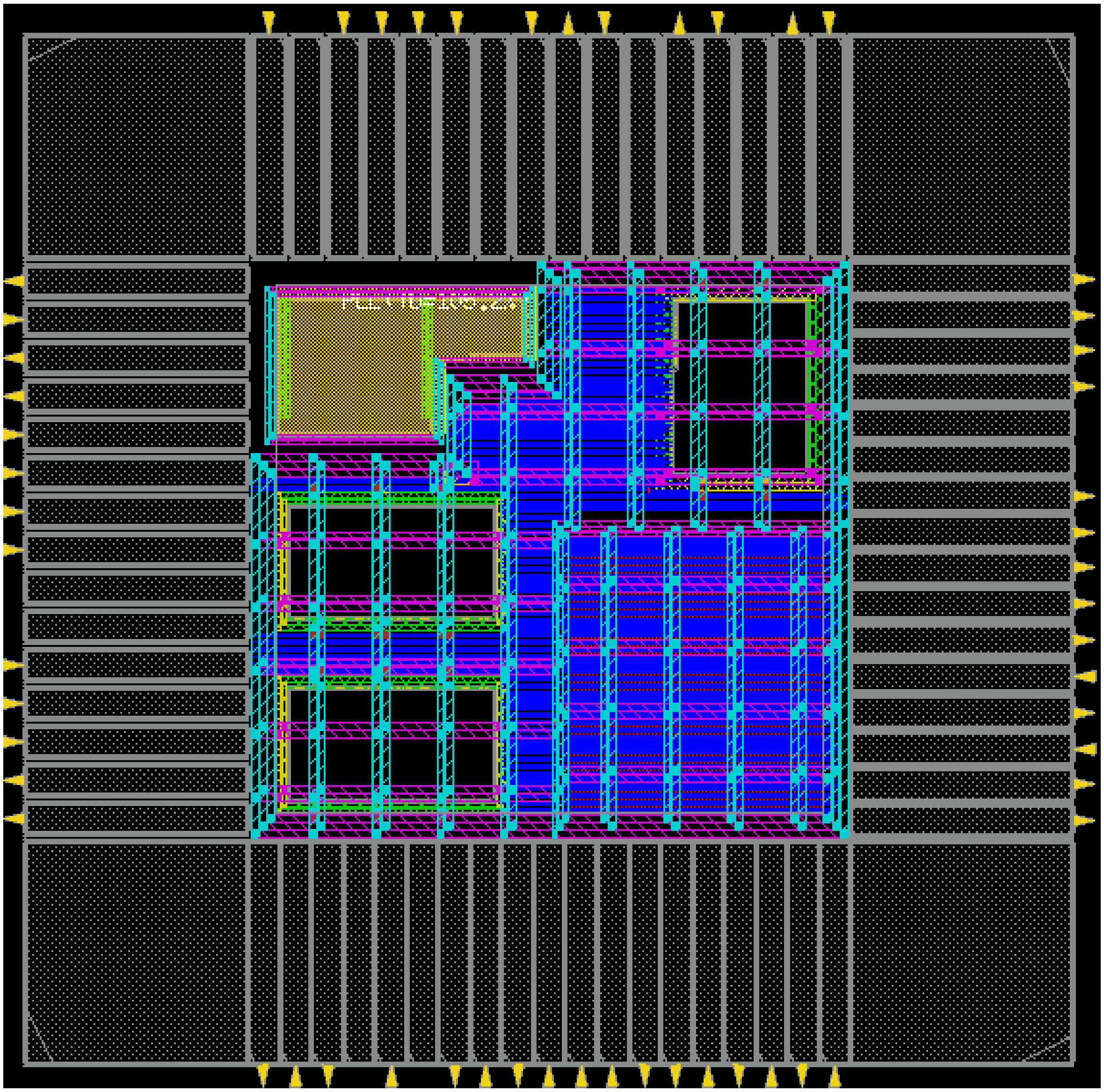
You need to run the following commands for this example:

```
addStripe -nets {vdd vss} -layer METAL4 -width 8 -set_to_set_distance 70 -
xleft_offset 37.9 -spacing 1
```

```
addStripe -nets {vdd vss} -direction horizontal -layer METAL7 -width 8 -
set_to_set_distance 70 -ybottom_offset 14 -spacing 1
```

```
addStripe -nets {vdd vss} -layer METAL8 -width 8 -set_to_set_distance 70 -
xleft_offset 37.9 -spacing 1
```

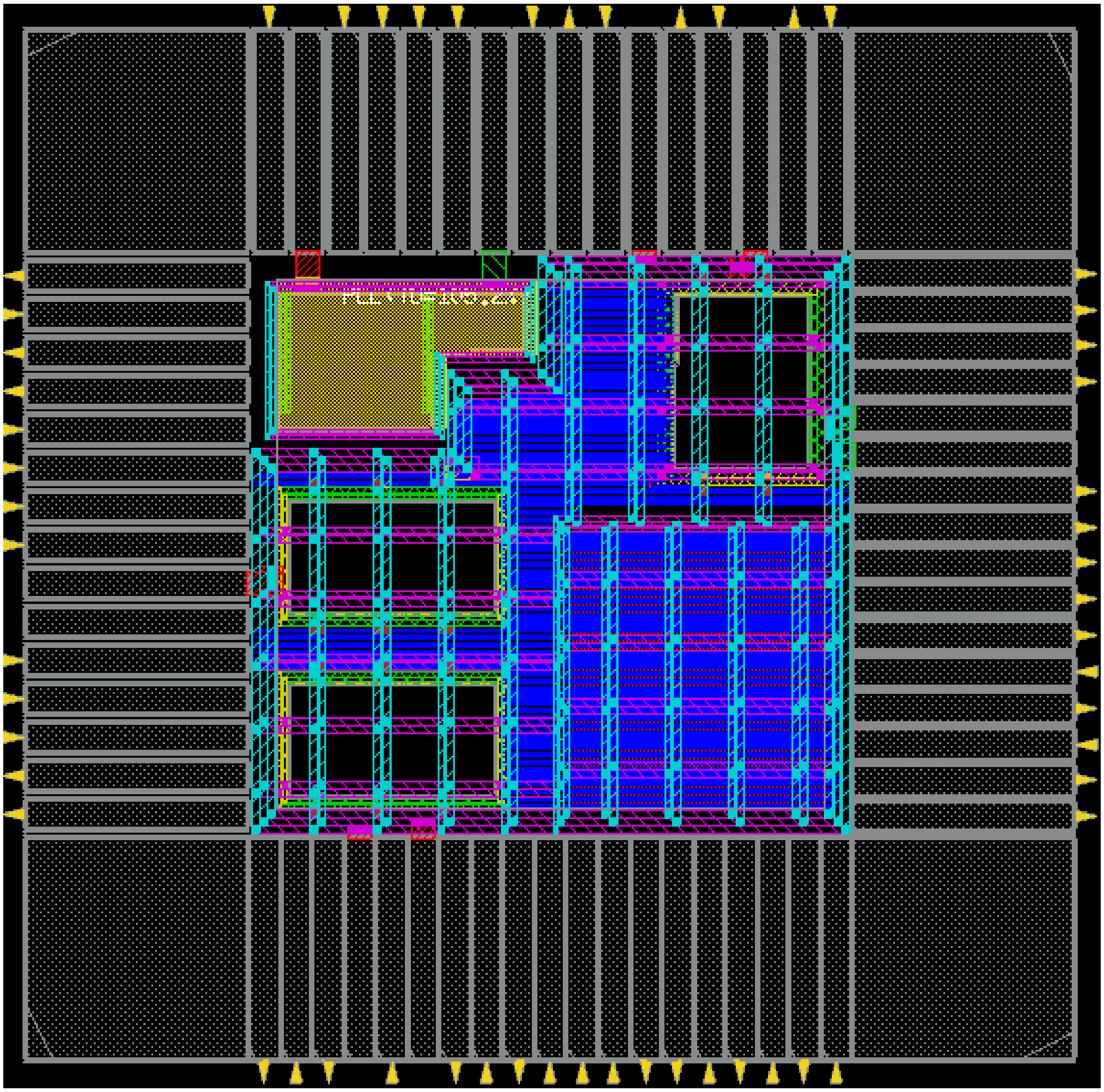
- In the figure below, standard cell pins are connected to form a followpin:



You need to run the following command for this example:

```
sroute -connect corePin
```

- In the figure below, padpins are connected to rings:



You need to run the following command for this example:

```
sroute -connect padPin
```

- The following command is an example of using the `saveDesign` command:

```
saveDesign lab/dtmf_pwr.enc
```

Related Information

- Generating Special Power Vias Using Viagen
- Generating Default Special Via
- Inserting Vias with a Specific Cutclass

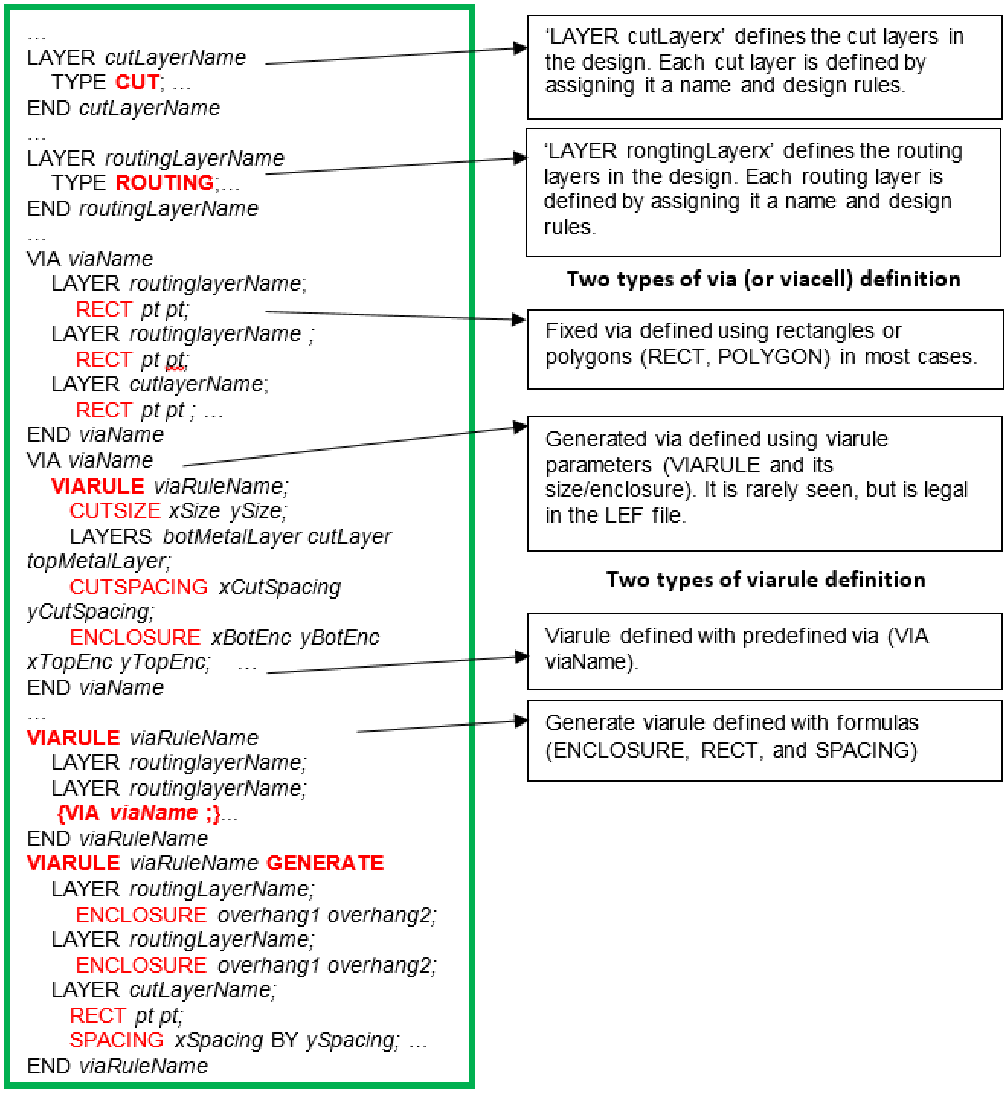
- Inserting a Via from Specific Viarule
- Trimming Redundant PG Stripes and Vias

Generating Special Power Vias Using Viagen

Defining Via and Viarule in LEF File

Refer to the following figure for understanding the definition of via and viarule in the LEF file. The via and viarule definition is presented in the green outline box in the following figure. The LEF section presented here does not contain the complete syntax, but helps in understanding the necessary syntax including the following definitions:

- Cut and routing layers
- Fixed and generated vias
- Predefined and generated viarules



```
…
LAYER cutLayerName
   TYPE CUT; …
END cutLayerName
…
LAYER routingLayerName
   TYPE ROUTING;…
END routingLayerName
…
VIA viaName
   LAYER routinglayerName;
      RECT pt pt;
   LAYER routinglayerName ;
      RECT pt pt;
   LAYER cutlayerName;
      RECT pt pt ; …
END viaName
VIA viaName
   VIARULE viaRuleName;
     CUTSIZE xSize ySize;
     LAYERS botMetalLayer cutLayer
topMetalLayer;
     CUTSPACING xCutSpacing
yCutSpacing;
     ENCLOSURE xBotEnc yBotEnc
xTopEnc yTopEnc;   …
END viaName
…
VIARULE viaRuleName
   LAYER routinglayerName;
   LAYER routinglayerName;
    {VIA viaName ;}…
END viaRuleName
VIARULE viaRuleName GENERATE
   LAYER routingLayerName;
      ENCLOSURE overhang1 overhang2;
   LAYER routingLayerName;
      ENCLOSURE overhang1 overhang2;
   LAYER cutLayerName;
     RECT pt pt;
     SPACING xSpacing BY ySpacing; …
END viaRuleName
```

'LAYER cutLayerx' defines the cut layers in the design. Each cut layer is defined by assigning it a name and design rules.

'LAYER rongtingLayerx' defines the routing layers in the design. Each routing layer is defined by assigning it a name and design rules.

Two types of via (or viacell) definition

Fixed via defined using rectangles or polygons (RECT, POLYGON) in most cases.

Generated via defined using viarule parameters (VIARULE and its size/enclosure). It is rarely seen, but is legal in the LEF file.

Two types of viarule definition

Viarule defined with predefined via (VIA viaName).

Generate viarule defined with formulas (ENCLOSURE, RECT, and SPACING)

Creating Auto Viarule

The special via engine, viagen, automatically creates 'auto viarule' internally for all the cutclasses based on the cut layer information available in the LEF file. Mostly, there are two (SINGLE and BAR), or at times three (SINGLE, BAR, and LARGE) cutclasses in each cut layer.

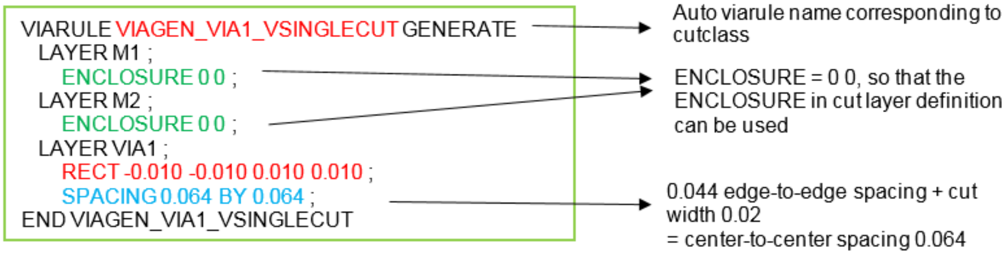
Following is an example of creating an auto viarule:

Cut layer definition:

```
PROPERTY LEF58_CUTCLASS "

       CUTCLASS VSINGLECUT WIDTH 0.020 LENGTH 0.020 CUTS 1 ;

       CUTCLASS VDOUBLECUT WIDTH 0.020 LENGTH 0.050 CUTS 2 ;" ;

PROPERTY LEF58_SPACINGTABLE "
```

```
        SPACINGTABLE  PRL -0.022  MAXXY

        CUTCLASS        VSINGLECUT      VDOUBLECUT SIDE

        VSINGLECUT      0.044 0.044     0.044 0.044

        VDOUBLECUT SIDE 0.044 0.044     0.044 0.044

PROPERTY LEF58_ENCLOSURE "

        ENCLOSURE CUTCLASS VSINGLECUT BELOW 0.020 0.000 ;

        ENCLOSURE CUTCLASS VSINGLECUT ABOVE 0.030 0.000 ;

        ENCLOSURE CUTCLASS VDOUBLECUT BELOW END 0.030 SIDE 0.000 ;

        ENCLOSURE CUTCLASS VDOUBLECUT BELOW END 0.030 SIDE 0.003 WIDTH 0.026 ;

        …
```

Auto viarules are created for VSIGNLECUT and VDOUBLECUT, as shown below:

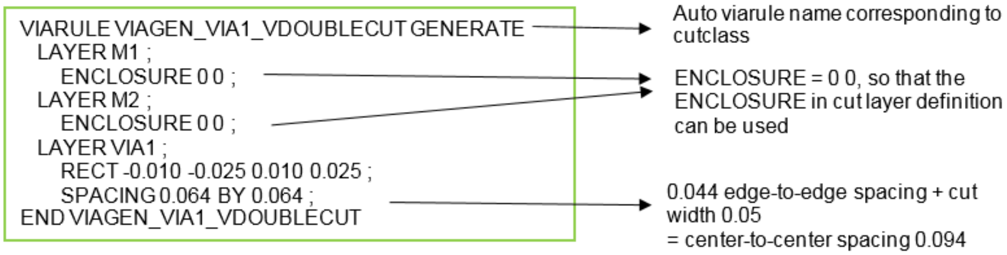


```
VIARULE VIAGEN_VIA1_VSINGLECUT GENERATE
  LAYER M1 ;
    ENCLOSURE 0 0 ;
  LAYER M2 ;
    ENCLOSURE 0 0 ;
  LAYER VIA1 ;
    RECT -0.010 -0.010 0.010 0.010 ;
    SPACING 0.064 BY 0.064 ;
END VIAGEN_VIA1_VSINGLECUT
```

Auto viarule name corresponding to cutclass

ENCLOSURE = 0 0, so that the ENCLOSURE in cut layer definition can be used

0.044 edge-to-edge spacing + cut width 0.02
= center-to-center spacing 0.064



```
VIARULE VIAGEN_VIA1_VDOUBLECUT GENERATE
  LAYER M1 ;
    ENCLOSURE 0 0 ;
  LAYER M2 ;
    ENCLOSURE 0 0 ;
  LAYER VIA1 ;
    RECT -0.010 -0.025 0.010 0.025 ;
    SPACING 0.064 BY 0.064 ;
END VIAGEN_VIA1_VDOUBLECUT
```

Auto viarule name corresponding to cutclass

ENCLOSURE = 0 0, so that the ENCLOSURE in cut layer definition can be used

0.044 edge-to-edge spacing + cut width 0.05
= center-to-center spacing 0.094

The database order of viarules is as follows:

1. Viarules in the LEF file's order
2. Auto viarules; larger cut area first (cutclasses from LARGE to SINGLE)

Related Information

Power Planning and Routing

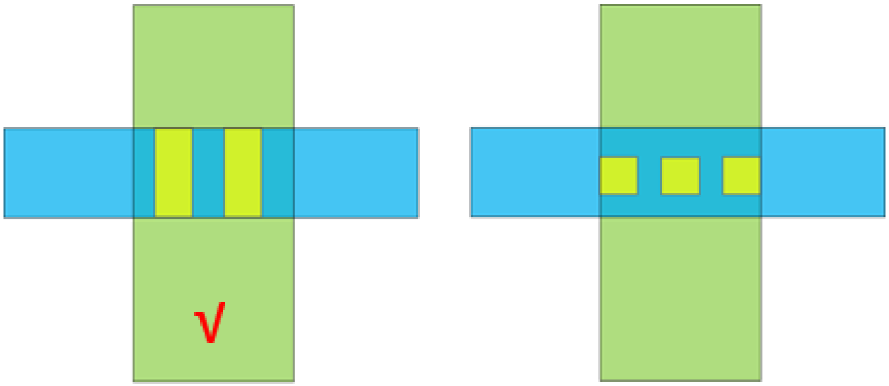
Generating Default Special Via

By default, the special via generation engine, viagen, uses only the viarule (including viarule available in the LEF file and internally generated auto viarule) to generate vias. The new viarule pickup criteria depends on the maximum total cut area for attaining the best connection and lowest IR drop.

To generate a default special via:

1. Find the crossover of two same-net special segments.
2. Go through all the vias generated by the various viarules generated in the database, including the LEF file viarule and auto viarule.
3. Pick up a viarule that generates a via with the maximum total cut area, and use it. Refer to the following examples:

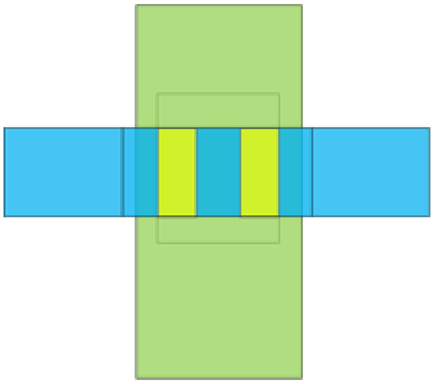
   Example 1: non-cross via, 2-bar-cut via wins 3-square-cut via



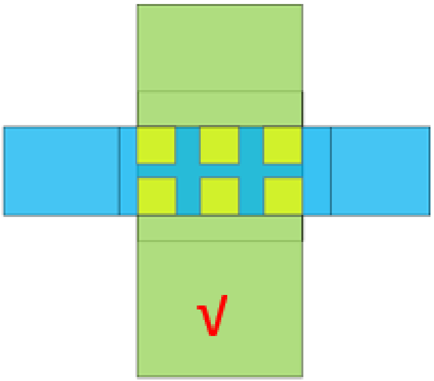
2-bar-cut via

3-square-cut via

Example 2: cross via, 6-square-cut via wins 2-bar-cut via



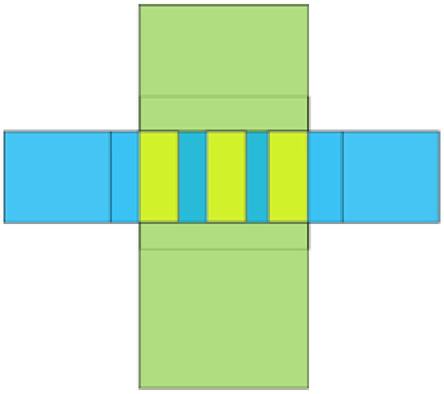
2-bar-cut via



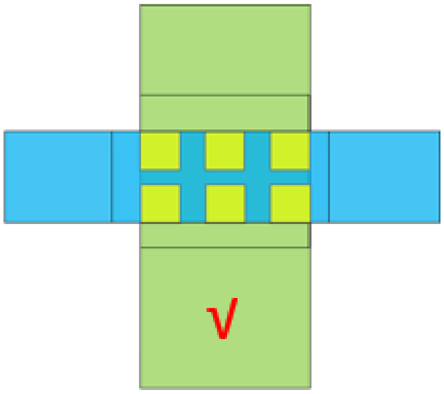
6-square-cut via

If different viarules generate same cut-area via at a specific intersection, the viarule in the front of the database is used. So as per the LEF file order, the LEF viarule is used first, followed by auto viarule (LARGE-> BAR -> SQUARE). Refer to the following example:

Example 3: A via with various cutclasses has the same total cut area, the 6-square-cut via from viarule list over the 3-bar-cut via from viarule list, as presented below:



3-bar-cut via



6-square-cut via

4. Once a viarule is determined, viagen gets parameters from the viarule as a soft guide, and the technical definition in cut/metal layer as the hard guide for generating vias. By default, the enclosure defined in the LEF viarule is ignored and the cut layer enclosure value is used. This option is controlled by running the following command:

```
setViaGenMode -ignore_viarule_enclosure {true | false}
```

Default: `true`

Related Information

Power Planning and Routing

Inserting Vias with a Specific Cutclass

Mostly, there are two (SINGLE and BAR), or at times three (SINGLE, BAR, and LARGE) cutclasses in each cut layer; refer to the LEF syntax. The following topics are covered under this section:

- Inserting Vias with the Preferred Cutclass
- Distinguishing Cutclasses Internally

Inserting Vias with the Preferred Cutclass

Use the following command parameter to insert a via with a specific cutclass:

```
setViaGenMode -cutclass_preference {default | {[square] [bar] [large]}
| cutclass_name_list | file_name}
```

Default: `default`

The following table illustrates the behavior of each of the above mentioned parameter values. If a specified cutclass cannot generate a DRC-clean via, the intersection remains OPEN.

Parameter Value	Description	Next Preference, if Violation Exists
`default`	Specifies the viarule with the cutclass that has the maximum total cut area	Next maximum total cut area
`square`	Uses the viarules with the SIGNLE cutclass (the cut width and length are the same and the smallest in that layer, for example, 0.032 by 0.032).	OPEN
`bar`	Uses the viarules with the DOUBLE cutclass (the cut width is the smallest; For example, 0.032 by 0.08, is equal to two SINGLE cutclasses in the LEF definition).	
`large`	Uses the viarules with LARGE cutclass (neither the cut width, nor the length, are smallest; For example, 0.08 by 0.08, is equal to 4 SINGLE cutclasses in the LEF definition).	

`cutclass_list`	Uses the viarules with the specified cutclass, by the order of the bigger to smaller total cut area.	OPEN
`file_name`	Uses the viarules with the cutclass written in a specified file, by the order of bigger to smaller total cut area. Various arguments can be separated by space or line break.	OPEN

The option setting does not support incremental specification. The last cutclass preference setting is used.

Distinguishing Cutclasses Internally

To differentiate cutclasses, the area of cut is used. For example, there are three cutclasses, the minimum size cut is ‘square’, the medium size is ‘bar’, and the maximum size is ‘large’; There are two cutclasses, the smaller size cut is ‘square’, the bigger size cut is ‘bar’.

When you specify ‘{[square] [bar] [large]}’, there should be two or three cutclasses defined in the LEF file. If the tool detects that there are more than three or less than two cutclasses, the setting is ignored.

Related Information

Power Planning and Routing

Inserting a Via from Specific Viarule

To insert the exact via or viarule defined in the LEF file, run the following command before the commands used for inserting an issued via:

```
setViaGenMode -viarule_preference {default | predefined | generated | list of via
rule/cell names | file name}
```

Default: `default`

The above command changes the priority of via or viarule to be considered by viagen, as mentioned in the following table. If a specified via or viarule cannot generate a DRC-clean via, the intersection remains OPEN.

Parameter Value	Description	Next Preference, if Violation Exists

`default`	Specifies the viarule that generates vias with the maximum total cut area.	Next maximum total cut area
`predefined`	Uses predefined via rules, by the order of total cut area (viarule without keyword).	OPEN
`generated`	Uses generated via rules (viarule with keyword GENERATE).	OPEN
`list of via rule/cell names`	Uses the list of specific via rules and/or via cells specified in the parameter. If via cells and rules are listed, viagen ranks vias with higher priority and uses the one-fit principle; ranks viarules with lower priority than vias, and uses the maximum total cut area principle. Refer to the following example: `setViaGenMode -viarule_preference {via23_1 viarule23_a via23_2 viarule23_b}` In this case, the specified vias and viarules are arranged in the following priority: 1. Priority: `via23_1 > via23_2 > viarule23_a > viarule23_b` 2. The tool tries `via23_1` first; If no violation occurs, it uses `via23_1` (one-fit principle). 3. If it fails, it tries `via23_2`; If no violation occurs, it uses `via23_2`; 4. If it fails, it tries the viarule between `viarule23_a` and `viarule23_b`, which generates the via with the bigger total cut area. 5. If it fails, it tries the viarule between `viarule23_a` and `viarule23_b`, which generates the via with the smaller total cut area. 6. If it fails, the intersection remains OPEN.	OPEN
`file name`	Uses the vias and/or via rules listed in the LEF file. It uses the same principle as that used for the list of via rule/cell names.	OPEN

The option setting supports incremental specification. All the specified vias and viarules in various commands are ranked with vias with higher priority. Refer to the following example:

```
setViaGenMode -viarule_preference {via23_1 viarule23_a via23_2 viarule23_b}
```

```
setViaGenMode -viarule_preference {viarule23_c via23_3}
```

In this case, the specified vias and viarules are arranged in the following priority:

1. `via23_1 > via23_2 > via23_3 > viarule23_a > viarule23_b > viarule23_c`
2. The tool tries `via23_1 via23_2 via23_3` with the one-fit principle.
3. If all the vias fail, it tries `viarule23_a viarule23_b viarule23_c` in the order of generating the bigger total cut area first, followed by the smallest total cut area at the last.
4. If all the vias and viarules fail, the intersection remains OPEN.

Note: It is not recommended to set both the `-cutclass_preference` and `-viarule_preference` parameters at the same time. Still if it happens, `-cutclass_preference` has higher priority. It means that the specified cutclass in `-viarule_preference` is attempted first, then the specified cutclass in LEF viarules and auto viarules; and then other vias and viarules in `-viarule_preference`.

Related Information

Power Planning and Routing

Trimming Redundant PG Stripes and Vias

You may trim redundant PG stripes and vias using an existing power grid pattern based on IR drop analysis reports. You the following command to trim the redundant PG stripes and vias:

```
trim_pg
```

This eliminates the need to manually adjust the power structure to reduce power grid pessimism for area/timing/power improvement in a design.

To trim a stripe:

1. Use `trim_pg -type stripe`.
2. Specify only one LEF layer name with the `-layer` parameter.
3. Specify only one string with the `-pattern` parameter. If more than one values are specified, the command displays an error.

For example, the following command trims 50% of the `VDD` stripes on a `M6` metal layer inside a `{0 0 500 500}` box:

```
trim_pg -net VDD -type stripe -layer M6 -area {0 0 500 500} -pattern 10
```

To trim a via (stack):

1. Use `trim_pg -type via`.
2. Specify two LEF layer names with the `-layer` parameter. The first layer stripe is used as the pattern reference.
3. Specify one or more strings with the `-pattern` parameter to indicate the via trip pattern.

For example, the following command trims a via (stack) along with half of the `M9 VDD` stripes, and one-third of its stack vias between the `M9` and `M6` metal layers:

```
trim_pg -net VDD -type via -layer {M9 M6} -area {0 0 500 500} -pattern {110 1}
```

Trimming Stripes

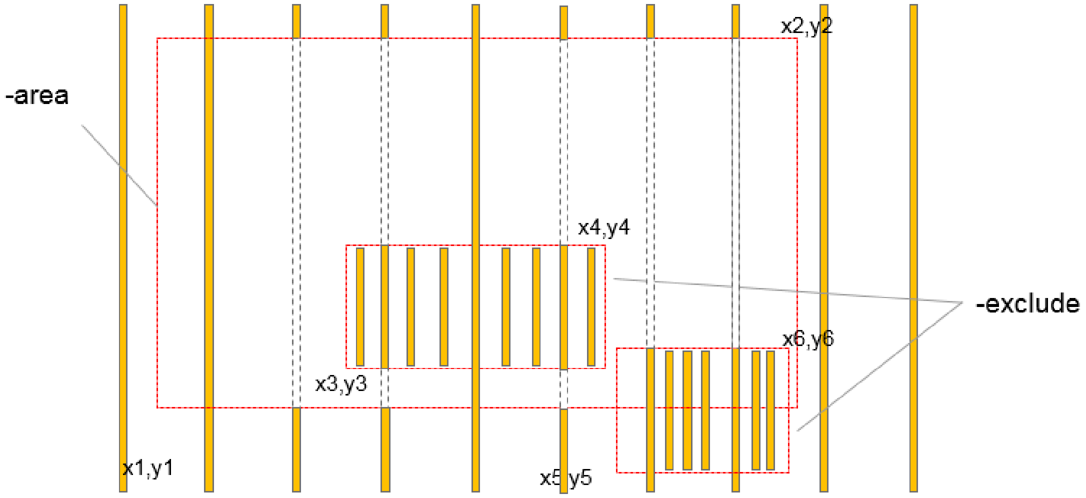
You need to specify a box for trimming a stripe. In some cases, the specifies box area also contains some local PG grid. You may keep the local grid or trim them separately using a different pattern with a combination of the following three parameters:

- `-area {`*`x1 y1 x2 y2`*`}`: Specifies the trimming area. It defines the stripes and vias to be trimmed. The stripe overlapped with the box defined by `-area` is trimmed. By default, the entire core area is trimmed.
- `-exclude {{`*`lx1 ly1 ux1 uy1`*`} {`*`lx2 ly2 ux2 uy2`*`}...}`: Excludes the box areas for the stripes and vias in which the part of stripes need not to be trimmed. This parameter is used along with the `-area` parameter to specify the exclusion areas. If a stripe partially overlaps with the exclusion area, the overlapping segment of the stripe is not trimmed. The exclusion box can partially overlap the `-area` box.
- `-exclude_local_grid {{`*`lx1 ly1 ux1 uy1`*`} {`*`lx2 ly2 ux2 uy2`*`}...}`: Excludes the areas in which the whole stripes are skipped for trimming. If a stripe is completely inside one of the `-exclude_local_grid box` (can touch the boundary), it is excluded from the trim candidates and is not counted when applying the trim pattern. It is useful when trimming is done on a large area but there are local stripes in this area, which could affect the global trim pattern. The exclusion box can partially overlap the `-area` box.

For example, if you want to trim 50% of the stripes inside an area. But there are some local stripes over a hard macro in this area. If you use `-exclude`, the local stripes are be trimmed. But they are counted when applying the `10` pattern. Through this, the long stripes are not trimmed. Refer to the command below:

```
trim_pg -net VDD -type stripe -layer M6 -area {x1 y1 x2 y2}
\
-exclude {x3 y3 x4 y4 x5 y5 x6 y6} -pattern 10
```

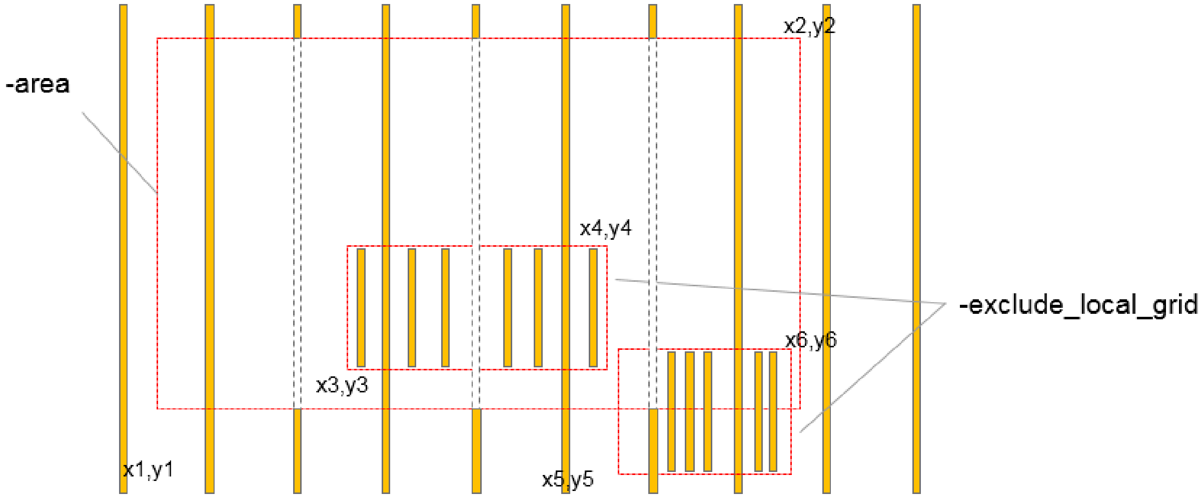
Following is the output:



For trimming the long stripes, use the `-exclude_local_grid` parameter. It prevents the local stripes from being counted in pattern and ensure that the desired pattern applied to the global stripes. You may also use it with the `-area` and `-exclude` parameters, as shown below:

```
trim_pg -net VDD -type stripe -layer M6 -area {x1 y1 x2 y2} \ -exclude_local_grid {x3
y3 x4 y4 x5 y5 x6 y6} -pattern 10
```

Following is the output:



Sorting Stripes

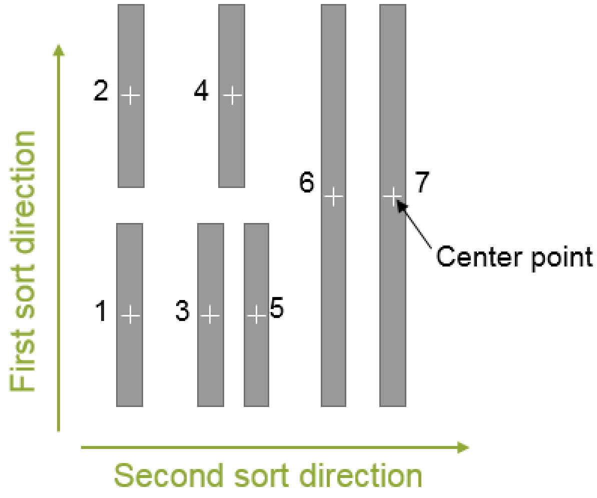
You may sort stripes before trimming a pattern. You may select stripes as trimming candidates by sorting the `-area` and `-exclude_local_grid` parameters using the following steps:

1. Determine the running direction of most of the stripes by comparing the length of the vertical

and horizontal edges of the stripe (vertical direction in the example below).

2. Sort the stripes based on their center point, by the direction derived from the previous step.
3. Sort stripes in the orthogonal direction (horizontal direction in the example below).

Refer to the example below:



Trimming a Pattern

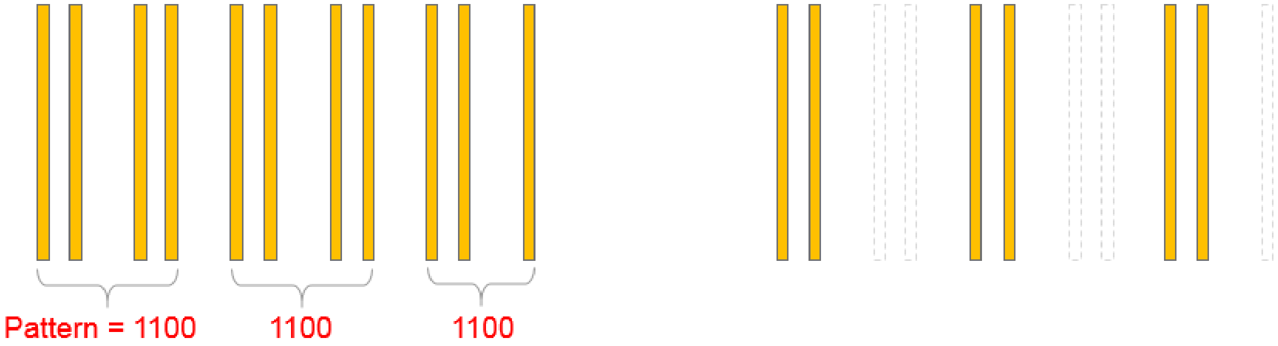
You may trim a pattern using a sequence of `0` and `1`, where `0` signifies to trim the stripe and `1` to keep the stripe. Follow the steps below:

1. Starting from the first stripe in the sorted queue, trim or keep it according to the pattern.
2. Repeat the pattern till the last stripe in the queue.

Refer to the example below:

```
trim_pg -net VDD -type stripe -layer M6 -area {0 0 500 500} -pattern 1100
```

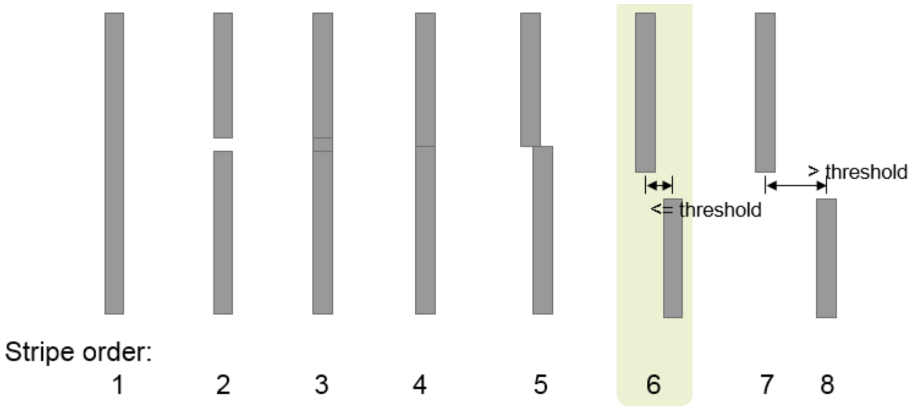
Following is the output:



Setting-up Threshold for Stripe Merging

In some cases, a long stripe is broken-up in to multiple segments such as through hard macro and power domain. The complexity of a real design power grid can also have the abut, overlapping, or slightly misaligned stripes. While trimming a PG pattern, you may consider it as a single stripe rather than multiple segments. You may define a spacing threshold along with the secondary sort direction. If the centerline spacing between two or more stripes is smaller than the value provided by the `-threshold` parameter, they are treated as a single stripe while trimming or keeping the stripe.

Refer to the example below:



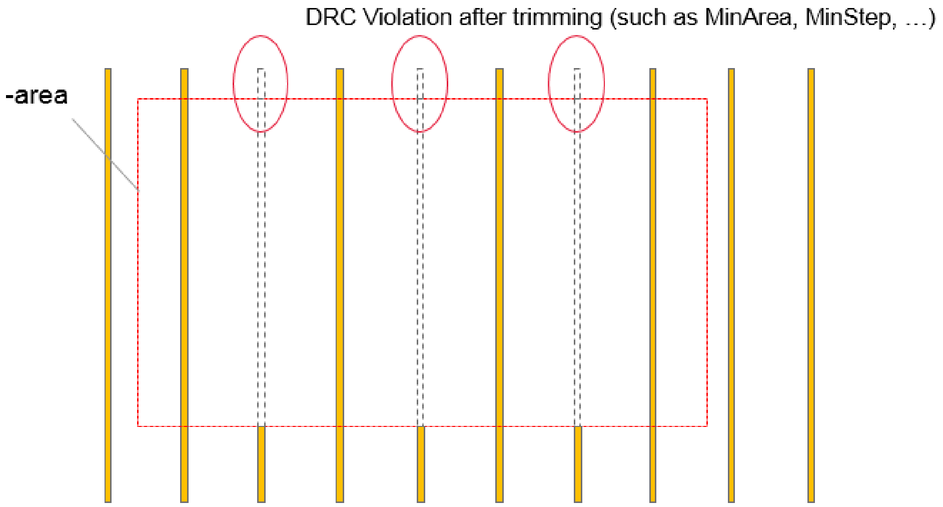
You may sort the spacing between the adjacent stripes as below:

```
-d6 < d3 < d5 < Threshold < d7 = d4 < d2 < d1 = d8
```

While grouping, start from two adjacent stripes with the smallest spacing, if those are lesses than the threshold, group them. In this example, the smallest spacing is `d6` between C and H, and is lesser than the threshold. So C and H are grouped. Then continue with the next smallest spacing. In this example, it is `d5`. But C is already grouped with H, measure the spacing between G and H. However `d5+d6` is greater than the threshold. So G will not be grouped with C and H. Refer to the final grouping below:

Avoiding New DRCs

While trimming stripes, you need to avoid creation of new DRCs. Otherwise, trimming will be continued on the new DRCs created. For example, the stripe segments highlighted in the red circles below are also trimmed, if new DRCs are created after trimming the stripe segments in the specified area:

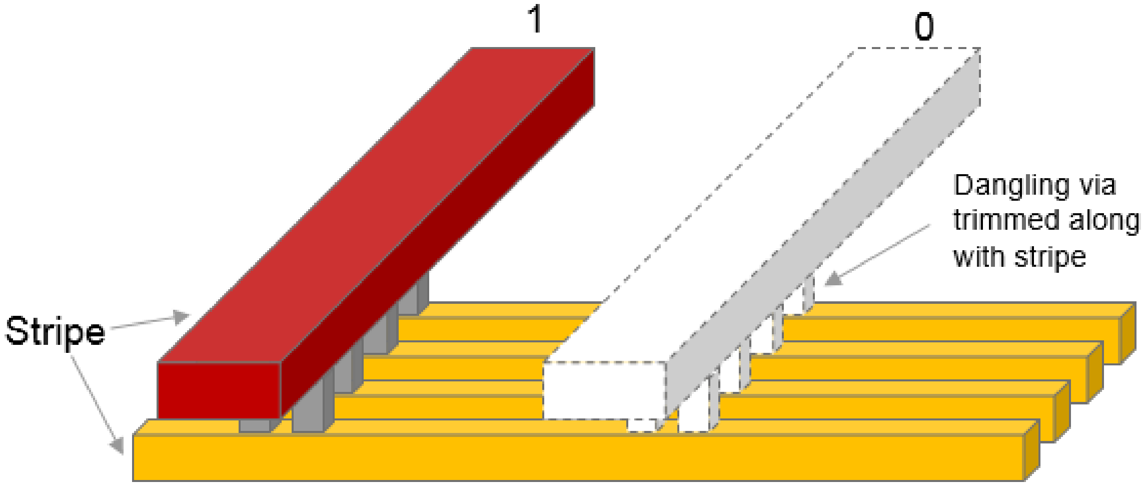

DRC Violation after trimming (such as MinArea, MinStep, ...)
-area

Keeping Dangling Vias While Trimming Stripes

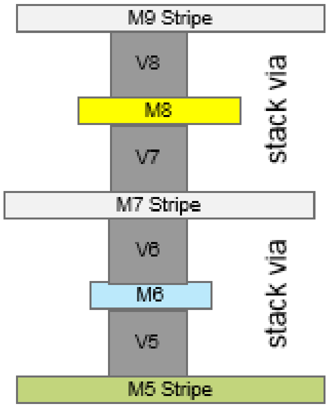
While trimming a stripe, if the connecting via (including stack via) becomes dangling, it is also be trimmed, by default. Use the `-keep_dangling_via` parameter to keep the dangling vias while trimming stripes. It searchs the stack vias before trimming stripe. For example, while trimming the following pattern:

```
M7 stripe – V7/V8 – M9 stripe – V9 – M10 stripe
```

After trimming the `M9` stripe, `M7` and `M10` are still connected through the `V7/V8/V9` stack. But as this stack is actually formed by the previous `V7/V8` and `V9`, so you need to check dangling for `V7/V8` and `V9` respectively.



The stack via formed after trimming is not treated as a stack via. For example, after trimming the `M7` stripe, `M5` and `M9` are still connected through the `V5~V8` stack. But keeping dangling vias while trimming checks the stack vias before trimming. So the `V7/V8` and `V5/V6` stacks will be removed by default.



Trimming Vias

Power vias, especially the stack vias, connecting low and high layer stripes often occupy lots of routing tracks, and brings congestions and routing difficulties to the signal routing. So you need to remove some of the stack vias on the power grid to save more routing resource. In such cases, trim the power vias and keep the stripes.

Use the `trim_pg -layer` parameter to accept one or two metal layers for stripe and via trimming. While trimming a stripe, specify *`layer_name_1`* on which the stripes are to be trimmed. While trimming via, specify *`layer_name_2`* between which the stack vias are to be trimmed. The first layer is used as the reference while applying the via trim pattern. For instance, `trim_pg -layer M9` means to trim `M9` stripes.

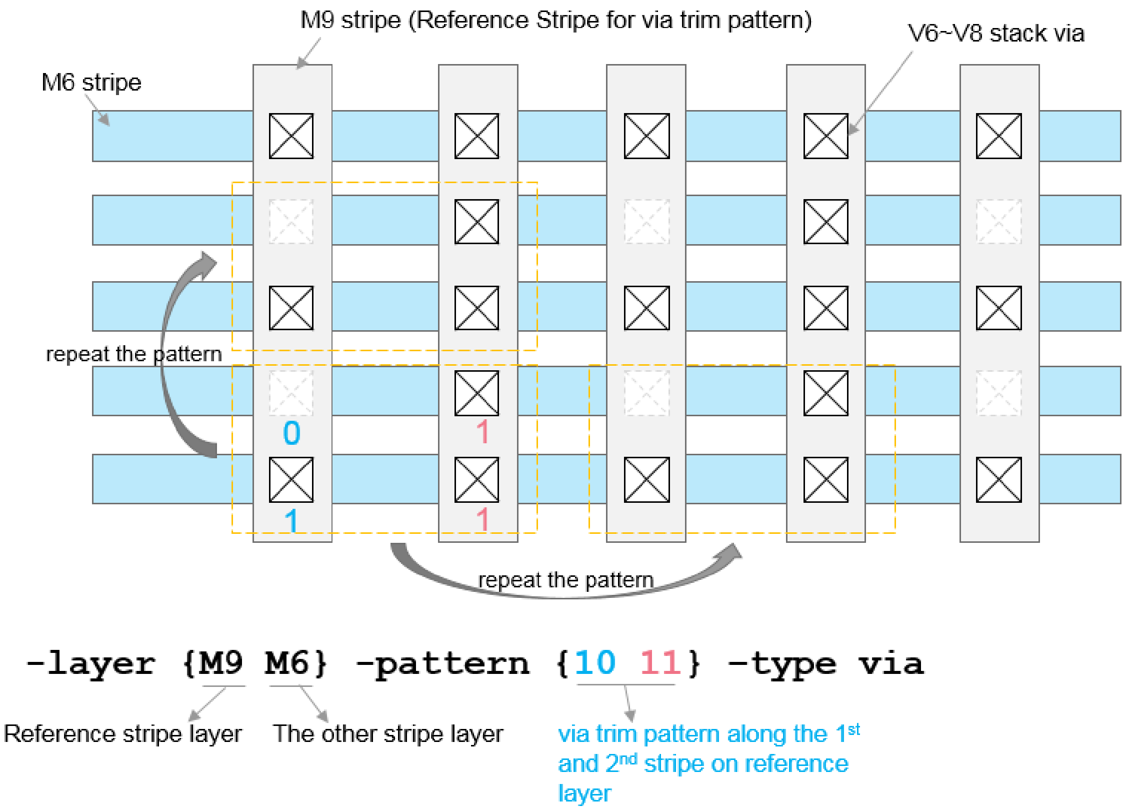
When two layer names specified, it means to trim via that connects the stripe on the two layers. The first layer stripe is used as the pattern reference, because two-dimensional pattern is needed for via. For einstance, `trim_pg -layer {M9 M6}` means to trim stack via between the `M9` and `M6` stripes, with `M9` stripe as the pattern reference.

Use the `trim_pg -pattern` parameter to accept one or multiple strings.

- While trimming one string pattern, `trim_pg -pattern 10` means to trim every other stripe.
- While trimming via, both one and multiple string patterns are allowed. `trim_pg -pattern {10 11}` means to trim every other via along number `1, 3, 5, 7,` … of the reference stripe, and keep all the vias along `2, 4, 6, 8,` … stripes.

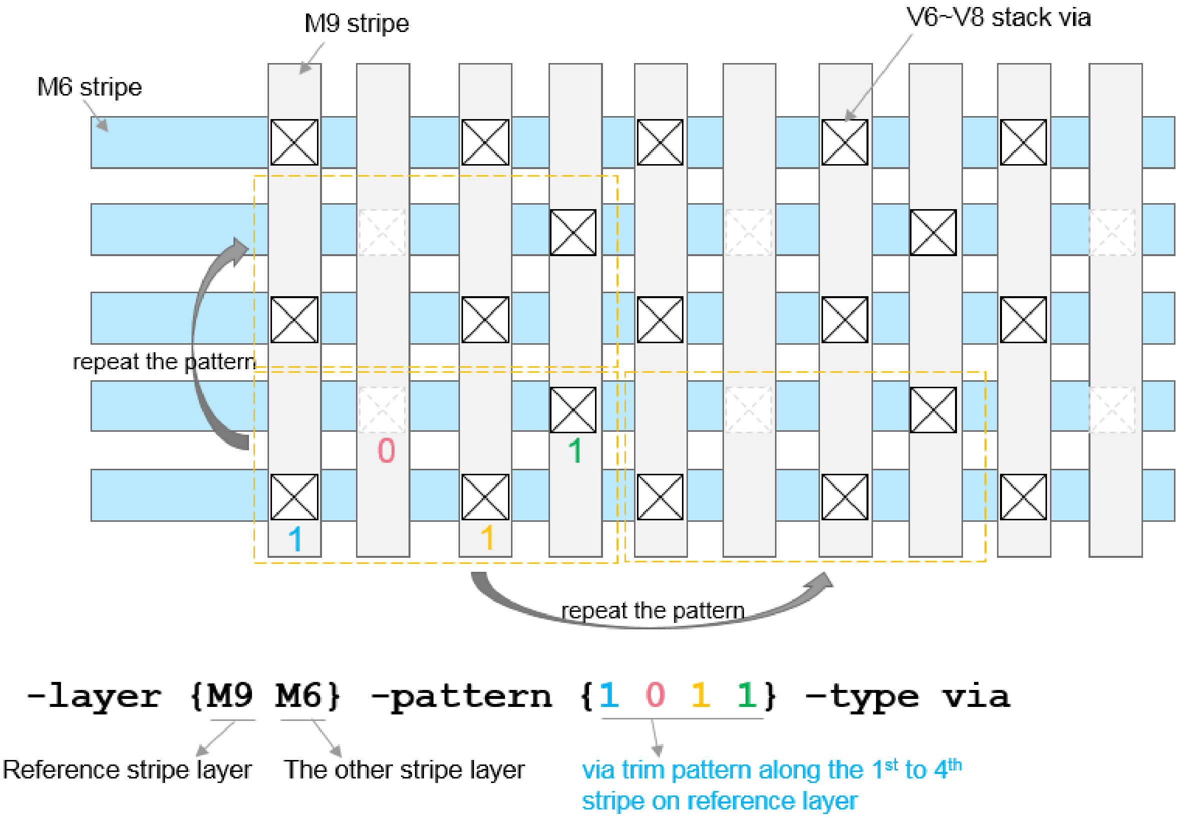
Examples

- Following is an example of trimming stack via between the `M9` and `M6` stripes:


M9 stripe (Reference Stripe for via trim pattern)
V6~V8 stack via
M6 stripe
repeat the pattern
0
1
1
1
repeat the pattern
Reference stripe layer
The other stripe layer
via trim pattern along the 1st and 2nd stripe on reference layer

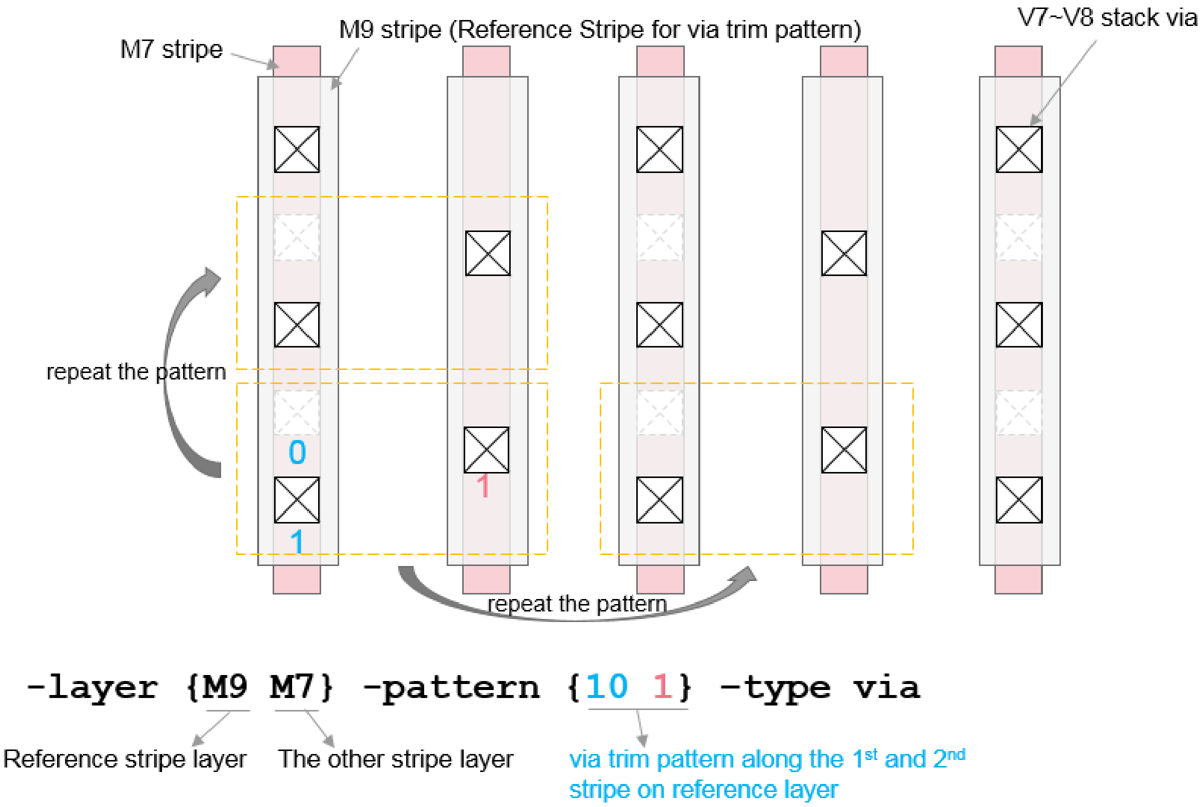
```
-layer {M9 M6} -pattern {10 11} -type via
```

- Following is an example of trimming stack via between the `M9` and `M6` stripes with checker board pattern:



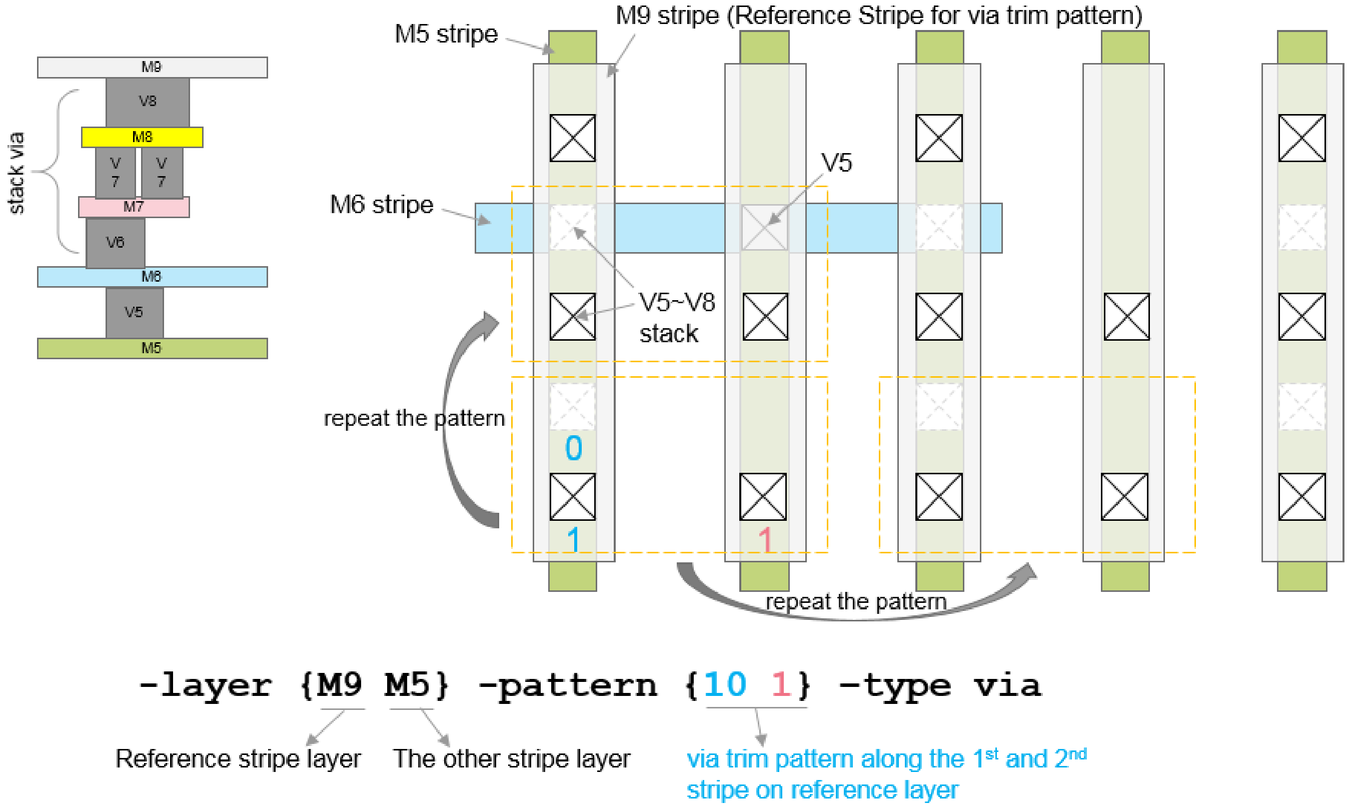
```
-layer {M9 M6} -pattern {1 0 1 1} -type via
```

- Following is an example of trimming stack via between parallel stripes on `M7` and `M9`:


M7 stripe
M9 stripe (Reference Stripe for via trim pattern)
V7~V8 stack via
repeat the pattern
0
1
1
repeat the pattern
Reference stripe layer
The other stripe layer
via trim pattern along the 1st and 2nd stripe on reference layer

```
-layer {M9 M7} -pattern {10 1} -type via
```

- Following is an example of trimming only the stack via that connects the stripes on the two specified layers, which are the candidates for trimming (here, `V5~V8` stack are candidates, the `V5` via that connects the `M5` and `M6` stripes, is not the candidate):



```
-layer {M9 M5} -pattern {10 1} -type via
```

Related Information

Power Planning and Routing

5

Design Implementation Capabilities

Using the Mixed Placer

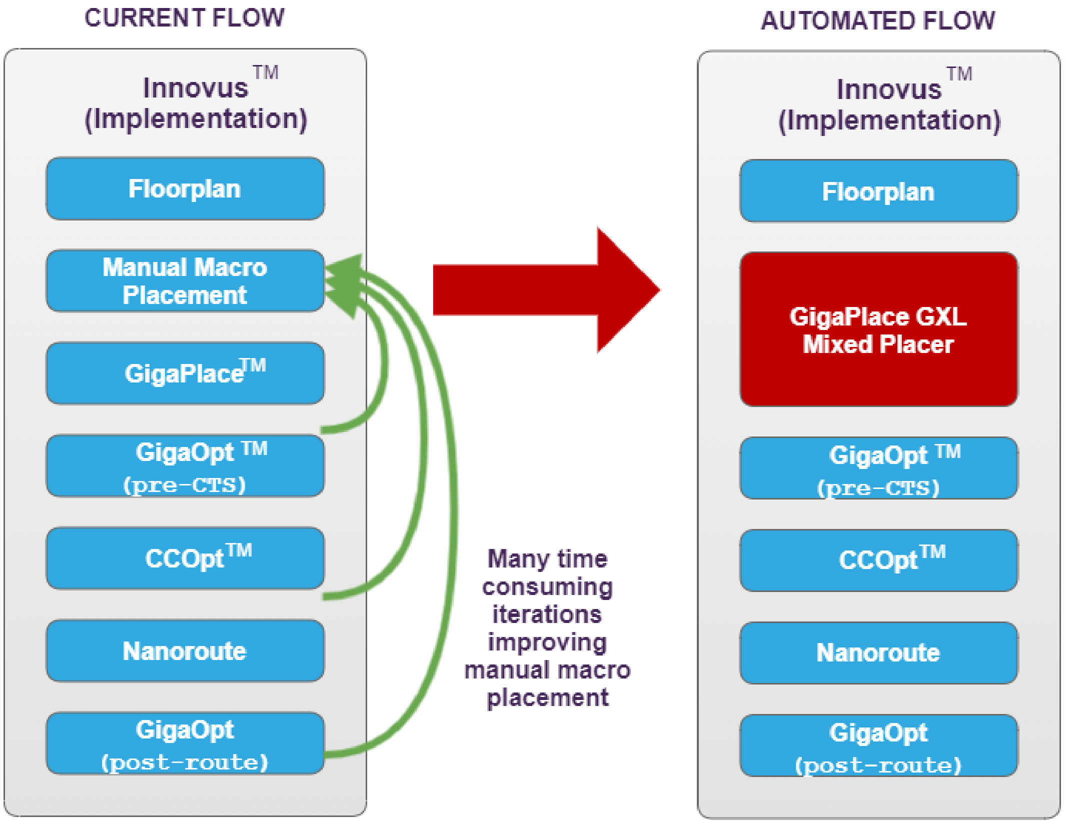
Mixed Placer Overview

In the traditional digital backend implementation flow, creating a floorplan is the first stage and the main target is to place those macros that can get a routable floorplan with minimized wire-length and good timing. It always takes design engineers a lot of effort to achieve these tasks and many design iterations are needed, which are trial-and-error approaches. Especially in advanced technology nodes, a good floorplan is important and critical to ensure the QoR convergence. In manual macro placement, many time-consuming iterations are required.

Increasing design size and complexity results in hundreds of macros that are difficult to place manually within reasonable time. The manual macro placement results in non-optimal placement. The GigaPlace GXL mixed standard cell and macros placement on the other hand ensures optimal location for all cells. In the mixed placement implementation flow, the macros and standard cells are placed concurrently by a powerful engine, which is driven by congestion, wire-length and timing. It reduces a lot of manual work and effort to achieve faster TAT with comparable or better QoR than the traditional flow.

Recommended Mixed Placement Flow

In the mixed placement implementation flow, the floorplanning stage is integrated with the placement stage, while the other stages are the same as the traditional flow. Below is a diagram showing the steps in the mixed placement implementation flow:



License Requirement

To run the mixed placement implementation flow, you require the following license:

- GigaPlace GXL

Using the Mixed Placer Flow

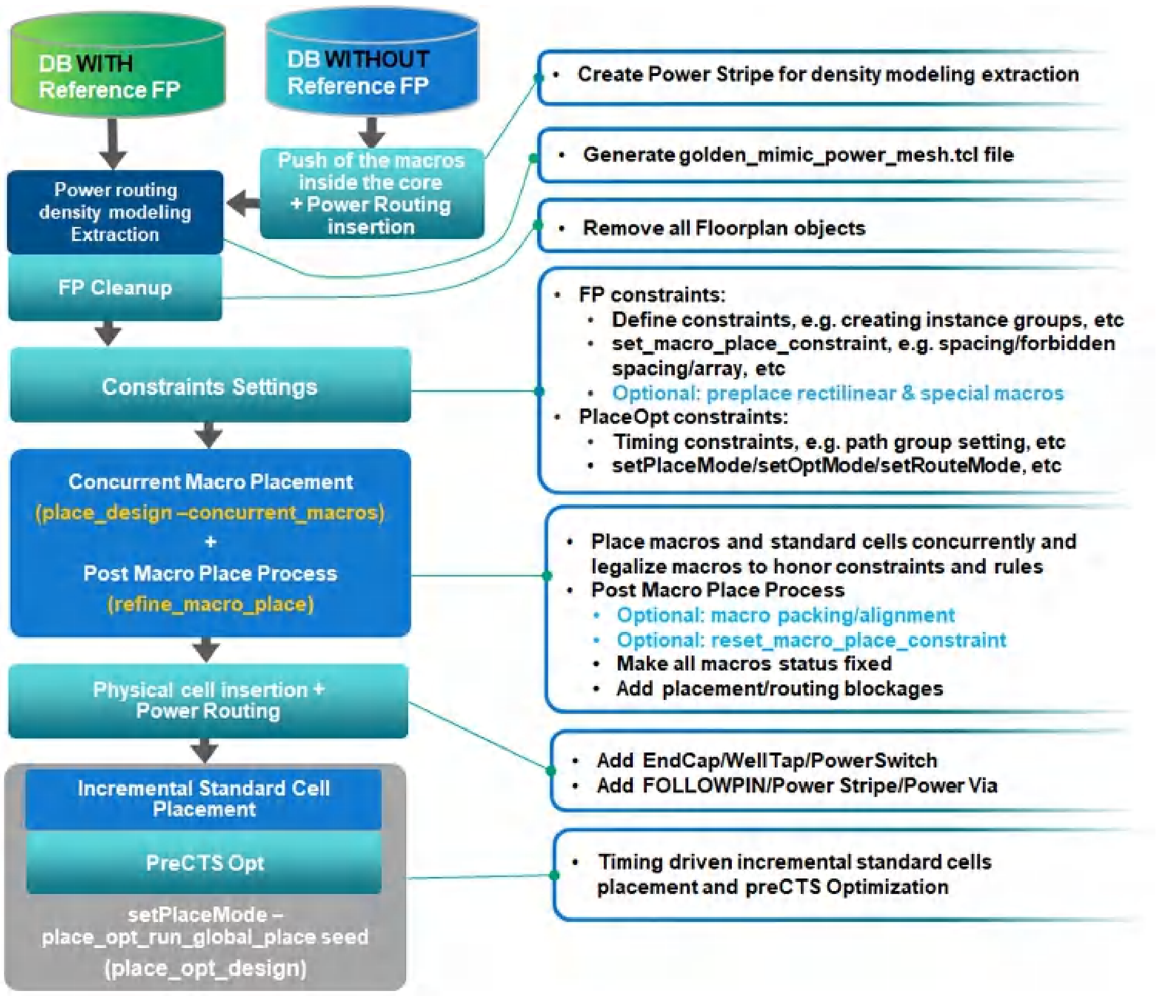
The mixed placement implementation flow enables you to place macros and standard cells concurrently and define constraints. The ideal design candidate for using the concurrent macro placer is a design with a large number of macros, a rectangular design shape and a Macro Area versus Total Area ratio less than 60 percent. The benefit of this flow are:

- Improved TAT for placing macros.
- Wire-length reduction
- Power improvement
- Lower congestion

Note: Large macros have a significant impact on the floorplan feasibility. It is recommended to pre-place them manually (and set a FIXED attribute).

For more information on the supported design styles, best design configuration, and limitations, see Supported Design Styles.

Use Model


DB WITH Reference FP
DB WITHOUT Reference FP
Power routing density modeling Extraction
Push of the macros inside the core + Power Routing insertion
FP Cleanup
Constraints Settings
Concurrent Macro Placement (place_design –concurrent_macros) + Post Macro Place Process (refine_macro_place)
Physical cell insertion + Power Routing
Incremental Standard Cell Placement
PreCTS Opt
setPlaceMode – place_opt_run_global_place seed (place_opt_design)
Create Power Stripe for density modeling extraction
Generate golden_mimic_power_mesh.tcl file
Remove all Floorplan objects
FP constraints:
Define constraints, e.g. creating instance groups, etc
set_macro_place_constraint, e.g. spacing/forbidden spacing/array, etc
Optional: preplace rectilinear & special macros
PlaceOpt constraints:
Timing constraints, e.g. path group setting, etc
setPlaceMode/setOptMode/setRouteMode, etc
Place macros and standard cells concurrently and legalize macros to honor constraints and rules
Post Macro Place Process
Optional: macro packing/alignment
Optional: reset_macro_place_constraint
Make all macros status fixed
Add placement/routing blockages
Add EndCap/WellTap/PowerSwitch
Add FOLLOWPIN/Power Stripe/Power Via
Timing driven incremental standard cells placement and preCTS Optimization

1. For using the mixed placer, you can start with or without a reference floorplan.
   - **Starting with a Reference Floorplan**: Lets you extract the power routing density file before cleaning the floorplan and running the mixed placer. This file is used to generate a better congestion and wire-length modeling while placing the macros.
   - **Starting without a Reference Floorplan**: Requires you to push all the macros inside the core and do power routing insertion by creating power stripes for density modeling extraction.
2. Extract the power routing density modeling information. You can use the `create_pg_model_for_macro_place` command to create PG models for the concurrent macro placement. This command saves PG modeling information to a Tcl file from the existing floorplan.
3. Floorplan Cleanup: Remove all the floorplan objects. Remove all the routing blockages, placement blockages, relative_floorplan constraints, placed instances, routing wires, boundary_constraints, and delete physical instances. Before proceeding with the macro placement, ensure that you generate the power mesh mimic file.
4. Set the constraints for macro placement. You can use the `set_macro_place_constraint` command to specify the constraints for placing the macros and standard cells concurrently. You can use this command to specify:
   - Macro Array Constraints

- Group Constraints
- Spacing Constraints
- Macro Orientation Constraints
- Maximum Stacking Length
- Fixed Macro Location
- I/O Pin Keep-out
- Macro Placement Halo
  The recommended modeling is to increase macro halos to “reserve some space” for the physical cells before the mixed placement and come back to a regular halo distance before the placement.
  For example, the following command adds additional halos to model physical insts before placement:
  `addHaloToBlock -allBlock {2 2 2 2}`
  The following command resets the macro halo to the regular size after `place_design -concurrent_macros`:
  `addHaloToBlock -allBlock {1 1 1 1}`

For more information on mixed placer constraints, see Mixed Place Constraints.

5. Concurrent Macro Placement: Place macros and standard cells concurrently and legalize the macros to honor constraints and rules. You can use the `place_design -concurrent_macros` command for concurrent timing driven placement of macros and standard cells. The `refine_macro_place` command MUST then be used to legalize the macros.
   Note: When `place_design -concurrent_macros` is run, it sets the value of `setPlaceMode -place_opt_run_global_place` to `seed`. With the seed option, the `place_opt_design` command uses seed placement from concurrent macro placement or other sources, runs incremental standard cell placement, then runs preCTS optimization.

   Flow usage is as follows:
   ```
   place_design -concurrent_macros
   saveDesign concurrentMacro.enc
   refine_macro_place

   …
   place_opt_design
   ```

   ⚠ **Note:** When the `-place_global_place_io_pins` parameter of the `setPlaceMode` command is set to true, the `place_design -concurrent_macros` command places the placed and unplaced I/O pins in the mixed placer flow and optimize their locations to reduce wire length and congestion.

   This support is a limited-access feature in this release. It is enabled by a variable specified using 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.

6. Post Macro Placement Process: The `refine_macro_place` command is used to legalize the macros based on constraints such as halo and blockages, forbidden spacing, min-space, and so on.
   Note: The `setPlaceMode -place_design_refine_macro` parameter is disabled by default to have a database saved before using the `refine_macro_place` command in order to see the macro placement just after the mixed placement. This helps in debugging. Having a saved database can also be useful just before calling the `refine_macro_place` command if you want to do several trials with different constraints (by `set_macro_place_constraint`).

   ```
   place_design -concurrent_macros
   saveDesign concurrentMacro.enc
   refine_macro_place
   ```

   When the `setPlaceMode -place_design_refine_macro` parameter is enabled, the `refine_macro_place` command is automatically called after `place_design -concurrent_macro`.

```
setPlaceMode -place_design_refine_macro false
```

Note: If you (do) want the `refine_macro_place` command to be automatically called just after `place_design -concurrent_macro` command, you can enable the `setPlaceMode -place_design_refine_macro` parameter. This however is NOT the recommended flow.

7. Physical Cell Insertion and Power Routing: When all macros are placed in legal locations (after `refine_macro_place`) and fixed in the floorplan, it is time to insert physical cells and create power stripes. For more information, see `addEndCap`, `addWellTap`, `addPowerSwitch`, `addStripe`, `editPowerVia`, etc.

   For example, the following command adds stripes in the vertical direction for the vdd and gnd nets:
   ```
   addStripe -direction vertical -nets {vdd gnd} -width 10 -spacing 1 -layer METAL6 -start_offset 50 -
   set_to_set_distance 50
   ```
8. Incremental Standard Cell Placement: At the end of the mixed placement (`place_design -concurrent_macro`) the standard cells have been placed (global placement) but they are not in legal positions yet. With the `set_place_mode - place_opt_run_global_place` set to `seed` automatically by the mixed placer, the place_opt_design command will start by an incremental placement to find a legal location for all the stdcells before calling the optimization.
9. PreCTS Optimization: The `place_opt_design` command will perform a PreCTS optimization to do more interleaving between placement and congestion. It leads to better timing and congestion QOR because placement is more aware of timing and congestion critical areas. For more information on optimizing the design, refer to the Optimizing Timing chapter in the *Innovus User Guide*.

Note: For large and complex designs containing hundreds of macros, the mixed placement flow cannot guarantee the perfect floorplan in one pass. Sometimes, a few iterations are needed.

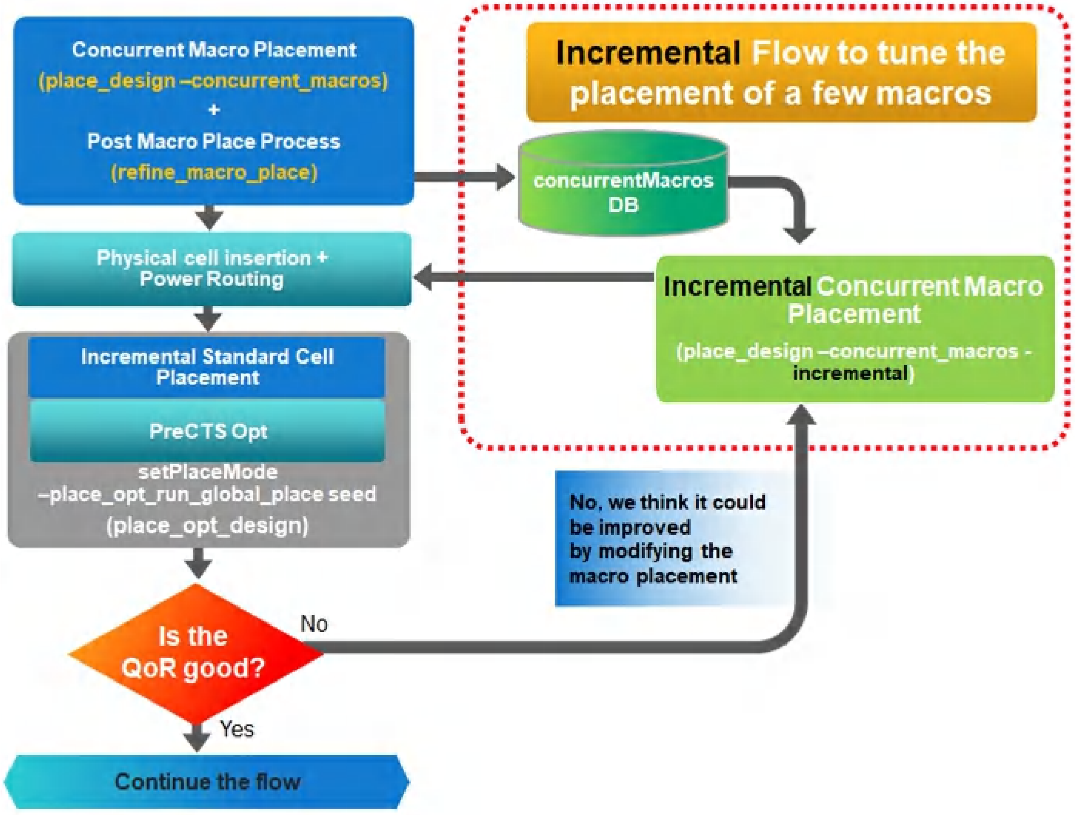
Note: To check if the mixed placement result is good, a first analysis can be done at the end of the `place_opt_design` command. If timing or congestion is bad and it confirms that the root cause is a non-optimal macro placement, the recommendation is to refine the constraints (`set_macro_place_constraint`, `setPlaceMode`, `setRouteMode`, and so on) and restart the flow. If only a few macros must be refined, an incremental flow can be used.

Tuning the Design Using Incremental Flow

If the result from the `place_opt_design` step is good, but you think that it can be bettered by tuning the placement by significant moves (movement of at least the size of the macro) of a few macros, you can use the incremental flow. Starting from the mixed place DB, do the following:

- Unfix ALL macros by using the `dbGet` command or through the Attribute Editor (GUI).
  - ***Why all macros?***
    The macros which have been moved manually can have a bigger impact that we think on the rest of the design. That is why it is recommended to unfix all macros to give more flexibility to the mixed placer to tune the floorplan.
- Update the constraints using the `set_macro_place_constraint` and `setPlaceMode` commands.
- Do incremental concurrent macro placement using the following command:
  ```
  place_design -concurrent_macros -incremental
  ```

This incremental flow can be called several times if needed. Manual placement tuning can help for some corner cases. After this macro placement tuning, you can go back to the “physical cell insertion and power routing” stage of the main flow as described in the following flowchart:


Concurrent Macro Placement
(place_design –concurrent_macros)
+
Post Macro Place Process
(refine_macro_place)
Incremental Flow to tune the placement of a few macros
concurrentMacros DB
Physical cell insertion + Power Routing
Incremental Concurrent Macro Placement
(place_design –concurrent_macros -incremental)
Incremental Standard Cell Placement
PreCTS Opt
setPlaceMode
–place_opt_run_global_place seed
(place_opt_design)
No, we think it could be improved by modifying the macro placement
Is the QoR good?
No
Yes
Continue the flow

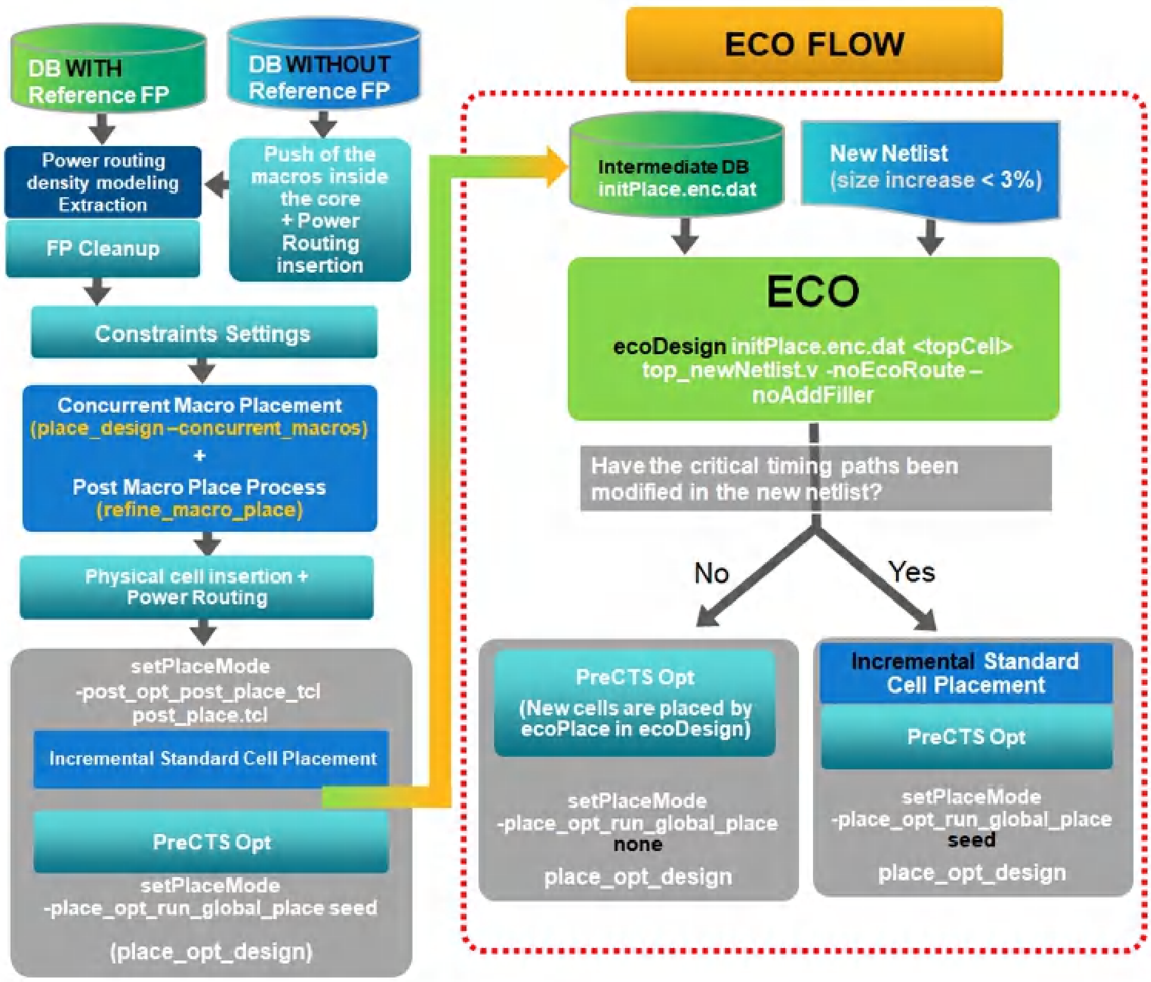
Using the ECO Flow for the New Netlist

The ECO flow can be used for an updated Verilog netlist with small changes and few impact on timing paths. The idea is to keep the floorplan elaborated using the previous netlist so that you do not need to redo the macro placement and the power routing. The DB containing the previous floorplan comes from the intermediate DB saved during the `place_opt_design` command.

Depending of the impact of this new netlist on the critical timing paths, the `-place_opt_run_global_place` parameter of the `setPlaceMode` command must be set differently. The `-place_opt_run_global_place` parameter changes the global placement behavior inside `place_opt_design`.

- `setPlaceMode -place_opt_run_global_place none`
  Skips place_design and runs only preCTS Optimization inside `place_opt_design`. With “none”, if the critical timing paths have not been modified, the new standard cells will be placed directly during ecoPlace.

- `setPlaceMode -place_opt_run_global_place seed`
  Uses seed placement from Concurrent Macro Placement or other sources and runs incremental standard cell placement, then runs preCTS optimization. With “seed”, if critical timing paths have been modified, the new standard cells will be placed incrementally by the `place_opt_design` command.


DB WITH Reference FP
DB WITHOUT Reference FP
Power routing density modeling Extraction
Push of the macros inside the core + Power Routing insertion
FP Cleanup
Constraints Settings
Concurrent Macro Placement
(place_design –concurrent_macros)
+
Post Macro Place Process
(refine_macro_place)
Physical cell insertion + Power Routing
setPlaceMode
-post_opt_post_place_tcl
post_place.tcl
Incremental Standard Cell Placement
PreCTS Opt
setPlaceMode
-place_opt_run_global_place seed
(place_opt_design)
ECO FLOW
Intermediate DB
initPlace.enc.dat
New Netlist
(size increase < 3%)
ECO
ecoDesign initPlace.enc.dat <topCell>
top_newNetlist.v -noEcoRoute –
noAddFiller
Have the critical timing paths been modified in the new netlist?
No
Yes
PreCTS Opt
(New cells are placed by ecoPlace in ecoDesign)
setPlaceMode
-place_opt_run_global_place
none
place_opt_design
Incremental Standard Cell Placement
PreCTS Opt
setPlaceMode
-place_opt_run_global_place
seed
place_opt_design

If the new netlist size is increased significantly or impacts a lot of critical paths, the recommendation is to rerun the full mixed placement flow or run the CPG flow.

Cadence Placement Guidance (CPG) Flow

Generally, the more the netlist changes, the more different the mixed placement result can be. For very small changes the ECO flow that does not touch the floorplan is a good approach to consider. For significant netlist changes if you want to get a similar floorplan as with the previous netlist, the CPG flow is a potential solution.

The idea of this flow is to keep the macro placement of the previous floorplan (through a guide file) as much as possible but let the tool move the macros, if necessary, to solve any new congestion or timing issue. A typical usage of the CPG flow is to support the insertion of DFT modules inside the netlist, assuming the DFT has been elaborated considering the macro placement of the pre-DFT floorplan. If the netlist change is due to new critical modules or functions, the CPG flow is not recommended. A new mixed placement flow from scratch is expected to provide better results.

To run the CPG flow, you must save a guide file from the old version DB and read it during the mixed placement flow on the new version netlist. Take the pre-DFT and post-DFT netlists for exampl. You have already run a mixed placement flow on the pre-DFT netlist and pushed the QoR to be converged. When receiving the post-DFT netlist, you can run the CPG flow as follows:

Innovus Session 1

1. Restore the placement DB for pre-DFT.

2. Use the `write_macro_place_constraint` command to write the constraints into a Tcl format file from the database:

   ```
   write_macro_place_constraint -sections {cpg} -cpg_scope {macro_only} -out_file macroLocs.tcl
   ```

   Example of what you can see inside this file:

   ```
   set_macro_place_constraint -cpg {instName x y orientation}
   ```

Innovus Session 2

1. Restore the initial DB for post-DFT.
2. Specify the CPG guide file using the `-place_global_cpg_file` parameter. When this option is specified, the CPG flow is turned ON automatically.

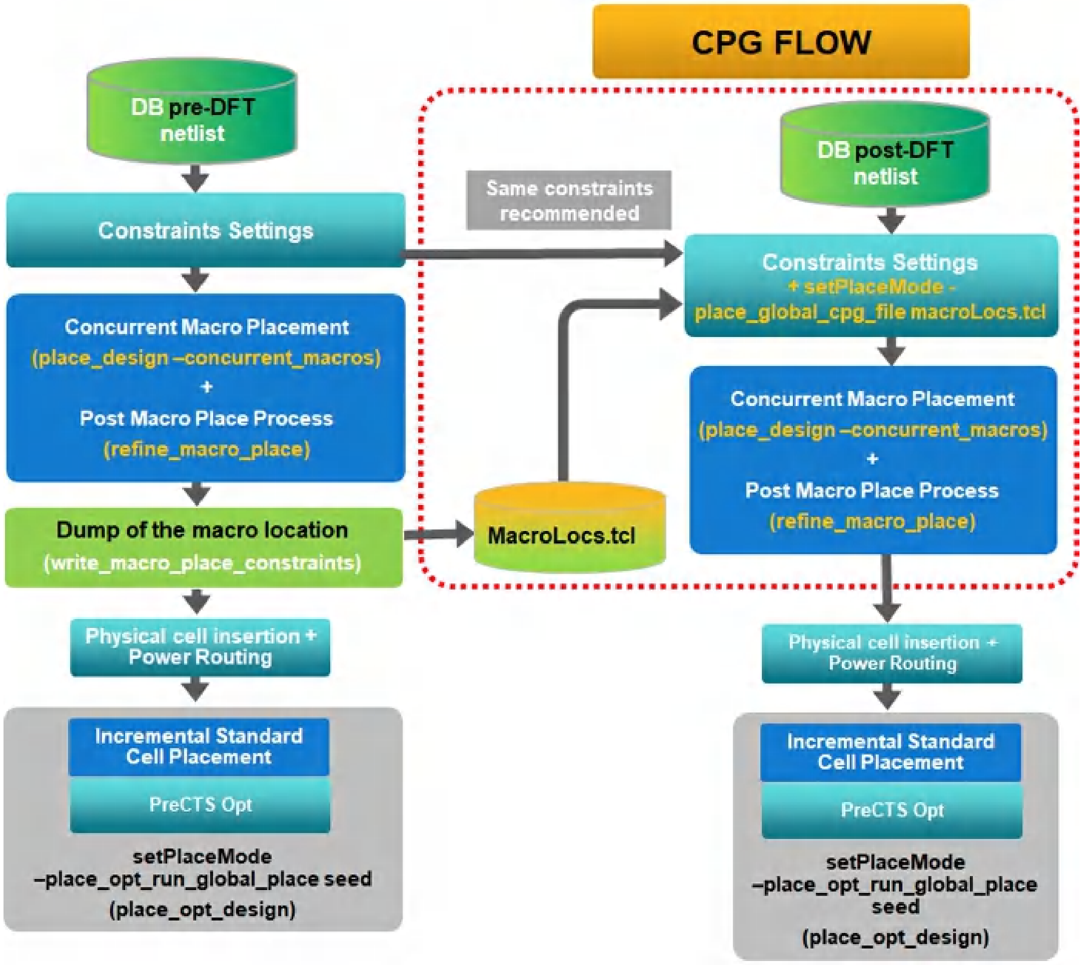
   ```
   setPlaceMode -place_global_cpg_file macroLocs.tcl
   ```

3. Do concurrent macro placement using the following command:

   ```
   place_design -concurrent_macros
   ```

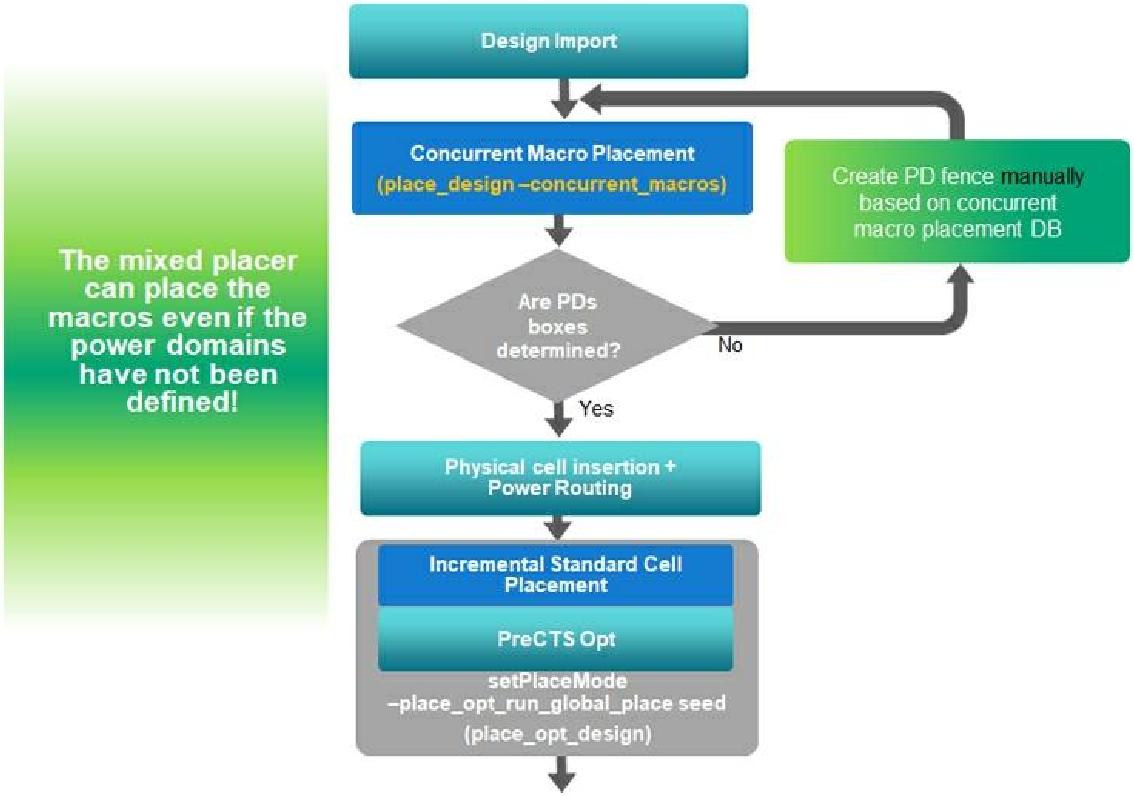
The other settings are the same as the default mixed placement flow.

Note: The `refine_macro_place` command does not honor the CPG guide. It is recommended to keep the same setting of macro placement constraints as the old version DB when running the CPG flow.


CPG FLOW
DB pre-DFT netlist
Constraints Settings
Concurrent Macro Placement
(place_design –concurrent_macros)
+
Post Macro Place Process
(refine_macro_place)
Dump of the macro location
(write_macro_place_constraints)
Physical cell insertion + Power Routing
Incremental Standard Cell Placement
PreCTS Opt
setPlaceMode
–place_opt_run_global_place seed
(place_opt_design)
Same constraints recommended
MacroLocs.tcl
DB post-DFT netlist
Constraints Settings
+ setPlaceMode -
place_global_cpg_file macroLocs.tcl
Concurrent Macro Placement
(place_design –concurrent_macros)
+
Post Macro Place Process
(refine_macro_place)
Physical cell insertion + Power Routing
Incremental Standard Cell Placement
PreCTS Opt
setPlaceMode
–place_opt_run_global_place seed
(place_opt_design)

Multiple Supply Voltage flow (MSV)

The Concurrent Macro Placement capability supports designs with MSV. If the power domain fences are already defined, each macro will be placed respectively in its power domain fence as expected. If the power domain fences are not defined, the mixed placement flow can be used so it can help you define them. The placement of the macros will give the first idea where each power domain should be placed inside the floorplan. From this placement, you can then draw the power domain fences manually. For more information, refer to Floorplanning the Design chapter of Innovus User Guide.
The power switching cells are inserted in the flow simultaneously with physical cells and power stripes. Isolation cells are placed by the incremental standard cell placement. Once the power domain fences are defined, the user can rerun the mixed placement to place each macro in their power domain fences.


Design Import
Concurrent Macro Placement
(place_design –concurrent_macros)
Create PD fence manually based on concurrent macro placement DB
The mixed placer can place the macros even if the power domains have not been defined!
Are PDs boxes determined?
No
Yes
Physical cell insertion + Power Routing
Incremental Standard Cell Placement
PreCTS Opt
setPlaceMode –place_opt_run_global_place seed
(place_opt_design)

Supported Design Styles

The Concurrent Macro Placement capability supports the following design styles:

- Flat designs
- MSV (Multiple Supply Voltage) designs with fences defined
  For more information, see Multiple Supply Voltage flow (MSV)
- ILM designs
  The Concurrent Macro Placement capability supports ILM designs, but has the following limitations:
  - It cannot move ILM blocks.
  - If there are too many fixed ILM blocks, the flow will have trouble to legalize macros and the placement result is compromised.

Note: The Concurrent Macro Placement capability does not support the hierarchical flow.

Best Design Configuration

The design style can also be different depending on many other parameters like the boundary shape, the macros shape and size, the density, the number of layers, and others. Following are the best design configurations supported:

- Designs with rectangular boundary
- Designs that do not have large sized macros.
- Designs with almost uniform sized macros.
- Designs where the macro density (macro area / design area) is less than 60 percent
- Designs with enough routing resources over macros (at least two routing layers for H and V)

The Concurrent Macro Placement capability may be ideal for GPU, networking, smart phones, AI chips.

Design Limitations

- **Macro Density**
  The QoR may be degraded when the density becomes larger than 60 percent. In this case, some iterations may be necessary to improve it. When macro density is more than 80%, it is quite complicated to get a good QoR. When running the mixed placement flow, macro density is printed in the log file for information.
  Example: Average macro density = 0.25 (means a macro density of 25 percent)
- **Large Sized Macros**
  Large macros have a significant impact on the floorplan feasibility. If the design has a few large macros, it is recommended to preplace the large macros manually. The macros that are considered large sized, have:
  - Area larger than 20 percent of the total design area.
  - Width or height longer than 50 percent of the design's width or height.
- **Limited Routing Resources over Macros**
  This flow may place the macros in the middle of the design like stand cells to get an optimal QoR. With this placement, the wire length is optimized as much as possible and is usually much better compared to the placement with the macro placement close to the design boundary. To handle long nets more easily, which could cross partially, the design would need to have nets routed over the macros. It is preferred to have at least two free routing layers per direction over the macros. If two free routing layers are not available, you need to push the macros to the design boundary after completing the mixed placement flow.
- **Rectilinear Shaped Designs**
  It can support different shapes of designs such as rectangle and simple L shape, without long 90 degrees outside corner edges, and so on. But it is difficult to support a complex shape of the design with more than six inner corners, for example, a U shape design with very long 90 degrees outside corner edges, where QoR may not be optimal.
- **Rectilinear Shaped Macros**
  Rectilinear macros are not fully supported. It is recommended to preplace them if there are only a few such macros.

Mixed Place Constraints

You can use the `set_macro_place_constraint` command to specify the following constraints for placing the macros and standard cells concurrently.

- Macro Array Constraints

- Group Constraints
- Spacing Constraints
- Macro Orientation Constraints
- Maximum Stacking Length
- Fixed Macro Location
- I/O Pin Keep-out
- Macro Placement Halo

Macro Array Constraints

Macro array constraints can be used while placing macros in an array (or matrix) and define the relative placement of each macro. The whole array, treated like one macro, can be flipped or mirrored (R0, MX, MY, R180). You can use the `-array` parameter of the `set_macro_place_constraint` command to specify the name of the macro array followed by the *`array_elements`* containing the instance members information of a row or column of the array in the following format:

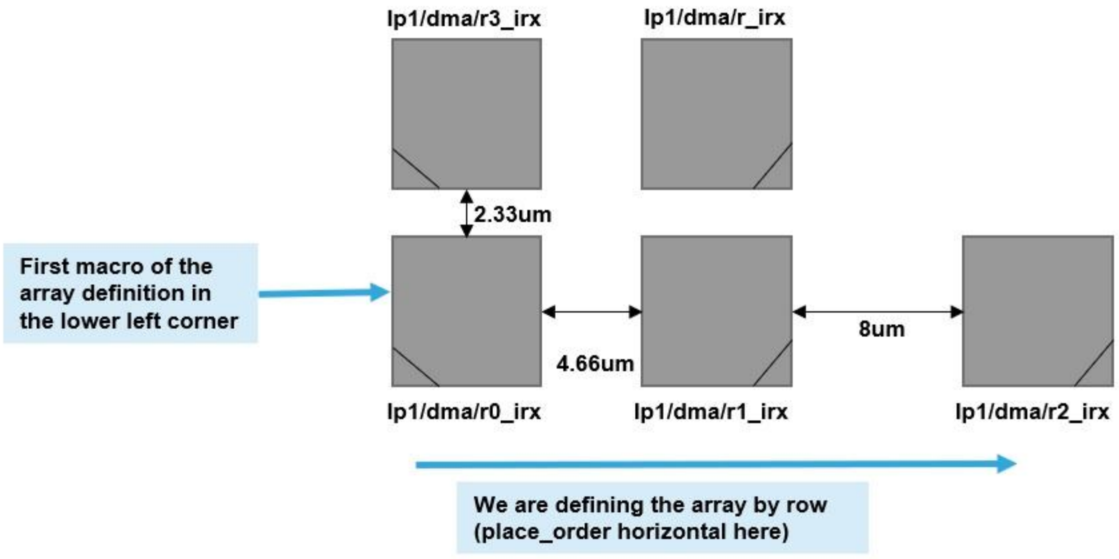
```
inst_name:orientation [spacing inst_name:orientation ] +
```

Note: The *`array_elements`* should be specified with the `-array` parameter.

Example:

The following command creates a macro array, `ram_array`, and defines it constraints.

```
set_macro_place_constraint -array ram_array {{ip1/dma/r0_irx:R0 4.66 ip1/dma/r1_irx:MY 8.0 ip1/dma/r2_irx:MY} 2.33
{ip1/dma/r3_irx:R0 4.66 ip1/dma/r_irx:MY}} -valid_group_orientations {R0 MY}
```


Ip1/dma/r3_irx
Ip1/dma/r_irx
2.33um
First macro of the array definition in the lower left corner
4.66um
8um
Ip1/dma/r0_irx
Ip1/dma/r1_irx
Ip1/dma/r2_irx
We are defining the array by row (place_order horizontal here)

Group Constraints

The group constraints are used to specify:

- The macros that should be placed close to each other as part of a macro group (so without specifying relative placement)
- The soft and hard constraints for macro groups

- The macros that are to be aligned as a group

For defining the group constraints, use the following:

- Guide/Region/Fence: These three constraints are same as the traditional flow. Fixed bounding box is needed by user input.
- Soft Constraints: The macros inside a group will be placed closely. No bounding box is defined.
  For example, the following commands create a new instance group, pipe1 as a soft guide constraint and adds macros to this group.

  ```
  createInstGroup pipe1 -softGuide
  addInstToInstGroup pipe1 {ip1/regbank1 ip1/dma/ramI2 ip1/alu/adderI3}
  ```
- Align Group: Specifies the macros that are to be aligned as a group. The group elements should only be macros with the same size. The alignment will be automatically done by the tool with soft constraint.
  For example, the following command specifies that the macros of group1 should be aligned as a group.

  ```
  set_macro_place_constraint -align_group group1 {ip2/dma/ram2A ip2/dma/ram2B}
  ```

Spacing Constraints

The spacing constraints are used to specify the minimum spacing and forbidden spacing of macro-to-macro and macro-to-core-boundaries. These spacing constraints are honored during "refine_macro_place".

Macro-to-core-boundaries

You can use the following parameters of the `set_macro_place_constraint` command to specify the minimum spacing and forbidden spacing of macro-to-core-boundaries:

- `-forbidden_space_to_core` *`value`*

  Specifies the value for the forbidden spacing between the macro to core boundary. If the space between the macro and core is smaller than this value, then the macro is abutted to the core. The minimum spacing value can be 0 and the maximum can be 100000. For example, if the specified value is 2 then any macro-to-core spacing between 0 to 2 is reduced to 0.

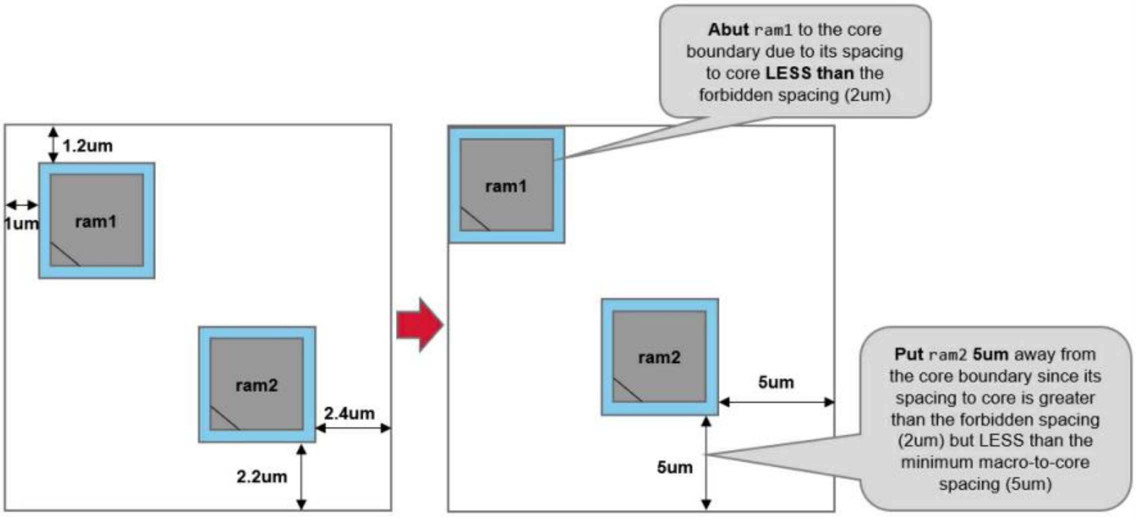
  Note: In case the `-min_space_to_core` parameter has been specified, then the value specified for the `-forbidden_space_to_core` parameter should be less than the value specified for the `-min_space_to_core` parameter.
- `-min_space_to_core` *`value`*

  Specifies the value for the minimum spacing between macro and core boundary. If the space between the macro and core is smaller than this value and larger than the forbidden space, then the macros are pushed away from the core to the minimum space. The minimum spacing value can be 0 and the maximum can be 100000.

Example:

The following command specifies the minimum space between macro-to-core boundary as 5 and the forbidden space between macro-to-core boundary as 2.

```
set_macro_place_constraint -forbidden_space_to_core 2 -min_space_to_core 5
refine_macro_place
```


Abut ram1 to the core boundary due to its spacing to core LESS than the forbidden spacing (2um)
1.2um
1um
ram1
ram2
2.4um
2.2um
ram1
ram2
5um
5um
Put ram2 5um away from the core boundary since its spacing to core is greater than the forbidden spacing (2um) but LESS than the minimum macro-to-core spacing (5um)

Macro-to-macro-boundaries

You can use the following parameters of the `set_macro_place_constraint` command to specify the minimum spacing and forbidden spacing of macro-to-macro-boundaries:

- `-forbidden_space_to_macro` *`value`*

  Specifies the value for the forbidden spacing between the macro-to-macro boundary. If the space between the macros is smaller than this value, then the macros are abutted. The minimum spacing value can be 0 and the maximum can be 100000. For example, if the specified value is 2 then any macro-to-macro spacing between 0 to 2 is reduced to 0.

  Note: In case the `-min_space_to_macro` parameter has been specified, then the value specified for the `-forbidden_space_to_macro` parameter should be less than the value specified for the `-min_space_to_macro` parameter.

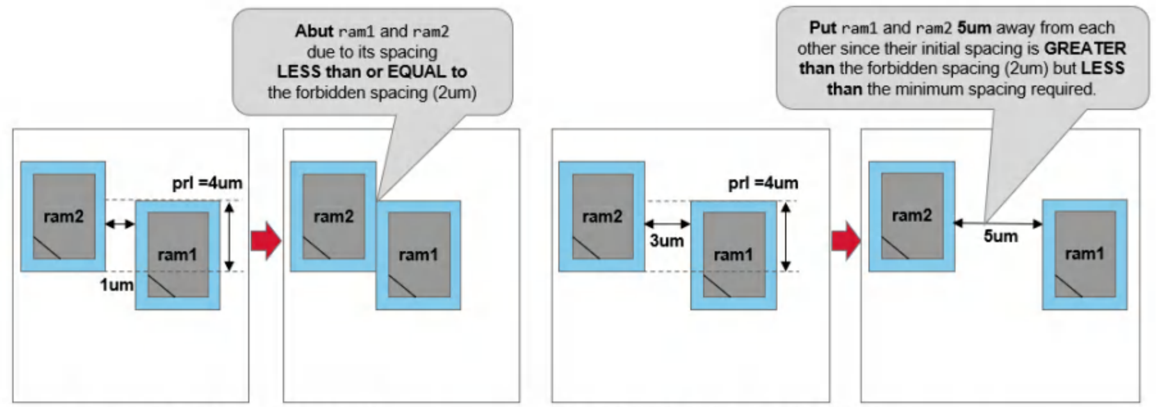
- `-min_space_to_macro` *`value`*

  Specifies the value for the minimum spacing between the macro-to-macro boundary. If the space between the macros is less than this value, then the macros are aligned. The minimum spacing value can be 0 and the maximum can be 100000.

Example:

The following command specifies the minimum space between the macro-to-core boundary as 5 and the forbidden space between macro-to-core boundary as 2.

```
set_macro_place_constraint -parallel_run_length 3 -forbidden_space_to_macro 2 -min_space_to_macro 5
refine_macro_place
```


Abut ram1 and ram2 due to its spacing **LESS than or EQUAL to** the forbidden spacing (2um)
Put ram1 and ram2 **5um** away from each other since their initial spacing is **GREATER than** the forbidden spacing (2um) but **LESS than** the minimum spacing required.
ram2
ram1
prl =4um
1um
ram2
ram1
ram2
ram1
prl =4um
3um
ram2
ram1
5um

Prerequisite: Parallel_run_length (prl) >3um
If spacing_to_macro <= forbidden_space_to_macro,
then the macros are aligned to each other

Prerequisite: Parallel_run_length (prl) >3um
If forbidden_space_to_macro < spacing_to_macro <= min_space_to_macro,
then put the macros 5um away from each other.

Notes:

- Macro-to-macro space checking is enabled only when the real parallel run length of macro is bigger or equal to the user specified "prl" setting (default : 0)
- By default, if at least one edge of the 2 macro edges facing each other has pins, the macros are not abutted (the parameter -avoid_abut_macro_edges_with_pins = true by default).

Macro Orientation Constraints

The macro orientation constraints are used to specify the legal orientations for macro instances. It is a local restriction to refine the solution and is stricter than LEF symmetry definition. You can use the `-orientation` parameter of the `set_macro_place_constraint` command to specify the allowed orientations (R0, MX, MY, R180, R90, MX90, MY90, R270) for the macros. The specified orientations should be legal in LEF.

Note: The specified orientation overwrites the previous definition.

Note: This constraint is instance-based and not cell-based.

Example:

The following command specifies the macros and their orientations.

```
set_macro_place_constraint -insts {ip2/dma/ram2A ip2/dma/ram2B} -orientation {R0 MX}
```

Maximum Stacking Length

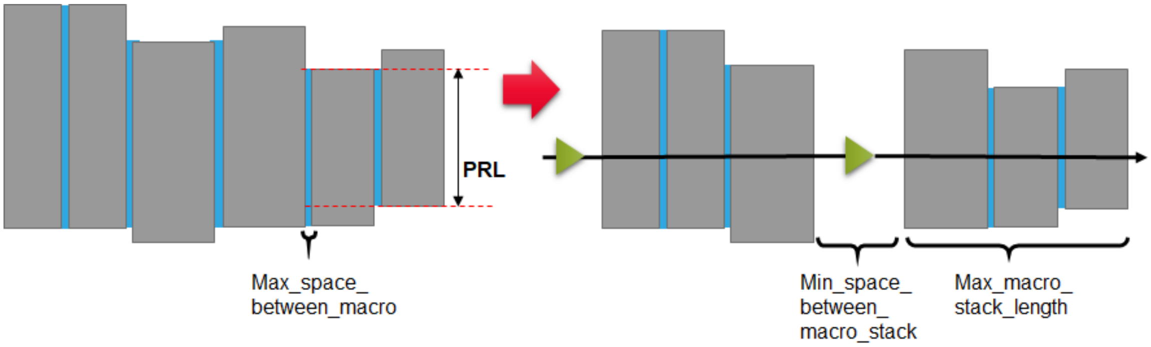
The purpose here is to split a big macro stack into smaller stacks of macros in order allow buffering or for IR drop purposes. When long nets travel across the stacking macros, the optimizer cannot add buffers to fix DRV or timing violations if there are no channels between them. You can use the `-horizontal_stacking` parameter to specify the maximum horizontal stack depth and the minimum horizontal spacing between stacks.

Example:

```
set_macro_place_constraint -horizontal_stacking {max_macro_stack_length min_space_between_macro_stack
max_space_between_macros} -parallel_run_length {prl}
```

- *`max_macro_stack_length`:* Specifies that the macro stacking length should not be more than the specified value.
- *`min_space_between_macro_stack`:* Specifies the spacing between different macro stacks.
- *`max_space_between_macro`: This parameter is an input* which specifies the maximum space between macros to consider these macros in the same initial stack. When the macro spacing is less than the specified value, the macros are counted in the

stacking length. For example, if the max_space_between_macro value is 9 then the macros will be considered as stacked if the space between the macros is between 0 and 9, however, if the real space is 10 then nothing will happened.



Note: refine_macro_place starts by spacing the macros considering the forbidden/min_space_to_macro. At the end of this first spacing, if most of the space between macros are equal to the `min_space_to_macro`, the use of the -horizontal/vertical_stacking option will be effective only if the `max_space_between_macro` parameter is larger than the -min_space_to_macro parameter.

Note: Specifying the parallel_run_length is mandatory for checking the stacking constraint. The macros are not considered to be stacked when the PRL is less than or equal to 0, since there is no channel-less problem. When the `-parallel_run_length` parameter is specified along with the `-horizontal_stacking` parameters, only the macros with the parallel run length greater than and equal to the specified value will be counted in the stacking length.

Fixed Macro Location

If you want to fix the macro location and not allow the placer to move it, the solution is to preplace the instance and change the status to fixed. You can use the `placeInstance` command to place an instance and use the `setInstancePlacementStatus` command to change its placement attributes status to fixed. The fixed component has a location and cannot be moved by automatic tools but can be moved using interactive commands.

Example:

The following command places the instance my/ip1/rom_512i2 with mirrored through Y axis orientation at location 102.33, 2022.39 in the floorplan. The placement status of instance my/ip1/rom_512i2 is set to the *fixed* status and the macro placer will not move this instance later:

```
placeInstance my/ip1/rom_512i2 102.33 2022.39 MY
setInstancePlacementStatus -name my/ip1/rom_512i2 -status fixed
```

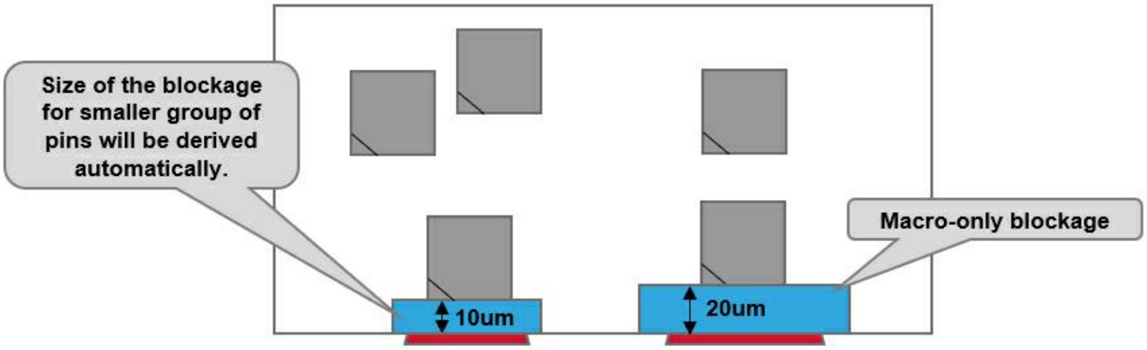
I/O Pin Keep-out

This constraint is used to prevent macros from being placed near I/O pins. You can use the `-max_io_pin_group_keep_out` parameter of the `set_macro_place_constraint` command to specify the maximum size of the macro-only blockages which will be placed automatically in front of the most important group of IO pins (number and density are considered) and reserve the keep-out space for pin routing. The size of the macro-only blockage for the other groups of IOs is calculated automatically.

Example:

The following command specifies the maximum height or width of 20um for adding macro-only blockages to IO pins.

```
set_macro_place_constraint -max_io_pin_group_keep_out 20
```


Size of the blockage for smaller group of pins will be derived automatically.
Macro-only blockage
10um
20um

Macro Placement Halo

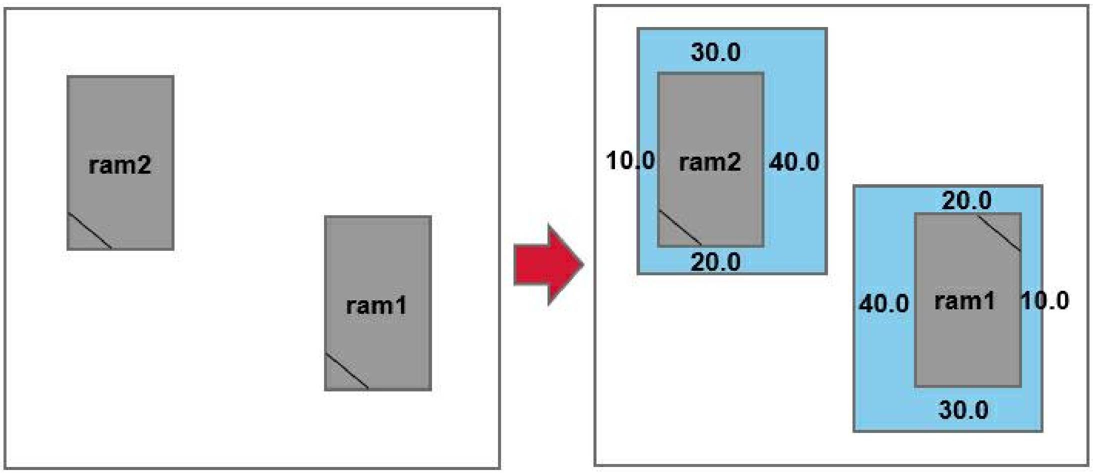
A halo is an area that prevents the placement of blocks and standard cells within the specified halo distance, measured from the edges of a hard macro to reduce congestion. This constraint is used to keep the instance-based macro spacing and model EndCap, WellTap, and PSW cell areas at the mixed placement stage.

Adds a halo to a block. A halo is an area that prevents the placement of blocks and standard cells within the specified halo distance from the edges of a hard macro, black box, or committed partition to reduce congestion. A block halo value is specified based on the current block orientation.

Example

The following command adds a halo around cells ram1 and ram2 in a R0 orientation that is 10.0µm from the left edge, 20.0µm from the right edge, and 40.0µm from the bottom and and 3.0µm from the top edge:

```
addHaloToBlock {10.0 20.0 40.0 30.0} -cell {ram1 ram2} -ori R0
```



Mixed Place Constraints Support List

The following table lists the mixed place constraints supported by the `place_design`, `refine_macro_place`, and `check_macro_place_constraint` commands.

			place_design -concurrent_macro	refine_macro_place	check_macro_place_constraint
1	Macro Array Constraints		Yes	Yes	No
2	Group Constraints	align_group	Yes	No	No
		softGuide	Yes	No	No
		Guide/Region/Fence	Yes	Yes	Check Region/Fence
3	Spacing Constraints		No	Yes	Yes
4	Macro Orientation Constraints		Yes	No	Yes
5	Over Macro Power Routing Modeling		Yes	No	No
6	Maximum Stacking Length		No	Yes	Yes
7	Fixed Location		Yes	Yes	No
8	I/O Pin Keep-out		Yes	Yes	No
9	Macro Placement Halo		Yes	Yes	Check overlap

Low Power Design

Overview

This chapter describes how the multiple supply voltage (MSV) feature can help you save power in your design.

There are two types of MSV designs:

- Multiple Supply Single Voltage (MSSV)
  Core logic runs at a single voltage, but some portions of the logic are isolated on their own power supply.
- Multiple Supply Multiple Voltage (MSMV)
  Supplies of different voltages are used for core logic.

A power domain (also known as voltage island) is a floorplan object in the Innovus™ Implementation System (Innovus) software. A non default and non virtual power domain has a fence constraintant physically; each power domain has a specific library (`.lib`, `.lef`) associated with it. Standard cell Instances that belong to a power domain can be placed only within that power domain. The exception to this rule is Macros, IP blocks and IOs. By constraining the design this way, a complete place and route flow can be used on an MSV design. You can automatically place level shifters, perform timing optimization, run clock tree synthesis (CTS) across domain boundaries, and obtain DRC-clean power routing.

Power Domain Shutdown and Scaling

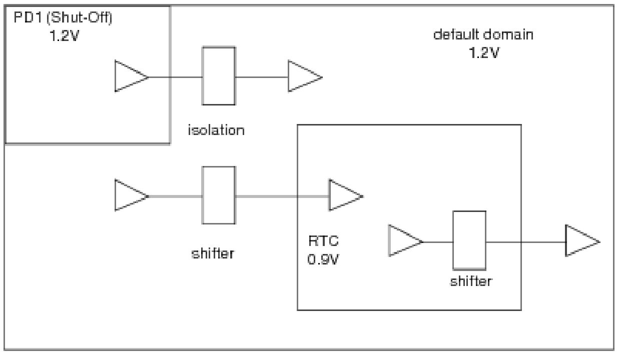
You can reduce power consumption either by shutting down a power domain or operating it at a reduced voltage (voltage scaling).

Power domain shutdown is a technique in which an entire power domain is shut down during a specific mode of operation. This results in both leakage power and dynamic power savings because the transistors are isolated from the supply and ground lines. You must use isolation cells when shutting down domains in order to drive the interface signals to predetermined known states. In many cases, a design in the shutdown mode operates at a single voltage throughout the design (an MSV 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 the on the threshold characteristics of the library for the scaled domain.

Note: These techniques can be used separately or together in the same design.

The following figure shows three power domains: RTC, PD1, and the default power domain, which contains PD1 and RTC.



- PD1 and default domains can share libraries since PD1 and default domains operate at the same voltage.
- Power switches enable PD1 to shut down.
- Power domain RTC operates at a different voltage than PD1 and the default domain.
- RTC can remain always on.

- You must insert voltage level shifters between the default domain and RTC, and between PD1 and RTC.
- Isolation cells (clamps) drive outputs of a power domain to known states when that power domain is shut down.

Support for the Common Power Format (CPF)

Cadence provides a Common Power Format (CPF) that enables you to freely exchange data between Cadence tools supporting the low-power design flows, and most importantly, capture low power design intent early in the design process rather than late in the back-end cycle. A CPF file captures all design and technology-related power constraints, which can be used throughout the design flow. Users need to create CPF file for their power constraint and read it in to Innovus for the low power design.

CPF commands perform functions such as the following:

- Creating power domains and specifying their power/ground connections
- Specifying timing libraries (Optional; users can define timing libraries in Innovus viewDefinition.tcl)
- Creating analysis view and defining library set for each power domain (Optional: users can define them in Innovus viewDefinition.tcl)
- Defining operating conditions (Optional; users can define them in Innovus viewDefinition.tcl)
- Defining low power cells
- Creating low power rules: isolation rules, level shifter rules, SRPG rules, power switch rules)

Note: If there is a minor CPF change during the flow, you can either perform CPF ECO (requiring CLP license) or a view-related update during the flow, without running the flow from the beginning.

CPF Version Support

The Innovus software supports the following versions of CPF:

- CPF 1.0
- CPF 1.0e
- CPF 1.1 (default)
- CPF 2.0

Innovus Commands Supporting CPF

- `read_power_intent -cpf`
- `commit_power_intent`
- `write_power_intent -cpf`

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:

- `read_power_intent -cpf`

  Reads a CPF file into Innovus for error checking

- `commit_power_intent`

  Executes (commits) the CPF commands within the Innovus environment

Loading the Design (init_design)

The design data is read through the `init_design` procedure. In case of CPF, `init_design` calls a special `read_power_intent -cpf` to generate the view definition file. The `init_design` then loads the libraries and sets up MMMC based on the *viewDefinition.tcl*. Since there is no minimum/maximum analysis, the CPF creates the analysis view. Also, since there is no analysis view in CPF, does not generate the script *viewdefinition.tcl* and `init_design` gives an error.

The command `init_design` works as follows:

- If there is a *viewDefinition.tcl* for Low Power design, `init_design` does not call special `read_power_intent -cpf` to re-generate *viewDefinition.tcl*.
- If there is no *viewDefinition.tcl* for Low Power design, `init_design` calls a special `read_power_intent -cpf` to generate *viewDefinition.tcl* based on the CPF.
- If both *veiwDefinition.tcl* and CPF are provided and *viewDefinition.tcl* has higher priority, `init_design` does not call the special `read_power_intent -cpf`.
- If there is no *viewDefinition.tcl* and CPF does not define analysis view, `init_design` gives an error.

For design portability, you can set the Library Primary Path in tcl. For example,

```
set libDir "libs/";
viewDefinition.tcl will refer to the tcl variable (libDir)
```

For all the library specifications like: `create_library_set -library $libDir/...`

If there is any network change, you can re-define the Library Primary Path and do not need to change anything else. To do this, the special `read_power_intent -cpf` generates the *viewDefinition.tcl*.

Note: Since 11.1 release of the software supports only MMMC, Low Power supports only the MMMC Flow.

- If design does not have *viewdefinition.tcl*, CPF should contain `create_analysis_view`.
- If design has *viewDefinition.tcl*, CPF does not require timing library information. The CPF is library-less, and only specifies the Power intent. All the timing information is specified in *viewDefinition.tcl*. **This setting is recommended from 14.1 and above releases of the software**.
- If CPF does not have and design does not have *viewdefinition.tcl*, `commit_power_intent` gives an error.

CPF Documentation

For more information about CPF, see the following documents:

- *Innovus Text Command Reference*

  Documents the following Innovus commands supporting the CPF flow. Descriptions on obsolete native Innovus commands are provided.

- *Menu Reference*

  Documents the `read_power_intent -cpf`/`write_power_intent -cpf` GUI.

- *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 1.0e commands in the Supported CPF 1.0e Commands chapter
- Provides a list of supported CPF 1.1 commands in the Supported CPF 1.1 Commands chapter
- Provides a sample CPF 1.0 script in the CPF 1.0 Script Example chapter
- Provides a sample CPF 1.0e script in the CPF 1.0e Script Example chapter.
- Provides a sample CPF 1.1 script in the CPF 1.1 Script Example chapter.

Support for IEEE1801

The IEEE1801 standard is supported from 13.2 and above releases of the software to specify the design's power intent.

Low Power Cell Definition

All the LP cells and related power pin information need to be defined in the Liberty file with the Liberty LP attributes.

Timing Information

For defining the timing information in the *Innovus MMMC viewDefinition.tcl* file and specify MMMC timing information, use the following command:

```
set init_mmmc_file viewDefinition.tcl
```

Note: The power domain library binding must be specified through the `update_delay_corner` command for each power domain before defining the timing information.

Load the Design for IEEE1801 Using the init_design Command

IEEE1801 specifies only the power intent like library-less CPF. All the timing information is specified in the *viewDefinition.tcl* file.

Innovus IEEE1801 Low Power Flow

Innovus IEEE1801 low power flow is similar to the Innovus CPF low power flow. All the Innovus implementation steps such as floorplan, placement, optimization, clock tree synthesis and routing have been enhanced to handle IEEE1801 so that the IEEE1801 low power flow can run smoothly. The IEEE1801 flow looks like:

`init_design` #use “set init_mmmc_file viewDefinition.tcl” to specify mmmc setting

`read_power_intent -1801` *`IEEE1801File`*

`commit_power_intent`

# The rest of IEEE1801 low power flow (same as CPF low power flow).

Innovus IEEE1801 Command Set Support

Innovus mainly supports a selected set of IEEE1801 2.0 commands and options. For the compatibility purpose of users, Innovus also supports some IEEE1801 1.0 and IEEE1801 2.1 commands and options.

Currently, the following IEEE commands/options are supported.

IEEE1801 Commands/Options	IEEE Version

`add_port_state port_name`
`{-state {name <nom | min max | min nom max | off>}}*`
	1.0

`add_power_state object_name`
`{-state state_name {`
`[-supply_expr {boolean_function}]`
`[-logic_expr {boolean_function}]`
`[-update]`
	2.0

`apply_power_model power_model_name`
`[-elements instance_list]`
`[-supply_map {{lower_scope_handle upper_scope_supply_set}}]`
	2.1

`add_pst_state state_name`
`-pst table_name`
`-state supply_states`
	1.0

`associate_supply_set supply_set_ref`
`-handle supply_set_handle`
	2.0

`begin_power_model power_model_name`
`[-for model_list]`
	2.1

`connect_logic_net net_name`
`-ports port_list`
	2.0

`connect_supply_net net_name`
`[-ports list]`
`[-pg_type {pg_type_list element_list}]*`
`[-pins list]`
`[-domain domain_name]`
	1.0

`connect_supply_set supply_set_ref`
`{-connect {supply_function {pg_type_list}}}*`
	2.0

`create_logic_net net_name`	2.0

`create_logic_port port_name`
`[-direction <in | out | inout>]`
	2.0

`create_power_domain domain_name`
`[-elements element_list]`
`[-exclude_elements exclude_list]`
`[-include_scope]`
`[-supply {supply_set_handle [supply_set_ref]}*]`
`[-update]`
`[-available_supplies {supply_set_ref}`
	1.0
	2.1

`create_power_switch switch_name`
`{-ack_port {port_name [net_name]}`
`{-output_supply_port {port_name}}`
`{-input_supply_port {port_name}}`
`{-control_port {port_name [net_name]}}`
`{-on_state {state_name input_supply_port {boolean_expression}}}*`
`[-domain domain_name]`
	1.0

`create_pst table_name`
`-supplies list`
	1.0

create_supply_net net_name -domain	1.0
create_supply_port port_name [-domain domain_name] [-direction <in |out>]	1.0
create_supply_set set_name [-function {func_name [net_name]}]* [-update]	2.0
end_power_model	2.1
load_upf upf_file_name [-scope instance_name]	
map_isolation_cell isolation_name -domain domain_name -lib_cells lib_cell_list	1.0
map_level_shifter_cell level_shifter_name -domain domain_name -lib_cells lib_cell_list	1.0
map_power_switch switch_name -domain domain_name -lib_cells lib_cell_list	1.0
map_retention_cell retention_name_list -domain domain_name [-lib_cells lib_cell_list] [-lib_cell_type lib_cel_type]	1.0
name_format [-isolation_prefix string] [-level_shifter_prefix string]	1.0
set_scope instance	2.0
set_domain_supply_net domain_name -primary_power_net supply_net_name -primary_ground_net supply_net_name	1.0

set_isolation isolation_name	
-domain ref_domain_name	
[-elements element_list]	
[-source source_supply_ref]	2.0
[-sink sink_supply_ref]	2.0
[-applies_to <inputs | outputs | both>]	2.0
[-isolation_power_net net_name]	
[-isolation_ground_net net_name]	
[-no_isolation]	
[-isolation_supply_set supply_set_list]	2.0
[-isolation_signal signal_list]	2.0
[-isolation_sense {<high | low>*}]	2.0
[-name_prefix string]	2.0
[-clamp_value {<0 | 1 | any | Z | latch | value>]*}]	2.0
[-location <automatic | self | other | parent>]	2.0
[-diff_supply_only <TRUE | FALSE>]	2.0
[-update]	2.0
[-exclude_elements]	
	2.1
set_isolation_control isolation_name	1.0
-domain domain_name	
-isolation_signal signal_name	
[-isolation_sense <high|low>]	
[-location <automatic | self | other | parent>]	
set_level_shifter level_shifter_name	
-domain domain_name	
[-elements element_list]	
[-no_shift]	
[-source source_domain]	2.0
[-sink sink_domain]	2.0
[-applies_to <inputs|outputs|both>]	
[-rule <low_to_high|high_to_low|both>]	
[-location <automatic | self | other | parent>]	
[-name_prefix string]	2.0
[-input_supply_set supply_set_name]	2.0
[-output_supply_set supply_set_name]	2.0
[-update]	
[-exclude_element]	
	2.1
set_pin_related_supply library_cell	1.0
-pins list	
-related_power_pin supply_pin	
-related_ground_pin supply_pin	
set_port_attributes	2.0
[-ports {port_list}] [-exclude_ports {port_list}]	
[-receiver_supply supply_set_ref]	
[-driver_supply supply_set_ref]	

`set_retention retention_name`
`-domain domain_name`
`[-elements element_list]`
`[-exclude_elements [-retention_power_net net_name]exclude_list]`
`[-retention_power_net net_name]`
`[-retention_ground_net net_name]`
`[-retention_supply_set  ret_supply_set]`
`[-no_retention]`
`[-save_signal logic_net <high | low | posedge |`
`[-restore_signal logic_net <high | low | posedge |`
`[-update]`
	1.0
	
	2.0
	
	2.0
	2.0
	2.0
	2.0

`set_retention_control retention_name`
`-domain domain_name`
`[-save_signal net_name`
`[-restore_signal net_name`
	1.0

`upf_version [string]`	1.0

`use_interface_cell interface_implementation_name`
`[-strategy list_of_isolation_level_shifter_strategies]`
`[-lib_cells lib_cell_list]`
	2.0

`find_objects scope`
`-pattern search_pattern`
`[-object_type <inst | port | net | process>]`
`[-direction <in | out | inout>]`
`[-transitive <TRUE | FALSE>]`
`[-non_leaf|-leaf_only]`
	2.0

`set_related_supply_net power_net`
`[object_list]`
`[-ground]`
`[-power]`
	SNPS Special

IEEE1801 Documentation

For more information about IEEE1801, see the following documents:

Innovus Text Command Reference

Documents the Innovus commands supporting the IEEE1801 flow. Descriptions on native Innovus commands are provided.

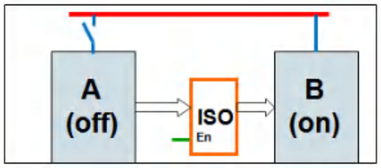
Flow Special Handling for Low Power

Low Power Cells and Usage

In the low power design, several different types of Low Power cells are required and used. These cells are defined in CPF or Liberty. A brief description of each type of cell is as below:

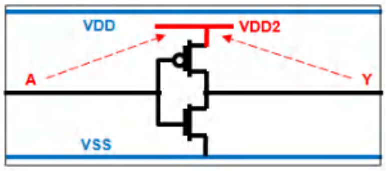
Isolation Cell

Isolation cell is inserted between two domains to prevent signals from floating when the driving domain is powered off. The Isolation logic types can be *high*, *low* or *keep last value*.



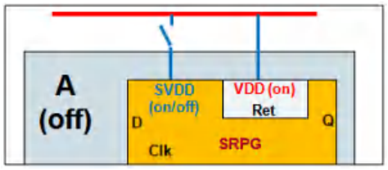
Always-On Cell

The Always-On cell is powered by its second power pin. If it is in a switchable power domain and powered by always-on power, it can stay on when switchable power domain is off.



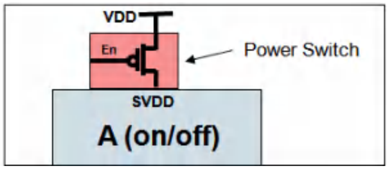
State Retention DFF

A special flop in a switchable domain that can retain its state value when the switchable domain's power supply is turned off. It has secondary power pin to power the retention logic.



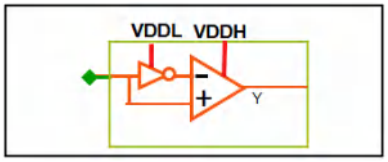
Power Gate or Power Switch Cell

This is a cell used to turn on/off the power supply of a domain.



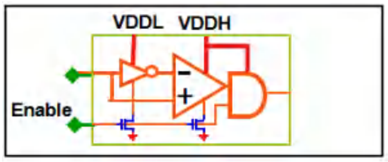
Level Shifter Cell

The Level Shifter cell shifts mainly from lower voltage signal to higher voltage signal or from higher voltage signal to lower voltage signal. It may have a significant delay impact.



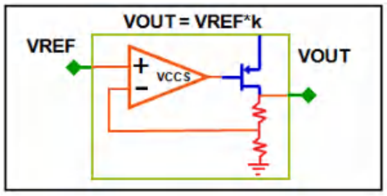
Level Shifter/Isolation Combo Cell

This is a combination of level shifter and isolation cell and is commonly used in designs having both MSV and PSO.



Voltage Regulator Cell (Optional on-chip)

This provides different voltage supply on a chip. Factors like Area, IR-drop and noise need special handling.



Specifying Power Intent

As the low power design becomes more and more complex, the **Power Intent File** is required to capture the low power information such as power domain and low power cell usages.

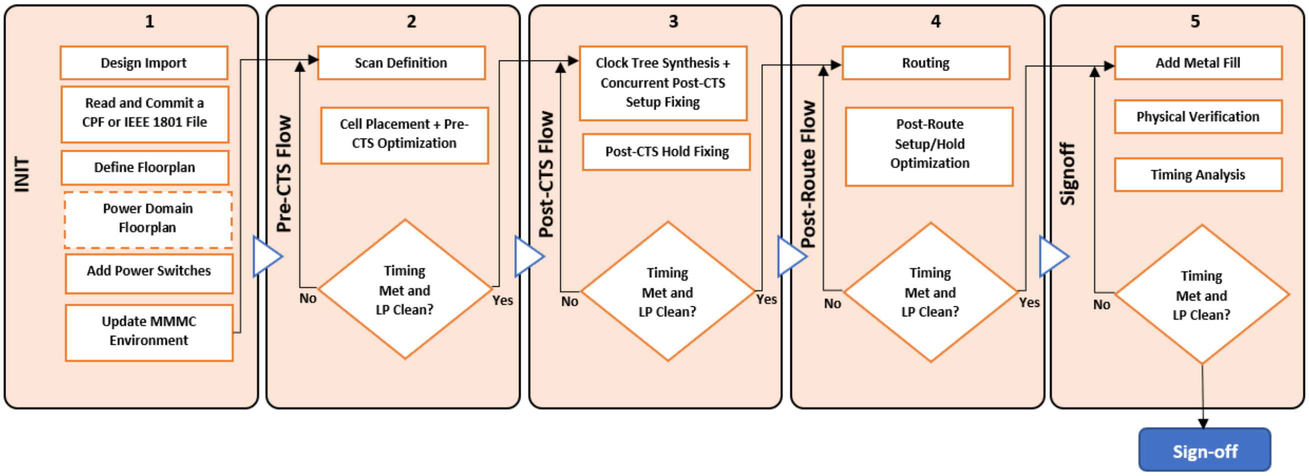
Power Intent File

The Power Intent File mainly specifies the following in TCL-based CPF or IEEE1801:

- Definition of Low Power cells (Can also be defined in Liberty)
- Definition of power nets and nominal condition (PD working voltage)
- Creation and updation of power domains
- Creation and updation of power modes/power state table
- Creation and updation of rules such as iso/ls rule, srpg and power switch rules

The Innovus Low Power Flow

The Low Power flow mainly includes the following steps:



Write power intent file in either CPF or IEEE1801Setting up the Low Power Flow

- Separate MMMC from Power Intent file
- **Write Innovus MMMC (viewDefinition.tcl) separately**
  Make sure each power domain has library binding

Example

Innovus viewDefinition.tcl

```
create_library_set -name wc_0v81\
-timing\
[list ./timing/tcbn45gsbwpwc.lib\
./timing/tcbn45lpbwp_c070208wc0d720d9_modified.lib\
./timing/tcbn45lpbwp_c070208wc0d90d9_modified.lib\
./timing/tcbn45lpbwp_c070208wc_modified.lib\
./timing/tpzn65lpgv2wc.lib]
create_library_set -name wc_0v81_1\
-timing\
[list ./timing/tcbn45gsbwpwc_1.lib\
list ./timing/tcbn45lpbwp_c070208wc0d90d9_modified.lib]
create_library_set -name bc_0v81\
-timing\
[list ./timing/tcbn45gsbwpbc.lib\
./timing/tcbn45lpbwp_c070208bc0d881d1_modified.lib\
./timing/tcbn45lpbwp_c070208bc1d11d1_modified.lib\
./timing/tcbn45lpbwp_c070208bc_modified.lib\
./timing/tpzn65lpgv2bc.lib]
create_op_cond -name PM_wc_virtual -library_file \
./timing/tcbn45gsbwpwc.lib -P 1 -V 0.81 -T 125
create_op_cond -name PM_bc_virtual -library_file \
./timing/tcbn45gsbwpbc.lib -P 1 -V 0.99 -T 0
 create_rc_corner -name rc_cworst \
-cap_table worst.CapTbl
create_delay_corner -name AV_PM_on_dc\
-library_set wc_0v81\
-opcond_library tcbn45gsbwpwc\
-opcond PM_wc_virtual -rc_corner rc_cworst
update_delay_corner -name AV_PM_on_dc -power_domain PDdefault\
-library_set wc_0v81_1\
-opcond_library tcbn45gsbwpwc\
-opcond PM_wc_virtual
update_delay_corner -name AV_PM_on_dc -power_domain PD1\
-library_set wc_0v81\
-opcond_library tcbn45gsbwpwc\
-opcond PM_wc_virtual
update_delay_corner -name AV_PM_on_dc -power_domain PD2\
-library_set wc_0v81\
-opcond_library tcbn45gsbwpwc\
-opcond PM_wc_virtual
update_delay_corner -name AV_PM_on_dc -power_domain PD3\
-library_set wc_0v81_1\
-opcond_library tcbn45gsbwpwc\
-opcond PM_wc_virtual
```

Note: Specify each domain binding by the `update_delay_corner -power_domain` command.

Low Power DB and ENV Creation

`read_power_intent -cpf` or `-1801/commit_power_intent` mainly does the following:

- Check the CPF/IEEE1801 syntax
- Create power domains (groups) and site list for each domain
- Read and commit CPF/IEEE1801 rules (if required)
- Synthesize the simple logic for low power enabled signals
- Generate global and tie connection specifications
- Create the implicit ISO/LS rules for optimization/CTS/`verifyPowerDomain`

Once the above is done, Lower power DB (called *.cpfdb), is generated to keep the low power information such as power domain, PG nets and low power rules. The DB is saved by `saveDesign` and can be restored by `restoreDesign`.

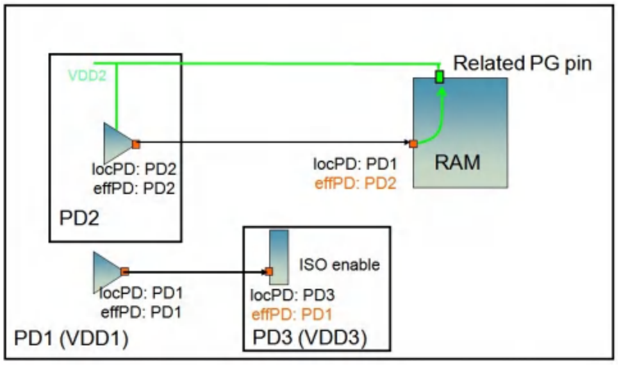
Pin/Term Power Domain Assignment

Each LEAF signal pin/term and boundary port are assigned to a power domain based on its location, or its related PG pin, or its driving instance, or the definition in CPF/IEEE1801.

The Low Power cell enable pin is assigned to its driving pin domain. Low Power cells and Macros signal pins are assigned according to its “related PG pin”. Some signal pins can also be assigned in CPF or IEEE1801. Regular standard cell leaf instance pin/or pin without “related PG pin” is assigned to its instance power domain (location domain).

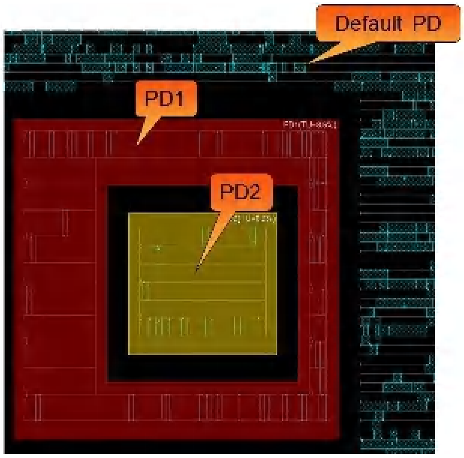
In the following diagram, locPD is the location power domain. effPD is the pin's effective power domain whose primary PG nets drive the pin in reality. locPD can be different from effPD.

The pin’s effective power domain is used to determine if there is domain crossing from the driver to receiver during low power cell insertion, optimization, CCOPT, and `verifyPowerDomain`.



Logical vs. Physical Power Domain

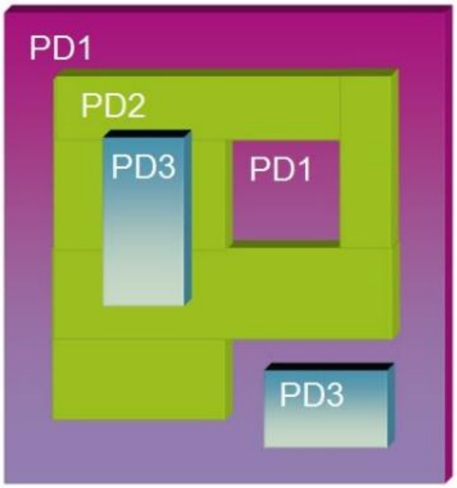
Apart from logical power domain definition in CPF or IEEE1801, power domain in Innovus also means that all of the standard cells (members) in the same domain share the same follow-pin net and standard cell row structure.



In the above example, LEF's "Site" is used to control which cells are allowed in this row. Note that different power domain may have different standard cell height, and the non-default power domain with standard cell members has fence constraint.

Complex Power Domain Floorplan

Innovus supports various types of the power domain fence shapes such as rectangular, rectilinear, donut and disjoint.



- Ring shape power domain (PD1); Non-default power domain in the middle
- Donut shape power domain (PD2)
- Nested power domain (PD3): It can be either physically nested (PD2), or logically nested, or both
- Disjoint power domain (PD3)

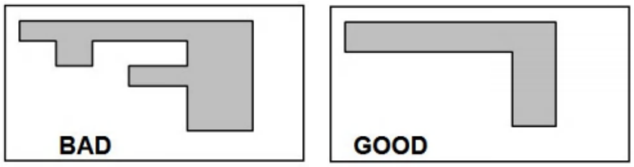
Power Domain Floorplanning Guideline

- The power domain floorplan (size, shape and location) greatly affects the Cell and ISO/LS placement QoR, Optimization QoR and Routing congestion.
- You should **try** to create the rectangular or simple rectilinear shapes with mini PD boundary edges. This will minimize the crossing-PD route patterns and help ISO/LS placement.
- **Avoid** dividing/blocking power domain fence into pieces, avoid creating the narrow channel between domains or abutted domain.

Example

Minimize the number of PD boundary edges

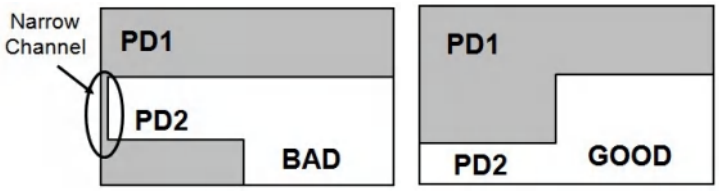
- Reduce the feedthrough routes
- Help ISO/LS placement



Example

Avoid narrow channel

Reduce the feedthrough routes and congestion in the channel



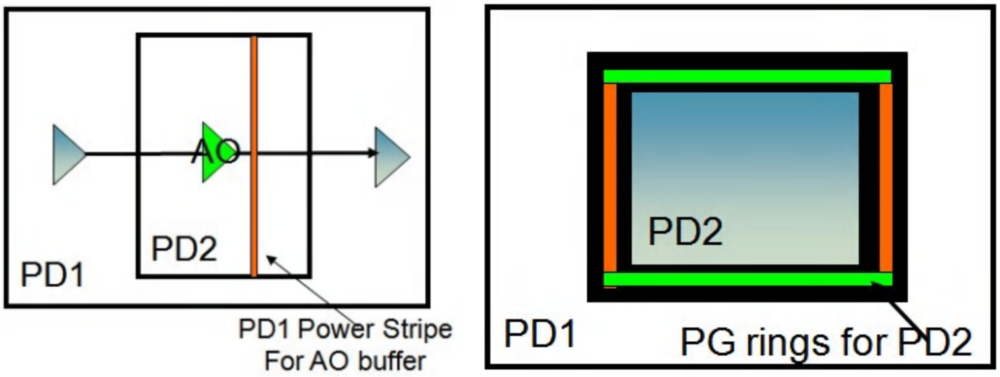
Low Power Planning and Routing

Power planning

In the Power Planning process, you need to plan power stripes for AO feedthrough bufferring inside the power domain. It is better to create PG rings for power domain to reduce IR drop.

Power routing

In the Power Routing process, you need to use `addStripe` command to route power switch always-on power over the switches, and the `routePGPinUseSignalRoute` command to route secondary PG pins of the Low Power cells such as AO, SRPG, isolation and level shifter.

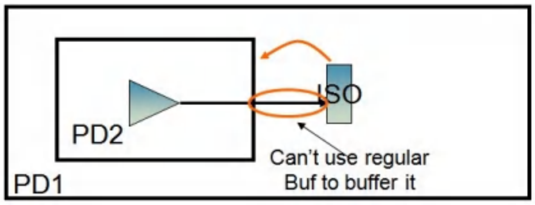


For more information on adding stripes, see `addStripe`.

Cell and ISO/LS placement

The cell placement is Power Domain (PD) aware and honors the PD fence constraints.

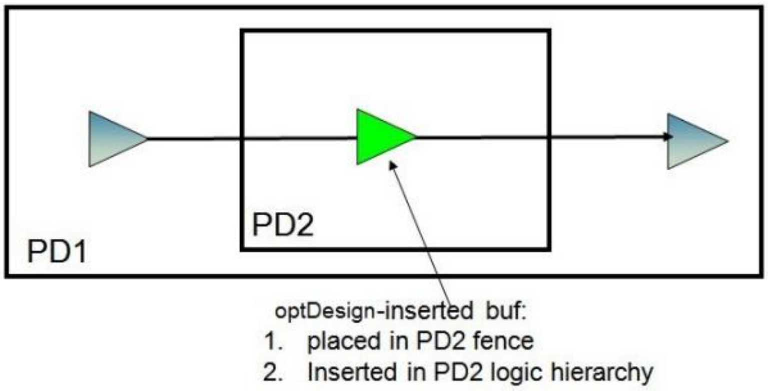
The goal of ISO/LS placement is to increase the chance that optimization can buffer either input or output net using as many regular buffers as possible. The placement needs to place ISO/LS close to power domain boundary or near its driver/receiver.



You can control ISO/LS placement or fix ISO/LS by using `setPlaceMode`, creating ISO/LS group constraints, setting the ISO/LS cell padding, creating the routing blockage along domain boundary and creating the pin guide.

Low Power Optimization

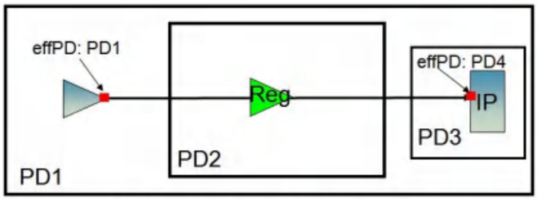
- `optDesign` is Power Domain (PD) aware and inserts buffers along the route
- Extra physical and logic constraints must be applied in LP Optimization
  - Physical constraint: Buffers needs to be placed in its domain fence
  - Logic constraint: Buffer needs to be pushed down/pulled up to its domain logic hierarchy



Example

Regular buffer insertion

- Determine the location: Green
- Get the location PD: PD2
- Power (domain) coverage analysis
  - Can PD1 drive PD2 and PD2 drive PD4 (no off->on or low->high)
  - Yes, Use Reg buffer
- Pick up Reg buffer from PD2 lib binding; insert it into PD2 logic hierarchy



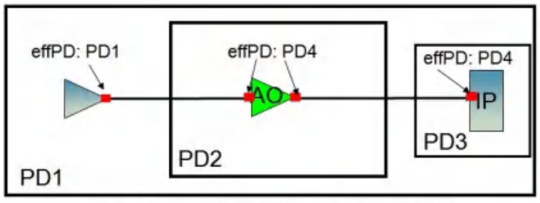
PD1 can drive PD2; PD2 can drive PD4

Example

Always On (AO) buffer insertion

- Determine the location: Green
- Get the location PD: PD2
- Power (domain) coverage analysis
  - Can PD1 drives PD2 and PD2 drives PD4 (no off->on or low->high)

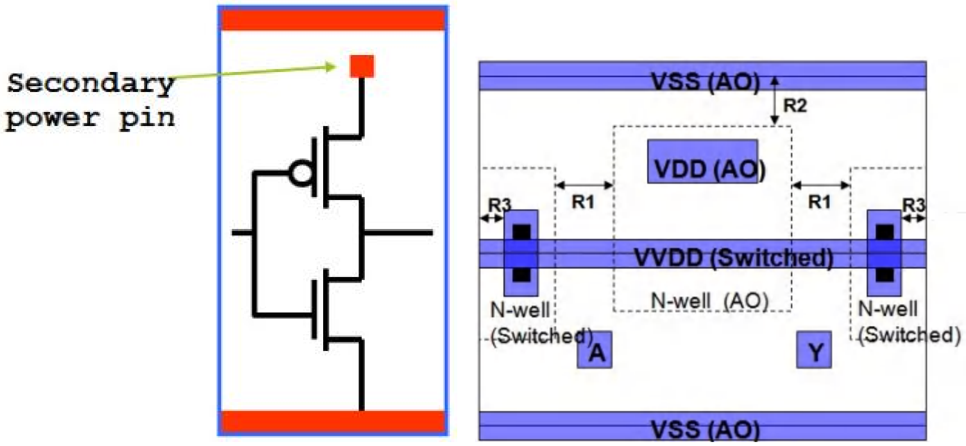
  - No, Use AO buffer
- Pick up AO buffer from PD2 lib binding; insert it into PD2 logic hierarchy
  - Connect AO 2nd power pin to PD4's power
  - Assign AO input and output pin to PD4



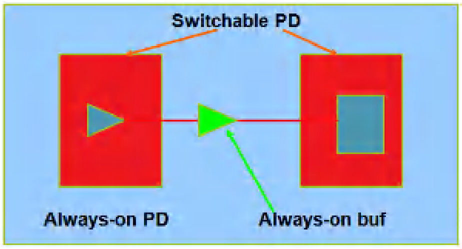
PD1 can drive PD2; PD2 can NOT drive PD4

LP Optimization Using Always-on Buffers

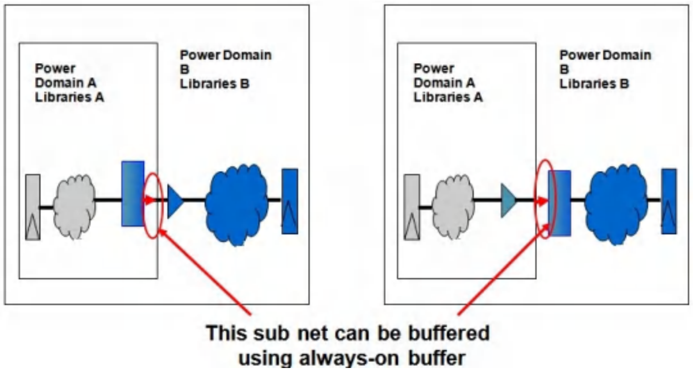
Always-on (AO) buffer is a buffer with two power/ground pins where the secondary power pin is used to supply the power for the always-on buffer. The cell can be mixed into a switched power domain because its primary power pin is compatible with the row's power rail (follow pins), has built-in Nwell for secondary and primary, powers for abutment with other cell. AO Buffers are required to maintain power domain compatibility when the local power domain is not compatible with its sink or receiver.



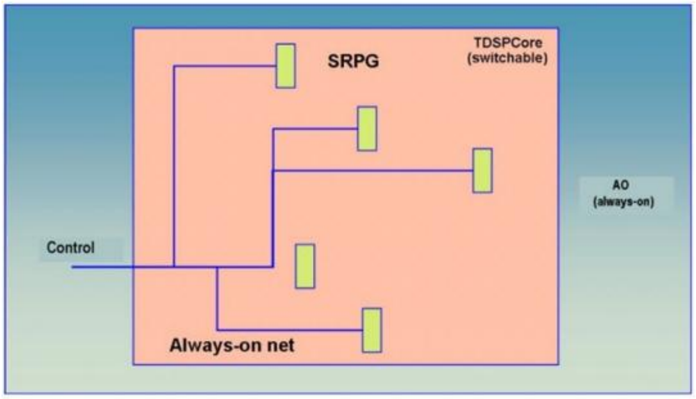
- Feedthrough Net



- The subnet between ISO/LS input/output pin and domain boundary


Power Domain A Libraries A
Power Domain B Libraries B
Power Domain A Libraries A
Power Domain B Libraries B
This sub net can be buffered using always-on buffer

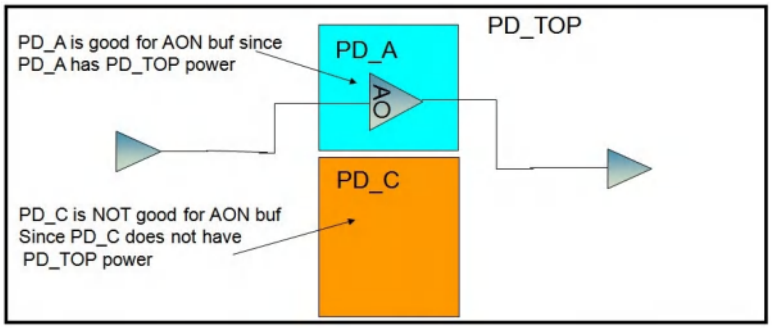
- AO net in switchable domain such as ISO, PS and SRPG enabled



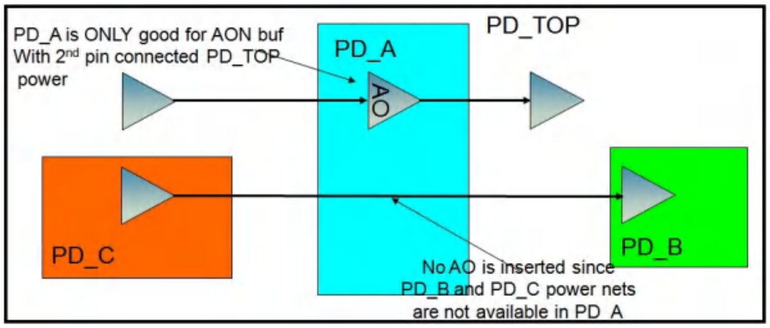
Disable AO Buffering Based On The Unavailable Power Nets In The Specific Domain

CPF Command:

```
update_power_domain -name PD_C -user_attribute {{disable_secondary_domains {PD_TOP}}}
```


PD_TOP
PD_A
AO
PD_C
PD_A is good for AON buf since PD_A has PD_TOP power
PD_C is NOT good for AON buf Since PD_C does not have PD_TOP power

Enable AO Buffering Based On The Available Power Nets In The Specific Domain


PD_A is ONLY good for AON buf
With 2nd pin connected PD_TOP
power
PD_A
AO
PD_TOP
PD_C
PD_B
No AO is inserted since
PD_B and PD_C power nets
are not available in PD_A

CPF Command:

```
update_power_domain -name PD_A -user_attribute {{enable_secondary_domains {PD_TOP}}}
```

PD_A can ONLY insert AO with its second power pin connected to PD_TOP’s power

IEEE1801 Command:

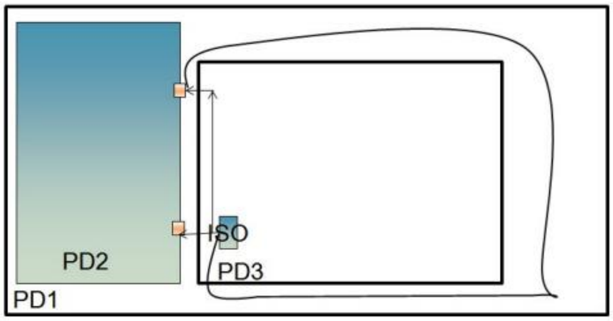
```
create_power_domain PD_A -available_supplies <PD_TOP’s supply set>
```

PD_A can ONLY insert AO with its 2nd power pin connected to PD_TOP’s power

Regular Buffer versus AO buffer (cost-based buffering)

Minimize the number of AO buffering. The AO buffer is set at a higher cost.

Optimization is able to generate different topologies/routes for regular or AO buffering. It can choose which topologies/route for buffering according to a pre-defined cost function.



ecoAddRepeater - A Handy Interactive Command

The `ecoAddRepeater` command provides flexibility to buffer the crossing domain nets or build buffer trees in any arbitrary logic hierarchy.

Secondary Power Pin Routing

- The following LP cells have the secondary power pins:
    - Always-on buffers, SRPG, Level Shifters, Isolation Cells, Comb Cell and Power Switch Cell
    - The cell’s secondary power pin is defined in the CPF or *lib
    - The second power pin connection is defined in CPF or IEEE1801 or floorplan file

  - This command can be used after Low Power cells are inserted and placed
- Route second power pin
  - The power switch second power pin is routed by power planning command `addStripe`
  - The second power pins for the other cells are routed by nanoroute commands `routePGPinUseSignalRoute` and `setNanoRouteMode`
- Power planning for second power pin route
  - Must add the second power stripes for the switchable domain
  - Need to add some other power stripes for the feedthrough AO buffering
- ECO routing in `optDesign -postRoute` can automatically do secondary power pin ECO route. ECO routes for the cells and pins are defined in `setPGPinUseSignalRoute cell1:pin1 cell2:pin2`
- Route secondary power pins before signal route

Low Power Design Verification

- The design must match the power intent such as:
  - Isolation cell is required from Off to On
  - Level shifter is required from Low to High
- To verify Low Power design against CPF/IEEE1801 by CPF/IEEE1801 rules, use the low Power verification command `verifyPowerDomain`
- To verify Low Power design against CPF/IEEE1801 by tracing PG connection in the physical netlist, use the command `runCLP` (physical)

Low Power Debugging Commands

Run the following command to get information about driver/receiver domain, domain coverage, pin power domain, related pg pin:

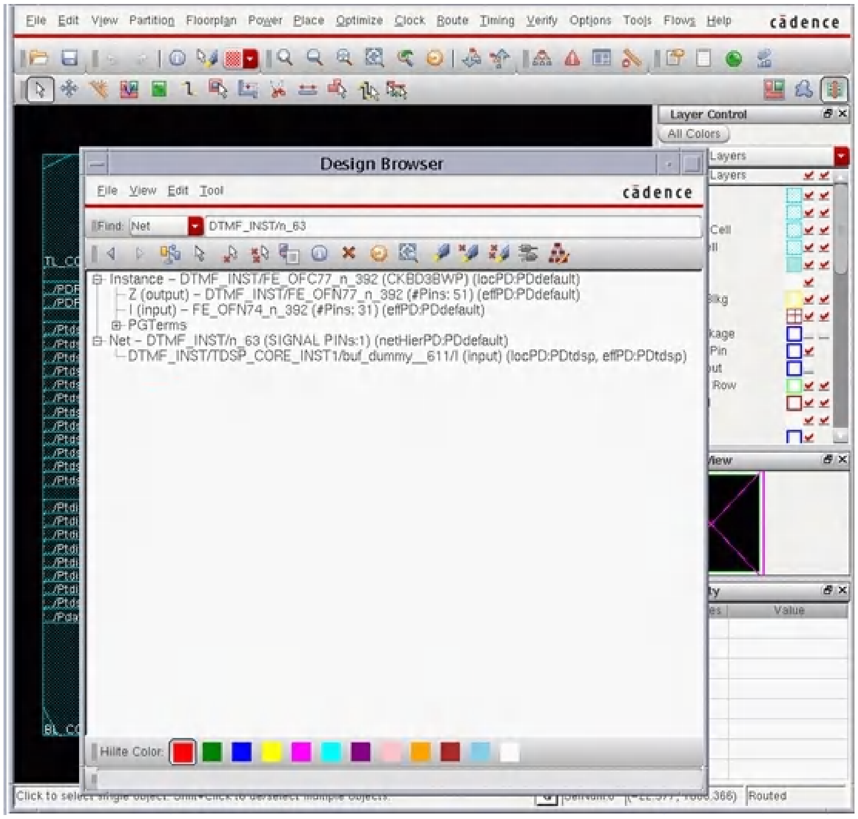
```
reportPowerDomain -inst |-net|-pin|-powerDomain -verbose
```

Tcl db access:

```
dbGet top.pds.??
```

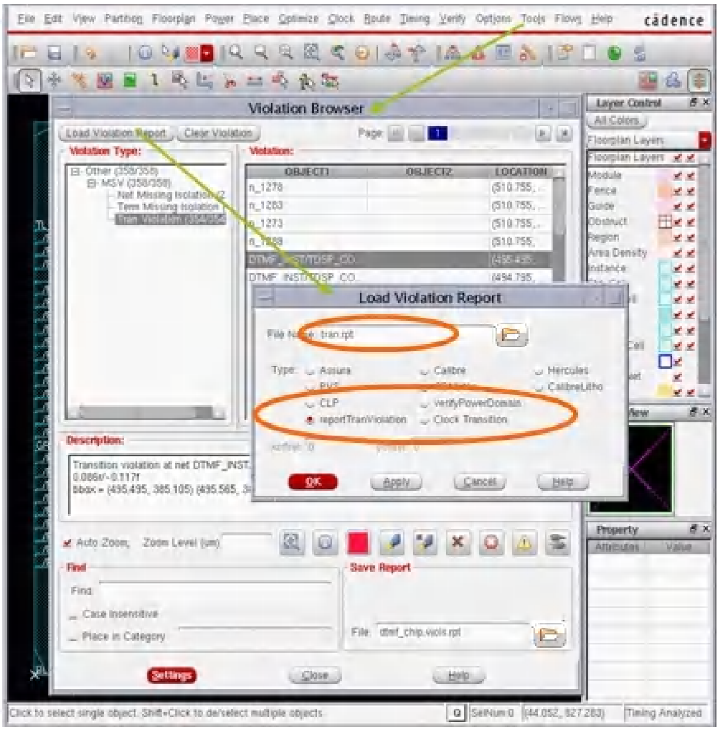
Low Power GUI Debugger

Design Browser



```
⊟ Instance – DTMF_INST/FE_OFC77_n_392 (CKBD3BWP) (locPD:PDdefault)
   ├ Z (output) – DTMF_INST/FE_OFN77_n_392 (#Pins: 51) (effPD:PDdefault)
   ├ I (input) – FE_OFN74_n_392 (#Pins: 31) (effPD:PDdefault)
   ⊞ PGTerms
⊟ Net – DTMF_INST/n_63 (SIGNAL PINs:1) (netHierPD:PDdefault)
   └ DTMF_INST/TDSP_CORE_INST1/buf_dummy__611/I (input) (locPD:PDtdsp, effPD:PDtdsp)
```

Violation browser and Schematic Viewer Screen Capture



Other (358/358)
MSV (358/358)
Net Missing Isolation (2
Term Missing Isolation
Tran Violation (354/354

OBJECT1	OBJECT2	LOCATION
n_1278		(510.755, ..
n_1283		(510.755, ..
n_1273		(510.755, ..
n_1268		(510.755, ..
DTMF_INST/TDSP_CO..		(495.495,
DTMF_INST/TDSP_CO..		(494.795,

File Name: tran.rpt

Type: ○ Assura ○ Calibre ○ Hercules ○ PVS ○ CalibreLitho ○ CLP ○ verifyPowerDomain ● reportTranViolation ○ Clock Transition

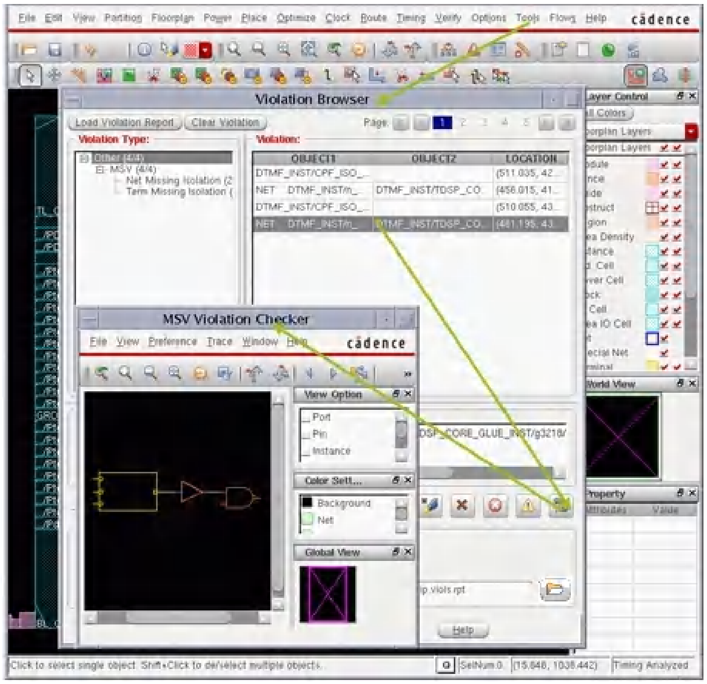
Description:
Transition violation at net DTMF_INST..
0.086r/-0.117f
bbox = (495.495, 385.105) (495.565, 3..

☑ Auto Zoom; Zoom Level (um)

Find:
_ Case Insensitive
_ Place in Category

File: dtmf_chip.viols.rpt

Click to select single object. Shift+Click to de/select multiple objects. SelNum:0 (44.052, 827.283) Timing Analyzed



Other (4/4)
MSV (4/4)
Net Missing Isolation (2
Term Missing Isolation (

OBJECT1	OBJECT2	LOCATION
DTMF_INST/CPF_ISO_..		(511.035, 42..
NET DTMF_INST/n_..	DTMF_INST/TDSP_CO..	(456.015, 41..
DTMF_INST/CPF_ISO_..		(510.055, 43..
NET DTMF_INST/n_..	DTMF_INST/TDSP_CO..	(481.195, 43..

..DSP_CORE_GLUE_INST/g3218/

..ip.viols.rpt

Click to select single object. Shift+Click to de/select multiple objects. SelNum:0 (15.848, 1038.442) Timing Analyzed

Multiple Supply Voltage Top-Down Hierarchical Flow

This section discusses the following topics:

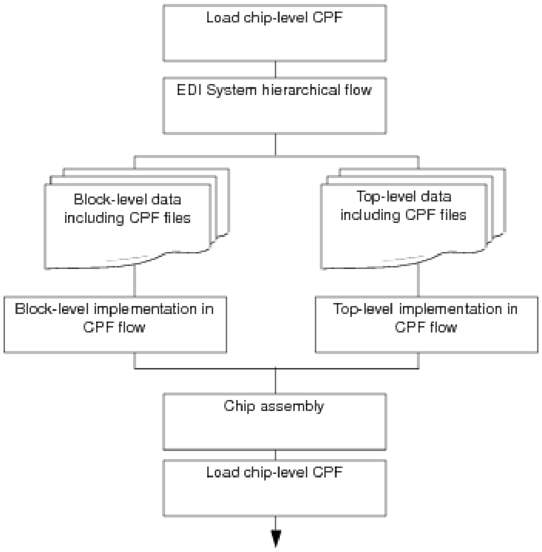
- Overview
- Always-On Feedthrough Handling
- Chip Partitioning
- Block-level CPF Generation
- Top-Level CPF Generation
- Transition to OpenAccess for Low Power Flow
- Block-Level Implementation
- Top-Level Implementation
- Chip Assembly

Overview

The Innovus tool supports the low power top-down CPF-based hierarchical flow built on the regular Innovus hierarchical flow. The difference is that, in the CPF-based hierarchical flow, you partition the chip-level CPF file into block-level CPF files and a top-level CPF file. You then use those CPF files to implement the block level and top level designs.

The CPF-based hierarchical flow supports the following scenarios:

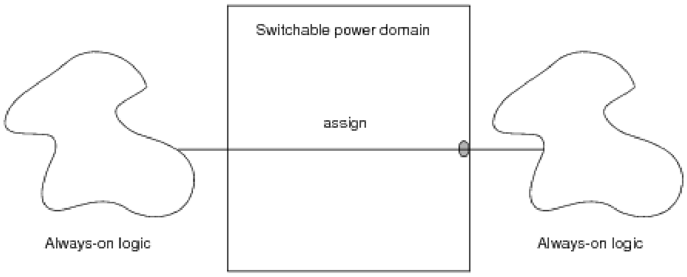
- The partition is physically the same as the power domain. The hierarchical instance is logically the same for power domain and partition.
- The power domain is physically inside partition. The power domain hierarchical instance is logically a sub hierarchical instance of partition.
- Partition is physically inside power domain. The partition hierarchical instance is logically one (and only one) of the hierarchical instances of power domain.

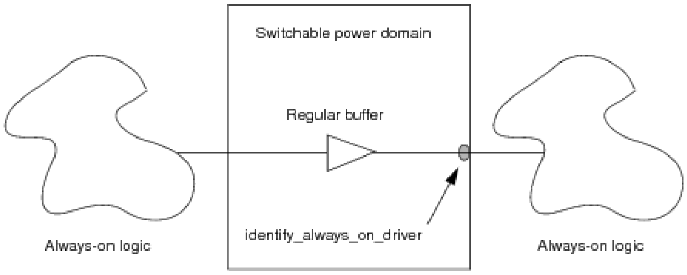

Load chip-level CPF
EDI System hierarchical flow
Block-level data including CPF files
Top-level data including CPF files
Block-level implementation in CPF flow
Top-level implementation in CPF flow
Chip assembly
Load chip-level CPF

Always-On Feedthrough Handling

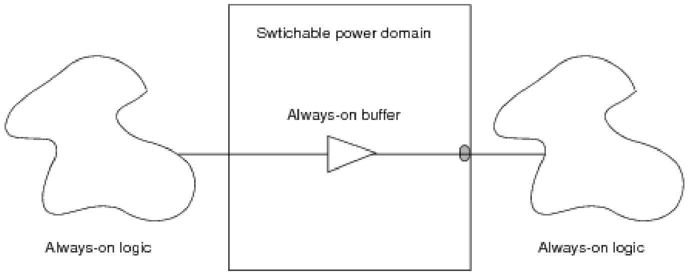
In the low power flow, the tool can insert a feedthrough buffer in a partition that resides in a shut-off power domain.

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. For more information, see the Timing Budgeting chapter of the *Innovus User Guide*.

   Note: Use the `deriveTimingbudget` and `saveTimingBudget` commands for deriving the MMMC timing budget.

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.

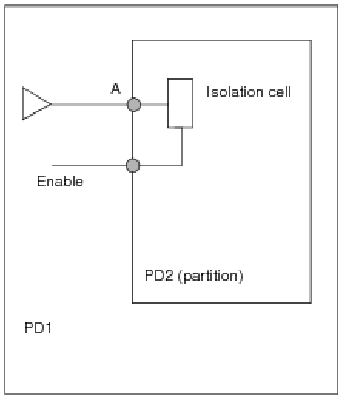
Block-level CPF Generation

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:

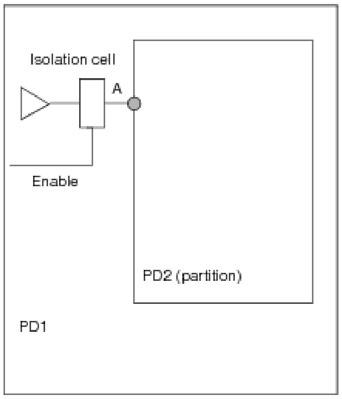
- Low power information in CPF
  - Pushes down naming style, hierarchy separator, CPF version, and library set definitions
  - Pushes down low power cell definitions
  - Creates the power domains referenced by the block-level CPF files
  - Creates the power domains' power/ground nets and connections
  - 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.

> 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 -cpfFile`.

2. Load the chip-level CPF after the assembly to restore the low power settings and continue to chip-level verification and chip finishing.

Example of Block-Level CPF Generated by Innovus

Note: A special construct is used to avoid duplicate definition for timing-related CPF files when sourced by top-level CPF. If the `[set_instance]==[set_hierarchy_separator]` condition is `true`, then the tool recognizes the implementation is at the block level, and loads the related timing information. If the condition is `False`, then the tools sources the block-level CPF file at top level, and does not load the timing-related information.

```
set_design tdsp_core \
-ports {n_41}

set_hierarchy_separator "/"

create_power_domain -name TDSPCore -default \
-shutoff_condition {n_41}

create_power_nets -nets VDD_TDSPCore \
-voltage 0.792 \
-internal

update_power_domain -name TDSPCore \
-internal_power_net {VDD_TDSPCore}

create_power_nets -nets VDD_TDSPCore_R \
-voltage 0.792

create_global_connection -net VDD_TDSPCore_R \
-domain TDSPCore \
-pins TVDD

create_global_connection -net VDD_TDSPCore \
-domain TDSPCore \
-pins VDD

create_ground_nets -nets VSS

create_global_connection -net VSS \
-domain TDSPCore \
-pins VSS

create_power_domain -name AO
-boundary_ports { clk reset SRPG_PG_in SRPG_PG_in_1 DFT_sen n_41,…}

create_power_nets -nets VDD -voltage 0.792

update_power_domain -name AO -internal_power_net {VDD}

if {[set_instance]==[set_hierarchy_separator]} {

define_library_set -name ao_wc_0v99
-libraries { ../../LIBS/N45/timing/wc.lib}

define_library_set -name ao_wc_0v792 \
-libraries { ../../LIBS/N45/timing/AOwc0d72.lib}

define_library_set -name tdsp_wc_0v792 \
-libraries { ../../LIBS/N45/timing/wc0d72.lib}

create_operating_corner -name WC08COM_AO \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set ao_wc_0v792

create_operating_corner -name WC08COM_TDSP \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set tdsp_wc_0v792

create_nominal_condition -name low_ao -voltage 0.792

update_nominal_condition -name low_ao -library_set ao_wc_0v792

create_nominal_condition -name off -voltage 0

create_power_mode -name PM_LO_FUNC \
-domain_conditions { AO@low_ao TDSPCore@off} \
-default

update_power_mode -name PM_LO_FUNC \
-sdc_files {../../RELEASE/mmmc/dtmf_recvr_core_dull.sdc}

create_analysis_view -name AV_LO_FUNC_MAX_RC1 \
-mode PM_LO_FUNC \
-domain_corners { AO@WC08COM_AO TDSPCore@WC08COM_TDSP}
}

define_isolation_cell -cells {LVLLH} \
-ground {VSS} \
-enable {NSLEEP} \
-valid_location {to} \
-power {VDD}

define_level_shifter_cell -cells {LVLH} -valid_location {to} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099}
-ground {VSS} -direction {down}

define_level_shifter_cell -cells {PTLVLH} \
-valid_location {to} \
-direction {down} \
-output_power_pin {TVDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} -input_voltage_range {0.792:0.99:0.099}

define_level_shifter_cell -cells {LVLLH} \
-valid_location {to} \
-input_power_pin {VDDL} \
-output_power_pin {VDD}
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-output_voltage_input_pin {NSLEEP}\
-ground {VSS} \
-direction {up}

define_level_shifter_cell -cells {LVLLHD} \
-input_voltage_range {0.792:0.99:0.099} \
-valid_location {to} \
-direction {up} \
-output_power_pin {VDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_power_pin {VDDL}

define_state_retention_cell -cells {RSDF} \
-ground {VSS} \
-save_function {SAVE} -power {TVDD} \
-restore_function {!NRESTORE} \
-clock_pin {CP} \
-power_switchable {VDD}

define_power_switch_cell -cells {HDRDID HDRDIA}
-stage_2_enable {!NSLEEPIN2} \
-stage_1_output {NSLEEPOUT1} \
-power {TVDD} \
-stage_2_output {NSLEEPOUT2} \
-power_switchable {VDD} \
-stage_1_enable {!NSLEEPIN1} \
-type {header}

define_always_on_cell -cells {PTBUFF PTLVLH} \
-ground {VSS} \
-power {TVDD} \
-power_switchable {VDD}

create_level_shifter_rule -name LSRULE_H2L \
-to {TDSPCore} \
-from {AO} \
-exclude {n_41 SRPG_PG_in SRPG_PG_in_1}

update_level_shifter_rules -names LSRULE_H2L \
-cells {LVLH} \
-location {to}

create_level_shifter_rule -name LSRULE_H2L_AO
-from {AO} \
-to {TDSPCore} \
-pins {n_41 SRPG_PG_in SRPG_PG_in_1}

update_level_shifter_rules -names LSRULE_H2L_AO
-location {to} \
-cells {PTLVLH}

create_state_retention_rule -name SRPG_TDSP \
-save_edge {SRPG_PG_in} \
-domain {TDSPCore} \
-restore_edge {!SRPG_PG_in_1}

update_state_retention_rules -names SRPG_TDSP \
-cell {RSDF} \
-library_set {tdsp_wc_0v792}

create_power_switch_rule -name TDSPCore_SW \
-domain {TDSPCore} \
-external_power_net {VDD_TDSPCore_R}

update_power_switch_rule -name TDSPCore_SW \
-prefix {CDN_SW_} \
-cells {HDRDID} \
-acknowledge_receiver {switch_en_out}

end_design
```

Example of Top-Level CPF Generated by Innovus

```
set_cpf_version 1.0

set_design dtmf_recvr_core

set_hierarchy_separator "/"

create_ground_nets -nets Avss

create_ground_nets -nets VSS

create_power_nets -nets VDD \
-voltage 0.792

create_power_nets -nets Avdd \
-voltage 0.990

create_power_nets -nets VDD_TDSPCore_R \
-voltage 0.792

create_power_nets -nets VDD_TDSPCore \
-voltage 0.792 \
-internal

define_library_set -name ao_wc_0v99 \
-libraries { ../LIBS/N45/timing/tcbn45lpbwp_c060907wc.lib}

define_library_set -name ao_wc_0v792 \
-libraries { ../LIBS/N45/timing/tcbn45lpbwp_c060907wc0d72.lib}

define_library_set -name tdsp_wc_0v792 \
-libraries { ../LIBS/N45/timing/tcbn45lpbwp_c060907wc0d72.lib}

source cpf_1.0_hierarchical_PD.tcl

set_instance TDSP_CORE_INST \
-port_mapping { {n_41 PM_INST/power_switch_enable}} \
-domain_mapping { {TDSPCore TDSPCore} {AO AO}}

source TDSP_CORE_INST.cpf

create_power_domain -name TDSPCore \
-shutoff_condition {PM_INST/power_switch_enable}

create_global_connection -net VDD_TDSPCore_R \
-domain TDSPCore \
-pins TVDD

create_global_connection -net VDD_TDSPCore \
-domain TDSPCore \
-pins VDD

create_global_connection -net VSS \
-domain TDSPCore \
-pins VSS

create_power_domain -name AO \
-default

update_power_domain -name AO \
-internal_power_net {VDD}

create_global_connection -net VDD_TDSPCore \
-domain AO \
-pins VDDL
```

```
create_global_connection -net VDD \
-domain AO \
-pins VDD

create_global_connection -net VSS \
-domain AO \
-pins VSS

create_power_domain -name PLL \
-instances { PLLCLK_INST}

update_power_domain -name PLL \
-internal_power_net {Avdd}

create_global_connection -net VDD \
-domain PLL \
-pins VDDL

create_global_connection -net Avdd \
-domain PLL \
-pins avdd!

create_global_connection -net Avdd \
-domain PLL \
-pins VDD

create_global_connection -net Avss \
-domain PLL \
-pins VSS

create_global_connection -net Avss \
-domain PLL \
-pins agnd!

create_operating_corner -name WC08COM_AO \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set ao_wc_0v792

create_operating_corner -name WC08COM_TDSP \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set tdsp_wc_0v792

create_nominal_condition -name low_ao \
-voltage 0.792

update_nominal_condition -name low_ao \
-library_set ao_wc_0v792

create_nominal_condition -name off -voltage 0

create_power_mode -name PM_LO_FUNC \
-domain_conditions { AO@low_ao PLL@high_ao TDSPCore@off}

update_power_mode -name PM_LO_FUNC \
-sdc_files {../RELEASE/mmmc/dtmf_recvr_core_dull.sdc

create_analysis_view -name AV_LO_FUNC_MAX_RC1 \
-mode PM_LO_FUNC \
-domain_corners { AO@WC08COM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}

define_isolation_cell -cells {LVLLH} \
-ground {VSS} \
-enable {NSLEEP} \
-valid_location {to} \
-power {VDD}

define_level_shifter_cell -cells {LVLH} \
-valid_location {to} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-direction {down}

define_level_shifter_cell -cells {PTLVLH} \
-valid_location {to} \
-direction {down} \
-output_power_pin {TVDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_voltage_range {0.792:0.99:0.099}

define_level_shifter_cell -cells {LVLLH} \
-valid_location {to} \
-input_power_pin {VDDL} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-output_voltage_input_pin {NSLEEP} \
-ground {VSS} -direction {up}

define_level_shifter_cell -cells {LVLLHD} \
-input_voltage_range {0.792:0.99:0.099} \
-valid_location {to} \
-direction {up} \
-output_power_pin {VDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_power_pin {VDDL}

define_state_retention_cell -cells {RSDF} \
-ground {VSS} \
-save_function {SAVE} \
-power {TVDD} \
-restore_function {!NRESTORE} \
-clock_pin {CP} -power_switchable {

define_power_switch_cell \
-cells {HDRDID HDRDIAO} \
-stage_2_enable {!NSLEEPIN2} \
-stage_1_output {NSLEEPOUT1} \
-power {TVDD} \
-stage_2_output {NSLEEPOUT2} \
-power_switchable {VDD} \
-stage_1_enable {!NSLEEPIN1} \
-type {header}

define_always_on_cell -cells {PTBUFF PTLVLH} \
-ground {VSS} \
-power {TVDD} \
-power_switchable {VDD}
```

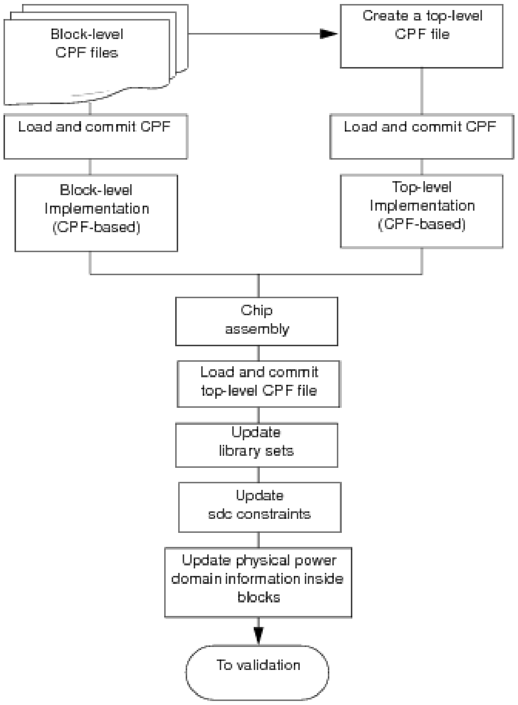
```
create_isolation_rule -name ISORULE \
-from {TDSPCore} \
-isolation_condition {!PM_INST/isolation_enable} \
-isolation_output {high}

update_isolation_rules -names ISORULE \
-location {to} \
-cells {LVLLH}

create_level_shifter_rule \
-name LSRULE_H2L_PLL \
-from {PLL} \
-to {AO}

update_level_shifter_rules -names LSRULE_H2L_PLL \
-location {to} \
-cells {LVLHLD}

end_design
```

Multiple Supply Voltage Bottom-Up Hierarchical Flow

This section discusses the following topics:

- Block-Level Implementation
- Top-Level Implementation
- Chip Assembly

The Innovus tool supports the low power bottom-up CPF-based hierarchical flow built on the regular Innovus bottom-up hierarchical flow. The difference is that you use the existing block-level CPF files to construct the top-level hierarchical CPF file, and implement the design using the CPF flow.


Block-level CPF files
Create a top-level CPF file
Load and commit CPF
Load and commit CPF
Block-level Implementation (CPF-based)
Top-level Implementation (CPF-based)
Chip assembly
Load and commit top-level CPF file
Update library sets
Update sdc constraints
Update physical power domain information inside blocks
To validation

Block-Level Implementation

You can use any combination of hard and soft blocks.

For the hard blocks (that are already implemented), place the blocks in top-level floorplan.

For the soft blocks,

- Load and commit the block-level CPF files.
- Implement the blocks using the block-level CPF implementation flow.

After completing block-level implementation,

- Save the block-level design in def format for chip assembly.

  ```
  saveDesign -def
  ```

- If a power domain exists inside a block, use the following command to obtain the physical information about the power domain:

  ```
  write_power_intent -cpf tmp.cpf
  ```

The tool restores the physical information for the power domain inside block (partition) after chip assembly.

Top-Level Implementation

Before top-level implementation, manually build the top-level CPF file by reusing the block-level CPF files as follows:

```
Set_instance HinstOfBlock -domain_mapping { {..} } -port_mapping { {..} }
Source block.cpf
```

The CPF file you create contains the same type of information as in the previous example file: Example of Top-Level CPF Generated by Innovus

To implement the design, use the CPF implementation flow using the hierarchical top-level CPF file.

After completing top-level implementation, use the following command to save the top-level design in DEF format:

```
saveDesign -def
```

Chip Assembly

To assemble the design's physical data, use the following command:

```
assembleDesign
```

After chip assembly,

- To restore the lower power setup for chip-level verification and chip finishing, load and commit the top-level hierarchical CPF file.
- (Optional) Update power domain physical information inside block.
  - To update power domain shape, use the following command:

    ```
    setObjFPlanBox
    ```

  - To update the power domain attribute, use the following command:

    ```
    modifyPowerDomainAttr
    ```

Note: These steps are necessary only if you have a power domain inside a partition.

- To update the library set, use the following command:

  ```
  update_library_set -name -timing {..}
  ```

- To apply chip-level timing constraints (sdc files), use the following command:

  ```
  update_constraint_mode -name -sdc_files
  ```

- Proceed to design verification.

Leakage Power Optimization Techniques

- Multi-Vth Optimization
- Substrate Biasing

Multi-Vth Optimization

You can optimize non-critical path logic for leakage, while preserving the critical timing slack (WNS).

Note: Run multi-Vth optimization only after your design meets timing.

- Report the total leakage power in the design using the following command:

  ```
  report_power -leakage
  ```

- If you want to obtain a report file, run `report_power -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%)
```

Optimize leakage power.

```
optPower
```

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: `optPower` 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 `optPower` without any options.

After running the `optPower` command, create a new leakage power report to view results.

```
report_power -leakage -outfile fileName
```

Optimizing Leakage Power While Running optDesign (Recommended)

You can optimize non-critical path logic for leakage power by using the `setOptMode` command in the following ways:

- If leakage power optimization is a second priority after timing convergence, then use `setOptMode -powerEffort low` after `place_design` along with the default setting of `-leakageToDynamicRatio 1.0`. The leakage power optimization is automatically done by `optDesign`.

- If leakage power optimization is as critical as timing convergence, then use `setOptMode -powerEffort high` after `place_design` along with the default setting of `-leakageToDynamicRatio 1.0`. The leakage power optimization is done by each `optDesign` step in high-effort mode.

Note: When `setOptMode -powerEffort` is set to `low` or `high`, the hold fixing steps ensure that no low-voltage threshold gates are inserted. Only high-voltage threshold gates are used to fix the hold-time violations.

Substrate Biasing

Substrate biasing is another technique for reducing leakage power. Changing the body voltage of the field effect transistor (FET) affects both the threshold voltage and the static leakage current.

To bias the substrate, insert biasing cells into a region of the design. In Innovus, you can do this in either of two ways:

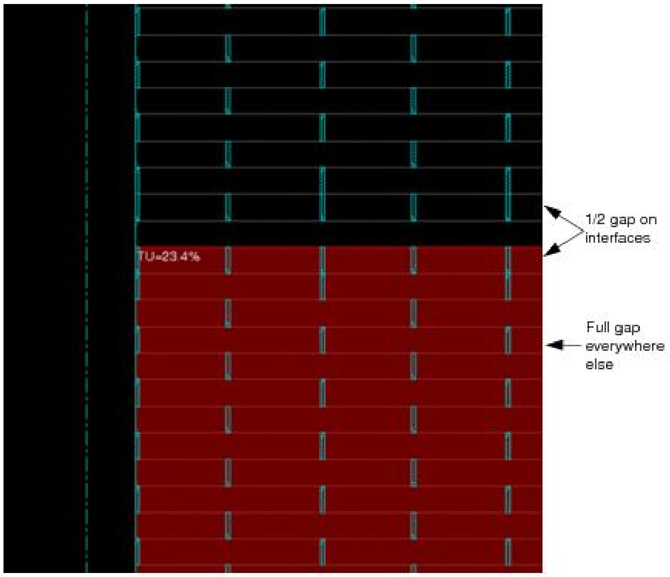
- Use the `addWellTap` command to add bias cells at regular intervals.

  ```
  addWellTap -maxGap
  ```

- Add well taps in a checkerboard configuration; for example,

  ```
  addWellTap -cellFILL1 -maxGap 20 -checkerBoard -fixedGap
  addWellTap -cellFILL2 -maxGap 20 -powerDomain PD -checkerBoard -fixedGap
  ```

These commands produce results such as those shown in the following figure:

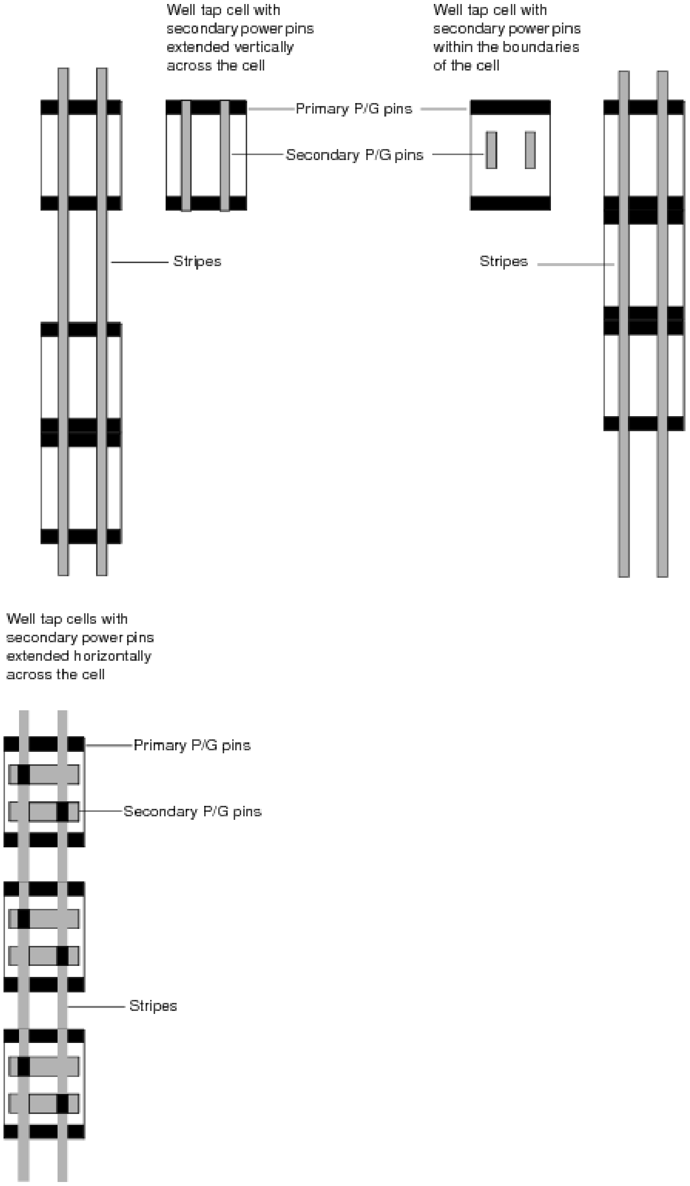


For well tap cells, you must add stripes to connect the secondary power/ground pins in the vertical or horizontal direction.

1. Select *Floorplan - Custom Power Planning - Add Stripes*.
2. On the Basic page, select *Over P/G pins*.
3. Click on *Master Name*.
4. Type the master name of the standard cell.
5. Select *Pin Layer*.

The top layer is the default.

The following figures show how well tap cells are connected. The software connects the secondary power/ground pins vertically or horizontally to the nearest secondary power/ground pins, regardless of whether the pins extend fully across the cell.


Well tap cell with secondary power pins extended vertically across the cell
Well tap cell with secondary power pins within the boundaries of the cell
Primary P/G pins
Secondary P/G pins
Stripes
Stripes
Well tap cells with secondary power pins extended horizontally across the cell
Primary P/G pins
Secondary P/G pins
Stripes

Power Shutdown Techniques

Power shutdown is a coarse-grain methodology for performing power gating. This technique shuts off a specific power domain under certain conditions. There are two styles of this methodology:

- Ring style: All switches are inserted outside the domain.
- Column style: All switches are inserted inside the power domain.

Data Preparation

For ring-style switch insertion, prepare the data as follows:

- Assign CLASS RING to the power switch cell in the LEF file (SYMMETRY X Y R90) . This is recommended, but not required for switch cells. No SITE information is required.
- Ensure that there are enable nets to drive the buffer inside of the power switch cell, and acknowledge nets to exit the power switch cell. These nets are used as input to the `-enableNetIn` and `-enableNetOut` options to `addPowerSwitch`.
- Specify the power/ground net and pin connects of the power switch cell.

For column-style switch insertion, prepare the data as follows:

- Assign CLASS CORE and the correct SITE definition for the switch cell in the LEF file.
- Specify the power/ground net and pin connections of the power switch cell.
- Specify the distance between the columns and switches in microns (horizontal pitch value).

For ring style, you need to know the following:

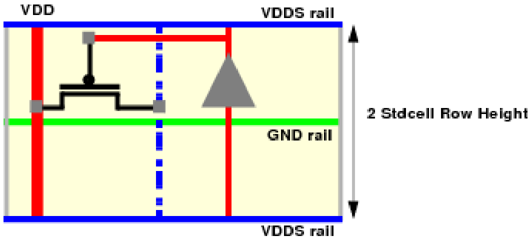
- For power planning, to ensure that the power stripes connect to the power switch cell
- NanoRoute connects enable signals. Abutment depends on the physical layout of the power switch cell.

For column style, you need to know the following:

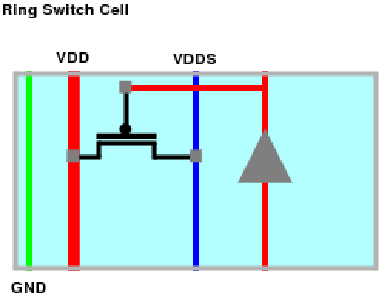
- In the `addPowerSwitch` command, the `-enableNetOut` option can only specify one net name, which will be the net base name. The tool adds the suffix `_`*`columnNumber`*.
- The dimension of the power switch cells must be an integer multiple of a single-height standard cell.

Buffer Styles

The following figure shows column switch cell, which contains a buffer. This cell has a height of two times the standard cell height.

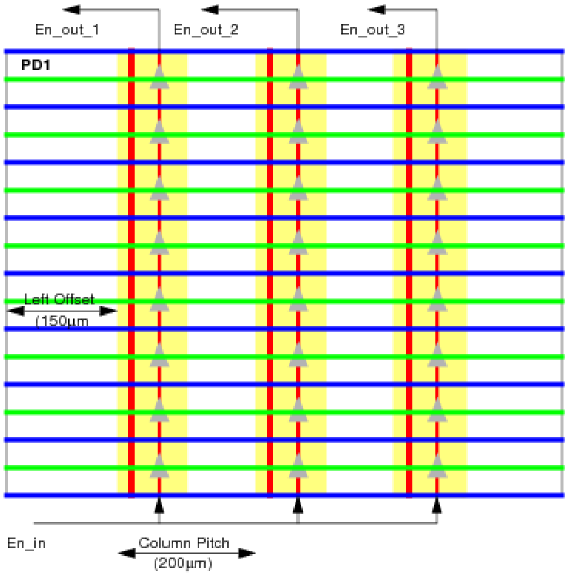


The following figure shows a ring switch cell, which contains a buffer. The cell has the same height as a standard cell. Ring switch cells can also contain two buffers with different directions.



Adding Column Switches

The column switch methodology adds power switches entirely within the power domain. The following figure shows an example of column switches within a power domain:



To add column switches, use the following command:

```
addPowerSwitch -column
```

The following parameters are required:

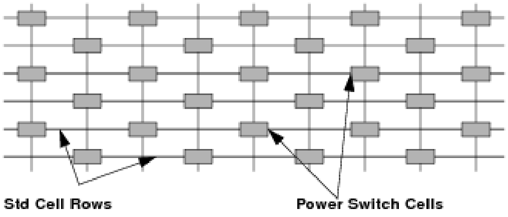
```
-powerDomain
-enablePinIn
-enablePinOut
-enableNetIn
-enableNetOut
-globalSwitchCellName
```

Optionally, you can specify the following:

- Offsets from the top, bottom, right, and/or left side of the power domain.
- Area in which the tool can place switches.
- Power/ground pin connections.
- Many other options.

Instead of using the text command, you could use the menu command as follows:

- From the main Innovus window, choose *Power - Multiple Supply Voltage - Power Switch Insertion*, then click on the *Column* button.
- With the text command, you can place the switches in a checkerboard pattern as follows:
  ```
  addPowerSwitch -column -checkerBoard
  ```



This command allows you to reserve space for other uses or reduce the number of switches, for example, and possibly reduce leakage power.

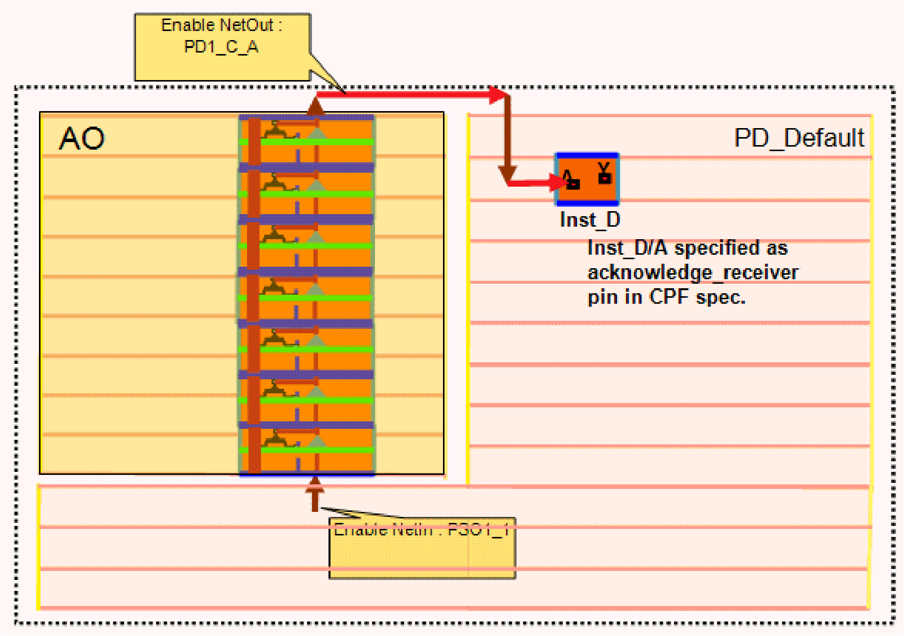
Attaching the Acknowledge Receiver Pin

In CPF, you can specify an input pin that must be connected to an output pin of the last power switch in the chain. This information is specified in CPF as follows:

```
update_power_switch_rule -name string
-acknowledge_receiver pin ...
```

The `addPowerSwitch` command can connect the output pin of the last switch cell to the acknowledge pin specified by `update_power_switch_rule`.

The following figure shows -`enableNetOut PD1_C_A` connected to `Inst_D` as specified in CPF:



Example

In the CPF file:

```
create_power_switch_rule -name sw1 -domain PD1 -external_power_net VDDH
update_power_switch_rule -name sw1 -cells COLUMN_SW -acknowledge_receiver Inst_D/A
```

Power switch insertion command:

```
addPowerSwitch -column -powerDomain PD1 -enablePinIn SWIN -enablePinOut SWOUT -enableNetIn PSO1_1
-globalSwitchCellName COLUMN_SW -leftOffset 3 -horizontalPitch 100
```

Verilog file after `addPowerSwitch` is run:

```
BUFXH Inst_D (.Y(switch_out), .A(PD1_C_A));
...
COLUMN_SW pso1_PD1_1_COLUMN_SW_9_52_3 (.SWOUT(PD1_C_A),
.SWIN(psoPSI_PD1_EnNet_1_3_9_49_0));
```

`Inst_D/A` is connected to the `PD1_C_A` net and is connected to the `SWOUT` pin of the last switch.

ⓘ Do not specify `-enableNetOut` because this setting interferes with and overrides the `-acknowledge_receiver` specification.

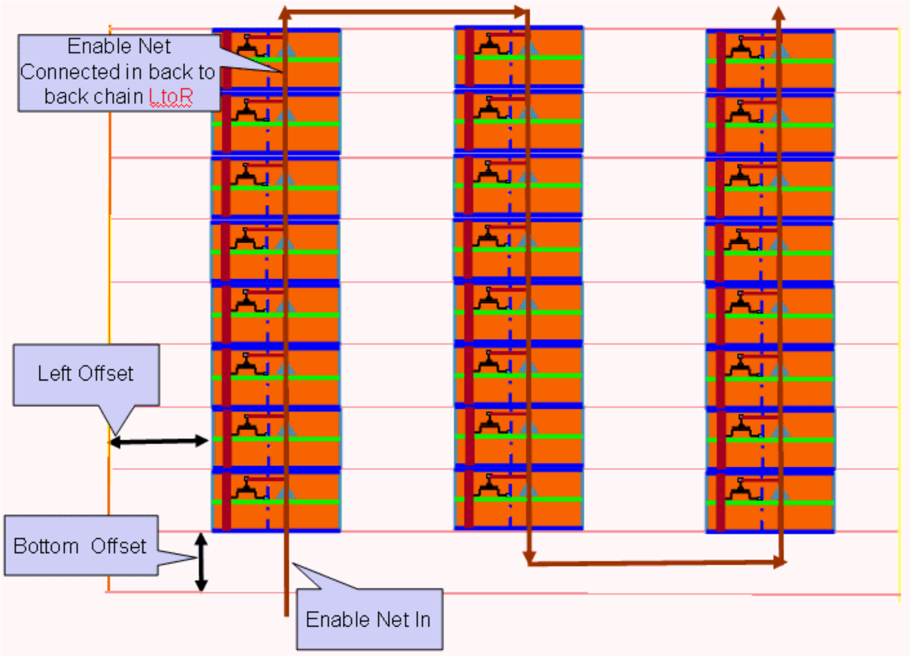
Enable Chaining

By default, the -`enableNetIn` is connected to the bottom of each column and the -`enableNetOut` exits from the top of each column, in parallel. The following commands let you create a columns with daisy-chain enables:

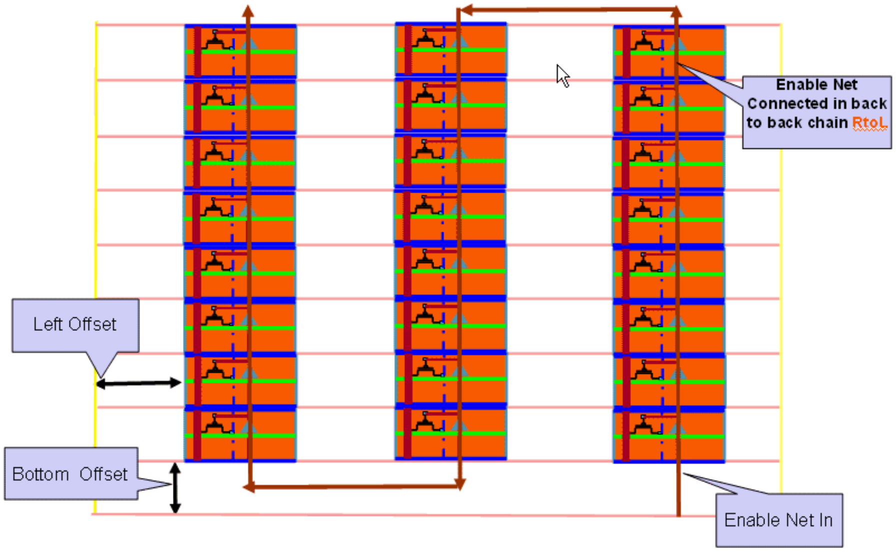
- `-backToBackChain`

  Connects the -`enableNetOut` at the top of a column to the -`enableNetIn` at the top of the next column, and connects the -`enableNetOut` at the bottom of a column to the -`enableNetIn` at the bottom of the next column.

  The following figure shows `-backToBackChain` with the `LtoR` (left-to-right) option:



The following figure shows `-backToBackChain` with the `RtoL` (right-to-left) option:



Example:

```
addPowerSwitch \
-column \
-powerDomain DSP \
-switchModuleInstance dummy_dsp_1 \
-enablePinIn {NSLEEPIN2} \
-enablePinOut {NSLEEPOUT2} \
-enableNetIn {UNCONNECTED249} \
-globalSwitchCellName HDRDIHVTD2 \
-enableNetOut {power_out_ack} \
-leftOffset 15.0 \
-bottomOffset 0.0
-horizontalPitch 150.0 \
-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`:

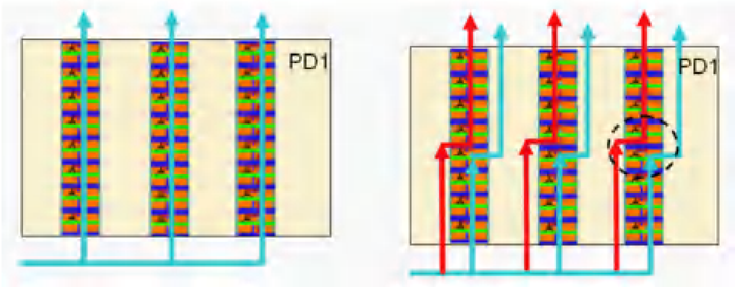
```
addPowerSwitch \
-column \
-powerDomain DSP \
-switchModuleInstance dummy_dsp_1 \
-enablePinIn {NSLEEPIN2 NSLEEPIN1} \
-enablePinOut {NSLEEPOUT2 NSLEEPOUT1} \
-enableNetIn {UNCONNECTED249} \
-globalSwitchCellName HDRDIHVTD2 \
-enableNetOut {power_out_ack} \
-leftOffset 15.0 \
-bottomOffset 0.0 \
-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 -powerDomain PD1 -maxChainDepth 4



In the first part of the figure, no -maxChainDepth specified. In the second part, -maxChainDepth = 4, this stops the chain at four switches.

Synthesizing Acknowledge Trees

The Innovus tool can automatically create acknowledge trees that you would otherwise build manually. The acknowledge tree collects enable signals exiting the power domain and funnels them to an acknowledge receiver pin.

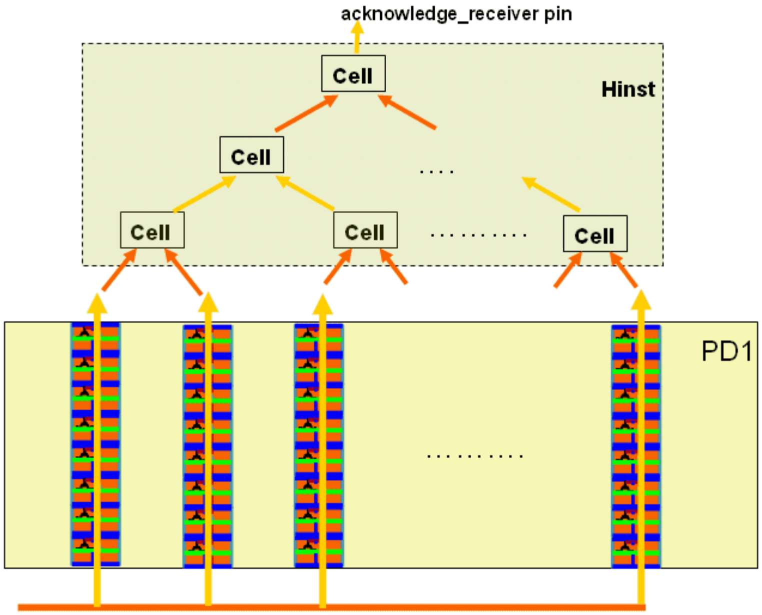
Use the following parameters to create the acknowledge tree:

`-acknowledgeTreeCell`

`-acknowledgeTreeHierInstance`

If you do not specify `-acknowledgeTreeHierInstance`, the tool places the cells in the top module.

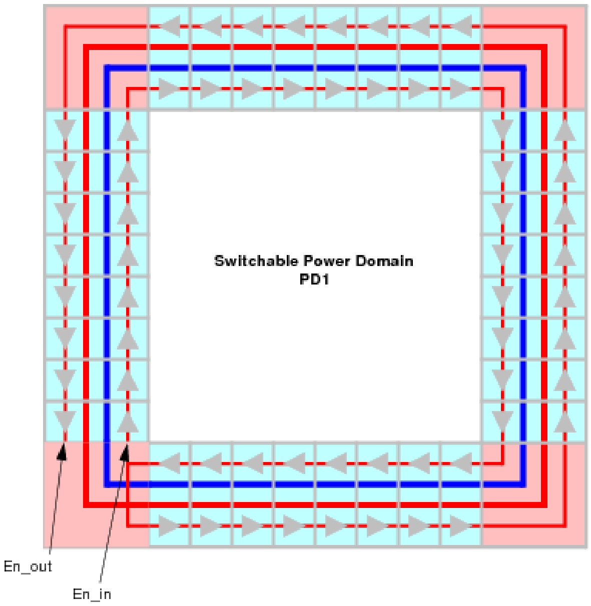
The following figure shows an acknowledge tree built from cells of type Cell, placed in hierarchical instance HInst.



`enableNetIn`

Adding Power Switch Rings

You can add power switches in a ring entirely outside the boundary of the power domain as shown below.



Note:
Switch cell has
2 built-in buffers
with different
directions

In this figure, the switches abut and connect to the next switch. Because the switches in this example contain two built-in buffers with different directions, the enable net loops around the inner side of the ring, connects at the corner, and loops back around to where it becomes the enable net out.

This technique is useful when the power domain is a pre-designed macro.

To create a power switch ring, use the following command:

```
addPowerSwitch -ring
```

The following parameters are required:

```
-powerDomain
-enablePinIn
-enablePinOut
-enableNetIn
-enableNetOut
```

By default, the tool distributes cells evenly in the ring. To stack cells instead, use the following command:

```
addPowerSwitch -ring -distribute 0
```

Instead of using the text command, you could use the menu command as follows:
From the main Innovus window, select *Power- Multiple Supply Voltage - Add Power Switch*, then click on the Ring tab. Select Distribute Switches on the Switch Cell Count form.

With ring options, you have many ways of configuring the switch ring. Among many possibilities, you can do the following:

- Control how switches are distributed around the ring
- Choose the sides on which you want to add switch cells
- Specify the breaker, buffer, filler, switch, and corner cells you want to use on specified sides
- Specify the distance between the power domain and each side of the ring
- Choose the orientation of cells on specified sides
- Arrange the buffer, breaker, filler, and switch cells in a pattern

Creating Patterns

When you create rings, you can specify a pattern that customizes switch placement. If you do not specify a pattern, the software adds switches evenly around the power domain.

The following command shows a pattern of cells that repeats on all sides:

```
addPowerSwitch -ring \
-powerDomain TDSP2 \
-enablePinIn {A0} -enablePinOut {Z0} \

enableNetIn swcontrol_2 -enableNetOut swack_2
-specifySideList {1 1 1 1 1 1 1 1}
-sideOffsetList {3 3 3 3 3 3 3 3 }
-globalSwitchCellName {{CDN_RING_SW S} {CDN_RING_SW_1 D} {CDS_RING_SW_2 G}}\
-cornerCellList CDN_RING_CORNER_UL \
-globalFillerCellName {{CDN_RING_FILLER F}}
-insideCornerCellList CDN_RING_CORNER_InCell \
-globalPattern {S S D D G G F}
```

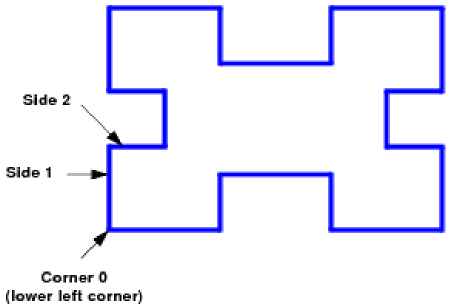
In this example, the command adds power switches in the following pattern:

```
CDN_RING_SW CDN_RING_SW CDN_RING_SW_1 CDN_RING_SW_1 CDN_RING_SW_2 CND_RING_SW_2 CDN_RING_FILLER
```

The command repeats the pattern on each side. If you want to continue the pattern on the next side or edge, use the `-continuePattern` parameter.

Ring Conventions

The Innovus software supports rectilinear power domains, such as the 20-sided power domain shown below:



The default side/corner numbering is clockwise from the starting corner (Corner 0), which is always the lower left corner of the power domain.

Corner numbering starts with 0. Side numbering starts with 1.

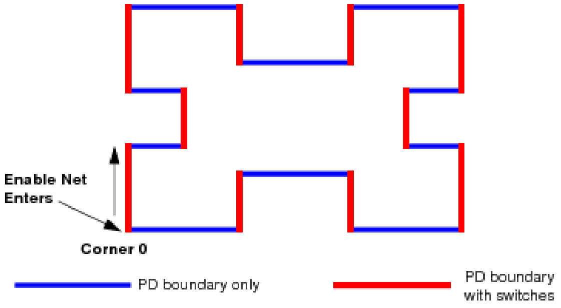
Specifying Sides in a Switch Ring

Use `addPowerSwitch -specifySideList` to add switches around a power domain of any number of sides.

Each value provided in the `-specifySideList` 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 `-specifySideList {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 -specifySideList {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}
```

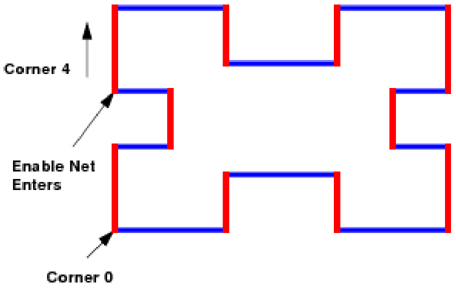


Starting the Enable Chain at a Different Corner

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.

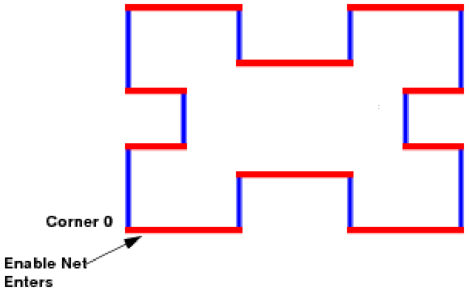
```
addPowerSwitch -ring -specifySideList {1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0} -startEnableChainAtCorner 4
```



Counter-Clockwise

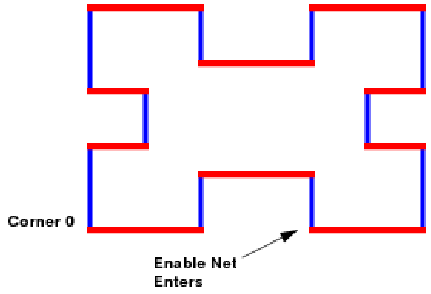
The `-counterclockwise` option reverses the corner/side numbering from Corner 0. What was side 20 in the previous example becomes side 1 in this example.

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



In the following example, side/corner numbering is counterclockwise, and the enable net enters at Corner 4. Note how location of the specified corner differs from the location of the corner specified without the `-counterclockwise` parameter.

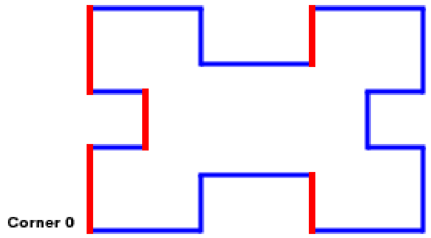
```
addPowerSwitch -ring -specifySideList {1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0} -counterclockwise -startEnableChainAtCorner 4
```



Left Sides

The following command adds switches only to the left sides of the power domain: Sides 1, 3, 5, 9, 17.

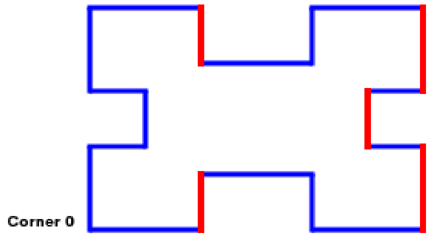
```
addPowerSwitch -ring -leftSide 1
```



Right Sides

The following command adds switches only to the right sides of the power domain: Sides 7, 11, 13, 15, and 19.

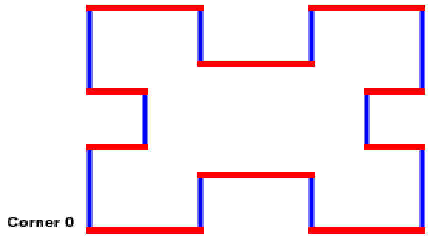
```
addPowerSwitch -ring -rightSide 1
```



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.

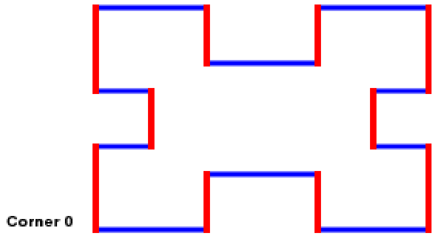
```
addPowerSwitch -ring -horizontalSide 1
```



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.

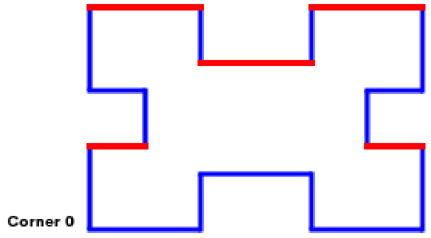
```
addPowerSwitch -ring -verticalSide 1
```



Top Sides

The following command adds switches only to the top sides of the power domain: Sides 2, 6, 8, 10, and 14.

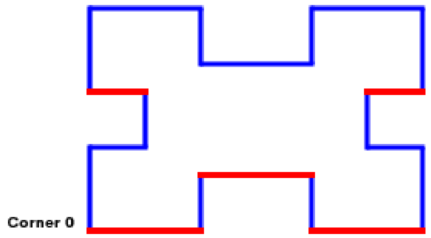
```
addPowerSwitch -ring -topSide 1
```



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 -bottomSide 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_bottom`
- `-startOffsetTop_top`
- `-startOffsetLeft_left`
- `-startOffsetRight_right`
- `-startOffsetHorizontal_horizontal`
- `-startOffsetVertical_vertical`
- `-sideStartOffsetListside__list {`*`value`*`...}`
- `-endOffset`
- `-endOffsetBottom_bottom`
- `-endOffsetTop_top`
- `-endOffsetLeft_left`
- `-endOffsetRight_right`
- `-endOffsetHorizontal_horizontal`
- `-endOffsetVertical_vertical`
- `-sideEndOffsetListside__list {`*`value`*`...}`
- `-forceOffset [0|1]`

- `-switchPitch`
- `-switchPitchBottom_bottom`
- `-switchPitchHorizontal_horizontal`
- `-switchPitchLeft_left`
- `-switchPitchRight_right`
- `-switchPitchTop_top`
- `-switchPitchVertical_vertical`
- `-switchPitchSideList_side_list {`*`value`*`...}`

Forcing Offsets

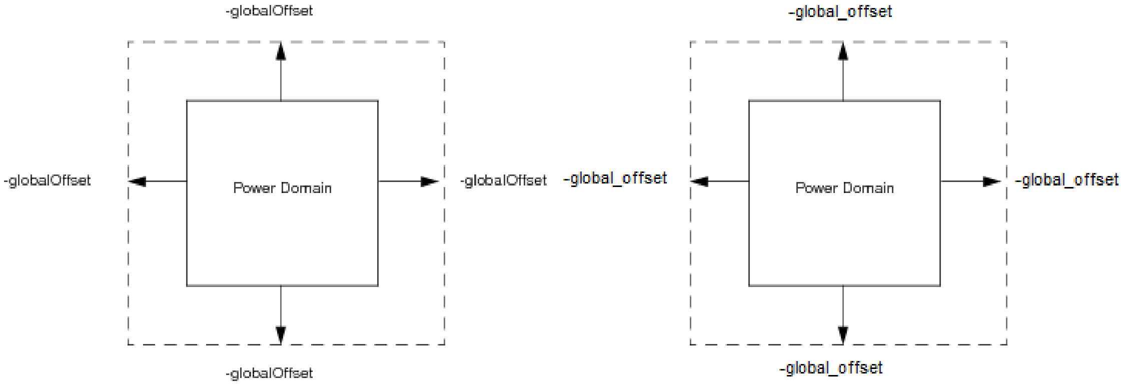
Offsets you specify are the *minimum* offsets you require to complete the power switch ring. Resulting offsets could be quite different from the ones you specify. To force the tool to comply as much as possible to the offset values, specify `-forceOffset 1`.

Setting the Global Offset

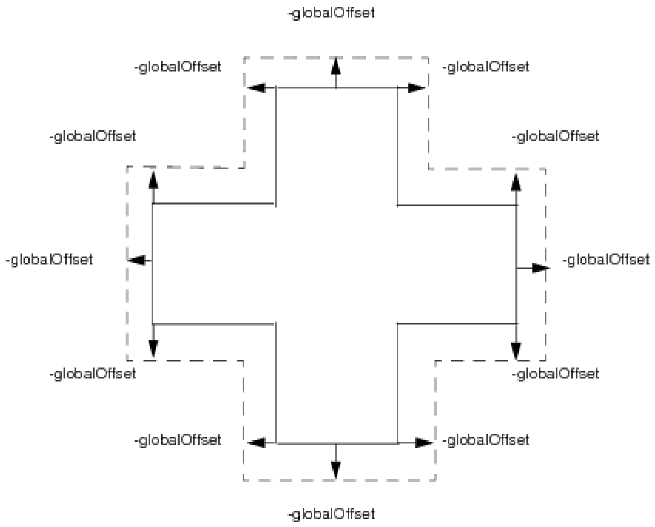
Instead of placing switches against the power domain boundaries, you can place them away from the boundaries by the specified distances. To specify the same offset for all sides, use the `-globalOffset` parameter.

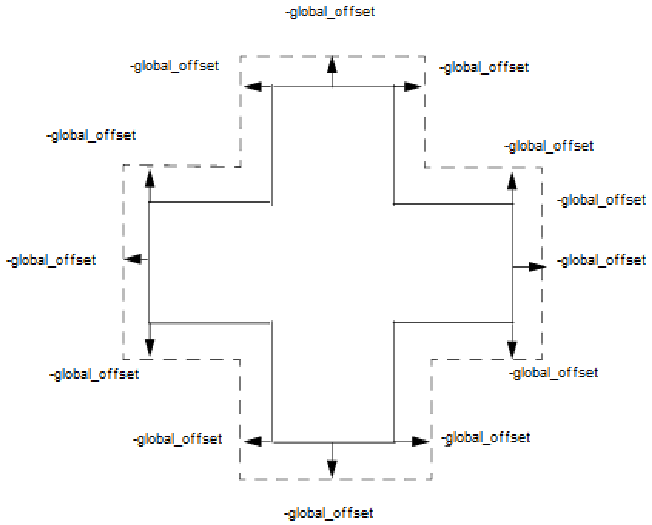
The default offset value is 0 (no offset).

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:

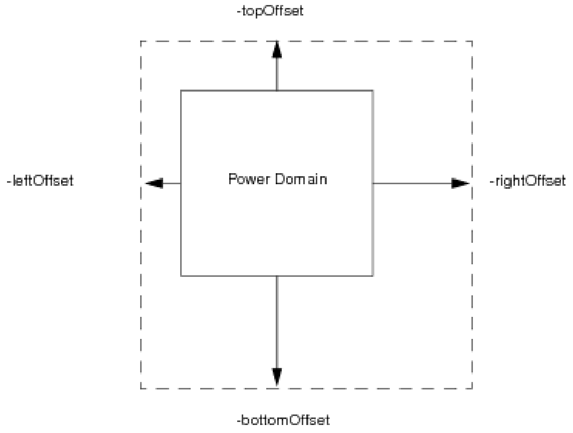


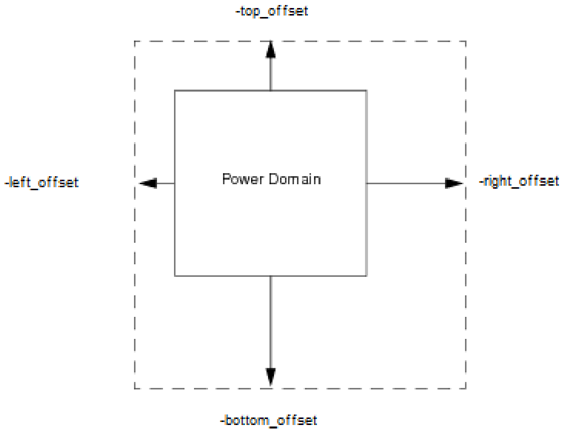


Setting Different Offsets for Different Sides

To specify different offsets for different sides, use the `-leftOffset`, `-rightOffset`, `-bottomOffset`, and/or `-topOffset` parameters.

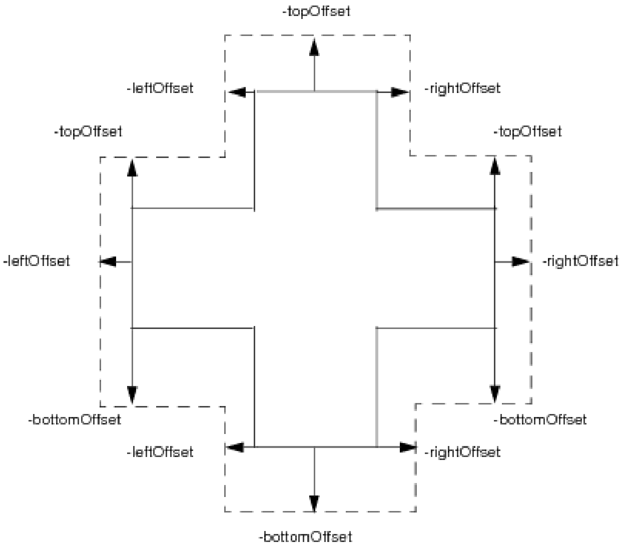
The following example shows a different offset for each side if `-forceOffset` is specified:





- 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**
`-sideStartOffsetListside__list`	`-sideEndOffsetListside__list`	`-switchPitchSideList_side_list`	Specified side(s)
`-startOffsetBottom_bottom`	`-endOffsetBottom_bottom`	`-switchPitchBottom_bottom`	Bottom side(s)
`-startOffsetTop_top`	`-endOffsetTop_top`	`-switchPitchTop_top`	Top side(s)
`-startOffsetRight_right`	`-endOffsetRight_right`	`-switchPitchRight_right`	Right side(s)
`-startOffsetLeft_left`	`-endOffsetLeft_left`	`-switchPitchLeft_left`	Left side(s)

`-startOffsetHorizontal_horizontal`	`-endOffsetHorizontal_horizontal`	`-switchPitchHorizontal_horizontal`	Horizontal side(s)
`-startOffsetVertical_vertical`	`-endOffsetVertical_vertical`	`-switchPitchVertical_vertical`	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_right` are specified, the tool uses the `-startOffsetRight_right` 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_top 1 -endOffsetLeft_left -2 -switchPitch 3`
  - For example, for a rectilinear power domain:
    `-startOffset -1 -switchPitchSideList_side_list {3, 4, 3, 0, 0, 0, 0, 0, 0, 0}`
- You can combine any side-specific parameters.
  - For example, for a rectangular power domain:
    `-startOffsetLeft_left 1 -startOffsetRight_right 2 -endOffsetTop_top -2 -switchPitch 3`
  - For example, for a rectilinear power domain:
    `-startOffsetRight_right -1 -switchPitchSideList_side_list {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.

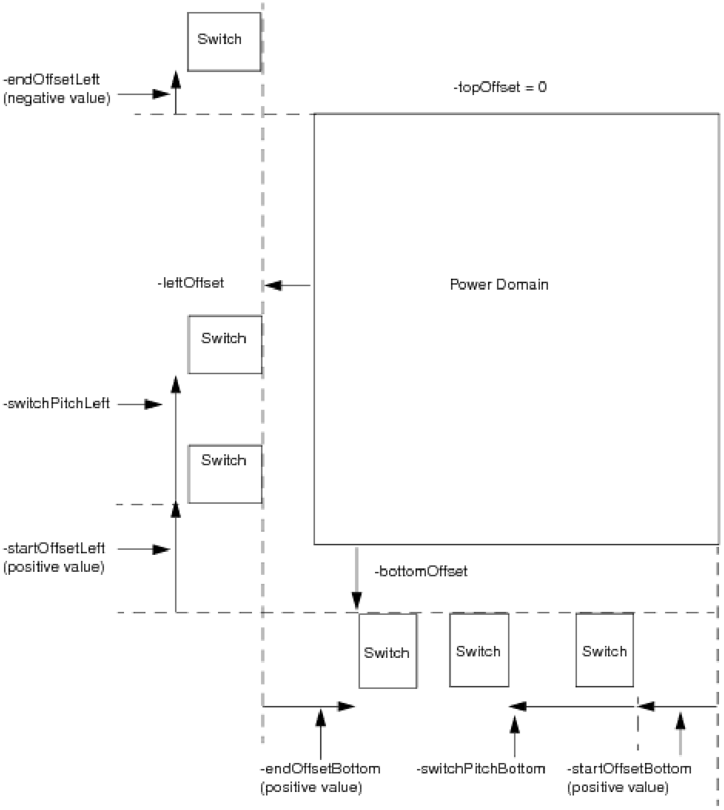
Example of Offsets

The following example shows a clockwise switch ring with the following parameters:

- `-leftOffset`
- `-bottomOffset`
- `-startOffsetLeft_left`
- `-startOffsetBottom_bottom`
- `-endOffsetLeft_left`
- `-endOffsetBottom_bottom`
- `-switchPitchLeft_left`

- `-switchPitchBottom_bottom`

Different switch pitches are specified for the left and bottom sides, and the start and end offsets are positive or negative.



- The first switch on the left side is placed a distance `-startOffsetLeft_left` from the edge of the `-bottomOffset` boundary.
- The second switch on the left side is placed a distance `-switchPitchLeft_left` from the bottom edge of the first switch on the side.
- The last switch on the left side is placed a distance `-endOffsetLeft_left` *above* the `-topOffset` because the `-endOffsetLeft_left` value is negative. In this case, the `-topOffset` is 0, so the last switch is placed above the top power domain boundary.
- The first switch on the bottom side is placed a distance `-startOffsetBottom_bottom` 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_bottom` from the right edge of the first switch on the side.
- The last switch on the bottom side is placed `-endOffsetBottom_bottom` to the *right* of the `-leftOffset` edge. The `-endOffsetBottom_bottom` parameter has a positive value.

Power Switch Prototyping

Innovus provides the facility for prototyping power switches. Power switch prototyping enables you to:

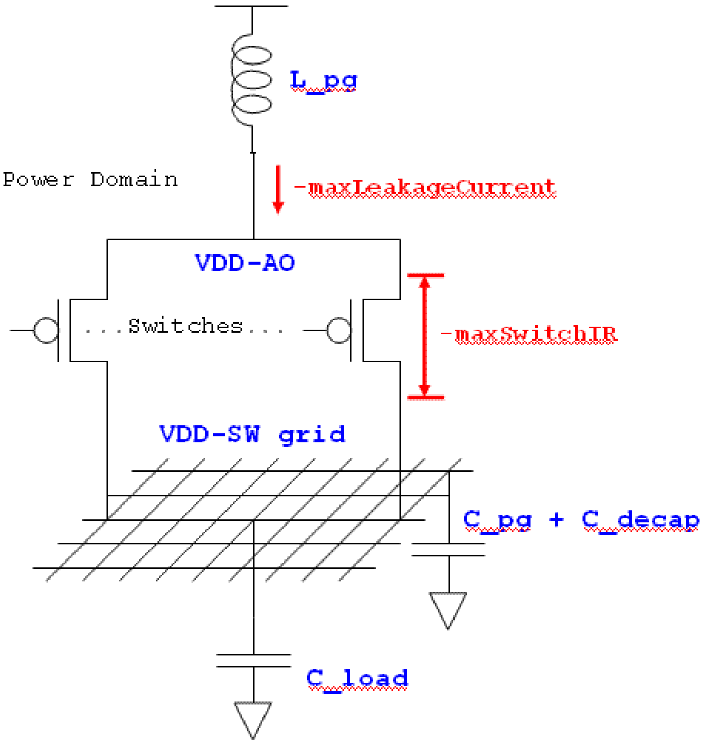
- Find optimal number of switches
- Sweep (Enumerate) number of switches
- Given the ramp up time, find optimal number of ramp up switches
- Given number of ramp up switches, find ramp up time
- Sweep (Enumerate) number of ramp up switches
- Find number of ramp up switches constrained by maximum ramp up current
- Find best switch delay constrained by maximum ramp up current

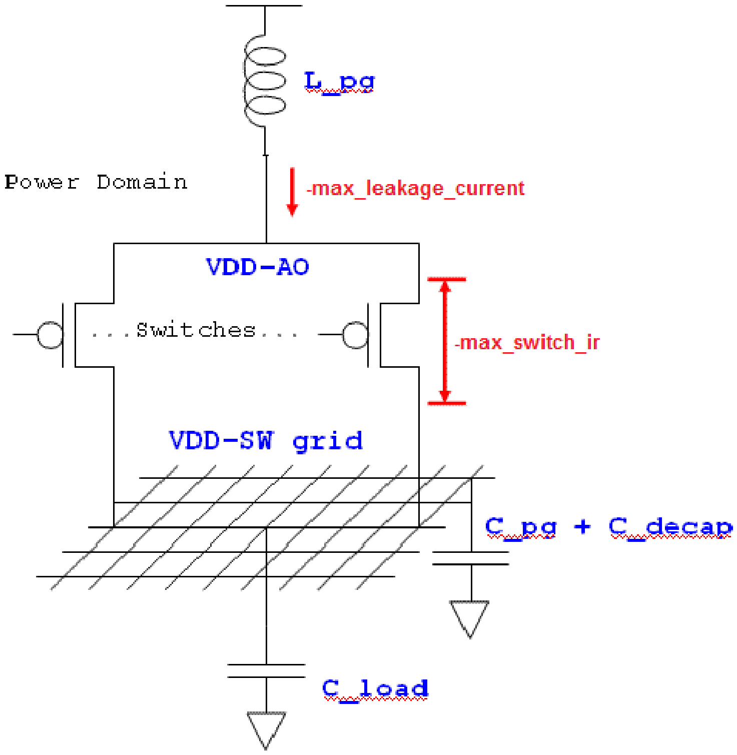
In this section we will cover:

- Power Domain Parameters and Specification
- Options Summary - Switch and Power Domain
- Options Summary - Prototyping Features
- Chain Style Impacts on Ramp Up Time and Rush Current
- Prototyping Results

Power Domain Parameters and Specification

The following diagram and description outlines the power domain parameters and specifications:





Attribute for the domain power:

- Total power: 1.0 mW
- Domain voltage: 1.0 V
- Max switch IR tolerance: 10 mV
- Max domain leakage current allowed: 2 uA
- Domain capacitance: 1 uF
- Package inductance: 0.1 nH

Attributes for the switch cells:

- Idsat: 1 mA
- Ron: 800 Ohm
- Ileak: 10 nA
- Switch buffer delay: 100ps

Options Summary - Switch and Power Domain

The following is summary of switch and power domains:

Switch Cell Characterization

`-Idsat`

`-Ileak`

`-rOn`

`-BufferDelay`

`-CellEM`

`-readPowerSwitchCell` *`filename`*

Power Domain Specification

`-totalPower`

`-voltage`

`-maxSwitchIR`

`-maxLeakageCurrent`

`-loadCapacitance`

`-pgCapacitance`

`-pgInductance`

`-rampUpRailVoltagePercent` *`(0+..100-)`*

`-numberSimultaneousRampUpChain` *`num`*

Options Summary - Prototyping Features

The following is the summary of prototyping features:

- To find optimal number of switches enter:

  `-prototypeNumberSwitches` *`0|1`*

- To find Sweep/Enumerate number of switches enter:

  `-prototypeSweepSwitchNumber` *`min max incremental`*

- To find Sweep/Enumerate number of ramp up switches enter:

  `-prototypeSweepChainDepth` *`min max incremental`*

- To find min/max number of ramp up switches given ramp up time enter:

  `-prototypeChainDepth` *`0|1`*

  `-prototypeMinChainDepth` *`0|1`*

  `-prototypeMaxChainDepth` *`0|1`*

  `-rampUpTime` *`min max`*

- To find ramp up time given number of ramp up switches enter:

  `-rampUpTime` *`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:

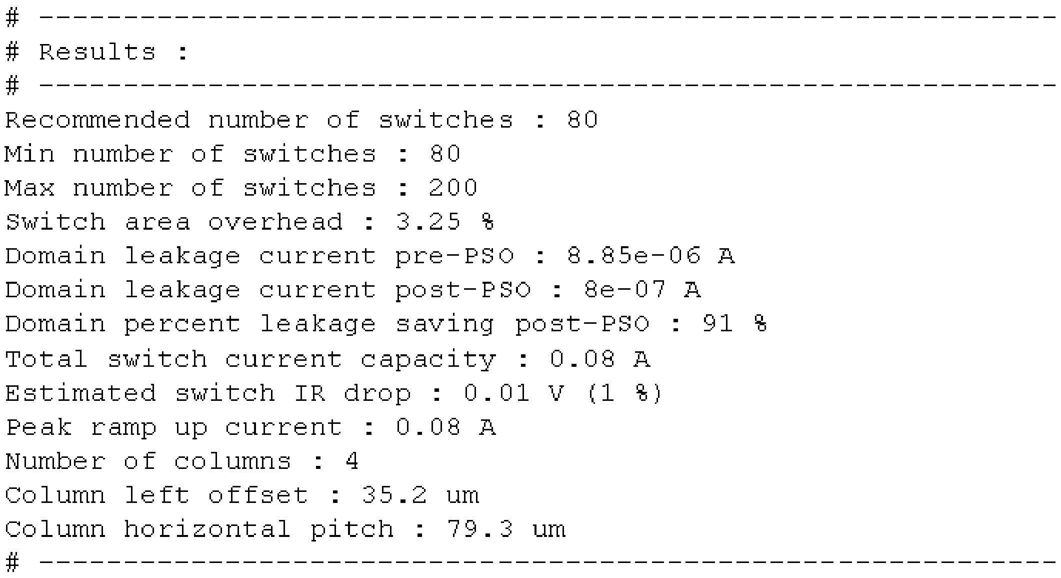
- Optimal Switch Results
- Switch Number Enumeration Results
- Ramp Up Switch Enumeration Results
- Number of Switches Given Current Maximum Ramp Up
- Switch Delay Given Current Maximum Ramp Up Current
- Ramp Up Time

Optimal Switch Results

To find optimal switch results, enter:

```
addPowerSwitch -column -powerDomain PD -globalSwitchCellName HEADER \
-prototypeNumberSwitches 1
```

The following type of result is displayed:



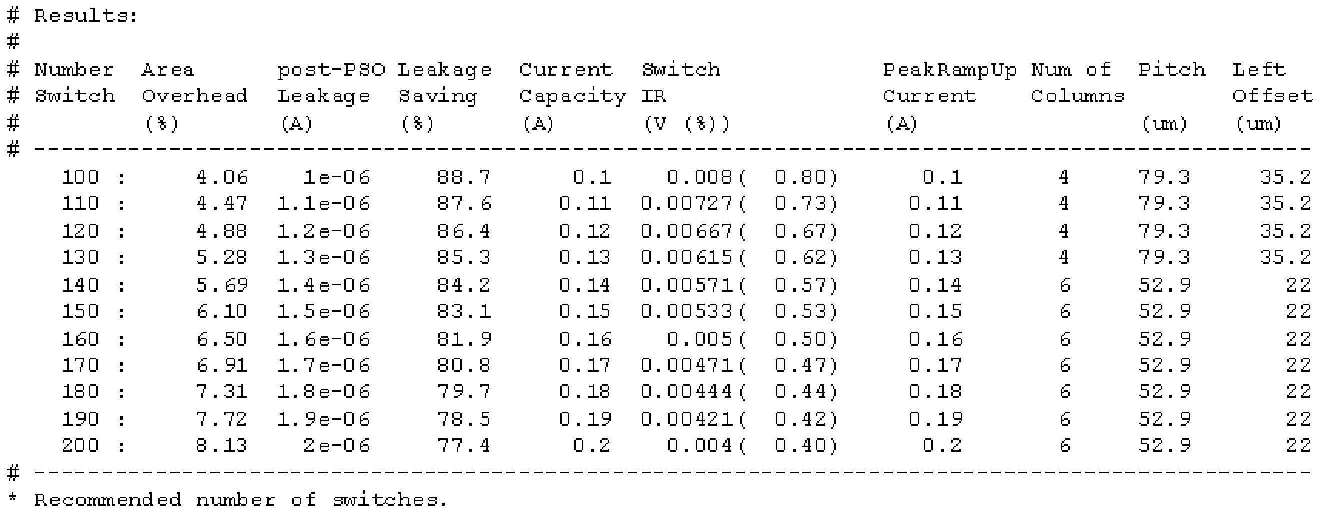
```
# ------------------------------------------------------------
# Results :
# ------------------------------------------------------------
Recommended number of switches : 80
Min number of switches : 80
Max number of switches : 200
Switch area overhead : 3.25 %
Domain leakage current pre-PSO : 8.85e-06 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:



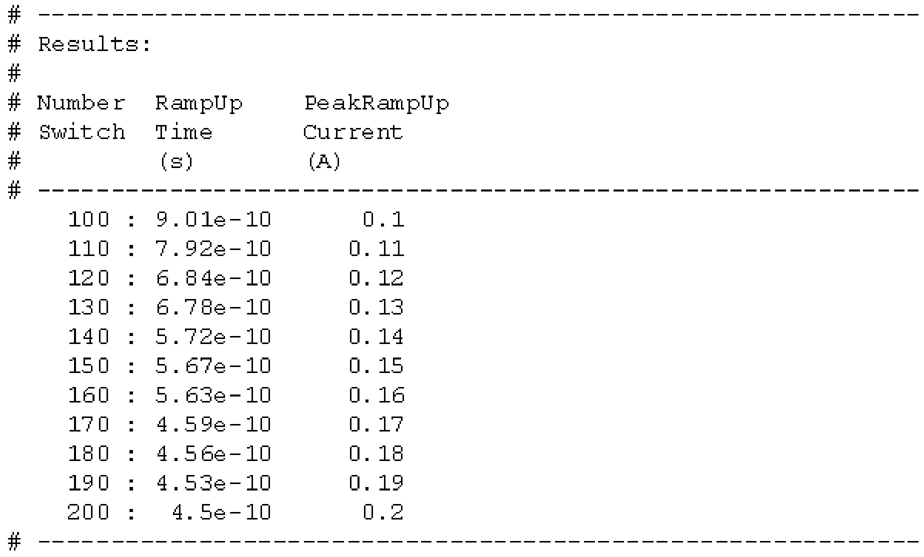
```
# Results:
#
# Number  Area      post-PSO Leakage  Current  Switch           PeakRampUp Num of  Pitch  Left
# Switch  Overhead  Leakage  Saving   Capacity IR               Current    Columns        Offset
#         (%)       (A)      (%)      (A)      (V (%))          (A)                (um)   (um)
# ---------------------------------------------------------------------------------------------
     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:



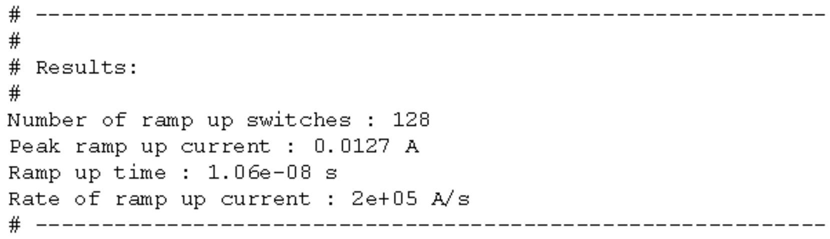
```
# ------------------------------------------------------------
# Results:
#
# Number  RampUp     PeakRampUp
# Switch  Time       Current
#         (s)        (A)
# ------------------------------------------------------------
    100 : 9.01e-10      0.1
    110 : 7.92e-10     0.11
    120 : 6.84e-10     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:



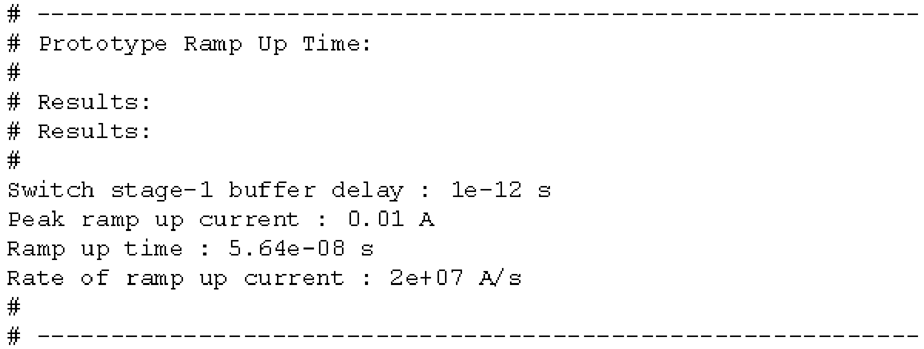
```
# ------------------------------------------------------------
#
# Results:
#
Number of ramp up switches : 128
Peak ramp up current : 0.0127 A
Ramp up time : 1.06e-08 s
Rate of ramp up current : 2e+05 A/s
# ------------------------------------------------------------
```

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:



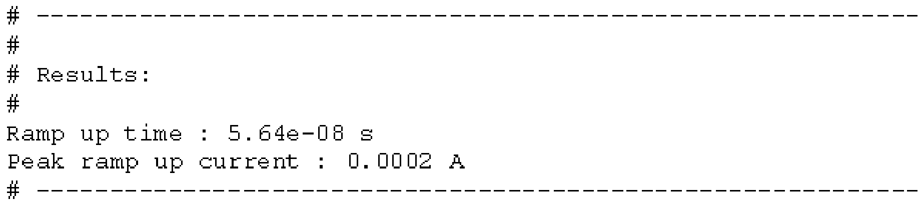
```
# -----------------------------------------------------------
# Prototype Ramp Up Time:
#
# Results:
# Results:
#
Switch stage-1 buffer delay : 1e-12 s
Peak ramp up current : 0.01 A
Ramp up time : 5.64e-08 s
Rate of ramp up current : 2e+07 A/s
#
# -----------------------------------------------------------
```

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

After floorplanning, place the cells in the design. Placement considers the modules that were placed during floorplanning and takes into account the hierarchy and connectivity of the design. It honors floorplanning constraints, including guides, regions, and fences. For descriptions of the constraint types, see "Module Constraint Types" section in the Floorplanning the Design chapter of the *Innovus User Guide.*

Placement also follows legalization rules, such as cells cannot overlap each other and takes into account short and spacing DRC rules required by routing.

After the cells are legally placed, next step is preCTS optimization.

Loading a Design

Load a design by using the `restoreDesign` command or the `init_design` command

For more information, see

- Design Import and Export in Stylus chapter of the Innovus Stylus Common UI User Guide.
    - `restoreDesign`
    - `init_design`

Preparing for Placement

Before placement, run the following commands and correct problems. Some of these commands generate reports that you can use as a baseline for comparisons later in the flow.

- Run `checkDesign` to check the integrity of the library and design data. For more information, see `checkDesign` in the "Import and Export Commands" chapter of the Innovus Text Command Reference.
- Run `timeDesign -prePlace` to get an idea of Zero Wire Load timing of the design. For more information, see `timeDesign` in the "Timing Analysis (Common Timing Engine) Commands" chapter of the *Innovus Text Command Reference*.
- Run to create blockages (this is usually done during floorplanning). For more information see "Guiding Placement With Blockages" or in the "Floorplan Commands" chapter of the *Innovus Text Command Reference*.
- Use one of the following methods to place and fix hard blocks.
- Run . For information, see in the "Floorplan Commands" chapter of the *Innovus Text Command Reference*.
    - Manually place and fix hard blocks.
- Run `checkPlace` (or `checkDesign -place`) or use the Violation Browser to check for violations caused by preplaced cells or blocks. For more information, see `checkPlace` in the "Placement Commands" chapter of the *Innovus Text Command Reference*.

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

Guiding Placement With Blockages

Use placement blockages to help guide placement.

Create the blockages during the floorplanning session by using the following command:

```
createPlaceBlockage
```

After creating a blockage, assign an attribute to it by using the Attribute Editor.

Alternatively, you can create placement blockages using the *Set Placement Blockage Options* form.

For more information, see Create Placement Blockage in the "Floorplan Menu" chapter of the *Innovus Menu Reference*.

A placement blockage has one of the following attributes:

Hard	The area cannot be used to place blocks or cells. By default, `createPlaceBlockage` creates blockages with this attribute.
Soft	Soft blockages accept: • any cell with <= 2 pins (buffers, inverters) • any cell with a CTS clock halo (regardless of pin count) - this includes clock logic and clock gates
Partial	Sets a percentage of the area that is available for placement. For example, a partial placement percentage of 75 percent means that up to 75 percent of placement density is allowed in the area. **Note:** A partial blockage of 100 percent behaves as no placement blockage.
Macro-Only	The area cannot be used to place blocks but can be used to place standard cells as a free area.

✓ If the design has routing congestion issues in the small channels between hard blocks, consider using the `setPlaceMode -place_global_auto_blockage_in_channel true` which automatically adds soft placement blockages in these areas. Although the blockages obstruct standard cells during placement step, they do not obstruct standard cells during optimization operations. Using soft blockages can help improve both timing and routability.

For more information, see

- `createPlaceBlockage` in the "Floorplan Commands" chapter of the *Innovus Text Command Reference*.

Placement Treatment of Preroutes

Placement treats preroutes the same way it treats routing blockages: It places standard cell instances at legal locations where there should not be any DRC violations against preroutes or routing blockages.

Typically, you use preroutes for special nets that are floorplanned (pre-designed) before placement, such as power, ground, and clock mesh nets, where you do not want any standard cells placed underneath. Instances placed next to power and ground stripes honor the design spacing rule. Instances placed next to routing blockage objects are set adjoined.

By default, the Innovus placement honors preroutes and routing blockages on *metal2* for a three-metal layer process and on *metal2* and *metal3* for a four or more metal layer process.

You can change this behavior by using the following command before running placement:

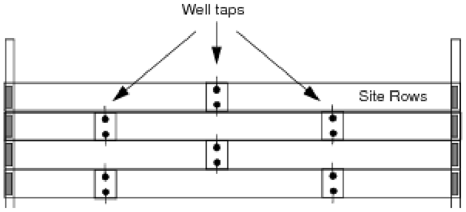
```
setPlaceMode -place_detail_preroute_as_obs true
```

For more information, see `setPlaceMode` in the "Placement Commands" chapter of the *Innovus Text Command Reference*.

Adding Well-Tap Cells

Well taps are physical-only filler cells that are required by some technology libraries to limit resistance between power or ground connections and wells of the substrate. Well-tap cells are placed in a preplaced status, so future placement commands do not move them.

The following diagram shows an example of well-tap cell placement. In this diagram, the cells are staggered in the site rows.



Add well taps after the floorplan is fixed and hard blocks are placed, but before placing standard cells.

Use one of the following methods to add well-tap cells:

- Add Well Tap Instances form
- `addWellTap` command

Controlling the Distance Between Well-Tap Cells

Use the `addWellTap -cellInterval` parameter to specify the maximum distance between well-tap cells in the same row.

- `-cellInterval` measures the distance from the center of one well-tap cell to the center of the next well-tap cell in the same row.

By default, the software always leaves a distance that is at least 45 percent of the specified maximum distance between well-tap cells in the same row. For example, if the specified maximum distance between same-row well-tap cells is 48.0 microns, the default minimum distance would be 21.6 microns.

Adding Well-Tap Cells to MSV Designs

In cases where there are different voltages in the same design, also known as a multi-voltage (MSV) design, specify the power domain in which to insert the well-tap cells by using the following command:

```
addWellTap -powerDomain
```

Deleting Well-Tap Cells

To remove added well-tap cells, use the Delete Instances form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes only well-tap cells that are completely contained within the area; it does not delete well-tap cells that cross the area boundary.

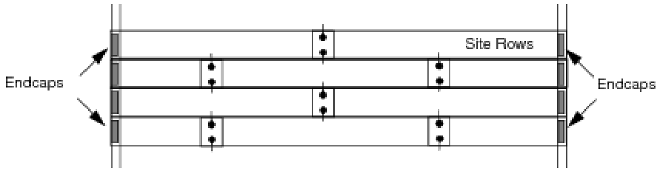
For more information see the following topics:

- `addWellTap` and `deleteFiller` in the "Placement Commands" chapter of the *Innovus Text Command Reference*.

Adding End-Cap Cells

End-cap cells are preplaced physical-only cells that are required to meet certain design rules. They are placed at the ends of the site rows, and are used in some technologies for power distribution. Besides of inserting the end-cap cells at the row ends, `addEndCap` also supports to place the end-cap cells at the top and bottom row to constitute an enclosed end-cap ring around the core area, placement blockage and hard macros. End-cap cells are placed in a preplaced status, so future placement commands do not move them. Add end-cap cells to the design before any other standard cells are placed, but after hard blocks are placed in the floorplan.

The following diagram shows an example of end-cap cell placement. The cells are placed at the ends of each site row.



To add end-cap cells, use the Add End Cap Instances form or the `addEndCap` command.

Adding End Cap Cells to MSV Designs

In cases where there are different voltages in the same design, specify the power domain in which to insert the end cap cells by using the following command:

```
addEndCap -powerDomain
```

Adding Different Kinds of End Cap Cells

You can add different kinds of end cap as per the site numbers between core or placement blockage or hard macro using the `setEndCapMode` command. There are two kinds of single-height end cap cells for left/right boundaries, for example, A type and B type. If the site number is odd, the end cap type on both sides of the core will be different. This means AB or BA end caps are placed on left/right boundaries. If the site number is even, the end cap type on both sides should be the same. This means AA or BB end caps are placed on left/right boundaries.

In the inner corner of core or blockage, double-height end cap cell is required. The double-height cells are also placed the same way as single-height cells. For A type or B type double-height cell, two kinds of power rail should be defined: VSS-VDD-VSS, VDD-VSS-VDD. So there are 4 types of double-height cell: A(VSS-VDD-VSS), A(VDD-VSS-VDD), B(VSS-VDD-VSS), B(VDD-VSS-VDD). Accordingly, two kinds of double-height sites should be defined: VSS-VDD-VSS and VDD-VSS-VDD. All those end cap cells and sites must be pre-defined in LEF.

Before end cap insertion, you need specify all the end cap cells for each boundary in `setEndCapMode` to constitute an enclosed end cap ring around the core area, placement blockage, and hard macros. For this, you use the following options of `setEndCapMode`.

- `-leftBottomCornerEven`
- `-leftBottomCornerOdd`
- `-leftBottomEdgeEven`
- `-leftBottomEdgeOdd`
- `-leftEdgeEven`
- `-leftEdgeOdd`
- `-leftTopCornerEven`

- `-leftTopCornerOdd`
- `-leftTopEdgeEven`
- `-leftTopEdgeOdd`
- `-rightBottomCornerEven`
- `-rightBottomCornerOdd`
- `-rightBottomEdgeEven`
- `-rightBottomEdgeOdd`
- `-rightEdgeEven`
- `-rightEdgeOdd`
- `-rightTopCornerEven`
- `-rightTopCornerOdd`
- `-rightTopEdgeEven`
- `-rightTopEdgeOdd`

The `addEndCap` command will honor these settings. Therefore, it is important to specify the correct end cap cell type, especially for the double-height cells.

Deleting End-Cap Cells

To remove end-cap cells, use the Delete Instances form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes end-cap cells that are completely contained within the area; it does not delete end-cap cells that cross the area boundary.

For more information see

- `addEndCap` and `deleteFiller` in the "Placement Commands" chapter of the *Innovus Text Command Reference*

Placing Spare Cells and Spare Modules

Placing Spare Cells That Are Included in the Netlist

If spare cell instances are included in the gate-level netlist, the software places them during preplacement processing; however, you must specify them during the floorplanning session.

> ✓ Cadence recommends that you place clusters of spare cells at different locations within the core area to allow easy access to the cells from different parts of the core.

1. Specify the spare cells by using the following command:
   `specifySpareGate`

   Use the following parameter to identify the module whose hierarchy contains the spare cell instances:
   `-hinst`

2. If the design contains hierarchical modules, specify the following `setPlaceMode` parameter to ensure that the spare cells within the modules are kept within bounds of the hierarchy, even if no constraint is set on it:

   ```
   -place_global_module_aware_spare
   ```

   For more information on using this parameter, see "Running Hierarchy-Aware Spare Cell Placement".
   Note: In the GUI, select the *Hierarchy Aware Spare Cell Placement* option on the Design - Mode Setup - Placement form.

Related Topics

For more information, see the following commands in the "Placement Commands" chapter of the *Innovus Text Command Reference*:

- `setPlaceMode`
- `specifySpareGate`

Placing Spare Cells That Are Not Included in the Netlist

If the netlist does not include spare cell instances, you must create a spare module and place it before you place the standard cells.

1. Use the following command to create a spare module:
   ```
   createSpareModule
   ```
2. Use the following command to place the module:
   ```
   placeSpareModule
   ```

To delete a spare module, use the following command:

```
deleteSpareModule
```

For more information, see the following commands in the "Placement Commands" chapter of the *Innovus Text Command Reference*:

- `createSpareModule`
- `placeSpareModule`
- `deleteSpareModule`

Spare Cell Placement Behavior

- If there are no floorplanning constraints, or if the design has not been floorplanned, the software places spare cell instances randomly in the core area.
- If a spare cell instance is contained in a fence or a region, the software places the instance randomly in the fence or region that includes the instance.
- If spare cell instances in the netlist are grouped into modules, the software places the modules in a grid fashion in the core area.

  Note: For information on controlling spare cell placement when hierarchy is an issue, see "Running Hierarchy-Aware Spare Cell Placement"

Spare cell distribution is dependent upon the way spare cells are connected.

- If the spare cells are floating (that is, if they are not connected) or they are connected to power or ground, they are evenly distributed in the placement area.
- If the spare cells have connections to other spare cells, they are treated as a spare cell group and are placed close to one another in the placement area.
- If the spare cells, or a group of spare cells, have a connection to a non-spare cell instance, they are placed close to that instance.

To instruct the software to disregard spare cell connections and distribute the cells evenly in the placement area, complete one of the following steps before running placement:

- Specify the following command:
  ```
  setPlaceMode -place_global_ignore_spare true
  ```
- Select the Ignore Spare Cell Connections option on the Placement page of the Design - Mode Setup form.

For more information, see `setPlaceMode` in the "Placement Commands" chapter of the Innovus Text Command Reference.

Running Hierarchy-Aware Spare Cell Placement

To control placement of spare cells or modules in the netlist when hierarchy is an issue, use the following commands:

- `specifySpareGate  {-hinst | -inst}`
- `setPlaceMode -place_global_module_aware_spare  {true | false}`

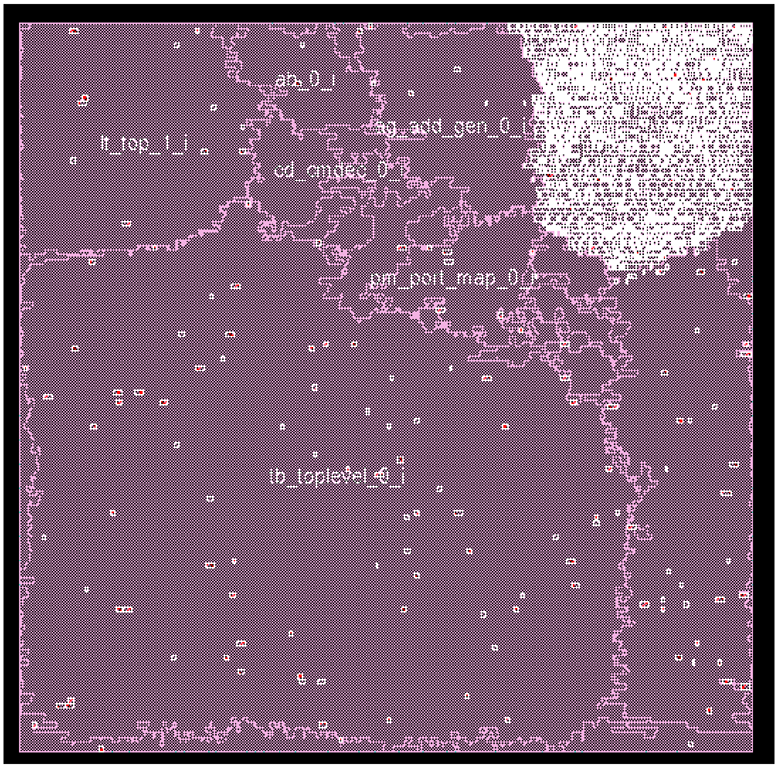
The following examples and figures show how these commands affect placement.

```
specifySpareGate -inst blk1/spare_1/*
# Works also for specifySpareGate -hinst
setPlaceMode -place_global_module_aware_spare true
place_design
```

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/*
}
place_design
```

Though spare gates in the netlist are under a particular hierarchy, they get evenly spread across the core area by default and are not restricted to hierarchy bounds. In the figure below, the instances highlighted in white belong to the same module. The red instances (enclosed in white) are the spare gates associated with the module. They are spread throughout by default.

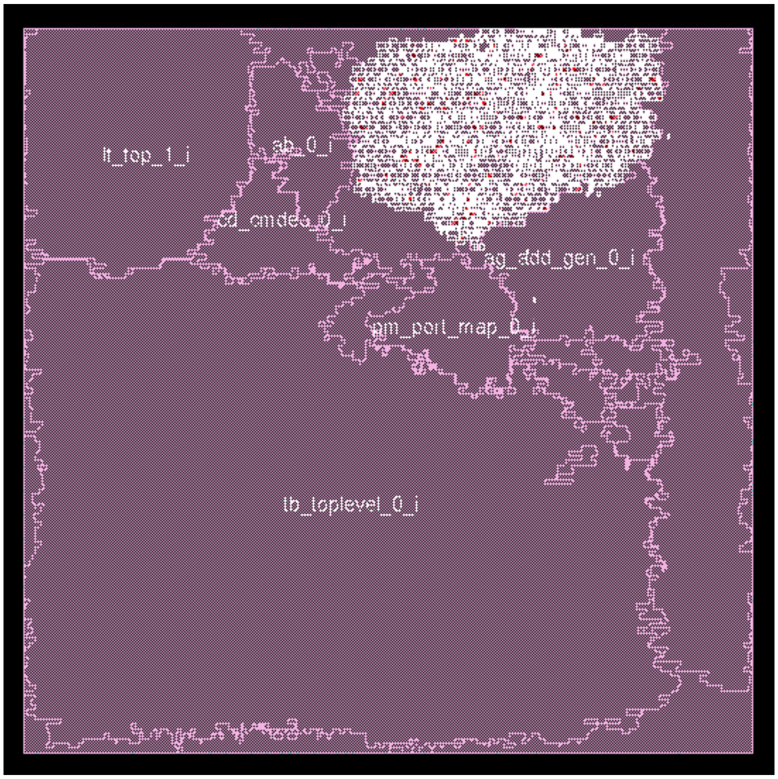


The log file, before Iteration 1, contains the following information: `Identified 240 spare or floating instances, with no clusters.`

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 -place_global_module_aware_spare true
place_design
```

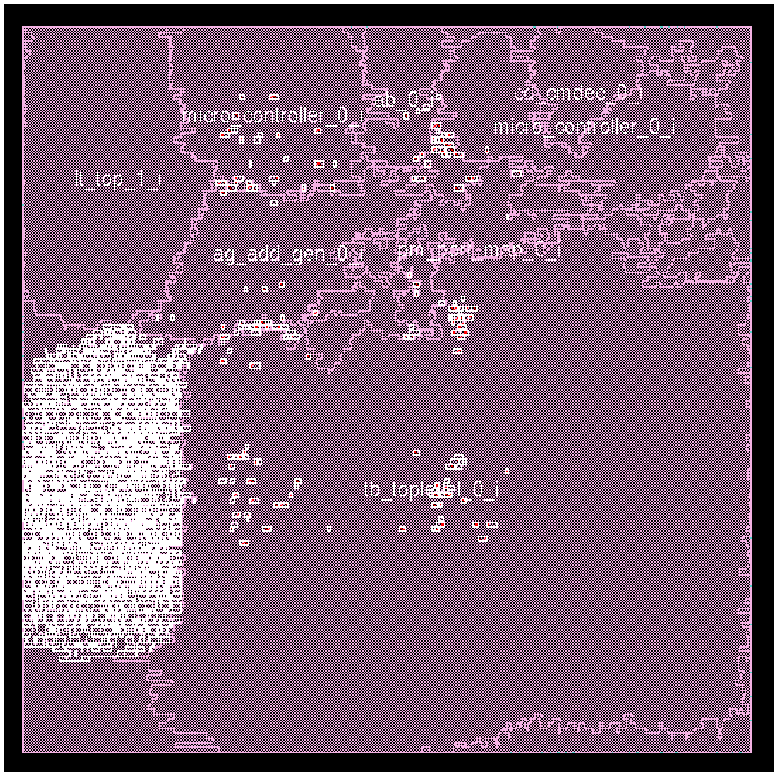
The following figure shows the spare gates (red) spread only within the bounds of the same module.



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:

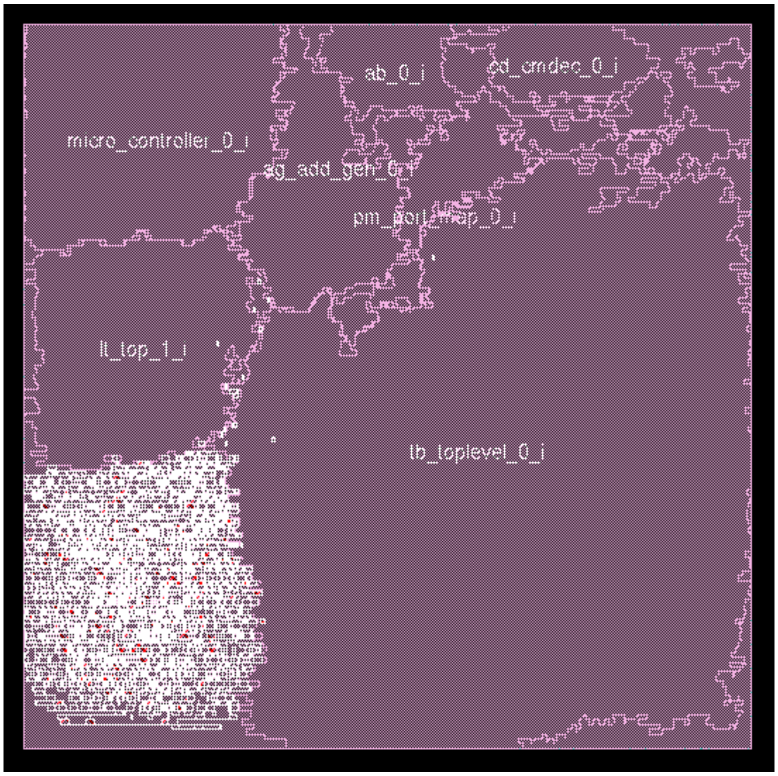
```
for {set x 0} {$x <6} {incr x 1} {
specifySpareGate -hinst lt_top_0_i/spare_${x}i/*
}
place_design
```



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 -place_global_module_aware_spare true
place_opt_design
```



The log file, before Iteration 1, contains the following information: `Identified 240 spares within logical modules, of which 240 are in 6 spare-only modules.`

Adding Padding

Add padding to reserve placement space for cells or routing added after placement, for example, to make sure there is room to insert clock buffers when running Clock Tree Synthesis (CTS) on a highly localized clock. The software adds the padding on the right side of placed instances at a default *metal2* pitch dimension.

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

> If the clock in your design is concentrated in a tight area, reserve three to five percent of the targeted final utilization to add clock buffers. If your initial settings do not provide sufficient space for the buffers, add more padding and rerun placement after CTS or placement optimization.

Adding Instance or Module Padding

Instance padding and module padding reserve space during global placement so it can be used later in the design flow, for cells added during placement legalization, Clock Tree Synthesis (CTS), or timing optimization.

Note: Cell padding is ignored by `refinePlace` if instances are fixed. Use `setPlaceMode [-place_detail_pad_fixed_insts {true | false}]` if cell padding needs to be honored for fixed instances. For more information, see `setPlaceMode` .

Adding Instance Padding

Instance padding is specified in terms of the number of sites occupied by each instance. For example, if a row fits 30 single-site instances without padding, you can specify padding of two sites for each instance in the row. In this case, each instance in the row will then occupy three sites, and the row will fit only 10 instances.

1. Specify the following command:

   ```
   specifyInstPad
   ```

2. (Optional) Report instance padding by using the following command:
   reportInstPad

To delete instance padding, use the following command:

```
deleteInstPad
```

For more information, see the following commands in the "Placement Commands" chapter of the *Innovus User Guide:*

- `specifyInstPad`
- `reportInstPad`
- `deleteInstPad`

Adding Module Padding

To reduce localized congestion, add module padding.

1. Specify the following command:

   ```
   setPlaceMode  -place_global_module_padding  module factor
   ```

   This command adds padding within hierarchical modules by spreading out the standard cell instances within the modules. The padding is specified in terms of a factor that is applied to the instance area of all the cells within the module. For example, a factor of 1.2 increases the area by 20 percent.

   Note: The software ignores factors that are less than 1.0.

2. Run standard cell placement.

For more information, see `setPlaceMode` in the "Placement Commands" chapter of the *Innovus User Guide.*

Adding Cell Padding

Cell padding adds hard constraints to placement. The constraints are honored by cell legalization, CTS, and timing optimization, unless the padding is reset after placement so those operations can use the reserved space. You can use cell padding to reserve space for routing.

- Specify the following command:

  ```
  specifyCellPad
  ```

  This command adds padding on the right side of library cells during placement. (Padding location is dependent on the orientation of the cell. For example, if the library cell is flipped when it is instantiated, the padding is on the left side.) The padding is specified in unit of placement SITE. For example, if you specify a value 2, the software ensures that there is additional clearance of two placement SITES on the right side of the specified cells.

To delete cell padding, use the following command:

```
deleteAllCellPad
```

For more information, see the following commands in the "Placement Commands" chapter of the *Text Command Reference:*

- `specifyCellPad`
- `deleteAllCellPad`

Placing Standard Cells

Place standard cells with the `place_opt_design` command. By default, the command runs preplacement optimization and standard cell placement. It also assigns or reassigns IO pins that are not in preplaced status and executes preCTS flow with both placement and preCTS optimization.

If you specified SDC timing constraints, it runs in timing-driven mode by default. If you specified scan information, it performs scan tracing and reordering by default.

> ⓘ If `place_opt_design` does not place the standard cells, for example if all instances are fixed or if there is no placeable area, then `place_opt_design` also skips I/O pin assignment.

This command was designed as a super command; that is, with some exceptions, you can use it to place standard cells without specifying any placement options for your initial placement.

- To reduce to switching power on power-critical nets, consider running the following command before placing standard cells:
  ```
  setPlaceMode -place_global_clock_power_driven true
  ```

Tune the initial placement by trying the following techniques:

- If the design is congested, try the following command, then rerun placement:
  ```
  setPlaceMode -place_global_cong_effort high
  ```
- If the design has local congestion, try the following command, then rerun placement:
  ```
  setPlaceMode -place_global_module_padding  module factor
  ```

Related Topics

- `place_design` and `setPlaceMode` in the "Placement Commands" chapter of the *Text Command Reference*

Running Placement in Multi-CPU Mode

The `place_opt_design` command and the `addFiller` command supports multi-threading. Multi-threading accelerates placement by splitting a job into two or more tasks that run concurrently on a single machine that has multiple processors. The placement acceleration is not linear, however, because some set-up and synchronization time is required.

Multi-threading requires additional licenses. The number of additional licenses required is dependent on the "base" Innovus Digital Implementation System license (the base license is the license used to invoke the software) and the number of threads you want to use.

> ⓘ To get the greatest benefit from multi-threading, your placement job should be the only job running on your machine. You should avoid all other tasks, even a regular system backup. For example, a machine with four CPUs that is running backup or other system tasks that occupy one CPU might show less speed-up with four threads than a machine running no system tasks that is running global placement with three threads.

Multi-Threading Placement Steps

To run multi-threading placement, complete the following steps. You can complete these steps before running any commands that run multiple-CPU processing, or before running placement. Because the Innovus software has a common interface for multiple-CPU processing (multi-threading or distributed processing), you need specify these commands only once per session, and any application that can run in multiple-CPU processing mode can use the additional licenses and processors.

1. (optional) Use the following command to specify the number of multiple-CPU licenses to check out and the license check-out order:

   ```
   setMultiCpuUsage [-acquireLicense  integer ] [-localCpu {integer | max}]\
   [-licenseList  licenses ]
   ```

   If you do not use this command, the software runs it automatically, using a default check-out order and requesting the appropriate number of licenses based on the parameters you set for `setMultiCpuUsage`.

2. Use the following command to specify the maximum number of threads to use:

   ```
   setMultiCpuUsage -localCpu {integer | max}
   ```

   If you request more threads than are available, the software uses the maximum number that are available.

   Note: It is generally not a good idea to request more threads than the number of CPUs in your machine, as it will slow down the machine and waste licenses.

3. (optional) Use the following command to release the additional licenses after global placement:

   ```
   setMultiCpuUsage -keepLicense true
   ```

   > ✓ Alternatively, run the following command after placement to release the additional licenses immediately:

   ```
   setMultiCpuUsage -releaseLicense
   ```

4. Run global placement.

   The log file reports the number of threads used for multi-threading placement just before it shows the global placement iterations, for example:

   ```
   Placement running 2 threads
   ```

5. (optional) Check the run-time information at the end of the `place_opt_design` section of the log file.

   ```
   (cpu for global=1:25:08) real=0:50:32***
   Placement multithread real runtime: 0:50:32 with 2 threads.
   Core Placement runtime cpu: 1:14:24 real: 0:41:42
   Starting refinePlace ...
   ```

The number to look for in the log is `Placement multithread real runtime`. In the preceding example, the `Placement multithread real runtime` is `0:50:32`.

Calculating Multi-Thread Speed-Up

The amount of time used for global placement is calculated by using the following formula:

```
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:

```
.log

Finished Global Placement (cpu=0:00:18.3, real=0:00:18.0, mem=1559.9M)

*** Starting place_detail (0:03:42 mem=1545.7M) ***

.logv

Finished Global Placement (cpu=0:00:18.3, real=0:00:18.0, mem=1559.9M)

Solver runtime cpu: 0:00:13.0 real: 0:00:13.1

Core Placement runtime cpu: 0:00:18.0 real: 0:00:18.0

Starting place_detail (0:03:42 mem=1545.7M) ***
```

For multithreading, the log would have reported the following:

```
.log:

Finished Global Placement (cpu=0:00:25.6, real=0:00:13.0, mem=1339.6M)

*** Starting place_detail (0:00:51.1 mem=1295.4M) ***

.logv

MT:

Finished Global Placement (cpu=0:00:25.6, real=0:00:13.0, mem=1339.6M)

Placement multithread real runtime: 0:00:13.0 with 8 threads.

Solver runtime cpu: 0:00:18.6 real: 0:00:06.3

Core Placement runtime cpu: 0:00:25.2 real: 0:00:12.0

*** Starting place_detail (0:00:51.1 mem=1295.4M) ***
```

Comparing the times from the two log segments gives the following calculations:

- `real time (1 thread) = 1:15:09 = 4509 seconds`
- `real time (2 threads) = 0:50:32 = 3032 seconds`

```
4509 / 3032 = 1.49
```

ⓘ If other jobs are running during multi-threading placement, the `real` time includes the run time for those jobs, so you do not get an accurate speed-up comparison.

Related Topics

- Accelerating the Design Process By Using Multiple-CPU Processing
- Multiple-CPU Processing Commands chapter in the *Text Command Reference*.

Checking Placement

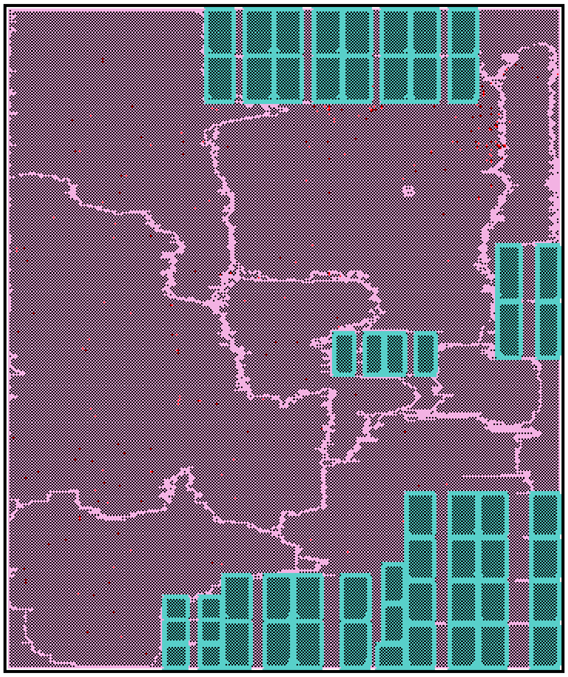
Use the following methods to check placement:

- Amoeba view
  For more information, see The Main Window chapter in the *Innovus Menu Reference*.
    - `checkPlace` command
- For more information, see `checkPlace` in the "Placement Commands" chapter of the *Innovus Text Command Reference.*
- Placement density map
  For information, see the following references:
- "Placement Commands" chapter of the *Innovus Text Command Reference*
    - `getDensityMapMode`
    - `reportDensityMap`
    - `setDensityMapMode`
- "Placement Menu" chapter of the *Innovus Menu Reference*
    - Display Density Map
- Violation Browser

  Note: The Violation Browser does not indicate the layer on which a placement violation occurs.

Using the Amoeba View

Use the Amoeba view to see the placement of modules and blocks. For example, in the following figure you can see the outlines of the hard blocks and the modules, and that the instances in each of the modules are placed closely together.



To display the Amoeba view, select the *Amoeba view* widget from the *Views* panel in the main Innovus window.

For more information, see The Main Window chapter in the *Innovus Menu Reference*.

Using the Density Map

Use one of the following methods to turn the display of the density map on or off:

- Select *Density Map* on the list of Visibility toggles in the main Innovus window.
- 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. If filler cells are added after routing, the software checks for DRC violations between regular nets and the filler cells.

To add filler cells, use the `addFiller` command.

> ✓ Provide a list of filler cells so that at least one filler cell can be used to fill the space without causing DRC violations.

Add Filler recognizes whether a filler cell has implant layer geometries and attempts to add fillers that honor the implant layers' width, spacing and minimum area rules. By judicious selection of filler cells, the software can correct implant layers' minimum spacing errors by putting in same voltage threshold implant layer fillers in spaces between two same implant layer cells. Add Filler also avoids creating implant layer minimum width and minimum area errors by abutting fillers of same implant layer as the adjacent cells, thus extending the implant layer width.

> ⓘ 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.

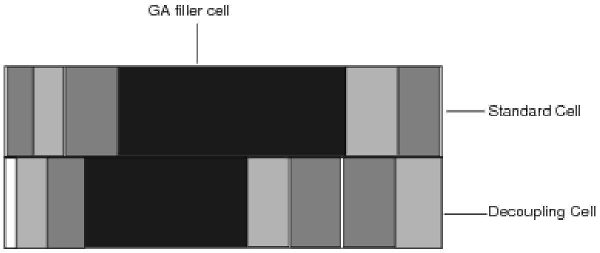
Placing Gate Array Style Filler Cells for Post-Mask ECO

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.

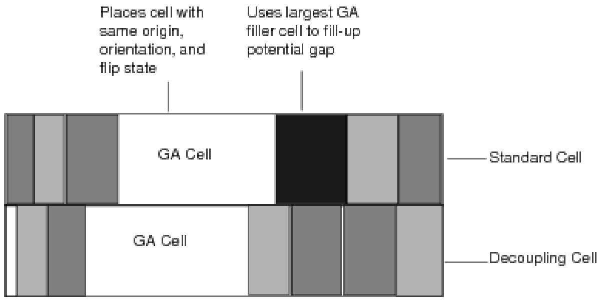
- Specify the following command:

```
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.


Places cell with same origin, orientation, and flip state
Uses largest GA filler cell to fill-up potential gap
GA Cell
GA Cell
Standard Cell
Decoupling Cell

The ecoPlace command also contains options that let you map unplaced standard cells to spare cells, and map GA cells to GA core sites. You can specify that instances that are PLACED cannot be moved.

For more information, see the following command in the Innovus Text Command Reference:

- `ecoPlace` in the "Interactive ECO Commands" chapter

Adding Decoupling Capacitance

Adding decoupling capacitance to a design can help maintain a stable voltage between power and ground when signal nets switch. This can reduce IR drop for power nets and limit bouncing on ground nets.

The Innovus software adds decoupling capacitance by choosing from the specified available decoupling capacitance cell candidates, and adding enough cells until their combined total capacitance value equals the user-specified value. You can insert decoupling capacitance homogeneously inside a specified area, or based on the peak current density of the instances in the area.

1. To define the cells to use for decoupling capacitance insertion, use the `addDeCapCellCandidates` command.

   For example, the following commands define two decoupling capacitance cell candidates: `DECAP10` has a capacitance value of 10fF, and `DECAP8` has a capacitance value of 15fF.

   ```
   addDeCapCellCandidates DECAP10 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 `DECAP10` and `DECAP8` cells:

   ```
   addDeCap  1000 -cells DECAP10 DECAP8
   ```

Deleting Decoupling Capacitance

- To clear all available decoupling cell candidates, use the `clearDeCapCellCandidates` command.
- To delete all of the decoupling capacitance cells in a design, use the `deleteDeCap` command.

Adding Logical Tie-Off Cells

Tie-off cell instances provide connectivity between the tie-hi and tie-lo logical input pins of the netlist instances to power and ground. This connectivity does not cross the hierarchy module boundaries. The number of tie-off instances added can be controlled by setting the distance and fanout constraints using the `setTieHiLoMode` command.

To add logical tie-off cells to the design after placing the netlist, use the Place Menu - Add Tie Hi/Lo form or the `addTieHiLo` command. To remove added logical tie-off cell instances, you can use the `deleteTieHiLo` command.

Saving Placement Data

You can save placement data in the Innovus place format or in DEF and PDEF placement data formats. This can be done at any time after running placement. To save placement data, use the `saveDesign` command.

Specifying and Placing JTAG and Other Cells Close to the I/Os

You can constrain the placement of JTAG cells and other cells so they are placed close to the outer core area. Place these cells before you run placement in the rest of the design.

When the software runs JTAG placement, it creates a temporary blockage over the area where the cells must not be placed and removes it after the placement.

You can constrain the placement of instances, hierarchical instances, or cells.

1. Use the following command to constrain placement:
   specifyJtag

   To include instances or cells other than JTAG cells, you must identify them with the `specifyJtag` command.

   To undo the specification, use the following command:
   unspecifyJtag

2. To place the instances or cells, use the following command:
   placeJtag
3. (optional) To generate a report of the JTAG placement, use the following command:
   reportJtagInst

To undo JTAG placement, use the following command:

unplaceJtag

> ✓ If you do not want to place regular instances in the JTAG outer core area after running JTAG placement, specify a placement blockage prior to running placement.

Related Topics

For more information, see the following commands in the "Placement Commands" chapter of the Innovus User Guide:

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

Optimizing and Reordering Scan Chains

The `place_opt_design` command reorders scan chains by default, unless it is in prototyping mode. If you decide not to reorder scan cells with `place_opt_design` but run standalone scan chain reorder flow, use the information provided in this section.

Related Topics

To see this step in the design flow, see "Place the Design and Run PreCTS Optimization" in the *Innovus Foundation Flows: Flat Implementation Flow Guide*.

Specifying Scan Cells

Scan cells are usually identified and read automatically from the timing library during design import. Use the `specifyScanCell` command to define scan cells that the software cannot retrieve from the library.

You can specify scan chains in a design by defining them in a DEF file, or by using the `specifyScanChain` command.

If scan chains are specified by reading in a DEF file, the software does a native scan trace. The scan DEF file is stored in the database and, when the `scanReorder` command runs, the software matches the scans and honors the ordered segments. However, if you run specify `setPlaceMode -place_global_reorder_scan 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.

When the Innovus software reads the netlist, it outputs the following messages:

```
Reading netlist ...

Reading verilog netlist ". fileName "

Inserting temporary buffers to remove assignment statements.
```

Reordering Scan Chains

Use one of the following approaches to scan chain reordering:

- Native scan reordering

  Use this approach in the following conditions:

  - Single-clock domain, single-edge chains
  - Multiple clock domain chain segments separated by data lockup elements
  - Shared functional output signal chains
- ScanDEF-based reordering

Use this approach in the following conditions:

- All simple scan chain architectures (handled by the native approach)
  - Implied domain transition scan chains (without data lockup elements)
  - Scan chains with ordered segments
  - Scan chains generated by LogicVision software

After reordering scan chains, save a netlist of the design using one of the following methods:

- *Save - Netlist* form (*Design - Save - Netlist*)
- `saveNetlist` command

Native Scan Reordering Approach

Use the native approach to scan chain reordering when you do not have a scanDEF file.

This approach requires that you use the `specifyScanChain` command to identify the `START` and `STOP` signals of the top-level chains, or chain segments, in the netlist. Using this information, the software identifies the scan flip-flops along the scan chain when running the `scanTrace` command to analyze the scan flip-flop connections. You can also auto-detect data lockup latch elements using the `scanTrace -lockup` command.

If the scan cells are not listed in the timing library, you must specify them before tracing the scan chains. You can identify scan cells with the `specifyScanCell` command.

After `scanTrace` has identified the elements along the chain, complete the following steps:

1. (Optional) Ignore the scan connections:
   ```
   setPlaceMode -place_global_ignore_scan true
   ```
2. (Optional) Set scan reorder options:
   ```
   setScanReorderMode -skipMode [skipNone | skipBuffer | skipFloatingBuffer]
   ```
3. Run placement:
   ```
   place_opt_design
   ```

The recommended flow for scan chains that have data lockup latches is as follows:

1. Specify a scan chain in the design:
   ```
   specifyScanChain
   ```
2. Trace the scan chain connection with the automatic detection lockup latch elements:
   ```
   scanTrace -lockup [-verbose]
   ```
3. (Optional) Ignore the scan connections:
   ```
   setPlaceMode -place_global_ignore_scan true
   ```
4. (Optional) Set scan reorder options:
   ```
   setScanReorderMode -skipMode [skipNone | skipBuffer | skipFloatingBuffer]
   ```
5. Run placement:

```
place_opt_design
```

The recommended flow for scan chains that have data lockup flip-flops is as follows

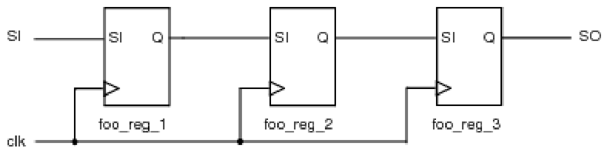
1. Specify a scan chain in the design:
   ```
   specifyScanChain ...
   ```
2. Specify a cell or instance as a lockup flip-flop element:
   ```
   specifyLockupElement ...
   ```
3. (Optional) Ignore the scan connections:
   ```
   setPlaceMode -place_global_ignore_scan true
   ```
4. (Optional) Set scan reorder options:
   ```
   setScanReorderMode -skipMode [skipNone | skipBuffer | skipFloatingBuffer]
   ```
5. Run placement:
   ```
   place_opt_design
   ```

Note: The `scanReorder` command automatically calls `scanTrace` internally if you have not previously run `scanTrace`. By default, this internal `scanTrace` run specifies that the tracing will not detect lockup elements (`-noLockup`); therefore, if you have lockup latches, Cadence recommends using the `scanTrace -lockup` command before `scanReorder`, or `specifyLockupElement` prior to running `scanReorder`.

Valid Design Types

You can use the native approach to scan chain reordering on designs comprising a simple scan chain architecture with the following characteristics:

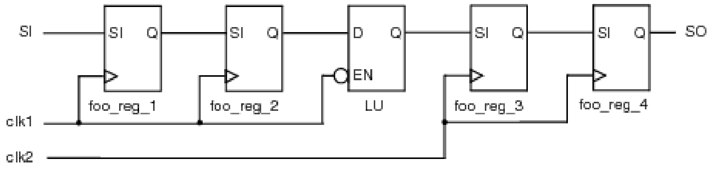
- Single-clock domain, single-edge chains
  In the following figure, all `foo_reg` scan flip-flops are triggered by the same clock domain and phase.



- `foo_reg_1`, `foo_reg_2`, and `foo_reg_2` scan flip-flops are triggered by `clk1` (positive edge).

```
specifyScanChain chain1 -start SI -stop SO
scanTrace [-verbose]
```

- Multiple clock domain chain segments separated by data lockup elements
  In the following figure, all domain or edge transitions are separated by a data lockup element.



- foo_reg_1 and foo_reg_2 scan flip-flops are triggered by clk1 (positive edge).
- foo_reg_3 and foo_reg_4 scan flip-flops are triggered by clk2 (positive edge).
    - LU represents a data lockup element of type latch.

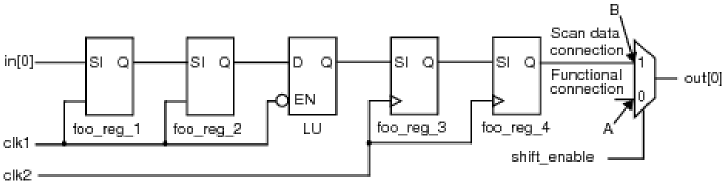
      ```
      specifyScanChain chain1 -start SI -stop SO
      scanTrace -lockup [-verbose]
      ```

      All elements along the scan chain are assumed reorderable from the specified START and STOP signals unless there is a data lockup element in the scan data path. The presence of a data lockup element works as a boundary so that the chain segments on either side of the lockup element are individually reordered. For this example, the top-level chain is reordered as two individual scan chain segments:

        - reorderable segment 1: SI > LU/D
        - reorderable segment 2: LU/Q > SO

- Shared functional output signal chains

  If the STOP signal of the scan chain is also a shared functional output, the endpoint of the scan chain must be specified to the scan input (SI) pin of the last register in the scan chain, or to the data input pin of the multiplexer (MUX), which drives the shared functional output signal. This is necessary because scanTrace does not perform the forward trace from the last flip-flop in the scan chain through the MUX instance. The following figure is an example of shared functional output:



The following command sequence performs the forward trace from the last flip-flop in the scan chain to the MUX instance:

```
specifyScanChain chain1 -start in[0] -stop MUX/B
scanTrace -lockup [-verbose]
```

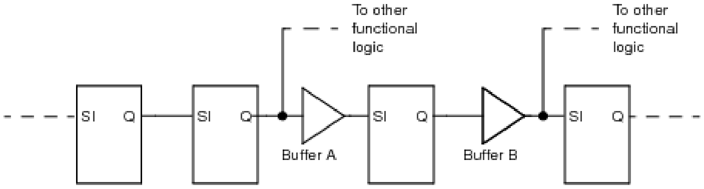
The following command sequence does not perform the forward trace from the last flip-flop through the MUX instance; scanTrace will not succeed:

```
specifyScanChain chain1 -start in[0] -stop out[0]
scanTrace -lockup [-verbose]
```

Scan Chains with Two-Pin Logic Cells

Scan chains often contain two-pin logic cells, usually buffers. The scan tracing algorithm always recognizes and traces through two-pin cells. The `scanReorder` command parameters control whether two-pin cells remain in the scan chain after scan reordering.

In the following scan chain example, buffer A is in the scan chain, but not part of the functional logic of the design, and therefore can be deleted. Buffer B is part of the functional logic, and must not be deleted.



The `scanReorder -skipMode skipNone` command retains all two-pin cells in the scan chain, and reordering changes only the connections to the two-pin cells' outputs. The nets connected to the two-pin cells' inputs will not be modified. In the preceding example, both buffers A and B would be retained, and scan reordering would be performed by rearranging the scan input pin connected to their output nets.

The `scanReorder -skipMode skipBuffer` command reconnects the scan chain so that buffers (as defined by `setBufFootPrint`) are skipped. Buffers that are not part of the functional logic are deleted. This disconnects scan inputs and reconnects them directly to a scan output pin, skipping all buffers. Any two-pin cells that are not buffers are retained in the scan chain in the same manner as `skipNone`.

In the preceding example, buffer A would be deleted and functional buffer B would be retained in the netlist. Scan reordering would disconnect the scan input pin from the output of buffer B, and reconnect some other scan input pin to the input net of buffer B, so buffer B would no longer be in the scan chain.

The `scanReorder -skipMode skip_floating_buffer` 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.

> ⓘ Because `scanReorder -skipMode skip_floating_buffer` does not consider the functionality of cells that it removes from the scan chain, it can change the scan chain in unpredictable ways. For example, if buffer A in the preceding example was an inverter, it would be removed, and the test pattern would have to be changed to account for the loss of inversion.

scanDEF-Based Reordering Approach

If you have a scanDEF file that describes the set of reorderable scan chains in the design, Cadence recommends using the scanDEF approach. To reorder scan chains with the scanDEF approach, complete the following steps:

1. Read in the `scanDEF` file:

   ```
   defIn -scanChain
   ```

   Note: In the case where a DEF file contains a SCANCHAIN section, the `defIn` command automatically reads in the `scanDEF` file, so the `-scanChain` parameter is not necessary.

2. Run placement:

   ```
   place_opt_design
   ```

Using the scanReorder Command

When running the `scanReorder` command, the Innovus software uses the begin and endpoints from the scanDEF chains to trace the connectivity of the scan chains in the netlist. This check verifies whether the elements in the netlist scan chains are represented as elements in their respective scanDEF chains. As a result of this check, an internal representation of each scanDEF chain is created in the Innovus database.

When a netlist-to-scanDEF file mismatch occurs, for each instance mismatched, `scanReorder` issues the following WARNING message:

```
WARNING (SOCSC-5003): The scan chain was found to pass through instance <inst> in the netlist, but this instance
does not appear in the DEF scan chain.
```

Mismatches of combinational components (buffers or inverters) in the scan data path can be expected if the netlist has undergone pre-placement optimization, or if the scanDEF file is not properly formatted, as described in Netlist-to-scanDEF mismatch section. Sequential mismatches are tolerated if the mismatch occurs for a scan flop from the `FLOATING` section only of the scanDEF chain. However, sequential mismatches are not expected and indicate a discrepancy between the scan chains in the netlist, and the scanDEF chains. You should investigate the source of the discrepancy before proceeding with reordering. If necessary, revise the scanDEF description of the scan chains.

Using the internal representation of the scanDEF chains, Innovus issues the following message prior to reordering the chains in the netlist:

```
INFO: Scan reorder based on traced netlist chains.

INFO: Medium effort Scan reorder

INFO: Reordering scan chain <chainName>
```

Netlist-to-scanDEF Mismatch

Netlist-to-scanDEF mismatches can occur if a driving scan flip-flop is buffered (or inverted) to the `SI` pin of the next scan flip-flop in the scan chain. In this situation, the driving scan flop and buffer (or inverter) should be captured to the scanDEF file as an `ORDERED` segment, rather than capturing the driving scan flip-flop as a freely reorderable element in the `FLOATING` section of the scanDEF chain. The correct syntax for the `FLOATING` and `ORDERED` sections of the scanDEF file is as follows:

```
- chain X
    + START PIN
    + FLOATING
        ....
        next_scan_flop_reg ( IN SI ) ( OUT SO )
    + ORDERED
        driving_scan_flop_reg ( IN SI ) ( OUT SO )
        buf_instance ( IN A ) ( OUT Y )
    + STOP
```

In previous releases of Innovus, when a scanDEF to netlist mismatch occurred, scan reorder would abort. If the mismatches were due to combinational components (buffers or inverters) in the scan data path, tool would automatically correct scan connection:

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

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
```

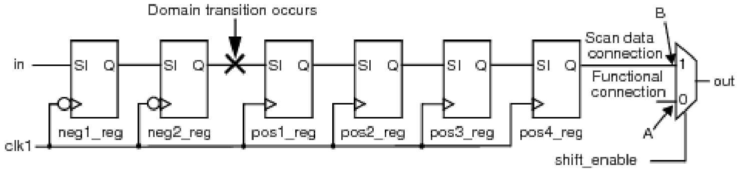
The logic synthesis tool writes the input `scanDEF` file after the top-level scan chains are created in the design. Each top-level scan chain can be segmented into multiple scanDEF chains because the elements along each scanDEF chain must belong to the same clock domain, and be triggered by the same 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 Genus 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 ;
```

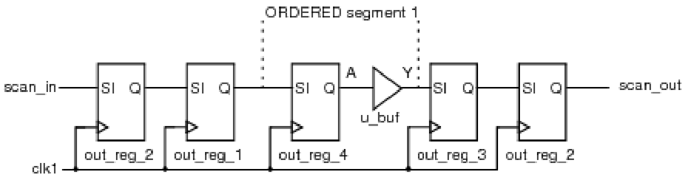
  The integer corresponds to as many `ORDERED` segments as defined in the original scan chain. For example, a scanDEF chain with one `ORDERED` segment is as follows:

```
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.

To save scan files, use the *Save DEF* form or the `defOut` command.

Loading Scan Files

To load scan files in DEF format, use the *Load DEF File* form. For DEF, use the `defIn` command.

Clock Tree Synthesis

Overview

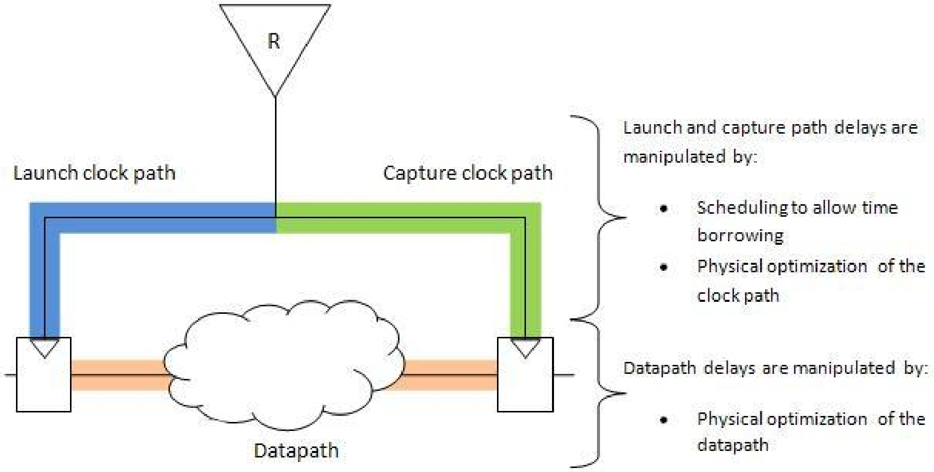
The Innovus™ Implementation System (Innovus) offers Clock Tree Synthesis (CTS) as part of full Clock Concurrent Optimization (CCOpt) and as a stand-alone function. To invoke full CCOpt, which always includes CTS, use the `ccopt_design` command . To invoke CTS as a stand-alone function, use the command, `ccopt_design -cts`.

CCOpt extends CCOpt-CTS to replace traditional global skew balancing with a combination of CTS, timing driven useful skew, and datapath optimization. In traditional CTS flows, an ideal clock model is used before CTS to simplify clock timing analysis. With the ideal clock model, launch and capture clock paths are assumed to have the same delay. After CTS, the ideal clock model is replaced by a propagated clock model that takes account of actual delays along clock launch and capture paths. In traditional CTS, global skew balancing attempts to make the propagated clock timing match the ideal mode clock timing by balancing the insertion delay (clock latency) between all sinks. However, a number of factors combine such that skew balancing does not lead to timing closure. These include:

- **OCV** – On-chip variation means that skew, measured using a single metric such as the 'late' configuration of a delay corner, no longer directly corresponds to timing impact because launch and capture paths have differing timing derates. In addition, Common Path Pessimism Removal (CPPR) and per-library cell timing derates mean that it is not possible to accurately estimate clock or datapath timing without synthesizing a clock tree. Advanced OCV (AOCV) further complicates this by adding path and bounding box dependent factors.
- **Clock gating** – Clock gating uses datapath signals to inhibit or permit clock edges to propagate from a clock source to clock sinks. The clock arrival time at a clock gating cell is unknown prior to CTS and this arrival time determines the required time for the datapath control signal to reach the clock gating cell enable input. Therefore, the setup slack at a clock gating enable input is hard to predict preCTS. In addition, clock gating cells have an earlier clock arrival time than regular sinks and are, therefore, often timing critical. Typically, the fan-in registers controlling clock gating may need to have an earlier clock arrival time than regular sinks in order to avoid a clock gating slack violation – which means the fan-in registers need to be skewed early.
- **Unequal datapath delays** – Front end logic synthesis will attempt to ensure that logic between registers is roughly delay-balanced to optimize the target clock frequency. However, with wire delay dominating many datapath stages, it is likely that after placement and preCTS optimization there will exist some combinational paths with unavoidably longer delays than others. Useful skew clock scheduling permits slack to be moved between register stages to increase clock frequency. In contrast, global skew balancing is independent of timing slack. In addition, CCOpt useful skew scheduling can avoid unnecessary balancing of sinks where there is excess slack in order to reduce clock area and clock power.

CCOpt treats clock launch, clock capture, and datapath delays as flexible parameters that can be manipulated to optimize timing. This is illustrated below.

Manipulating Clock Delays and Logic Delays


R
Launch clock path
Capture clock path
Datapath
Launch and capture path delays are manipulated by:
Scheduling to allow time borrowing
Physical optimization of the clock path
Datapath delays are manipulated by:
Physical optimization of the datapath

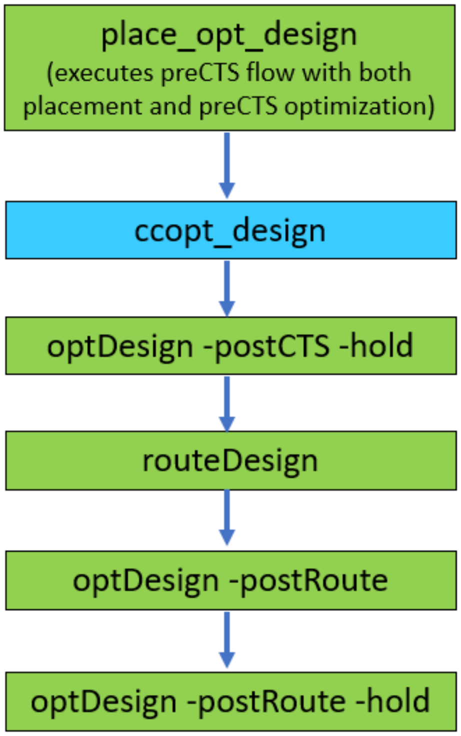
At each clock sink (flip-flop) in the design, CCOpt can adjust both datapath and clock delays in order to improve negative setup timing slack – specifically the high-effort path group(s) WNS. This is performed using the propagated clock timing model at all times.

For detailed information about the concept of moving slack between register stages and the concept of worst chains, see the Chains section.

Flow and Quick Start

The diagram below highlights the CCOpt and CCOpt-CTS steps as part of the standard Innovus block implementation flow. In the CCOpt flow, postCTS optimization is not required because `ccopt_design` includes postCTS style optimization both as part of CCOpt and as a further final internal optimization step.

CCOpt and CCOpt-CTS in the Design Flow


place_opt_design
(executes preCTS flow with both placement and preCTS optimization)
ccopt_design
optDesign -postCTS -hold
routeDesign
optDesign -postRoute
optDesign -postRoute -hold

An important consideration for the CCOpt and CCOpt-CTS flows is the need for high quality multi-mode timing constraints. The best strategy is to use postCTS timing constraints but with clocks in ideal clocking mode.

CTS is normally configured with multi-mode SDC constraints. This is recommended to get a complete clock tree network for CTS and eliminate the need for multi-pass CTS flows. It also becomes critical when global timing optimization with CCOpt useful skew scheduling is used for timing closure.

However, early clock tree feasibility and refinement of CTS strategy, flow, and settings can still be done with a limited constraint set. When certain mode constraints are not yet available or complete, you can choose to do the following:

- Continue CTS using the available modes - in early design stages only functional constraints may be available.
- Use CCOpt commands to define clock trees and skew groups for modes where the SDC may not be ready.

Note: Avoid using ‘CTS-specific’ SDC, which may be used in other tools or inherited from older FE-CTS based flows.

Define the set of analysis views for CTS along with the constraint modes, depending on the maturity of the design and constraints and what you need to achieve at this stage of the design.

Final tape out runs typically include multiple setup views and hold views. Earliest CTS feasibility is possible with one setup view.

CCOpt uses a “primary CTS delay corner” as the dominant corner for CTS construction and applies the global max_trans and skew constraints in this delay corner. If not specified, the last specified setup corner from the SDC constraints is used. When using multiple setup delay corners, it is recommended to explicitly set this to avoid changes to SDC or the order of constraint mode loading accidentally changing this setting:

```
set_ccopt_property primary_delay_corner <chosen setup delay_corner>
```

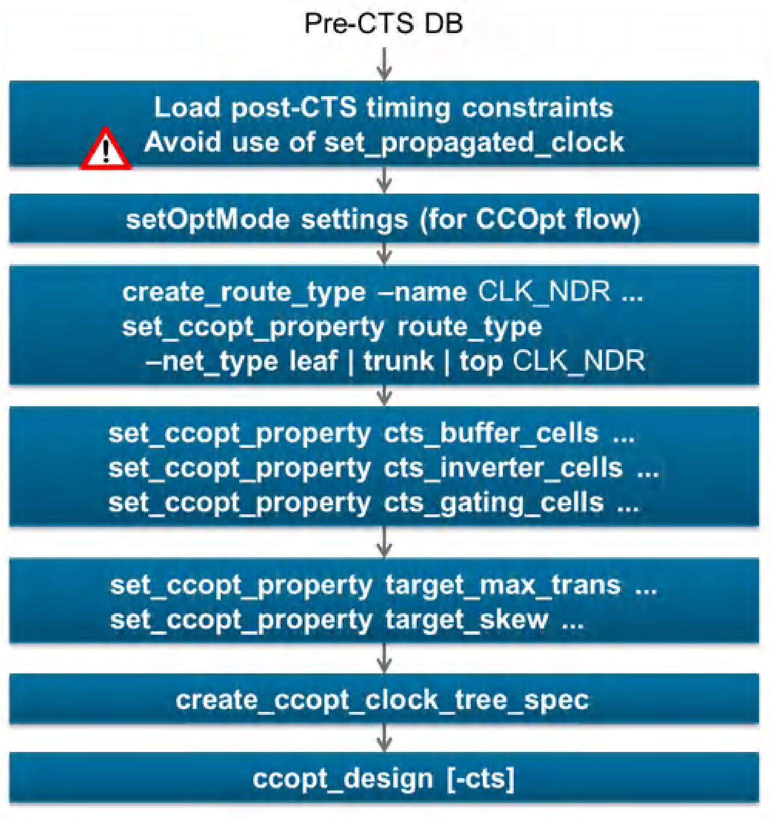
To determine the number of views that are sufficient for CTS, follow these recommendations:

- For early runs - one setup view may be sufficient
- For robust clock tree implementation (DRV and so on) - sufficient rc_corners, for example, Cmax and RCmax or equivalent views

- For setup timing optimization - all dominant setup views, which are typically one or two views
- For hold fixing - all dominant hold views, which are typically one to three views
- For final DRV checks - all necessary corners, but this checking can also be done later, for example, in Tempus

The diagram below summarizes the key configuration steps.

CCOpt and CCOpt-CTS Configuration Steps


Pre-CTS DB
Load post-CTS timing constraints
Avoid use of set_propagated_clock
setOptMode settings (for CCOpt flow)
create_route_type –name CLK_NDR ...
set_ccopt_property route_type
–net_type leaf | trunk | top CLK_NDR
set_ccopt_property cts_buffer_cells ...
set_ccopt_property cts_inverter_cells ...
set_ccopt_property cts_gating_cells ...
set_ccopt_property target_max_trans ...
set_ccopt_property target_skew ...
create_ccopt_clock_tree_spec
ccopt_design [-cts]

It is important to apply the intended postCTS configuration before invoking CCOpt but with clocks still in ideal timing mode. This is also recommended for use with CCOpt-CTS.

Switch to propagated timing model – CCOpt and CCOpt-CTS switch clocks to propagated mode and update source latencies of clock root pins such that the average arrival time of clocks after CTS matches the before CTS ideal mode arrival times. This process does not happen for clocks that are initially in propagated mode. The source latency update is important so that optimization of inter-clock and I/O boundary paths during `ccopt_design` operates with correct timing. In contrast, in a traditional flow this would be done as a separate flow step. For detailed information about the source latency update scheme, see the Source Latency Update section.

PostCTS configuration – CCOpt combines CTS with datapath optimization, replacing the need for a separate postCTS setup timing optimization step. Before running CCOpt the session should be configured with postCTS uncertainties, CPPR enabled, OCV timing derates or AOCV enabled, active analysis views, and all other settings appropriate for postCTS optimization. The `update_constraint_mode` , `setAnalysisMode` , and `setOptMode` commands are most commonly used for configuring relevant settings.

The example below illustrates a typical CCOpt/CCOpt-CTS configuration and run script. Some of these settings may also be configured via the Foundation Flow environment.

Quick Start Example

Load postCTS timing constraints. This example loads functional mode and scan mode constraints intended for postCTS use. You should ensure that clocks are not switched to propagated mode in these SDC files.

```
update_constraint_mode -name func -sdc_files func_postcts_no_prop_clock.sdc
```

```
update_constraint_mode -name scan -sdc_files scan_postcts_no_prop_clock.sdc
```

Declare the analysis views to be used during `ccopt_design`.

```
set_analysis_view  -setup {func_max_setup scan_max_setup} -hold {func_min_hold scan_min_hold}
```

Define route types. A route type binds a non-default routing rule and preferred routing layers. NDRs can be defined via LEF or using `the add_ndr` command.

```
create_route_type -name leaf_rule -non_default_rule CTS_2W1S  -top_preferred_layer M5 -bottom_preferred_layer M4
```

```
create_route_type -name trunk_rule -non_default_rule CTS_2W2S -top_preferred_layer M7 -bottom_preferred_layer M6 –
shield_net VSS
```

```
create_route_type -name top_rule -non_default_rule CTS_2W2S -top_preferred_layer M9 -bottomf_preferred_layer M8 –
shield_net VSS
```

Specify that the route types defined above will be used for leaf, trunk, and top nets, respectively. Note that top routing rules will not be used unless the `routing_top_min_fanout` property is also set.

```
set_ccopt_property -net_type leaf route_type leaf_rule
set_ccopt_property -net_type trunk route_type trunk_rule
set_ccopt_property -net_type top route_type top_rule
set_ccopt_property routing_top_min_fanout 10000
```

Specify that top routing rules will be used for any clock tree net with a transitive sink fanout count of over 10000.

Configure library cells for CTS to use. In this example, the `logic_cells` property is not defined so when resizing existing logic cell instances CTS will use matching family cells which are not `dont_use`. The specification of a library cell overrides any `dont_use` setting for that library cell.

```
set_ccopt_property buffer_cells {BUFX12 BUFX8 BUFX6 BUFX4 BUFX2}
set_ccopt_property inverter_cells {INVX12 INVX8 INVX6 INVX4 INVX2}
set_ccopt_property clock_gating_cells {PREICGX12 PREICG8 PREICGX6 PREICGX4}
```

Include this setting to use inverters in preference to buffers.

```
set_ccopt_property use_inverters true
```

Configure the maximum transition target.

```
set_ccopt_property target_max_trans 100ps
```

Configure a skew target for CCOpt-CTS (`ccopt_design -cts`). This is ignored by CCOpt (`ccopt_design`).

```
set_ccopt_property target_skew 50ps
```

Create a clock tree specification by analyzing the timing graph structure of all active setup and hold analysis views. The clock tree specification contains `clock_tree`, `skew_group,` and property settings.

Alternatively, the specification can be written to a file for inspection or debugging purposes and then loaded.

```
create_ccopt_clock_tree_spec  #create_ccopt_clock_tree_spec -file ccopt.spec
#source ccopt.spec
```

Run CCOpt or CCOpt-CTS

```
ccopt_design
#ccopt_design -cts
```

Report on timing

```
timeDesign -postCTS -expandedViews -outDir
```

Report on clock trees to check area and other statistics.

```
report_ccopt_clock_trees -file clock_trees.rpt
```

Report on skew groups to check insertion delay and, if applicable, skew.

```
report_ccopt_skew_groups -file skew_groups.rpt
```

Open the CCOpt Clock Tree Debugger Window. Alternatively, use the “CCOpt Clock Tree Debugger” entry in the main *Clock* menu.

```
ctd_win
```

For a more detailed explanation and recommendation on each of the above settings, see the Configuration and Method section. For details of the clock tree specification system, see the Concepts and Clock Tree Specification section.

Early Clock Flow

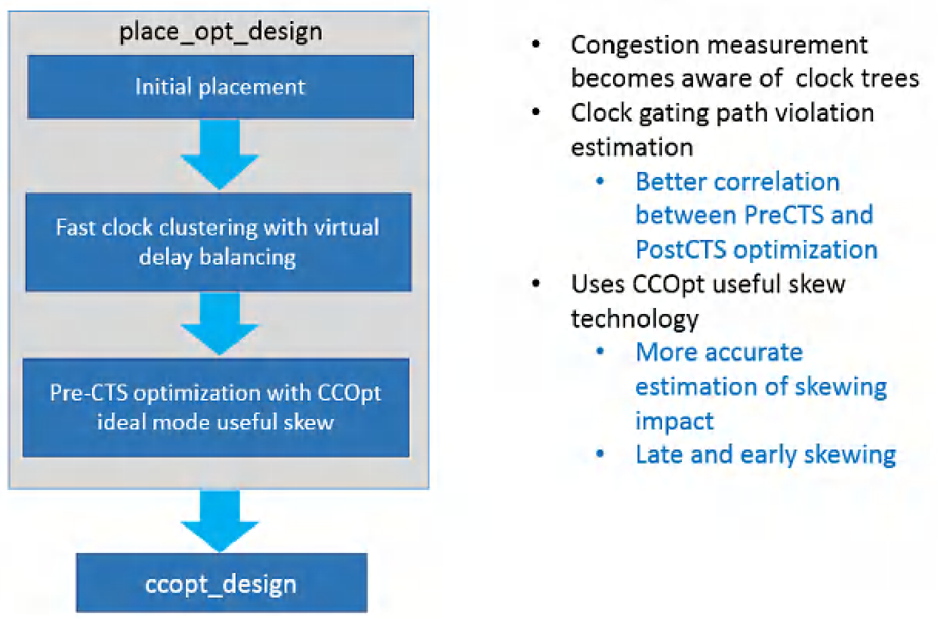
The `setDesignMode -earlyClockFlow true` setting enables the early clock flow inside `place_opt_design` .

When this feature is enabled, CCOpt creates clock trees during `place_opt_design` , based on an initial clustering, and annotates clock latencies for timing optimization. The flow also enables CCOpt ideal mode useful skew during the WNS/TNS fixing step inside `place_opt_design`. The high-level flow is shown in the diagram below.

Note:

- To use the early clock flow, the CTS configuration must be set up before running the `place_opt_design` command. For details, see the Flow and Quick Start section.
- Early clock flow mode `place_opt_design -incremental` reuses the existing early clock flow clock tree without any logical or physical modification and makes adjustments only to useful skew using virtual delays. It is also essential to note that the CCOpt clock tree spec must not be deleted between the original and incremental `place_opt_design` steps, just like it must not be done between `place_opt_design` and `ccopt_design`, otherwise any useful skew will be lost.

Early Clock Flow


place_opt_design
Initial placement
Fast clock clustering with virtual delay balancing
Pre-CTS optimization with CCOpt ideal mode useful skew
ccopt_design

- Congestion measurement becomes aware of clock trees
- Clock gating path violation estimation
  - Better correlation between PreCTS and PostCTS optimization
- Uses CCOpt useful skew technology
  - More accurate estimation of skewing impact
  - Late and early skewing

Use Model

Use the following commands to enable the early clock flow:

The `setDesignMode` `-earlyClockFlow` is set to `true`. By default it is set to `false`.

The table below outlines the combination of settings for early clock flow and useful skew CTS and the impact on `place_opt_design` behavior as a result.

earlyClockFlow	usefulSkewPreCTS	Behavior in place_opt_design
`false (default)`	`false`	No clock tree building No preCTS useful skew
`false (default)`	`true (default)`	No clock tree building PreCTS useful skew (skewClock)
`true`	`false`	Clock tree clustering No useful skew
`true`	`true`	Clock tree clustering CCOpt ideal mode useful skew

Configuration and Method

CCOpt Properties

Configuration of CCOpt-CTS and CCOpt is performed using a combination of the clock tree specification and CCOpt properties.

To set a property:

```
set_ccopt_property [-object_type <object>] <property name> <property value>
```

To get a property:

```
get_ccopt_property [-<object_type> <object>] <property name>
```

To obtain help on a property, or to find properties matching a wildcard pattern:

```
get_ccopt_property -help <property name or pattern>
```

To obtain a list of all available properties use this command:

```
get_ccopt_property -help *
```

The help for each property indicates which object type(s) the property applies to. Many properties are global properties for which an object type is not specified, but there are also properties that are applicable to specific object types including pins, skew groups, clock trees, and types of nets.

For example:

```
get_ccopt_property -help target_max_trans
...
Optional applicable arguments: "-delay_corner name", "-clock_tree name", "-net_type name", "-early" and "-late".
```

For a summary of all parameters of the `get_ccopt_property` and `set_ccopt_property` commands, see the *Innovus Text Command Reference*. You can also use the man command, for example: `man get_ccopt_property`

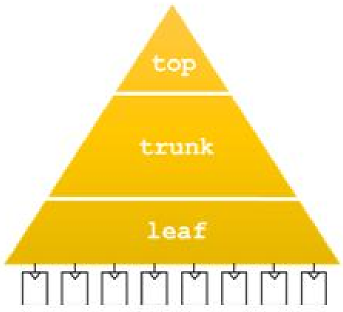
Note that some properties are read-only and can not be set. For further details of property manipulation, see the CCOpt Property System section.

For descriptions of all public CCOpt properties, see the CCOpt Properties chapter.

Route Types

CCOpt-CTS and CCOpt use the concept of top, trunk, and leaf net types as illustrated below.

Clock Tree Net Types



- **Leaf nets** – Any net that is connected to one or more clock tree sinks is a leaf net. By default, CCOpt-CTS and CCOpt will insert buffers so that no buffer drives both sinks and internal nodes.

- **Trunk nets** – Any net that is not a leaf net is by default a trunk net.
- **Top nets** – If you configure the `routing_top_min_fanout` property, then any trunk net which has a transitive fanout sink count higher than the configured count threshold will instead be a top net. For example, if the property is set to 10,000, then any trunk net that is above (in the clock tree fan-in cone) 10,000 or more sinks will be a top net.

You can define route types. A route type binds together a non-default routing rule (NDR), preferred routing layers, and a shielding specification. For each net type (leaf, trunk, and optionally top) you can specify the route type that should be used. Non-default routing rules can be defined via LEF or using the `add_ndr` command.

For example, the following command creates a route type with the name `trunk_type` that uses the `CTS_2W2S` non-default rule, with preferred routing layers `M6` and `M7`.

```
create_route_type -name trunk_type -non_default_rule CTS_2W2S
-top_preferred_layer M7 -bottom_preferred_layer M6
-shield_net VSS
```

The following command specifies that the `trunk_type` route type should be used for the trunk net type:

```
set_ccopt_property -net_type trunk route_type trunk_type
```

Top Net Configuration

As discussed above, the `routing_top_min_fanout` property can be configured with a sink count threshold to determine which nets are considered top nets instead of trunk nets. For example:

```
set_ccopt_property -clock_tree clk500m routing_top_min_fanout 10000
```

By default, each clock tree sink counts as ‘1’ for the purposes of top net thresholds. For macro clock input pins it may be desirable to treat the clock input pin as having a higher count, for example representing the number of internal state elements.
The `routing_top_fanout_count` property can be used to configure this.

For example, to specify that the clock input `mem0/clkin` to a memory should count as `1000` sinks, use the following command:

```
set_ccopt_property -pin mem0/clkin routing_top_fanout_count 1000
```

Routing Rule Recommendations

Clock net routing rules are crucial to obtaining low insertion delay and avoiding signal integrity problems. Especially for small geometry process nodes, the following recommendations should be considered:

- Configure the trunk net type to use double width double spacing and shielding. Prefer middle to higher layers subject to the power grid pattern. Double width is recommended to reduce resistance and permit use of bar shape vias, with double spacing to reduce the capacitance impact of shielding. Shielding is essential to avoid aggressors impacting clock trunk net timing, as impact on clock trunk timing is often significant for both WNS and TNS.
- Configure the leaf net type to use double width and prefer middle layers. Double width is recommended to reduce resistance. Extra spacing is desirable, but extra spacing and/or shielding may consume too much routing resource.
- Try to arrange for each net type (leaf, trunk, top) to use a single layer pair, one horizontal and one vertical, which have the same pitch, width, and spacing. This increases the correlation accuracy between routing estimates before clock nets are routed and actual routed nets.

Library Cells

The library cells used by CCOpt-CTS and CCOpt are configured with the properties listed below. These cell lists may be configured per-power domain and per-clock tree by using the `-power_domain` and `-clock_tree` keys for these properties, respectively.

`buffer_cells`
`inverter_cells`
`clock_gating_cells`
	Specifies buffer, inverter, and clock gating cells. It is recommended to explicitly configure buffer, inverter, and clock gating cells.

`logic_cells`	Specifies logic cells. If logic cells are not specified, CTS will use any library cell which has the same logic function and is not dont_use when resizing existing logic cell instances.
`use_inverters`	Specifies that CTS should prefer inverters to buffers.

To view detailed information about above properties, run the following command:

```
get_ccopt_property -help propertyname
```

For example:

```
get_ccopt_property -help buffer_cells
```

A Tcl list of symmetric buffer cells. All buffers created in the clock tree under a power domain will be instances of these cells.

Set the global property to specify buffer cells for all clock trees and all power domains:

```
set_ccopt_property buffer_cells {bufA bufB bufC}
```

Set the per-clock tree/power domain property to specify buffer cells for a particular clock tree and ALL power domains:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk
```

Set the per-clock tree /power domain property to specify buffer cells for a particular clock tree and power domain:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk -power_domain pd
```

By default CCOpt automatically selects appropriate cells.

Valid values: `list lib_cell`

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

The Quick Start Example illustrates commands to configure library cells. Specifying that CTS can use a library cell overrides any user or library dont_use setting for that library cell.

The following are some recommendations:

- Always specify library cells for buffers, inverters and clock gating.
- Use low voltage threshold (LVT) cells. The resulting insertion delay will be lower leading to less impact of OCV timing derates, therefore reducing the datapath dynamic and leakage power increase from timing optimization.
- For many, but not all, low geometry processes inverters result in lower insertion delay and lower power than buffers. In older technologies, buffers may be more efficient. The exact preference here is technology, library, and design target dependent.
- Permitting the largest library cells to be used may be undesirable for electromigration reasons and can increase clock power.
- Very weak cells, for example, X3 and below in many libraries, are usually undesirable due to poor cross-corner scaling characteristics and are sensitive to detailed routing jogs and changes.
- Limiting the number of library cells to no more than 5 per cell type may help reduce run time.

Note: Starting with the Innovus 16.2 version, the library cell list will be pre-screened at the start of CTS, and only those cells with the best drive strength and area characteristics will be used during CTS.

- Do not exclude small cells, such as X4, as otherwise CTS will be forced to use larger more power and area consuming cells to balance skew or implement the useful skew schedule.
- Include always-on buffers and inverters in designs with multiple power domains.

Transition Target

The maximum transition target to CCOpt is specified using the following command:

```
set_ccopt_property target_max_trans value
```

The value can be specified in library units, for example “100”, or specified in explicit units for example “100ps”, “0.1ns”.

The transition target can be specified by net type, clock tree and per power domain. For example, it may be desirable to have a tighter transition target at sink pins to improve flop `CK->Q` arc timing, but relax the transition target in trunk nets to reduce clock area and power. Shielding and extra spacing could be used for trunk nets to further reduce clock power whilst avoiding signal integrity problems. This example configures trunk nets to have a 150ps transition target whilst leaf nets have a 100ps transition target:

```
set_ccopt_property -net_type trunk target_max_trans 150ps
set_ccopt_property -net_type leaf target_max_trans 100ps
```

You can set transition targets per power domain. CTS deals with power contexts, which is a combination of a location power domain - the domain that the cell is physically in - and effective power domain - the logical domain. These are often the same, but may not be in the case of always-on buffering, for example. Transition targets apply to the effective power domain, while any per-domain cell lists apply to the location domain. In the case where multiple power contexts share the same effective domain, the maximum target over all relevant power contexts is used.

If a target max transition is not specified, CCOpt-CTS and CCOpt will examine the `target_max_trans_sdc property` (see the SDC Transition Targets section), and if that is not defined then an automatically generated target is chosen. It is recommended to set a transition target unless intentionally using settings that are to be obtained from the SDC constraints.

Note: Setting a max transition target that is either too low or too high can lead to poor tool runtime and quality of results. The command `check_design -type cts` provides extensive cross-checking of user transition targets.

Skew Target

A global skew target for CTS can be set using the following:

```
set_ccopt_property target_skew value
```

This will be applied in the primary CTS delay corner.

Additionally, per-skew group (and per-delay corner) skew targets can be set, for example:

```
set_ccopt_property -skew_group ck200m/func target_skew 0.1ns
```

CTS will auto-generate skew targets where none are specified. These may not be optimal for all designs. Note that extreme-effort CTS will ignore skew targets, except where skew groups have been explicitly configured to restrict useful skew.

Creating the Clock Tree Specification

The recommended method for generating a clock tree specification is to use the command `create_ccopt_clock_tree_spec` . This will analyze the timing graph of all active setup and active hold analysis views and create the specification. For details on how the specification creation process operates and the commands used in the specification, see the Concepts and Clock Tree Specification section.

To write the specification to a file, instead of just applying it immediately in memory, add the `-file` parameter. This example writes the specification to a file and then loads the specification.

```
create_ccopt_clock_tree_spec -file ccopt.spec
```

```
source ccopt.spec
```

The specification file that is generated records the reasons for the constraint settings and provides information about which constraints result in which settings. This information is useful for analyzing the specification later, if required. In case the `-file` parameter is not specified, no information about the reasons for constraint settings will be stored.

Configuration Check

The command, `ccopt_design -check_prerequisites` can be used to perform setup, library, and design validation checks without running CTS. It is otherwise identical to the checks normally performed near the start of `ccopt_design`.

When `ccopt_design` cannot run because to some configuration problems, the software prints out tables listing the problems. An example output for a scenario when there are both design and clock tree configuration problems is shown below.

```
###############################################################
# Generated by: Cadence Innovus <version>
# OS: <OS>
# Generated on: Thu Sep 21 12:12:27 2017
# Design: nastyclock
# Command: ccopt_design
###############################################################

CCOpt configuration status: cannot run ccopt_design.
Check the log for details of problem(s) found:

---------------------------------------------------
Design configuration problems
---------------------------------------------------
Maximum source to sink net length is too small
One or more clock trees have configuration problems
Too many clock tree insts are locked
---------------------------------------------------

Clock tree configuration problems:

------------------------------------------------------
Clock tree Problem
------------------------------------------------------
divclk2 The maximum transition target is too low
divclk2 The selected drivers are too weak
------------------------------------------------------
divclk The maximum transition target is too low
divclk The selected drivers are too weak
------------------------------------------------------
clk The selected drivers are too weak
------------------------------------------------------
clk4 The selected drivers are too weak
```

Controlling Useful Skew Effort in CCOpt

The `-opt_skew_ccopt` parameter of `setOptMode` specifies the level of useful skew effort applied during the `ccopt_design` or `optDesign -postCTS` commands. The possible settings are:

- `none`: No useful skew is allowed.
- `standard`: Allows local modifications to the clock tree network during post-CTS optimization. This is the default setting when

`setDesignMode-flowEffort` is set to `standard`.

- `extreme`: Enables concurrent optimization of the clock tree and datapath to improve setup time, in addition to the standard effort modifications. This is the default when `setDesignMode -flowEffort` is set to `extreme`.

Note: Useful skew effort level `extreme` is only available in the `ccopt_design` command. Invoking effort level `extreme` will significantly increase the overall runtime. Before invoking effort level `extreme`, ensure that you have achieved good ideal mode timing results. It is recommended that you deploy the Early Clock Flow before invoking effort level `extreme`.

Common Specification Modifications

Stop and Ignore Pins

Stop pins and ignore pins both stop the clock tree specification process from tracing beyond a pin. A stop pin is treated as a sink to be balanced whereas an ignore pin is not balanced. For details about stop and ignore pins, see the Clock Tree Sink Pin Types section.

Macro Clock Input Pins

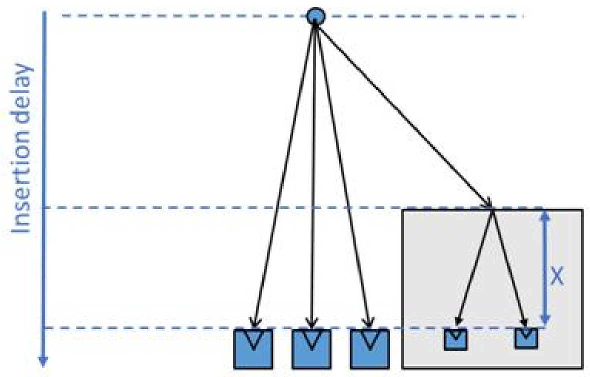
The clock input pins of macros (.lib model) must usually be earlier than other sinks, which means they will have a lesser clock arrival time to take account of the internal clock path inside the macro. If this is represented by a pin specific network latency, `set_clock_latency`, command in the SDC timing constraints then the automatically-generated clock tree specification will take this into account. This is discussed further in the Network Latencies section.

If the clock offset at a macro clock pin is not captured in the timing constraints, then you must add this. For example:

```
set_ccopt_property -pin mem1/CK insertion_delay 1.2ns
```

Note that the property setting is the delay to be assumed inside the macro. Positive numbers will reduce the clock arrival time at the pin, negative numbers will increase it. This is illustrated in the following diagram, where X represents the property setting value.

Pin Insertion Delay



It is possible to set a pin insertion delay at any clock sink to adjust the skew of the sink relative to other sinks without such a setting. This can be used to implement manual or preCTS useful skew. Note that setting a pin insertion delay on large number of pins is not recommended and may increase clock area and power.

Architectural Clock Gates

An architectural clock gate is a clock gate typically very early (small insertion delay) in a clock tree that is used to enable and disable entire functions or logical partitions of a design. The flops controlling such a clock gate may also need to be scheduled early to avoid setup slack violations at the clock gate enable input. This can be achieved by adding an additional skew group to balance the flops with the clock gate. For an example of this, refer to the example, Balancing flops with a clock gate, in the Modifying Skew Groups section. Such additional configuration for architectural clock gates is frequently recommend with CCOpt, and will be essential for timing closure with CCOpt-CTS.

Restricting CCOpt Skew Scheduling

CCOpt will initially compute the maximum insertion delay over all skew groups. By default, CCOpt skew scheduling is restricted such that the insertion delay of any sink may not exceed some factor multiplied by the initially computed maximum insertion delay. This factor is set by the property, `auto_limit_insertion_delay_factor`, which defaults to 1.5. This permits useful skew scheduling to increase the global maximum insertion delay by up to 50%. Useful skew scheduling is unrestricted by how much it can decrease the insertion delay to a sink.

To change this restriction on late useful skew set the property. For example to lower the restriction to a 20% insertion delay increase:

```
set_ccopt_property auto_limit_insertion_delay_factor 1.2
```

To restrict the skew of a given skew group in CCOpt set the target_skew property on the skew group and set the constrains property of the skew group to include the keyword ‘ccopt’. For syntax details, see the Defining Skew Groups section. For example, to place a hard limit on the skew of all skew groups to 400ps, irrespective of the impact on timing closure, use the following commands:

```
foreach sg [get_ccopt_skew_groups *] {
set_ccopt_property target_skew 400ps -skew_group $sg
set_ccopt_property constrains all -skew_group $sg
}
```

Method

The recommended method for setting up CCOpt or CCOpt-CTS on a new design is to use the following steps:

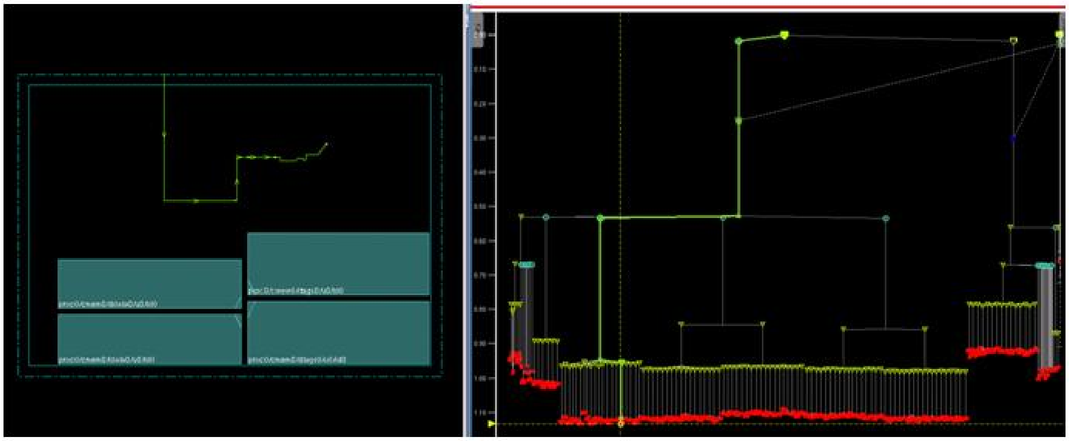
1. Configure and create the clock tree specification as per the Quick Start Example and configuration instructions above.
2. Before invoking the `ccopt_design` command use the CCOpt Clock Tree Debugger in unit delay mode to inspect the clock tree. This will permit examination of the clock tree structure. For more information, see the Unit Delay section.
3. Invoke only the clustering step of CCOpt or CCOpt-CTS which performs buffering to meet design rule constraints but does not perform skew balancing or timing optimization. Check the maximum insertion delay path looks sensible in the CCOpt Clock Tree Debugger. For designs with narrow channels, many blockages, or complex power domain geometries this is a good time to check for large transition violations caused by floorplan issues. The screenshot, "Cluster Maximum Insertion Delay", below shows the placement view (left) and the CCOpt Clock Tree Debugger view (right) with the maximum insertion path delay highlighted in green.
4. For a CCOpt flow with a simple clock tree, for example a CPU core, switch to using full `ccopt_design`. For a design with a complex clocking architecture consider using trial mode, which will perform clustering and then balancing using virtual delays. The trial balancing can be inspected to look for large skew or insertion delay increases due to conflicting skew group constraints. The design can be timed using `timeDesign -postCTS` to check for large timing slack violations, for example, due to incorrect balancing constraints. Virtual delays will appear in timing reports as additional arrival time increments.
5. Run full `ccopt_design`. Inspect the log file for errors and warnings. For CCOpt, a summary table of timing slack and other metrics at each stage of the `ccopt_design` internal flow is reported.
6. For CCOpt, check the worst chain report in the log. Note that there may be multiple worst chain reports in the log. It is recommend to look at the worst chain report after the last occurrence of skew adjustment before any re-clustering steps in the log, this is usually the second last chain report. This report will indicate if useful skew scheduling has hit constraint limits that are limiting optimization. For more information on the worst chain report, see the Worst Chain section .
7. Report on clock trees and skew groups. For example, it is recommended to check skew group maximum insertion delay and clock tree area even if setup timing slack is closed. For more information, see the Reporting section.

As mentioned above, CCOpt and CCOpt-CTS can be configured between `cluster`, `trial`, or `full` mode using the `balance_mode` CCOpt property.

```
set_ccopt_property balance_mode cluster | trial | full
ccopt_design -cts
```

The default is `full` mode. The concepts of clustering and trial virtual delay balancing are detailed further in the Graph-Based CTS section.

Cluster Maximum Insertion Delay



Flexible H-Tree and Multi-Tap Clock Flow

A structured top of tree clock distribution scheme is typically deployed to improve cross-corner scaling in combination with large drivers and top of stack low RC delay routing layers to reduce clock latency. The most common such structure is an H-tree. The Innovus flexible H-tree feature provides the electrically symmetric buffering and balanced wire length benefits of an H-tree, but relaxes the requirement to be geometrically symmetric, therefore, enabling automated synthesis even in floorplans with placement restrictions. Multi-Tap Clock Tree Synthesis, also known as Multi-Source Clock Tree Synthesis, is fully integrated with the flexible H-tree feature and extends regular clock synthesis to provide local buffering and balancing between the structured top of tree and the clock sinks.

The flexible H-trees are mostly created and synthesized using the following two-step flow:

- Create each H-Tree one at a time with the `create_ccopt_flexible_htree` command.
- Synthesize all created H-Trees with the `synthesize_ccopt_flexible_htrees` command. H-Tree synthesis includes adding all H-Tree instances to the netlist, placing them, and performing detailed routing of all H-Tree nets. Some of you may want to customize the design – typically by adding or widening power/ground wires – after the H-Tree instances are placed but before the H-Tree nets are detail routed. To do this, you can deploy the following three-step flow:
  1. Create each H-Tree one at a time with the `create_ccopt_flexible_htree` command.
  2. Synthesize all created H-Trees with estimated routes using `synthesize_ccopt_flexible_htrees -use_estimated_routes`.
  3. Perform detailed routing of all H-Tree nets with the `route_ccopt_flexible_htrees` command.

Note: When using the `route_ccopt_flexible_htrees` command, the software expects the input routes of nets in the flexible H-tree to be fully connected, as it is the case for routes generated by `synthesize_ccopt_flexible_htrees -use_estimated_routes`. If this expectation is not met, then unexpected results such as open nets may occur and you should check for connectivity-related warnings in the log file.

For details of the use model for this feature, see the Flexible H-tree and Multi-Tap Clock Flow in Innovus Application Note on the Cadence Online Support website.

Concepts and Clock Tree Specification

Graph-Based CTS

CCOpt-CTS and CCOpt use a graph-based CTS algorithm to perform skew balancing (CCOpt-CTS) or initial seed balancing (CCOpt). The main internal steps in this are as follows:

- **Clustering** – This step groups nearby clock sinks into clusters and buffers clock trees to meet maximum transition, capacitance, and length constraints, such as DRV (Design Rule Violation) constraints. After clustering, the maximum insertion delay is approximately known.
- **Constraints analysis and virtual delay balancing** – Constraints analysis identifies how the balancing constraints (skew and insertion delay constraints) interact and identifies where delay should be added to the clock graph to best meet these constraints. For example, a common scenario is identifying where to add delay to balance test mode clock skew without impacting functional mode clock insertion delay. This happens automatically without any user intervention or need for user-driven sequential steps. Virtual delay balancing is simply the process of annotating clock nodes in the timing graph with additional delay that is added to the propagated clock arrival time to achieve the solution found by constraints analysis. CCOpt uses both clustering and virtual delays to initially balance clocks to obtain initial propagated mode timing and to permit run-time efficient what-if style analysis during useful skew scheduling.
- **Implementation** – This step synthesizes virtual delays using real physical cells. It is followed by a refinement process to account for the difference between virtual delays and those achievable with physical cells. For CCOpt-CTS, this is essentially the last step. For CCOpt, further post-CTS style datapath optimization is performed.

The use of this graph-based CTS approach, combined with the multi-mode clock specification generated by the `create_ccopt_clock_tree_spec` command enables CCOpt-CTS and CCOpt to cope with large complex clock structures, typically, with zero or minimal user intervention.

Clock Trees and Skew Groups

Clock trees and skew groups are the two key object types used in the CCOpt clock specification. The term object is used here because clock tree and skew group objects can be defined, modified, and deleted using commands. For example, `create_ccopt_clock_tree` , `create_ccopt_skew_group` , `modify_ccopt_skew_group` , and `delete_ccopt_skew_groups` .

Properties can be set per skew group or clock tree instead of globally.

For example,

```
set_ccopt_property  -skew_group name  target_skew value
```

The `report_ccopt_clock_trees` and `report_ccopt_skew_groups` commands can be used to generate reports on clock trees and skew groups. For more information, see the Reporting section.

Clock Trees

- The union of all clock trees specifies the subset of the circuit graph that CTS will buffer. The circuit subset covered by clock tree definitions is best referred to as a clock tree graph since clock trees may interact, for example via clock logic cells. The clock tree graph is a single physical graph even in a multi-mode timing environment.
- Maximum transition times, route types and other physical properties are associated with the clock tree graph or with individual trees in the clock tree graph.

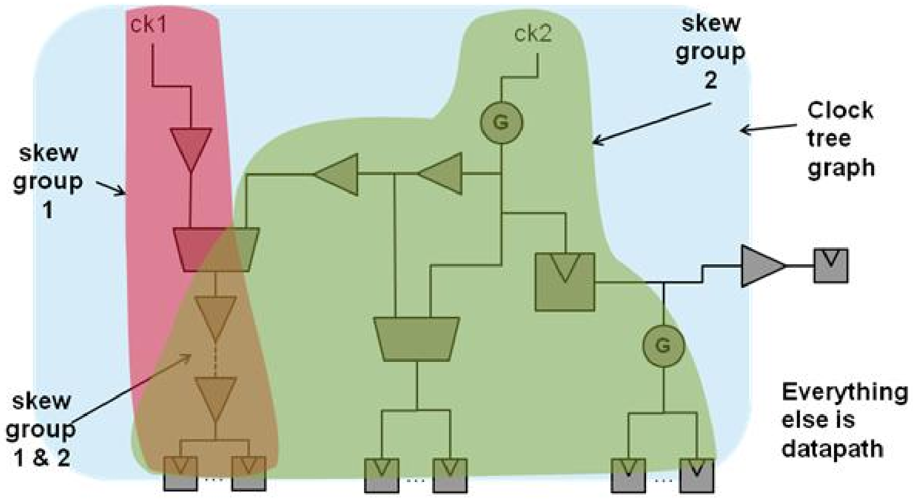
- In all but rare exceptional circumstances, the clock tree definitions created by `create_ccopt_clock_tree_spec` do not require user modification.

Skew Groups

- A skew group represents a balancing constraint and is the CTS equivalent of an SDC clock. The automatically generated clock tree specification will create one skew group per SDC clock per mode.
- Each skew group has one or more sources and a number of sinks. Among other properties, a skew target and insertion delay target can be set per skew group. Any pin in the clock tree graph can be a skew group source or sink and pins can be designated a skew group specific ignore pin such that the specific skew group does not propagate beyond the pin.
- CCOpt-CTS global skew balancing aims to achieve an equal delay, subject to the skew target, from all sources to all sinks within each skew group. CCOpt virtually balances skew groups to zero skew to determine initial clock tree timing with propagated clocks before optimization starts.
- A skew group can be viewed as a subset of the clock tree graph superimposed on top of the clock tree graph. Skew groups can overlap, share sources, and/or sinks.
- In complex cases or with CCOpt-CTS where the SDC timing constraints do not fully capture the balancing requirements, user adjustment to the skew group configuration may be required and/or additional skew groups can be defined.

The diagram below illustrates the relationship between the clock tree graph and skew groups. Note the path to the data input of a flip-flop at the right hand side, the clock tree graph is ‘pruned back’ to exclude this path, the input to the right most buffer will be an ignore pin – clock pin types are discussed later.

Clock Tree Graph and Skew Groups



Pin Insertion Delays

A pin insertion delay (PID) is a skew group balancing constraint that is asserted at some clock pin. The PID value affects how skew group paths that sink at that pin are timed by the CTS clock timer. A given PID value only has influence if it is asserted against the sink of some skew group. A PID value that is not present at the sink of a skew group has no effect on the clock timer.

Pin Insertion Delay Keys

The full qualification of a PID value is as follows:

- The pin against which it is asserted
- The timing half corner that is applicable, commonly the CTS primary half corner
- The clock edge: usually both rise and fall edges

An additional element to the key is the category that will allow you to record the origin of the various PID values present at a given pin.

When computing the latency of a given skew group sink, the clock timer will search for a PID value with matching qualification. If one is found, it is incorporated into the latency calculation.

Pin Insertion Delay Categories

A given pin may have its PID changed by multiple factors. Therefore, it is not sufficient to record a category alongside the PID value. Instead, different portions of the PID value must be allotted to different categories. Each factor will manipulate only the relevant portion of the PID. The total PID will be computed as an algebraic sum of the constituent portions. The total PID is what the clock timer will use.

The software allows categorization of the PID at a pin, tagging portions of the PID values by their origin (for example, user, SDC, useful skew, and so on).

The following categories of PID have been defined:

- `user`: The PID value was asserted by user Tcl scripting or has been restored from a saved CCOpt state. This is the default category. This means that if a PID value is configured and that value is not expressly categorized, it should appear under the user category.
- `sdc`: The PID value was extracted from the SDC set_clock_latency by spec creation.
- `ilm`: The pin carrying this PID value has been identified as an “ILM stop pin” by spec creation. The PID value records the mean clock latency inside the ILM.
- `useful_skew`: A PID value generated by pre-CTS early clock flow (ECF) useful skewing.
- `total`: The overall PID value that is presented to the clock timer. It is the formal sum of the PID values from the other categories.

Properties for Pin Categorization

PID categorization is reflected through the following pin property that exposes the categorization as a Tcl dictionary:

```
insertion_delay_sources
```

It specifies the amount of insertion delay under this pin, broken down by the source of the PID. The value of this property is a Tcl dictionary of PID values, keyed on the source of that PID. This provides an 'exploded' view into the total PID that is present, categorizing the various contributions by their origin.

For detailed description of the property in the software, use the following command:

```
get_ccopt_property -help insertion_delay_sources
```

The key for this property is identical to that of the `insertion_delay` property; it will have exploded names that allow for qualification by early/late rise/fall and total/wire. Each member of the family will be indexed on delay corner. This property is used to manipulate the global (current skew group mode) PID values for the given pin.

The `insertion_delay` property operates on the “total” PID value.

Querying the insertion_delay property returns the total PID value. Setting the insertion_delay property sets the total PID to the specified value. To accomplish this:

- All categories are reset to auto.
- The **user** category is updated with the specified value.
- The **total** PID value is recalculated.

For details of the properties, see CCOpt Properties.

Reporting for Specific PID Categories

When `report_ccopt_pin_insertion_delays -sources {string1 string2 ...}` parameter is specified, it restricts the PID report to those skew group sink pins that have a PID value from one of the listed sources. This `-sources` list of categories takes wildcard values that are expanded to matching categories.

For example:

```
report_ccopt_pin_insertion_delays -sources u*
```

This is equivalent to:

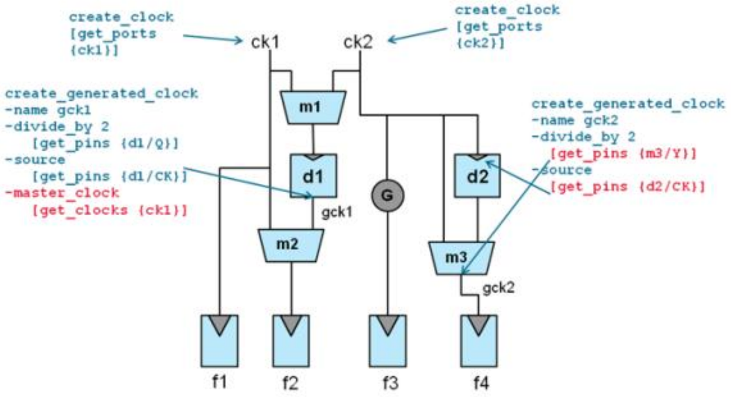
```
report_ccopt_pin_insertion_delays -sources {useful_skew user}
```

Automatic Clock Tree Specification Creation

Single Mode Example

The diagram below shows a single constraint mode example with two clocks, some multiplexers, and two clock dividers. The SDC clock definitions are illustrated.

Note the precise definitions of the generated clocks carefully.
Single Mode Example – SDC Clock Definitions



```
create_clock
[get_ports
{ck1}]
```

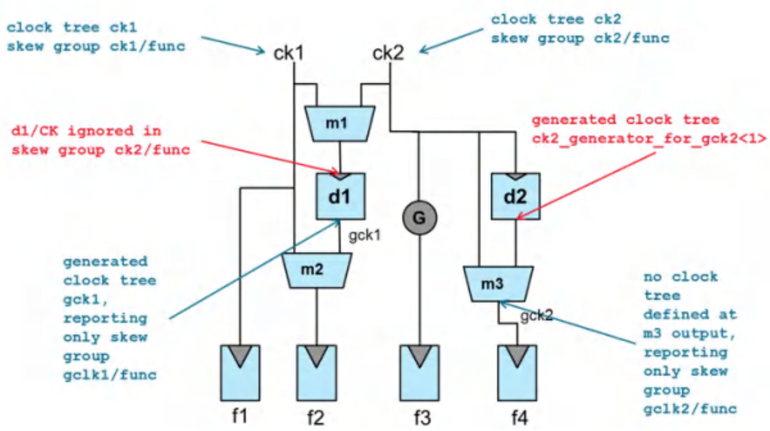
```
create_clock
[get_ports
{ck2}]
```

```
create_generated_clock
-name gck1
-divide_by 2
   [get_pins {d1/Q}]
-source
   [get_pins {d1/CK}]
-master_clock
   [get_clocks {ck1}]
```

```
create_generated_clock
-name gck2
-divide_by 2
   [get_pins {m3/Y}]
-source
   [get_pins {d2/CK}]
```

On the left side, the generated clock gck1 refers to master ck1 such that ck2 does not propagate to f1 or f2. On the right side, the definition of gck2 is such that the path from d2/CK to m3/Y is considered part of the clock generator circuit. Both these points have implications for the resulting clock tree specification output that is annotated in the diagram below.

Single Mode Example – Clock Tree Specification



In the generated specification, a clock tree is defined at each of the primary inputs ck1 and ck2.

On the left side, a generated clock tree is defined at the output of divider d1 to distinguish d1 as a sequential element in the clock graph. Without this generated clock tree definition CTS would treat d1 as a regular sink. Additionally, at the output of divider d1 a skew group `gck1/func` is defined, but note that this skew group is non-constraining so does not influence CTS. It is present purely for reporting purposes. Sinks f1 and f2 are balanced together by skew group `ck1/func`. Skew group ck2/func is ignored at the input to d1, this corresponds to the master_clock specification in the SDC.

On the right side, no generated clock tree is defined at the output of multiplexer m3, since m3 is a combinational cell. However, a non-constraining skew group is defined at the output of multiplexer m3 for reporting purposes. So that CTS does not treat divider d2 as a regular clock sink and so that the path from d2 to m3 is included in the clock tree graph, a generated clock tree is defined at the output of d2.

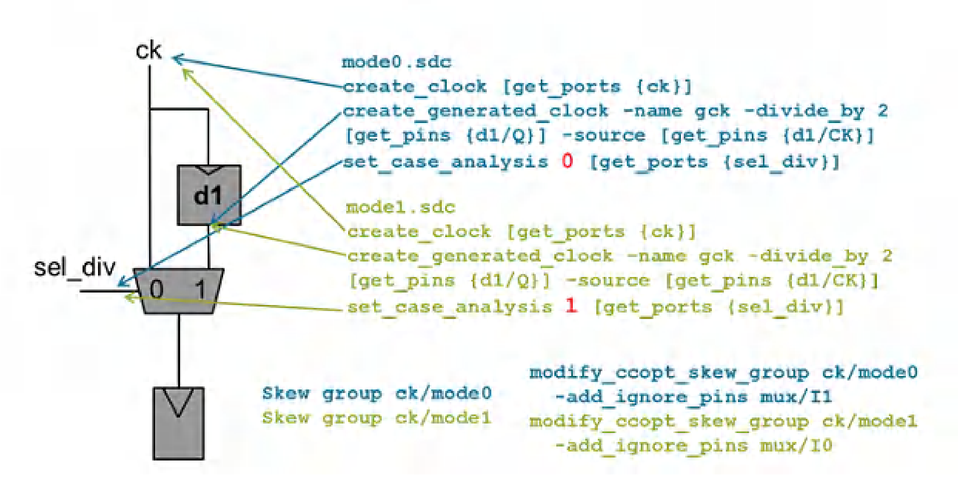
Key lines from the output of `create_ccopt_clock_tree_spec -file ccopt.spec` for the example are given below.

Single Mode Example – create_ccopt_clock_tree_spec Output

```
create_ccopt_clock_tree -name ck2 -source ck2 -no_skew_group
create_ccopt_generated_clock_tree -name ck2_generator_for_ck2<1> -source d2/Q -generated_by d2/CK
create_ccopt_clock_tree -name ck1 -source ck1 -no_skew_group
create_ccopt_generated_clock_tree -name gck1 -source d1/Q -generated_by d1/CK
create_ccopt_skew_group -name ck1/func -sources ck1 -auto_sinks
create_ccopt_skew_group -name ck2/func -sources ck2 -auto_sinks
modify_ccopt_skew_group -skew_group ck2/func -add_ignore_pins d1/CK
create_ccopt_skew_group -name gck1/func -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck1/func none
create_ccopt_skew_group -name gck2/func -sources m3/Y -auto_sinks
set_ccopt_property constrains -skew_group gck2/func none
```

Multi-Mode Example

The diagram below shows a simple multi-mode example annotated with SDC constraints and skew group information.



```
mode0.sdc
create_clock [get_ports {ck}]
create_generated_clock -name gck -divide_by 2
[get_pins {d1/Q}] -source [get_pins {d1/CK}]
set_case_analysis 0 [get_ports {sel_div}]
```

```
mode1.sdc
create_clock [get_ports {ck}]
create_generated_clock -name gck -divide_by 2
[get_pins {d1/Q}] -source [get_pins {d1/CK}]
set_case_analysis 1 [get_ports {sel_div}]
```

```
Skew group ck/mode0
Skew group ck/mode1
```

```
modify_ccopt_skew_group ck/mode0
  -add_ignore_pins mux/I1
modify_ccopt_skew_group ck/mode1
  -add_ignore_pins mux/I0
```

The resulting specification contains the following:

- Two clock trees, ck and gck. The clock tree definitions tell CTS which parts of the circuit are included in the clock tree graph and are not mode specific.
- Two skew groups, `ck/mode0` and `ck/mode1`. The skew groups tell CTS how to perform balancing.
- Each skew group has an ignore pin defined at the appropriate multiplexer input. This represents the fact that there is no need to balance the direct clock path with the divided clock path as the paths are never active in the same mode at the same time.

Key commands from the specification are listed below. Some details have been omitted for clarity.

Multi-Mode Example – create_ccopt_clock_tree_spec Output

```
create_ccopt_clock_tree -name ck -source ck -no_skew_group
create_ccopt_generated_clock_tree -name gck -source d1/Q -generated_by d1/CK
create_ccopt_skew_group -name ck/mode0 -sources ck -auto_sinks
create_ccopt_skew_group -name ck/mode1 -sources ck -auto_sinks
modify_ccopt_skew_group -skew_group ck/mode0 -add_ignore_pins mux/I1
modify_ccopt_skew_group -skew_group ck/mode1 -add_ignore_pins mux/I0
create_ccopt_skew_group -name gck/mode0 -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck/mode0 none
create_ccopt_skew_group -name gck/mode1 -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck/mode1 none
```

SDC Transition Targets

The `create_ccopt_clock_tree_spec` command will translate `set_max_transition` constraints. For example, consider the SDC constraint “`set_max_transition 0.123 [get_clocks {ck1}]`”.

The automatically generated specification will contain the following line:

```
set_ccopt_property target_max_trans_sdc -clock_tree ck1 0.123
```

CCOpt-CTS and CCOpt will look at the `target_max_trans` property. If this is set to the default value of `auto,` then the `target_max_trans_sdc` will be inspected. If `target_max_trans_sdc` is not set, then an automatic default will be computed.

Network Latencies

The create_ccopt_clock_tree_spec command will translate clock network latency settings to an insertion delay target on the corresponding skew group. For example, consider the functional mode SDC constraint, “set_clock_latency 1.456 [get_clocks {ck1}]”. The automatically generated specification will contain the following line:

```
set_ccopt_property target_insertion_delay -skew_group ck1/func 1.456
```

Similarly, pin network latency settings are translated to the insertion_delay property of a pin. This property is often referred to as a pin insertion delay. A pin insertion delay represents the delay ‘underneath’ a clock sink. For example, for a macro clock input pin, the pin insertion delay would represent the internal clock path delay inside the macro. Continuing the above example, add the constraint “ set_clock_latency 0.234 [get_pins {mem1/CK}] ”. The automatically generated specification will additionally contain the following line:

```
set_ccopt_property  insertion_delay  -pin mem1/CK 1.222
```

The property setting indicates that the delay internal to the macro clock input mem1/CK is 1.222 . The value 1.222 is computed as the difference between the clock latency of 1.456 and the pin latency of 0.234 . Note that SDC pin-specific latencies override clock latencies, which means they are not added together.

Clock Tree Convergence

In some circumstances, the clock tree graph undesirably propagates into datapath and includes what should be datapath as part of the clock tree graph. For example, this can happen due to missing set_case_analysis or other incorrect SDC constraints. Including significant datapath logic as part of the clock tree graph can result in excessive CCOpt or CCOpt-CTS run time due to large numbers of paths existing between a skew group source and sink due to multiple levels of re-convergent logic. Additionally, such paths would not be optimized by datapath optimization.

To help detect cases where run time would be adversely affected, the automatically generated clock tree specification includes an invocation of the check_ccopt_clock_tree_convergence command. This command traces the number of paths to every sink and issues a warning if the number of clock paths to any sink is greater than a default threshold of 100 paths. The report_ccopt_clock_tree_convergence command can be used to report sinks with large numbers of clock paths.

To remedy this situation, either correct the SDC constraints, for example by adding set_case_analysis , set_clock_sense -stop_propagation or other suitable commands, or use a clock tree stop, ignore, or exclude pin as appropriate. These pins are described in the next section.

Clock Tree Sink Pin Types

Clock tree sink pin types can be manually overridden before invoking the `create_ccopt_clock_tree_spec` command with the following property setting:
`set_ccopt_property -pin` *pin_name* sink_type ignore | stop | exclude

Note: Set the `sink_type` property before creating the clock tree specification, otherwise it will not take effect.

Adding an ignore pin before the clock tree specification stops clock tree propagation through the pin and removes the pin from skew balancing. If this pin is not considered a clock tree sink, the [transitive fanout] clock tree below this pin will not be defined and build as a clock tree.

However, adding the same ignore pin after creating the clock tree specification will only mark the pin as an ignore pin and if this pin is not considered a clock tree sink, the clock tree below this pin will remain and be build as a clock tree.

For more information, see the following troubleshooting article at http://support.cadence.com:

- Should I define ignore, exclude and stop pins before or after creating my CCOpt spec

Ignore pin (ignore)

An ignore pin is considered as a part of the clock tree graph. CTS will perform DRV buffering up to the pin, but the pin will not be considered as a sink in any skew group, which means the latency to an ignore pin is not important. Tracing through and beyond the pin will be disabled. Sometimes such a pin is referred to as a clock tree ignore pin. An alternative strategy to deploying an ignore pin would be to use the SDC constraint, `set_clock_sense -stop_propagation`. This may be preferable since it would keep the timing model in synchronization with the CTS configuration.

Note: If an ignore pin is added beyond the pins where clock phases would propagate, clock tree spec creation attempts to find an unbroken timing path to the ignore pin and extend the clock tree network to the pin, which means the software ignores the SDC such as `set_clock_sense -stop_propagation`. If these downstream `sink_type` settings are removed, the software stops the clock tree where CTE says the SDC clock stops.

Stop pin (stop)

A stop pin is considered as a part of the clock tree graph. CTS will perform DRV buffering up to the pin and by default the pin will be considered a sink to be balanced in any skew group that reaches the stop pin. Tracing through and beyond the pin will be disabled.

Note: If a stop pin is added beyond the pins where clock phases would propagate, clock tree spec creation attempts to find an unbroken timing path to the stop pin and extend the clock tree network to the pin, which means the software ignores the SDC such as `set_clock_sense -stop_propagation`. If these downstream `sink_type` settings are removed, the software stops the clock tree where CTE says the SDC clock stops.

Exclude pin (exclude)

An exclude pin is a pin that is not part of the clock tree graph but might still be connected to a clock net anyway if the same net has other clock fanout. Specifically, the clock tree graph must not extend beyond an exclude pin but it can be pruned back from an exclude pin. The `create_ccopt_clock_tree_spec` command will prune back from an exclude pin and, if possible, specify an ignore pin earlier in the fanin cone. The Shared Clock and Data Concerns section discusses how to add buffers to disconnect exclude pins from any clock tree nets they may be connected to. This can be important where clock and datapath overlap.

Note: In addition to the above pin types, it is possible to make any pin that is within the clock tree graph a skew group specific ignore or sink pin. This is discussed in the subsequent sections.

Manual Setup and Adjustment of the Clock Specification

Following are some important recommendations for setting up and adjusting the clock tree specification:

- It is recommended that the `create_ccopt_clock_tree_spec` command is used to create the clock tree specification.
- It is not recommended to edit the specification file generated by the `create_ccopt_clock_tree_spec -file` *`filename`* command. A more stable flow is obtainable either by adjusting the SDC, configuring CCOpt properties, setting clock tree sink pin types before generating the specification, or making skew group adjustments after loading the specification.
- Consider making adjustments to the SDC timing constraints instead of the CTS specification, if applicable. This will ensure that timing analysis uses a clock propagation model consistent with the CTS configuration. For example, setting a clock logic instance input pin to be a clock tree ignore pin will stop CTS tracing through the pin, but will not stop `report_timing` from propagating a clock through the pin. The `create_ccopt_clock_tree_spec` command has been engineered to create a clock tree specification consistent with the active timing constraints.

The table below shows commonly used commands for manipulating clock trees and skew groups:

Command Name	Usage
`create_ccopt_clock_tree`	Adds a new clock tree definition into the in memory clock tree specification.
`delete_ccopt_clock_trees`	Removes a clock tree definition from the in memory clock tree specification.
`create_ccopt_skew_group`	Adds a new skew group definition into the in memory clock tree specification.
`delete_ccopt_skew_groups`	Removes a skew group definition from the in memory clock tree specification.
`modify_ccopt_skew_group`	Permits adjustment to sinks and ignore pins of a skew group.
`set_ccopt_property -skew_group` *`skew_group_name property_name value`*	Adjusts skew group specific properties. The most commonly adjusted properties are `target_skew` and `target_insertion_delay`.

Note: The above commands do not modify the design or perform any CTS but manipulate the in-memory clock tree specification.

Defining Clock Trees

Clock trees are defined using the `create_ccopt_clock_tree` and `create_ccopt_generated_clock_tree` commands. For example:

```
create_ccopt_clock_tree -name ck -source ck -no_skew_group
create_ccopt_generated_clock_tree -name gck -source d1/Q -generated_by d1/CK
```

The optional `-name` parameter can be used to specify the name of the clock tree. Alternatively, the source pin name will be used as the clock tree name. The mandatory `-source` parameter specifies the clock tree root pin from which clock tree tracing will be performed. The `-no_skew_group` parameter disables the automatic creation of a corresponding skew group, otherwise a skew group with the same name as the clock tree is automatically created. In addition, the definition of a generated clock tree requires the `-generated_by` parameter to specify the input side of the clock generator, which is typically the clock input pin of a divider flip-flop.

When a clock tree is defined, CCOpt traces the circuit connectivity from the specified source pin, adding the nets and cell instances it encounters to the clock tree graph. Tracing continues until encountering a clock pin (such as a flip-flop, latch, or macro input), a user-defined stop, ignore, or exclude pin. A generated clock tree definition must normally be used at the output of a sequential cell to continue tracing.

Defining Skew Groups

Skew groups are defined using the `create_ccopt_skew_group` command. The complete syntax of this command is detailed below:

```
create_ccopt_skew_group
[-help]
[-constrains cts | ccopt_initial | ccopt | subset_of_values | all | none]
[-from_clocks clock_names]
[-from_constraint_modes constraint_mode_names]
[-from_delay_corners delay_corner_names]
-name skew_group_name
[-rank rank]
[-sinks pins | -shared_sinks pins | -exclusive_sinks pins | -auto_sinks | -filtered_auto_sinks pins | -
balance_skew_groups skew_groups]
[-sources pins | -balance_skew_groups skew_groups]
[-target_insertion_delay value]
[-target_skew value]
```

Note: The parameters taking a list of pins operate with either a plain TCL list of hierarchical pin names or with a collection of pins obtained from the `get_pins` command.

Skew Group Rank

The rank of a skew group determines whether a sink pin is an active sink in that skew group or not. A pin is only an active sink in the skew group(s) with the highest rank out of all the skew groups to which the pin belongs. An active sink is a pin that will be balanced against other active sinks in the same skew group.

For example, consider the following sequence of commands:

```
create_ccopt_skew_group -name SG1 -sources get_pins top -shared_sinks [get_pins */D]
create_ccopt_skew_group -name SG2 -sources get_pins top -exclusive_sinks [get_pins *XYZ*/D]
create_ccopt_skew_group -name SG3 -sources get_pins top -exclusive_sinks [get_pins *XYZ_01*/D]
```

The rank of a skew group can be accessed via the `exclusive_sinks_rank` property.

Finding Active Skew Group Sinks

Use the following command to find all the sink members of a skew group:

```
get_ccopt_property -skew_group name sinks
```

Use the following command to find all the active sink members of a skew group:

```
get_ccopt_property -skew_group name sinks_active
```

Use the following command to find all the skew groups for which a pin is a sink member:

```
get_ccopt_property -pin name skew_groups_sink
```

Use the following command to find all the skew groups for which the pin is an active sink:

```
get_ccopt_property -pin name skew_groups_active_sink
```

Use the following command to find all the skew groups which are active at a pin, either passing through the pin or for which the pin is an active sink:

```
get_ccopt_property -pin name skew_groups_active
```

Note : In debugging CCOpt-CTS skew or CCOpt initial balancing, the ‘active’ properties above should be used, since these reflect the constraints CTS will respect. For example, if a pin is configured as a sink of skew group but the skew group does not propagate to the pin due to a lack of connectivity, the pin will not be an active sink of the skew group. After defining skew groups or modifying existing skew groups it is recommended to invoke the `report_ccopt_skew_groups` or `ccopt_design` command to ensure that the CTS timer is updated before checking the active sinks properties.

Modifying Skew Groups

The `modify_ccopt_skew_group` command is used to make changes to the sink and ignore pins associated with a skew group. The syntax of the command is provided below.

```
modify_ccopt_skew_group
[-help]
[-make_exclusive]
-skew_group skew_group_name
[-add_sinks pins | -remove_sinks pins]
[-add_ignore_pins pins | -remove_ignore_pins pins]
```

The `-add_sinks` and `-remove_sinks` parameters are used to add and remove sinks. The `-add_ignore_pins` and `-remove_ignore_pins` parameters are used to add and remove ignore pins, and are discussed below.

The `set_ccopt_property` command can be used to modify properties of a skew group, including the `target_insertion_delay`, `target_skew`, and `constrains` properties.

Skew Group Ignore Pins

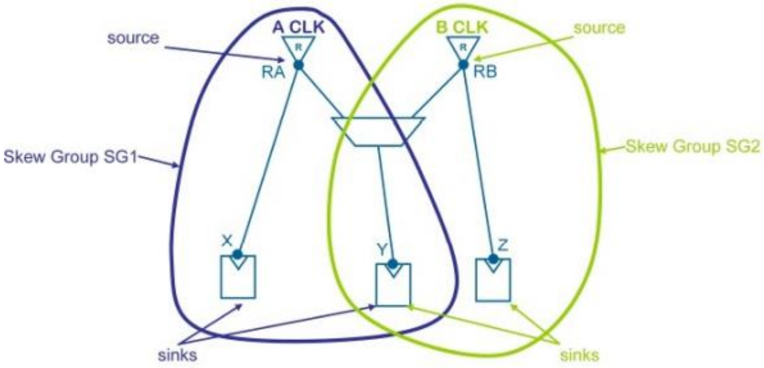
Specifying a pin as an ignore pin of a skew group stops CTS from considering the latency to that pin in that specific skew group, and stops that specific skew group propagating through and beyond that pin. Other skew groups at the pin are not affected. Skew group ignore pins are always applicable regardless of the skew group rank.

For example, if a leaf flip-flop clock pin is specified as a skew group ignore pin, CTS will not balance that flip-flop with other sinks for the same skew group. Balancing of other skew groups, possibly involving the same pins, would not be affected.

If a non-leaf pin is specified as a skew group ignore pin, for example a multiplexer input, CTS will ignore both the latency to and through that multiplexer input in the given skew group. Other skew groups passing through the same multiplexer input would not be affected. In such an example, any flip-flops in the fanout of the multiplexer would cease to be active sinks of the skew group.

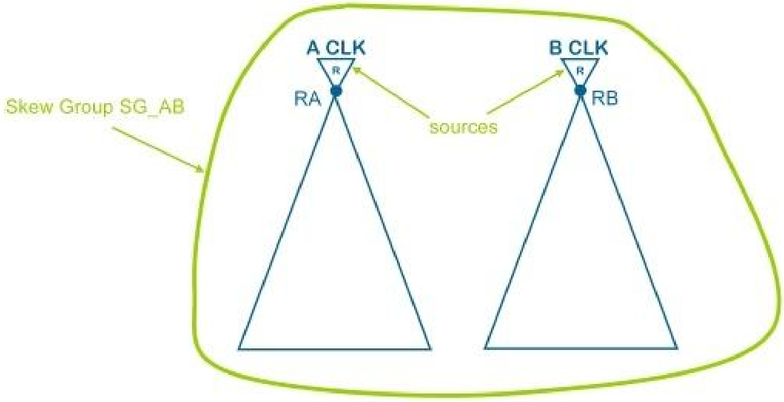
Example – Overlapping Skew Groups

The diagram below illustrates an example with two clock trees, A and B, with corresponding skew groups, SG1 and SG2. The sink Y is an active sink of both skew group SG1 and skew group SG2. Sink X and Y are balanced together and sink Y and Z are balanced together. The insertion delay difference between X and Z is not constrained. Constraint analysis during CTS will identify the most efficient place to put this delay, which in may be at the multiplexer inputs. For example, to add delay to balance the B-Y path with the B-Z path, delay can be added at the right hand multiplexer input without increasing the insertion delay of the A-Y path.



Example – Balancing Independent Clock Trees

The next example below again has two clock trees, A and B, but with a single skew group, SG_AB. The skew group has two sources and all the sinks of clock tree A and clock tree B. CTS will balance all paths from A to the sinks and all paths from B to the sinks together.



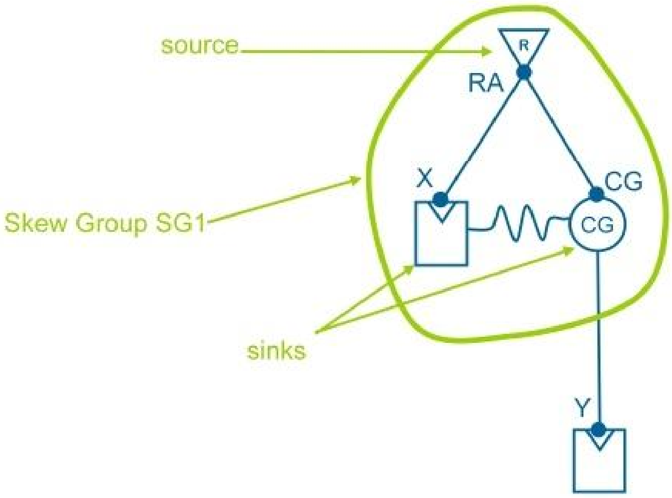
A variant of the above example would also have skew group, SG_A and skew group, SG_B corresponding to each of the two clock trees, which would be the default behavior of automatic clock tree specification. The user could then use `create_ccopt_skew_group -name SG_AB -balance_skew_groups {SG_A SG_B}` to create a combined skew group.

Example – Balancing Flops with a Clock Gate

When using CCOpt-CTS it may be necessary to reduce the clock insertion delay to sinks at the source of paths to a clock gate to avoid a setup violation at the clock gate enable input. In the example below, this is done by creating an additional skew group, SG1, to balance the flip-flop pin X with the clock gate pin CG as follows:

```
create_ccopt_skew_group -name SG1 -sources RA -exclusive_sinks {X CG}
```

The pins, X and CG, are made exclusive sinks so that X is no longer an active sink in other existing skew groups.



With CCOpt, rather than CCOpt-CTS, an additional user skew group would not normally be required to do this as useful skew scheduling will automatically adjust the insertion delay of X and CG to optimize the setup slack at the clock gate enable input.

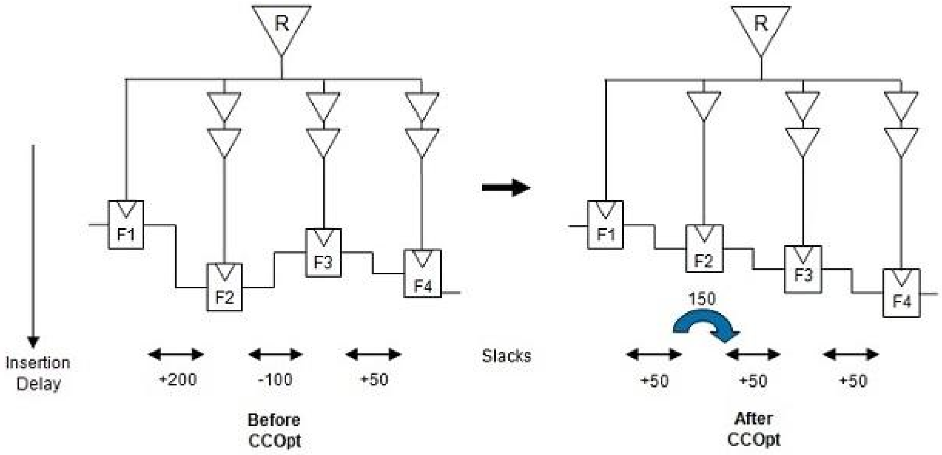
Deleting the Clock Tree Specification

The `delete_ccopt_clock_tree_spec` command can be used to remove all skew groups, clock trees, and associated data. However, this command does not reset property settings on pins, instances and other database entities. The `reset_ccopt_config` command can be used to remove both the clock tree specification and all CCOpt property settings. However, you can preserve the pin insertion delays on clock tree sinks by specifying the `-preserve_sink_insertion_delays` parameter provided in both these commands.

Chains

CCOpt uses useful skew to adjust clock delays, therefore, moving slack between datapath stages. The limit of WNS optimization is not determined by a single flop-to-flop datapath stage but a chain of such paths. At each flop slack can be shifted from the capture or launch side as illustrated below.

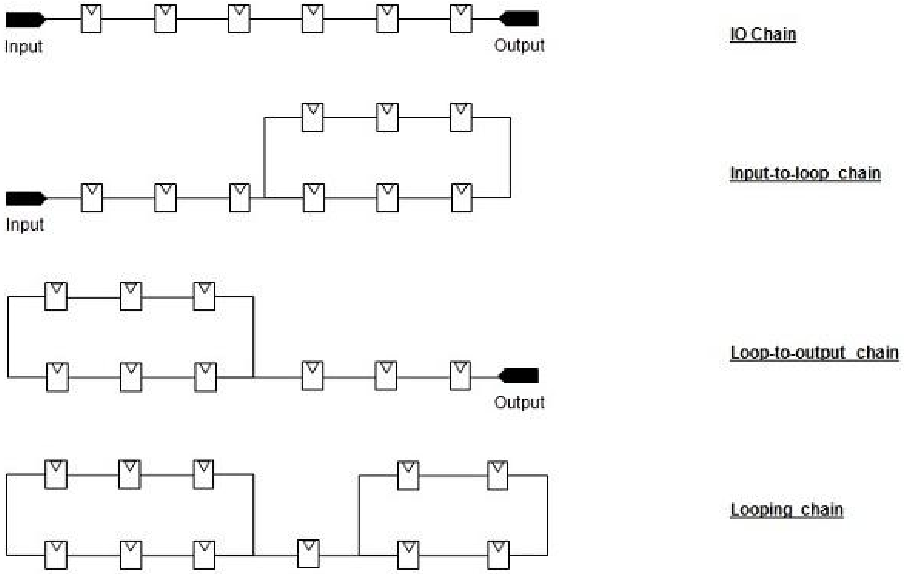
Moving Slack between Datapath Stages



In the example above, CCOpt moves 150ps of slack from the F1→F2 path and moves it to the F2→F3 path to address the negative slack of -100ps. This is done by reducing the delay on the F2 clock path, illustrated by the removal of a buffer in the above simplified diagram. However, the ability to move slack between datapath stages is not unlimited. It must stop when the chain of paths either loops back on itself or reaches an input or output port. This gives rise to different types of chains:

- Input-to-output chain – a chain of flops starting at an input pin and ending at an output pin
- Input-to-loop chain – a chain of flops starting at an input pin and ending at a looped path
- Loop-to-output chain – a chain of flops starting at a looped path and ending at an output pin
- Looping chain – a chain of flops starting and ending at a looped path

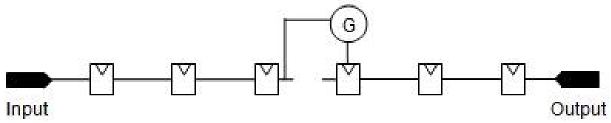
The different types of chains are shown below.



A variant of the input and output chains is a chain containing a flop which either does not launch or does not capture paths, for example if such paths are subject to `set_false_path` exception.

Chains can contain clock gates as illustrated below. CCOpt can adjust the clock insertion delay both to the clock gate and the flops during useful skew scheduling. Adjusting the clock insertion delay to a clock gate may impact the insertion delay to the gated flops, which in turn impacts the slack of timing paths launched by those flops.

Clock Gate in a Chain

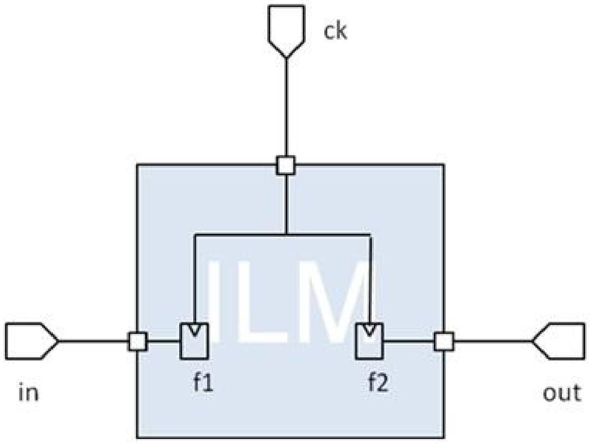


The design worst chain is the chain constructed by taking the global WNS path and expanding the chain around this path such that each path within the chain is a local WNS path. The worst chain is reported in the log during `ccopt_design`, and the format of this chain report is discussed further in the "Worst Chain" section.

Disjoint Chains

The worst chain may pass through an ILM partition or "`.lib`" macro. The example below illustrates an ILM partition in which a single clock input clocks both an input register and an output register inside the ILM. The example contains two timing paths, in-to-f1 and f2-to-out. CCOpt cannot independently adjust the insertion delay of flop f1 and flop f2 because the ILM contents are read-only.

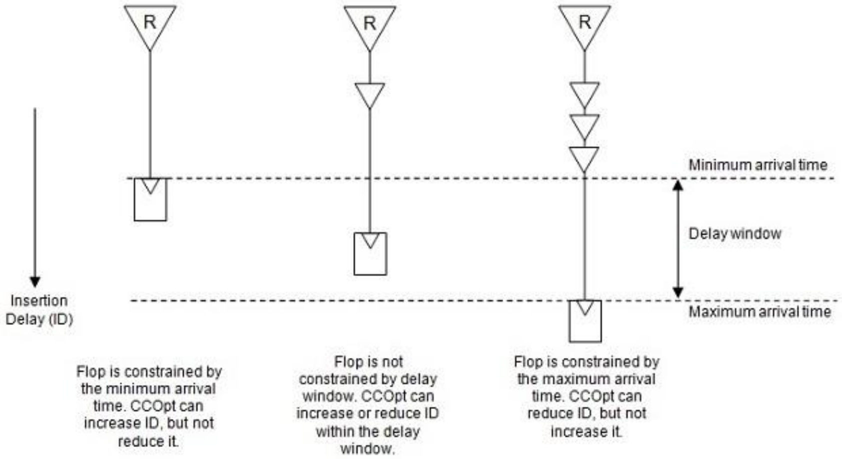
Partition or Macro in a Disjoint Chain



Constraint Windows

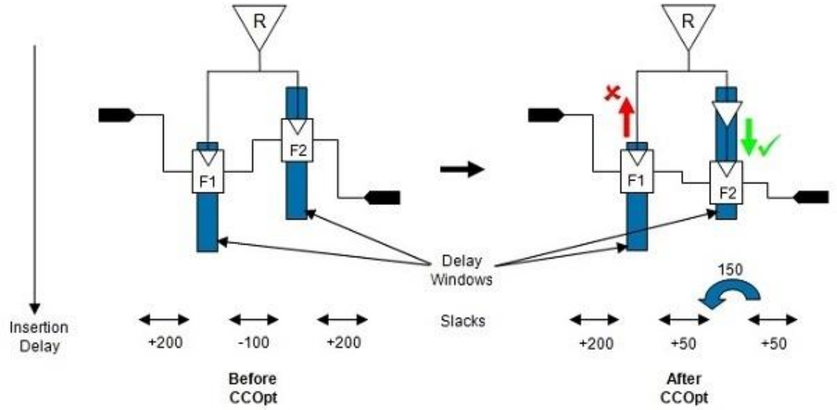
CCOpt determines a delay constraint window for every sink representing the minimum and maximum clock insertion delay (clock arrival time) for the sink. This is illustrated below.

Constraint Windows


R
R
R
Minimum arrival time
Delay window
Maximum arrival time
Insertion Delay (ID)
Flop is constrained by the minimum arrival time. CCOpt can increase ID, but not reduce it.
Flop is not constrained by delay window. CCOpt can increase or reduce ID within the delay window.
Flop is constrained by the maximum arrival time. CCOpt can reduce ID, but not increase it.

When calculating delay constraint windows, CCOpt considers all the constraints applicable to a particular sink including physical constraints, for example the minimum buffering delay from the clustering step, skew group constraints and insertion delay limit. Useful skew can, therefore, only take place if permitted by the delay constraint window. Consider the example below.

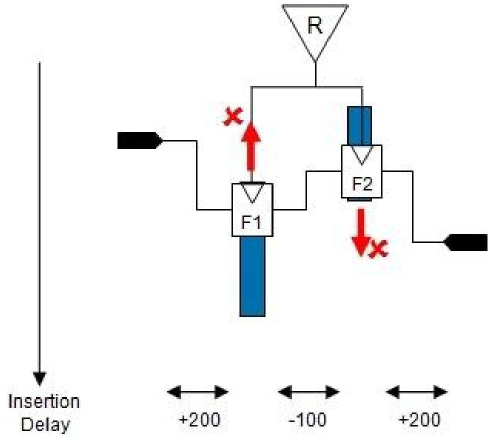
Skew Scheduling within Constraint Windows



In this example, flop F2 needs to be scheduled later than flop F1 to improve the negative slack of -100ps. To achieve this, CCOpt could decrease the insertion delay to flop F1 but F1 is already close to the top of the delay constraint window. Alternatively, increasing the insertion delay to flop F2, which is in the middle of its delay constraint window, permits the movement of 150ps of slack from launch side to the capture side of F2.

However, consider the same situation with different constraints and, therefore, different delay constraint windows. In the example below, CCOpt is unable to fix negative slack because of the delay constraint windows.

Skew Scheduling Restricted by Constraint Windows



CCOpt is unable to reduce the insertion delay to flop F1 because it is already at the top of its delay window. Similarly, CCOpt is unable to increase the insertion delay to flop F2 because it is already at the bottom of its delay window. Therefore, the negative slack between F1 and F2 cannot be fixed by useful skew. It might be possible to optimize the datapath between F1 and F2 further, but note that CCOpt will typically only skew clock sinks when datapath optimization is unable to progress.

Timing Windows

Further window types are the "chosen window" that appears in worst chain reports and "timing windows" that are viewable in the CCOpt Clock Tree Debugger. The chosen window represents an insertion delay range that useful skew scheduling would like a sink to be within, in order to progress timing optimization.

The timing window represents the final window used by the implementation step. Each sink is assigned a timing window such that so long as the sink is within the timing window the sink will not be at risk of degrading the high effort path group(s) WNS and will not adversely impact hold timing. Implementation is able to group sinks together that are physically nearby with overlapping timing windows such that clock tree area and power is reduced by avoiding the need to strictly balance less critical sinks.

For more information on worst chain reporting, both in the log and via the `report_ccopt_worst_chain` command, see the Worst Chain section.

Reporting

There are several commands available for generating reports in CTS. The table below provides a snapshot of the information being sought, the reporting commands for writing out the reports including this information, and the alternative methods that can be used for obtaining the information.

Information	Reporting Command	Alternatives
CTS QoR (clock tree area, cell/wire capacitance, wirelength, max_trans, net segment length)	`report_ccopt_clock_trees`	DAG stats in log file Clock tree debugger (CTD) (`ctd_win`) `get_ccopt_clock_tree_*` commands

CTS QoR Violations (max_trans, capacitance, net segment length violations)	`report_ccopt_clock_trees`	Log file (IMPCCOPT messages) CTD
Balancing insertion delay, skew, skew window occupancy, skew histograms, skew group sizes	`report_ccopt_skew_groups`	Log file CTD `get_ccopt_skew_group_*` commands
Logical Clock Tree Structure	`report_ccopt_clock_tree_structure`	Unit delay mode CTD
Clock Tree Convergence	`report_ccopt_clock_tree_convergence`	CTD
Don't Touch Ports	`report_ccopt_preserved_clock_tree_ports`	Unit delay mode CTD
Library Cell Filtering CTS cell list auto-trimming	`report_ccopt_cell_filtering_reasons`	Log file
Cell Halo Violations	`report_ccopt_cell_halo_violations`	Log file Violation Browser within the GUI
Pin Insertion Delays user and useful skews	`report_ccopt_pin_insertion_delays`	Log file CTD
Worst Chain Setup (optionally hold)	`report_ccopt_worst_chain`	Log file
Clock EM Violations	`verifyACLimit`	None

Skew Groups

The `report_ccopt_skew_groups -file` *`filename`* command creates a report including a summary of all defined skew groups with insertion delay data per delay corner and the maximum and minimum insertion delay paths per skew group and delay corner. The optional `-histograms` parameter can be used to include an insertion delay histogram per delay corner.

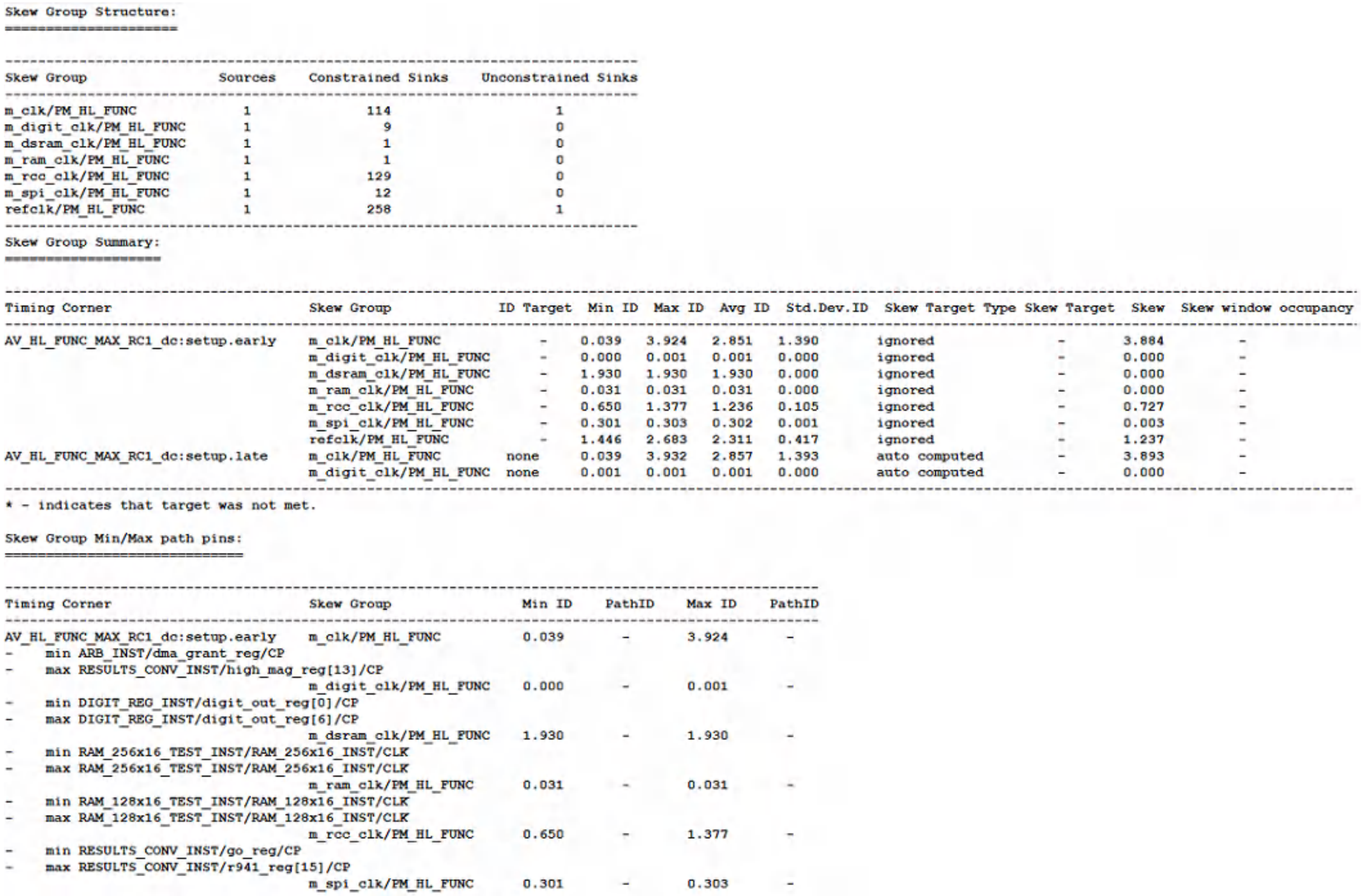
The main sections in the skew group report are:

- Skew group structure summary indicating number of active sinks
- Skew group summary indicating maximum and minimum insertion delay per skew group per late/early conditions of each delay corner
- Table of skew group minimum and maximum insertion delay paths per skew group and delay corner with sink pin names. Each path is given an ID number.
- Detailed path listings, using the same path ID number

An example of the skew group summary is illustrated below.

Skew window occupancy column is defined as the percentage of sinks in the skew group that fall within the user defined skew target. It is computed by finding a min and max latency window in which the largest proportion of sinks fall. The intent of this column is to indicate, when the skew target is not met, some measure of how badly it has not been met by showing the largest proportion of sinks that occupy the desired skew target.

A '*' indicates that a target insertion delay or skew target was not met.



```
Skew Group Structure:
=====================

-----------------------------------------------------------------------------
Skew Group                Sources    Constrained Sinks    Unconstrained Sinks
-----------------------------------------------------------------------------
m_clk/PM_HL_FUNC             1              114                    1
m_digit_clk/PM_HL_FUNC       1                9                    0
m_dsram_clk/PM_HL_FUNC       1                1                    0
m_ram_clk/PM_HL_FUNC         1                1                    0
m_rcc_clk/PM_HL_FUNC         1              129                    0
m_spi_clk/PM_HL_FUNC         1               12                    0
refclk/PM_HL_FUNC            1              258                    1
-----------------------------------------------------------------------------
Skew Group Summary:
===================

---------------------------------------------------------------------------------------------------------------------------------------------------------------------
Timing Corner                        Skew Group             ID Target  Min ID  Max ID  Avg ID  Std.Dev.ID  Skew Target Type Skew Target  Skew  Skew window occupancy
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
AV_HL_FUNC_MAX_RC1_dc:setup.early    m_clk/PM_HL_FUNC             -   0.039   3.924   2.851   1.390       ignored                -      3.884          -
                                     m_digit_clk/PM_HL_FUNC       -   0.000   0.001   0.001   0.000       ignored                -      0.000          -
                                     m_dsram_clk/PM_HL_FUNC       -   1.930   1.930   1.930   0.000       ignored                -      0.000          -
                                     m_ram_clk/PM_HL_FUNC         -   0.031   0.031   0.031   0.000       ignored                -      0.000          -
                                     m_rcc_clk/PM_HL_FUNC         -   0.650   1.377   1.236   0.105       ignored                -      0.727          -
                                     m_spi_clk/PM_HL_FUNC         -   0.301   0.303   0.302   0.001       ignored                -      0.003          -
                                     refclk/PM_HL_FUNC            -   1.446   2.683   2.311   0.417       ignored                -      1.237          -
AV_HL_FUNC_MAX_RC1_dc:setup.late     m_clk/PM_HL_FUNC        none     0.039   3.932   2.857   1.393       auto computed          -      3.893          -
                                     m_digit_clk/PM_HL_FUNC  none     0.001   0.001   0.001   0.000       auto computed          -      0.000          -
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
* - indicates that target was not met.

Skew Group Min/Max path pins:
=============================

---------------------------------------------------------------------------------------------------
Timing Corner                        Skew Group                Min ID    PathID    Max ID    PathID
---------------------------------------------------------------------------------------------------
AV_HL_FUNC_MAX_RC1_dc:setup.early    m_clk/PM_HL_FUNC          0.039        -      3.924        -
-    min ARB_INST/dma_grant_reg/CP
-    max RESULTS_CONV_INST/high_mag_reg[13]/CP
                                     m_digit_clk/PM_HL_FUNC    0.000        -      0.001        -
-    min DIGIT_REG_INST/digit_out_reg[0]/CP
-    max DIGIT_REG_INST/digit_out_reg[6]/CP
                                     m_dsram_clk/PM_HL_FUNC    1.930        -      1.930        -
-    min RAM_256x16_TEST_INST/RAM_256x16_INST/CLK
-    max RAM_256x16_TEST_INST/RAM_256x16_INST/CLK
                                     m_ram_clk/PM_HL_FUNC      0.031        -      0.031        -
-    min RAM_128x16_TEST_INST/RAM_128x16_INST/CLK
-    max RAM_128x16_TEST_INST/RAM_128x16_INST/CLK
                                     m_rcc_clk/PM_HL_FUNC      0.650        -      1.377        -
-    min RESULTS_CONV_INST/go_reg/CP
-    max RESULTS_CONV_INST/r941_reg[15]/CP
                                     m_spi_clk/PM_HL_FUNC      0.301        -      0.303        -
```

The `report_ccopt_skew_groups` command accepts various parameters to restrict the reporting to specified skew groups or delay corners, or to restrict to particular paths using `-through` and `-to` in a similar manner to the `report_timing` command. The `-summary` parameter can be used just to report the summary. For more details, see the *Innovus Text Command Reference*.

Note that the skew group report uses the same timing model as the CCOpt-CTS engine. To report on timing clocks, use the `report_clock_timing` command.

Including Non-Reporting Skews in Reports

The `create_ccopt_clock_tree_spec` command, by default, creates reporting-only skew groups for generated clocks. A reporting-only skew group is a skew group whose `constrains` property is set to `none`. Such a skew group imposes no clock balancing constraint and will not be considered by CTS. You can choose to not include the reporting-only skew groups in the reports and the log file by default.

The parameter, `-include_reporting_only_skew_groups` provided in below commands is used to specify that the report should include reporting-only skews.

- `report_ccopt_clock_tree_structure`
- `report_ccopt_pin_insertion_delays`
- `report_ccopt_skew_groups`

By default, the reporting-only skew groups are not included in the report. If a reporting-only skew group is explicitly specified using the `-skew_groups` parameter, then that skew group is always included in the report output. However, if both `-include_reporting_only_skew_groups` and `-skew_groups` parameters are specified together, the software errors out.

The corresponding property, spec_config_create_reporting_only_skew_groups can also be used to control whether spec creation will synthesize reporting-only skew groups. When this property is set to `false`, reporting-only skew groups are completely omitted from the generated spec. When set to `true`, reporting-only skew groups are included in the generated spec but are marked with the `constrains` property set to `none`. Either way, they impose no balancing constraint.

For details about the property, see CCOpt Properties.

In the GUI also, you can control whether or not the reporting-only skew groups are displayed in the CTD. Use the `-include_reporting_only_skew_groups` parameter of the `ctd_win` command. When this parameter is specified, the GUI displays the reporting-only skew groups. By default, this setting if off, which means the reporting-only skew groups are omitted from the CTD display.

Clock Trees

The `report_ccopt_clock_trees -file` *`file`* command creates a report including statistics per clock tree and statistics over all clock trees, including transition violations. The `-histograms` parameter can be used to enable histograms of various data and the `-list_special_pins` parameter will add a detailed listing of clock tree stop and ignore pins. The clock tree report also shows more details of special pin types. Parts of a sample report are shown below.

Example1: **Clock Timing Summary table**

```
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 Trunk Slew Trunk Slew
                                            Leaf Slew     Leaf Slew    Trunk Slew    Trunk Slew   TargetType Target
  TargetType   Target
---------------------------------------------------------------------------------------------------------------------
------------------------
m_clk    AV_HL_FUNC_MAX_RC1_dc:setup.early  9.692        11.003       9.687        10.997      ignored        -
 ignored         -
m_clk    AV_HL_FUNC_MAX_RC1_dc:setup.late   9.725        11.014       9.720        11.008      auto computed -
 auto computed   -
m_clk    AV_HL_FUNC_MIN_RC1_dc:hold.early   4.365        4.330         4.353        4.314       ignored        -
 ignored         -
m_clk    AV_HL_FUNC_MIN_RC1_dc:hold.late    4.384        4.333         4.372        4.317       ignored        -
 ignored         -

---------------------------------------------------------------------------------------------------------------------
------------------------

* - indicates that target was not met.

auto extracted - target was extracted from SDC.
auto computed - target was computed when balancing trees.
```

Example2: Fanout Histogram for all Trees

```
Fanout histogram across all clock trees:

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

```
Fanout Non-leaf  Leaf
       nets      nets
----------------------
1     18        3
2     0         1
5     0         1
6     0         2
8     0         7
9     0         4
11    1         0
12    0         1
16    0         14
20    0         1
24    0         1
32    0         3
58    0         1
----------------------
```

Example3: Clock Tree Special Pins

```
Clock tree special pins:

----------------------------------------
Pin                        Type
----------------------------------------
TDSP_DS_CS_INST/g395/A1  explicit ignore
----------------------------------------
```

Clock Tree Network Structure

The `report_ccopt_clock_tree_structure` command reports the structure of the clock network as a text report. The syntax of the command is as follows:

```
[-help]
[-check_type {setup | hold}]
[-clock_trees {string1 string2 ...}]
[-delay_corner corner_name]
[-delay_type {early | late}]
[-expand_below_logic]
[-expand_generated_clock_trees {independently | inline | independently_and_inline}]
[-file file_name]
[-include_reporting_only_skew_groups]
[-show_sinks]
[-update_timing]
```

The parameters, `-expand_below_logic` and `-expand_generated_clock_trees` control how the report addresses clock convergence/reconvergence at clock logic cells and at clock generator paths, which are instances with more than one clock input. Such a multi-input instance will appear multiple times in the report. By default, the command aims for brevity, and will therefore emit the subtree below the multi-input instance once, at the first occurrence; and at subsequent occurrences will simply indicate that the fanout has been omitted. However, this behavior can be modified by specifying the `-expand_below_logic` and `-expand_generated_clock_trees` parameters.

When the `-expand_below_logic` parameter is specified, the subtree below each multi-input logic will be printed.

when the `-expand_generated_clock_trees` parameter is specified, you can choose how generated clock trees should be displayed. The following options are available:

- `independently`: display generated clock trees at the top level. This is the default display.
- `inline` : display generated clock trees as fanout of their generator inputs
- `independently_and_inline`: display generated clock trees both at the top level and as fanout of their generator inputs

The report can be customized to include or exclude the skew-group latencies at each sink. for this, you can use the `-show_sinks` parameter. By default, the clock sink networks, which are the non-generator flops/latches are not included in the report. Instead, a count of the number of sinks is displayed,for example. `"... 209 sinks omitted"`. However, when this parameter is specified, per-skew group latencies for each sink are included.

The report identifies macros in the clock tree. This means that if your clock sink is a macro, the report will show “macro sink” instead of just “sink”.

When the `-update_timing` parameter is specified, the report includes detailed timing information including pin slew, capacitance, and location along with the clock tree structure. When the command is run without specfying the `-update_timing` parameter, the report format is as follows:

- The summary information is added below the name of the clock tree
- The slew, capacitance, and co-ordinate information is not included
- The input and output pin information is merged in a single line

When the `-include_reporting_only_skew_groups` parameter is specified, the report additionally includes the reporting-only skew groups. By default, the reporting-only skew groups are not included in the report.

Sample reports are shown below.

Sample 1: Clock Tree Structure report for clock trees, `m_clk` and `m_digit_clk: .`

```
report_ccopt_clock_tree_structure -clock_trees {m_clk m_digit_clk}
```

```
Clock tree m_clk:
Total FF: 115
Max Level: 5
  (L1) TEST_CONTROL_INST/g137/ZN (ND2D1BWP)
  \_  (L1) port TEST_CONTROL_INST/m_clk
       \_ ... (21 sinks omitted)
       \_  (L2) DMA_INST/CPF_LS_158_m_clk/I -> Z (LVLHLD2BWP)
        |   \_ ... (8 sinks omitted)
        |   \_  (L3) DMA_INST/RC_CG_HIER_INST0/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
        |       \_ ... (8 sinks omitted)
        \_  (L2) RESULTS_CONV_INST/CPF_LS_159_m_clk/I -> Z (LVLHLD2BWP)
        |   \_ ... (15 sinks omitted)
        |   \_  (L3) RESULTS_CONV_INST/RC_CG_HIER_INST1/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
        |   |   \_ ... (9 sinks omitted)
        |   \_  (L3) RESULTS_CONV_INST/RC_CG_HIER_INST2/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
        |   |   \_ ... (16 sinks omitted)
```

....

```
Clock tree m_digit_clk:
 Total FF: 9
 Max Level: 3
   (L1) TEST_CONTROL_INST/g141/ZN (IOA21D1BWP)
   \_  (L1) port TEST_CONTROL_INST/m_digit_clk
        \_ ... (9 sinks omitted)
```

Sample 2: Clock Tree Structure report for clock tree, `m_clk` when the `-update_timing` parameter is specified.

```
report_ccopt_clock_tree_structure -clock_trees {m_clk} -update_timing
```

```
Clock tree m_clk:
TEST_CONTROL_INST/g137/ZN root output at (273.210,301.000), lib_cell ND2D1BWP, level 1, slew 9.724ns, wire_cap
0.096pF, load_cap 0.452pF
\_ ... (21 sinks omitted)
\_ DMA_INST/CPF_LS_158_m_clk/I logic input at (169.210,300.720), lib_cell LVLHLD2BWP, level 2, slew 9.725ns
| DMA_INST/CPF_LS_158_m_clk/Z logic output at (169.770,300.860), lib_cell LVLHLD2BWP, level 2, slew 0.449ns,
wire_cap 0.010pF, load_cap 0.006pF
| \_ ... (8 sinks omitted)
| \_ DMA_INST/RC_CG_HIER_INST0/RC_CGIC_INST/CP cgate input at (115.450,305.760), lib_cell CKLNQD1BWP, level 3, slew
0.449ns
| DMA_INST/RC_CG_HIER_INST0/RC_CGIC_INST/Q cgate output at (116.290,305.900), lib_cell CKLNQD1BWP, level 3, slew
0.320ns, wire_cap 0.004pF, load_cap 0.005pF
| \_ ... (8 sinks omitted)
\_CONV_INST/CPF_LS_159_m_clk/I logic input at (199.170,320.880), lib_cell LVLHLD2BWP, level 2, slew 9.724ns
| CONV_INST/CPF_LS_159_m_clk/Z logic output at (199.730,321.020), lib_cell LVLHLD2BWP, level 2, slew 0.864ns,
wire_cap 0.030pF, load_cap 0.015pF
| \_ ... (15 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST1/RC_CGIC_INST/CP cgate input at (110.130,313.320), lib_cell CKLNQD1BWP, level 3, slew
0.865ns
| |  CONV_INST/RC_CG_HIER_INST1/RC_CGIC_INST/Q cgate output at (109.290,313.460), lib_cell CKLNQD1BWP, level 3, slew
0.338ns, wire_cap 0.004pF, load_cap 0.006pF
| | \_ ... (9 sinks omitted)
| \_ RESULTS_CONV_INST/RC_CG_HIER_INST2/RC_CGIC_INST/CP cgate input at (155.490,341.040), lib_cell CKLNQD1BWP, level
3, slew 0.865ns
| |  RESULTS_CONV_INST/RC_CG_HIER_INST2/RC_CGIC_INST/Q cgate output at (156.330,341.180), lib_cell CKLNQD1BWP, level
3, slew 0.668ns, wire_cap 0.011pF, load_cap 0.010pF
| | \_ ... (16 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST3/RC_CGIC_INST/CP cgate input at (154.650,354.760), lib_cell CKLNQD1BWP, level 3, slew
0.865ns
| | CONV_INST/RC_CG_HIER_INST3/RC_CGIC_INST/Q cgate output at (155.490,354.620), lib_cell CKLNQD1BWP, level 3, slew
0.636ns, wire_cap 0.010pF, load_cap 0.010pF
| | \_ ... (16 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST6/RC_CGIC_INST/CP cgate input at (111.110,338.520), lib_cell CKLNQD1BWP, level 3, slew
0.865ns
| | CONV_INST/RC_CG_HIER_INST6/RC_CGIC_INST/Q cgate output at (110.270,338.660), lib_cell CKLNQD1BWP, level 3, slew
0.246ns, wire_cap 0.003pF, load_cap 0.004pF
| | \_ ... (6 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST7/RC_CGIC_INST/CP cgate input at (112.790,324.520), lib_cell CKLNQD1BWP, level 3, slew
0.865ns
|    CONV_INST/RC_CG_HIER_INST7/RC_CGIC_INST/Q cgate output at (111.950,324.380), lib_cell CKLNQD1BWP, level 3, slew
0.349ns, wire_cap 0.005pF, load_cap 0.005pF
|    \_ ... (8 sinks omitted)
\_ SPI_INST/RC_CG_HIER_INST17/RC_CGIC_INST/CP cgate input at (133.630,125.440), lib_cell CKLNQD1BWP, level 2, slew
9.725ns
   SPI_INST/RC_CG_HIER_INST17/RC_CGIC_INST/Q cgate output at (134.470,125.300), lib_cell CKLNQD1BWP, level 2, slew
0.335ns, wire_cap 0.010pF, load_cap 0.005pF
   \_ ... (8 sinks omitted)
```

Sample 3: Clock Tree Structure for clock tree, `m_clk` to include skew-group latencies.

```
report_ccopt_clock_tree_structure -clock_trees {m_clk m_digit_clk} -expand_generated_clock_trees inline

Clock tree m_clk:
```

```
Total FF: 115
Max Level: 5
(L1) TEST_CONTROL_INST/g137/ZN (ND2D1BWP)
\_  (L1) port TEST_CONTROL_INST/m_clk
   \_ ... (21 sinks omitted)
   \_   (L2) DMA_INST/CPF_LS_158_m_clk/I -> Z (LVLHLD2BWP)
   |    \_ ... (8 sinks omitted)
   |    \_ (L3) DMA_INST/RC_CG_HIER_INST0/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
   |        \_ ... (8 sinks omitted)
   \_   (L2) RESULTS_CONV_INST/CPF_LS_159_m_clk/I -> Z (LVLHLD2BWP)
   |    \_ ... (15 sinks omitted)
   |    \_ (L3) RESULTS_CONV_INST/RC_CG_HIER_INST1/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
   |   |    \_ ... (9 sinks omitted)
   |    \_ (L3) RESULTS_CONV_INST/RC_CG_HIER_INST2/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
   |   | \_ ... (16 sinks omitted)
   |    \_ (L3) RESULTS_CONV_INST/RC_CG_HIER_INST3/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
   |   |    \_ ... (16 sinks omitted)
   |    \_ (L3) RESULTS_CONV_INST/RC_CG_HIER_INST6/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
   |   |    \_ ... (6 sinks omitted)
   |    \_ (L3) RESULTS_CONV_INST/RC_CG_HIER_INST7/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
   |        \_ ... (8 sinks omitted)
   \_   (L2) SPI_INST/RC_CG_HIER_INST17/RC_CGIC_INST/CP -> Q (CKLNQD1BWP)
         \_ ... (8 sinks omitted)

Clock tree m_digit_clk:
Total FF: 9
Max Level: 3
(L1) TEST_CONTROL_INST/g141/ZN (IOA21D1BWP)
\_   (L1) port TEST_CONTROL_INST/m_digit_clk
     \_ ... (9 sinks omitted)
```

Pin Insertion Delays

The `report_ccopt_pin_insertion_delays` command reports pin insertion delays at clock tree sinks. The report provides a quick overview of the distribution and nature of the pin insertion delays found, which can be used to identify the problem areas such as excessively large skews, surprisingly numerous skews, or large numbers of small skews, and so on. The syntax of the command is as follows:

```
[-help]
[-bin_size string]
[-check_type {setup | hold}]
[-clock_trees {string1 string2 ...}]
[-delay_corner delay_corner_name]
[-delay_type {early | late}]
[-file file_name]
[-skew_groups {string1 string2 ...} | -include_reporting_only_skew_groups]
[-sources {string1 string2 ...}]
```

The `-clock_trees` and `-skew_groups` parameters are used to restrict the scope of the report as a subset of the clock tree sinks, which means that when the `-clock_trees` parameter is specified, only those sinks that are under the listed clock trees are selected. The `-include_reporting_only_skew_groups` parameter is used to include the reporting-only skew groups in the report. By default, the reporting-only skew groups are not included in the report. The `-skew_groups` parameter restricts the report to the specified skew groups. This means that only those sinks that are under the specified skew groups are selected for reporting. The `-include_reporting_only_skew_groups` and `-skew_groups` parameters are mutually exclusive. If both these parameters are specified together, the software errors out. By default, the report analyzes the pin insertion delays that are found under the CCOpt primary half corner. The `-delay_corner`, `-check_type`, and `-delay_type` parameters permit the selection of another half corner to report against.

The `-bin_size` parameter specifies a bin (bucket) size for the report histograms. By default the software uses the histogram bin size specified using the CCOpt property, pin_insertion_delay_histogram_bin_size.

Example

The following command reports the pin insertion delays at all clock tree sinks.

```
report_ccopt_pin_insertion_delays
```

Note: The CCOpt pin insertion delays are +ve valued if they produce an advance, and -ve valued if they produce a delay. This can be viewed in the report output shown below.

```
Positive (advancing) pin insertion delays
================================================

Found 1 advances (2.041% of 49 clock tree sinks)

------------------------------------------------
From (ns)    To (ns)    Count
------------------------------------------------
0.000         0.010     1
------------------------------------------------

Mean              : 0.006ns
Std.Dev           : 0.000ns

Smallest advance  : 0.006ns at d2211a/CK
Largest advance   : 0.006ns at d2211a/CK

Negative (delaying) pin insertion delays
================================================

Found 23 delays (46.939% of 49 clock tree sinks)

------------------------------------------------
From (ns)    To (ns)   Count
------------------------------------------------
-0.120      -0.110     1
-0.110      -0.100     0
-0.100      -0.090     1
-0.090      -0.080     2
-0.080      -0.070     3
-0.070      -0.060     2
-0.060      -0.050     4
-0.050      -0.040     4
-0.040      -0.030     3
-0.030      -0.020     1
-0.020      -0.010     2
-----------------------------------------------

Mean             : -0.056ns
Std.Dev          : 0.025ns

Smallest delay   : -0.014ns at d3211a/CK
Largest delay    : -0.110ns at d1111d/CK
```

Timing Data for CTS-Specific Reports

For CTS-specific reports, the clock tree timing engine (CTS timing engine) is used to calculate the timing data that is displayed in the clock tree and skew group reports. By default, the clock tree timing engine is invalidated when you generate the reports, so all the timing data is recalculated. This increases the time taken to generate the reports. Use the `-no_invalidate` parameter of the reporting commands to specify that the clock tree timing engine should not be invalidated and that the existing timing data, if any, should be used in the reports.

As with CTS itself, these reports place priority on the late half of the CTS primary delay corner. For example, when timing data is included in the `report_ccopt_clock_tree_structure` command, it includes timing data only from the CTS primary half corner.

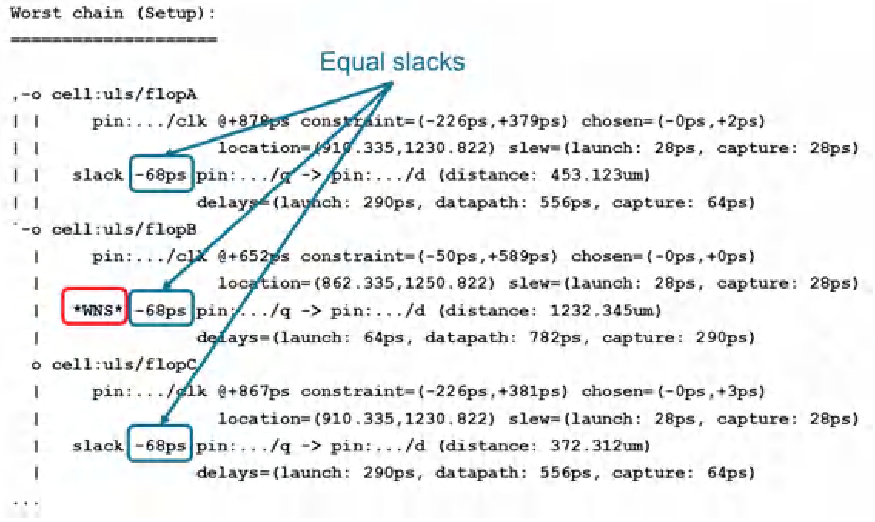
Worst Chain

For an explanation about the concept of chains, see the Chains section.

The worst chain is reported from time to time in the log during CCOpt and an examination of the log may help identify reasons limiting timing optimization. In addition, the `report_ccopt_worst_chain` command can be used to report the worst timing chain after `ccopt_design` has completed, but note that this will reflect the current worst chain, not the worst chain during optimization.

The illustration below shows a perfectly balanced worst chain. Each sequential element in the chain is identified by a “cell:name” line with ASCII art on the left representing the chain connectivity. In this example, there is a loop between flops A and B. The data between each sequential element summarizes the combinational path between adjacent sequential elements. For example, the timing slack is identified, and the WNS marked with “*WNS*”. In this example, the slack between each stage is identical suggesting that it is not possible to further move slack between stages. Such a chain is balanced.

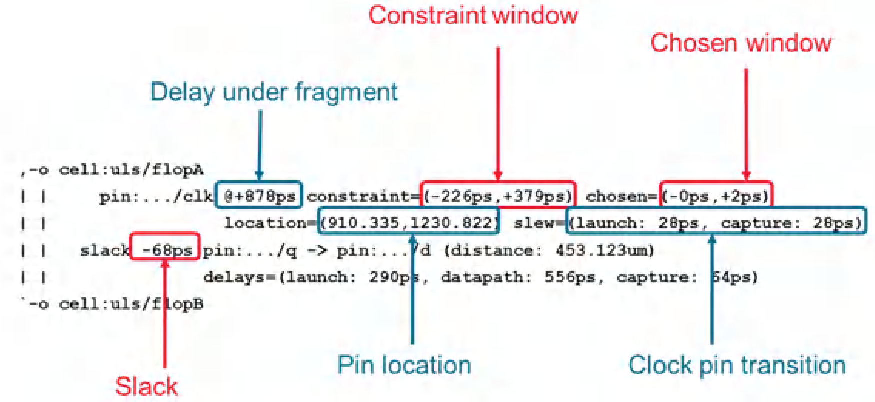
Example of Worst Chain



```
Worst chain (Setup):
====================

,-o cell:uls/flopA
| |      pin:.../clk @+878ps constraint=(-226ps,+379ps) chosen=(-0ps,+2ps)
| |                  location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| |   slack -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| |             delays=(launch: 290ps, datapath: 556ps, capture: 64ps)
`-o cell:uls/flopB
  |      pin:.../clk @+652ps constraint=(-50ps,+589ps) chosen=(-0ps,+0ps)
  |                  location=(862.335,1250.822) slew=(launch: 28ps, capture: 28ps)
  |   *WNS* -68ps pin:.../q -> pin:.../d (distance: 1232.345um)
  |             delays=(launch: 64ps, datapath: 782ps, capture: 290ps)
  o cell:uls/flopC
  |      pin:.../clk @+867ps constraint=(-226ps,+381ps) chosen=(-0ps,+3ps)
  |                  location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
  |   slack -68ps pin:.../q -> pin:.../d (distance: 372.312um)
  |             delays=(launch: 290ps, datapath: 556ps, capture: 64ps)
...
```

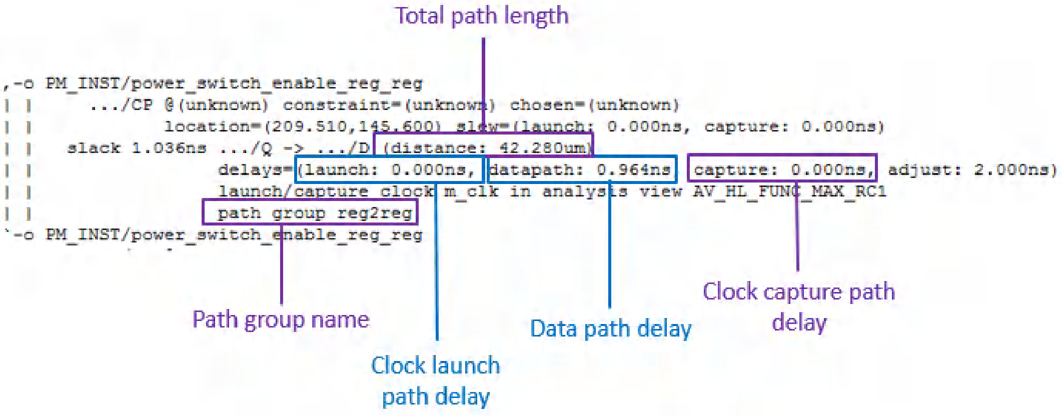
The two diagrams below label the various fields in the worst chain report.

Worst Chain Data - Diagram 1



```
,-o cell:uls/flopA
| |     pin:.../clk @+878ps constraint=(-226ps,+379ps) chosen=(-0ps,+2ps)
| |                 location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| |   slack -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| |               delays=(launch: 290ps, datapath: 556ps, capture: 54ps)
`-o cell:uls/flopB
```

Worst Chain Data - Diagram 2



```
,-o PM_INST/power_switch_enable_reg_reg
| |      .../CP @(unknown) constraint=(unknown) chosen=(unknown)
| |               location=(209.510,145.600) slew=(launch: 0.000ns, capture: 0.000ns)
| |   slack 1.036ns .../Q -> .../D (distance: 42.280um)
| |                   delays=(launch: 0.000ns, datapath: 0.964ns, capture: 0.000ns, adjust: 2.000ns)
| |                   launch/capture clock m_clk in analysis view AV_HL_FUNC_MAX_RC1
| |                   path group reg2reg
`-o PM_INST/power_switch_enable_reg_reg
```

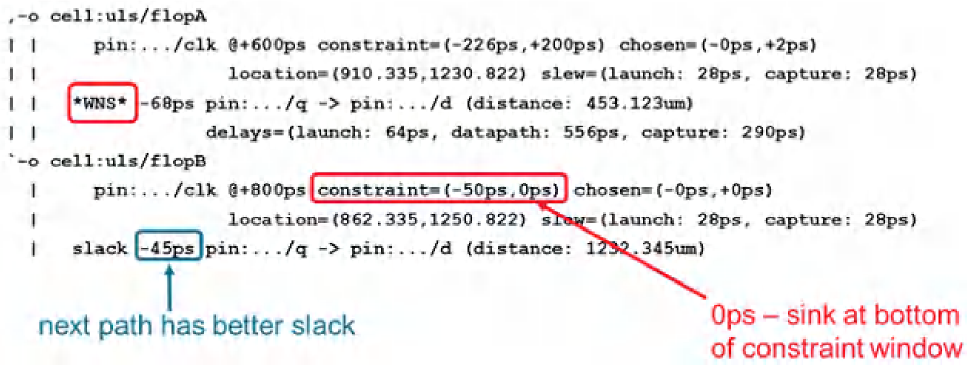
The above labels are described in detail in the table below.

Filed Name	Description
Constraint window	The constraint window is the range of permissible insertion delay modification subject to the minimum buffering delay from the clustering step, the automatic insertion delay limit and optional user skew constraints. For more information, see the Constraint Windows section. In this example, the `-226ps` indicates that the sink insertion delay can be scheduled up to `226ps` earlier and the `+379ps` indicates that the sink can be scheduled up to `379ps` later.
Chosen window	The chosen window represents the insertion delay range, relative to the current insertion delay, within which useful skew scheduling desires to place the sink.
Slack	The datapath slack between two sequential elements in the chain.
Delay under fragment	This is an internal number representing the delay between the non-buffer parent of a sink and the sink.
Pin location	The placement coordinate of the sink pin.
Clock Pin transition	The transition time at the sink pin, both for launch and for capture.

Clock launch path delay	The launch clock arrival time.
Data path delay	The datapath delay.
Total path length	The physical length of the path obtained by summing the distance between each pin pair along the path.
Clock capture path delay	The capture clock arrival time.
Path group name	The name of the path group for each path.

The example below illustrates a worst chain report where the slacks on either side of flopB are unequal. To further improve the WNS, it is desirable to move slack from the launch side of flopB to the capture side of flopB. To do this would require the insertion delay of flopB to be increased but this is not possible because flopB is at the bottom of the constraint window. The source of such a limit is the automatic insertion delay limit as discussed in the Restricting CCOpt Skew Scheduling section.

Example of Worst Chain – sink at bottom of constraint window



```
,-o cell:uls/flopA
| |     pin:.../clk @+600ps constraint=(-226ps,+200ps) chosen=(-0ps,+2ps)
| |                 location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| |   *WNS* -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| |               delays=(launch: 64ps, datapath: 556ps, capture: 290ps)
`-o cell:uls/flopB
  |     pin:.../clk @+800ps constraint=(-50ps,0ps) chosen=(-0ps,+0ps)
  |                 location=(862.335,1250.822) slew=(launch: 28ps, capture: 28ps)
  |   slack -45ps pin:.../q -> pin:.../d (distance: 1232.345um)
```

next path has better slack

0ps – sink at bottom of constraint window

Halo Violations

Clock cell halo violations can be reported with the `report_ccopt_cell_halo_violations` command. The report can be viewed as a text file and in the Violation Browser within the GUI.

Use the following set of commands to view the clock halo violations in the Violation Browser.

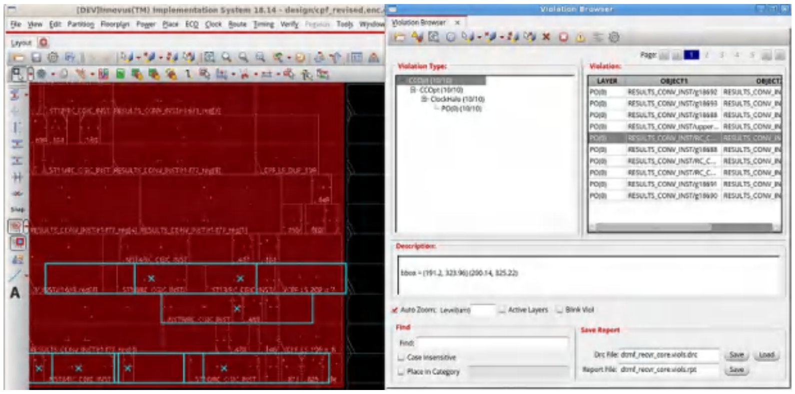
- Create cell halo violations using the following property:

  ```
  set_ccopt_property cell_halo_x 3
  ```

- Use the following command to add the detected clock halo violations to the Violation Browser and display them in the GUI:

  ```
  report_ccopt_cell_halo_violations -add_markers
  ```

  You will see the following:



[DEV]Innovus(TM) Implementation System 18.14 - design/cpf_revised.enc
Violation Browser
Page: 1 2 3 4 5

Violation Type:

CCOpt (10/10)
CCOpt (10/10)
ClockHalo (10/10)
PO(0) (10/10)

Violation:

LAYER	OBJECT1	OBJECT2
PO(0)	RESULTS_CONV_INST/g18692	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/g18693	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/g18688	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/upper...	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/RC_C...	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/g18688	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/RC_C...	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/RC_C...	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/g18691	RESULTS_CONV_IN
PO(0)	RESULTS_CONV_INST/g18690	RESULTS_CONV_IN

Description:

bbox = (191.2, 323.96) (200.14, 325.22)

☑ Auto Zoom: Level(um) □ Active Layers □ Blink Viol

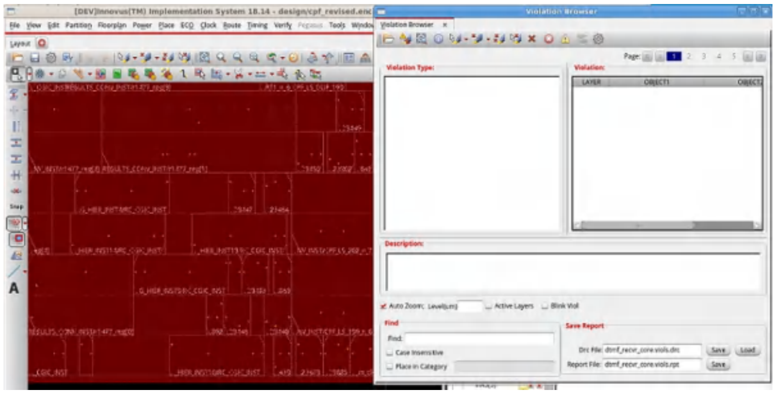
Find
Find:
□ Case Insensitive
□ Place in Category

Save Report
Drc File: dcmf_recvr_core.viols.drc
Report File: dcmf_recvr_core.viols.rpt

- Use the following command to clear the clock halo violations in the Violation Browser and delete markers in the GUI:

  ```
  report_ccopt_cell_halo_violations -clear_markers
  ```

  You will see the following:



[DEV]Innovus(TM) Implementation System 18.14 - design/cpf_revised.enc
Violation Browser
Page: 1 2 3 4 5

Violation Type:

Violation:

LAYER	OBJECT1	OBJECT2

Description:

☑ Auto Zoom: Level(um) □ Active Layers □ Blink Viol

Find
Find:
□ Case Insensitive
□ Place in Category

Save Report
Drc File: dcmf_recvr_core.viols.drc
Report File: dcmf_recvr_core.viols.rpt

For more information about cell halos, see the Cell Halos section.

Cell Name Information

All new cell instances created by CCOpt have an identifying code in their name to let you determine why that cell was created. For example, some common instance name prefixes are:

```
ccl       Cts: Created by the clustering process to meet the slew target
ccl_a     Cts: Created during clustering by the agglom clustering algorithm.
cdb       Cts: Created by CTS to balance the delays in the clock tree.
cdc       Cts: A clock driver created by adding driver cell process for property add_driver_cell.
cpc_drv   Cts: A clock driver created by post conditioning.
cpc_sk    Cts: A clock driver created by post conditioning....
```

To view all the available instance name prefixes and their descriptions, run the `show_ccopt_cell_name_info` command.

Clock Tree Convergence

The `report_ccopt_clock_tree_convergence` command reports statistics on the number of paths leading to clock tree sinks, and a list of the top 10 sinks with the most paths. In the example report below, 5 design IOs (primary input/outputs) have 1400 clock paths leading to them. These sinks are likely to be problematic and need further investigation. For details, see the Clock Tree Convergence section.

```
Convergence above clock sinks

=============================

----------------------------------------------------------------

Number of paths to sink PIOs DFFs  Other sinks Total

----------------------------------------------------------------

1                       1    10118 0           10119

8                       54   2134  183         2371

16                      13   311   18          342

32                      2    0     0           2

700                     3    164   5           172

1400                    5    0     0           5

----------------------------------------------------------------

Sinks with most paths leading to them
=======================================

---------------------------------

Sink             Number of paths

---------------------------------

Owest/nout[0]    1400

Owest/nout[1]    1400

Owest/nout[2]    1400

Owest/nout[3]    1400

Owest/nout[4]    1400

--------------------------------

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

Cell Filtering Reasons

The `report_ccopt_cell_filtering_reasons` command reports the reasons for filtering the specified cell types. The syntax of the command is as follows:

```
[-help]
[-cell_type buffer | inverter | logic | delay | clock_gate]
[-clock_trees {string1 string2 ...}]
[-file  file_name ]
[-power_domain string]
[-tcl_list]
```

The report can be saved as a file by specifying the `-file` parameter of the command or it can be viewed as a Tcl list by specifying the `-tcl_list` parameter. The output of the command is also included in the log.

In the report, "all" is used for the Clock tree names where the reason is common to all clock_trees. Some of the reasons for filtering that are reported and their brief descriptions are provided below:

- `Unbalanced rise/fall delays` : The difference between rise and fall delays for the cell is bigger than 20%; or any of rise or fall timing arcs is missing in the library.
- `Too tall`: The cell height is bigger than the maximum cell height specified using the `max_cell_height` property.
- `Wrong power context` : The cell does not match the power context requirements. For example, do not have same power context on all pins.
- `Not available in all views` : The cell is not available in all analysis views because the library does not contain all the specified views (corner cases) in the design for that specific cell. To avoid this filtering, check the libraries and the MMMC configuration to see why the cell is not available in some views.
- `Always on but not in cpf`: The cell is marked as always on but does not match the CPF description. You can avoid this filtering by checking/updating the CPF.
- `Weak max cap cell`: The cell is unable to drive enough capacitance at certain clock frequencies. This check is controlled by the CCOpt property, `frequency_dependent_max_cap_usability_check_max_cap_fanout_factor`. The default value of this property is 4. The cells that cannot drive four instances of that cell are filtered. You can reduce this value to avoid the filtering, for example, by relaxing the requirement so that the cell only has to be able to drive one or two instances. Alternatively, since this is controlled by the period on the clock_tree, you can check the `effective_clock_period` property to see if those are correct.
- `Library trimming`: The cell was removed automatically because it is an inefficient cell. The cell was removed as an inefficient cell to reduce the library size and save runtime. To avoid removing such cells, disable library trimming.
- `Can't use due to library mismatch`: Check the library data for the cell - both timing and physical data for the cell are needed. There is a mismatch between LEF and liberty libraries because the cell is missing in one of them.
- `Invalid for balancing`: The cell is filtered out because it does not match several predefined properties for balancing, such as: 1) cell delay is too small; 2)cell has an invalid (rise/fall) timing arc.
- `No test enable signal`: The clock gate cell is filtered out because it does not have a test enable input.
- `Has test enable signal`: The clock gate cell is filtered out because it has a test enable input.
  Note: The above two cell filtering reasons are context dependent. The software filters all clock gates, reporting the clock gates with test enables as one set and the clock gates without test enables as another set. A clock gate either has a test enable or it does not have one, so all clock gates are listed in one of the two sets. However, if you need a clock gate with no test enable signal, but there are no such clock gates, you can use a clock gate that has a test enable signal. But the other way around is not possible, which means if you need a clock gate with test enable signal but there is no such clock gate, you cannot use one that does not have a test enable signal.

- `Has observable output`: Clock gates without observability outputs are preferred, if they are available. Normally any cells with observability outputs will be filtered out. But if all the available cells have observability outputs they will not be filtered.
- `Edge sensitive`: The clock gate cell is filtered out because it is not positive edge sensitive.
- `Cannot be legalized`: The cell is filtered because it cannot physically be legalized in the power domain.

Sample Report

```
report_ccopt_cell_filtering_reasons -cell_type buffer


Filtering reasons for cell type: buffer
=======================================

-------------------------------------------------------------------------------
Clock trees  Power domain  Reason             Library cells
-------------------------------------------------------------------------------
all          AO           Library trimming  { BUFFD12BWP BUFFD2BWP BUFFD3BWP
                                                 BUFFD4BWP BUFFD6BWP BUFFD8BWP
                                                 PTBUFFD2BWP  }
all         PLL           Unbalanced        { BUFFD12BWP BUFFD2BWP BUFFD3BWP
                          rise/fall delays   BUFFD4BWP BUFFD6BWP BUFFD8BWP
                                                PTBUFFD2BWP }

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

Retrieving Information using Get Commands

In addition to the reporting commands described above, there are several `get_ccopt_*` commands that aid TCL scripting and combined with `get_ccopt_property` and other Innovus commands, for example `dbGet`, aid the generation of custom checks and reports. The most commonly used `get_ccopt_*` commands are listed below. For each command, additional parameters may be necessary. For details of these commands, refer to the *Innovus Text Command Reference*.

Command Name	Usage
`get_ccopt_clock_tree_cells`	Returns all cells which are part of the clock tree graph. Cells that are leaf sinks are excluded.
`get_ccopt_clock_spines`	Returns a list of clock spines whose names match the specified pattern. ⚠ This command is part of CCOpt spine functionality, which 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.
`get_ccopt_clock_tree_capacitance`	Retrieves the capacitance on the specified pin.
`get_ccopt_clock_tree_nets`	Returns all nets that are part of the clock tree graph.
`get_ccopt_clock_tree_sinks`	Returns all clock tree sinks, that is automatically determined sinks, stop pins, ignore pins which are part of the clock tree graph.
`get_ccopt_clock_trees`	Returns all clock trees.

get_ccopt_dag_traversal	Traverses the clock tree network DAG, starting at the specified pins and traversing to their fanin or fanout, returning a list of the resulting pins.
get_ccopt_delay_corner	Returns the name of the CCOpt primary-half delay corner, which is the main corner that CTS uses for balancing clock trees.
get_ccopt_flexible_htrees	Returns the names of the flexible H-trees whose names match the specified pattern.
get_ccopt_effective_max_capacitance	Returns the value of the frequency-dependent effective maximum capacitance constraint that the software will apply at a given pin in the clock tree.
get_ccopt_clock_tree_slew	Returns the slew information for the user-specified pin.
get_ccopt_skew_group_delay	Returns the insertion delay for a particular skew group or sink within a skew group.
get_ccopt_skew_group_path	Returns a path for a particular skew group or sink within a skew group.
get_ccopt_skew_groups	Returns all skew groups.
get_ccopt_clock_tree_source_groups	Returns a list of clock tree source group objects matching the supplied pattern.
get_ccopt_preferred_cell_stripe	Returns a list of preferred cell stripes objects matching the supplied pattern.

Applying Library Cell Halos

Setting Cell Halos

CTS uses clock halos to enforce a minimum separation between clock instances in the design. CTS has two independent sets of properties to determine the x and y halos. The following properties can be used to define the x direction clock halos within CTS:

- cell_halo_x
- cell_density
- cell_halo_sites

The following properties can be used to define the y direction clock halos within CTS:

- cell_halo_y
- adjacent_rows_legal
- cell_halo_rows

By default, you will get halos associated with the cell_density (default = 0.75) and adjacent_rows_legal (default = false) properties. The halo value will be equal to 1/3 cell width in the x direction and 1 row high in the y direction.

For details about these properties, see CCOpt Properties.

Examples and Idiosyncrasies of the Clock Halo Properties

- `cell_halo_x` and `cell_halo_y` can be used to specify the clock halo for all cells, clock trees and power domains via:

  ```
  set_ccopt_property cell_halo_x 0.25
  set_ccopt_property cell_halo_y 0.2
  ```

  The default units for both `cell_halo_x` and `cell_halo_y` is microns.
- `cell_halo_x` and `cell_halo_y` can also be used to specify a clock halo for a particular cell type, clock tree and power domain via:

  ```
  set_ccopt_property -cell CKBD6BWP24P90ULVT -power_domain PD1 -clock_tree CLK cell_halo_x 0.5
  set_ccopt_property -cell CKBD6BWP24P90ULVT -power_domain PD1 -clock_tree CLK cell_halo_y 0
  ```

  `cell_halo_x` and `cell_halo_y` are the only halo properties that can specify the clock halo per clock tree, cell type, and power domain in this way.
- Any x direction clock halo set by `cell_density` scales linearly with cell width. The halo is defined by formula `halo_x = (1/a-1)*cell_width`, where `a` is the setting. Below are some examples:
  - `set_ccopt_property cell_density 0.25` sets the x direction clock halo equal to 3 * cell width.
  - `set_ccopt_property cell_density 0.5` sets the x direction clock halo equal to cell width.
  - `set_ccopt_property cell_density 0.75` sets the x direction clock halo equal to 1/3 * cell width.
  - `set_ccopt_property cell_density 1` sets the x direction clock halo equal to zero.
- Any y direction clock halo defined by `adjacent_rows_legal` is either zero or one row high. The following sets the clock halo in the y direction to one row high:

  ```
  set_ccopt_property adjacent_rows_legal false
  ```

  The following sets the clock halo in the y direction to zero:

  ```
  set_ccopt_property adjacent_rows_legal true
  ```
- The properties, `cell_halo_sites` and `cell_halo_rows` simply specify the clock halo in sites (for x direction) and rows for the y direction. For example, the following sets the clock halo to be 15 sites in the x direction and 2 rows in the y direction:

  ```
  set_ccopt_property cell_halo_sites 15
  set_ccopt_property cell_halo_rows 2
  ```

Clock Halo Priority Rules

Only one of these properties is used to determine the clock halo in the x direction. The following rules determine which one:

- If `cell_halo_x` (default is `auto`) is set to a non-auto value, then this defines the `x` direction clock halo. The properties `cell_density` and `cell_halo_sites` have no effect.
- If `cell_halo_x` is set to `auto` (default) and `cell_density` (default is `0.75`) is set to a non-auto value, then `cell_density` defines the clock halo in the x direction. The property, `cell_halo_sites` has no effect.
- If both `cell_halo_x` (default is auto) and `cell_density` (default is `0.75`) are set to auto then `cell_halo_sites` defines the clock halo in the x direction.

Only one of these properties is used to determine the clock halo in the `y` direction. The following rules determine which:

- If `cell_halo_y` (default is auto) is set to a non-auto value, then this defines the y direction clock halo. The properties, `adjacent_rows_legal` and `cell_halo_rows` have no effect.

- If `cell_halo_y` is set to `auto` (default) and `adjacent_rows_legal` (default is `false`) is set to a non-auto value, then `adjacent_rows_legal` defines the clock halo in the y direction. The property, `cell_halo_rows` has no effect.
- If both `cell_halo_y` (default) and `adjacent_rows_legal` (default is false) are set to auto then `cell_halo_rows` defines the clock halo in the y direction.

Effective Clock Halos

There are four read only properties to output the clock halo in microns and which property defined it. These are:

- `effective_clock_halo_x`
- `effective_clock_halo_y`
- `effective_clock_halo_x_source`
- `effective_clock_halo_y_source`

Examples

-
  ```
  get_ccopt_property -inst CTS_ccl_1234 effective_clock_halo_x
  0.720
  ```

-
  ```
  get_ccopt_property -inst CTS_ccl_1234 effective_clock_halo_x_source
  cell_density
  ```

-
  ```
  get_ccopt_property -inst CTS_ccl_1234 effective_clock_halo_y_source
  adjacent_rows_legal
  ```

Density Halos and Large Cells

As stated above “Any x direction clock halo defined by `cts_cell_density` scales linearly with cell width”. When the x direction halo is defined in this way and is applied to large cells this can create correspondingly large halos. Such halos on crowded floorplans can result in legalization problems.

For this reason, CTS internally disables halos on large cells defined using the attributes `cts_cell_density` and `cts_adjacent_rows_legal`. Upon disabling a halo the following log message will be issued:

```
**WARN: (IMPCCOPT-2406): Clock halo disabled on instance 'NAME'. Clock halo defined by properties 'cell_density' and
'adjacent_rows_legal'. Physical cell width = '36.000um' and height = '57.600um'. To avoid this please specify clock
halo via the property pairs (cell_halo_x and cell_halo_y) or (cell_halo_sites and cell_halo_rows).
```

As suggested in the message clock halos defined by other properties are never disabled.

Clock Halos and Siteless Cells

Clock halos are not assigned to instances corresponding to cell types without a site.

Clock Halo Sum Mode

The property, `cell_halo_mode` defines how clock halos are used to determine the minimum legal separation between a pair of clock instances. There are two possible modes; `max` and `sum`. When set to `max` the minimum legal separation is the larger of the two clock halos. When set to `sum` the minimum legal separation is the sum of the two clock halos.

The default setting is `max`.

Note: Reporting of clock halo compliance (when halos are defined by `cell_halo_x` and `cell_halo_y`) is available via the `report_ccopt_cell_halo_violations` command. For more information, see the Halo Violations section.

Enabling Timing Connectivity-Based Skew Groups

This feature deals with whether and how CTS should balance flops that are clocked by different SDC clocks. Balancing constraints are specified to CTS by means of skew groups. This feature controls the structure of the skew groups that are synthesized by spec creation.

Default Balancing Constraints

Default balancing constraints are used when the timing connectivity-based skew groups feature is disabled. By default, a fixed set of rules are used to determine which flops need to balance together.

- All clocks derived from the same ultimate master clock (i.e. non-generated clock) are synchronous. Flops that share the same ultimate master clock need to be balanced together.
- Clocks that are derived from different ultimate master clocks are asynchronous. Flops that do not share the same ultimate master clock do not need to be balanced together.

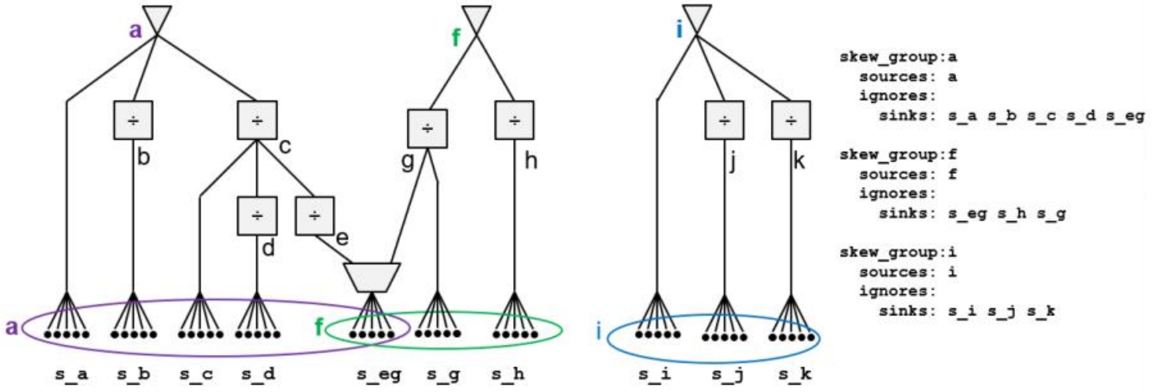
These rules are used to produce a set of balancing constraints to CTS: namely a set of skew groups. The default balancing constraints are as follows:

- Spec creation produces one skew group for each SDC clock.
- The skew group for a given SDC clock is shaped so that it spans the domain of that clock, and the domain of every clock derived from it, both directly and indirectly. This structure constrains CTS to balance the flops of all these clocks together.
- To remove redundant constraints, spec creation marks the skew group of each generated clock as constrains `none`.

Example of Default Balancing Constraints

The figure below illustrates default balancing constraints.

Default Balancing Constraints



```
skew_group:a
  sources: a
  ignores:
    sinks: s_a s_b s_c s_d s_eg

skew_group:f
  sources: f
  ignores:
    sinks: s_eg s_h s_g

skew_group:i
  sources: i
  ignores:
    sinks: s_i s_j s_k
```

In this clock network, a collection of SDC clocks is labelled from `a` to `k`. Clocks `a, f,` and `i` are non-generated clocks. The rest are derived clocks: b is generated by `a, e` is generated by `c`, and so on.
Given this clock structure, and using the default balancing constraints, spec creation produces eleven skew groups, one for each clock. However, almost all of them are constrains none. Only the skew groups for `a, f` ,and `i` will constrain CTS.

The skew group for clock a will source at the root pin of a. It will span the domain of clocks a, b, c, d, and e. Its sinks will comprise the flops s_a, s_b, s_c, s_d, and s_eg. All these flops will be balanced together.

Similarly, the skew group for clock f will source at the root pin of clock f. It will span the domain of clocks f, g, and h. It will sink on s_eg, s_g, and s_h, which will be balanced together. Note that flops s_eg appear in two constraining skew groups: those for a and f. This overlapping constraint arises from the clock mux that drives s_eg. Since these two skew groups overlap, their balancing constraints are inter-related. CTS will need to find a balancing solution that satisfies their constraints simultaneously.

Lastly, the skew group for clock i will source at the root pin of i and will span the domain of clocks i, j, and k. It will sink on s_i, s_j, and s_k, and these will be balanced together.

Timing Connectivity-Based Skew Groups

In the presence of set_clock_group and clock/clock set_false_path, the default balancing constraints can be needlessly restrictive. Such SDC constraints can false out the timing paths between a given pair of SDC clocks. In such cases, there is no need to balance their flops to the same insertion delay: slack is not affected by the relative clock path arrival times.

This is where timing connectivity-based skew groups come in. When this feature is enabled using the timing_connectivity_based_skew_groups property, spec creation analyzes the clock/clock false pathing, and identifies clocks and, therefore, flops that need not be balanced together. Spec creation then adjusts the skew groups that are produced to reflect these relaxed balancing constraints.

To do this, spec creation computes a balancing relationship for every pair of clocks:

- Two clocks "direct balance" if there are active (slack-generating; not falsed out) timing paths between them.
- Two clocks "indirect balance" if there are no active timing paths between them, but they each direct balance with some common third-party clock(s), which means they are forced to balance because of the transitive closure over direct balancing.
- Two clocks are in a "need not balance" relationship if they neither directly nor indirectly balance.

After analyzing the above, spec creation then adjusts the skew groups that it produces, relaxing the constraints between clocks that need not balance together:

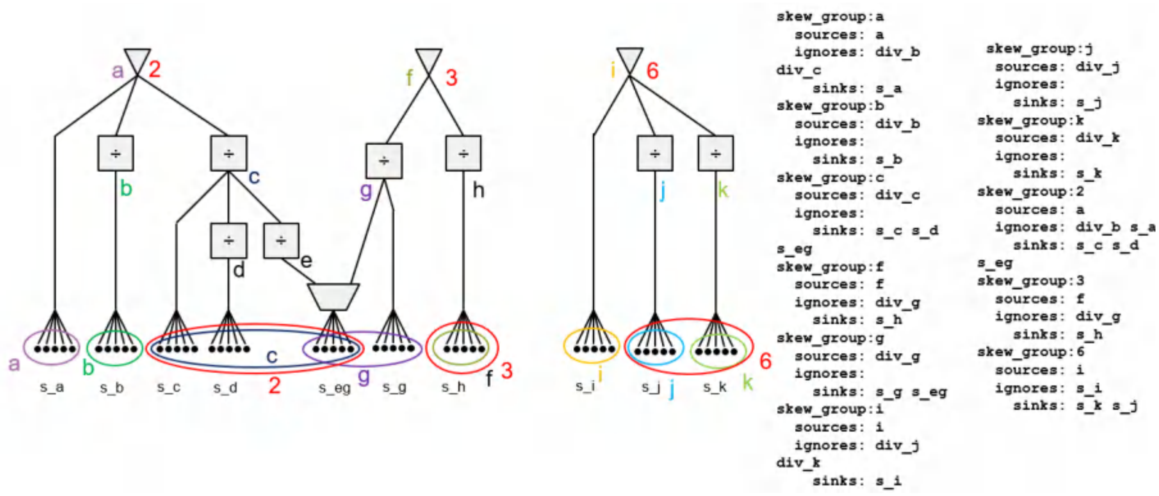
- Spec creation identifies that a generated clock need not balance with its master clock. To implement this, spec creation adds one or more ignore pins to the skew group of the master clock so that it does not span the domain of the generated clock.
- The skew group of a generated clock may no longer be a redundant constraint. In such a case it is not marked as “constrains none”.
- Additional clock group skew groups may be added to ensure balancing constraints between sibling-generated clocks are maintained.
- The generated spec includes configuration of the timing_connectivity_info property to record the clock/clock balancing relationships that were determined by spec creation. This includes details of the third-party clocks that produce “indirect balance” relationships.
- The generated spec, if written to Tcl file, includes a comment reporting on the clock/clock balancing relationships that were determined by spec creation, again including details of the third-party clocks involved in “indirect balance” relationships.

Note: Spec creation continues to assume that flops with differing ultimate master clocks do not need to be balanced together. In other words: enabling timing connectivity-based skew groups only relaxes balancing constraints between flops, it does not impose any new balancing constraints. This behavior can be controlled. See example below.

Example

The figure below shows a clock network. In this network, some set_clock_group and clock/clock set_false_path assertions are added, and timing connectivity-based skew groups are enabled. This change affects the skew groups produced by spec creation.

Clock False Paths



```
skew_group:a
  sources: a
  ignores: div_b
div_c
    sinks: s_a
skew_group:b
  sources: div_b
  ignores:
    sinks: s_b
skew_group:c
  sources: div_c
  ignores:
    sinks: s_c s_d
s_eg
skew_group:f
  sources: f
  ignores: div_g
    sinks: s_h
skew_group:g
  sources: div_g
  ignores:
    sinks: s_g s_eg
skew_group:i
  sources: i
  ignores: div_j
div_k
    sinks: s_i
```

```
 skew_group:j
  sources: div_j
  ignores:
    sinks: s_j
skew_group:k
  sources: div_k
  ignores:
    sinks: s_k
skew_group:2
  sources: a
  ignores: div_b s_a
    sinks: s_c s_d
s_eg
skew_group:3
  sources: f
  ignores: div_g
    sinks: s_h
skew_group:6
  sources: i
  ignores: s_i
    sinks: s_k s_j
```

```
set_clock_groups -asynchronous -group {a b} -group {c d e g} -group {f h i} -group {j k}

set_false_path -from [get_clocks {a}] -to [get_clocks {b}]

set_false_path -from [get_clocks {b}] -to [get_clocks {a}]

set_ccopt_property timing_connectivity_based_skew_groups clock_false_path

set_ccopt_property timing_connectivity_based_skew_groups_balance_master_clocks false

create_ccopt_clock_tree_spec
```

The understand the changes, see the default balancing constraints illustrated in previous figure. Relaxations are added to these constraints based on the SDC assertions.

Under the default balancing constraints, clocks a, b, c, d, and e would be balanced together under the skew group for clock a. However, the clock/clock set_false_path assertions mean that you no longer need to balance a with b. Furthermore, the set_clock_group assertion groups clocks, a and b separately from c, d, and e. Since c, d, and e are members of the same clock group, they must still balance together. However, none of them need to balance with a, and none of them need to balance with b.

You can continue in this way, analyzing the clocks into the five groups shown. Clocks in different groups need not balance together. For each pair of "need not balance" clocks, you can relax the skew group constraints in the produced spec.

For example, in the default balancing constraints there is a single skew group, a that balanced together flops s_a, s_b, s_c, s_d, and s_seg. However, clock a need not balance with any other clock. Therefore, you can adjust its skew group so that the flops s_a are balanced separately from all the other flops. A similar observation holds for clock b. Conversely, clocks c, d, and e must continue to balance together.

To implement these relaxations, following changes are made to the spec:

- Add ignore pins to the skew group for clock a at the generator inputs to clocks b and c. The skew group for clock a now only spans the domain of a, sinking at s_a only.
- The skew group for clock b is no longer marked constrains none.
- The skew groups for the other clocks are left unchanged.
- A special "clock group 2" skew group is synthesized to span clocks c, d, e, and g. This skew group is sourced on a, f and sinks on s_c, s_d, s_eg only: ignore pins are added to ensure it goes nowhere else.

The purpose of clock group 2 skew group is to impose a missing total path constraint. Note that the flops of c, d, e, g must balance together. This is true even though g belongs to a different ultimate master. This is because g shares flops in common with e. Notice that the insertion delay to the sources c and g is only constrained by the clock group skew group: no other skew group does this. Without the clock group 2 skew group you cannot ensure that all paths to the flops of c, d, e, and g are balanced.

Similarly, a clock group skew group for clocks `j` and `k` is needed. They must balance together, but need not balance with `i`, their master clock. The skew groups for j and k alone do not span all the paths you need to balance together. A clock group skew group is to be added to span both of `j` and `k`.

A clock group skew group is not always required. For example, `b` needs no such skew group: the latency to the source of `b` does not affect the balancing of the flops within `b`.

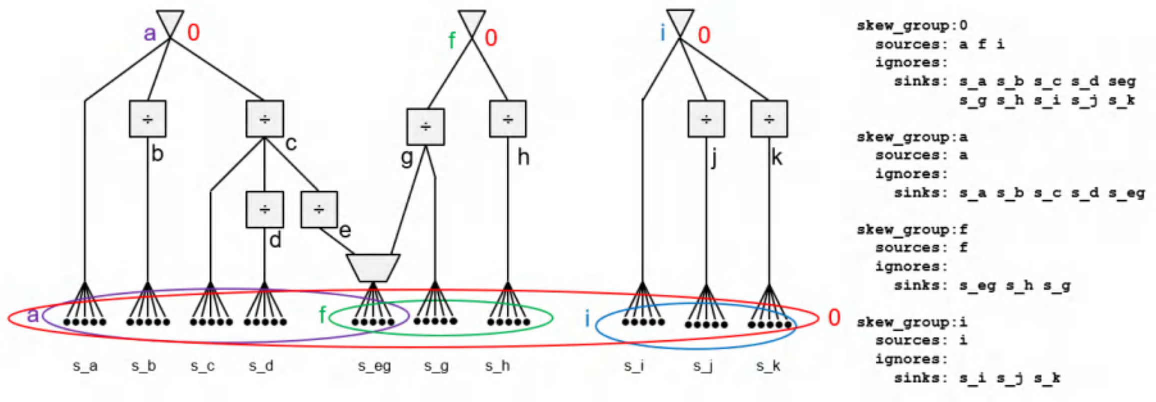
Balancing Ultimate Master Clocks

As detailed above, spec creation assumes that flops with differing ultimate master clocks do not need to be balanced together. This is true even when the timing connectivity-based skew groups feature is enabled. This behavior can be overridden with the `timing_connectivity_based_skew_groups_balance_master_clocks` property.

When this property is set to `true`, the default balancing constraints are changed. This is true regardless of the value of the `timing_connectivity_based_skew_groups` property.

The effect of this property on the default balancing relationships is illustrated in below figure.

Balance Master Clocks



```
skew_group:0
  sources: a f i
  ignores:
    sinks: s_a s_b s_c s_d seg
           s_g s_h s_i s_j s_k

skew_group:a
  sources: a
  ignores:
    sinks: s_a s_b s_c s_d s_eg

skew_group:f
  sources: f
  ignores:
    sinks: s_eg s_h s_g

skew_group:i
  sources: i
  ignores:
    sinks: s_i s_j s_k
```

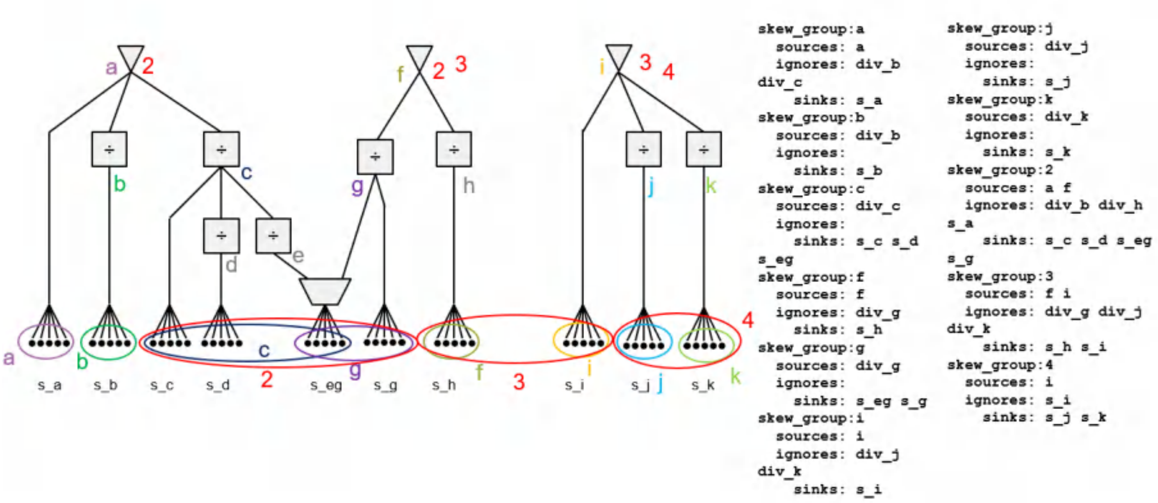
```
set_ccopt_property timing_connectivity_based_skew_groups_balance_master_clocks true
```

```
create_ccopt_clock_tree_spec
```

All clocks are treated as synchronous, and as such they must be balanced together. To implement this, spec creation synthesizes a clock group skew group spanning the entire clock network, sourcing on `a, f, i` and sinking at every flop.

This is a restrictive constraint and is typically not required when using the default balancing constraints. It has more use when timing connectivity-based skew groups are enabled. This is illustrated in below figure.

Balance Master Clocks and Clock False Paths



```
skew_group:a
  sources: a
  ignores: div_b
div_c
    sinks: s_a
skew_group:b
  sources: div_b
  ignores:
    sinks: s_b
skew_group:c
  sources: div_c
  ignores:
    sinks: s_c s_d
s_eg
skew_group:f
  sources: f
  ignores: div_g
    sinks: s_h
skew_group:g
  sources: div_g
  ignores:
    sinks: s_eg s_g
skew_group:i
  sources: i
  ignores: div_j
div_k
    sinks: s_i
skew_group:j
  sources: div_j
  ignores:
    sinks: s_j
skew_group:k
  sources: div_k
  ignores:
    sinks: s_k
skew_group:2
  sources: a f
  ignores: div_b div_h
s_a
    sinks: s_c s_d s_eg
s_g
skew_group:3
  sources: f i
  ignores: div_g div_j
div_k
    sinks: s_h s_i
skew_group:4
  sources: i
  ignores: s_i
    sinks: s_j s_k
```

```
set_clock_groups -asynchronous -group {a b} -group {c d e g} -group {f h i} -group {j k}

set_false_path -from [get_clocks {a}] -to [get_clocks {b}]

set_false_path -from [get_clocks {b}] -to [get_clocks {a}]

set_ccopt_property timing_connectivity_based_skew_groups clock_false_path

set_ccopt_property timing_connectivity_based_skew_groups_balance_master_clocks true

create_ccopt_clock_tree_spec
```

In this case, some structural changes are made to the clock group skew groups that reflect spec creation starting from the assumption that all clocks are synchronous. You can manage the balancing relationships imposed on CTS by means of the SDC clock/clock `set_false_path` and `set_clock_group` assertions.

The Clock/Clock Balancing Relationships Report

This report is only generated if timing connectivity-based skew groups are enabled. The report appears as a comment in the generated spec when it is written to the Tcl file.

The balancing report is divided into sections, one for each constraint mode. Within each section are reported, the inferred balancing relationships between the SDC clocks in that constraint mode. For each clock, the report displays its relationship with the other clocks in the mode, partitioning the other clocks into three categories depending on the relationship determined: direct balance; indirect balance; and need not balance.

This report is also written to the `timing_connectivity_info` property so that it can be queried from the command prompt and Tcl scripts.

Related Properties

This feature is implemented using the following CCOpt properties:

- `timing_connectivity_based_skew_groups`: Specifies whether SDC assertions are considered when creating the skew groups. When set to `off` the software uses the default balancing constraints. When set to `clock_false_path`, the clock/clock `set_false_path` and `set_clock_group` assertions are considered and used to shape the spec-generated skew groups.
- `timing_connectivity_based_skew_groups_balance_master_clocks`: Controls how CCOpt will address the synchrony of disjoint (i.e. non-overlapping) ultimate master SDC clocks. When set to `false`, all disjoint master clocks are assumed to be mutually asynchronous. When set to `true`, the synchrony of disjoint master clocks is determined by consulting the SDC assertions in

accordance with the value of property, `timing_connectivity_based_skew_groups`.

- `timing_connectivity_info`: This property is populated by spec creation to record the clock/clock balancing relationships that were determined for the production of timing connectivity based skew groups. This property is documenting only.

For detailed information about these properties, see CCOpt Properties.

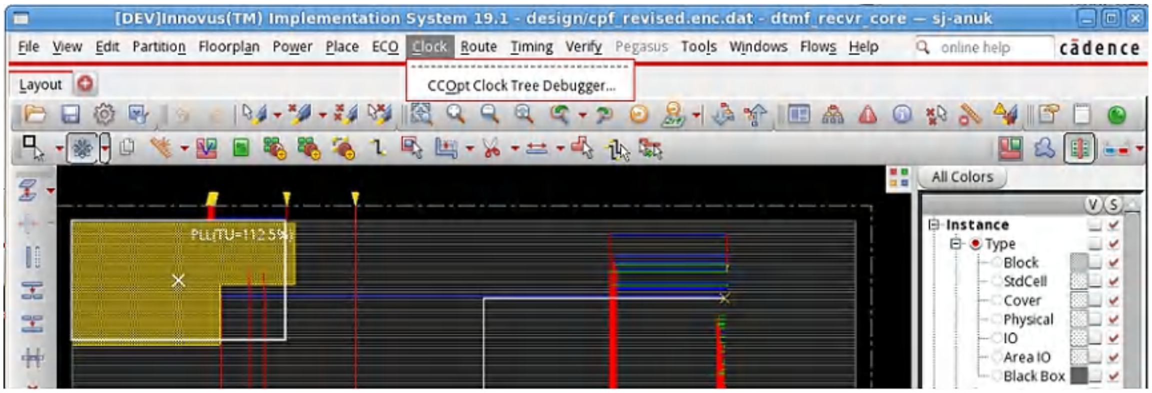
CCOpt Clock Tree Debugger

The CCOpt Clock Tree Debugger (CTD) provides a graphical interface to explore clock trees and provides debugging capabilities to aid understanding of CCOpt-CTS and CCOpt results. The interface is based on a top-down tree view of all defined clock trees with the vertical axis representing insertion delay. Cells and nets can be colored by various attributes, sections of the clock tree graph can be hidden and unhidden to facilitate navigation of complex clock architectures, and cross probing with the layout view is supported.

Many of the features of the debugger, for example, coloring are self-explanatory from exploring the interface. This section discusses some of the more in-depth features. For details of the *CCOpt Clock Tree Debugger*, see the "CCOpt Clock Tree Debugger" section in the Clock Menu chapter in *Innovus Menu Reference*.

Launching the CCOpt CTD

The tool can be accessed from the *Clock Menu* of Innovus. Choose *Clock < CCOpt Clock Tree Debugger*.

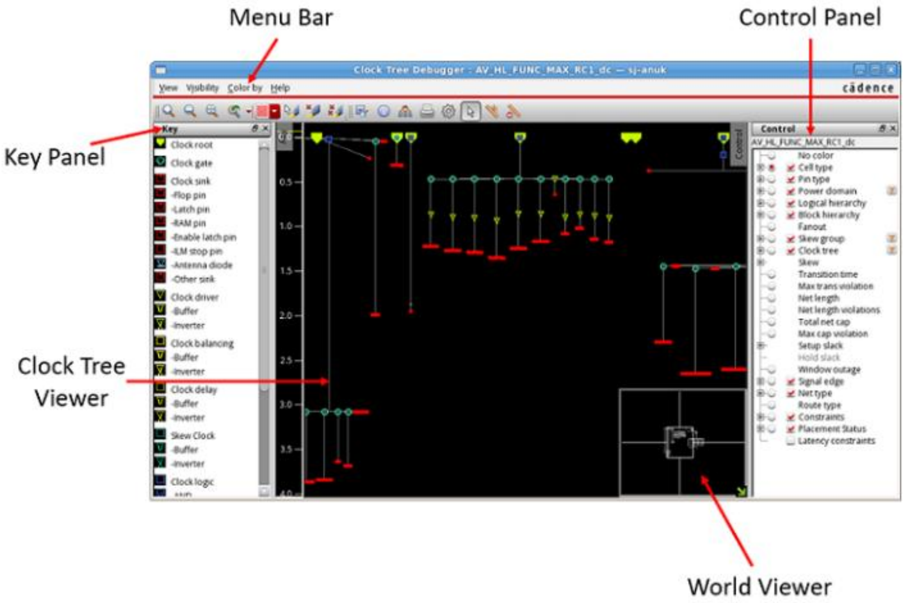


Note: Deleting any clock tree or skew group definitions will automatically close all open CTD windows. Multiple windows may be opened, and the following commands permit manipulation of the CTD windows:

Command Name	Usage
`ctd_win`	Open a debugger window. An ID and title can be optionally specified. The ID is used to identify this particular window when using the other commands below.
`ctd_save_histogram`	Saves the CTD histogram to a file.
`ctd_save_view`	Saves the current snapshot of the clock tree viewer into a file.
`close_ctd_win`	Closes either the specified window by ID, all windows, or the most recently active window.
`get_ctd_win_id`	Get the ID of the most recently active window or the IDs of all open windows.
`get_ctd_win_title`	Get the title if the specified window by ID, or the titles of all open windows.
`set_ctd_win_title`	Set the title of the most recently active window or the specified window by ID.

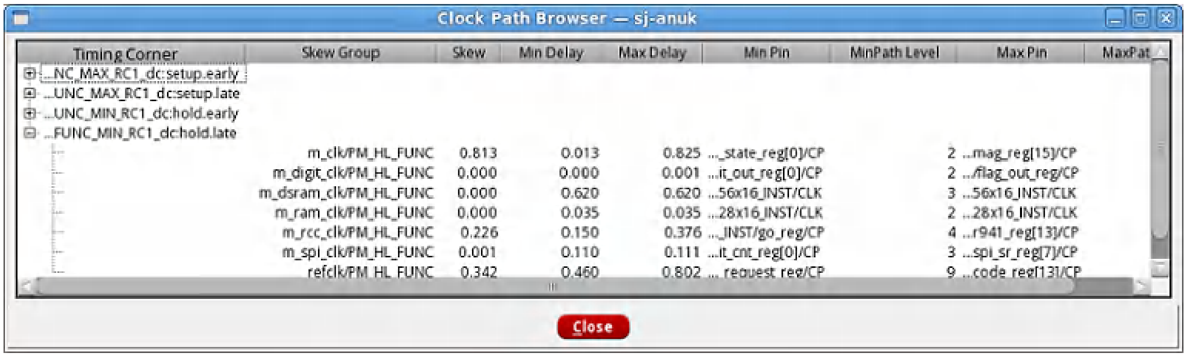
`ctd_trace`	Highlight the path to a sink in the active debugger window.
`gui_zoom_ctd`	Zooms into or out of the CTD window.

The main components in the CTD window are shown below.
CCOpt CTD – Main Window Components



- *Clock Tree Viewer* – Displays a top-down tree view of clock trees.
- *World Viewer* – Provides an overview even whilst the Clock Tree Viewer is zoomed in on a smaller region. Clicking in the world viewer will navigate to that area.
- *Control Panel* – Contains controls to determine visibility of different object types and coloring. Additional controls are available via the *View*, *Visibility*, and *Color by* menu items.
- *Key Panel* – A reference key indicating symbols and/or coloring used within the Clock Tree Viewer.

The *Path Browser*, which displays skew group path summary data in a table is available from the View menu of the CTD. It opens in a separate window as shown below.

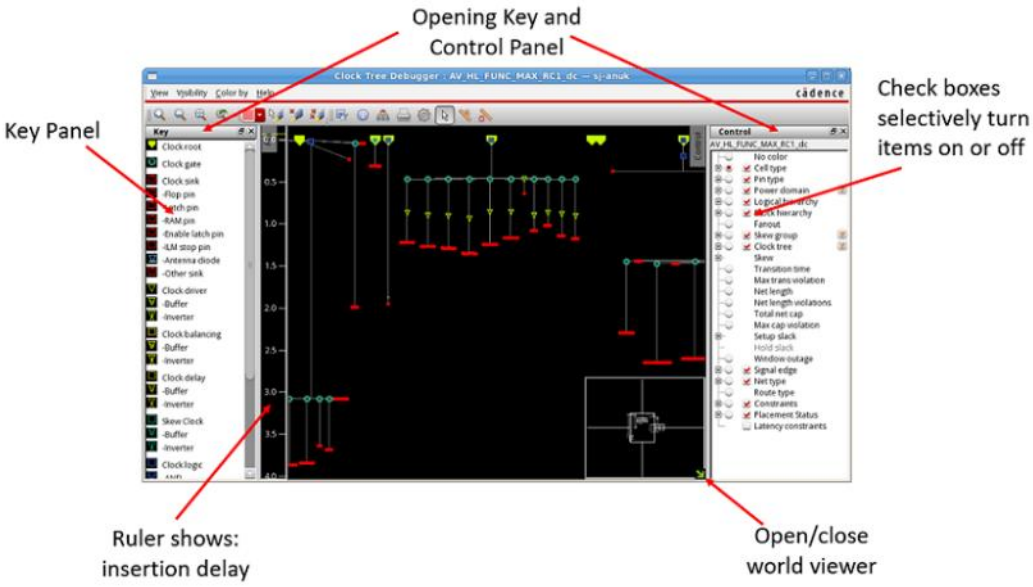


Timing Corner	Skew Group	Skew	Min Delay	Max Delay	Min Pin	MinPath Level	Max Pin	MaxPat
⊞ ...NC_MAX_RC1_dc:setup.early								
⊞ ...UNC_MAX_RC1_dc:setup.late								
⊞ ...UNC_MIN_RC1_dc:hold.early								
⊟ ...FUNC_MIN_RC1_dc:hold.late								
	m_clk/PM_HL_FUNC	0.813	0.013	0.825	..._state_reg[0]/CP	2	...mag_reg[15]/CP	
	m_digit_clk/PM_HL_FUNC	0.000	0.000	0.001	...it_out_reg[0]/CP	2	.../flag_out_reg/CP	
	m_dsram_clk/PM_HL_FUNC	0.000	0.620	0.620	...56x16_INST/CLK	3	...56x16_INST/CLK	
	m_ram_clk/PM_HL_FUNC	0.000	0.035	0.035	...28x16_INST/CLK	2	...28x16_INST/CLK	
	m_rcc_clk/PM_HL_FUNC	0.226	0.150	0.376	..._INST/go_reg/CP	4	...r941_reg[13]/CP	
	m_spi_clk/PM_HL_FUNC	0.001	0.110	0.111	...it_cnt_reg[0]/CP	3	...spi_sr_reg[7]/CP	
	refclk/PM_HL_FUNC	0.342	0.460	0.802	... request reg/CP	9	...code reg[13]/CP	

Double-clicking on a row or using the right-click context menu permits opening the *Clock Path Analyzer* window. By default, the Path Browser opens at the bottom of the window. The *Clock Path Analyzer*, when invoked, replaces the *Clock Path Browser.*

By default, the *Control Panel* and *Key Panel* are hidden. These panels can be exposed or hidden as illustrated below.

CCOpt CTD – Opening the Key and Control Panel



Key Features of the CTD

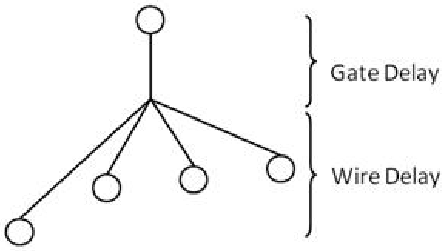
The key features of the CTD are briefly explained in the subsequent sections.

For details of the *CCOpt Clock Tree Debugger*, see the "CCOpt Clock Tree Debugger" section in the Clock Menu chapter in *Innovus Menu Reference* .

Clock Tree Representation

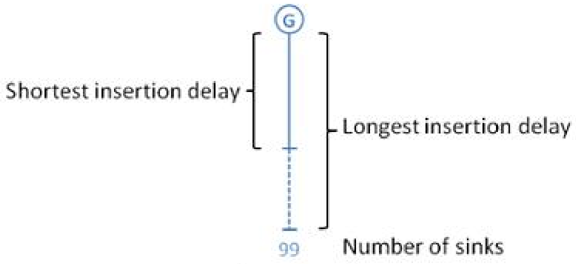
Clock trees are drawn in a top-down tree like manner with the vertical axis representing insertion delay. Different symbols are used for differing cell types, for example buffers, inverters, clock gates, logic and sinks. Gate and wire delay are separately represented, as illustrated below.

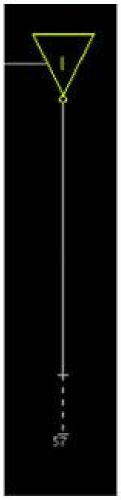
Gate and Wire Delay



Expanding and Collapsing Sub-trees

Any node in the tree may be either expanded or collapsed. The sub-trees of collapsed nodes are shown as a vertical summary bar indicating the maximum and minimum insertion delay below the node, and the number of sinks in the sub-tree, as illustrated below.



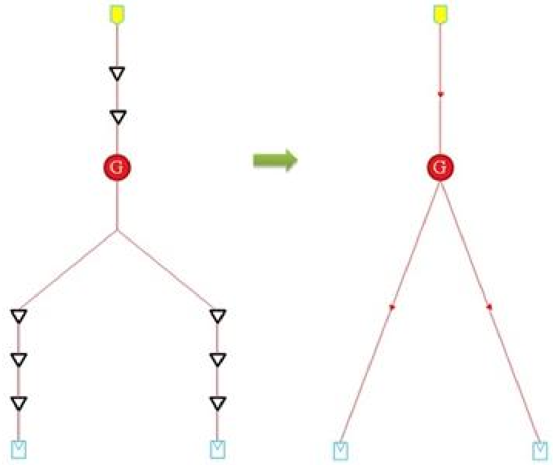


The expanded and collapsed state of a node may be toggled by double-clicking on the node, or using the Expand/Collapse item on the context menu. Additionally, a sub-tree can be marked as un-collapsible by using the context menu. An un-collapsible sub-tree will not be collapsed when its parent is collapsed.

Simplification

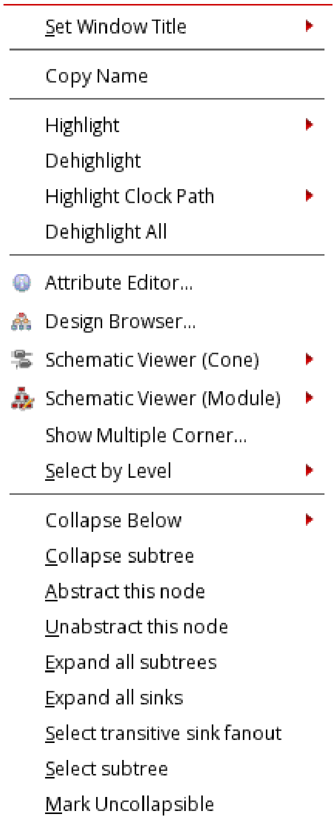
The *View – Simplify* option in the menu bar can be used to further manage visibility, including the following:

- *Mark All Collapsible* – Mark all nodes as collapsible.
- *Collapse all* – Collapse all sub-trees such that each clock tree is fully collapsed.
- *Expand All* – Expands everything so that no subtree is collapsed.
- *Expand All by Skew Group* – Expand all sub-trees that pass through the specified skew group.
- *Hide* – Specified cell types or cell instances can be hidden using the *View -> Simplify -> Hide* menu option. Hidden cells are simply omitted, as represented in the diagram below where buffer cells, but not clock gates, have been hidden.
- *Show unhidden* – Expands all the hidden instances and cells in the Clock Tree Viewer.



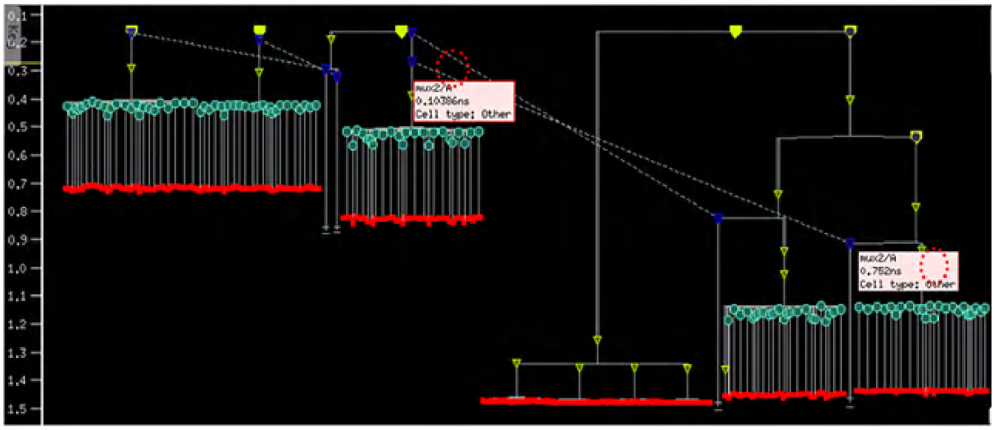
Context Menu

Right-clicking on a cell opens a context menu. This menu permits various operations to be performed, such as copying the cell name, highlighting the cell, highlighting paths to the cell, opening a schematic viewer, or opening cell properties.



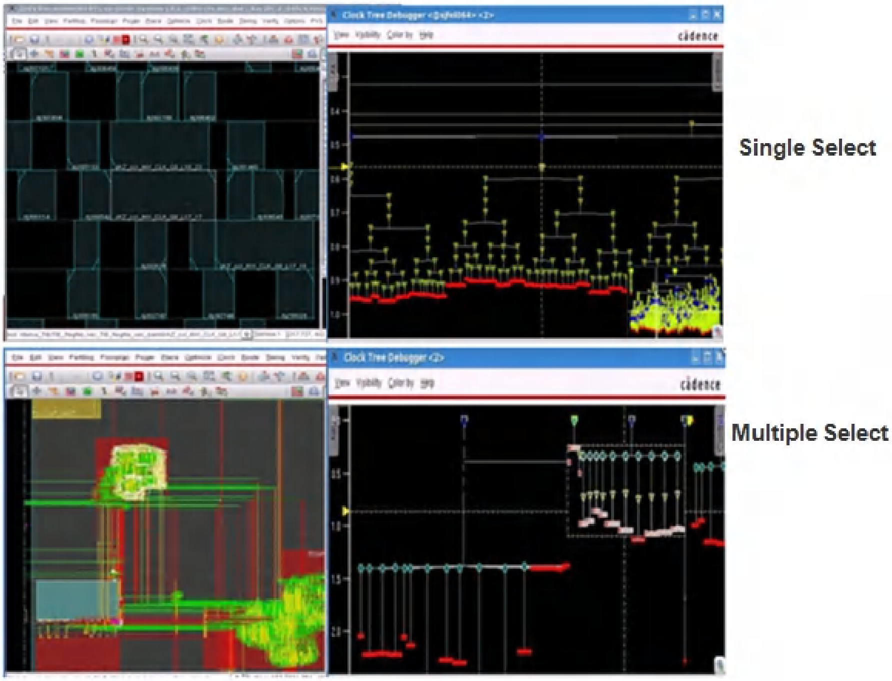
Multiple Input Cells

A multiple input cell, for example a multiplexer, can exist in more than one clock tree. Multiple input cells are shown in multiple clock trees, but the sub-tree underneath the cell can only be expanded in one clock tree at a time. A dotted line is drawn between the instances of the same cell as illustrated below.



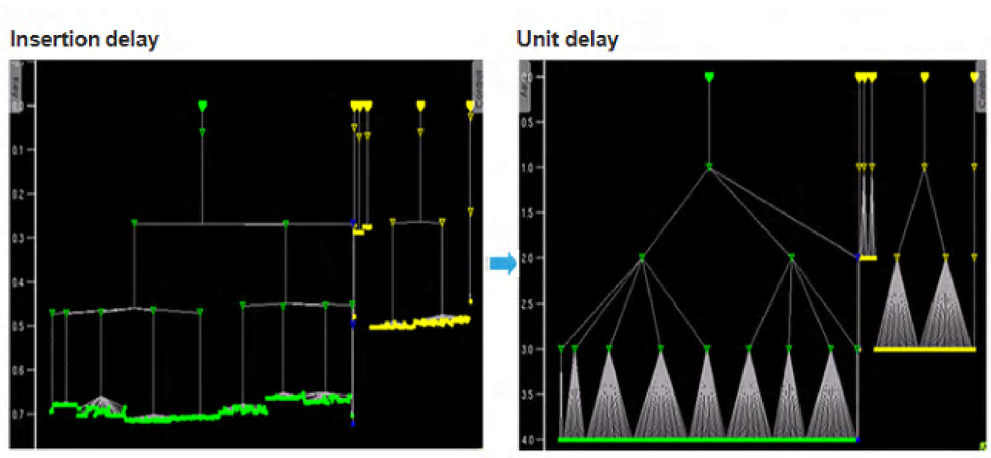
Cross Probing

When an object is selected in the main Innovus layout window it will also be selected in the CTD window. Conversely, objects selected in the debugger window will be selected in the layout window. The right-hand mouse button can be used to draw a bounding box to perform multiple selections.



Unit Delay

The *Visibility – Unit Delay* menu option changes to a unit delay model where each cell has a delay of 0 and each wire a delay of 1.0. This mode is useful for inspecting the clock graph structure before running CCOpt or CCOpt-CTS.



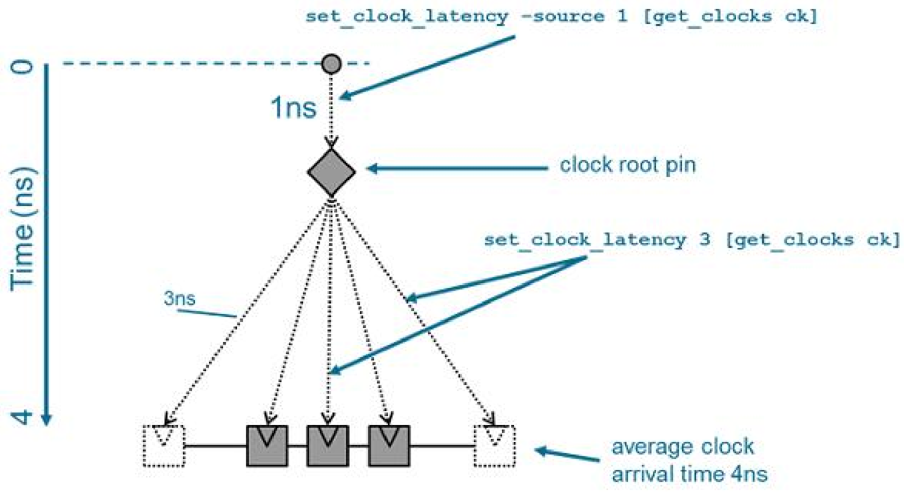
Additional Topics

Source Latency Update

Source latency update is performed to ensure that after CTS when clocks are switched to propagated mode that I/O timing and inter-clock timing is consistent with the ideal mode timing model. Stated succinctly:

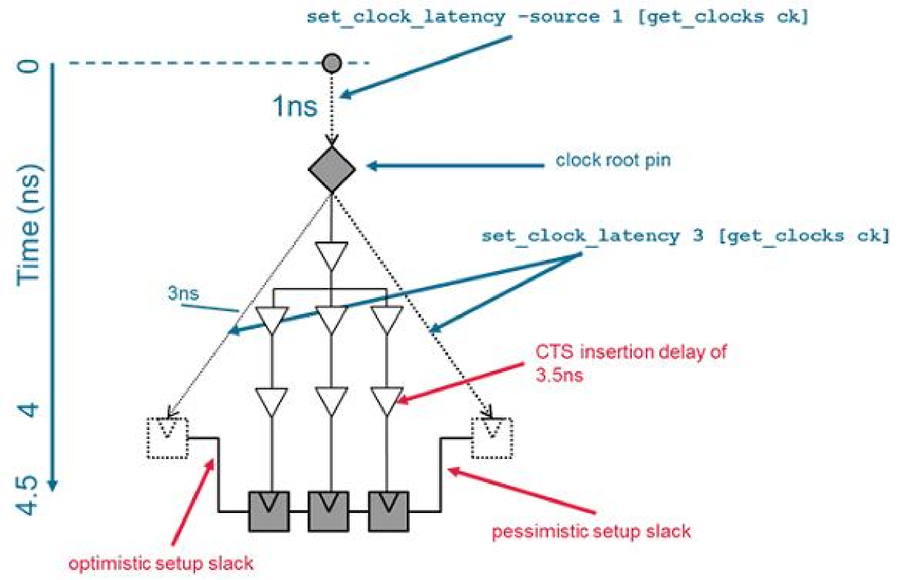
- The source latency update step updates the source latencies at clock root pins such that the per clock average clock arrival time is identical postCTS as it was preCTS.
- This mechanism is best explained using an example. The diagram below represents the before CTS ideal clock mode timing. In this example, there is a single clock with a clock source latency of 1ns and a network latency of `3ns`. Therefore, the average clock arrival time at both the I/O pins (represented by dotted flops) and the real sinks is `4ns`.

Source Latency Update – Before CTS


set_clock_latency -source 1 [get_clocks ck]
0
1ns
clock root pin
Time (ns)
set_clock_latency 3 [get_clocks ck]
3ns
4
average clock
arrival time 4ns

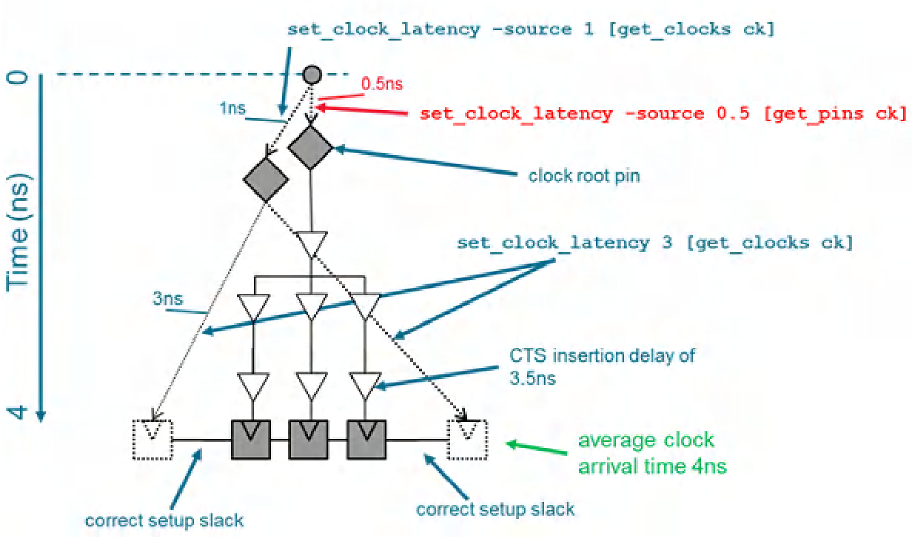
The next diagram below illustrates what would happen if CTS were performed but instead of achieving a 3ns insertion delay CTS achieves a `3.5ns` insertion delay. The clock arrival time at the I/O pins is unchanged, but the clock arrival time at the real flops is now 3.5ns. This results in optimistic setup slack on input paths and pessimistic setup slack on output paths.

Source Latency Update – CTS without Source Latency Update



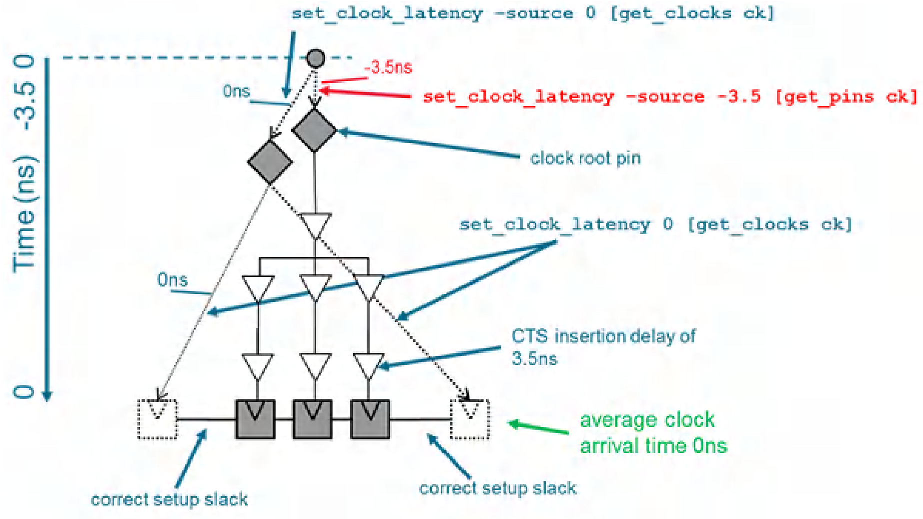
The diagram below illustrates the effect of source latency update. The clock source latency and network latency are unchanged, and the I/O pin timing is unchanged. The clock root pin has the source latency overridden to be 0.5ns instead of 1ns. This adjusts the clock arrival time at the real sinks such that it remains at 4ns. The input paths and output paths are now timed in the manner which was intended preCTS. Unlike other ‘I/O latency modification’ schemes, this scheme operates correctly in the presence of multiple clock domains communicating with I/O pins without any need for averages or need to match up virtual clocks with real clocks. The source latency modification is performed per clock root pin per clock, such that cross-clock timing consistency is also maintained from before CTS to after CTS. Virtual clocks, if present, do not need modification.

Source Latency Update – CTS with Source Latency Update


set_clock_latency -source 1 [get_clocks ck]
0
0.5ns
1ns
set_clock_latency -source 0.5 [get_pins ck]
clock root pin
Time (ns)
set_clock_latency 3 [get_clocks ck]
3ns
CTS insertion delay of 3.5ns
4
average clock arrival time 4ns
correct setup slack
correct setup slack

The diagram below is the same example but with the initial "before CTS" clock and network latencies both at zero. This gives rise to a negative source latency set at the clock root pin. Before CTS the average clock arrival time is zero and this is maintained after CTS via the source latency update.

Source Latency Update – Variation with Zero Initial Latencies


set_clock_latency -source 0 [get_clocks ck]
0
-3.5
-3.5ns
0ns
set_clock_latency -source -3.5 [get_pins ck]
clock root pin
Time (ns)
set_clock_latency 0 [get_clocks ck]
0ns
CTS insertion delay of 3.5ns
0
average clock arrival time 0ns
correct setup slack
correct setup slack

In the CCOpt-CTS flow, the source latency update step is performed near the end. In the CCOpt flow, the source latency update step is performed after initial virtual delay balancing before timing optimization and useful skew scheduling commences. The source latency updates are reflected in the Innovus timing constraints and will be saved in any saved database or exported in SDC as normal.

In a block-level design with multiple clocks it is likely that each clock will obtain a different source latency modification. For example, a small clock tree might have a source latency modification of `-0.5ns` while a large clock tree might have a source latency modification of `-2ns`. This correctly maintains the validity of the timing constraints which were present before CTS for after CTS usage. However, it means that the `1.5ns` difference between the small and large clock tree needs to be synthesized at the top level, but it is usually considerably more efficient to do this at the top level than it is at a block level. Typically, at the top level, an ILM model of blocks is used, in which case CTS will be able to directly see the clock paths inside the ILM so the users need take no action to configure this `1.5ns` offset.

By default, the software updates all loaded views for a design and not only those that are currently active.

The source latency modification scheme can be disabled using `set_ccopt_property update_io_latency false`. The latency modification scheme should be disabled for top-level chips as balancing clocks outside the chip is unlikely to be practical or if the flow requires balancing between clocks to be performed inside the block level. When latency modification is disabled, the designer needs to correctly estimate the achievable clock tree insertion delay and configure clock network latencies accordingly to avoid a timing jump over CTS and the switch to propagated clock mode timing.

Converting Library Path Delays to Clock Latencies

A new command `convert_lib_clock_tree_latencies` is provided to convert the Liberty `max_clock_tree_path/min_clock_tree_path` (MCTP) data to per pin clock latency adjustments.Run this command before running the `create_ccopt_clock_tree_spec` command so that the output of `convert_lib_clock_tree_latencies` impacts the clock tree specification that is created. It is also recommended to use the command before `place_opt_design` to ensure placement and pre-CTS optimization observes the latency adjustments. The syntax of the command is as follows:

```
[-help]
[-latency_file_prefix prefix_name]
[-pins pin_list]
[-views view_list]
[-override_existing_latencies | -sum_existing_latencies]
[-sum_existing_latencies | -override_existing_latencies_pins pin_list]
[-override_existing_latencies | -sum_existing_latencies_pins pin_list]
[-sum_existing_latencies | -sum_existing_latencies_pins pin_list]
[-override_existing_latencies | -override_existing_latencies_pins]
```

It is required that the analysis mode be set to on-chip variation (OCV) because the conversion process computes individual early and late network latency values. This can be achieved by using the below command:
`setAnalysisMode -analysisType onChipVariation`

Views – By default all active setup and active hold analysis views are considered. The `-views` option can be used to restrict to a sub-set of the active views.

By default, the conversion will consider all instance pins with MCTP data in all analysis views. Pins which have any existing network latency, differing from the corresponding clock network latency will be skipped completely for all clocks and for all views – this avoids the conversion process from interfering with any SDC specified pin latencies. Alternatively, existing latency offsets can be summed or overridden for all or names pins using the appropriately named override and sum option. Additionally, the `-pins` option can be used to consider only a specified list of pins instead of all pins with MCTP data.

By default, the conversion will internally apply the resulting `set_clock_latency` commands to the in memory timing constraints. Alternatively the `-latency_file_prefix` parameter can be specified to have one file per analysis view written containing the `set_clock_latency` commands.

Examples:

Example 1 – Single clock, no clock network latency, no pre-existing latency overrides

```
Circuit
ck1 --> icg --+--> flop1/CK (regular flop pin without MCTP data)
+--> macro1/CK (pin with MCTP data, of 0.222R/0.232F at 0.125 transition time)
+--> macro2/CK (pin with MCTP data, of 0.333R/0.343F at 0.125 transition time)
```

```
SDC
create_clock -period 1.0 [get_ports ck1]
set_clock_transition 0.125 [get_clocks ck1]
```

```
Command and log output
convert_lib_clock_tree_latencies -latency_file_prefix lat_ -views default_analysis_view_setup
Converting library clock tree path delays to clock latencies for analysis_view default_analysis_view_setup
Found 2 of 3 clock endpoints with library clock tree path data
+-----------------------------------------------+
| Clock endpoint | Status |
|------------------------+----------------------|
| macro1/ck | converted |
| macro2/ck | converted |
+-----------------------------------------------+
Writing latencies to file 'lat_default_analysis_view_setup.sdc'
```

```
Output file
set_clock_latency -0.222 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -rise
set_clock_latency -0.232 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -fall
set_clock_latency -0.333 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -rise
set_clock_latency -0.343 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -fall
```

Example 2 – Single clock, no clock network latency, pre-existing latency overrides

```
Circuit

  ck1 --> icg --+--> flop1/CK   (regular flop pin without MCTP data)

              +--> macro1/CK  (pin with MCTP data, of 0.222R/0.232F at 0.125 transition time)

                +--> macro2/CK  (pin with MCTP data, of 0.333R/0.343F at 0.125 transition time)

SDC

  create_clock -period 1.0 [get_ports ck1]

  set_clock_transition 0.125 [get_clocks ck1]

  set_clock_latency -0.300 [get_pins {macro1/ck}]

Command and log output

convert_lib_clock_tree_latencies -latency_file_prefix lat_ -views default_analysis_view_setup

Converting library clock tree path delays to clock latencies for analysis_view default_analysis_view_setup

Found 2 of 3 clock endpoints with library clock tree path data

  +-----------------------------------------------------------+

  | Clock endpoint         | Status                          |

  |------------------------+---------------------------------|

  | macro1/ck              | skipped, existing latency found |

  | macro2/ck              | converted                       |

  +-----------------------------------------------------------+

Writing latencies to file 'lat_default_analysis_view_setup.sdc'

Output file

  set_clock_latency -0.333 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -rise

  set_clock_latency -0.343 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -fall
```

Example 3 – Single clock, no clock network latency, pre-existing latency overrides, using the -override_existing_latencies option

```
Circuit
```

```
  ck1 --> icg --+--> flop1/CK   (regular flop pin without MCTP data)

                +--> macro1/CK  (pin with MCTP data, of 0.222R/0.232F at 0.125 transition time)

                +--> macro2/CK  (pin with MCTP data, of 0.333R/0.343F at 0.125 transition time)

SDC

  create_clock -period 1.0 [get_ports ck1]

  set_clock_transition 0.125 [get_clocks ck1]

  set_clock_latency -0.300 [get_pins {macro1/ck}]

Command and log output

convert_lib_clock_tree_latencies -latency_file_prefix lat_ -override_existing_latencies -views
default_analysis_view_setup

Converting library clock tree path delays to clock latencies for analysis_view default_analysis_view_setup

Found 2 of 3 clock endpoints with library clock tree path data

  +-------------------------------------------------------------------+

  | Clock endpoint         | Status                                   |

  |------------------------+------------------------------------------|

  | macro1/ck              | converted, existing latency overwritten |

  | macro2/ck              | converted                                |

  +-------------------------------------------------------------------+

Writing latencies to file 'lat_default_analysis_view_setup.sdc'

Output file

  set_clock_latency -0.222 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -rise

  set_clock_latency -0.232 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -fall

  set_clock_latency -0.333 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -rise

  set_clock_latency -0.343 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -fall
```

Example 4 – Single clock, no clock network latency, pre-existing latency overrides, using the -sum_existing_latencies option

```
Circuit

  ck1 --> icg --+--> flop1/CK   (regular flop pin without MCTP data)

                +--> macro1/CK  (pin with MCTP data, of 0.222R/0.232F at 0.125 transition time)

                +--> macro2/CK  (pin with MCTP data, of 0.333R/0.343F at 0.125 transition time)

SDC

  create_clock -period 1.0 [get_ports ck1]

  set_clock_transition 0.125 [get_clocks ck1]

  set_clock_latency -0.300 [get_pins {macro1/ck}]

Command and log output

  convert_lib_clock_tree_latencies -latency_file_prefix lat_ -sum_existing_latencies sum -views
default_analysis_view_setup

  Converting library clock tree path delays to clock latencies for analysis_view default_analysis_view_setup
  Found 2 of 3 clock endpoints with library clock tree path data

  +------------------------------------------------------------------+

  | Clock endpoint         | Status                                  |

  |------------------------+-----------------------------------------|

  | macro1/ck              | converted, existing latency summed     |

  | macro2/ck              | converted                               |

  +------------------------------------------------------------------+

  Writing latencies to file 'lat_default_analysis_view_setup.sdc'

Output file

  set_clock_latency -0.522 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -rise

  set_clock_latency -0.532 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -fall

  set_clock_latency -0.333 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -rise

  set_clock_latency -0.343 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -fall
```

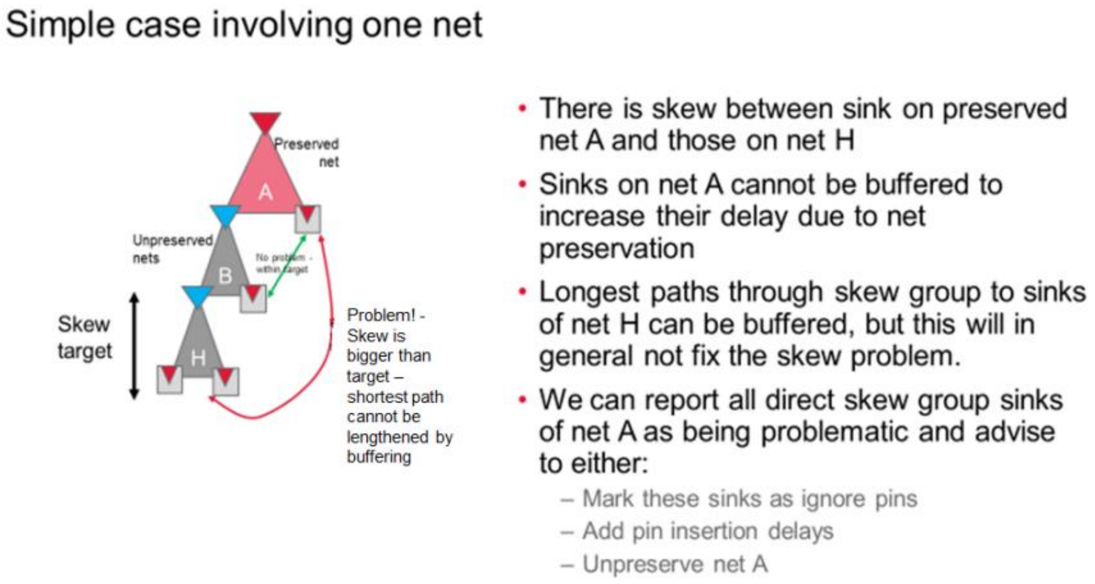
Example 5 – Two clocks, existing clock network latency, pre-existing latency override, using the -override_existing_latencies_pins and –sum_existing_latencies_pins options

```
Circuit

  ck1 ---> +-----+

           | mux | -----> icg --+--> flop1/CK    (regular flop pin without MCTP data)

  ck2 ---> +-----+              +--> macro1/CK   (pin with MCTP data, of 0.222R/0.232F at 0.125 transition time)

                                +--> macro2/CK   (pin with MCTP data, of 0.333R/0.343F at 0.125 transition time)

SDC

  create_clock -period 1.0 [get_ports ck1]

  create_clock -period 1.0 [get_ports ck2]

  set_clock_transition 0.125 [get_clocks ck1]

  set_clock_transition 0.125 [get_clocks ck2]

  set_clock_latency 1.5 [get_clocks {ck1}]

  set_clock_latency 2.0 [get_clocks {ck2}]

  set_clock_latency 1.2 [get_pins {macro1/ck}] -clock [get_clocks {ck1}]

  set_clock_latency 1.7 [get_pins {macro2/ck}] -clock [get_clocks {ck1}]

Command and mock-up log output

convert_lib_clock_tree_latencies -latency_file_prefix lat_ -override_existing_latencies_pins macro1/ck -
sum_existing_latencies_pins macro2/ck -views default_analysis_view_setup

Converting library clock tree path delays to clock latencies for analysis_view default_analysis_view_setup

Found 2 of 3 clock endpoints with library clock tree path data

  +------------------------------------------------------------------+

  | Clock endpoint         | Status                                  |

  |------------------------+-----------------------------------------|

  | macro1/ck              | converted, existing latency overwritten |
  | macro2/ck              | converted, existing latency summed      |

  +--------------------------------------------------------------------+

Writing latencies to file 'lat_default_analysis_view_setup.sdc'

Output file (the comments should not be written in the file)

# 1.5 is used since the existing latency override (1.2 instead of 1.5) is itself to be overwritten

  set_clock_latency 1.278 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -rise /1.5-0.222=1.278

  set_clock_latency 1.268 [get_pins {macro1/CK}] -clock [get_clocks {ck1}] -fall /1.5-0.232=1.268

  set_clock_latency 1.778 [get_pins {macro1/CK}] -clock [get_clocks {ck2}] -rise /2.0-0.222=1.778

  set_clock_latency 1.768 [get_pins {macro1/CK}] -clock [get_clocks {ck2}] -fall /2.0-0.232=1.768 # 1.7 is used since
the existing latency override is to be included (summed)

  set_clock_latency 1.367 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -rise /1.7-0.333=1.367

  set_clock_latency 1.357 [get_pins {macro2/CK}] -clock [get_clocks {ck1}] -fall /1.7-0.343=1.357

# 2.0 is used since there is no existing latency override for this clock. Note the log messaging will indicate
"existing latency summed" if applicable for any clock. Putting per clock reporting in the log seems to verbose to be
useful.

  set_clock_latency 1.667 [get_pins {macro2/CK}] -clock [get_clocks {ck2}] -rise /2.0-0.333=1.667

  set_clock_latency 1.657 [get_pins {macro2/CK}] -clock [get_clocks {ck2}] -fall /2.0-0.343=1.657
```

Debugging Unresolvable Skew Targets

As part of the CTS configuration, you may specify a desired skew target and expect the tool to achieve this. However, you may also mark some nets as being preserved using the dont_touch flag. This preservation prevents buffering of that net and might, in some circumstances, prevent CTS from achieving the desired skew target.

There are many ways in which net preservation can cause skew issues. A simple net case involving one net is detailed below.



Simple case involving one net

Preserved net
A
Unpreserved nets
B
No problem - within target
Skew target
H
Problem! - Skew is bigger than target – shortest path cannot be lengthened by buffering

- There is skew between sink on preserved net A and those on net H
- Sinks on net A cannot be buffered to increase their delay due to net preservation
- Longest paths through skew group to sinks of net H can be buffered, but this will in general not fix the skew problem.
- We can report all direct skew group sinks of net A as being problematic and advise to either:
  - Mark these sinks as ignore pins
  - Add pin insertion delays
  - Unpreserve net A

The term, “simple” here refers to single clock nets:

- That are preserved - marked as dont_touch or ideal network

- Have a clock sink directly attached, where the sink is configured to be a skew group sink for a skew group, and
- For that skew group, the difference in latency between the driver of the net and the sink vs. the longest path delay from the driver of the net is bigger than the skew target.

The software detects and reports simple cases of net preservation. The course of action depends upon whether the `ignore_problematic_skew_as_result_of_dont_touch_nets` property is set or not. When it is set to `true`, the software ignores sinks directly connected to nets that are causing unfixable skew problems for skew balancing. By default, this property is set to `false`.

For details, see CCOpt Properties.

The software takes one of the following actions:

1. Issues a warning requiring you to fix the setup to address the problem
2. Issues a warning, and internally modifies the balancing configuration to fix the problem

Scenario 1:

The following warning is issued when problematic preserved nets are found and the `ignore_problematic_skew_as_result_of_dont_touch_nets` property is not set.

Type `man <message_ID>` to read the complete message.

```
NAME

       IMPCCOPT-1482

SUMMARY

       Found  %d  dont_touch  nets whose immediate fanout are skew group sinks

       causing unfixable skew problems in skew groups. This may  impact  clock

       tree skew balancing QoR.

DESCRIPTION

       This  message  occurs  when a dont_touch net is found in the clock tree

       that will stop the skew target being achieved due to not being able  to

       add buffers to that net. When observed, the user should review the sub-

       sequent table of net names and pins attached to those nets  and  choose

       to:

       1.Ignore  the  message.  The  skew  of the listed skew groups will be

       impacted by the amount listed in the table and lower skew targets  will

       not be achievable.

     2. Remove  the  dont_touch from the nets named in the table. This will

       allow the tool to buffer the nets to fix the skew problem.

       3.Mark the skew group sink pins in the immediate fanout of the net  as

       ignore  pins,  to  take  them out of the balancing problem for all skew

       groups.

       4.Remove the immediate fanout pins from the skew group sink  list  for

       the affected skew groups.

       5.Set  the ignore_problematic_skew_as_result_of_dont_touch_nets prop-

       erty and rerun.
```

Scenario 2:

In the case where the `ignore_problematic_skew_resulting_from_dont_touch_nets` property is set to `true`, a different message will be emitted.

Type `man <message_ID>` to read the complete message.

```
NAME
       IMPCCOPT-1483
SUMMARY
       Found  %d  dont_touch  nets whose immediate fanout are skew group sinks
       causing unfixable skew problems. %d sinks directly connected  to  these
       nets have been ignored for skew balancing.
DESCRIPTION
       This  message  occurs  when a dont_touch net is found in the clock tree
       that would stop the skew target being achieved due to not being able to
       add  buffers  to  that net. The tool has internally marked the impacted
       sinks as ignore pins such that overall skew  balancing  should  not  be
       impacted by these sinks.
       One  consequence  of  this message is that the skew numbers seen in the
       log during the run will not match the skew  numbers  reported  post-CTS
       using report_ccopt_skew_groups.
       To resolve this warning, the user should review the subsequent table of
       net names and pins attached to those nets and choose to:
       1. Ignore the message. When reporting post-CTS, the skew of the  listed
       skew  groups will be impacted by the amount listed in the table for the
       sinks listed in the table.
       2. Remove the dont_touch constraint from the nets named in  the  table.
       This will allow the tool to buffer the nets to fix the skew problem.
       3.  Mark the skew group sink pins in the immediate fanout of the net as
       ignore pins. To take them out of the balancing  problem  for  all  skew
       groups,  the  following tcl may be used to identify dont_touch nets and
       mark the sinks directly attached as ignore pins for balancing:
   set skew_group_sinks {}
       foreach net [get_db -u clock_trees .nets -if {.is_dont_touch}] {
       set is_problem_net false
       foreach load [get_db $net .loads] {
       foreach sg [get_db $load .cts_skew_groups_active_sink] {
       dict lappend skew_group_sinks $sg $load
       set is_problem_net true
       }
       }
       if {$is_problem_net} {
    Puts "Found problematic skew group sinks on preserved net $net"
       }
       foreach sg [dict keys $skew_group_sinks] {
       modify_ccopt_skew_group  -skew_group  $sg  -add_ignore_pins  [dict  get
       $skew_group_sinks $sg]
       }
       4.  Remove  the immediate fanout pins from the skew group sink list for
       the affected skew groups.
```

Output

A sample output is shown below:

```
Preserved nets causing skew problems:
==========================================
--------------------------------------------------------------------------
  Net   Skew       Skew     Skew         Num sinks   Example
        group      target   Achievable   on net      sink
                     (ns)     (ns)
--------------------------------------------------------------------------
wire1  clk/func    0.100    0.516        2          top/ffer333/reg12/CK
wire2  clk/test    0.200    0.711        16         top/iiiwww3/reg34/CK
--------------------------------------------------------------------------
```

Updating Annotations After Clock Tree Specification Creation

This feature allows for communicating the ideal net status, annotated delays and transitions via the clock tree spec. You can update ideal nets and annotations after clock specification creation.

Use the below command to update all ideal nets, transition and delay annotations to match active timing constraints.

```
update_clock_tree_spec_annotations
```

You can also write the updated settings to a file using the `-out_file` option or apply the settings immediately.

The following new properties are provided to model the data captured by `set_ideal_network, set_annotated_delay, set_annotated_transition, set_ideal_latency` and `set_ideal_transition` commands:

- `ideal_net`: If this attribute is set, the clock tree timing engine will consider the specified net as ideal.
- `annotated_delay_to` - Overrides any clock tree timing engine computed cell arc or net arc delays to this pin, in a similar manner to SDC `set_annotated_delay`.
- `annotated_transition` - Overrides any clock tree timing engine computed transition at this pin, in a similar manner to SDC

`set_annotated_transition`.

For details about the above properties, see CCOpt Properties.

For command description and details of usage of the command and examples, see `update_clock_tree_spec_annotations` in the *Innovus Text Command Reference*.

Preserving Components in the Clock Tree

Preserving Ports

The following setting is used to preserve the module ports:

```
dbSet [dbGetHTermByInstTermName name] .dontTouch true
```

Preserving Instances

The following setting is used to preserve instances without permitting resizing:

```
dbSet [dbGetInstByName name].dontTouch true
```

The following setting is used to preserve instances while permitting resizing:

```
dbSet [dbGetInstByName name].dontTouch sizeOk
```

Note: `sizeSameHeightOk` and `sizeSameFootprintOk` are not supported and do not enable sizing.

Preserving Nets

The following setting is used to specify that the net should not be buffered:

```
dbSet [dbGetNetByName name].dontTouch true/ dbSet [dbGetHNetByName name].dontTouch true
```

The software issues the following warning suggesting that the clock tree instances that are user dont_touch and are not resizable:

```
**WARN: (IMPCCOPT-1437):   Found %d clock tree instances which are user dont_touch and are not resizable - this can
impact clock QoR. Previous versions of the software would by default ignore the dont_touch setting when deciding to
size an instance. The command 'report_preserves' can be used to report on the preservation status of all instances.
Consider using the dont_touch setting of 'sizeOk' as an alternative to full dont_touch.
```

Power Management

CCOpt-CTS and CCOpt respect power management constraints specified via CPF or UPF. Typically, on most designs it is important to permit CTS access to “always-on” buffers that have additional non-switched power. This is important so that CTS can buffer across power domains where the primary power is switched. This is performed simply by including always-on buffers and inverters with the regular buffers and inverters in the cell selection settings. For example:

```
set_ccopt_property inverter_cells {INVX4 INVX8 INVX12 PMINVX4 PMINVX8}
set_ccopt_property buffer_cells {BUFX4 BUFX8 BUFX12 PMBUFX4 PMBUFX8}
```

Note: The order of the cells in the lists is not important.

Just after `ccopt_design` is invoked, the log file will report which library cells are being used in each power domain. For example:

```
Clock tree balancer configuration for clock_tree ck:
CCOpt power management detected and enabled.
For power_domain SW and effective domain power_domain SW:
Buffers: BUFX8 BUFX4 BUFX1
```

```
Inverters: INVX8 INVX4 INVX1
For power_domain SW and effective domain power_domain AO:
Buffers: PMBUFX8 PMBUFX2
Inverters: PMINVX8 PMINVX2
For power_domain AO and effective domain power_domain SW:
Buffers: PMBUFX8 PMBUFX2
Inverters: PMINVX8 PMINVX2
For power_domain AO and effective domain power_domain AO:
Buffers: BUFX8 BUFX4 BUFX1
Inverters: INVX8 INVX4 INVX1
Unblocked area available for placement of any clock cells in power_domain SW: 178511.090um^2
Unblocked area available for placement of any clock cells in power_domain AO: 5000.000um^2
```

If CTS detects an illegal effective power domain crossing in the clock tree, it will attempt to manage the situation by temporarily overriding the effective domain of the fan-out of a violating clock tree net to match the driver's effective domain. If this happens, then in the log, there will be messages like these:

```
**ERROR: (IMPCCOPT-1044): CTS has found the clock tree is inconsistent with the power management setup: cell buf1 (a
lib_cell BUFX2) at (10.000,0.000), in power domain PDA has power_domain PDA and effective domain power_domain PDA
but drives modb/flop2/clk which has power_domain PDB and effective domain power_domain PDB.
```

To disable this behavior and have CTS abort with an error instead, set the `manage_power_management_illegalities` property to `false`. To disable all power management checks completely, set the `consider_power_management` property to `false`.

Unbufferable Regions

It is possible that the power management constraints prevent a particular net from being buffered. In this situation, the CTS QoR is likely to be severely hampered. CTS logs the following message:

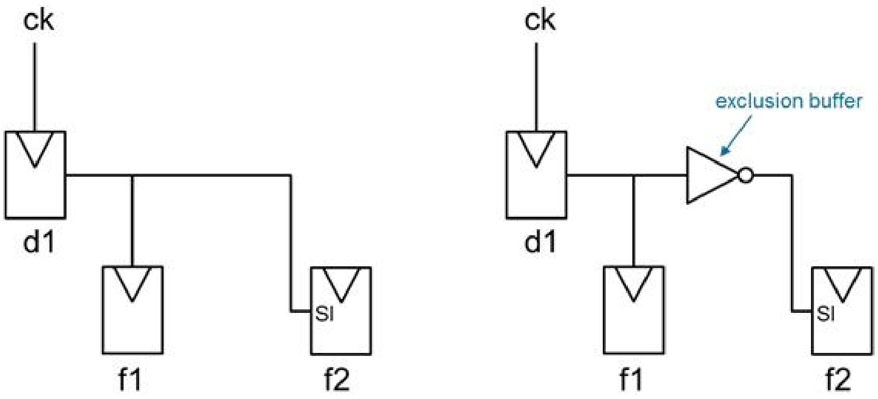
```
**WARN: (IMPCCOPT-1172): CTS cannot select a library cell to use as a driver at one or more points of clock_tree ck.
For more detailed reporting, run the report_ccopt_cell_filtering_reasons command.
```

Run `report_ccopt_cell_filtering_reasons` will help user understand the detailed reason why no library cell can be used. In order to see the details of which nets are involved, set the `detailed_cell_warnings` property true, and the software will log many IMPCCOPT-1169 and IMPCCOPT-1170 messages with detailed information.

Shared Clock and Data Concerns

In some situations, a net may be used for both clock and datapath purposes. Consider the left-hand side example below with clock divider flop d1 clocking flop f1. However, the output of d1 is additionally connected to the SI input of f2 as part of the scan chain. The net driven by d1 is part of the clock tree graph and may not be operated on by datapath optimization including datapath hold fixing. This restriction prevents datapath transforms from disrupting clock timing. After CTS it is possible that there exists a hold violation at the input of f2, but datapath hold buffer insertion will not be able to insert a buffer to fix it.

Addition of an exclusion buffer



The SI input of flop f2 will by default be a clock tree exclude pin. The command, `ccopt_add_exclusion_drivers` can be used to add exclusion drivers to isolate exclude pins from the clock tree graph. The inputs to exclusion drivers are explicitly set to be clock tree ignore pins.

As illustrated in the right-hand side example, once the exclusion buffer is added, datapath hold fixing, or other datapath optimization is free to operate on the net between the exclusion buffer and flop f2.

Inverting Clock Gates (ICG) CTS Transforms

You can enable transforms between non-inverting and inverting ICGs. This can help reduce inverter counts in designs featuring both positive-edge triggered inverting and non-inverting ICGs.

> ⚠ This is a limited-access feature in this release. It is enabled by a variable specified using 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.

This limited-access feature enables the following two steps:

1. The first step removes any inverting ICGs from the netlist before the clustering step and replaces them with an equivalent non-inverting ICG and an inverter (if possible).
2. The second step runs after the clustering step and tries to swap out non-inverting ICGs for inverting ICGs in the case where doing so would save inverters. There is no consideration of power, but this transform may reduce inverter count.

For this feature to work, you need to:

- Buffer with inverters
- Enable the limited-access feature
- Enable activity propagation
- Have both inverting and non-inverting ICGs in your cell lists

Note: The ICGs must be similar - apart from their polarity - so that the software can swap between them. The ICGs will not be swapped if they have different number of pins or different edge sensitivity.

CCOpt Property System

This section details the way the CCOpt property system operates behind the `set_ccopt_property` and `get_ccopt_property` commands. Note that some properties are read only and so many more properties are accessible to `get_ccopt_property` than `set_ccopt_property`.

Setting Properties

Properties have a name and are assigned a value. Properties can be global, or per object type. The property name, value, and if desired or required the object type and object pattern can be specified. The following syntax are acceptable:

1. `set_ccopt_property <name> <value> [-obj <obj_pattern>]`
2. `set_ccopt_property <name> [-obj  <obj_pattern>] <value>`
3. `set_ccopt_property [-obj <obj_pattern>] <name> <value>`

Where,

`name` is the name of the property to be set

`value` is the new value to which you want to set the property

The most commonly used object types are: `-skew_group`, `-clock_tree`, `-pin`, `-inst`, `-lib_pin`, `-delay_corner`, `-net_type`.

All of the commands below are equivalent:

1. `set_ccopt_property target_skew 0.1 -skew_group sg1`
2. `set_ccopt_property target_skew -skew_group sg1  0.1`
3. `set_ccopt_property -skew_group sg1 target_skew 0.1`

The property name and value parameters are positional and must appear in order. The object type and object pattern specification can appear anywhere. If an object type is not specified, then the setting will apply to all relevant objects including ones that are created in the future.

Wildcards

The object name can involve wildcard style patterns, for example:

```
set_ccopt_property use_inverters -clock_tree test* true
```

Wildcards are resolved when the command is issued not when the property value is used. In the command shown above, if a new clock tree `test_new` was defined after the `set_ccopt_property` command was issued, it would not have the `use_inverters` property set.

Optional Object Type Switches

If the `-object_type` parameter is omitted, this implies a "forall" on that type value. The example below specifies that CTS should use inverters for clock tree ct1:

```
set_ccopt_property use_inverters -clock_tree ct1 true
```

The example below does not specify the `-clock_tree` object type:

```
set_ccopt_property -use_inverters true
```

And is equivalent to:

```
foreach ct [get_ccopt_clock_trees *] {
set_ccopt_property -use_inverters -clock_tree $ct true
}
```

Delay Corner Special Handling

There are some exceptions to the above rule that are designed to capture common user intent. The main exceptions are properties for which the `-delay_corner` object type and `-early/-late` parameters apply. When these parameters are omitted, the default behavior is that the property is not set across all delay corners but instead it will be set just for the primary delay corner. In the following example, the leaf-level properties are maintained internally and one or more properties can be set at a time by specifying additional information. The example illustrates the internal representation of the `target_skew` property in a design with one delay corner used for setup timing and two delay corners used for hold timing. The initial default values are shown below:

	Early	**Late**
SetupDC	Ignore	Auto
Hold1DC	Ignore	Ignore
Hold2DC	Ignore	Ignore

If the command, `set_ccopt_property target_skew 0.05` is set, the internal representation will change to become:

	Early	**Late**
SetupDC	Ignore	0.05
Hold1DC	Ignore	Ignore
Hold2DC	Ignore	Ignore

If the command “`set_ccopt_property target_skew -delay_corner Hold1DC 0.1`” is issued, the representation will change to:

	Early	**Late**
SetupDC	Ignore	0.05
Hold1DC	0.1	0.1
Hold2DC	Ignore	Ignore

Note that the specification of “Hold1DC” matches two cells in the table. To restrict further, add `-early` or `-late`.

Getting Properties

The `get_ccopt_property` command is used to retrieve the values of properties. For example, in the last table shown above, the command “ `get_ccopt_property -target_skew -delay_corner SetupDC -late`” will return a value of `0.05`.

However, if some or all of the “`-key_name key_value`” switches are omitted, then multiple values may be returned in a list format. An example is shown below.

```
get_ccopt_property -target_skew -delay_corner SetupDC
```

returns the following:

```
{ \
  { -delay_corner SetupDC -early -value default } \
  { -delay_corner SetupDC -late -value 0.05 } \
}
```

If all the selected cells have the same value then a single value instead of a list will be returned. To force the return of a fully expanded list, even if all values are the same, use the `-list` parameter. For a summary of all parameters of the `get_ccopt_property` and `set_ccopt_property` commands, see *Innovus Text Command Reference*.

Getting a List of Properties and Descriptions

To obtain help on a property, or to find properties matching a wildcard pattern:

```
get_ccopt_property -help propertyName or pattern
```

To obtain a list of all available properties:

```
get_ccopt_property -help *
```

For example:

```
get_ccopt_property -help target_max_trans
This specifies the target skew for clock tree balancing. This may be set to a numeric value, or one of 'auto',
'ignore' or 'default'.

If set to 'auto' this indicates that an appropriate skew target should be computed.

If set to 'ignore' this indicates that skew should not be balanced for this corner/path combination.

If unspecified then the value of this property is 'default'.

If the value of the property is 'default' the target skew for late delays in the primary delay corner is interpreted
as 'auto' and as 'ignore' otherwise.

Valid values: default | auto | ignore | double
Default: default
Optional applicable arguments: "-skew_group <name>", "-delay_corner <name>", "-early" and "-late".
```

Optimizing Timing

Overview

Optimize timing after running placement followed by early global route and RC extraction, after clock tree synthesis (CTS), and after routing. The goals of timing optimization are to correct design rule violations (DRVs) and signal integrity (SI) violations and meet timing. Timing optimization includes the following operations, depending on the design stage:

- Adding buffers
- Resizing gates
- Restructuring the netlist
- Remapping logic
- Swapping pins
- Deleting buffers
- Moving instances
- Applying useful skew

Use the `setOptMode` command (or the Tools - Set Mode- Mode Setup form) to specify global timing optimization parameters. Use the `optDesign` super command (or the ECO - Optimize Design form) to optimize timing. Use the `place_opt_design` command to do placement and preCTS optimization together.

Before You Begin

Before you optimize timing for the first time, complete the following steps:

- Reserve enough additional placement space in the design in addition to the starting cell area to account for growth of the design during timing optimization and clock tree insertion.
- Specify preRoute and postRoute extraction scale factors by using the following commands:
  - `generateRCFactor`
  - `create_rc_corner`
- Use the following method to set input transitions for the high fanout nets for delay calculation:
  - Set the `delaycal_input_transition_delay` variable.
- Create and load footprints. (Optional)

You are not required to specify footprints. For more information, see the "Using the Footprintless Flow" section.

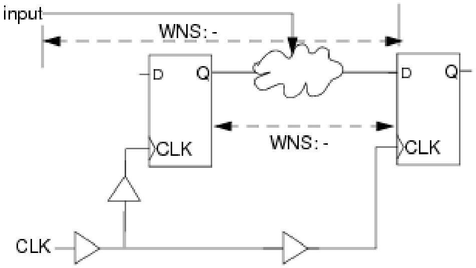
Results

After optimizing timing, the Innovus™ Implementation System (Innovus) appends the log file with the following information:

- Worst negative slack, total negative slack (TNS) and the number of failing (violating) paths. The software also reports hold violations if you specify the `-hold` parameter in postCTS or postRoute mode. It writes the values to the log file and writes reports to the working directory.

  Note: The overall TNS and number of failing paths of a design might not be equal to the total of the TNS and failing paths of the individual path groups. This is because the TNS and number of failing paths are based on the end-point of the path and are not path based.

  For example, the following figure has a register with two paths, one from a primary input with a slack of `-0.6ns` and other from another register with a slack of `-0.3ns`.



  In this case the overall TNS will be `-0.6ns` with `1` violating path (end point-based). But the individual reg2reg TNS will be `-0.3ns` with `1` violating path and input to register with a TNS of `-0.6ns` with `1` violating path. Therefore, the sum total of individual path group TNS is not the same as overall TNS.

- Number of `max_tran`, `max_cap`, and `max_fanout` violations
- Utilization (density)

If you specify path groups, the software produces a slack file and `tarpt` report for them. If you do not specify path groups, the software produces the following violation reports:

- Register-to-register
- Register-to-clock-gate
- Default: Includes all other paths, including paths to and from inputs/outputs.

The reports contain information about the following violations for the top 50 critical paths:

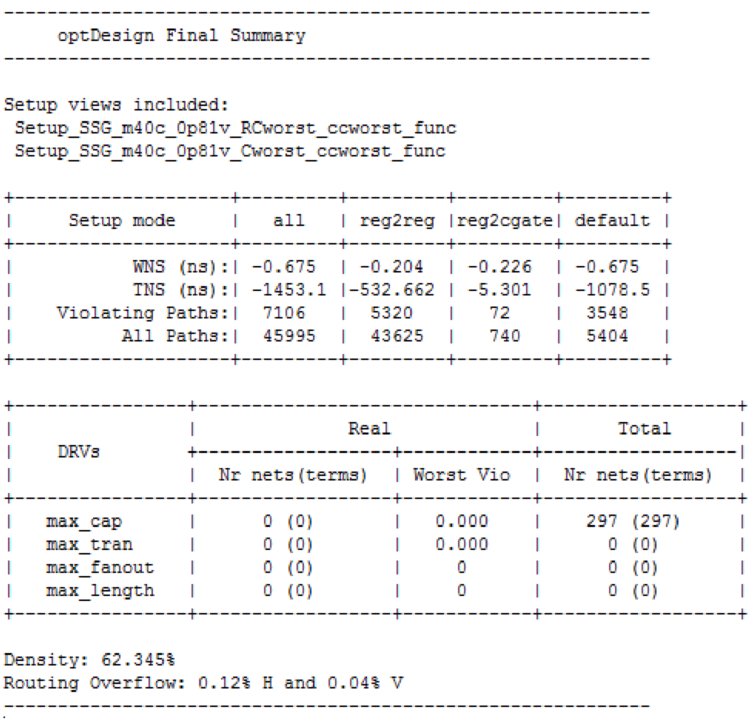
- Setup violations
- Hold violations
- DRVs (maximum capacitance, maximum transition, and maximum fanout violations)

The software generates the reports and saves them in the file specified by `optDesign -outDir` (or in the `timingReports` directory if `-outDir` is not specified).

The filenames are as follows:

- `*designName* _preCTS_ *pathGroup* .tarpt`
- `*designName* _postCTS_ *pathGroup* .tarpt`
- `*designName* _postRoute_ *pathGroup* .tarpt`

The summary report has the following format:



```
------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

Setup views included:
 Setup_SSG_m40c_0p81v_RCworst_ccworst_func
 Setup_SSG_m40c_0p81v_Cworst_ccworst_func

+--------------------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):| -0.675  | -0.204  | -0.226  | -0.675  |
|           TNS (ns):| -1453.1 |-532.662 | -5.301  | -1078.5 |
|    Violating Paths:|  7106   |  5320   |   72    |  3548   |
|          All Paths:|  45995  |  43625  |   740   |  5404   |
+--------------------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |    297 (297)     |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+

Density: 62.345%
Routing Overflow: 0.12% H and 0.04% V
------------------------------------------------------------
```

For more information on timing reports, see the `timeDesign` command.

Interrupting Timing Optimization

To stop timing optimization, use the Ctrl+C key combination. On pressing Ctrl+C, the command runs until the database is in a state where the command can stop safely. When the command stops, the software presents the Interrupt menu.

For information on the Interrupt menu, see "Interrupting the Software" in the Getting Started chapter.

Warning: When you interrupt the software with `Ctrl+C`, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.

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 `setTieHiLoMode` .

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.

Performing Optimization Before Clock Tree Synthesis

The following topics are covered in this section:

Correcting Violations in PreCTS Mode for the First Time

- Before optimizing timing in preCTS mode, you must break all timing loops by disabling arcs in the constraint file. If you do not disable the arcs, the software cannot make a valid comparison of worst negative slack (WNS) between two different runs since it might not break the loops at the same point each time. Use the following command:

  ```
  set_disable_timing
  ```

- Use the following command to optimize timing:

  ```
  optDesign -preCTS
  ```

- Use the following command to repair only DRVs:

  ```
  optDesign -preCTS -drv
  ```

Note: By default, `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 -opt_fix_fanout_load true
```

Performing Rapid Timing Optimization for Design Prototyping

To optimize timing using low-effort mode for design prototyping, use the following commands:

```
setDesignMode -flowEffort express
```

```
optDesign -preCTS
```

In low-effort mode, `optDesign` resizes gates and performs global buffer insertion and repair DRVs, but does not restructure the netlist.

Using Additional PreCTS Timing Optimization Parameters

You can use the following `optDesign` features either separately or in combination.

- To run optimization with useful skew, use the following commands:

  ```
  setOptMode -opt_skew true -opt_skew_pre_cts true
  ```

  ```
  optDesign -preCTS
  ```

- To disable area reclaiming, use the following commands (`optDesign` reclaims area by default):

  ```
  setOptMode -opt_area_recovery false
  ```

```
optDesign -preCTS
```

Performing Incremental PreCTS Optimization

Optimize timing incrementally to optimize setup times and area on critical paths. You can use the following features either separately or together:

- To run incremental setup-only optimization, use the following command:

```
optDesign -preCTS -incr
```

- To run incremental optimization with useful skew, use the following commands:

```
setOptMode -opt_skew true -opt_area_recovery true
optDesign -preCTS -incr
```

Changing Default Settings in PreCTS Mode

You can change or add parameters for the following commands that `optDesign` runs automatically:

`setAnalysisMode`	`optDesign` sets `-clkSrcPath false` and `-clockPropagation forcedIdeal` parameters by default: You cannot override these values. You can add other parameters.
`setExtractRCMode`	`optDesign` sets the extraction engine to `preRoute`. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.
`setOptMode`	`optDesign` sets the following parameters: • `-opt_drv_margin` If you use `setOptMode -opt_drv_margin`, the value you specify is added to a dynamically calculated, internal margin. For example, if you set a margin of `0.2` (20 percent), this multiplies the `max_cap` and `max_tran` SDC constraints by 0.8. The margin can be positive or negative. If you set a margin of `-0.2`, this multiplies the `max_cap` and `max_tran` SDC constraints by 1.20. `optDesign` writes the margin value to the log file. • `-opt_hold_target_slack` If you use `setOptMode -opt_hold_target_slack`, the value you specify is added to a dynamically calculated, internal margin. The `optDesign` command writes the hold target slack value to the log file. • `-opt_setup_target_slack` If you use `setOptMode -opt_setup_target_slack`, the value you specify is added to a dynamically calculated, internal margin. The default `-opt_setup_target_slack` value is `0`. The `optDesign` command writes the setup target slack value to the log file.
`setRouteMode`	You can add parameters, but you cannot override the default settings.

Using the Early Clock Flow

You can use the early clock flow in which, CCOpt creates clock trees during `place_opt_design` , based on an initial clustering, and annotates clock latencies on the clock gate pins for timing optimization. The flow also enables CCOpt ideal mode useful skew during the WNS fixing step inside `place_opt_design`.

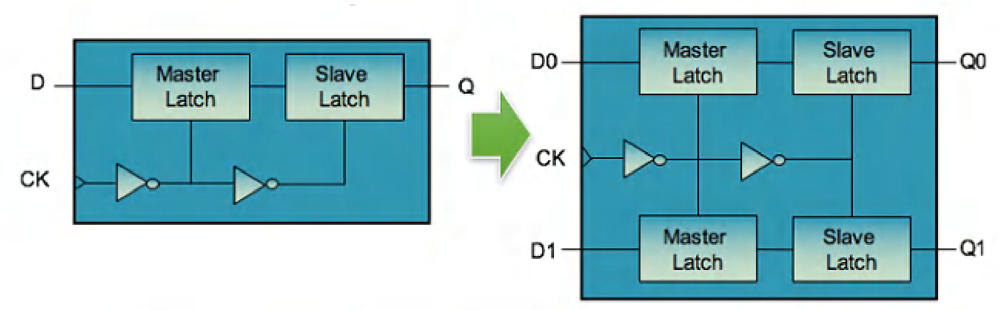
Note: Early clock mode `place_opt_design -incremental` reuses the existing early clock flow clock tree without any logical or physical modification and makes adjustments only to useful skew using virtual delays. It is also essential to note that the CCOpt clock tree spec must not be deleted between the original and incremental `place_opt_design` steps, just like it must not be done between `place_opt_design` and `ccopt_design`, otherwise any useful skew will be lost.

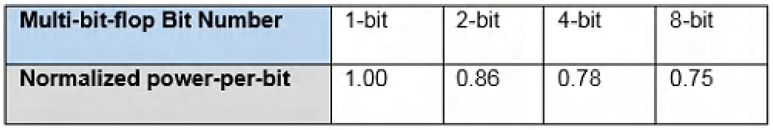
For details, see Early Clock Flow in the Clock Tree Synthesis chapter.

Using the Multi-Bit Flip-Flop Flow

The optional Multi-bit flip-flop (MBFF) flow permits clock tree dynamic power reduction by optimizing usage of multi-bit sequential cells, without degrading timing. This flow is available during preCTS optimization.

An MBFF is a single standard cell instance that has several single-bit flip-flops incorporated into the cell and typically having all outputs with the same drive strength. Merging the single-bit flip-flops is good for optimizing power because the internal clock inverters are shared, thereby, achieving power-per-bit savings. This is shown in the diagram below.





Multi-bit-flop Bit Number	1-bit	2-bit	4-bit	8-bit
Normalized power-per-bit	1.00	0.86	0.78	0.75

- There is less flexibility for timing closure. Once merged, each bit of the MBFF cannot be sized independently.
- There is less flexibility for placing individual bits that may result in longer routes.

Merging Single-bit Flip-Flops

In this flow, power reduction is achieved by merging single-bit flip-flops into MBFFs as much as possible in the preCTS optimization stage after a reasonable timing is achieved. After the single-bit flip-flops are merged once to create MBFFs, the flow then attempts to merge these MBFFs into larger MBFFs. This includes the MBFFs that are already present in the initial netlist.

As compared to the single-bit flip-flops, MBFFs are good for power, but less flexible for timing. Flip-flops on or close to the critical path are not merged. Later, while performing preCTS optimization, timing closure is further achieved through splitting the MBFFs that are still on the critical paths.

Splitting MBFFs

Splitting MBFFs can be done either with or without flip-flop merging with Innovus. Splitting MBFFs is done to recover timing. For example, if there is a netlist coming from synthesis and it has MBFFs, the software splits them if it can achieve an improvement in timing.

For the timing critical paths that include MBFFs, splitting of the MBFF will make it more flexible for timing. By using both the split and merge flows together, you can attain a timing similar to that attained when the MBFF flow is not implemented.

Consider an example to understand how the two flows are used together. If you merge two flip-flops at different locations into a new 2-bit flip-flop and put this at a new location, the routes connected to the flip-flops may become longer, therefore, increasing the delay. The preCTS optimization will try to recover the timing by using optimization methods such as buffering and sizing. However, if standard optimization is not sufficient to recover timing, then the splitting of the merged flip-flops will be done. Splitting critical-path multi-bit flip-flops may improve the timing because single-bit flip-flops are easy to resize and move.

The MBFF split transform is called several times during the course of preCTS optimization.

Use Model

To enable the MBFF flow, follow the below steps:

- Set `setOptMode -opt_multi_bit_flop_opt` to `true | mergeOnly | splitOnly`
- Set `setOptMode -opt_allow_multi_bit_on_flop_with_sdc` to `true | false | mergeOnly | splitOnly`
- Run `place_opt_design`

You can choose any of the following options to run the flow:

```
setOptMode -opt_multi_bit_flop_opt true | false|  mergeOnly | splitOnly
```

By default, the parameter is set to `splitOnly`, which means the flow is enabled for splitting MBFFs.

- Specify `true` to enable the MBFF flow
- Specify `mergeOnly` to enable only merging of MBFFs
- Specify `false` to disable the flow

By default, the `setOptMode -opt_allow_multi_bit_on_flop_with_sdc` parameter is set to `true`. This means that by default, the software will not honor the SDC constraints while merging or splitting of flops in the MBFF flow. To honor SDC constraints while merging or splitting of flip-flops, you can set this parameter to `false`. Use `mergeOnly` to ignore the SDC constraints while merging but honor for splitting. Use `splitOnly` to ignore the SDC constraints while splitting but honor for merging.

Note: The SDC constraints on D and Q pins are preserved and copied over to new flip-flops in the merging and splitting of flip-flops even when the `-opt_allow_multi_bit_on_flop_with_sdc` parameter is set to `true`. Set the parameter to `false` only if you have constraints on scan-input and scan-enable pins that you want to preserve.

Note: The MBFF flow is independent of power techniques included in the command, `setDesignMode -powerEffort`.

Specifying the Timing Effort Level for MBFF Splitting and Merging

You can specify the timing effort level for MBFF splitting and merging. Use the following parameters of the `setOptMode` command:

- `setOptMode -opt_multi_bit_flop_split_timing_effort {low | medium | high}`: This parameter specifies the timing effort level for MBFF splitting. You can specify any of the following:
  - `low` effort setting means splitting will be done only when absolutely necessary for critical paths.
  - `medium` effort setting means splitting happens for more critical paths to improve timing and remove critical timing bottlenecks .
  - `high` effort setting does aggressive splitting and try to improve timing .

  The default setting of this parameter is `low`.
- `setOptMode -opt_multi_bit_flop_merge_timing_effort {low | medium | high}`: This parameter specifies the timing effort level for MBFF merging. You can specify any of the following:
  - `low` effort setting allows degrading in timing to achieve merging.
  - `medium` effort means that merging happens either without or with minimal degradation of timing after merging.
  - `high` effort setting does not allow any degradation and does merging only if it either maintain or improves timing.

  The default setting of this parameter is `medium`.

Reordering Mixed-Drive Strength Multi-Bit Cells

There are multi-bit (library) cells available with mixed-drive strength (MDS) for bits, which means that for the same cell, different bits can have different drive strength. These bits can be reordered to achieve better timing and power optimization. You can improve either timing or power optimization or both. For this, the following parameter is provided:
`setOptMode -opt_multi_bit_flop_reorder_bits {false | true | timing | power}`

When set to:

- `false`: the software does not identify the MDS cells and no bit-reordering is done
- `timing`: software identifies and uses the MDS cells for timing optimization
- `power`: software identifies and uses the MDS cells for power optimization
- `true`: software identifies and uses the MDS cells for both timing and power optimization

By default, this parameter is set to `false`.

Reporting of Mixed-Drive Strength Multi-Bit Cells Used in Bit-Swapping

You can report MDS multi-bit cells that are used in bit-swapping based on either strong or weak driving bits. When the `setOptMode -opt_multi_bit_flop_reorder_bits` parameter is set to true, power, or timing, a column, "Bit-Strength (strong to weak)" is added in the multi-bit flip-flop reports generated using the `reportMultiBitFFs -info`, `-all`, and `-cell` parameters.

Sample reports are shown below.

```
reportMultiBitFFs -all
All Mixed Driving strength MBFF cells / Library Related Information :
---------------------------------------------------------------------------------------------------------------------
Mixed Driving Strength cells  NumBits     Dimensions ScanType ………….. LibName            Bit-Strength (strong to
weak)
MB6SRLDFQM6P5ULVT             6           9910*2400   serial        tcbn20socbwp_ecsm   {Q1,Q4}{Q2,Q6}{Q5,Q3}
MB8SRLHFDM4P2ULVT             8           8971*2400   Parallel      tsbn07socbwp_ecsm   {Q5,Q7}{Q2,Q8,Q1,Q3}{Q4,Q6}

---------------------------------------------------------------------------------------------------------------------
-

reportMultiBitFFs -cell MB6SRLDFQM6P5ULVT
The Mixed Driving strength MBFF cell / Library Related Information:
--------------------------------------------------------------------------------------------------------------------
Mixed Driving Strength cells NumBits   Dimensions  ScanType ………….. LibName            Bit-Strength (strong to weak)
MB6SRLDFQM6P5ULVT            6         9910*2400   serial          tcbn20socbwp_ecsm {Q1,Q4}{Q2,Q6}{Q5,Q3}

This MBFF cell's Usage Report :
--------------------------------------------------------------------------------------------------------------------
MBFF Cell NumBits Count Total Area Total Leakage(W) Total Dynamic (W) Min-Max Total Power Range (W)
MB6SRLDFQM6P5ULVT 6 7894 59854 - -

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

Multi-Bit Merging with Unused Bits

You can provide information about the number of unused bits in a multi-bit cell and enable the flow to control the merging with unused bits in multibit cell. To enable the flow, use the following parameter:

```
setOptMode -opt_multi_bit_unused_bits {false | true | prePlace | preCTS}
```

By default, at prePlace stage Innovus tries to merge the single-bit registers to multi-bit instances without unused bits. If during merging, the software is unable to find a suitable multibit lib_cell with a width that matches the left-over bits of a bus, it chooses the closest matching width, which leaves the minimum unused bits and, therefore, increases the multi-bit merging coverage.

When the `-opt_multi_bit_unused_bits` parameter is set to `false`, any merging that results in unused (undriven/unloaded) bits is prevented. For example, if 6 single bit cells are there to be merged and there are 4-bit and 8-bit Multibit flops cells available in the library, then the tool will pick 4-bit Multibit cell library (with two bits unmerged).

When set to `true`, the software tries to merge the single-bit register to multi-bit instances with unused bits at both, prePlace and preCTS stages. For example, if 6 single-bit cells are there to be merged and there are 4-bit and 8-bit Multibit flop cells available in the library, then the tool will pick 8-bit Multibit cell from the library (including the two unused bits).

When set to `preCTS`, the software merges the single-bit register to multi-bit instances with unused bits at the preCTS stage.

To specify the maximum number of unused bits to be allowed in each multi-bit cell, use the following parameter:

```
-opt_multi_bit_unused_bit_count
```

By default, this parameter is set to `0`. You can specify any number from `1` to `32`. To use this parameter, the `-opt_multi_bit_unused_bits` parameter must be set to `true`.

Use Model

Consider the following scenarios:

- When `setOptMode -opt_multi_bit_unused_bits` is set to `true`, and you want to set the maximum unused bits on the multibit cell to be allowed for merging. Then, you need to provide a value (equal to or greater than 1) using the `-opt_multi_bit_unused_bit_count` parameter.
  Then tool allows merging with the maximum unused bits (in the merged multi-bit cell) as the value provided using the `-opt_multi_bit_unused_bit_count` parameter.
  For example, if 6 single-bit cells are there to be merged and there are 8-bit and 16-bit Multibit flop cells available in the library , and the `-opt_multi_bit_unused_bit_count` parameter is set to 2, then the tool picks 8-bit Multibit cell from the library (with 2-bit unused).
  or,
  For example, if 5 single-bit cells are there to be merged and there are 4-bit and 8-bit Multibit flop cells available in the library, and the `-opt_multi_bit_unused_bit_count` parameter is set to 2, then the tool picks 4-bit Multibit cell from library (1-bit unmerged).
- When `setOptMode -opt_multi_bit_unused_bits` is set to `false` and you set the value of the `-opt_multi_bit_unused_bit_count` parameter, the software issues a warning:
  "`set setOptMode -opt_multi_bit_unused_bits` to true to use this option."
- When the value of `setOptMode -opt_multi_bit_unused_bit_count` parameter is greater than the maximum allowed limit, the tool does not merge the group of those single-bit cells and prints the following reason for not merging:
  ```
  Unused Bit Reason: Number of unused bits exceeded the maximum allowed limit.
  ```

Multi-Bit Combinational Cell Optimization

The multi-bit combinational flow allows merging and splitting of combinational cells. Combinational cells (muxes, inverters, nand, nor, xor, xnor) share all pins. This flow permits reduction in area and power by optimizing usage of multi-bit combinational cells, without degrading timing.

As compared to the single-bit combinational cell the multi-bit combinational cells are good for area.

The following commands are provided for this flow:

- `ecoMergeCombinationalCell`: Merges combinational cells into multi-bit cells for the specified instance. You can specify the names of the combinational instances to be merged or you can specify the file containing names of single bit instances to merge.
- `ecoSplitCombinationalCell`: Splits all multi-bit combinational cell instances to single-bit combinational cells. By default, a specific combinational cell is split only if equivalent single-bit combinational cell can be found in libraries and there is timing gain after the split

The following parameters are provided for this flow:

- `setOptMode -opt_multi_bit_combinational_opt {false | true | mergeOnly | splitOnly}`: Enables and disables the multi-bit combinational flow. This flow is called as a part of the preCTS optimization stage. By default, the parameter is set to `splitOnly`, which means the flow is enabled for splitting combinational cells.
- `setOptMode -opt_multi_bit_combinational_merge_timing_effort {low | medium | high}`: Specifies the timing effort level for multi-bit combinational merging. By default, this parameter is set to `medium`, which allows merging without/minimal degradation in timing.
- `setOptMode -opt_multi_bit_combinational_split_timing_effort {low | medium | high}`: Specifies the timing effort level for multi-bit combinational splitting. By default, it is set to low, which means the software allows splitting to very critical paths only where it is necessary.

Conditions for Merging

- Combinational cells are of same hierarchy
- Combinational cells are of same functionality
- Combinational cells have same input and output

Naming Convention During Merging/Splitting

When combinational cells are merged, the corresponding instance names are combined as follows:

Example

- `hier1/hier2/i_0 + hier1/hier2/i_1 + hier1/hier2/i_2 + hier1/hier2/i_3 --> hier1/hier2/CDN_MB_i_0_MB_i_1_MB_i_2_MB_i_3`
- `abc_x[0] + abc_x[1] + abc_x[2] + abc_x[3] --> CDN_MB_abc_x[0]_MB_x[1]_MB_x[2]_MB_x[3]`

When splitting Multibit combinational cells, it will split the names as follows:

Example

- `hier1/hier2/i_[0:3] --> hier1/hier2/i_[0:3]__0, hier1/hier2/i_[0:3]__1, hier1/hier2/i_[0:3]__2, hier1/hier2/i_[0:3]__3`
- `CDN_MB_abc_MB_def --> abc, def`

Reporting the MBFFs

Use the `reportMultiBitFFs` or the `reportMultibit` command to report all the MBFFs that have been identified while loading the design. This command is used after all the libraries are loaded. The syntax of the two commands is provided below.

```
reportMultibit
[-help]
[-outFile file_name]
[-type {comb | latch | flop | seq | iso | ls}]
[-library  | -reasonNotMerged unmerged_reason | -notMergedSummary]
```

```
reportMultiBitFFs
[-help]
[-latches]
[[[-info | -equiv | -usage] -all] | -cell cell name | -statistics | -notMergedInstList | -
reasonNotMerged unmerged_reason]
[-outFile file_name]
```

Using reportMultibit

When the `reportMultibit` command is specified with no parameters, a default summary report is printed. By default, the sequential cells are reported. Both, the multi-bit percentages merged, and a summary of why flops were not merged is printed. This is equivalent to the `reportMultiBitFFs -statistic`s, with the addition of unused bits. There are three basic types of multibit: sequential, combo, and isolation/level shifter. This command supports specifying the type of multibit to report (or all). The `-type` parameter is a global filter for all other options. Use `-type all` to print an exhaustive report.

A sample default summary report is shown below.

Use the following command to print a default summary report. By default, the sequential cells will be reported..

```
reportMultibit
```

The software returns the following information:

```
------------------------------------------------------------
Current design flip-flop statistics

Single-Bit FF Count          : 2854
Multi-Bit FF Count           : 1451
- 2-Bit FF Count             : 170
- 4-Bit FF Count             : 148
- 8-Bit FF Count             : 1133
Total Bit Count              : 12850
Total FF Count               : 4305
Bits Per Flop                : 2.985
Total Clock Pin Cap(FF)      : 3816.318
Multibit Conversion Ratio(%) : 77.79
Unused bits on Multi-Bit FF  : 3
------------------------------------------------------------
------------------------------------------------------------
Multi-bit cell usage statistics

Usabel 2-bit cell         : 8
Dont-use/touch 2-bit cell : 0
Usabel 4-bit cell         : 8
Dont-use/touch 4-bit cell : 0
Usabel 8-bit cell         : 8
Dont-use/touch 8-bit cell : 0
------------------------------------------------------------
============================================================
Sequential Multibit cells usage statistics
```

```
---------------------------------------------------------------------------
           Not Merged-bits  Merged-bits  Multibit Conversion%  Bits Per Flop
---------------------------------------------------------------------------
-FlipFlops 3814             9996         72.38                   2.62
---------------------------------------------------------------------------

------------------------------------------------------------
Seq_Mbit libcell Bitwidth                         Count
MB4SRLSDFQ_SXG2422_D1_H169_L3_P45_LLL 4           148
MB2SRLSDFQ_SXG2222_D1_H169_L3_P45_LLL 2           170
MB8SRLSDFQ_SXG4844_D1_H169_L3_P45_LLL 8           1133
------------------------------------------------------------
Total 1451
============================================================
------------------------------------------------------------

Category   Num of Insts Rejected   Reasons
------------------------------------------------------------------------
Partition  2546                  Odd number of flops in one partition
Contraint  306                   Constraints by sdc,rule or user settings
                                 (check 'Nonremovable single-bit flip-flop' in
                                 the 'Pre-merge design statistics')
Timing     2                     Exceed critical path threshold or mbff merging
                                 makes timing degradation
-------------------------------------------------------------------------
```

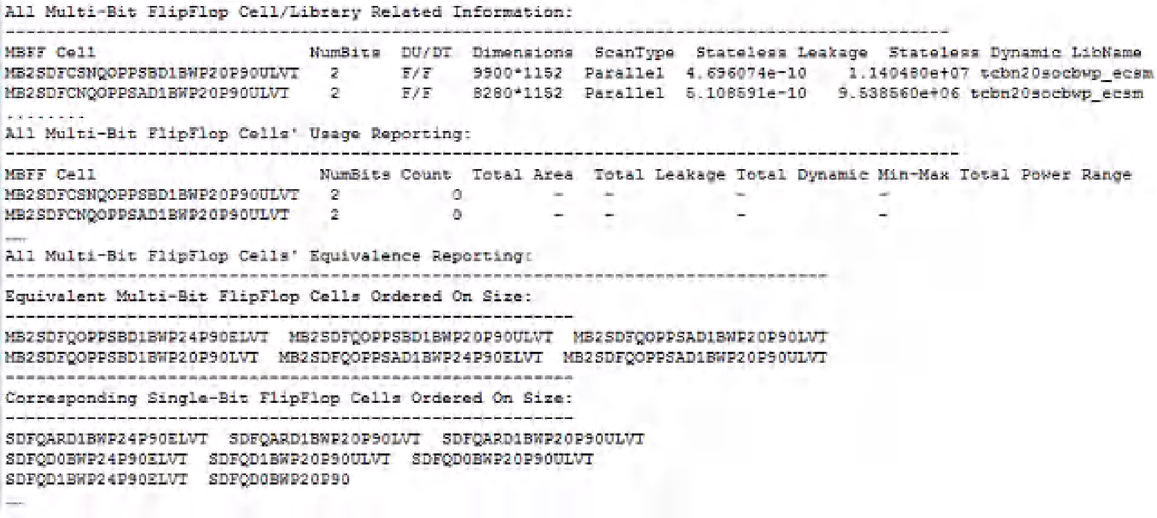
Using reportMultiBitFFs

Using this command, you can choose to report information about either all the MBFFs or one MBFF cell by specifying the name of the cell. The reported information can be printed on the console and written to a file.

Sample reports are shown below.

Use the following command to display information about all the flip-flops in the design.
`reportMultiBitFFs -all`

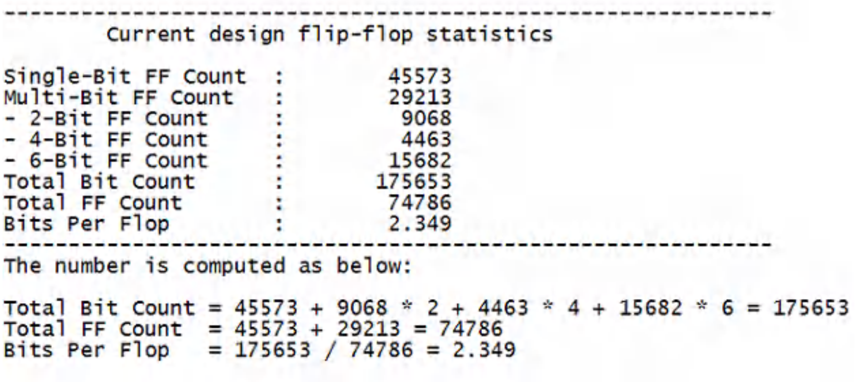
The software returns the following information:



```
All Multi-Bit FlipFlop Cell/Library Related Information:
------------------------------------------------------------------------------------------------
MBFF Cell                        NumBits  DU/DT  Dimensions  ScanType  Stateless Leakage  Stateless Dynamic LibName
MB2SDFCSNQOPPSBD1BWP20P90ULVT    2        F/F    9900*1152   Parallel  4.696074e-10     1.140480e+07 tcbn20socbwp_ecsm
MB2SDFCNQOPPSAD1BWP20P90ULVT     2        F/F    8280*1152   Parallel  5.108591e-10    9.538560e+06 tcbn20socbwp_ecsm
.........
All Multi-Bit FlipFlop Cells' Usage Reporting:
-------------------------------------------------------------------------------------------------
MBFF Cell                         NumBits Count  Total Area  Total Leakage Total Dynamic Min-Max Total Power Range
MB2SDFCSNQOPPSBD1BWP20P90ULVT    2           0         -    -            -            -
MB2SDFCNQOPPSAD1BWP20P90ULVT     2           0         -    -            -            -
....
All Multi-Bit FlipFlop Cells' Equivalence Reporting:
--------------------------------------------------------------------------------
Equivalent Multi-Bit FlipFlop Cells Ordered On Size:
------------------------------------------------------------
MB2SDFQOPPSBD1BWP24P90ELVT  MB2SDFQOPPSBD1BWP20P90ULVT  MB2SDFQOPPSAD1BWP20P90LVT
MB2SDFQOPPSBD1BWP20P90LVT  MB2SDFQOPPSAD1BWP24P90ELVT  MB2SDFQOPPSAD1BWP20P90ULVT
------------------------------------------------------------
Corresponding Single-Bit FlipFlop Cells Ordered On Size:
------------------------------------------------------------
SDFQARD1BWP24P90ELVT  SDFQARD1BWP20P90LVT  SDFQARD1BWP20P90ULVT
SDFQD0BWP24P90ELVT  SDFQD1BWP20P90ULVT  SDFQD0BWP20P90ULVT
SDFQD1BWP24P90ELVT  SDFQD0BWP20P90
....
```

Use the following command to report the single-bit and multi-bit flip-flop count in the current design.

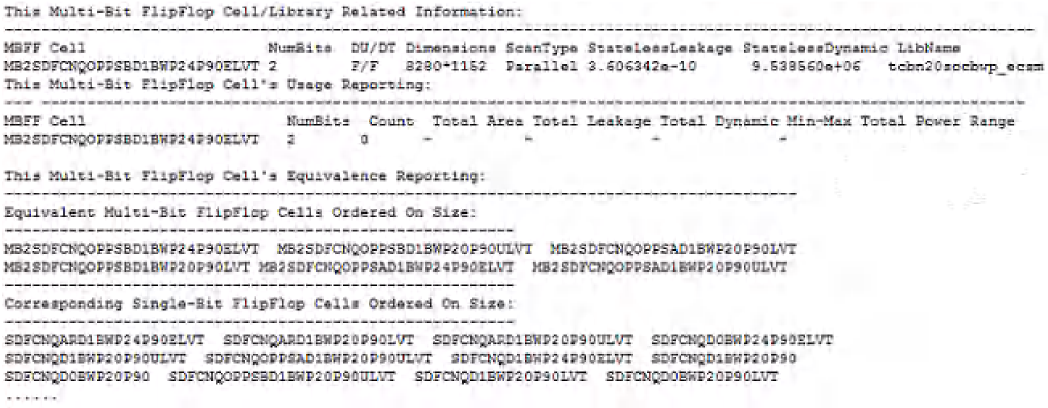
`reportMultiBitFFs -statistics` The software returns the following information:



```
------------------------------------------------------------
        Current design flip-flop statistics

Single-Bit FF Count   :        45573
Multi-Bit FF Count    :        29213
- 2-Bit FF Count      :         9068
- 4-Bit FF Count      :         4463
- 6-Bit FF Count      :        15682
Total Bit Count       :       175653
Total FF Count        :        74786
Bits Per Flop         :        2.349
------------------------------------------------------------
The number is computed as below:

Total Bit Count = 45573 + 9068 * 2 + 4463 * 4 + 15682 * 6 = 175653
Total FF Count  = 45573 + 29213 = 74786
Bits Per Flop   = 175653 / 74786 = 2.349
```

Use the following command to display information about the multi-bit flip-flop for the specified cell:

```
reportMultiBitFFs -cell MB2SDFCNQOPPSBD1BWP24P90ELVT
```



```
This Multi-Bit FlipFlop Cell/Library Related Information:
------------------------------------------------------------------------------------------------
MBFF Cell                    NumBits  DU/DT Dimensions ScanType StatelessLeakage StatelessDynamic LibName
MB2SDFCNQOPPSBD1BWP24P90ELVT 2        F/F   8280*1152  Parallel 3.606342e-10     9.538560e+06    tcbn20socbwp_ecsm
This Multi-Bit FlipFlop Cell's Usage Reporting:
--- ---------------------------------------------------------------------------------------------
MBFF Cell                      NumBits  Count  Total Area Total Leakage Total Dynamic Min-Max Total Power Range
MB2SDFCNQOPPSBD1BWP24P90ELVT   2        0      -          -             -             -

This Multi-Bit FlipFlop Cell's Equivalence Reporting:
-------------------------------------------------------------------------------
Equivalent Multi-Bit FlipFlop Cells Ordered On Size:
-------------------------------------------------------
MB2SDFCNQOPPSBD1BWP24P90ELVT  MB2SDFCNQOPPSBD1BWP20P90ULVT  MB2SDFCNQOPPSAD1BWP20P90LVT
MB2SDFCNQOPPSBD1BWP20P90LVT MB2SDFCNQOPPSAD1BWP24P90ELVT  MB2SDFCNQOPPSAD1BWP20P90ULVT
-------------------------------------------------------
Corresponding Single-Bit FlipFlop Cells Ordered On Size:
-------------------------------------------------------
SDFCNQARD1BWP24P90ELVT  SDFCNQARD1BWP20P90LVT  SDFCNQARD1BWP20P90ULVT  SDFCNQD0BWP24P90ELVT
SDFCNQD1BWP20P90ULVT  SDFCNQOPPSAD1BWP20P90ULVT  SDFCNQD1BWP24P90ELVT  SDFCNQD1BWP20P90
SDFCNQD0BWP20P90  SDFCNQOPPSBD1BWP20P90ULVT  SDFCNQD1BWP20P90LVT  SDFCNQD0BWP20P90LVT
......
```

Reporting the Multi-Bit Latches

Use the `reportMultiBitFFs -latches` parameter to report the multi-bit latches in the design similar to the reporting of MBFFs. All other parameters work with this parameter just as they work with the flip-flops.

When this parameter is specified with the below parameters, the following information about the latches is reported:

- `-all`: Reports all the multi-bit latches in the design.
- `-cell` *`cell_name`*: Specifies the name of the latch for which information is reported. You can specify both, single-bit and multi-bit latch for reporting this information.
- `-equiv`: Reports multi-bit latch equivalence. It reports all the equivalent multi-bit latch cells and corresponding single-bit latch cells in the design.
- `-info`: This parameter reports the following properties of all the multi-bit latch cells in the design:
  Name, NumBites, Don't Use/ Don't Touch, Dimensions, ScanType, Stateless Leakage, Stateless Dynamic, and LibName
- `-outFile`: Specifies the name of the output file for the latch information.
- `-statistics`: Reports the single-bit and multi-bit latch count in the current design.
- `-usage`: Reports the multi-bit latch usages. This parameter reports the following usage of all the multi-bit latch cells in the design:
  Name, NumBits, Count, Total Area, Total Leakage, Total Dynamic, and Min-Max Total Power Range

Sample report is shown below.

Use the following command to display information about all the latches in the design.

```
reportMultiBitFFs -latches -all

All Multi-Bit Latch Cell/Library Related Information:
------------------------------------------------------------------------------------------------------------------
Multi-Bit Latch Cell NumBits DU/DT Dimensions ScanType Stateless Leakage(W) Stateless Dynamic(W) LibName
MBL4SQ0P9ULVT   4    F/F     2816*2400        Serial   1.178539e-06         1.198869e-06         tsbn05sfnd_ecsm
MBL8SP1P5SLVT   8    F/F     4864*2400        None     2.075658e-06         2.693444e-06         tsbn05sfnd_ecsm
………………
------------------------------------------------------------------------------------------------------------------
All Multi-Bit Latch Cells' Usage Report:
------------------------------------------------------------------------------------------------------------
Multi-Bit Latch Cell NumBits Count Total Area Total Leakage(W) Total Dynamic(W) Min-Max Total Power Range(W)
MBL4DP0P9LLVT        4       0     0          _                _                _
MBL8SP2P9SLVT        8       0     0          _                _                _
------------------------------------------------------------------------------------------------------------

All Multi-Bit Latch Cells' Equivalence Report:
--------------------------------------------------
Multi-Bit Latch Cell ‘MBL8SSQ_33':
--------------------------------------------------
--------------------------------------------------
Equivalent Single-Bit Latch Cells Ordered On Size:
--------------------------------------------------
SBLPV2Q0P9LVT SBLFV2G2P9SVT SBLSV1DF1P9HVT
SBLDV1Q1P9ULT SBLGV3R1P9LVT SBLFV3GH2P9SVT
SBLSRRV2P9UVT SBLDV1S0P9HVT SBLHV2JK0P9ULT
……………
--------------------------------------------------
```

Dumping the MBFF Mapping Files

Use the `dumpMultiBitFlopMappingFile` command to dump the MBFF mapping files. The syntax of the command is shown below.

```
[-help]
[-mapOutputPins {false | true | all}]
[-output directory]
[-prefix fileName_prefix]
```

The command writes the following files:

- `multi_bit_pin_mapping:`
  includes multi-bit input pin mapping information by default. In addition, it appends the output pins when the `-mapOutputPins` option is specified.
- `multi_bit_mapping:`
  includes multi-bit optimization mapping information

The `multi_bit_pin_mapping` file has two columns. The first column provides the full path name of the flip-flop input D pin as it was in the initial netlist. The second column provides the full path name of flip-flop input D pin in the current state of the database that corresponds to the first column.

For example:

If SBF1, SBF2, SBF3, SBF4 are single-bit flip flops, and MBF1, and MBF2 are two-bit flip-flops, then,

The **multi_bit_mapping file** contains the following:

```
Merged Flops:

SBF1, SBF2 merged to MBF1

Split Flops:

MBF2 split to SBF3, SBF4
```

When the `-mapOutputPins` parameter is set to `false`, the software writes out only input pin mapping in the pin mapping file. For example:

```
MBF1/D1  SBF1/D
MBF1/D2  SBF2/D
```

When the `-mapOutputPins` parameter is set to `true`, the software writes out only the mapped output terms in the MBFF pin mapping file. For example:

```
MBF1/D1  SBF1/D
MBF1/D2  SBF2/D
MBF1/Q1  SBF1/Q
MBF1/Q2  SBF2/Q
```

The **multi_bit_pin_mapping** file by default contains the following**:**

```
SBF1/D    MBF1/D1

SBF2/D    MBF1/D2

MBF2/D1   SBF3/D

MBF2/D2   SBF4/D
```

Note: The `multi_bit_pin_mapping` file contains all information of the flops whose input D pin names have changed.

For example, if you have the following in the original netlist:

```
4BitMBF1         (4 bit MBF, D1-4)

SBF1             (single bit FF)
```

Then, an example of **merging** will be as follows:

If the incoming netlist has `4BitMBF1/D4` dangling (no signal attached) and MERGE, merged `SBF1/D` into `4BitMBF1/D4`. The following line will be included in `multi_bit_input_pin_mapping` file:

```
SBF1/D         4BitMBF1/D4
```

And, an example of **splitting** will be as follows:

If incoming netlist has valid signals for all 4 bits of `4BitMBF1` and splits `4BitMBF1/D4` out to `SDF2/D` leaving `4BitMBF1/D4` dangling, then the following lines will be included in `multi_bit_input_pin_mapping` file:

```
*VOID*         4BitMBF1/D4

4BitMBF1/D4    SBF2/D
```

Splitting Multi-Bit Flip-Flops

In this flow, you can split a multi-bit flip-flop (MBFF) into single-bit flip-flops for the specific instances. By default, a specific flip-flop is split only if equivalent single-bit flip-flops can be found in libraries and there is timing gain after the split. You can also specify that the split flip-flops should be placed in legal locations. By default, the positions of the split flip-flops are not guaranteed to be legal.

Use the `-all` parameter to split all MBFFs into smaller-bit flip-flops including single-bit flip-flops (SBFFs).

Use the `-inst` parameter to specify pairs of MBFFs to be split and the corresponding single-bit flops to be used. You can also specify the names of the flop instances to be split. You can specify a single instance or a tcl list of instances to be split.

Use the `-listFile` parameter to specify the file containing pairs of MBFFs to be split and the corresponding single-bit flops to be used. You can also just specify the names of MBFF instances to be split. You can specify up to 50,000 instances to split.

Note: The `-all`, `-inst` and `-listFile` parameters are mutually exclusive. If the `-all` parameter is not specified, one of these two parameters, `-inst` or `-listFile`, must be specified.

For this, the following command is provided.

```
ecoSplitFlop

[-help]
[-batch]
[-force]
[-fullSplit]
{-inst { {Mbff1 flopCell1} {Mbff2 flopCell2}..... }  | -list_file fileName | all}
[-legal]
[-power]
{-preCTS | -postCTS | -postRoute}
[-skipRoute]
```

Splitting Complex Flip-Flops

In this flow, a complex flip-flop is split into a simple flip-flop and combinational logic for the specified instance. A complex flip-flop means the D side has some combinational logic, such as AND, OR, MUX, AOI, and so on. By default, a specific flip-flop is split only if equivalent simple flip-flops and combinational logic can be found in libraries and there is timing gain or power gain after the split.

For this, the following command is provided.

```
ecoSplitComplexFlop
[-help]
[-force]
{-inst {inst1, inst2,...} | -listFile instNamesFile | -all}
[-legal]
[-preCTS | -postCTS | -postRoute]
[-power]
[-skipRoute]
```

You can also specify that the split flip-flops should be placed in legal locations. By default, the positions of the split flip-flops are not guaranteed to be legal.

Performing PostCTS Optimization

The following topics are covered in this section:

- Correcting Violations in PostCTS Mode
- Performing Incremental PostCTS Optimization
- Changing Default Settings in PostCTS Mode

Correcting Violations in PostCTS Mode

- To optimize timing after the clock tree is built, use the following commands:
  `optDesign -postCTS`
  `optDesign` in post-CTS fixes DRVs, reclaims area, and fixes setup and hold violations.
  Note: If using CCOpt, instead of CCOpt-CTS, then additional post-CTS setup optimization is not normally required as CCOpt combines CTS and postCTS style datapath optimization. Refer to Clock Tree Synthesis for further details of CCOpt.
  By default, `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 -opt_fix_fanout_load true`
- To repair setup and hold violations, use the following commands:
  `optDesign -postCTS`
  `optDesign -postCTS -hold`
- To repair only DRVs, use the following command:
  `optDesign -postCTS -drv`
- To repair only hold violations, use the following command:
  `optDesign -postCTS -hold`

Skipping Path Groups During Hold Fixing

You can instruct the software to exclude path groups from hold fixing by using the `-opt_hold_ignore_path_groups` parameter of the `setOptMode` command.

Reporting Violations Remaining After Hold Fixing

You can generate report files that help diagnose the remaining hold violations after hold fixing by using the following commands:
`optDesign -postCTS -hold -holdVioData` *fileName*

`optDesign -postRoute -hold -holdVioData` *`fileName`*

A sample report is provided below.

```
// Path 1
Endpoint:
some/name/D
Beginpoint
:
some/name/Q
Analysis View:
hold_func
Hold Slack:
-
1615.8ps
Node:
some/name/pin
Setup Slack:
-
167.2ps
Analysis View:
```

```
setup_func
Failed Reason:
 LegalSingleBuffer
  SetupTNSDegradation(65), LocalSetupWNSDegradation(7),
 ResizeLegal
  NoCellSelected (1),  HoldSlackDegraded (4),  LocalSetupWNSDegradation (5),
 ECO Safe Resize
  TooBigToFixByResize
```

A brief description of some of the important reasons that may appear in the "Failed Reason" section of the report are as follows:

"`ResizeLegal`" + "`LegalSingleBuffer`" = "`NoLegalLoc`" means could not find the legal location for resizing to bigger cell or inserting new buffer/delay cell
"`LocalSetupWNSDegradation`" means that this move leads to degrading worst setup slack on connected input/output nets
"`NetDrvFailed`" means that this move leads to DRV (Tran/Cap) degradation
"`SetupTNSDegradation`" means that this move leads to setup TNS degradation on connected input/output nets
"`NoCellSelected`" means that no cell is selected for resizing due to don’t touch or other constraint
"`HoldSlackDegraded`" means that this move causes hold slack degradation instead of improvement
"`TooBigToFixByResize`" means resizing is not allowed as the hold violations are too big to be fixed by resizing alone

Note: `LegalSingleBuffer`, `Resizelegal` and `ECO Safe Resize` are the names of the transforms.

Using Additional PostCTS Timing Optimization Parameters

You can focus timing optimization on specific paths using path groups. Running the `createBasicPathGroups` command creates `reg2reg` and `reg2cgate` (if present in the design) path groups and sets them to high effort. It also creates the `in2reg`, `reg2out`, and `in2out` path groups when the `-expanded` option is used and sets these additional groups to low effort. To optimize these path groups, run the following commands:

```
createBasicPathGroups
optDesign -postCTS
```

If path groups are not defined, the `optDesign` command will temporarily generate two high effort `path_groups` (reg2reg and reg2cgate).

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

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

```
    group_path -name path_group_name -from from_list -to to_list -through through_list

    setPathGroupOptions ...

    optDesign -preCTS -incr
```

Creating path groups is not mandatory to achieve the best results because of the following reasons:

- Too many custom path groups may impact the runtime significantly.

- Too many overlapping or nested path groups may impact TNS timing closure.

Performing Incremental PostCTS Optimization

- To optimize setup time incrementally and reduce area, use the following commands:

```
setOptMode -opt_area_recovery true
optDesign -postCTS -incr
```

- To take advantage of useful skew when optimizing timing in incremental postCTS mode, use the following commands:

```
setOptMode -opt_skew true -opt_skew_ccopt true
optDesign -postCTS -incr
```

- To run incremental postCTS optimization if your design has a clock mesh, use the following commands:

```
setOptMode -opt_skew false
optDesign -postCTS -incr
```

Changing Default Settings in PostCTS Mode

You can change or add parameters for the following commands and parameters that `optDesign` runs automatically:

`setAnalysisMode`	`optDesign` sets `-clockPropagation` to `autoDetectClockTree` and `-clkSrcPath` to `true` by default: You cannot override these values. You can add other parameters.
`setExtractRCMode`	`optDesign` sets the extraction engine to `preRoute`. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.

Performing PostRoute Optimization

The following topics are covered in this section:

- About PostRoute Optimization
- Using the route_opt_design Flow
- Correcting Violations and 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

About PostRoute Optimization

In postRoute mode, the software corrects setup violations and design rule violations unless you specify otherwise. It first operates on register-to-register paths (and register-to-clocks path groups, if present), and then on the default path group. The software performs RC extraction and delay calculation, and runs the NanoRoute router to perform ECO routing. In case the final timing after ECO routing is degraded as compared to what was expected before ECO routing, the software automatically calls an incremental optimization to recover the setup timing and/or DRV.

If filler cell information has been defined, `optDesign` removes or adds them as needed, following the information given by the `setFillerMode` command.

If only non-SI Timing is being looked at, there should be very few timing violations that need correction. The primary sources of these violations include the following:

- Inaccurate prediction of the routing topology during preRoute optimization due to congestion-based detour routing
- Minor correlation issues between preRoute and postRoute RC extraction.

> ⓘ Because the violations at this stage are due to inaccurate modeling of the final route topology and the attendant parasitics, it is critical not to introduce additional topology changes beyond those needed to correct the existing violations.

Making unnecessary changes to the routing at this point can lead to a scenario where fixing one violation leads to the creation of other violations. This cascading effect creates a situation where it becomes impossible to close on a final timing solution with no DRVs. One of the strengths of postRoute optimization is its ability to simultaneously cut a wire and insert buffers, create the new RC graph at the corresponding point, and modify the graph to estimate the new parasitics for the cut wire without re-running extraction.

To take even more advantage of this feature, you can provide an external SPEF file generated by a sign-off extraction tool for improved convergence. However,you must provide a full SPEF (reduced SPEF does not work) and one of the following conditions must be met:

- The SPEF file must be generated with node locations using the `starN` format.
  or
- The resistance values in the LEF file must match those in the technology file used by signoff extraction to generate the SPEF file, which enables the extraction engine to match the routes with the SPEF RC graph.

Using the route_opt_design Flow

The route_opt_design command is provided to combine the routing and postRoute optimization flows into a single flow. The benefit of this approach is to interleave the optimization step with routing step. When this command is run, the software performs global and detail routing, setup/hold timing optimization, and DRV fixing. The flow aims to improve the runtime with similar or better QoR by optimizing timing before detail routing.

> ⚠ This command is part of a limited-access feature in this release. It is enabled by a variable specified using 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.

The route_opt_design command can be used instead of the following commands:

- routeDesign
- optDesign -postRoute -setup -hold

The route_opt_design flow is different from routeDesign followed by optDesign -postRoute -setup -hold in the following ways:

- The route_opt_design command performs more optimization during the routing stage
- Extraction must be performed using the TQuantus extraction engine for performing interleaved optimization with routing. If TQuantus extraction fails during routing, the route_opt_design command executes two separated commands
  - routeDesign followed by optDesign -postRoute -setup -hold.

The route_opt_design command has parameters to split the routing part (-route) and the optimization part (-opt). When these parameters are specified, the flow exits the interleaved mode, and goes back to the traditional flow with separate commands. This means that route_opt_design -route is equivalent to routeDesign, and route_opt_design -opt is equivalent to optDesign -postRoute -setup -hold. By default, the optimization of route_opt_design fixes both setup and hold timing in the interleaved mode with routing. To fix setup timing only, use route_opt_design -setup. When the -setup parameter is specified, the software performs optimization, and skips hold timing fixing.

Correcting Violations and Signal Integrity Issues using GigaOpt Technology in PostRoute Mode

GigaOpt technology is default for the postRoute flow, including setup/hold/power optimization. GigaOpt simplifies the postRoute closure flow. It is recommended that onChipVariation should be turned on and CPPR be used as shown below:

```
setAnalysisMode  -analysisType onChipVariation -cppr both
```

GigaOpt in PostRoute Setup Timing Flow

GigaOpt technology has two phases:

- Design-rule violations fixing and SI slew and glitch fixing
- Setup timing fixing on base and SI delay

GigaOpt does multi-threading combined base and SI delay timing optimization. It supports the following:

- Power-driven optimization
- External SPEF-flow (with node location)
- Filler cells deletion/insertion

- Smart ECO routing
- ILM/GigaFlex flow and MSV flow

GigaOpt in PostRoute Hold Timing Flow

In setup aware hold fixing, hold violations are fixed while having the full setup timing graph in memory.

GigaOpt supports the following:

- External SPEF-flow
- Filler cells deletion/insertion
- Smart ECO routing

GigaOpt hold fixing generates detailed diagnostic report for all remaining hold-violated nets. GigaOpt hold fixing performs the following:

- Buffer insertion along the route
- Wire and RC cutting
- Legal location searching
- Cell resizing
- Distributed hold analysis

GigaOpt in PostRoute Use Model

To optimize timing setup and hold with base and SI delay, use the following commands:

```
optDesign -postRoute
optDesign -postRoute -hold
```

To optimize timing setup and hold with base delay only, use the following commands:

```
setDelayCalMode -SIAware false
optDesign -postRoute
optDesign -postRoute -hold
```

Note: For run-time reduction, you can also perform combined setup and hold optimization. Use the following command instead of running both `optDesign  -postRoute` and `optDesign -postRoute -hold`:

```
optDesign -postRoute -setup -hold
```

Examples

- To run GigaOpt for performing setup timing TNS optimization on base and SI delay, use the following commands ensuring that the default for `setDesignMode -flowEffort` is being used:

  ```
  setDesignMode -flowEffort standard
  optDesign -postRoute
  ```

- To run GigaOpt for performing setup and leakage power optimization on base and SI delay, use the following commands:

  ```
  setOptMode -opt_power_effort high -opt_leakage_to_dynamic_ratio 1.0
  optDesign -postRoute
  ```

- To run GigaOpt for performing setup with SI slews optimization on base and SI delay, use the following commands:

  ```
  setOptMode -opt_post_route_fix_si_transitions true
  ```

```
optDesign -postRoute
```

Note: By default, the `optDesign` command does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

```
setOptMode -opt_fix_fanout_load true
```

Note: Hold repair does not degrade the setup worst slack to less than the original value or the setup target slack value. You can override the setup target slack value value by specifying `setOptMode -opt_setup_target_slack` 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 -opt_hold_allow_setup_tns_degradation false
```

You can instruct the software to exclude path groups from hold fixing. For more information, see "Skipping Path Groups During Hold Fixing".

- To take clock reconvergence pessimism removal (CRPR) into consideration when running timing optimization, use the `setAnalysisMode` command before you run `optDesign`. For example:

```
set_timing_derate -max -clock -early 0.8 -late 1.2
set_timing_derate -min -clock -early 0.8 -late 1.2
setAnalysisMode -cppr both
optDesign -postRoute
```

- To run postRoute timing optimization on designs containing Interface Logic Models(ILMs), use the following command:

```
optDesign -postRoute
```

This will automatically flatten ILMs, optimize timing, and then unflatten the ILMs.

- To run postRoute setup or hold optimization based on external SPEF files (with node locations using the `starN` format) for a design with four active RC corners (two for setup and two for hold), run the following:

```
spefIn -rc_corner cornerMax1 rcMax1.spef
spefIn -rc_corner cornerMax2 rcMax2.spef
spefIn -rc_corner cornerMin1 rcMin1.spef
spefIn -rc_corner cornerMin2 rcMin2.spef
optDesign -postRoute -hold
```

Note: You must provide SPEF information for each active `rc_corner` (setup and hold). If one corner does not have a SPEF, the software will rerun RC extraction for every corner.

Changing Default Settings in PostRoute Mode

You can change or add parameters for the following commands that `optDesign` runs automatically:

`setAnalysisMode`	`optDesign` sets `-clockPropagation` to `autoDetectClockTree` and `-clkSrcPath` to `true` by default: You cannot override these values.
`setExtractRCMode`	`optDesign` sets the extraction mode to `postRoute -effortLevel medium` (TQuantus). If no Quantus techfile is available then `-effortLevel low` will be used. Note, you can always change this to `-effortLevel high` (IQuantus) or `-effortLevel signoff` (Standalone Quantus) but both require a Quantus license. Ensure that you set the appropriate extraction scale factors if using the low/medium/high effort levels.

Performing Target-Based PostRoute Optimization

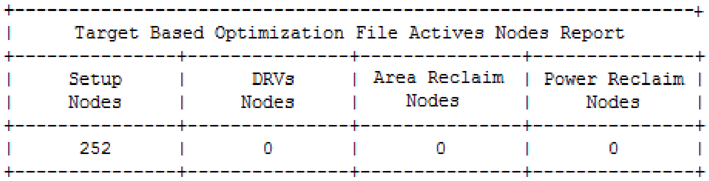
The following topics are covered in this section:

About Target-Based PostRoute Optimization

The purpose of target-based Optimization (TBOpt) is to focus on critical nodes in the design at the postRoute stage and optimize these by giving worst negative slack (WNS), area, power, max transition, and signal integrity (SI) glitch improvements with a minimum disturbance to NanoRoute.

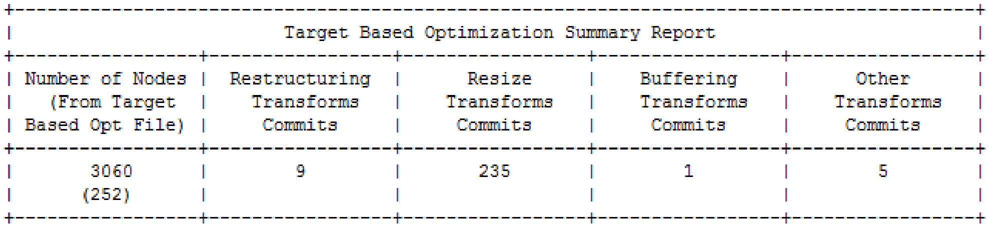
There are currently two main use models, one where the software can automatically select timing critical nodes and the other where you provide the node data in a file and choose the optimization you want to perform on these nodes. Note that with the first use model, the software automatically selects bottlenecks only for timing and not for area, power, max transition, and SI glitch. Also note that this feature is not designed as a replacement for `optDesign -postRoute` but as an enhancement that can be called afterwards, if required. It does not work on the design's total negative slack (TNS). It will output useful tables to the log detailing the nodes it is working on, whether they are from the target-based optimization file or not and the transforms used to fix them in the case of timing optimization.

When a TBOpt file is specified, a table similar to the one shown below will be generated by default.



```
+---------------------------------------------------------------+
|      Target Based Optimization File Actives Nodes Report
+---------------+---------------+---------------+---------------+
|    Setup      |      DRVs     | Area Reclaim  | Power Reclaim |
|    Nodes      |     Nodes     |     Nodes     |     Nodes     |
+---------------+---------------+---------------+---------------+
|     252       |       0       |       0       |       0       |
+---------------+---------------+---------------+---------------+
```

In addition, a summary table similar to the one shown below will be generated at the end of timing optimization.



```
+----------------------------------------------------------------------------------------+
|                             Target Based Optimization Summary Report                   |
+-----------------+-----------------+-----------------+-----------------+-----------------+
| Number of Nodes | Restructuring   |     Resize      |    Buffering    |      Other      |
|   (From Target  |   Transforms    |   Transforms    |   Transforms    |   Transforms    |
| Based Opt File) |     Commits     |     Commits     |     Commits     |     Commits     |
+-----------------+-----------------+-----------------+-----------------+-----------------+
|       3060      |        9        |       235       |        1        |        5        |
|      (252)      |                 |                 |                 |                 |
+-----------------+-----------------+-----------------+-----------------+-----------------+
```

Automatic Selection

The most straight forward use model is to allow the software to pick the most critical nodes itself and optimize those for timing. To do this, use the following command:

```
optDesign -postRoute -targeted
```

Using a Target File

To select nodes by referencing instances, nets, and paths to pass to the software to optimize directly, use the following set of commands:

```
setOptMode -opt_target_based_opt_file tb_opt.txt
```

To optimize only these nodes, use the following commands:

```
optDesign -postRoute -targeted
```

The format of the TBOpt file, `tb_opt.txt` is as follows with the syntax required in **bold**. You can use any of the below syntax individually or collectively. They are all independent of each other and each contributes its own nodes for TBOpt to work on:

```
NETS : {net1} {net2} {netN}
FROMTOS : {fromTerm1 toTerm1} {fromTerm2 toTerm2} {fromTermN toTermN}
INSTS : {inst1} {inst2} {instN}
PATHS : {endpoint1} {endpoint2} {endpointN}
```

Example 1

```
NETS : {mod_i/mod_sub_i/reset_l_sync_3} {mod_i/mod_sub_i/FE_OCPN210_reset_l_sync_3}
INSTS : {mod_i/mod_sub_i/reset_l_sync_reg_15} {mod_i/mod_sub_i/FE_OCPC210_reset_l_sync_16}
PATHS : {mod_i/mod_sub_i/ser_en/data_d1_reg_32/D}
```

Example 2

```
INSTS : {mod_i/mod_sub_i/reg_20}
FROMTOS : {mod_i/mod_sub_i/u1/z mod_i/mod_sub_i/u460/D}
```

Using a Target File to Perform End Point Adjustment

You can set individual target slacks on different end point nodes, as required. To perform end point adjustment, use the syntax provided below inside the `tb_opt.txt` file. A positive value of say, `200` means the slack will be `200ps` worse at that node and optimization will work to over-fix it by `200ps`. Note that the syntax in **bold** must be adhered to:

```
END_POINT_ADJ : {endpoint1} {slack_adj:value in ps}
```

Or with a list of end points:

```
END_POINT_ADJ : {endpoint1} {endpoint2} ... {slack_adj:value in ps}
```

Note that for this to work, either of these must be combined with the usual node specification. For example, the following adjusts end point `u460/D` and works on nodes from `u1/z` to `u460/D`:

```
END_POINT_ADJ : {u460/D} {slack_adj:200.0}
FROMTOS: {u1/z u460/D}
```

The following example adjusts end point `u3/D` and works on `n1`, `n2,` and `n3` nets:

```
END_POINT_ADJ : {u3/D} {slack_adj:300.0}
NETS : {n1} {n2} {n3}
```

The following example adjusts end point `u3/D` and will work on all paths to `u3/D`:

```
END_POINT_ADJ : {u3/D} {slack_adj:300.0}
PATHS : {u3/D}
```

If there are multiple adjustments set on the same endpoint using the same format then the final one will take precedence.

In addition, all tokens (`NETS, INSTS, FROMTOS` and so on) in the TBOpt file are cumulative except `slack adjust`, which will not add any node into the list of nodes to work on. This means that each time a token is read, TBOpt will add it to the list of nodes to evaluate. So to summarize:

`NETS : {net1} ->` Add `net1` into list of nodes to work on.
`INSTS : {i1} ->` Add `i1` into list of nodes.
`PATHS : {p1} ->` Add all nodes belonging to this path into nodes to work on.
`FROMTOS : {f1 t1} ->` Add all nodes from `f1` to `t1` into the nodes to work on.
`END_POINT_ADJ : ->` Adjust slack on selected end point, but do not add any node into nodes to work on.

You can also perform view-specific end point adjustment. Using the special view syntax in **bold**, you can adjust the end point for the view, "`setupView1`" in the following example:

```
END_POINT_ADJ : {U6/D} {slack_adj:200.0 view:setupView1}
NETS : {n1} {n2} {n3}
```

Alternatively, if the name of the view is very long, you can use a `VIEW_ID` token as shown in the example below:

```
VIEW_ID : {Setup_TTF_m40c_0p81v_RCworst_ccworst_func_view2 1}
END_POINT_ADJ : {U10/D} {slack_adj:200.0 view: 1}
PATHS : {U10/D}
```

Note: You can use wildcards for specifying END POINT adjustments, NETS, PATHS, INSTS, and FROMTOS in the target file. For example,

```
END_POINT_ADJ : {U*/D} {slack_adj:200.0 view:setupView1}
```

Using a Target File to do Area, Power, Max Transition, and SI Glitch Targeting

To target a certain module of a chip to do area optimization, use the following syntax inside the `tb_opt.txt` file:
`AREA_RECLAIM : {`*`ModuleName`*`} {x1 y1 x2 y2}`

To target a certain module of a chip to do power optimization, use the following syntax inside the `tb_opt.txt` file:
`PWR_RECLAIM : {`*`ModuleName`*`} {x1 y1 x2 y2}`

To target certain nets to do max transition optimization, use the following syntax inside the `tb_opt.txt` file. The value specified will be in `ps`:
`DRVS : {netName1} {netName2} {max_tran: e.g. 100}`

It is possible to do DRV fixing on a net without any `max_tran`, which means fixing any existing max transition on the net:
`DRVS : {netName1}`

To target certain nets to do SI Glitch optimization, use the following syntax inside the `tb_opt.txt` file. The value specified will be a percentage of VDD. Therefore, it is best to specify a low target to ensure that the net is fixed. Note the syntax in **bold** must be adhered to:

`SI_GLITCH: {` *netName* `or` *netNameList* `} {` **value** `: e.g. 0.1}`

Using the createTBOptFile Command to Generate the Target File

The `createTBOptFile` command creates a target-based optimization (TBOpt) file for use with postRoute targeted optimization. Once created, this file is passed to the `setOptMode -opt_target_based_opt_file` command. You can either provide the command a timing report in a machine-readable format by using the `-gtdFile` parameter or let the command generate the timing report internally by using the values specified for the `-min_slack`, -`max_slack,` and `-max_paths` parameters or the internal default values for these parameters.

Based on this timing report, a bottleneck analysis is performed and the critical instances are stored in a file that is specified using the `-outFile` *`filename`* parameter. This file uses the TBOpt format with the INSTS token. The bottleneck analysis computes the number of times an instance appears (on only the failing paths by default) and creates "n" number of path categories for the instances with the highest number of occurrences. Note that "n" is the number you specify using the -`bottleneckInstanceNumber` parameter of this command.

Also note that by default, multiple instances per path are allowed. This means that if there are multiple critical instances on a path, all of them will be returned, therefore, leading to the generation of the most accurate list of critical instances.

A few examples are provided below:

- The following command generates a TBOpt file with five critical instances using the default values for `-min_slack`, `-max_slack`, and `-max_paths` that will be written to the file, `TBOpt_bottleneck_5.txt`:

  ```
  createTBOptFile -bottleneckInstanceNumber 5 -outFile TBOpt_bottleneck_5.txt
  ```

- The following command generates a TBOpt file, `usergenerated.mtarpt` with three critical instances based on a machine-readable timing report generated using the `-machine_readable` parameter of the `report_timing` command:

  ```
  createTBOptFile -bottleneckInstanceNumber 3 -outFile TBOpt_bottleneck_3.txt -gtdFile usergenerated.mtarpt
  ```

- The following command generates a TBOpt file with six critical instances based on only the `reg2reg` path group with the maximum number of worst paths as 3000:

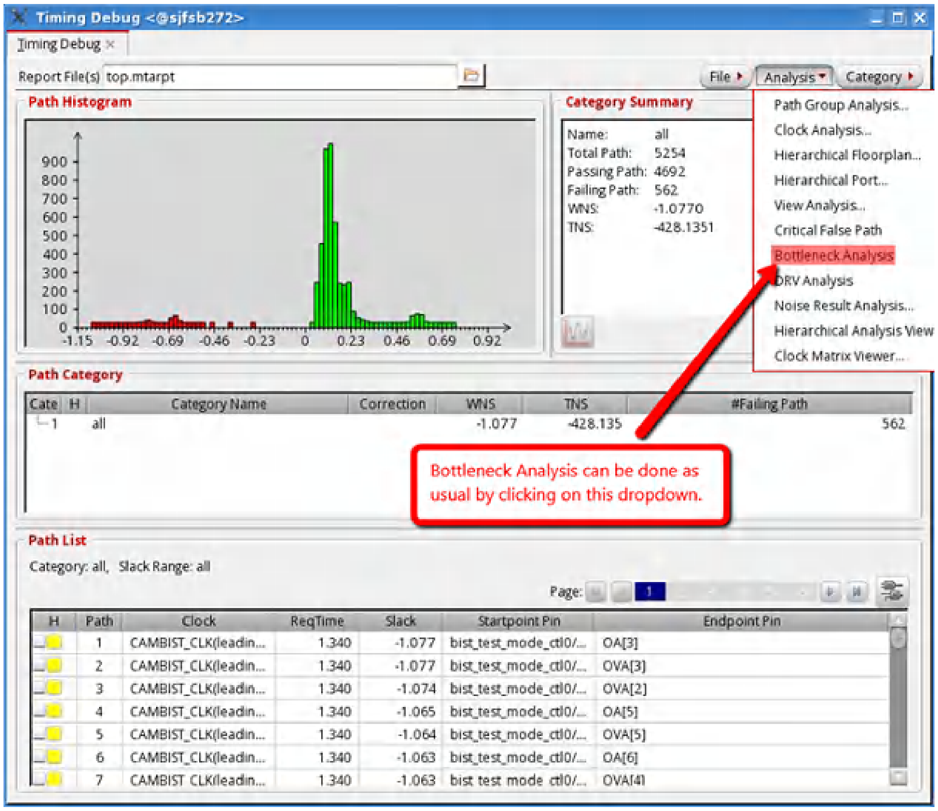
  ```
  createTBOptFile -bottleneckInstanceNumber 6 -outFile TBOpt_bottleneck_6.txt -path_group reg2reg -max_paths 3000
  ```

- The following command generates a TBOpt file with nine critical instances based on a timing analysis of the worst 10,000 paths between `+0.025ns` and `-1.0ns`:

  ```
  createTBOptFile -bottleneckInstanceNumber 9 -outFile TBOpt_bottleneck_9.txt -min_slack -1.0 -max_slack +0.025 -
  max_paths 10000
  ```

Using Timing Debug to Generate the Target File

Below is a step by step guide to using the Timing Debug Bottleneck Analysis to look at instances that occur frequently and are common to many critical paths. Use this feature to ‘Save Paths()’ to create a file that can then be directly applied to the `setOptMode -opt_target_based_opt_file tb_opt.txt` file.



Report File(s) top.mtarpt

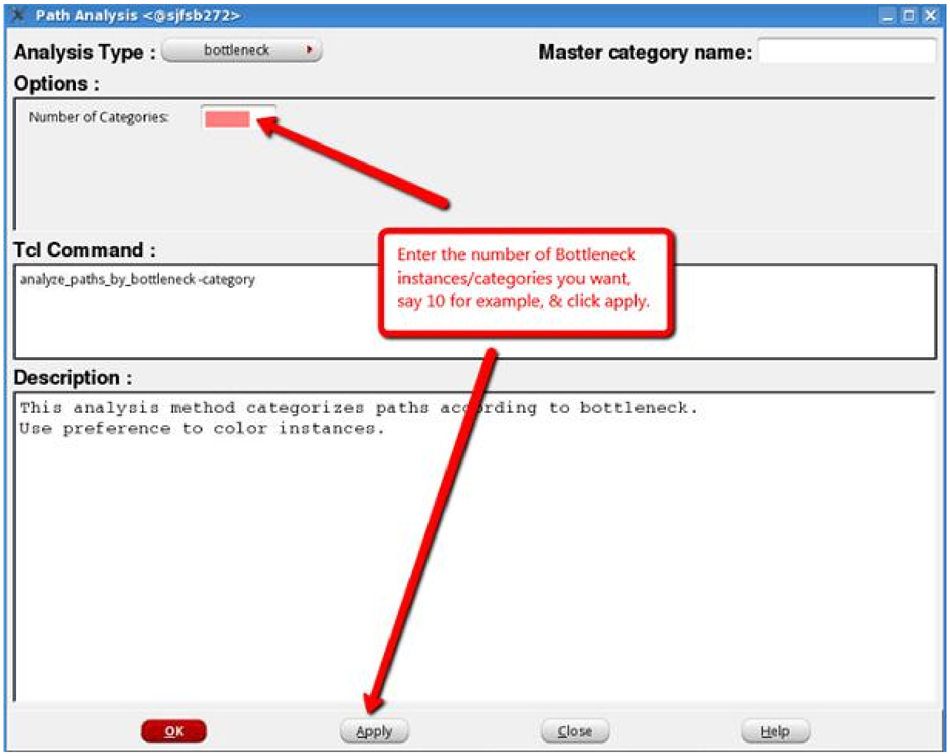
Name: all
Total Path: 5254
Passing Path: 4692
Failing Path: 562
WNS: -1.0770
TNS: -428.1351

Cate	H	Category Name	Correction	WNS	TNS	#Failing Path
1		all		-1.077	-428.135	562

Bottleneck Analysis can be done as usual by clicking on this dropdown.

Category: all, Slack Range: all

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
	1	CAMBIST_CLK(leadin...	1.340	-1.077	bist_test_mode_ctl0/...	OA[3]
	2	CAMBIST_CLK(leadin...	1.340	-1.077	bist_test_mode_ctl0/...	OVA[3]
	3	CAMBIST_CLK(leadin...	1.340	-1.074	bist_test_mode_ctl0/...	OVA[2]
	4	CAMBIST_CLK(leadin...	1.340	-1.065	bist_test_mode_ctl0/...	OA[5]
	5	CAMBIST_CLK(leadin...	1.340	-1.064	bist_test_mode_ctl0/...	OVA[5]
	6	CAMBIST_CLK(leadin...	1.340	-1.063	bist_test_mode_ctl0/...	OA[6]
	7	CAMBIST_CLK(leadin...	1.340	-1.063	bist_test_mode_ctl0/...	OVA[4]



Analysis Type : bottleneck

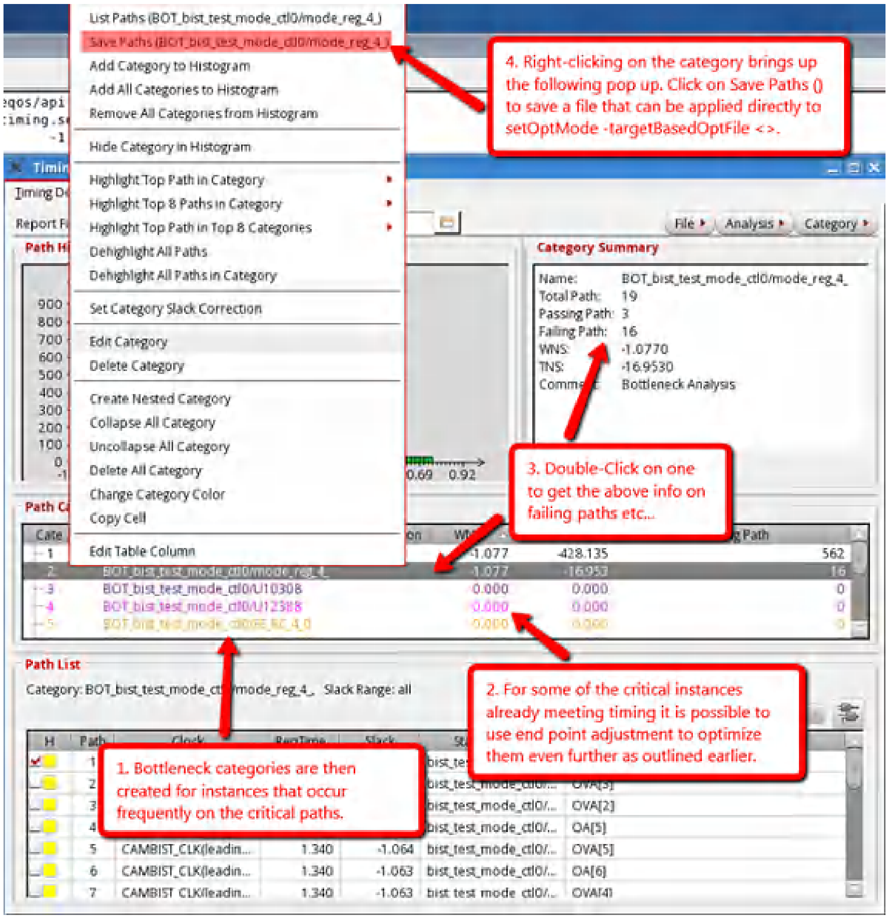
Master category name:

Number of Categories:

Enter the number of Bottleneck instances/categories you want, say 10 for example, & click apply.

```
analyze_paths_by_bottleneck -category
```

```
This analysis method categorizes paths according to bottleneck.
Use preference to color instances.
```


List Paths (BOT_bist_test_mode_ctl0/mode_reg_4_)
Save Paths (BOT_bist_test_mode_ctl0/mode_reg_4_)
Add Category to Histogram
Add All Categories to Histogram
Remove All Categories from Histogram
Hide Category in Histogram
Highlight Top Path in Category
Highlight Top 8 Paths in Category
Highlight Top Path in Top 8 Categories
Dehighlight All Paths
Dehighlight All Paths in Category
Set Category Slack Correction
Edit Category
Delete Category
Create Nested Category
Collapse All Category
Uncollapse All Category
Delete All Category
Change Category Color
Copy Cell
Edit Table Column
File
Analysis
Category
Category Summary
Path List
Category: BOT_bist_test_mode_ctl0/mode_reg_4_ Slack Range: all
900
800
700
600
500
400
300
200
100
0
0.69
0.92
H
Path
Clock
ReqTime
Slack
CAMBIST_CLK(leadin...
1.340
-1.064
-1.063
bist_test_mode_ctl0/...
OVA[2]
OA[5]
OVA[5]
OA[6]
OVA[4]
BOT_bist_test_mode_ctl0/mode_reg_4_
BOT_bist_test_mode_ctl0/U10308
BOT_bist_test_mode_ctl0/U12388
-1.077
-428.135
-16.953
0.000
562
16

Name: BOT_bist_test_mode_ctl0/mode_reg_4_
Total Path: 19
Passing Path: 3
Failing Path: 16
WNS: -1.0770
TNS: -16.9530
Comment: Bottleneck Analysis

4. Right-clicking on the category brings up the following pop up. Click on Save Paths () to save a file that can be applied directly to setOptMode -targetBasedOptFile <>.

3. Double-Click on one to get the above info on failing paths etc...

2. For some of the critical instances already meeting timing it is possible to use end point adjustment to optimize them even further as outlined earlier.

1. Bottleneck categories are then created for instances that occur frequently on the critical paths.

Using a Target File to Perform Hold Optimization

To use a target file to perform hold optimization, first create the TBOpt file based on hold timing's critical nodes. For this, specify the `-early` parameter while running the `createTBOptFile` command.

```
createTBOptFile -early -bottleneckInstanceNumber 5 -outFile TBOpt_bottleneck_5_hold.txt
```

Then, use the following commands to specify the above generated target file for hold optimization.

```
setOptMode -opt_target_based_opt_hold_file TBOpt_bottleneck_5_hold.txt
```

```
optDesign -postRoute -targeted -hold
```

Note: You can also launch setup optimization followed by hold optimization using the following command:

```
optDesign -postRoute -targeted -setup -hold
```

Note: Hold optimization is only called in the `-opt_target_based_opt_only_file` mode.

Default Naming Conventions for the TBOpt Flow

The names of the cells and nets added as a result of target-based optimization are annotated with the prefix `FE_TBOPT`.

Prefix	Description
`FE_TBOPTPSBC`	Cell added during postRoute multi-buffering (NBF) in TBOpt flow.
`FE_TBOPTPSBN`	Net added during postRoute multi-buffering (NBF) in TBOpt flow.
`FE_TBOPTC`	Cell added during postRoute single-buffering in TBOpt flow.
`FE_TBOPTN`	Net added during postRoute single-buffering in TBOpt flow.

Optimizing SI Slew and SI Glitches in PostRoute Optimization

By default, postRoute optimization does not fix SI slew violations. To enable this, set the `setOptMode -opt_post_route_fix_si_transitions` parameter to `true`. Before doing this, ensure that the `setDelayCalMode -SIAware` parameter is set to `true`. When this is turned on for `optDesign`, the software issues information and sets the `setSIMode -report_si_slew_max_transition` parameter to true. This is to ensure consistency, so that when either `timeDesign -postRoute` or `reportTranViolation` is run after `optDesign`, the same max trans violations are seen containing the SI slew.

Note: If only `setSIMode -report_si_slew_max_transition` is set to `true`, only SI slew reporting will be turned on and `optDesign` will not fix the SI slews. For this, `setOptMode -opt_post_route_fix_si_transitions` parameter must be set to `true`.

By default, postRoute optimization fixes SI glitch violations. To turn this off , set the `setOptMode -opt_post_route_fix_glitch` parameter to `false` .

Optimizing Signal EM Violations at PostRoute Stage

Electromigration (EM) violations are becoming more common for advanced technologies, especially 28nm and below, and need to be optimized. They usually occur in the design at the output of large drivers. The following topics in this section deal with how to report and optimize them:

- Initial Steps
- Optimization Strategies
- Setting the Switching Activity
- Reporting Signal EM Violations
- Optimizing Signal EM Violations

Initial Steps

Before starting ensure the following:

- The DB is fully legalized and routed. The `fixACLimitViolation` command is not supported in PreRoute mode. It is recommended to have a DRC-clean DB to start with `timeDesign -postRoute` already done.
- EM model file (ICT-EM Tech file) is prepared, if available.
- TCF file is prepared, if available.

Optimization Strategies

There are three fixing strategies:

- Wire widening (NDR)
- Downsizing driver
- Buffering

Among these, NDR is the default strategy. To turn on the downsizing and buffering, set the following options of the `fixACLimitViolation` command:

```
-allowDownsize true
-allowAddBuffer true
```

To turn off NDR, use the following option:

```
-useNDR false
```

Note: Using NDR alone is the preferred approach for fixing EM violations because it causes the least disruption to design timing.

Setting the Switching Activity

If a TCF file is available, use it directly:

```
read_activity_file -format TCF -set_net_freq true test.tcf
```

If a TCF file is not available, you can generate the file. The following example shows how to first set the default switching activity to 0.2 for both input and sequential cells, and then generate and read the TCF file.

```
set_default_switching_activity -reset
set_default_switching_activity -input_activity 0.2 -seq_activity 0.2
propagate_activity
write_tcf test.tcf
read_activity_file -format TCF -set_net_freq true test.tcf
```

Note: For the `read_activity_file` command, ensure that you the `-set_net_freq` parameter set to `true`.

Reporting Signal EM Violations

Signal EM reporting is done by using the `verifyACLimit` command.

To get the initial AC limit violation report, run the `verifyACLimit` command. If you have set the frequency through TCF as specified earlier, use the `-use_db_freq` parameter. To generate the report, specify the `-report` *`reportFileName`* parameter.

Note: This VAC report is then fed to the `fixACLimitViolation` command to read the EM violation information. This will ensure the best fixing quality.

If you have an ICT file for EM model, specify the file using the `-ict_em_models` parameter of the `verifyACLimit` command. The following example shows how to verify the RMS, average, and peak currents using the ICT file, and to print the report with fixWidth information.

```
verifyACLimit -ict_em_models test.ictem -report test_1.rpt -method {rms avg peak} -use_db_freq
```

Note: It is recommended that the design be DRC-clean before you begin. To get the initial DRC number before EM fixing, use the `verify_drc` command. For example:

```
verify_drc -limit 10000000
```

Optimizing Signal EM Violations

Signal EM optimization is done by using the `fixACLimitViolation` command.

If you want to use NDR only, but want to have multiple iterations, do not use the `-maxIter` option. This is because the option, `-maxIter` is not compatible with the `-useReportFile` option, which gives the best estimate of width and distance to be widened.

The best way to achieve multiple iterations of `fixACLimitViolation` (note that one iteration may be sufficient) is described in the following example. In this example, NDR is based on the report file provided by specifying the `-useReportFile` parameter.

```
verifyACLimit -report report1.rpt
fixACLimitViolation -useReportFile report1.rpt
verifyACLimit -report report2.rpt
fixACLimitViolation -useReportFile report2.rpt
verifyACLimit -report report3.rpt
```

Optimizing Power During optDesign

During timing optimization, when the correct power effort is specified, the software is fully aware of the impact of each optimization technique in terms of both leakage and, if specified, dynamic optimization, and it will choose the best one considering all metrics. This functionality is called power-driven optimization. It specifically calls both leakage and dynamic power optimizations. The following topics are covered in this section:

- Leakage Power Optimization
- Dynamic Power Optimization
- Leakage and Dynamic Power Optimization Combined
- Power-Driven Optimization for Different Optimization Modes
- Migrating from Leakage and Dynamic Power Optimization to Power-Driven Optimization
- Specifying the Correct Power Views for Optimization

Leakage Power Optimization

To activate leakage power optimization during timing optimization, run the following command:

```
setOptMode -opt_power_effort {none | low | high} -opt_leakage_to_dynamic_ratio 1.0
```

- When `-opt_power_effort` is set to `none`, no power optimization is performed. This is the default option.
- When `-opt_power_effort` is set to `low`, `optDesign` will optimize leakage power after each timing optimization step. It is fully leakage power-aware at every step of optimization and makes decisions based on that. There are no high leakage cells inserted during hold fixing.
- When `-opt_power_effort` is set to `high`, `optDesign` will optimize leakage power at all stages and it will use a more complex preRoute flow to gain more leakage savings than just a simple call to leakage optimization. There are no high leakage cells inserted during hold fixing. During timing optimization, `optDesign` will use high-effort techniques to ensure the best possible leakage impact of each timing optimization change.

Note: To achieve the best leakage power results, load all the different Vth libraries. It is important to ensure that the correct power view is specified using the `-leakage` parameter of the `set_analysis_view` command. The optimal view for leakage is the one with higher temperature corners (85/125 degrees) and typical libraries.

Dynamic Power Optimization

To activate dynamic power optimization during timing optimization, run the following command:

```
setOptMode -opt_power_effort {none | low | high} -opt_leakage_to_dynamic_ratio 0.0
```

- When `-opt_power_effort` is set to `none`, `optDesign` will not optimize dynamic power. This is the default option.
- When `-opt_power_effort` is set to `low`, `optDdesign` will only optimize dynamic power in the pre-CTS setup optimization phase.
- When `-opt_power_effort` is set to `high`, `optDesign` will optimize dynamic power in the entire setup optimization phases.

Note: Ensure that the correct power view is specified using the `-dynamic` parameter of the `set_analysis_view` command. It is also recommended that you provide an activity file. This can be done by using the following command:

```
read_activity_file -format {VCD | TCF | SAF} file_name
```

In the absence of a switching file, it is recommended you use the following command:

```
set_default_switching_activity-input_activity 0.2 -seq_activity 0.2

propagate_activity

write_tcf tcfName

read_activity_file -format TCF -set_net_freq true tcfName
```

This will ensure both predictability and consistency throughout the flow.

Leakage and Dynamic Power Optimization Combined

To activate leakage and dynamic power-driven optimization together during timing optimization, you can specify a `setOptMode -opt_leakage_to_dynamic_ratio` value between 0.0 and 1.0. In general, there is little value in specifying any value that is not an increment of 0.1.

Note: To decide on the value, it is recommended that you run the `report_power` command once the design has been setup correctly with respect to the following:

- Correct power views are specified
- MVT library is setup
- Switching activity is supplied, and so on

If overall total power reduction is the goal, then looking at the overall impact of internal and switching power versus leakage power, you can decide on the ratio to be specified to achieve the greatest impact. Some examples are provided below:

- `setOptMode -opt_leakage_to_dynamic_ratio 0.5` : This implies that timing optimization will be both dynamic and leakage power-driven and trade-offs need to be done to achieve a balance between leakage and dynamic power optimization.
- `setOptMode -opt_leakage_to_dynamic_ratio 0.1` : This implies that timing optimization will be both dynamic and leakage power-driven but dynamic optimization will be favored.
- `setOptMode -opt_leakage_to_dynamic_ratio 0.9` : This implies that timing optimization will be both dynamic and leakage power-driven but leakage optimization will be favored.

To activate leakage and dynamic power-driven optimization together during timing optimization, you can specify the `set_db opt_leakage_to_dynamic_ratio` attribute value between 0.0 and 1.0. In general, there is little value in specifying any value that is not an increment of 0.1.

Note: To decide on the value, it is recommended that you run the `report_power` command once the design has been setup correctly with respect to the following:

- Correct power views are specified
- MVT library is setup
- Switching activity is supplied, and so on

If overall total power reduction is the goal, then looking at the overall impact of internal and switching power versus leakage power, you can decide on the ratio to be specified to achieve the greatest impact. Some examples are provided below:

- `set_db opt_leakage_to_dynamic_ratio 0.5` : This implies that timing optimization will be both dynamic and leakage power-driven and trade-offs need to be done to achieve a balance between leakage and dynamic power optimization.
- `set_db opt_leakage_to_dynamic_ratio 0.1` : This implies that timing optimization will be both dynamic and leakage power-driven but dynamic optimization will be favored.
- `set_db opt_leakage_to_dynamic_ratio 0.9` : This implies that timing optimization will be both dynamic and leakage power-driven but leakage optimization will be favored.

Power-Driven Optimization for Different Optimization Modes

The `setOptMode -opt_leakage_to_dynamic_ratio` parameter controls the priority of the power-driven optimization. You can set any value between 0 and 1. This parameter is used along with the `-opt_power_effort` parameter. When `setOptMode -opt_power_effort` is set to `low`, power-driven optimization is not the highest priority. In this scenario, power-driven optimization is performed only at certain stages of the `optDesign` flow, which is dependent on the value of the `setOptMode -opt_leakage_to_dynamic_ratio` parameter. This is shown below.

When `setOptMode -opt_power_effort` is set to `high`, power-driven optimization is the highest priority. In this scenario, power-driven optimization is performed at all stages of the `optDesign` flow. However, the type of optimization, dynamic or leakage, depends upon the value of the `setOptMode -opt_leakage_to_dynamic_ratio` parameter. This is shown below.

leakageToDynamicRatio	preCTS	postCTS	postRoute
0.0	Dynamic optimization	Dynamic optimization	Dynamic optimization
> 0.0 and < 1.0	Dynamic and leakage optimization	Dynamic and leakage optimization	Dynamic and leakage optimization
1.0	Leakage optimization	Leakage optimization	Leakage optimization

Migrating from Leakage and Dynamic Power Optimization to Power-Driven Optimization

This section provides a rough guide on how to move from the previous leakage and dynamic power optimization flows to using power-driven optimization. Since power-driven optimization functionality is introduced in the Innovus 15.1 release, there is no direct one to one mapping from previous to new. However, some close approximations are provided below:

- `setOptMode -leakagePowerEffort low  -dynamicPowerEffort none` is roughly equivalent to `setOptMode -opt_power_effort low -opt_leakage_to_dynamic_ratio 1.0`
- `setOptMode -leakagePowerEffort low  -dynamicPowerEffort low` is roughly equivalent to `setOptMode -opt_power_effort low -opt_leakage_to_dynamic_ratio 0.5`
- `setOptMode -leakagePowerEffort low  -dynamicPowerEffort high` is roughly equivalent to `setOptMode -opt_power_effort high -opt_leakage_to_dynamic_ratio 0.1`
- `setOptMode -leakagePowerEffort high -dynamicPowerEffort none` is roughly equivalent to `setOptMode -opt_power_effort high -opt_leakage_to_dynamic_ratio 1.0`
- `setOptMode -leakagePowerEffort high -dynamicPowerEffort low` is roughly equivalent to `setOptMode -opt_power_effort high -opt_leakage_to_dynamic_ratio 0.9`
- `setOptMode -leakagePowerEffort high -dynamicPowerEffort high` is roughly equivalent to `setOptMode -opt_power_effort high -opt_leakage_to_dynamic_ratio 0.5`
- `setOptMode -leakagePowerEffort none -dynamicPowerEffort none` is roughly equivalent to `setOptMode -opt_power_effort none -opt_leakage_to_dynamic_ratio <any value>`
- `setOptMode -leakagePowerEffort none -dynamicPowerEffort low` is roughly equivalent to `setOptMode -opt_power_effort low -opt_leakage_to_dynamic_ratio 0.0`
- `setOptMode -leakagePowerEffort none -dynamicPowerEffort high` is roughly equivalent to `setOptMode -opt_power_effort high -opt_leakage_to_dynamic_ratio 0.0`

Specifying the Correct Power Views for Optimization

As mentioned above, it is important to specify the correct leakage and dynamic view for optimization. The optimal view for leakage is the one with higher temperature corners (85/125 degrees) and typical libraries. The optimal view for dynamic power is dependent both on the design and on your inputs. Ensure that the activity is provided by one or the other methods mentioned above. For specifying the power view, consider the following:

- If the leakage and dynamic view is to be the same, then run the following command:

  `set_power_analysis_mode -leakage_power_view` *`power_view_name`* `-dynamic_power_view` *`power_view_name`*

  You can still use the `-analysis_view` *`power_view_name`* parameter but this parameter will be made obsolete in a future release, so it is not recommended.

- If the leakage and dynamic view is to be different, then run the following command:

  `set_power_analysis_mode -leakage_power_view` *`leakage_view_name`* `-dynamic_power_view` *`dynamic_view_name`*

If the view is not an active view, it will be automatically handled by the optimization code. However, the `report_power` command does not support non-active views. So, for this command, you will need to add the view to the active views using the `set_analysis_view` command and then call the `report_power -view` *`power_view_name`* command. Also, in terms of leakage, if the view is not active then the optimization will be forced to set the `-state_dependent_leakage` parameter of the `set_power_analysis_mode` command to `false`.

Note: If you want to have state-dependent leakage (`-state_dependent_leakage true`) optimization, then the view needs to be made part of the active view list. Also, it is important to ensure that the specified views used are always well defined from both a power and timing point of view to get the optimal QOR.

Using Useful Skew

The useful skew feature in the software modifies the clock arrival time on sequential elements to improve the datapath timing between sequential elements.

In a default flow, Innovus deploys useful skew in all the major commands of the Innovus flow, such as `place_opt_design` and `ccopt_design`. With `setOptMode` options, you can control useful skew at each individual flow step, or disable useful skew altogether for all Innovus commands.

To disable useful skew throughout Innovus, use the following setting:

```
setOptMode -opt_skew {true | false}
```

The `false` setting disables useful skew for all Innovus commands, regardless of any other useful skew settings. The default `true` enables useful skew, subject to the controls for individual commands.

Using Useful Skew in PreCTS Mode

PreCTS optimization inside `place_opt_design` takes advantage of useful skew by default starting from Innovus 16.1 release. You can enable or disable the preCTS useful skew with the following setting:

```
setOptMode -opt_skew_pre_cts {true | false}
```

With default optimization settings, sequential elements are only advanced. With the Early Clock Flow enabled, sequential elements may be either advanced or delayed. For more information about the early clock flow, see Early Clock Flow in the Clock Tree Synthesis chapter.

With default optimization settings, the resulting advances or delays are modeled as SDC network latencies on the corresponding pins, which are then translated into pin insertion delays for CTS during clock tree specification creation.

In the Early Clock Flow, the resulting advances or delays are expressed as both pin network latencies and pin insertion delays. For more information, see Network Latencies in the Clock Tree Synthesis chapter.

Using Useful Skew in PostCTS Mode

PostCTS optimization takes advantage of useful skew by default starting from Innovus 16.1 release. The useful skew deployed in the `ccopt_design` command depends on the optimization effort selected. The `-opt_skew_ccopt` parameter of the `setOptMode` command is used to specify the optimization effort. The available options are `none`, `standard`, and `extreme`. The possible settings are:

- `none`: low effort ccopt_design with no skewing
- `standard`: standard effort `ccopt_design`. This includes the optimization steps in low effort `ccopt_design` with instance sizing and buffer/inverter insertion in the clock tree to improve timing.
- `extreme`: high effort `ccopt_design`. This includes the optimization steps in medium effort `ccopt_design`, using multiple phases of concurrent clock and datapath optimization.

Note: Setting the effort level changes multiple internal properties ‘under the cover’. Advanced users using internal properties advised by their support contact should be aware of this. for more information, see "Controlling Useful Skew Effort in CCOpt" section in the Clock Tree Synthesis chapter.

Note: The `skewClock` command inside `optDesign -postCTS` performs only setup optimization, with awareness of hold timing. There are no options to control performing hold timing optimization inside either `optDesign -postCTS` or `-postRoute`.

Using Useful Skew in PostRoute Mode

PostRoute optimization inside `optDesign -postRoute` takes advantage of useful skew by default starting from Innovus 16.1 release. You can enable or disable postRoute useful skew with the following setting:

```
setOptMode -opt_skew_post_route {true | false}
```

When set to `true`, which is the default setting, the postRoute optimization steps include instance sizing and buffer or inverter insertion in the clock tree to improve setup timing.

Note: The `skewClock` command inside `optDesign -postRoute` performs only setup optimization, with awareness of hold timing. There are no options to control performing hold timing optimization inside either `optDesign -postRoute` or `-postCTS`.

Controlling Useful Skew Optimization

To report the current `setUsefulSkewMode` settings, use the following command:

```
getUsefulSkewMode
```

To control how the software employs useful skew, use the following command:

```
setUsefulSkewMode
```

To exclude boundary sequential cells in useful skew calculations, use the following command:

```
setUsefulSkewMode -opt_skew_no_boundary true
```

If you do not specify this parameter, the software takes boundary cells and ordinary sequential elements into account when calculating useful skew.

To limit the amount of slack, the software can borrow from neighboring flip-flops when performing useful skew operations. By default the software allows a delay of 1 ns. Use the following command to adjust the delay:

```
setUsefulSkewMode -opt_skew_max_allowed_delay value
```

The software's delay calculation and RC extraction methods might differ from those of other sign-off tools, so other setup violations might occur if the 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 software automatically borrows the amount of slack needed (there is no maximum) to reduce setup violations.

Applying Useful Skew Limits to the Complete Flow

The `setUsefulSkewMode` `-opt_skew_max_allowed_delay` parameter imposes a limit on the useful skew applied at a given flop. This limit should include skews generated across all the pre-CTS flow tools but exclude the user-specified pre-skews. However, the pre-existing skews, encoded as pin insertion delay (PID) values, are not considered while imposing the skew limit specified by the `-opt_skew_max_allowed_delay` parameter.

To include pre-existing skews while imposing the useful skew limits, use the `-opt_skew_apply_delay_limits_to_full_flow` parameter. When this parameter is set to `true`, the Early Clock Flow (ECF) is modified so that all pre-existing tool-generated useful skews count towards the limit imposed by the `-opt_skew_max_allowed_delay` parameter. The tool-generated useful skew comprises any PID of category useful_skew or global_skew. PIDs of other categories are not considered.

By default, this parameter is set to `false`, which means the pre-existing skews are not considered while imposing the skew limit.

Distributed Timing Analysis for Hold Fixing

In hold fixing, a significant portion of the CPU runtime is spent computing the setup and hold timing before and after the actual hold fixing step. To reduce the elapsed time, the software supports distributed setup and hold timing analysis.

The distribution is either local or on remote hosts, depending on the EDP settings applied using the `setMultiCpuUsage` command. It is enabled by default and the use model (on a local machine) is :

```
setMultiCpuUsage -localCpu number
```

```
optDesign -hold -postCTS/-postRoute
```

Note: The timing computed in the distributed mode can be different than in default mode when `set_global timing_cppr_threshold_ps` is applied with a value higher than 1ps. But this does not impact timing convergence since in distributed mode the timing would always be on the pessimistic side.

Using Active Logic View for Chip-Level Interface Circuit Timing Closure

The Innovus software provides a top-level interface timing operation flow to perform partitioning and budgeting on a trimmed-down version of the timing graph: an active logic view. This flow helps you close the timing issues of the interface top-level paths as your design has gone through the hierarchical flow until the postRoute stage. This flow also saves the memory usage and provides faster runtime on large designs.

To perform optimization using an active logic view at the postRoute stage, complete the following steps:

1. Load the hierarchical design in the database that is created by `assembleDesign` using the entire post-routed block partition and the top-level partition. Specify the partition information in the database.
   ```
   restoreDesign assembled.enc.dat toplevel_design_name
   ```
2. Perform timing analysis on the design to identify the timing of the full-chip design.
   ```
   timeDesign -postRoute -prefix preOpt
   ```
3. Set the optimization mode to use active logic view. If you specify this parameter, `optDesign` observes the floorplan fence constraint when moving or adding cells.
   ```
   setOptMode -opt_post_route_art_flow true
   ```
4. Run `optDesign`. The `optDesign` command honors active logic view.
   ```
   optDesign -postRoute
   ```
5. Perform timing analysis again to ensure that there are no timing issues.
   ```
   timeDesign -postRoute -prefix postOpt
   ```

Optimizing Timing in On-Chip Variation Analysis Mode

PostRoute timing optimization must be done using on-chip variation (OCV) to account for variations in process, voltage, and temperature (PVT) across the die. When it takes OCV into account, the software calculates early and late delays, and uses them to evaluate setup and hold timing checks. You introduce the delays into the analysis by specifying different min/max corner timing libraries and operating conditions. Early/late variation might also be present due to slew merging effects of multiple input gates in the clock path.

To enable the software to consider multiple libraries and operating conditions, you must specify a multi-mode/multi-corner (MMMC) environment. The MMMC environment must be set up and OCV must be turned on, otherwise the `optDesign` command exits with an error message.

For more information on OCV and MMMC, see the following sections:

- Specifying the MMMC Environment
- Optimizing Timing in OCV Mode Using the Default Delay Calculator

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 stdcmos90T12 \ -late_opcond cmos90T125_slow \ -early_opcond_library
stdcmos90T125 \ -early_opcond cmos90T125 \           -rc_corner rc_worst
create_delay_corner -name delay_corner_min \
            -library_set libs_min \            -late_opcond_library stdcmos90Tm40 \  -late_opcond cmos90Tm40 \
-early_opcond_library stdcmos90Tm40 \ -early_opcond cmos90Tm40_fast \
            -rc_corner rc_best
create_analysis_view -name postCts_max \
            -delay_corner delay_corner_max \
            -constraint_mode postCTS
create_analysis_view -name postCts_min \
            -delay_corner delay_corner_min \
            -constraint_mode postCTS
set_analysis_view -setup postCts_max -hold postCts_min
```

- Two worst-case libraries and two best-case libraries per corner

```
create_library_set -name libs_min_std -timing [list $bestcase_lib_std]
create_library_set -name libs_max_std -timing [list $worstcase_lib_std]
create_library_set -name libs_min_fast -timing [list $bestcase_lib_fast]
create_library_set -name libs_max_fast -timing [list $worstcase_lib_fast]

create_rc_corner -name rc_worst -cap_table CMAX.capTbl
create_rc_corner -name rc_best -cap_table CMIN.capTbl

create_constraint_mode -name postCTS [list xxx.sdc]

create_delay_corner -name delay_corner_max
-late library_set libs_max_std \
-late_opcond_library stdcmos90T125 \
-late_opcond cmos90T125 \
-early library_set libs_max_fast \
-early_opcond_library fastcmos90T125 \
-early_opcond cmos90T125 \
-rc_corner rc_worst

create_delay_corner -name delay_corner_min
-late_library_set libs_min_std \
-late_opcond_library stdccmos90Tm40 \
-late_opcond cmos90Tm40 \
-early_library_set libs_min_fast \
-early_opcond_library fastcmos90Tm40 \
-early_opcond cmos90Tm40 \
-rc_corner rc_best

create_analysis_view -name postCts_max \
-delay_corner delay_corner_max \
-constraint_mode postCTS
create_analysis_view -name postCts_min \
-delay_corner delay_corner_min \
-constraint_mode postCTS

set_analysis_view -setup postCts_max -hold postCts_min
```

Optimizing Timing in OCV Mode Using the Default Delay Calculator

After you specify the MMMC environment, use the following commands to set OCV on and enable Clock Path Pessimism Removal (CPPR), which is highly recommended:

```
setAnalysisMode  -analysisType onChipVariation -cppr
optDesign  -postRoute -hold
```

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:

```
addRepeaterByRule
```

Optimizing Timing When the Constraint File Includes the set_case_analysis Constraint

If you include the `set_case_analysis` constraint in the timing constraint file, the Innovus software sets a constant value on specified signals before performing timing analysis. This constant value is then propagated through the path.

If you use the same timing constraint file for timing optimization, the software does not perform timing optimization on the constant nets because the delays are `0`.

To run timing optimization on these nets, you must first specify the following command:

```
setAnalysisMode -caseAnalysis false
```

Using the Footprintless Flow

By default, the software creates an internal footprint structure based on cell functionality. It is possible that cells with same functionality may be split across 2 or 3 different footprints, based on certain other characteristics, such as, drive strength. This methodology is referred to as the footprintless flow, and has the following advantages over a flow that relies on footprint information from the libraries:

- The libraries do not need to contain footprints, and you do not need to specify a footprint file.
- The following commands are not necessary because the software detects the functionality for inverters and buffers and decides whether a buffer is a delay cell, based on the cell's timing characteristic. The commands have no effect if specified in this flow.
- `setBufFootPrint`
- `setInvFootPrint`
- `setDelayFootPrint`
- The software considers cells with the same functionality but different function syntax as equivalent and allows sizing between such cells.
- If the cell functionality is not defined, the software considers cells with the same cell `user_function_class` as equivalent and allows sizing between such cells. For example:

```
cell (AND2X1) {
  user_function_class : my_and2_class;
}
cell (AND2X2) {
    user_function_class : my_and2_class;
}
```

Since both cells have the same `user_function_class` value, "`my_and2_class`", they will be treated as functionally equal.

Note: Ensure that the following statement is included in the library file, otherwise the `user_class_function` is not read:

```
define(user_function_class,cell,string);
```

- The software prints the list of usable and unusable ("don't use") buffers, inverters, and delay cells to the log file after reading in the libraries, for example:

```
Total number of combinational cells: 620
Total number of sequential cells: 247
Total number of tristate cells: 42
Total number of level shifter cells: 0
Total number of power gating cells: 0 Total number of isolation cells: 0
List of usable buffers: BFX1 BFX2 BFX3 BFX4
Total number of usable buffers: 4
List of unusable buffers: BFX20 BFX32
Total number of unusable buffers: 2
List of usable inverters: IVX1 IVX2 IVX3 IVX4
Total number of usable inverters: 4
List of unusable inverters:
Total number of unusable inverters: 0
List of identified usable delay cells: DLY2 DLY4 DLY8
Total number of identified usable delay cells: 3
List of identified unusable delay cells:
Total number of identified unusable delay cells: 0
```

To revert to the behavior in previous releases (that is, to rely on footprint information in the libraries), use the `loadFootPrint` command. As in those releases, you must specify buffers, inverters, and delay cell footprints according to what was loaded in the footprint file. For more information, see "Using Cell Footprints".

To exclude cells from timing optimization, for example, if the libraries have clock buffers or clock inverters that should be used during CTS but not during timing optimization, set the "don't use" attribute in the timing constraints file, library, or command shell. Timing optimization can resize a "don't use" cell, but does not insert it.

Note: This is the default and recommended methodology since all of it is automated.

For more information see the `setDontUse` command.

Using Cell Footprints

Timing optimization can use information in a footprint file. For example, the buffering mechanisms in `optDesign` add cells only if they are defined in the buffer footprint file.

To disable the footprintless flow (the default timing optimization flow) and load a footprint file, specify the following command:

```
loadFootPrint -infile  footprint_file_name
```

Define footprints in your library or a footprint file by using the following commands, which are enabled when you specify `loadFootPrint`:

- setBufFootPrint
- setInvFootPrint
- setDelayFootPrint

Note: This is not the recommended methodology and should only be used as a workaround.

Viewing Added Buffers, Instances, and Nets

After running timing optimization, use the Design Browser to view the added buffers, instances, and nets. The names of the buffers, instances, and nets added as a result of timing optimization are annotated with the prefix FE_.

For information on using the Design Browser, see the Design Browser section in the Tools Menu chapter of *Innovus Menu Reference*.

Default Naming Conventions

Prefix	**Description**	**Command**
FE_MDBC	Instance added by multi-driver net buffering	optDesign
FE_MDBN	Net added by multi-driver net buffering	optDesign
FE_OCP_RBC	Instance added by rebuffering	optDesign
FE_OCP_RBN	Net added by rebuffering	optDesign
FE_OCPC	Instance added by critical path optimization during preRoute optimization	optDesign
FE_OCPN	Net added by critical path optimization during preRoute optimization	optDesign
FE_OFC	Buffer instance added by addRepeaterByRule or DRV fixing during preRoute optimization	addRepeaterByRule/optDesign
FE_OFN	Buffer net added by addRepeaterByRule or DRV fixing during preRoute optimization	addRepeaterByRule/optDesign
FE_PHC	Instance added by hold time repair	optDesign
FE_PHN	Net added by hold time repair	optDesign
FE_PSBC	Instance added by buffer insertion during postRoute optimization	optDesign

FE_PSBN	Net added by buffer insertion during postRoute optimization	optDesign
FE_PSRC	Instance added by postRoute restructuring	optDesign
FE_PSRN	Net added by postRoute restructuring	optDesign
FE_PSC	Instance added by postRoute setup repair	optDesign
FE_PSN	Net added by postRoute setup repair	optDesign
FE_PDC	Instance added by postRoute DRV fixing	optDesign
FE_PDN	Net added by postRoute DRV fixing	optDesign
FE_RC	Instance created by netlist restructuring	optDesign
FE_RN	Net created by netlist restructuring	optDesign
FE_USKC	Instance added during useful skew optimization	optDesign/skewClock
FE_USKN	Net added during useful skew optimization	optDesign/skewClock
FE_ARRC	Instance added by addRepeaterByRule	addRepeaterByRule
FE_ARRN	Net added by addRepeaterByRule	addRepeaterByRule

Using Signoff Timing Analysis to Optimize Timing and Power

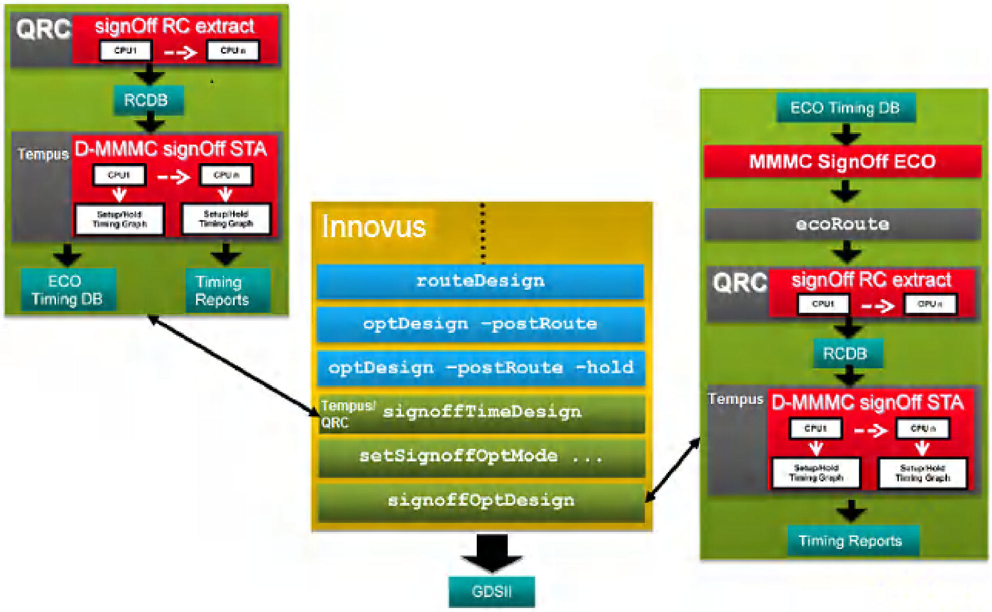
The signoff timing optimization feature lets you run timing and power optimization within Innovus on Signoff parasitic from Quantus and Signoff timing from Tempus. This feature gives a complete automated solution for using the entire signoff tools through one high level super command.

In a good timing closure methodology, at the implementation stage the timing targets that are set by the signoff Static Timing Analysis (STA) tool should be met. In order to ensure that the design state is close to sign-off quality, the timing reported by the implementation tool must correlate as much as possible to the signoff STA tool. In Innovus, signoff timing analysis and optimization capabilities provide the following features:

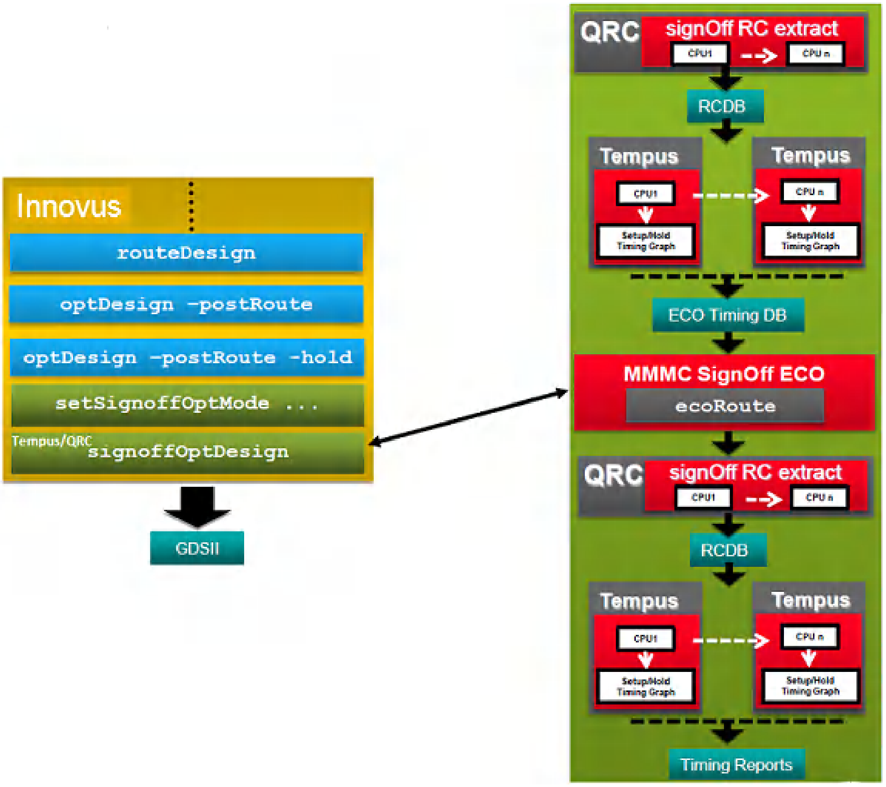
- Lets you run timing and power optimization on signoff timing within Innovus.
- Maximizes the usage of Cadence tools - Innovus will enable you to run extraction (Quantus) and Tempus without extra effort.

Running MMMC SignOff ECO within Innovus

To provide signoff timing report in Innovus using Tempus, you can use the `signoffTimeDesign` command. This command uses Quantus and Tempus in standalone mode to perform signoff STA using the DMMMC infrastructure and save an ECO Timing DB per view. This signoff timing can be optimized using `signoffOptDesign` . Similarly, `signoffOptDesign` can be used to perform power optimization on this signoff timing. The following diagram illustrates the flow and architecture of each signoff command:



In case the ECO Timing DB are not available when `signoffOptDesign` command is run, then the super command will automatically run `signoffTimeDesign` under the hood, as shown in the figure below:



Signoff Timing Optimization provides a flexible flow to accommodate any specific methodology:

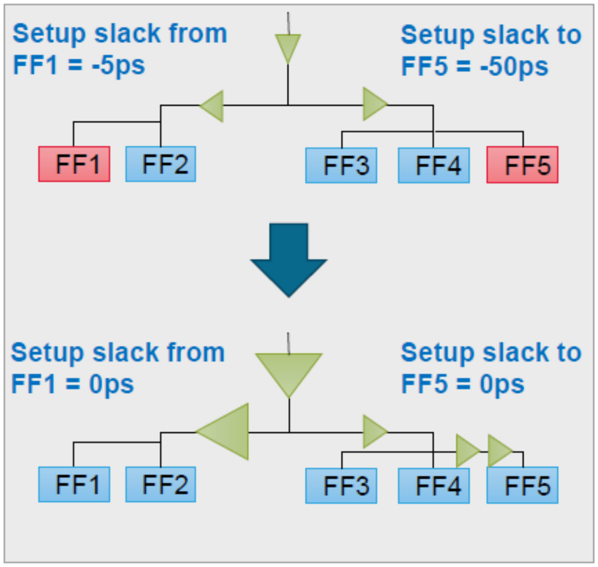
- If parasitics should be extracted using TQuantus or IQuantus, this can be done manually by the user. Then `signoffTimeDesign` can be used with the `-reportOnly` option to reuse those parasitics and skip the Quantus call.
- In case specific steps/options should be performed before/during ECO routing, this step can be performed manually after running `signoffOptDesign` with the `-noEcoRoute` option.
- When specific signoff STA commands/options/globals should be applied, you can set these in a file and pass it to Tempus through `setSignoffOptMode -preStaTcl` *`file`* option.

Performing Clock Skewing for Setup Timing Closure

Tempus ECO provides the ability to fix clock paths in order to fix the remaining setup timing violations. If the data path is fully optimized, the software uses useful skew, which includes two methods: early clock and late clock. This either reduces the launch clock delay or increases capture clock delay. The Tempus ECO engine can advance and delay clock arrival time to improve setup timing while not creating any new hold timing violations.
Below are the key features of a skew engine:

- Advanced clock tree manipulation to allow larger flexibility for clock skewing
- Sizing, deletion, and buffering (includes inverter pair and load cell) at any clock tree level
- Concurrent data path and clock path optimization for optimal timing closure
- Sophisticated push-pull weight and timing bottleneck-based engine
- Support for max level skewing to avoid excessive padding using `setSignoffOptMode -clockMaxLevel <`*`value`*`>`

Multi-Level Sizing and Buffering


Setup slack from FF1 = -5ps
Setup slack to FF5 = -50ps
FF1
FF2
FF3
FF4
FF5
Setup slack from FF1 = 0ps
Setup slack to FF5 = 0ps
FF1
FF2
FF3
FF4
FF5

Note: It is mandatory to provide a list of clock cells that are allowed to be used on the clock tree. Those cells can be inverter, buffer, or any other combinational cells used as clock gaters.
Example:

```
setSignoffOptMode -clockCellList {CKBUFX1 CKBUFX4}
setSignoffOptMode -allowSkewing true
SignoffOptDesign -setup
```

The above commands performs clock skewing during setup fixing. Only `CKBUFX1` and `CKBUFX4` can be added to the clock tree.

Signoff Timing Analysis in Innovus using Timing Debug

Signoff timing analysis is performed by Tempus that saves signoff timing report `(in .mtarpt`) files per view. These files can be loaded in Innovus through the global timing debug (GTD) interface in order to perform fine grain timing analysis.

Fixing SI Glitch, SI Slew, and SI Crosstalk Delta Delay Violations

Signoff ECO is able to fix different violations related to SI glitch, SI slew, and SI crosstalk delta delay.

SI Glitch Violations

A signal integrity noise glitch, also called voltage bump, generated by crosstalk coupling can propagate and amplify while traveling along a path. As a result, this glitch can cause incorrect signal state change. The software provides the ability to fix SI glitch violations in signoff ECO. This is done using the `-fixGlitch` parameter of the `setSignoffOptMode` command.

Syntax:

```
setSignoffOptMode -fixGlitch true | false
```

When set to `true`, the software performs SI glitch fixing during DRV fixing using resizing and buffering techniques.

The default value is `false`.

Notes:

- This parameter must also be set during ECO DB generation.
- The glitch fixing is done before max_tran/max_cap fixing.
- It is mandatory to set the `-enable_glitch_report` parameter of the `setSIMode` command to true and use the `report_noise` command to analyse and get a signoff glitch report.
- For any remaining SI glitches that are not fixed using the resizing and buffering techniques, it is recommended to select those nets and re-route them using an extra SI property.

Example:

- The example below shows fixing of SI glitch violations only:

  ```
  setSignoffOptMode -fixMaxCap false -fixMaxTran false -fixGlitch true
  ```

- The example below shows fixing of SI glitch violations using DRV fixing in addition to the regular `max_tran/max_cap` violations:

  ```
  setSignoffOptMode -fixGlitch true
  signoffOptDesign -drv
  ```

SI Slew Violations

Crosstalk effects caused by parasitic capacitance between adjacent nets can lead to a large signal transition (SI slews) on the victim nets. The software provides the ability to consider SI slews apart from base slews while performing transition violation fixing during DRV optimizations. All other optimizers that include setup, hold, leakage, area, power, and dynamic do not consider SI slews. This must be run as the first step of optimization before proceeding for any other incremental optimization.

This is done using the `-fixSISlew` parameter of the `setSignoffOptMode` command.

Syntax:

```
setSignoffOptMode -fixSISlew true | false
```

When this parameter is set to `true`, the tool checks max_tran rule against the SI slew and such violations are resolved during DRV fixing using resizing and buffering techniques.

The default value is `false`.

Note:

- This parameter must be set during ECO DB generation in addition to `setSIMode -enable_drv_with_delta_slew true`
- To analyze and report SI slew violations, run the following commands:

```
setSIMode -enable_drv_with_delta_slew true
report_constraint -drv_violation_type max_transition -all_violators
```

Examples:

- The below example shows fixing SI slews violations only:
  `setSignoffOptMode -fixMaxCap false -fixSISlew true`
- To analyze and report SI slew violations use the following commands:
  `setSIMode -enable_drv_with_delta_slew true`
  `report_constraint -drv_violation_type max_transition -all_violators`

SI Crosstalk Delta Delay Violations

When aggressors are transitioning opposite the victim, it causes the arrival time to increase (positive delta delay). However, when aggressors are transitioning at the same time as the victim, it causes the arrival time to reduce (negative delta delay). The software provides the ability to fix crosstalk delta delay in signoff ECO. This helps in improving the design robustness, SI-delay fixing convergence, and minimizing the setup versus hold timing conflicts on critical paths.

This is done using the `-fixXtalk` parameter, of the `setSignoffOptMode` command.

Syntax:

```
setSignoffOptMode -fixXtalk true | false
```

When set to `true`, the software reduces the crosstalk on the net that violates the thresholds set by you.

The default value is `false`.

Notes:

- This parameter must also be set during ECO DB generation.
- xtalk delta delay fixing is done after max_tran/max_cap fixing as a separate phase.
- It is mandatory to set the below SI mode options:
  `setSIMode -separate_delta_delay_on_data true -delta_delay_annotation_mode lumpedOnNet`

In addition, the following parameters of the `setSignoffOptMode` command are used to select nets for xtalk fixing during optimization:

- To fix the nets with the crosstalk delta delay value equal to or greater than a specified threshold value:
  a. For setup views: `setSignoffOptMode -setupXtalkDeltaThreshold` *`value in ns`* (default 0.3 ns)
  b. For hold views: `setSignoffOptMode -holdXtalkDeltaThreshold` *`value in ns`* (default 0.3 ns)
- To fix the nets with the slack threshold value equal to or less than a specified threshold value:
  a. For setup views: `setSignoffOptMode -setupXtalkSlackThreshold` *value in ns* (default 1000 ns)
  b. For hold views: `setSignoffOptMode -holdXtalkSlackThreshold` *value in ns* (default 1000 ns)

Example:

The following example uses DRV fixing to reduce the crosstalk delay on all nets that violated the `300ps` threshold rule in setup views:

```
setSignoffOptMode -setupXtalkDeltaThreshold 0.3

setSignoffOptMode -fixXtalk true

signoffOptDesign -drv
```

Optimization in Path-Based Analysis (PBA) Mode

At implementation stage, tools are using the Graph-Based Analysis (GBA) mode because that allows fast turnaround time to analyze timing and to update timing incrementally. The drawback is that the GBA mode is generating pessimistic timing. At the signoff stage, where perfect accuracy is needed, you can enable the PBA mode and perform the final pass of timing closure. In addition, it is recommended to run the area or power recovery engine to reach the best possible Power Performance Area (PPA) metrics. When enabling PBA, each timing path is timed in its own context so that the slews or AOCV factors are computed based only on the path being timed.
The following five options are specific to the PBA mode for Tempus ECO:

- To enable PBA and select the retiming mode:
  `setSignoffOptMode -retime none`
- To select whether PBA is applied to Setup or Hold or both:
  `setSignoffOptMode -checkType early`
- To specify the maximum slack to be considered for retiming:
  `setSignoffOptMode -maxSlack 0`
- To specify the maximum number of paths to be retimed in total:
  `setSignoffOptMode -maxPaths -1`
- To specify the number of paths to be retimed for each endpoint:
  `setSignoffOptMode -nworst -1`

The following is an example of SOCV PBA based Leakage optimization:

```
restoreDesign postroute.enc.dat topChip
<set all signoff STA views>
<apply signoff STA settings/options/globals >
<load parasitic>
setMultiCpuUsage -localCpu 16 -remoteHost 4 -cpuPerRemoteHost 4
setSignoffOptMode -retime path_slew_propagation
setSignoffOptMode -checkType both
setSignoffOptMode -maxSlack 10
setSignoffOptMode -maxPaths 5000000
setSignoffOptMode -nworst 50
setSignoffOptMode -pbaEffort high
setSignoffOptMode -saveEcoOptDb ECO-DB-PBA
signoffTimeDesign -reportOnly
setSignoffOptMode -loadEcoOptDb ECO-DB-PBA
signoffOptDesign -noEcoRoute -leakage
```

Note: In the above example, the `-maxSlack` option was set to 10 (DB timing unit) so that positive GBA slack paths are also getting retimed in order to get more positive setup slack on those paths. That extra positive slack can then be used to perform more power recovery.

Total Power Optimization

The `signoffOptDesign` command is able to perform Total Power optimization to reclaim total power in the design. This is done using the `-power` parameter of the `signoffOptDesign` command. Using this parameter, the tool replaces the need to run the leakage and dynamic power optimizations in consecutive steps. During total power optimization the engine concurrently minimizes leakage, switching, and Internal power to achieve the lowest overall total power consumption.

During total power optimization, the `report_power` command needs to be run before ECO DB writing. The netlist activity can either be computed using default toggling rate on the inputs or coming from an external VCD/SAIF file. In addition, the software performs swapping and sizing on both the combinational and sequential cells, and also does buffer removal when enabled.

Example:

```
report_power
setSignoffOptMode -deleteInst true -optimizeSequentialCells true
signoffOptDesign -power
```

Note: To allow sequential elements to be resized during power optimization, ensure that there is no fixed attribute applied on them. If you measure setup timing degradation after doing a power optimization, see section *Setup Timing Recovery After a Large Leakage or Total Power Optimization.*

Setup Timing Recovery After a Large Leakage or Total Power Optimization

It is quite common to get a large number of ECOs generated when doing power optimization. This means that thirty to eighty percent of the netlist has been changed. Although the tool is doing an accurate timing estimation for each of the ECOs generated, it is not able to guarantee a zero impact on setup timing. To recover the setup timing up to an initial value, a few ECOs having low or no power cost might be needed. This setup timing recovery can be achieved by running a new signoff timing analysis and then calling setup optimization with `setSignoffOptMode -setupRecovery true`. The recovery will be done with Vth swapping only, which means that same size and same pin geometry cell sizing. Because of this there is no need to re-route the design and final timing can be generated.

Getting the Best Total Power Optimization Recipe

To achieve the best QoR for total power optimization, ensure the following:

- Timing analysis is performed in the PBA mode
- Retiming in PBA mode is performed on positive GBA slack paths
- The `setSignoffOptMode -pbaEffort` parameter is set to `high`
- The list of usable cells is as large as possible and any fixed placement attribute on sequential elements is removed to allow them to be sized
- No sequential elements are fixed otherwise they will not be considered for resizing
- While performing aggressive power optimization, a large number of netlist changes are generated and the timing histogram shows a huge peak around the `0ns` slack

  Note : In this context, the software cannot completely avoid setup timing degradation and a light setup recovery based on Vth cell swapping only helps to get back to the initial timing. The solution is to perform a setup timing recovery optimization after generating fresh signoff timing post-power optimization.

Template Script for Total Power Optimization in Innovus Cockpit

```
source postroute.enc
<set all signoff STA views>
<apply signoff STA settings/options/globals>
extractRC
setDistributeHost -lsf ...
setMultiCpuUsage -localCpu 16 -remoteHost 3 -cpuPerRemoteHost 8
setSignoffOptMode -preStaTcl preStaTcl.cl \
-retime path_slew_propagation \
-checktype both -pbaEffort high \
-maxSlack 10 -maxPaths 10000000 -nworst 50 \
-deleteInst true \
-saveEcoOptDb ECO-DB-PBA
signoffTimeDesign -reportOnly -outDir RPT-PBA-init
report_power
setSignoffOptMode -loadEcoOptDb ECO-DB-PBA
signoffOptDesign -noEcoRoute -power
ecoRoute
extractRC
setSignoffOptMode -maxSlack 0 -maxPaths 10000000 -nworst 50 \
-saveEcoOptDb ECO-DB-PBA2
signoffTimeDesign -reportOnly -outDir RPT-PBA-recovery
setSignoffOptMode -loadEcoOptDb ECO-DB-PBA2 -setupRecovery true
signoffOptDesign -noEcoRoute -setup
signoffTimeDesign -reportOnly -outDir RPT-PBA-recovery -noEcoDB
```

Path Group Support

Tempus ECO supports the following ways for path group-based fixing :

1. Endpoint-based inclusion/exclusion per view
2. Endpoint-specific slack adjustment per view (both positive and negative adjustments)
3. Option to fix only register to register paths

Path group support for Tempus ECO can be used for both Hold and Setup fixing.

Note: This feature is currently not supported for DRV or leakage optimization.

1. Endpoint-based inclusion/exclusion per view

You can specify endpoints that need to be included or excluded during optimization for a view with the following `setSignoffOptMode` parameters:

- `-selectHoldEndpoints`
- `-selectSetupEndpoints`

File format:

`<View> include/exclude <Endpoints>`

`<View>` can be specified as either *`viewName`* or `V*` (if the endpoints are to be included or excluded for all views)

`<Endpoints>` can be specified as a list of endpoint names separated by a space. For example, `E1 E2 E3` slack range in nanosecond: `<minSlack> <maxSlack>` where all endpoints with a slack greater than or equal to `minSlack` and slack less than or equal to `maxSlack` will be included or excluded. For example, `-2.0 -1.0 E*`, if all endpoints for the view need to be included or excluded.

Sample configuration files

The lines starting with # are comments and will be skipped.

File: `hold_exclude_example`

```
# Exclude endpoints EXECUTE_INST/pc_acc_reg/SE and EXECUTE_INST/read_prog_reg/SE from view core+typ-rcMin for hold fixing
core+typ-rcMin exclude EXECUTE_INST/pc_acc_reg/SE EXECUTE_INST/read_prog_reg/SE
```

File: `hold_include_example_with_range`

```
# Include only the endpoints that have slacks >= -0.5 ns and <= 0.04 ns for view core+best-rcTyp for hold fixing
core+best-rcTyp include -0.5 0.04
```

2. Endpoint-specific slack adjustment per view

You can specify slack adjustment (margin) for selected endpoints during optimization for a view with the following `setSignoffOptMode` parameters:

- `-specifyHoldEndpointsMargin`
- `-specifySetupEndpointsMargin`

File format:

`<View> <Margin> <Endpoints>`

`<View>` can be specified as either *`viewName`* or `V*` (if the endpoints are to be included or excluded for all views)

`<Margin>` is specified as float value in nanosecond. Margin value is subtracted from the endpoint slack so a positive margin means that the endpoint slack would be degraded by the margin amount.

`<Endpoints>` can be specified as a list of endpoint names separated by a space. For example, `E1 E2 E3` slack range in nanosecond: `<minSlack> <maxSlack>` where all endpoints with a slack greater than or equal to `minSlack` and slack less than or equal to `maxSlack` would be included or excluded. For example, `-2.0 -1.0 E*` if all endpoints for the view need to be included or excluded.

Sample configuration files

The lines starting with # are comments and would be skipped.

File : `hold_margin_allViews_slackRange_example`

```
# Apply margin of 0.1 nanosecond to all endpoints that have slacks between -0.07 to -0.03 for all hold views
V* 0.1 -0.07 -0.03
```

File : `hold_margin_allEndPoints_example`

```
# Apply margin of 0.2 nanosecond to all endpoints for hold view core+typ-rcMin
core+typ-rcMin 0.2 E*
```

File : `hold_margin_selectedEndPoints_example`

```
# Apply margin of 0.6 ns to endpoints TDSP_CORE_MACH_INST/phi_6_reg/SE and TDSP_CORE_MACH_INST/phi_1_reg/SE for hold view core+best-rcTyp
core+typ-rcMin 0.6 TDSP_CORE_MACH_INST/phi_6_reg/SE TDSP_CORE_MACH_INST/phi_1_reg/SE
```

3. Option to fix only register-to-register paths

You can choose to fix only register-to-register paths during Hold/Setup fixing with the parameter specified below. Other path group configurations will be ignored in this mode. The timing for non-register-to-register paths for fixing mode (Hold or Setup) will be adjusted to 0 but the timing for other mode (Setup during Hold fixing/Hold during Setup fixing) is not affected.

```
setSignoffOptMode -optimizeCoreOnly {true | false}
```

Sample Template Scripts

- The following example shows graph-based analysis (GBA) signoff STA followed by Setup and Hold optimization:

```
source postRoute.enc
setMultiCpuUsage -localCpu 4 -remoteHost 3 -cpuPerRemoteHost 4
setSignoffOptMode -postStaTcl postStaTcl.tcl
signoffTimeDesign
signoffOptDesign -setup -noEcoRoute
signoffOptDesign -hold
```

- The following example shows GBA signoff STA followed by Setup optimization that is based on TQuantus:

```
source postRoute.enc
setMultiCpuUsage -localCpu 4 -remoteHost 3 -cpuPerRemoteHost 4
setExtractRCMode -coupled true -engine postRoute -effortLevel medium
extractRC
setSignoffOptMode -preStaTcl preStaTcl.tcl
signoffTimeDesign -reportOnly
signoffOptDesign -setup -noEcoRoute
ecoRoute
extractRC
signoffTimeDesign -reportOnly -noEcoDb
```

- The following example shows how to fix IR drop voltages for all instances in the design without creating any DRV violations.

```
signoffTimeDesign -reportOnly
setSignoffOptMode -fixIrDrop true
setSignoffOptMode -loadIrdropDb my_ir_db
setSignoffOptMode -fixMaxTran false -fixMaxCap false
signoffOptDesign -noEcoRoute -drv
```

- The following example shows signoff STA followed by Hold optimization in PBA Setup and Hold mode:

```
source postRoute.enc
setMultiCpuUsage -localCpu 4 -remoteHost 3 -cpuPerRemoteHost 4
setSignoffOptMode -preStaTcl preStaTcl.tcl
setSignoffOptMode -retime aocv_path_slew_propagation -checkType both
signoffTimeDesign
signoffOptDesign -hold
```

Note: The `preStaTcl.tcl` script allows you to apply any signoff STA related settings or reset any of the `set*mode` commands.

- The following example shows the AOCV PBA-based leakage optimization flow with positive PBA slack path retiming and high-effort PBA mode:

```
source postroute.enc
<set all signoff STA views>
<apply signoff STA settings/options/globals >
extractRC
setDistributeHost -lsf …
setMultiCpuUsage -localCpu 16 -remoteHost 3 -cpuPerRemoteHost 8
setSignoffOptMode -preStaTcl preStaTcl.tcl \
                  -retime aocv_path_slew_propagation \
                  -checkType both -pbaEffort high \
                  -max_slack 10 -max_paths 10000000 -nworst 50 \
                  -saveEcoOptDb ECO-DB-PBA
signoffTimeDesign -reportOnly -outDir RPT_PBA_init
setSignoffOptMode -loadEcoOptDb ECO-DB-PBA
```

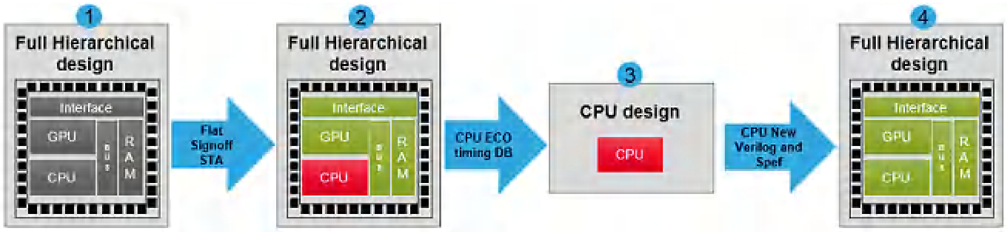
```
signoffOptDesign -noEcoRoute -leakage
setSignoffOptMode -max_slack 0 -max_paths 10000000 -nworst 50 \
                  -saveEcoOptDb ECO-DB-PBA2
signoffTimeDesign -reportOnly -outDir RPT_PBA_recovery
setSignoffOptMode -loadEcoOptDb ECO-DB-PBA2 -setupRecovery true
signoffOptDesign -noEcoRoute -setup
signoffTimeDesign -reportOnly -outDir RPT_PBA_recovery -noEcoDB
```

Top Down Block ECO flow using Tempus Signoff Timing

The Top Down Block ECO flow is used to optimize a block-level design while using ECO Timing database (DB) generated during top-level full flat STA. The software fixes the timing violations for large-scale designs with faster turnaround time and consumes less memory, as an ECO fixing session only loads the block-level data.

Using this flow, you can run timing analysis on a hierarchical design and find timing violations in one particular block. Instead of running ECO at the full-chip level, you can generate ECO Timing DB for a specific block. Then, in a subsequent software session, you can load data just for the identified block (including physical data) and then reuse the ECO Timing DB generated earlier and rerun ECO fixing. Once the ECOs are implemented, you can rerun timing analysis on the hierarchical design and as a result, fewer or no timing violations are reported, for the identified block.

The following diagram shows the steps in Top Down Block ECO flow:



Step 1: Design is assembled and final signoff timing constraints are applied.
Step 2: Timing is met except in the CPU block. At block level, CPU design does not include the latest timing signoff constraints.
Step 3: Load Verilog/DEF/Libs for CPU block only, and run ECO using ECO Timing DB generated at top level.
Step 4: Timing is met for all blocks.

The `-blockScopeName` parameter of the `setSignoffOptMode` command is used to automate the top-down block ECO flow. Using this parameter, you can provide a module name or hierarchical instance name.

The following scripts are used for running top down block ECO flow:

Full-Chip Script:

```
<load hierarchical design>
<load parasitics>

setSignoffOptMode -blockScopeName "top/CPU"
setSignoffOptMode -saveEcoOptDb ECO-DB-CPU
```

Block-Level Script:

```
<load block level>
<load parasitics>

setSignoffOptMode -loadEcoOptDb ECO-DB-CPU

signoffOptDesign -setup
```

Notes:

- At block level, you do not need to set this parameter as the tool automatically identifies the running of ECO fixing on an ECO timing DB, which is generated in block scope context.

- There is no need to assemble the physical data for the full-chip session.
- You do not have to provide SDC at block level. All timing information extracted from top level is embedded in the ECO Timing DB directory.
- This flow does not support Master/Clone module, but you can select one clone by providing a hierarchical instance name and then run the Top down Block ECO flow.

Metal ECO Flow

The Metal ECO flow, also known as Post-Mask flow supports the Gate Array filler cells. Using this flow, the timing closure engine performs the netlist change without touching the base layer mask.

Before starting metal ECO, perform the following sanity checks:

- All instances of the design must be placed (no unplaced instance allowed).
- The initial design placement density must be 100%, that is no empty spaces left.
- The list of Gate Array filler cells must contain enough granularity to ensure that any empty space created during metal ECO can be filled back.
- Do not specify cells through setTieHiLoMode if Tie cell insertion is not allowed.

The following command is used to run the metal ECO flow:

```
setSignoffOptMode  -postMask true
```

The software performs the following netlist changes:

- Inserts the Gate Array buffers or an inverter pair in place of existing Gate Array filler cells.
- Resizes regular cells to Gate Array cells. It also resizes Gate Array cells to Gate Array cells.
- Deletes a regular cell.

The software automatically identifies the Gate Array cells by checking which library instances have the same SITE name as the Gate Array filler cells specified by the user. Here, empty spaces will be filled up by the Gate Array filler cells to ensure 100% placement density.

By default, the `-postMask` parameter is set to `false`.

The following command lists all the Gate Array filler cells, which can be deleted or inserted back:

```
setSignoffOptMode  -useGaFillerList list_of_Gate_Array_filler_cells_name
```

Note that this parameter is applicable in metal ECO mode only.

Note:

The Metal ECO flow is supported by DRV, Setup, and Hold fixing.

When `setTieHiLoMode -cell {{tieLo_name tieHi_name}}` setting is used, this flow will insert the TieLo cells to connect dangling input pins, in case the existing TieLo cells in the floorplan are extremely far or too overloaded.

Example:

The following command enables Hold fixing in the metal ECO mode by allowing the listed Gate Array filler cells to be deleted or inserted during the ECO process.

```
setSignoffOptMode  -postMask true

setSignoffOptMode -useGaFillerList {GFILL1BWP GFILL2BWP GFILL4BWP GFILL10BWP}

signoffOptDesign -hold
```

One Pass Logical Equivalence Check (LEC)

The one pass LEC flow lets you verify netlist changes during the place and route (PnR) flow for logical equivalence in a single step. In this flow, you can do a Register Transfer Level (RTL) to netlist comparison between RTL and postRoute optimization (PRO) netlist - any PnR stage netlist - in a single step.

One Pass LEC Flow

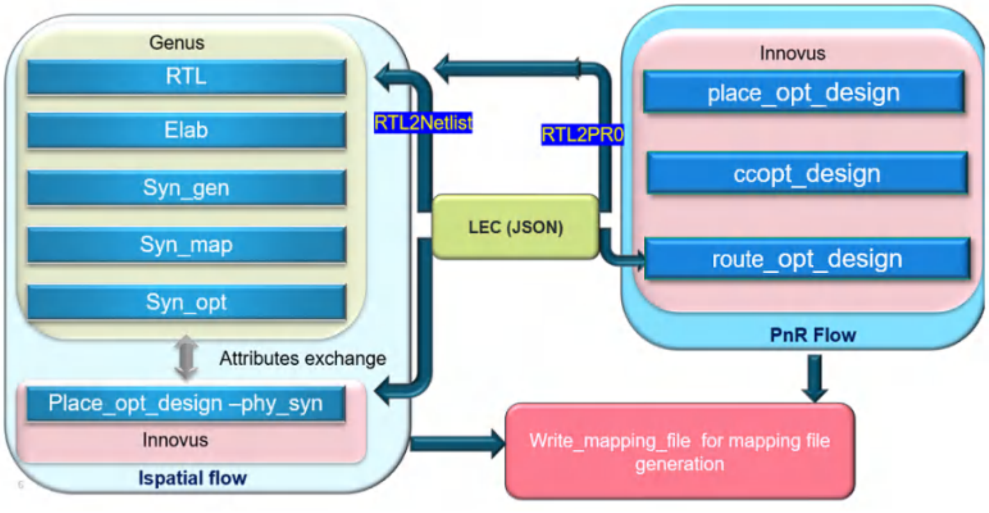
The one pass LEC flow is enabled by setting the following parameter to `true`.

```
setOptMode -opt_one_pass_lec
```

By default, this parameter is set to `false`.

The output is a mapping file that lists the initial netlist key points names and corresponding current netlist key points names. This is generated using the following command:

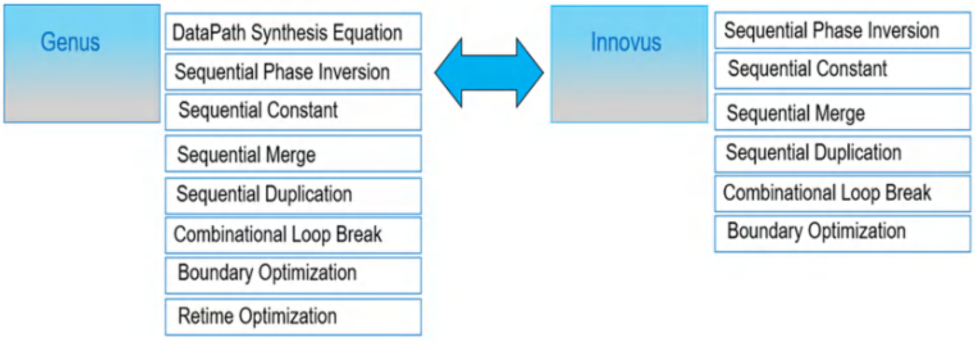
```
write_name_mapping
```



Attribute Exchange between Genus and Innovus

For this flow, Genus and Innovus exchange optimization done in their respective tools.

This is done using attributes that are stored at various stages in various optimization steps. Multiple attributes are created in Innovus for following optimization:



Clock Gate Handling

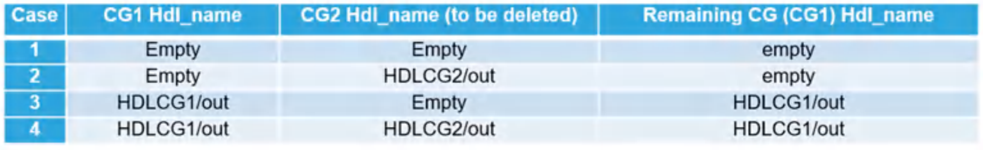
Clock gate handling is available for the CGs already present in the RTL. There is no verification required for the CGs inserted during the implementation.

The non-RTL CGs are skipped during mapping file generation.

Mapping depends on the value of the attribute, hdl_name on the CG. If the value is non-empty, it indicates that the corresponding CG is present in the RTL. Only the CG pins with non-empty hdl_name will be mapped.

In case of merging of two CGs, the `hdl_name` attribute of the remaining CGs will be copied to the resulting CG.

Example:



Case	CG1 Hdl_name	CG2 Hdl_name (to be deleted)	Remaining CG (CG1) Hdl_name
1	Empty	Empty	empty
2	Empty	HDLCG2/out	empty
3	HDLCG1/out	Empty	HDLCG1/out
4	HDLCG1/out	HDLCG2/out	HDLCG1/out

Record of deleted CG pins

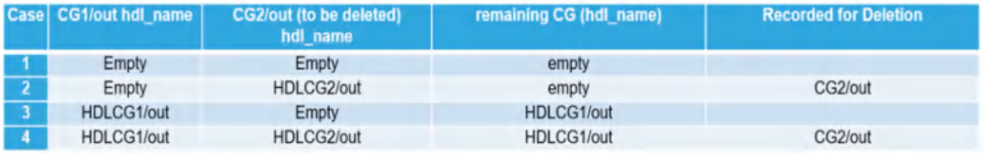
The deleted CGs (with non-empty `hdl_name`) are printed by using the `seq_reason_deleted` attribute.

```
dbGet designs .seq_reason_deleted
```

This attribute captures the name of the deleted CGs and prints the original name of the CG in the following format in the shell:

```
{{icg1} {HDLCG2 optimized}}
```

Example:



Case	CG1/out hdl_name	CG2/out (to be deleted) hdl_name	remaining CG (hdl_name)	Recorded for Deletion
1	Empty	Empty	empty	
2	Empty	HDLCG2/out	empty	CG2/out
3	HDLCG1/out	Empty	HDLCG1/out	
4	HDLCG1/out	HDLCG2/out	HDLCG1/out	CG2/out

Name Mapping File

The initial netlist key points names and corresponding current netlist key points names are written/updated to a file. The mapping file consists of pin mapping information of all changes done during flow (between the comparison points) including polarity changes.

The following is used to write out name mapping file from Innovus.

```
write_name_mapping
```

Example

The following is an example of the usage of mapping command for 1 Pass LEC:

```
RTL2NETLIST:
```

```
write_name_mapping -hdl -map_pins output -pin_polarity true
```

```
NETLIST2NETLIST:
```

```
write_name_mapping -map_pins output -pin_polarity true
```

Using the NanoRoute Router

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

Routing Phases

Full routing consists of global and detailed routing. You can repeat detailed routing incrementally on a routed database. Incremental detailed routing is not the same as ECO routing. For information, see Global Routing and Detailed Routing.

ECO routing consists of incremental global and detailed routing passes on a routed design. During ECO routing, the router completes partial routes and makes minimal changes to existing wire segments. For information, see Running ECO Routing.

Global Routing

During this phase, the router breaks the routing portion of the design into rectangles called global routing cells (gcells) and assigns the signal nets to the gcells. The global router attempts to find the shortest path through the gcells, but does not make actual connections or assign nets to specific tracks within the gcells. It tries to avoid assigning more nets to a gcell than the tracks can accommodate. The detailed router uses the global routing paths as a routing plan.

The router can generate a map of the gcells, called a congestion map, that you can examine to see the approximate number of nets assigned to the gcells. The congestion map uses colors to indicate whether there are too few, too many, or the correct number of nets assigned to the gcells. If the router assigns too many nets to a gcell, it marks the gcell as over-congested. You can also read the Congestion Analysis Table in the Innovus log file to learn the distribution and severity of the congestion after global routing.

Related Topics

- For more information on gcells, see "GCell Grid" in the "DEF Syntax: chapter of the LEF/ DEF Language Reference.
- For more information on the congestion map and table, see Checking Congestion.

Detailed Routing

During this phase, the NanoRoute router follows the global routing plan and lays down actual wires that connect the pins to their corresponding nets. The detailed router creates shorts or spacing violations rather than leave unconnected nets. You can run detailed routing on the entire design, a specified area of the design, or on selected nets. In addition, you can run incremental detailed routing on a database that has already been detail routed. The router runs search-and-repair routing during detailed routing. During search and repair, it locates shorts and spacing violations and reroutes the affected areas to eliminate as many of the violations as possible. The primary goal of detailed routing is to complete all of the required interconnect without leaving shorts or spacing violations.

During detailed routing, the router divides the chip into areas called switch boxes (SBoxes), which align with the gcell boundaries. The SBoxes can be expressed in terms of gcells; for example, a 5x5 SBox is an SBox that encompasses 25 gcells. The SBoxes overlap with each other and their size and amount of overlap might vary during search-and-repair iterations. The router also runs postroute optimization as part of detailed routing. During postroute optimization, it runs more rigorous search and repair steps. Detailed routing stops automatically if it cannot make further progress on routing the design. The routed data is saved as part of the Innovus database.

NanoRoute Router in the Innovus Flow

The NanoRoute router is part of the block implementation and the top-level implementation stages of the Innovus flow. Run the router early in the design flow to identify and fix routability problems or avoid them altogether. You can run the router in non-timing-driven mode after the default parasitic extraction step to establish a baseline for future steps. If the design is congested or unroutable, stop and resolve problems before continuing.

Before You Begin

The NanoRoute router reads designs directly from Innovus. Before running the router, ensure your design meeting the following conditions:

- It is fully placed and the placement is legal, without any overlaps. Use the `checkPlace` command to check for overlaps.
- Power is routed. Use the `sroute` command to route power structures.

Checking Your LEF Files

You can avoid violations and save time if you ensure your LEF files are optimized for routing. Check the following statements and edit the files with a text editor if necessary:

- `MINSIZE`

  The router does not support specifying `MINSIZE` without specifying `AREA`. `MINSIZE` allows a geometry that is smaller than `AREA`.

- `UNITS`

  The router does not support a value of 100 for `DATABASE MICRONS` in the `UNITS` statement. If the LEF technology file specifies `DATABASE MICRONS 100`, run the following command before importing the design:
  `setImportMode -minDBUPerMicron 1000`

- `MANUFACTURINGGRID`

  The router requires that you define the manufacturing grid.

- `MACRO`

  To improve pin access, ensure that all standard cell macros are defined as `CLASS CORE`.

  You must use real shapes, not block-style abstracts, for the shapes on the layers where you expect the router to connect to pins of standard cell macros.

- `VIA`

  The `TOPOFSTACKONLY` keyword is unnecessary if there are LEF `LAYER AREA` statements, because the router automatically derives `TOPOFSTACKONLY` vias based on the `AREA` statements. If a default via satisfies the `AREA` statement, the router tags it internally as a `TOPOFSTACKONLY` via.

  If there is no `AREA` statement for a routing layer, the router looks for `TOPOFSTACKONLY` vias that go up to the next metal layer. If `TOPOFSTACKONLY` vias exist, it derives the `AREA` rule from those vias--the smallest area of the bottom layer metal of all such vias becomes the `AREA` rule. This feature provides backward compatibility with LEF files that do not have `AREA` rule support.

Related Topics

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 the `checkPlace` command.
- 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.
- 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.

Adding Tracks

In the Innovus environment, the router generates tracks automatically, based on the routing pitch, layer width and spacing, and minimum via widths. If you import a DEF file, run the `add_tracks` command prior to global routing to correct faulty track definitions and tune the tracks to routing.

Related Topics

For information on importing DEF files, see the Import and Export Commands chapter in the *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 `setDesignMode` parameters to specify hard layer limits:

- `-bottomRoutingLayer`
- `-topRoutingLayer`

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 *metal8* and `pin_b` is on *metal7*. The pins overlap in the X and Y direction. If you specify that the top routing layer is *metal6* , the router connects to `pin_a` by using stacked vias, creating a short with `pin_b`.

Specifying Soft Layer Limits

When you specify soft layer limits, the router attempts to route specific nets within a layer range, but might route some nets outside the layer range if necessary to complete routing without creating violations. In addition, you can specify the effort level for staying within the range. You can also route specific nets within the layer range and others outside the layer range. For example, you can route critical nets within a narrower layer range than you route the rest of the nets in order to improve timing.

Use the following `setAttribute` parameters to specify soft layer limits and set the effort level toward honoring the limits:

- `-bottom_preferred_routing_layer`
- `-top_preferred_routing_layer`
- `-preferred_routing_layer_effort`

Interrupting Routing

To interrupt routing, press Ctrl+C. The `routeDesign` or `globalDetailRoute` command continues to run until the database is in a state where the command can stop safely. When the software stops, it presents the Interrupt menu.

Warning: When you interrupt routing with Ctrl+C, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.

Related Topics

For information on the Interrupt menu, see "Interrupting the Software" in the Getting Started chapter.

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. The `routeDesign` command 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:
    - `-route_with_timing_driven true`
    - `-route_with_si_driven true`
- It changes the status of clock nets from FIXED to ROUTED so it can modify them during routing and routes them before routing other nets. Once the status of the clock nets is set to ROUTED, it does not change it back to FIXED.
    - To keep clock nets' status FIXED, run the following command before running `routeDesign`:

      ```
      setNanoRouteMode -route_fix_clock_nets true
      ```
    - To stop the router from routing clock nets first, run the following command before running `routeDesign`:

      ```
      setNanoRouteMode -route_route_clock_nets_first false
      ```
- It runs a placement check prior to routing to ensure that the placement is clean. To turn off the placement check, specify the `routeDesign -noPlacementCheck` parameter.
- 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` options for via and wire optimization are `-viaOpt` and `-wireOpt`.
    - The `setNanoRouteMode` parameters that work only with `routeDesign` are `-route_fix_clock_nets` and `-route_route_clock_nets_first`.

Related Topics

For more information, see the following commands in the Route Commands and Global Variables chapter of the *Text Command Reference*.

- `detailRoute`
- `globalDetailRoute`

- `routeDesign`
- `globalRoute`
- `setAttribute`
- `setNanoRouteMode`

Results

The NanoRoute router outputs can include the following (depending on the run-time options you set):

- Section in the Innovus log file
- Routed DEF file
- GDSII file
- SDF or SPEF file

The following reports:

- Routing statistics.

  For information, see the `reportRoute` command.
- Routing connectivity. For information, see the `verifyConnectivity` command.
- Wire statistics, including wirelength. For information, see the `reportWire` command.
- Shielding statistics
- Timing analysis
- Capacitance. For information, see RC Extraction.
- Design rule checking (DRC) and layout versus schematic (LVS)
- Process antenna violations.
- Signal integrity. For information on signal integrity reports, see Analyzing and Repairing Crosstalk.

Note: The number of nets that were not routed due to the existence of mixed signal constraints are reported in the log file by the NanoRoute Router.

Use Models

Running the NanoRoute Router with Innovus Menu Commands and Forms

Use the NanoRoute GUI forms to route the design and specify the following:

- Net attributes
- Most commonly used run-time options
- Routing type (global, detailed, or both)
- Congestion map style and options

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.
- *Route 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.

For information on route GUI, see the Route Menu chapter, in the *Menu Reference*.

Use the following forms to route the design.

- *Route 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.

For information on route GUI, see the Route Menu chapter, in the *Menu Reference*.

Running the NanoRoute Router with Innovus Text Commands

Use the following commands to set NanoRoute attributes and options, generate tracks, 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.

- `add_tracks`
  Generates optimized tracks for the router (only necessary if you import a non-native DEF file).
- `getAttribute` and `setAttribute`
  Display and set net attributes.
- `getNanoRouteMode` and `setNanoRouteMode`
  Display and set run-time options for the router.
- `globalRoute`, `detailRoute`, `ecoRoute`, `globalDetailRoute`, and `routeDesign`
  Route the design.
  Note: The recommended design flows use `routeDesign` command. For more information, see Using the routeDesign Supercommand.

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 on certain layers, they are protected by extra spacing, and signal integrity violations that affect them are repaired after routing.
- Options determine run-time operations.

For example, use options to perform the following run-time operations: run global or detailed routing, route selected nets only, repair antenna or design-rule violations, run timing driven or signal integrity driven routing, and specify the number of processors to use.

The following table lists attribute and option characteristics:

Characteristic	Attributes	Options
Application	Apply locally to a net object	Apply globally to a process or command
Specification	• *Route Attributes* form • `setAttribute` command • Some attributes can only be specified by `setAttribute`.	• *NanoRoute* form • `setNanoRouteMode` command • Some options can only be specified by `setNanoRouteMode`.
Persistence	Saved with the database. When you set an attribute and save the database and exit, the attribute setting is saved. If you reload the database, the object retains the attribute setting.	Saved with the database. When you set an option, save the database and exit, the option setting is saved. If you reload the database, the router retains the option value.
Format	*-attribute_name* • Example: `-avoid_detour` • Case sensitive (all lower case) • Mandatory underscores separate words	*-option_name* • Example: `-route_detail_auto_stop` • Case sensitive (all lower case) • Mandatory underscores separate words
See settings for this run …	Use the `getAttribute` command	Use the `getNanoRouteMode` command

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 `setAttribute -si_post_route_fix` to true for the net and route the design with the `-route_with_si_driven` option set to true.

Using text commands, issue the following commands:

```
setAttribute -net net1 -si_post_route_fix true
setNanoRouteMode -route_with_si_driven true
globalDetailRoute
```

Using the GUI, make the following selections:

- On *Route Attributes* form,
  a. Type the name of the net in the *Net Name(s)* text box.
  b. Select TRUE for *SI Post Route Fix.*

- On the NanoRoute form,
    a. Select both *Global Route* and *Detail Route*.
    b. Select *SI Driven* and then *Post Route SI*.

Accelerating Routing with Multi-Threading and Superthreading

Innovus products accelerate routing by using more than one processor in the same machine and by distributing routing to multiple machines. The NanoRoute router can use more than one processor in the same machine. This is called multi-threading. The NanoRoute detail router accelerates routing even more by distributing detailed routing across the network to multiple machines. This capability combines multi-threading with distributed processing, and is called Superthreading. When used with a gigabit Ethernet connection, Superthreading makes data communication time negligible.

Superthreading has the following features:

- Uses available Innovus licenses. No special licenses are necessary.
- Platform independent.
    - Different operating systems - including Solaris, Linux, and HP-UX - can be used in the same job.
    - Different hardware - including Sun, IBM, and HP - can be used in the same job.
    - 64-bit and 32-bit versions of the NanoRoute router can be used in the same job. For example, you can start a large job on a 64-bit server and run the job on smaller 32-bit clients.
- Can run using the `rsh` command, and with LSF, Sun Grid, or SSH configurations.
    - The RSH and SSH method tie multi-threaded jobs together.
    - The LSF and Sun Grid methods tie single jobs together.

Related Topics

- Accelerating the Design Process By Using Multiple-CPU Processing chapter in the *User Guide*
- Multiple-CPU Processing Commands chapter in the *Text Command Reference*
- *Set Multiple CPU Usage* form in the Tools Menu chapter, in the *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.
- 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 Metal8 = 4
#Total number of violations on LAYER Metal9 = 1
#Total number of violations on LAYER Metal10 = 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* GUI form. You can look at the Innovus log file to see the processes that the NanoRoute router runs and the problems it encounters. Then you can adjust the net attributes or run-time options to improve your results. The strategy presented in this section shows how you can break the routing processes into steps, so you can analyze and solve problems easily. After each step, check for data problems and congestion and make repairs. Repeat the step and repair remaining violations. Continue this process until the design is free of violations before going to the next step.

Using the Innovus Text Commands

The following commands show the basic routing strategy using the Innovus text commands.

1. The router globally routes the design:

   ```
   globalRoute
   ```

2. The router does the initial detailed routing (iteration 0 does not include a search-and-repair step), and saves the design as `droute0`:

   ```
   setNanoRouteMode -route_detail_end_iteration 0

   detailRoute

   saveDesign droute0
   ```

3. The router does the first search-and-repair iteration and saves the design for analysis:

   ```
   setNanoRouteMode -route_detail_end_iteration 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 -route_detail_end_iteration 19

   detailRoute

   saveDesign droute19
   ```

5. The router runs postroute optimization (`route_detail_end_iteration default`) and additional search-and-repair operations and saves the design as `droute`:

   ```
   setNanoRouteMode -route_detail_end_iteration default

   detailRoute

   saveDesign droute
   ```

Using the Innovus GUI

The following section describes the basic routing strategy using the GUI.

Run Global Routing

1. Choose *Route - NanoRoute - Route*.
2. Select *Global Route* on the *NanoRoute* form.
3. Click *OK* .
4. Save as `groute`.
5. Check the congestion map.

If you see congested areas after global routing, your design cannot be routed.

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 the "Evaluating Violations" section for an excerpt of a log file from a design with many violations on *metal1* and *metal2*.
For information on the LEF file, see the *LEF/DEF Language Reference*.

Run Search and Repair

Break search and repair into two phases. Check congestion after each phase and repair violations.

To run the first phase of search and repair, complete the following steps:

1. Choose *Route - NanoRoute - Route* .
2. Set the following options on the NanoRoute form:
   - *Detail Route*
   - *Start Iteration* `1`
   - *End Iteration* `1`

     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`

     In this phase, the router makes additional search-and-repair passes. It reroutes nets with violations within a local area (a switch box). In each successive pass, the size of the switch box size increases, so the router can make the repairs over larger areas.
3. Click *OK* .

4. Save the design as `droute19`.
5. Check congestion.

If you still have many violations (more than 1,000) or an unbalanced distribution of violations, you might still have a problem with the data or a congested design.

For help resolving the violations, see the "Evaluating Violations" section.

Checking Congestion

Check congestion in your design after global routing by using the Congestion Analysis Table in the Innovus log file and the congestion map in the Innovus main window.

Using the Congestion Analysis Table

The congestion analysis table shows the distribution and severity of congestion in global routing cells (gcells) on each routing layer.

Note: For information on global routing and on gcells, see Global Routing.

Following is an example of a Congestion Analysis table:

```
Congestion Analysis:
                OverCon       OverCon      OverCon      OverCon
                 #Gcell        #Gcell       #Gcell       #Gcell     %Gcell
Layer             (1-2)         (3-4)        (5-6)       (7-12)     OverCon
---------------------------------------------------------------------------
Metal 1       22(0.01%)     10(0.00%)     0(0.00%)     0(0.00%)     (0.01%)
Metal 2     5531(2.39%)   1680(0.73%)   370(0.16%)   123(0.05%)     (3.33%)
Metal 3     4114(1.78%)     19(0.01%)     0(0.00%)     0(0.00%)     (1.79%)
Metal 4     1333(0.58%)    137(0.06%)     0(0.00%)     0(0.00%)     (0.64%)
Metal 5     5852(2.53%)      4(0.00%)     0(0.00%)     0(0.00%)     (2.53%)
Metal 6       27(0.01%)      0(0.00%)     0(0.00%)     0(0.00%)     (0.01%)
---------------------------------------------------------------------------
  Total    16879(1.22%)   1850(0.13%)   370(0.03%)   123(0.01%)     (1.39%)

#Max overcon = 12 tracks.
#Total overcon = 1.39%
#Worst layer Gcell overcon rate = 2.53%
```

- The first column, `Layer`, lists the metal layers that have over-congested gcells. The NanoRoute router marks a gcells as over-congested if the global router has assigned more nets to the gcell than the gcell has available tracks.
- The second through fifth columns, labeled `OverCon #Gcell`, list the number and percentage of gcells on each layer that are over-congested.
- The numbers in parentheses after `OverCon #Gcell` indicate how many additional tracks within the gcell are needed to accommodate the global routing assignments. For example, `OverCon #Gcell  (1 - 2)` means that one or two additional tracks are needed to accommodate all the nets that the global router has assigned the gcells listed in the column. As you move from left to right in the table, congestion increases because the difference between the number of nets assigned to the gcell by the global router and number of available tracks within the gcell increases.
- The number of columns in the table is determined by the number of additional tracks needed by the gcells with the worst congestion. For example, if the most over-congested gcells need only four additional tracks, the table would include columns for 1-2 and 3-4 tracks, but not for 5-6 or more tracks.

- The NanoRoute router creates only one column for gcells that need seven or more additional tracks. In the example, all gcells that need seven to 12 additional tracks are listed in the column labelled `OverCon  #Gcell  (7 - 12)`.
- The NanoRoute router displays the maximum number of tracks needed in the last `OverCon #Gcell` column. In the example, the maximum number of tracks needed is 12. If some gcells needed 14 more tracks, the column would be labelled `OverCon #Gcell (7-14)`. If the maximum number of tracks needed were only eight, the column would be labelled `OverCon #Gcell (7-8)`.
  Within each column, the table does not indicate exactly how many additional tracks are needed. For example, in the column labelled `OverCon #Gcell (7-12)`, The NanoRoute router does not distinguish between gcells that need seven, eight, nine, ten, 11, or 12 additional tracks.
- The last column, `%Gcell OverCon`, lists the percentage of all gcells on the layer that are over-congested. In the example, on layer `Metal 1`, only 0.01% of the gcells are over-congested.
- The last row of the table, `Total`, lists the total number and percentage of over-congested gcells in each column. In the example, 1,850 gcells in the design, or 0.13% of all gcells, need three or four more tracks.
- The last row of the last column displays the overall percentage of over-congested gcells in the design. In the example, 1.39% of all cells are over-congested.
- Following the table NanoRoute summarizes a few key values. The maximum number of tracks any Gcell needed, the total over congestion number for all layers, and the worst layers Gcell congestion rate.
- The worst layer Gcell overcon rate is intended to report the routing congestion so the pin access layer or the layer below the pin access layer is not reported even if it is higher.

Interpreting the Table

- Read the table horizontally to see the distribution and percentage of gcells on each layer that have a greater demand for tracks than they have supply of tracks.
- Read the table vertically to see which layers have the most over-congested gcells and how severe the congestion is.
- The table does not show how closely the over-congested gcells are clustered. Look at the congestion map in the GUI to see clusters of congestion and their exact location.
- There is no specific number that determines whether the design is routable. In general, the more columns, and the more the percentages increase toward the right side of the table, the worse the congestion.

Using the Congestion Map

Check obstructions and congestion in your design graphically by analyzing a congestion map. The information in the map is directly extracted from the router after you run global routing. You choose the layers to display on the map. The Innovus software displays the congestion map in the main window when you complete the following steps:

1. Globally route the design.
2. Select *Physical view* in the *Views* area of the Innovus main window.
3. Click the *All Colors* button. This displays *Color Preferences* form.
4. Select the *View Only* tab.
5. Make *Congestion* viewable.
6. Select both *Horizontal Congest* and *Vertical Congest*.

For more information on selecting the objects and colors, see the The Main Window chapter in the *Innovus Menu Reference*.

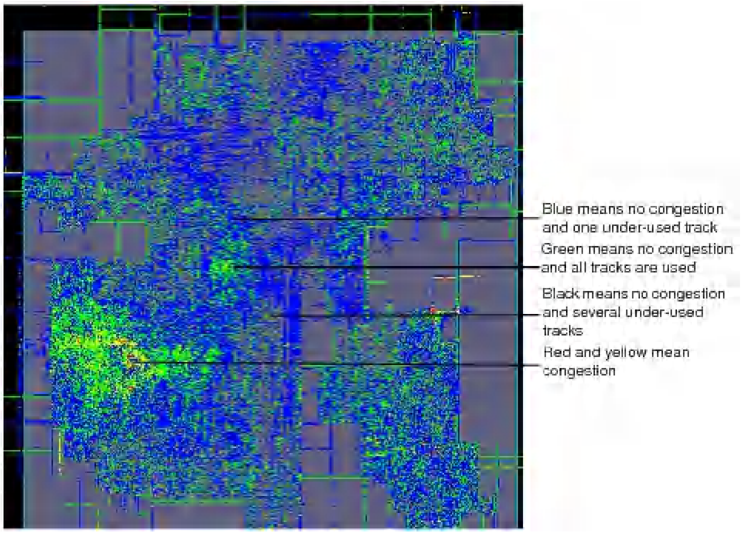
Interpreting the Congestion Map

In the map, blue or black indicate an acceptable level of congestion; white indicates an unacceptable level. However, this depends on your design. For example, a design that is mostly uncongested might have small areas (often called hot spots) that are highly congested. You must look at the overall congestion graphically to assess routability.

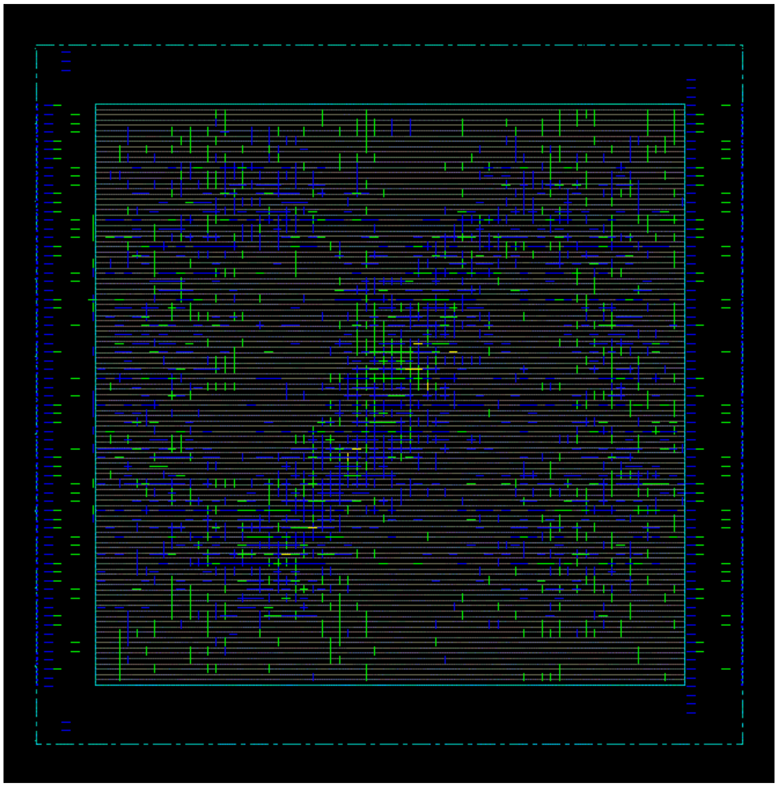
The following table explains the meaning of the default colors in the congestion map:

Color	Explanation
Black	No congestion: You have at least two tracks that are under-used.
Blue	No congestion: You probably have one track that is under-used.
Green	No congestion: All the tracks are used.
Yellow	Low congestion: You probably have one track that is over-used.
Red	Some congestion: You probably have two tracks that are over-used.
Magenta	Moderate congestion: You probably have three tracks that are over-used.
White	High congestion: You probably have at least four tracks that are over-used.

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


Blue means no congestion and one under-used track
Green means no congestion and all tracks are used
Black means no congestion and several under-used tracks
Red and yellow mean congestion

In the congestion map shown below, the design is not congested.



Resolving Open Nets

If the router cannot complete the connection of a net during routing, it generates an open net warning message in the Innovus log file and sets the net status to open. Additionally, the log file provides a list of open nets in summary format.

- During detailed routing, problems with pin modelling, routing track definitions, floorplanning, or conflicts between `setNanoRouteMode` option settings can cause open nets.
- During global routing, missing power or ground routing can cause open nets.

To resolve open net problems, complete the following steps:

1. Run `check_tracks` to diagnose a subset of open net problems in standard cells. This command generates a report in the Innovus log file. Use the report to determine the specific cause of the open net. For more information, see Diagnosing Problems Using the check_tracks Command.
2. Determine the cause of the remaining problems – mostly those caused by option conflicts or libraries – by manual analysis. For more information, see Resolving Additional Open Net Problems.
3. Resolve the problems.
4. Re-run global and detailed routing.

Log File Examples

The following examples show sections of an Innovus log file that includes five open net warning messages generated during detailed routing:

```
#Start Detail Routing.
```

```
#start initial detail routing ...
#WARNING (NR) Fail to route NET example56/cp_aclk_2 in region ( 302.295 272.894 331.695 306.495) Set net status to
open.
#WARNING (NR) Fail to route NET example56/cp_aclk_3 in region ( 302.295 272.894 331.695 306.495) Set net status to
open.
...

#start 1st optimization iteration ...
#WARNING (NR) Fail to route NET example12/cp_bclk_5 in region ( 402.295 372.894 431.695 406.495) Set net status to
open.
#WARNING (NR) Fail to route NET example12/cp_bclk_6 in region ( 402.295 372.894 431.695 406.495) Set net status to
open.
...
#start 2nd optimization iteration ...
#WARNING (NR) Fail to route NET example99/cp_cclk_8 in region ( 502.295 472.894 531.695 506.495) Set net status to
open.
...
```

The following section of the same log file includes the open net summary:

```
# number of violations = 0
#cpu time = 00:00:01, elapsed time = 00:00:01, memory = 51.15 (Mb)
#Complete Detail Routing.
#WARNING (NR) There are 5 open nets.
#Please refer to Innovus User Guide for details of open net messages and possible root causes.
#After resolving it, please re-run globalDetailRoute command.
#List of open nets :
# example56/cp_aclk_2
# example56/cp_aclk_3
# example12/cp_bclk_5
# example12/cp_bclk_6
# example99/cp_cclk_8
#
#Total wire length = 340827 um.
#Total half perimeter of net bounding box = 298122 um.
```

Diagnosing Problems Using the check_tracks Command

The `L_check_tracksS_check_tracks` command reports the following types of problems in the Innovus log file:

- Pins that are too far inside a blockage
- Pins that are not aligned with routing tracks – Align pins with routing tracks to assure the maximum number of pickup points.
- Pins that are above or underneath power stripes on the adjacent metal layer – The router might not able to access a pin if it is blocked by a power stripe.

For more information, see Macro Obstruction Statement syntax and the accompanying figures in the LEF Syntax section of *LEF/DEF Language Reference*.

Resolving Additional Open Net Problems

If the router generates an open net message after you correct the problems reported by the `check_tracks` command, or if the `check_tracks` command 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.
- Problems caused by the `setNanoRouteMode` command settings or conflicts between parameter settings and library specifications
  - No via access in pin but `setNanoRouteMode -route_with_via_only_for_stdcell_pin true` is specified
  - No via access in pin but `setDesignMode -bottomRoutingLayer` is too high or `setDesignMode -topRoutingLayer` is too low for the router to connect without using a via
  - Via stacking is not allowed but `setDesignMode -bottomRoutingLayer` is higher than the pin layer (or `setDesignMode -topRoutingLayer` 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.
```

If the router generates an open net message after you correct the problems reported by the `check_tracks` command, or if the `check_tracks` command 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 `check_place` command to check for cell overlaps.
- Problems caused by the routing attribute settings or conflicts between attribute settings and library specifications:
  - No via access in pin but `route_with_via_only_for_stdcell_pin true` is specified

- ◦ No via access in pin but `design_bottom_routing_layer` is too high or `design_top_routing_layer` is too low for the router to connect without using a via
- ◦ Via stacking is not allowed but `design_bottom_routing_layer` is higher than the pin layer (or `design_top_routing_layer` is lower than the pin layer) so via stacking is required to reach the pin

- Problems caused by missing power or ground routing
  - ◦ Missing special routes for stripes or followpins to connect tie-high or tie-low nets causes open power or ground nets during global routing.
    The global router issues open net warning messages such as the following:

```
#WARNING (NR) There is no prerouted stripe wire within routing layer range 1:9 for special net VSS.
#WARNING (NR) Please reroute special net wires before running NR.
```

Running Timing-Driven Routing

In the Innovus environment, during timing-driven routing, the router uses the Common Timing Engine (CTE) by default. All the related tasks (route estimation for the timing graph, capacitance extraction, timing analysis, timing graph generation) are executed within the Innovus environment. Timing-driven routing might cause longer run time and more violations than non timing-driven routing. For information, see Violations in Timing-Driven Routing.

Input Files

To run timing-driven routing you need the following files:

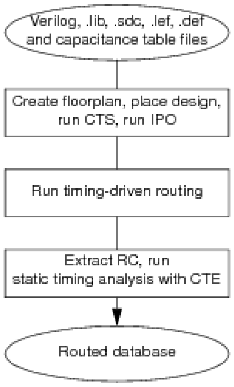
- Physical libraries in LEF
- Timing library in `.lib` format
- Timing constraints in `.sdc` format or a timing graph
- Extended capacitance table generated by the Innovus software
- Netlist in DEF or Verilog format
- Placed design in DEF

Using the CTE and the NanoRoute Router

```
setNanoRouteMode -route_with_timing_driven true
globalDetailRoute
```

Figure 1: Flow for Routing - NanoRoute in Native Mode

Native NanoRoute-CTE timing-driven routing flow


Verilog, .lib, .sdc, .lef, .def and capacitance table files
Create floorplan, place design, run CTS, run IPO
Run timing-driven routing
Extract RC, run static timing analysis with CTE
Routed database

Routing Clocks

Route clock nets before routing the rest of the signal nets. If you are using the `routeDesign` super command, the NanoRoute router changes the status of clock nets from `FIXED` to `ROUTED` so they can be moved and routes them before routing other nets.

This section gives additional information on you can use to route clocks manually.

Layer assignments for clock nets might not correlate in global and detailed routing. For tight control over clock timing, run global and detailed routing on clock nets before routing other nets. Fix the locations of the nets during detailed routing and unfix them during postroute optimization. Use net weights to ensure priority during search and repair.

Setting Attributes for Clock Nets

If clock nets are marked `USE CLOCK` in the DEF file or you have defined a clock net in the Innovus database, the router automatically sets the following values. You can change the values by setting attributes on the NanoRoute Attributes form. If the clock nets are not defined, type the name of a clock net in the *Net Name(s)* text box to set attributes for the net.

- *Weight*
- The default net weight for clock nets is `10` to give clock nets priority during global routing (the default net weight for other nets is `2`). During global routing, the router goes from global routing cell to global routing cell within each switch box, and routes the nets with the highest weight first.
- *Bottom Layer*
  The default bottom layer for routing clock nets is `3`, to ensure that the router has access to *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 upper layers, if necessary to complete the routing.
- *Avoid Detour*
  Avoid Detour is *True* for clock nets, so they are routed as straight as possible.

Set the following attribute:

```
setAttribute -preferred_extra_space 1
```

The `-preferred_extra_space` parameter adds spacing around the clock nets, which improves coupling capacitance. It is not included on the NanoRoute/Attributes form.

Tip: Select *SI Prevention True* to set *Weight, Avoid Detour* and `-preferred_extra_space` all at once. *SI Prevention True* sets *Weight* to `10` , *Avoid Detour* to `True` , and `-preferred_extra_space` to `1` for clock nets.

Routing Clock Nets Using the GUI Forms

Specify the following options on the *NanoRoute* form:

- *Selected Nets Only*
  Specify *Selected Nets Only* 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

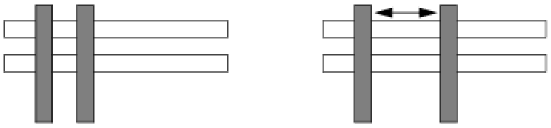
To prevent rip-up and rerouting of clock nets during postroute optimization, specify the following:

- On the *Route Attributes* form, keep the attributes you have already set, and select *TRUE* for *Skip Routing*.
- On the *NanoRoute* form, specify *End Iteration* `default`.

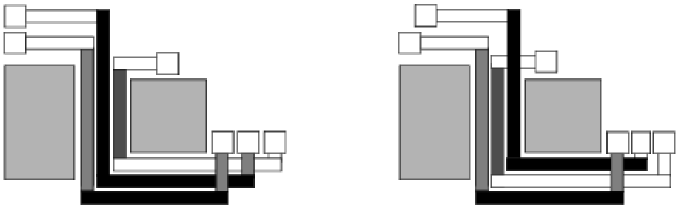
Preventing and Repairing Crosstalk Problems

During SMART routing, the NanoRoute router automatically prevents crosstalk problems by wire spacing, net ordering, minimizing the use of long parallel wires, and selecting routing layers for noise-sensitive nets. The router performs these operations concurrently with other operations. In addition to the operations it performs automatically, the router also performs shielded routing to protect critical wires from crosstalk. During postroute signal integrity repair, the router performs these same operations. The following sections describe the crosstalk prevention and repair operations the router performs, and whether you can set net attributes to control them.

- Wire Spacing
  The router automatically adds extra space between critical nets. You can also use the `setAttributeset_route_attributes` `-preferred_extra_space` attribute to add space.

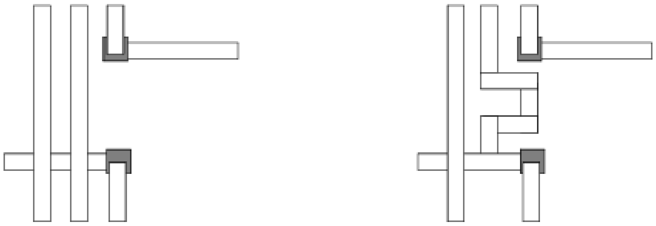


- Net Ordering
  The router automatically routes critical nets first and avoids detours on those nets so they are as short as possible.
  - You can also use the `-weight` attribute to give priority to critical nets within a switch box, so they are routed first.

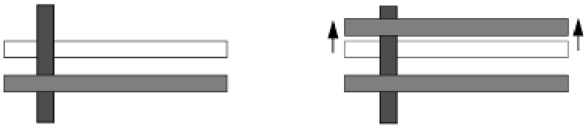


  - Minimizing the use of long parallel wires

The router automatically minimizes the use of long parallel wires, based on an internal algorithm. You cannot set an attribute to control this feature.



- Selecting routing layers
  The router automatically restricts routing layers for critical nets to reduce both coupling and resistance. It routes clocks on layers 3 and 4.
  - You can set the `-bottom_preferred_routing_layer` and `-top_preferred_routing_layer` attributes to specify preferred layers for critical nets.
  - You can specify how strictly to enforce these attributes by specifying the `-preferred_routing_layer_effort` attribute.



- Shielding
  The router can shield critical nets with power or ground wires to protect them from coupling. Shielding is not an automatic operation--you control it with the `-shield_net` attribute.

Related Topic

- Performing Shielded Routing

Crosstalk Prevention Options

To minimize problems caused by crosstalk, set the following NanoRoute options:

```
setNanoRouteMode -route_with_timing_driven true
setNanoRouteMode -route_with_si_driven true
```

These options specify timing-driven and signal integrity-driven routing and fine-tune the priorities the router assigns to timing, signal integrity, and congestion. Use these options together to minimize crosstalk. After meeting the timing requirements of your design, adjust the values and rerun routing, following these guidelines:

- If your design is congested, use a low timing-driven effort.
- If your design is not congested, use a high timing-driven effort
  Tip: Because designs with severe signal-integrity problems are usually not congested, use a high timing-driven effort for those designs.
- If increasing the timing-driven effort creates a jump in the number of timing violations, decrease the timing-driven effort.

For more information, 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 pre-route wires to guide global ECO routing. The router does not globally reroute nets that are automatically pre-routed, such as clock nets, but it might make minor routing changes to pre-routes to increase the routability of the design. Examples of minor routing changes include the following:
  - Completely moving a pre-route
  - Changing the routing topology within the current routing switch box.
- Retains fully pre-routed 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 the `globalDetailRoute` command.
- 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:

```
setNanoRouteMode -route_with_eco true
globalDetailRoute
```

Info: The `-route_with_eco` parameter 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 -route_with_eco true
setNanoRouteMode -route_detail_use_multi_cut_via_effort low
globalDetailRoute
```

For more information on using Innovus ECO commands and flows, see Interactive ECO.

Evaluating Violations

DRC Marker Name Comparison Table

During the detail routing stage, the NanoRoute router reports violations after each iteration. The NanoRoute violation summary is printed to log file by default. The following is an example of the violation table:

```
# number of DRC violations = 13975
#
# By Layer and Type:
#               MetSpc  Notch Short NdrSpc Mar EolOpp Others Totals
#    Metal1         0     0     0      0   0      0      1      1
#    Metal2         1     0     0      0   0      0      1      2
#    Metal3         1     0     0      0   0      1      0      2
#    Metal4         2     1    15      0   1      2      4     25
#    Metal5         0     0     7      0  39      1      1     48
#    Metal6         2     5    19      0   5      1      5     37
#    Metal7         1     0     0   7527   0      0      2   7530
#    Metal8         0     0     4   6322   0      0      1   6327
#    Metal9         1     0     0      0   1      0      1      3
#    Totals         8     6    45  13849  46      5     16  13975
#
# By Non-Default Rule:
#    Rule                      Metal1 Metal2 Metal3 Metal4 Metal5 Metal6 Metal7 Metal8 Totals
#    DBLCUT_DBLSPACE_RULE  +S   0      0      0      0      0      0      8396   6617   15013
#    Totals                     0      0      0      0      0      0      8396   6617   15013
```

The violation table is *Per-Layer* and *Per-Rule* violation count. Hence, `Metal7 NdrSpc 7527` means there are 7527 NDR spacing violations on Metal7 layer, and such NDR rules are defined as hard rules(with HARDSPACING). The Non-Default Rule table is *Per-NDR-Rule* and *Per-Layer* VIOLATED-NDR-NET count.

Hence, `DBLCUT_DBLSPACE_RULE 8296 Metal7` means there are 8296 NDR net violation locations.

More precisely,

- If one NdrSpc violation is between a NDR net and a regular rule net, this counts for 1 in NDR table.
- If one NdrSpc violation is between a NDR net and another NDR net, this counts for 2 in NDR table.

Note: +SW means the spacing/width defined with non-default value in NDR rule for at least one layer.
+S means the spacing defined with non-default value in NDR rule for at least one layer.
+W means the width defined with non-default value in NDR rule for at least one layer.

The following table describes the DRC violations with abbreviations:

Short Name	Marker Name
AdjCSM	Same Mask Adjacent Cut Spacing
AdjCut	Adjacent Cut Spacing
Ant	Antenna
AntAR	Antenna Area Ratio
AntCAR	Antenna Cumulative Area Ratio
AntCSA	Antenna Cumulative Side Area Ratio
AntSAR	Antenna Side Area Ratio
ArSpac	Metal Area Spacing
BlockM	Self-Aligned Double Patterning Block Mask Violation
C2CCol	Same Layer Same Mask Cut Spacing
C2MCon	Cut To Metal Concave Corner Spacing
C2MCvx	Cut To Metal ConvexCorner Spacing
C2MO	Cut To Metal Orthogonal Spacing
C2MSpc	Cut To Metal Spacing
C2MWW	Cut to Wrong Way Metal Spacing
ColChg	Metal Color Change
ColCrSp	Corner Spacing Same Mask
Color	SameMask Spacing
CorFil	Corner Fill Spacing
CorSpc	Corner Spacing
CShort	Cut Short
CutCtr	Cut On CenterLine
CutEol	Cut EolSpacing
CutFbd	Cut Forbidden Spacing
CutInr	Different Layer Cut Spacing
CutOrt	Cut Orthogonal Spacing
CutSpc	Same Layer Cut Spacing
Detour	Wire detours
DSLCol	Directional SpanLength SameMask Spacing
DSLSpc	Directional SpanLength Spacing

Enc	Enclosure
Enc2Jt	EnclosureToJoint
EncEdg	EnclosureEdge
EncEO	EnclosureEdge Opposite
EncPrl	Enclosure Parallel
EncSpc	Enclosure Spacing
EOLColor	End Of Line Color Spacing
EolExCol	EolExtension Spacing
EolExt	EolExtension Spacing
EolKO	EndOfLine Keepout
EolOpp	OppositeEol Spacing
EOLSpc	End Of Line Spacing
FbdSp	Forbidden Spacing
FeedTh	Feedthrough On Pin
FltPatch	Floating Patch
HVGeo	Pin Access
InfSpc	Influence Spacing
IsoCut	Isolated Cut
JCSCol	Joint Corner Spacing SameMask
JCSpc	Joint Corner Spacing
Litho	Litho
Loop	Metal Loop
LthBDG	Litho Bridging
LthEOL	Litho EndOfLine
LthNCK	Litho Necking
Mar	Minimum Area
MaxStk	MaxViaStack
MaxWid	Maximum Width
MetSpc	ParallelRunLength Spacing
MinCut	Minimum Cut
MinEnc	Minimum Enclosed Area (Min Hole)

MinStp	MinStep
MinWid	Minimum Width
MtMCut	Minimum Cut
NdrSpc	Non Default Rule Spacing
Notch	Notch Spacing
NSMet	Non-sufficient Metal Overlap
OffGrd	Off Grid or Wrong Way
Open	Physical Open
PinAcc	Pin Access Constraint
Protru	Protrusion
Prvent	Preventive Violation
Rect	Rect Only
RtHalo	Routint Halo wrong way
SharedE	Same-Metal-Share-Edge Spacing
ShEdgH	Same-Metal-Share-Edge Spacing
ShEdgV	Same-Metal-Share-Edge Spacing
Short	Metal Short
SLTbl	Span Length Table
SpacH	Horizontal Spacing
SpacV	Vertical Spacing
SpnSpc	ParallelSpanLength Spacing
V_EOL	EndOfLine Spacing
V_NSM	Non-sufficient Metal Overlap
V_VirG	Pin Access
V_WMJ	JogToJog Spacing
Via	Via
ViaClu	Via Cluster Violation
ViaEol	EndOfLine Spacing
ViaMSt	Minimum Step
ViaWmj	JogToJog Spacing
VMinST	Minimum step

VNotch	Notch Spacing
VolSpc	Voltage Spacing
WidTbl	WidthTable
WireFuse	Fuse On Wire
WMJ	JogToJog Spacing
WreExt	Wire Extension

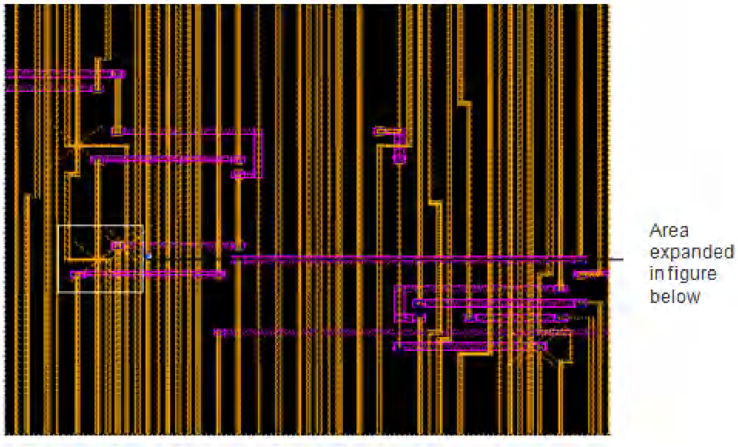
Violations on Upper Metal Layers

Upper layers are typically used to route on top of macros where only a few routing layers are allowed. These upper layers typically have larger vias than lower layers. When the routing pitch is not set at line-to-via distance, two types of violations are likely to occur:

- Via-to-wire violations
- Shorts

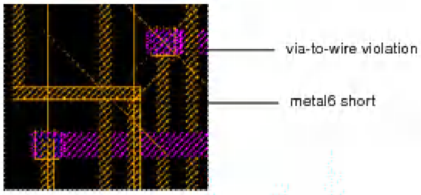
Figure 3, Figure 4, and the LEF and DEF file excerpts that follow show a design with many violations on *metal6* .

Figure 3: Design with Violations



Area expanded in figure below

Figure 4: Violations on Metal 6



The relevant LEF file excerpt is:

```
LAYER Metal6
  TYPE ROUTING ;
  PITCH 0.46 ;
  WIDTH 0.2 ;
  SPACING 0.21 `
  DIRECTION VERTICAL ;
END Metal6
LAYER Metal7
  TYPE ROUTING ;
  PITCH 0.82 ;
  WIDTH 0.4
  SPACING 42 ;
  DIRECTION HORIZONTAL ;
END Metal7
VIA via6 DEFAULT
  LAYER Metal6 ;
  RECT -0.19 -0.23 0.19 0.23 ;
  LAYER Via6 ;
  RECT -0.18 -0.18 0.18 0.18 ;
  LAYER Metal7 ;
  RECT 0.29 -0.2 0.29 0.2 ;
  RESISTANCE 0.68
END via6
```

The relevant DEF file excerpt is:

```
TRACKS X -4749270 D0 6324 STEP 460 LAYER Metal6
```

To repair the shorts and via-to-wire violations, align the tracks as much as possible without sacrificing them. Change the `TRACKS` statement in the DEF file to have at least line-to-via `STEP` (pitch).

The line-to-via calculation for *metal6* is:

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

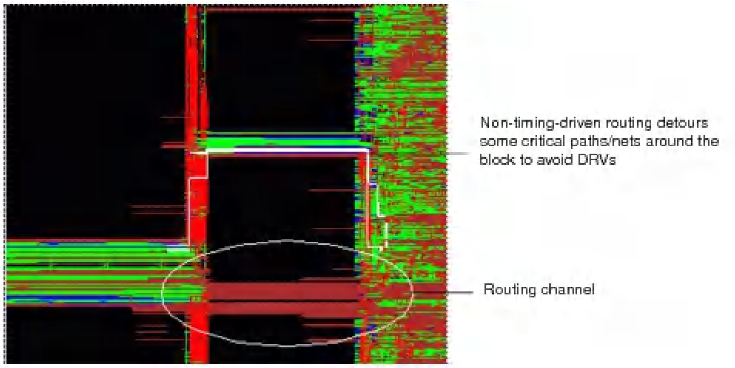
Violations in Timing-Driven Routing

Run time and the number of violations often increase during timing-driven routing because the router restricts the routing of timing-critical nets. During non-timing-driven routing, the router might detour some nets in order to avoid creating violations. In timing-driven mode, however, the router does not detour timing-critical nets. Instead, it forces them to be routed as short as possible, which can create congestion in the channels. Later, when design-rule checking takes precedence, the router detours timing-critical nets in overly congested channels.

For information on the timing-driven routing flow, see Running Timing-Driven Routing.

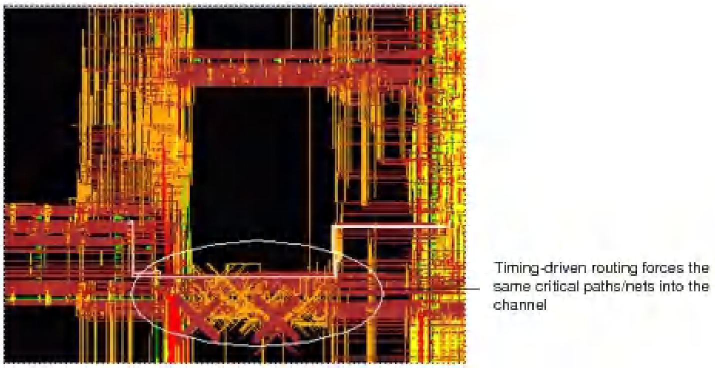
Figure 5 and Figure 6 illustrate non-timing-driven and timing-driven routing results for the same design.

Figure 5: Non-Timing Driven Routing Results



Non-timing-driven routing detours some critical paths/nets around the block to avoid DRVs
Routing channel

Figure 6: Timing Driven Routing Results


Timing-driven routing forces the same critical paths/nets into the channel

Deleting Violated Nets

To delete violated nets, use the `-regular_wire_with_drc` parameter of the `editDelete` 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 -route_with_eco`

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 the Identifying and Viewing Violations chapter of the *Innovus User Guide*.
- For information on process antenna options, see Repairing Process Antenna Violations.
- For information on wire editing, see Editing Wires.

Core Congestion

Ensure that blocks are placed in corners and near boundaries to help ease core congestion.

Concurrent Routing and Multi-Cut Via Insertion

The NanoRoute router can insert multiple-cut vias during detailed routing in order to achieve a high ratio of multiple-cut to single-cut vias, minimize the number of vias in the design, and increase yield. To specify the effort level for inserting multiple-cut vias and route the design concurrently, run the following commands:

```
setNanoRouteMode -route_detail_use_multi_cut_via_effort {medium | high}
```

`detailRoute`

Postroute Via Optimization

The NanoRoute router can optimize vias on a fully routed design by replacing single-cut vias with multiple-cut vias or with fat vias (single or multi-cut vias with an extended metal overhang). The router does not replace multiple-cut vias during this step. The router replaces vias by substituting vias in the following order:

1. Fat double-cut vias
2. Normal double-cut vias
3. Fat single-cut vias

Ensure the following before replacing the vias:

- Double-cut vias and fat vias are automatically generated or defined in the LEF file.
- The design is completely global and detailed routed. If you delete any wires after routing, reroute the design before replacing the vias.
- The design is free of all DRC violations.

Complete the following steps:

1. To run postroute via reduction, type the following:
   - `setNanoRouteMode -route_detail_min_slack_for_opt_wire slack`
   - `setNanoRouteMode -route_concurrent_minimize_via_count_effort` *`value`*
   - `routeDesign -viaOpt`

   Note: When you run these commands, the software optimizes and reduces the via count.
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 -route_detail_min_slack_for_opt_wire slack`
      - `setNanoRouteMode -route_detail_post_route_swap_via multiCut`
      - `routeDesign -viaOpt`

   b. To run non-timing-driven postroute single-cut via to multiple-cut via swapping, type the following:
      - `setNanoRouteMode -route_with_timing_driven false`
      - `setNanoRouteMode -route_detail_post_route_swap_via multiCut`
      - `routeDesign -viaOpt`
3. To raise some via priority in multiple-cut via swapping, type the following:
   - `setNanoRouteMode -route_with_timing_driven false`
   - `setNanoRouteMode -route_via_weight "`*`viaName viaWeight`*`"`
   - `setNanoRouteMode -route_detail_post_route_swap_via multiCut`
   - `routeDesign -viaOpt`

Optimizing Vias in Selected Nets

To optimize vias in selected nets, set the `setAttribute -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

The following `setNanoRouteMode` parameters can be used for via optimization:

- `-route_detail_post_route_swap_via`
- `-route_via_weight`
- `-route_detail_use_multi_cut_via_effort`
- `-route_concurrent_minimize_via_count_effort`

To control the router to choose vias with the largest overhang first, specify the following option with higher *`viaWeight`* than the other vias:
`setNanoRouteMode -route_via_weight {`*`viaName viaWeight`*`}`

Note: You can specify the priority for any via by using the `-route_via_weight` parameter, not just the largest overhang vias.

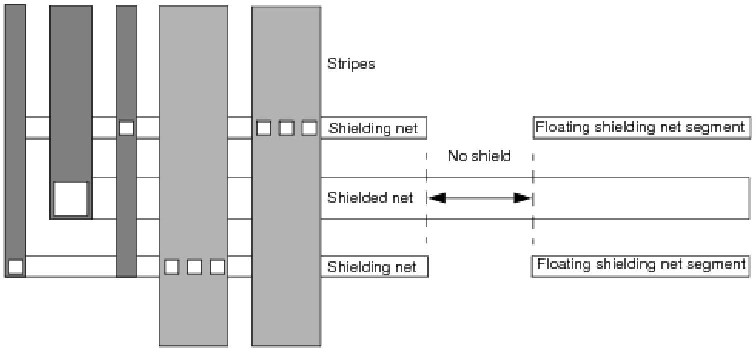
Performing Shielded Routing

The NanoRoute router can protect noise-sensitive nets, such as clock nets, from crosstalk by shielding them with power or ground wires. NanoRoute automatically generates a shielding and statistics report after routing. For information on the report, see the "Interpreting the Shielding Report" section.

The Figure below shows a section of a design with a shielded net. In the figure, you can see the following:

- The signal net is shielded by a power net on one side and a ground net on the other side.
- Multiple vias can be dropped where a stripe crosses the shielding net at a right angle, if the stripe is wide enough to accommodate them.
- A segment of the signal net is not shielded.
- There are some floating shielding net segments.

Figure: Shielded Routing


Stripes
Shielding net
Floating shielding net segment
No shield
Shielded net
Shielding net
Floating shielding net segment

Shielding Option

You can add more vias on the shielding wire for power stripe connection using the `editPowerVia` command after using the `createShield` command.

```
editPowerVia
-add_vias 1
-skip_via_on_wire_status {[routed][fixed] [cover] [shield]}
-skip_via_on_wire_shape {[Blockring] [Stripe][Followpin] [Corewire] [Blockwire] [Iowire] [Padring] [Ring] [Fillwire]
[Noshape]}
```

Note: Some vias might be removed by NR during ECO. Use the `editPowerVia` command each time the shielding is recreated.

Performing Shielded Routing Using the GUI

1. From the main menu, choose *Route - NanoRoute - Specify Attribute* . This opens the *Route Attributes* form.
2. On the NanoRoute/Attributes form, enter the name of the net to shield (this is the shielded net in the figure) in the *Net Name(s)* field.
3. Enter the name of the power ground net (or both) in the *Shield Net(s)* field. These are the shielding nets in the figure.
    - To shield both sides with ground wires, enter the name of the ground net.
    - To shield one side with a ground wire and one side with a power wire, enter both the ground and the power net names.
4. Click *OK* or *Apply* .
5. Use the Innovus `selectNet` command to specify the net to shield. It must be the same as the net you specified on the NanoRoute/Attributes form.
6. From the main menu, choose *Route - NanoRoute - Route* . This opens the *NanoRoute* form.
7. On the *NanoRoute* form, select the following:
    - In the *Job Control* area, select *Selected Nets* .
    - In the *Mode* area select both *Global Route* and *Detail Route* .
8. Click *OK* or *Apply*.

To route the remaining nets, complete the following steps:

1. On the NanoRoute/Attributes form, set the *Skip Routing True* for the shielded nets.
   Tip: You can also skip routing on prerouted nets by issuing the following command:
   ```
   setAttribute -net @PREROUTED -skip_routing true
   @PREROUTED applies to a net that has any wiring, including partial wiring.
   ```
2. On the NanoRoute form, deselect *Selected Nets Only* .
3. Click *OK* or *Apply* to reroute the design.

Performing Shielded Routing Using Text Commands

- The following command shields `net1` with a ground wire:
  ```
  setAttribute -net net1 -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
```

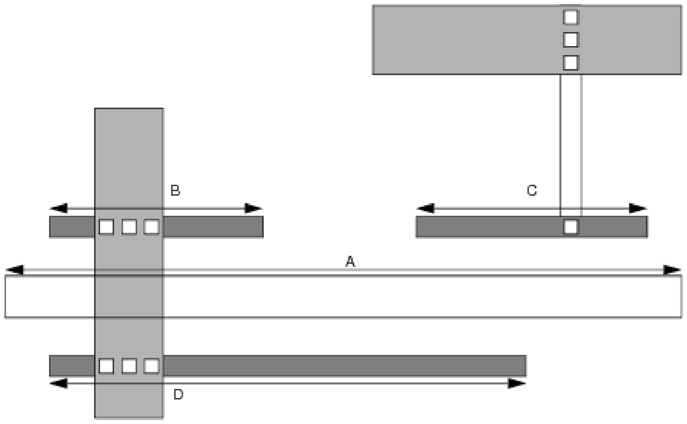
Interpreting the Shielding Report

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:

$$\text{Shielding Ratio} = \frac{B + C + D}{A \times 2}$$

Reporting the Tapping Information for Shielded Nets

The software generates a shielding report for nets that fail the tapping check. The tapping information is printed in a report file with the name, `<design_name>_shield.rpt`. This file is generated by default when you run the `reportShield` command. The report lists the required tapping distance, the names of the shielded nets for which the check is being performed, the names of the violating layers, and the total number of missing tapping points. The report syntax and example are provided below.

```
--------------------------------------------------------------------------
Required tapping distance: <micron>
-------------------------------------------------------------------------

Shielded Nets :  <Netname>
Violating Wire : <layer name> <shape box coordinates> :  <micron>
Violating Wire : <layer name> <shape box coordinates> :  <micron>

Shielded Nets :  <Netname>
Violating Wire : <layer name> <shape box coordinates> :  <micron>
Violating Wire : <layer name> <shape box coordinates> :  floating

-------------------------------------------------------------------------
Total number of missing tapping points: <number>
-------------------------------------------------------------------------
```

A sample report is provided below:

```
-------------------------------------------------------------------------
Required tapping distance: 100.000
-------------------------------------------------------------------------

Shielded Nets: Net1
Violating Wire : Metal8 81.462 618.222 82.462 619.222 :  194.480
Violating Wire : Metal7 728.080 1.200 782.462 9.222 :  150.200

Shielded Nets: Net2
Violating Wire : Metal8 81.462 618.222 82.462 619.222 :  194.480
Violating Wire : Metal7 728.080 1.200 782.462 9.222 :  floating
-------------------------------------------------------------------------
Total number of missing tapping points = 728
-------------------------------------------------------------------------
```

Routing Wide Wires

The NanoRoute router automatically tapers wide wires when connecting to pins, including input/output pins of standard cells, macro cells, and block output pins. The tapered portion of a wire uses the minimum-width wire (the default width). If you do not want tapering at the output pins, specify the following:

```
setNanoRouteMode -route_detail_no_taper_on_output_pin true
```

Note: By default, the NanoRoute router prohibits tapering on top level pins.

Using Non-Default Rules

By default, the NanoRoute router treats non-default rule spacing as a soft option; that is, when routing resources are available, it honors the non-default rule. If the area is too congested, and resources are not available, the router might not honor the rule. If you enable signal-integrity driven routing, the router attempts to minimize overall coupling capacitance in the design. If you enable timing-driven routing, the router also favors critical nets when choosing spacing. You can use up to 254 nondefault rules. Nondefault rules do not necessarily decrease the routing speed. Routing speed does decrease, however, due to the following factors:

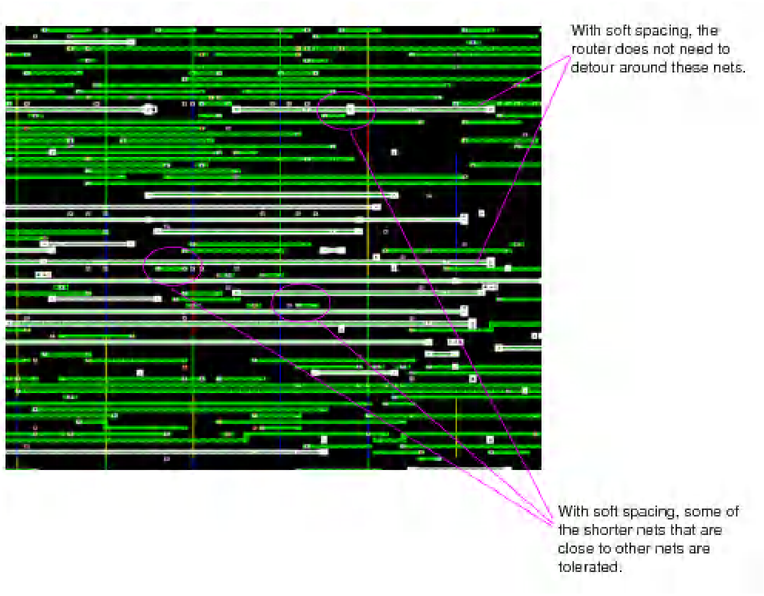
- The ratio of non-default rule wires to default rule wires increases.
- The amount of space between wires increases.
- The number of additional nondefault vias increases, due to the nondefault rules.

In congested areas, the router might violate nondefault spacing rules in order to avoid design-rule violations and complete the routing. Its flexibility with regard to nondefault spacing decreases the overall wirelength and benefits timing and signal integrity because it allows some shorter nets to be more easily tolerated near adjacent nets without causing violations.

Note: You can force the router to honor the nondefault rules by specifying the following option: `setNanoRouteMode -route_strict_honor_route_rule true`

Figure 8 illustrates nondefault spacing ("soft spacing") routing.

Figure 8: Non-Default Spacing Routing


With soft spacing, the router does not need to detour around these nets.
With soft spacing, some of the shorter nets that are close to other nets are tolerated.

Repairing Process Antenna Violations

The NanoRoute router can repair process antenna violations concurrently with DRC violations during the search-and-repair step. During the postroute optimization step, when there are no more DRC violations, the router repairs additional process antenna violations. This two-step methodology allows the router to use more aggressive methods to repair the process antenna violations early on and saves CPU time. During postroute optimization, the router repairs antenna violations by changing layers (also called antenna stapling or layer hopping). It also repairs process antenna violations by inserting diode cells as close as possible to input gates to discharge current, and deleting and rerouting nets with violations.

Note: After routing, run the `globalNetConnect` command to ensure connectivity to power and ground pins in antenna cells added during process antenna repair.

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 chapter of *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 `route_antenna_cell_name` option on the text command line.

Deleting and Rerouting Nets with Violations

If the design has more than 100 DRC violations, and you are using LEF 5.4 or later, the router deletes and reroutes nets with process antenna violations.

Repairing Violations on Cut Layers

The NanoRoute router detects antenna violations on cut layers and repairs them by inserting diodes. To repair these violations, you must specify a value in your LEF file for the `ANTENNADIFFAREARATIO` (or `ANTENNACUMDIFFAREARATIO`) for the cut layers. For each cut layer, the value for `ANTENNADIFFAREARATIO` (or `ANTENNACUMDIFFAREARATIO`) must be larger than the value for `ANTENNAAREARATIO` (or `ANTENNACUMAREARATIO`).

Info: If you do not use diodes to repair process antenna violations, the router cannot repair the violations on cut layers.

Tips:

- To highlight the diodes that the router inserts, choose *Edit - Select by Name.*
- To highlight the diodes, type *_antenna_* .
- To specify the diode cells, use `setNanoRouteMode -route_antenna_cell_name`.
- To force the router to do more layer changing and skip diode insertion, specify `setNanoRouteMode -route_antenna_diode_insertion false`

  After the router repairs as many violations as possible by layer changing, reset this option to `true` and repeat process antenna repair.

Process Antenna Options

Use the following options to repair violations caused by process antennas:

- `setNanoRouteMode` options:
  - `-route_detail_fix_antenna`
  - `-route_antenna_cell_name`
  - `-route_ignore_antenna_top_cell_pin`
  - `-route_antenna_diode_insertion`
  - `-route_diode_insertion_for_clock_nets`
- `setAttribute -net` *`netName`* `-skip_antenna_fix`

Examples

- The following commands show the basic strategy for repairing process antenna violations:

  ```
  setNanoRouteMode -route_detail_fix_antenna true
  setNanoRouteMode -route_antenna_cell_name "ANTENNA"
  setNanoRouteMode -route_antenna_diode_insertion true
  globalDetailRoute
  globalNetConnect
  ```

  The NanoRoute router runs global and detailed routing. After repairing DRC violations, it repairs as many process antenna violations as it can by layer hopping during postroute optimization. If any process antenna violations remain, the router repairs them by inserting antenna diode cells named ANTENNA.

- The following commands repair process antenna violations by inserting diodes and filler cells. The filler cells are specified by the `setFillerMode -core` command. They fill any gaps that are left when a diode replaces a large filler cell.

  ```
  setNanoRouteMode -route_antenna_diode_insertion true
  globalDetailRoute
  globalNetConnect
  ```

For information on adding filler cells, see the `setFillerMode` and `addFiller` commands.

Creating RC Model Data in TQuantus Model File

You can create and store RC model data in a TQuantus model file, which is generated by the TQuantus extraction engine. The TQuantus extraction engine is an advanced extraction engine that is enabled by default for postRoute effort level medium extraction. The TQuantus engine is tightly integrated with NanoRoute and drives track assignment-based timing and SI optimization/postRoute optimization, and timing-driven routing.

The TQuantus model file generated using this engine has a benefit over the Quantus techfile in that it provides a simplified RC model data for implementation as compared to the Quantus techfile that has a complicated RC table. However, there is a small tradeoff for accuracy and runtime. You can create the TQuantus model file by using the `createTQuantusModelFile` command inside the Innovus environment.

Before creating the TQuantus model file, ensure that the following are available in the Innovus database:

- QRC techfiles
- Pitch information from the technology LEF file

- RC corners

Use model for TQuantus Model File

The use model for the creation and usage of TQuantus model file is detailed below:

1. Generate the TQuantus model file using the `createTQuantusModelFile -file` command.
2. Once the TQuantus model file is available, run the following command to bring it into the Innovus environment:
   `setExtractRCMode -engine postRoute -effortLevel medium -tQuantusModelFile tQuantus_model_file`
3. Use the `checkTQuantusModelFile` command to check the consistency of the TQuantus model file against the various input files listed above inside the Innovus database. The following commands automatically invoke the `checktQuantusModelFile` command to check the TQuantus file, if TQuantus is chosen as the extraction engine.
   a. `routeDesign -trackOpt`
   b. `timeDesign`
   c. `optDesign -postRoute`
   d. `extractRC`

   Note: The software issues warnings if inconsistencies are detected in the RC corner information, QRC techfile, or pitch information in the LEF technology file. If such warnings are issued, the TQuantus model file will be generated automatically.

To use the TQuantus model file during track assignment-based timing and SI optimization (TA-Opt) and `optDesign -postRoute` optimization, specify the `-trackOpt` option of the `routeDesign` command. Specifying this parameter ensures that TQuantus model file is used during `timeDesign` and `optDesign -postRoute` optimization.

```
route_design -track_opt
```

Note: Specify the `-idealClock` parameter along with the `-trackOpt` parameter to enable the ideal clock mode for optimization.

```
routeDesign -trackOpt -idealClock
```

Note: The TQuantus flow is enabled by default. In this mode, the software instructs `timeDesign` and `optDesign -postRoute` to use the TQuantusModel file for optimization. You can specify a file name for the TQuantus model file by using the `-tQuantusModelFile` parameter of the setExtractRCMode command. If this file is not specified, it is generated automatically by the software.

Example1: Use Model for Track Assignment-Based Timing and SI Optimization

The example below details the use model for TA-Opt:

```
createTQuantusModelFile -file tQuantus_model.bin
setSIMode -reset
setExtractRCMode -engine postRoute -effortLevel medium -tQuantusModelFile tQuantus_model.bin
setDelayCalMode -SIAware true
setAnalysisMode -analysisType onChipVariation -cppr both
routeDesign -trackOpt
```

Note: When `routeDesign -trackOpt` is specified, the `setDelayCalMode -SIAware` parameter must be set to `true`. If it is set to `false`, the software will error out. Also, you can enable the ideal clock mode for optimization by specifying the `-idealClock` parameter along with the `-trackOpt` parameter.

Example2: Use Model for optDesign -postRoute and timeDesign –postRoute Optimization

The example below details the use model for `optDesign -postRoute` and `timeDesign -postRoute` optimization:

```
createTQuantusModelFile -file tQuantus_model.bin
setSIMode -reset
setExtractRCMode -engine postRoute -effortLevel medium -tQuantusModelFile tQuantus_model.bin
setDelayCalMode -SIAware true
setAnalysisMode -analysisType onChipVariation -cppr both
optDesign -postRoute/timeDesign -postRoute
```

Support for High Frequency Routing

The NanoRoute High Frequency Router (NRHF) has been implemented to address the custom and high frequency routing needs of mixed-signal users. It is built on NanoRoute infrastructure for full interoperability. This capability enables routing of high frequency digital signals but requires special constraints to achieve critical performance. Before using the NanoRoute high frequency router, you must make sure that the design has routing constraints. The routing constraints can be specified though any of the following:

- *Integration Constraint Editor* form in Innovus
- *Constraint Manager* in Virtuoso
  Note: For constraints entered in Virtuoso, OA based flow should be used. The multiple entry mechanism for routing constraints is supported to take additional benefit of being fully interoperable between Innovus and Virtuoso.
- `setIntegRouteConstraint` command

The NanoRoute high frequency router can be invoked before or after placement of standard cells and blocks. All nets which have the constraints will be routed by default. To route specific nets you can use the `setNanoRouteMode -route_selected_net_only true` command.
Note: Partially routed nets will not be routed.

The following `routeDesign` command parameter supports high frequency routing:

- `-highFrequency`
  Runs high frequency routing. It routes high frequency digital signals to achieve critical performance.

The following `setNanoRouteMode` command parameters support high frequency routing:

- `-route_high_freq_constraint_groups {order net match bus pair shield}`
  Routes only the specified constraint groups. The constraint groups can be `net`, `match`, `bus`, `pair`, or `shield`.
  Note: You can add "order" in front to control the route order. If order is present, the NanoRoute high frequency router will honor the order, if not, the router will determine the order.
- `-route_high_freq_match_report_file` *`filename`*
  Specifies the name of the match report file.
- `-route_high_freq_num_reserved_layers` *`value`*
  Specifies the number of metal layers to reserve for high frequency routing above the standard cell area so that routing has more resources to access standard cell pins.
- `-route_high_freq_remove_floating_shield {true|false}`
  Removes floating shield segments for high frequency nets.
- `-route_high_freq_search_repair {true|false}`
  Runs search and repair to remove violations.
- `-route_high_freq_search_repair` *`value`*
  Specifies the shielding wire length. All shielding wires longer than the specified value are preserved by NanoRoute for high frequency nets.

Additionally, the NRHF router honors the `setAttribute -ndr_si_length_limit` settings. The `-ndr_si_length_limit` parameter specifies the table of parallel run length on each layer. It relaxes a hard NDR into a soft NDR whenever the parallel run length between the NDR wire and another wire is smaller than the specified value. Honoring this attribute enables NRHF router to ignore the NDR for a portion of the route which is a percentage of the total wire length for areas where the number of tracks are limited. With this, NRHF can maintain default spacing for the short PG wires and use routing resources more effectively.

Using the Third-party ECO Flow

Innovus supports using third-party data for ECO routing and layer only ECO routing. It supports non-DEF connectivity models, multiple NDRs per net, and other third-party structures in order to enable ECO routing on non-native data. You can enable the third-party support using the `setNanoRouteMode -route_third_party_data true` command.

Note: Since the third-party data uses DEF data types for different purposes and is formed for a completely different use model, it is important to translate it into Cadence types. With proper setup Innovus accepts third-party data for ECO Routing. Performing the ECO and resolving the remaining violation functions much same as with standard Innovus data and is compatible with Cadence flows.

The recommended steps for importing third-party data into Innovus for ECO routing or post-mask layer only ECO routing are:

- Set the third-party option
- Read in the DEF file while preserving the non-native DEF connectivity shapes
- Convert the data to Innovus forms

Sample TCL Script

The following is a sample TCL script that outlines this process:

```
# Enable the third-party options #
setNanoRouteMode -route_third_party_data true
setMultiCpuUsage -localCpu 8

# Load the data #
set init_lef_file {
  lef/tech.lef
  lef/cells.lef
  lef/macros.lef
}

set init_verilog net/design.v
init_design

# Read the DEF file while preserving the non DEF connected shapes
defIn -preserveShape design.def

# Convert special nets to regular nets with auto NDR generation
prepareForEcoRoute

# Introduce ECO, ecoDefIn, optDesign, add/remove term etc.
ecoRoute -modifyOnlyLayers 1:4
```

Setup Considerations

Conversion

The conversion step changes DEF special nets to regular nets wires so that ECO routing can be performed. Since all special nets are not required for ECO, you can mark any special net segment that may be excluded from the conversion as +FIXED before converting the data for ECO. This may include power and ground nets secondary powers and clocking mesh structures. Clock mesh would be detected as a loop and would be subjected to ecoRoute changes if not marked FIXED.

Note: Cadence recommends that you perform this process carefully as the lower layers of the clock may route from the mesh to the pins and may be changed if included in an ECO. You must focus on the intent and scope of the ECO and the design to determine which structures need to remain special nets and which can be converted to regular wires for ECO.

Note: NanoRoute will not change any segment marked as a special net.

Example:
The following commands set the state of Snet power_ mesh in Metal6-Metal7 to FIXED.

```
editSelect -net power_mesh -via_cell {*67*}
editSelect -object_type {Wire Via} -type Special -net power_mesh -layer {Metal6 Metal7}
setSelectedStripBoxState all FIXED
```

Tech.lef

If you are using the Innovus System for the first time, you may face issues with the technology file and the design setup that may require investigation.

- Load the design
- Verify the DRC Violations

The `ecoRoute` command attempts to fix any DRC violations so investigating any violation is important. The following are some potential issues and some useful tips that enable you to fix them:

- For Post-Mask ECO, the design used should be tape-out clean and should not have any DRC violations in Innovus. If Innovus reports violations, it indicates that the Innovus tech.lef is a different version than the one used to tape out the design.
- DRC violations at this stage can also indicate a rule interpretation discrepancy and require support for your Cadence representative. You must make sure that the rule sets match. There may be differences in the Innovus Tech.LEF file and the tape-out rule set that need to be fixed. Investigate any violations to align the status before proceeding with ECO routing.
- Verify the recommended rules that may have been optionally disabled in the original tape out routing. The design rules must match and show 0 violations before proceeding with ECO routing.

Floorplan

Check the floorplan objects like blockages, cell rows, and tracking. The cells need matching rows for the placer to work with the design and for diode cells and fillers to be replaced. The tracking should align with the routing for the `ecoRoute` command to interact correctly with the existing routing and cell pin access.

Note: Partial routing blockages are supported in ECO routing.

Large number of violations in one area may indicate a blockage that should be removed. The third-party routing blockages are generally more tool specific and should be removed.

Troubleshooting

If you have problems with your design, try the following troubleshooting tips:

- Check the log file for errors and warnings. Correct the problems and continue routing or reroute, as appropriate. For example, the router might stop routing automatically if it finds too many violations. If the router stops unexpectedly, check the log file for a message that the router has reached the maximum number of violations and then set the `setNanoRouteMode -route_detail_auto_stop` parameter to `false` to continue routing.
- Verify connectivity and geometry before and after routing and compare results. You can also use the `verifyConnectivity` command to check the connectivity.
- Save the database after routing and restore it in a new session in the Innovus software. Saving and restoring the database

clears temporary data structures in memory.

- 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

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, the `verifyMetalDensity` and `verifyCutDensity` commands enable you to verify that the metal density of the metal and cut layers is within the minimum and maximum density values specified by the LEF file or the `setMetalFill` and `setViaFill` commands, respectively.

The software uses parameters specified in the LEF file or the fill commands to analyze the density and determine the size and position of the fill. It divides the design into windows and adds metal or cuts to the open areas in each window until the metal and cut densities meet the density requirements.

You can add the fill to one or more layers at both the chip and block levels.

If you perform additional routing after inserting the fill, you can trim away fill that causes DRC violations.

Before You Begin

- Complete detailed routing.

To make sure the metal fill is viewable, select the following options in the main window:

- Floorplan or Physical view
- *Special Net* visibility toggle
- *Metal Fill* visibility toggle - To view the *Metal Fill* visibility toggle, click the *All Colors* button in the *Layer Control* window.

For more information on setting object visibility, see *The Main Window* chapter in the *Innovus Menu Reference* .

Adding Metal Fill in the Multiple-CPU Processing Mode

You can add the metal fill to the design in the multi-threading mode by running the following command before adding it:

- `setMultiCpuUsage`

For more information on this and other multiple-CPU commands, see the Multiple-CPU Processing Commands chapter in the *Innovus Text Command Reference*.

Alternatively, fill in the appropriate parameters on the Options - Set Multiple CPU Usage - Multiple CPU Processing form. (You can also access this form by clicking the *Set Multiple CPU* button on the Route -- Metal Fill -- Add -- Add Metal Fill form.)

For more information, see Accelerating the Design Process By Using Multiple-CPU Processing.

After You Complete Adding Via and Metal Fills

After adding via and metal fills, extract parasitics and run timing and signal-integrity analysis, as needed. The metal fill and verify usage is not normally used as sign-off (although it is possible with a strict methodology). In practice, you will still run a final sign-off script on the full-chip to add any fill inside hard-blocks. 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 Quantus QRC including the extra fills from GDS of the sign-off script.

Metal Fill Features

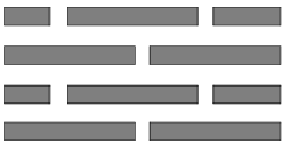
The metal fill has the following features:

- 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.

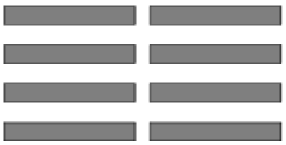
Staggered Metal Fill Pattern

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.

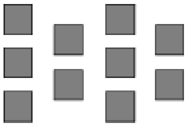
Staggered rectangular metal fill



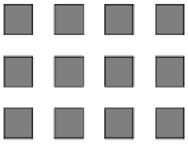
Non-staggered rectangular metal fill



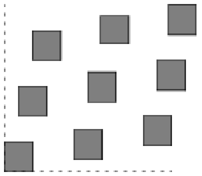
Staggered square metal fill



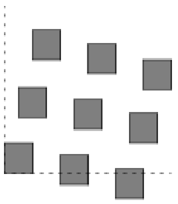
Non-staggered square metal fill



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:



addMetalFill -stagger diag
(horizontal)



addMetalFill -stagger diag
(vertical)

Connected and Floating Metal Fill

Metal fill segments can be connected (tied-off) to power or ground shapes on adjacent routing layers or left unconnected (floating). 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:

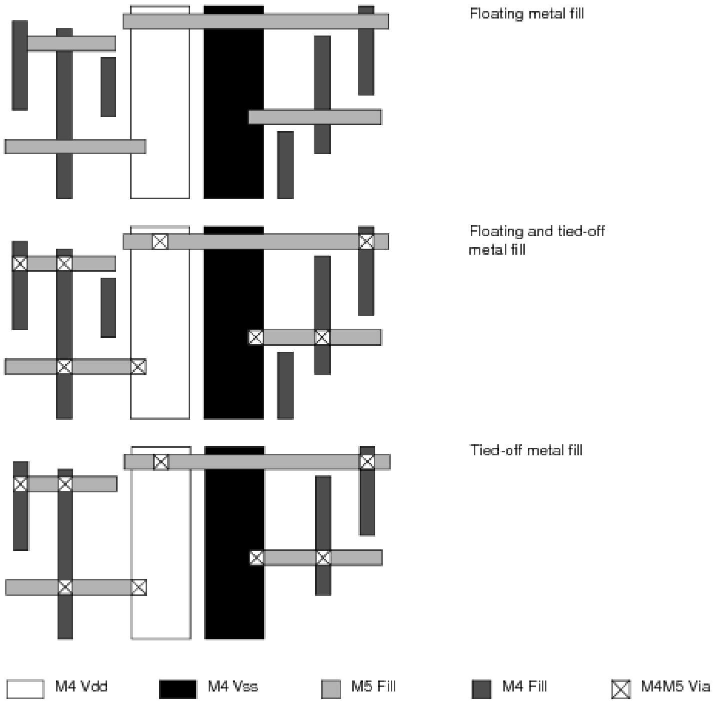
1. Lower cross coupling capacitance, especially if you specify short metal fill segments (long metal fill segments behave like they are really tied off.
2. Easier to trim when there are violations. You can trim a floating metal fill that causes DRC violations with the `trimMetalFill` command. If you 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.

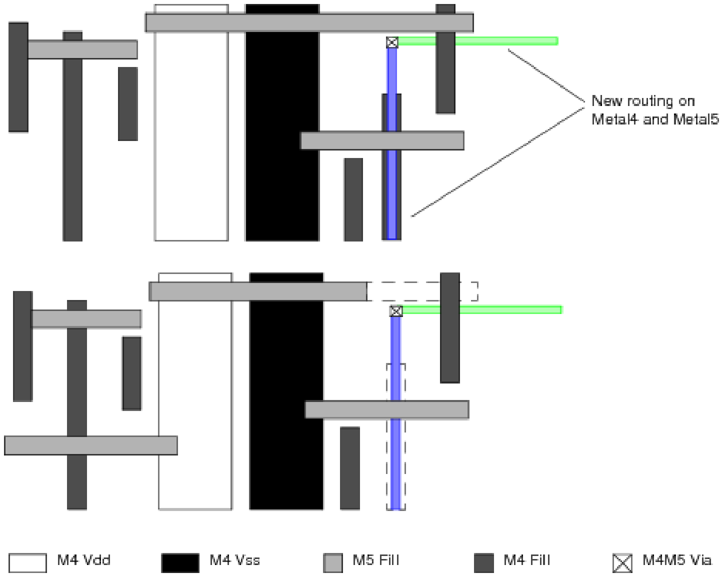
For more information, see the figures below and "Trimming Metal Fill".

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.

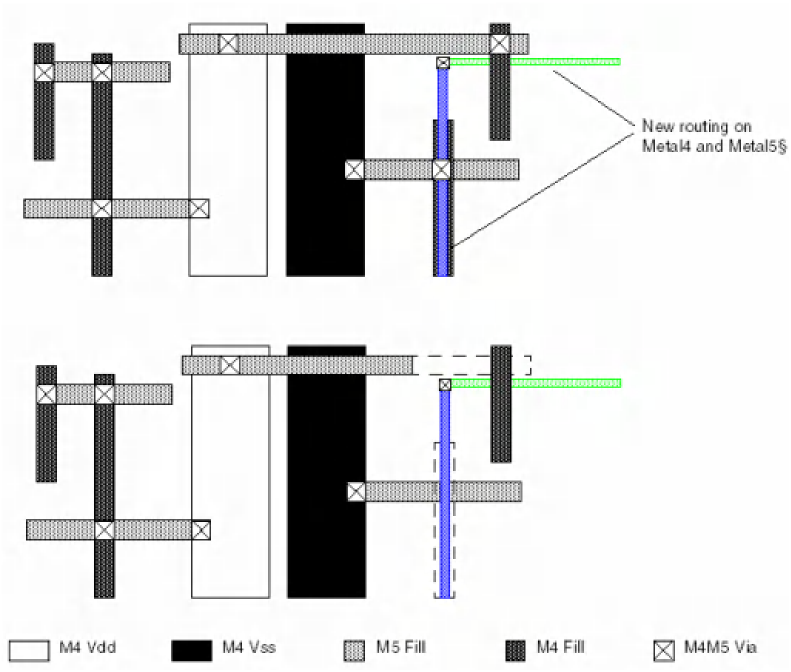


The figures below show the same design after an ECO, in which routing was added on *Metal4* and *Metal5*.

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.



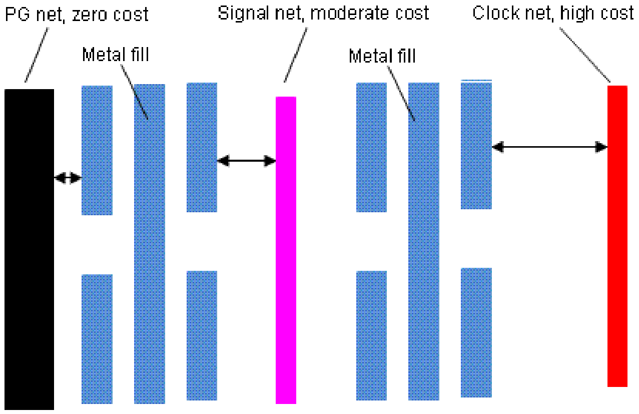
These figures show what happens when you use a tied-off metal fill. The first figure shows the design with the added routing. The second figure shows the design after the metal fill is trimmed. The dotted lines show where the metal fill was trimmed.



Timing-Aware Metal Fill

When it adds a timing-aware metal fill, the Innovus software avoids adding the fill near clock and signal nets and adds more near the power and ground nets.

The software assigns a high cost to adding a metal fill near clock nets, a moderate cost to adding it near signal nets, and zero cost to adding it near power and ground nets. It adds the fill, based on the cost, to achieve the preferred metal density with the least effect on timing.



The software adds timing-aware metal fill by default.

- To add a non-timing aware metal fill, type the following command:

```
addMetalFill   -timingAware off
```

- To use the Innovus common timing engine (CTE) for static timing analysis (STA), type the following command:

```
addMetalFill -timingAware sta -slackThreshold  value
```

If the timing graph is already built, the software adjusts the costs as a function of the slack (nets with the worst slack have the highest cost). For more information, see Timing Analysis.

When you run the software in the STA mode, it assigns costs to four categories of nets:

- Clock nets are assigned the highest cost.
- Signal nets that have a slack value less than the threshold are assigned a moderate cost. You can use `-slackThreshold` and `-extraCriticalNet` to choose critical signal nets.
- Non-critical signal nets are assigned a small cost.
- Power and ground nets (nets marked `+ USE POWER` or `+ USE GROUND` in the DEF file) are assigned zero cost.

Specifying Metal Fill Parameters

Some of the metal fill parameters can be specified in the Layer (Routing) section of the LEF file. All the parameters can be specified by the Innovus metal fill commands or forms. The parameters that can be specified in the LEF file are listed in the table below.

If a parameter is specified in the LEF file, use the specified value. If a parameter is not specified, check the chip manufacturer's DRC manual for the correct metal fill (or dummy fill) values and specify them manually with the command or form.

If not specified properly, a metal fill can cause DRC violations and increase capacitance unnecessarily. Parameters specified by the metal fill commands override parameters specified in the LEF file only if they are more restrictive than the LEF parameters.

The following table lists the metal fill parameters that can be specified in the LEF file and the corresponding Innovus metal fill parameters:

Description	**LEF Statements**	**setMetalFill Parameter**	**Setup Metal Fill Parameter**
Minimum distance between a segment of metal fill and another type of object in the design, such as a signal wire	`FILLACTIVESPACING`	`-activeSpacing`	*Active Spacing*
Minimum density allowed in the design	`MINIMUMDENSITY`	`-minDensity`	*Metal Density % Min*
Maximum density allowed in the design	`MAXIMUMDENSITY`	`-maxDensity`	*Metal Density % Max*
Area the Innovus software uses to examine metal density	`DENSITYCHECKWINDOW`	`-windowSize`	*Step Size X* *Step Size Y*
Distance the window moves for each metal fill iteration	`DENSITYCHECKSTEP`	`-windowStep`	*Window Size X* *Window Size Y*

The Innovus software maintains the values specified for these parameters until you reset them or you restart the software.

For more information on LEF, see the LEF Syntax section of 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 post-mask 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 100 100 -windowStep 50 50\
     -minDensity 15 -maxDensity 85
```

The following command sets values for the gap spacing, preferred density, minimum and maximum width, and minimum and maximum length for the thin-metal layers:

```
setMetalFill -layer "1 2 3 4 5 6"
     -preferredDensity 35 -gapSpacing 0.2 \
     -minWidth 0.1 -maxWidth 1.0 \
     -minLength 0.1 -maxLength 10.0
```

The following command sets values for the gap spacing, preferred density, minimum and maximum width, and minimum and maximum length for the thick-metal layers:

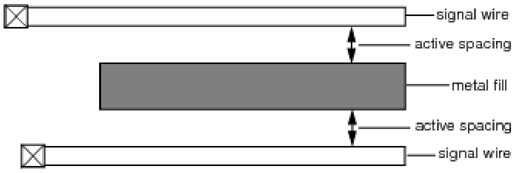
```
setMetalFill -layer "7 8"
     -preferredDensity 35 -gapSpacing 0.4 \
     -minWidth 0.2 -maxWidth 2.0
     -minLength .2 -maxLength 20.0
```

The following command adds metal fill to all eight layers:

```
addMetalFill -layer "1 2 3 4 5 6 7 8 -timingAware sta"
```

Specifying the Active Spacing Value

The space between metal fill and nonmetal-fill geometries is called the *active spacing*, as shown in the following figure.



The Innovus software uses the `FILLACTIVESPACING` value in the LEF file for the active spacing. If `FILLACTIVESPACING` is not specified, you can set it manually by using one of the following methods:

- Specifying a value for `setMetalFill -activeSpacing` on the text command line

  Note: The `setMetalFill -activeSpacing` settings are used for creating regular `FILLWIRE` shapes. For `FILLWIREOPC` shapes, design rules are used.

- Specifying a value for *Active Spacing* on the Setup Metal Fill form

If no value is specified in the LEF file, and you do not specify one manually, the software uses 0.6 microns for thin layers (less than 0.24 microns) and 0.8 microns for thick layers as the default active spacing value.

The default active spacing value is usually large enough that you can avoid using Optimal Proximity Correction (OPC) for the metal fill shapes. In addition, the default active spacing minimizes the increase in cross-coupling capacitance caused by the metal fill, which in turn reduces the additional timing delay.

As you increase the active spacing value, you reduce the space available for metal fill. A large increase--for example, 1 um to 2 um for a 28 nm or 45 nm process--might prevent you from meeting the minimum density rule for some windows.

Adding Metal Fill Over Macros

For adding metal fill to macros, the added metal fill is based on the recalculated metal density for that metal layer. If the added fill is less than the preferred metal density (the default is 35 percent), the software tries to add metal fill shapes on that metal layer to meet the preferred density goal. Otherwise, it does not add any metal fill shapes.

For better metal density accuracy, use the LEF `DENSITY` table--a list of rectangles with metal density numbers. These rectangles cover the entire macro bounding box for that metal layer. The rectangles are defined in the `MACRO` section of the LEF file and are honored by the `addMetalFill` and `verifyMetalDensity` commands. To force `addMetalFill` to place correct metal fill shapes for a layer, place obstructions on the layer to block areas where metal fill should not be placed.

The `DENSITY` rectangles on a layer should not overlap and should cover the entire area of the macro. Choose the size of the rectangles based on the uniformity of the density of the block. If the density is uniform, a single rectangle can be used. If the density is not uniform, the size of the rectangles can be specified to be 10 to 20 percent of the density window size, so that any error due to non-uniform density inside each rectangle area is small.

For ex ample, 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.

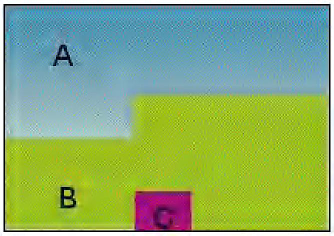
A routing blockage can have the following attributes:

- 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 `verifyMetalDensity` Area ratio > 0.5, A and B are considered as preferred density, C is considered as 100%.
- If `verifyMetalDensity` Area ratio <= 0.5, A and B are considered as 0, C is considered as 100%.
- If OBS is outside of overlap area or macro boundary, the density of OBS is considered as 0.
- If overlap is outside of macro boundary, the density of overlap is considered as 0.
- If no OBS/PIN/OVERLAP in the macro area, the density is considered as 0.

If OBS or PIN is defined on some layers, it means the layers are used by macro. By default, the macro area should not be inserted with metal fill without `-onCell` option. With `-onCell` option, the metal fill can be inserted in macro area but the metal fill should not touch the OBS/PIN shapes.

If no OBS or PIN is defined on a layer, the layer is not used by macro. By default, the macro area should be inserted with metal fill without `-onCell` option.

Recommendations for Power Strapping Mode

In power strapping mode, the software makes mesh connections to power and ground bus wiring, instead of the tree-type connections used in regular connected mode. This configuration allows the metal fill shapes to carry current as part of the power and ground structure. Power strapping uses the maximum possible number of cuts in vias, based on the intersection area between layers, instead of using the minimum-cut based connections used in regular connected mode.

To get the best results in power strapping mode, follow these recommendations:

- Use longer maximum lengths (at least 100 um).
  Longer lengths increase the number of times a single metal fill shape intersects with the existing power/ground mesh.
- Use higher values for preferred density (40 percent to 50 percent).
  Higher preferred density increases the number of metal fill segments retained as candidates for power strapping.
- Use wider metal fill

Adding Via Fill

When it adds via fill, the Innovus software does the following:

- Attempts to add vias that meet cut density requirements
- Uses metal width and spacing values specified by the `setMetalFill` command (or Set Metal Fill form) to determine size and allowed placement locations
- Adds either tied-off or floating vias until the preferred cut density is met
- In tied-off vias, either the top or bottom layer is connected to power or ground.
- Floating vias are not connected to power or ground.
  Floating vias could be inserted between floating metal fills or in white space.

> ⓘ To get the best results from via fill, add it before adding metal fill. You can minimize the need for via fill by inserting multiple-cut vias with the NanoRoute router prior to adding via fill. For information, see `setNanoRouteMode`.

Use the fill commands in the following recommended order:

1. `setViaFill`
2. `setMetalFill`
3. `addViaFill`
4. `addMetalFill`

Note: You can use the `verifyMetalDensity` and `verifyCutDensity` commands to verify that the metal density of the metal layers and cut layers is within the minimum and maximum density values specified by the `setMetalFill` and `setViaFill` commands, respectively.

Recommendations for Metal/Via Fill Flow

In the recommended flow, the software adds via fill to free space prior to adding other metal fill shapes. It does not connect via fill to metal fill.

Use the fill commands in the following order:

1. Set via and metal layer parameters.

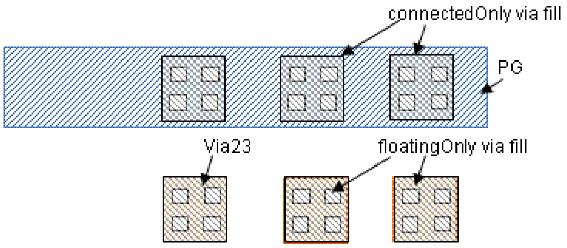
   ```
   setViaFill -layer "Via23" -windowSize 50 50 -windowStep 25 25 -minDensity 0.005 -maxDensity 30 ...

   setMetalFill -layer "Metal2 Metal3" -activeSpacing 0.4 -gapSpacing 0.1 -maxWidth 0.1 -maxLength 10 -windowSize
   50 50 -windowStep 25 25 -minDensity 15 -maxDensity 85 ...
   ```

   You also can specify the parameters in the GUI using the *Setup Metal Fill Options* and *Setup Via Fill Options* forms.
   The `addViaFill` and `addMetalFill` commands will honor the setting to add via and metal fill.

2. Add via fill with specified options.

   ```
   addViaFill -layer "Via23" -mode floatingOnly -area "0 0 100 100"
   ```

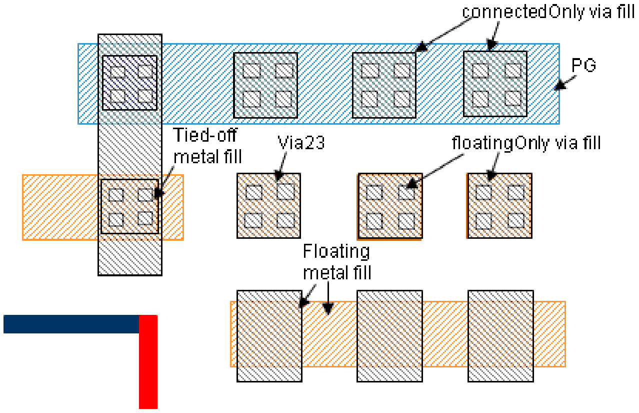
   This will add via fill in white space to meet the via density requirements according to the specified rules. Via fill can be connected to power or ground nets (tied off) or unconnected (floating). Via fill cannot be connected to signal nets.



3. Add metal fill with specified options.

```
addMetalFill -layer {Metal2 Metal3} -area 0 0 100 100 -stagger on -timingAware sta -onCells -net {VDD VSS} -
mesh
```

This will insert inactive metal into white space to achieve the metal density that is required by a specific manufacturing process. However, the inactive metals do not touch any other metal fill.



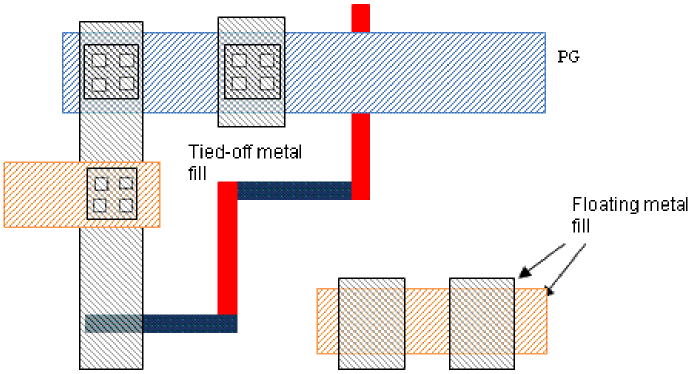
Innovus now provides an alternative flow in which the software adds metal fill to free space prior to adding via fill shapes. It can connect metal fill with special via fill. In this alternative flow, you:

1. Set metal layer parameters.

```
setMetalFill -layer "Metal2 Metal3" -activeSpacing 0.4 -gapSpacing 0.1 -maxWidth 2.0 -maxLength 10 -decrement 2
-diagOffset 0.4 0.4 -windowSize 50 50 -windowStep 25 25 -minDensity 15 -maxDensity 85 ...
```

2. Add metal fill with specified options.

```
addMetalFill -layer {Metal2 Metal3} -excludeVia Dvia -net {VDD VSS} -area 0 0 100 100 -stagger on -timingAware
sta -onCell -mesh
```



This step inserts inactive metal into a placed and routed design to achieve the required metal density according to the specified parameters. The software attempts to connect metal fill to the first net in the list, then the next net, and so on. However, the Dvia should not be used to connect to special nets. If the metal fill cannot connect to special nets, keep them floating.

3. Set via layer parameters.

```
setViaFill -layer "Via23 " -windowSize 50 50 -windowStep 25 25 -minDensity 0.005 -maxDensity 30 ...
```

The parameters honor settings in the following order:

a. `setViaFill`

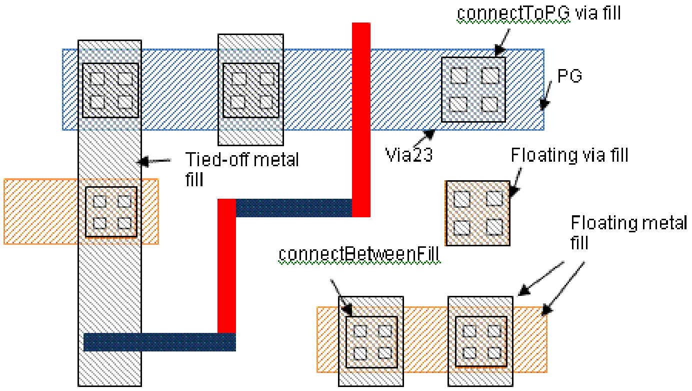
b. `setMetalFill`

c. LEF

d. Manufacture process default

4. Add via fill with the specified options.

```
addViaFill -layer "Via23" -area "2 4 6 8" -mode {floating connectToPG connectBetweenFill} -includeVia Dvia
```



This connects 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 Pegasus/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.

If using Pegasus:

```
streamOut -mapFile $gds_map -outputMacros -units $gds_unit pegasus.fill.gds
```

`run_pegasus_metal_fill` calls the Pegasus engine to insert metal fill and then dump the metal fill in Innovus. The DEF mapping file is required to ensure that the metal fill is put in the correct layers. The top cell name is also required.

```
run_pegasus_metal_fill -ruleFile $MF_RULE_DECK -defMapFile $def_out -gdsFile pegasus.fill.gds -cell $top_cell
```

If using PVS:

```
streamOut -mapFile $gds_map -outputMacros -units $gds_unit pvs.fill.gds
```

`run_pvs_metal_fill` calls the PVS engine to insert metal fill and then dump the metal fill in Innovus. The DEF mapping file is required to ensure that the metal fill is put in the correct layers. The top cell name is also required.

```
run_pvs_metal_fill -ruleFile $MF_RULE_DECK -defMapFile $def_out -gdsFile pvs.fill.gds -cell $top_cell
```

2. Analyze the timing impact by metal fill.

   After the metal fill is inserted, run `timeDesign` to check the timing.

   ```
   timeDesign -postRoute -outDir postfill
   ```

   If the timing result is acceptable, the metal fill step is complete.

3. Trim metal fill for timing closure.

   If the inserted metal fill impacts timing, use `trimMetalFillNearNet` to trim the metal fill near nets for timing closure.

   You can trim the same and upper/bottom layer metal in timing aware mode. Use `-layer` to specify the layers on which metal can be trimmed. The spacing between metal fill and critical nets can be specified with the three spacing options. You can specify different spacing for different slack net with multi-pass trimming.

   You can specify the critical net with the `-net` option. You can also specify the slack threshold. If you do so, the tool decides the critical net list with the slack threshold.

   If the minimum metal density needs to be controlled, specify the window size and step. The metal density can then be calculated with window setting.

   a. Set metal fill parameters

      ```
      setMetalFill -minDensity 10 -maxDensity 85 -preferredDensity 35  -windowStep 62.5 62.5 -windowSize 125 125
      ```

If the minimum density target is specified, trimming stops as soon as the minimum density is achieved. If you do not specify a minimum density target, metal fill is trimmed with the specified spacing.

b. Trim the metal fill near more critical nets with bigger spacing.

```
trimMetalFillNearNet -createFillBlockage -slackThreshold $slack1 -spacing  1.0 -spacingAbove 1.0 -
spacingBelow 1.0 -minTrimDensity $min_density
```

c. Trim the metal fill near less critical nets with comparatively smaller spacing.

```
trimMetalFillNearNet -createFillBlockage -slackThreshold 0.0 -spacing 0.4 -spacingAbove 0.4 -spacingBelow
0.4 -minTrimDensity $min_density
```

4. Analyze the timing impact after trimming metal fill.

```
timeDesign -postRoute -outDir posttrimfill
```

If the timing result is still not acceptable, repeat Step 3 until timing closure.

Note: Innovus metal fill does not support 20nm and below node design rules. We strongly recommend the Pegasus/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 Pegasus/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` does not support 20nm and below node design rules.***

Signoff Fill - Pegasus Hierarchical Metal Fill

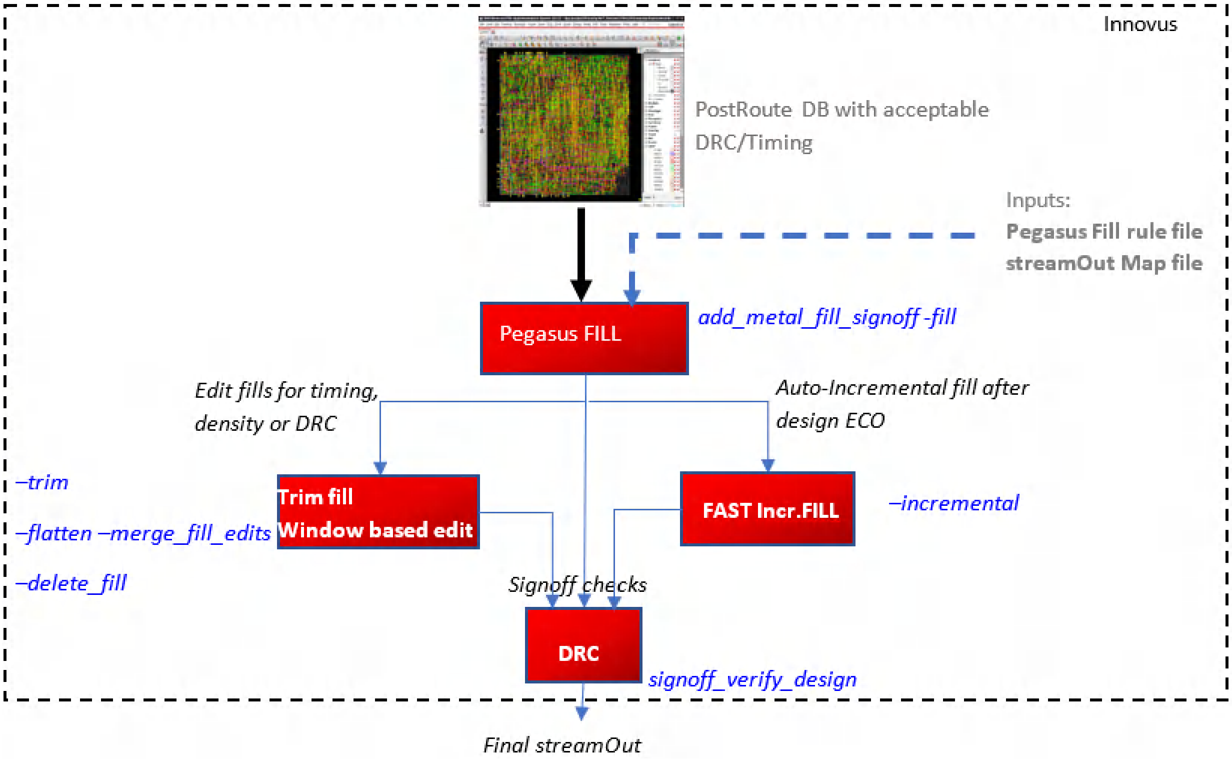
The recommended flow to signoff fill within Innovus is based on the Hierarchical Metal Fill (HMF) model, which uses the Pegasus signoff fill rule file. HMF does not support fill with external tools (OT).

HMF provides unified metal fill database across Innovus, Pegasus, and Quantus, and offers the following features:

- Lesser data storage requirement and memory footprint.
- Lower runtimes at every stage of:
    - Metal fill generation
    - Final `streamOut`
    - `saveDesign`/`restoreDesign`
- Utilities that help reduce the turnaround time through:
    - Timing aware setup
    - Trim fill function integrated into flow to recover timing
    - Incremental fill after ECO
    - Window-based editing of fills
    - Background run to release Innovus licenses during Pegasus fill

- Run extraction and timing seamlessly

The following figure explains the HMF flow:



HMF Commands and Parameters

Metal Fill

- `set_metal_fill_signoff_mode`: Sets fill, `streamOut`, and trim parameters.
- `get_metal_fill_signoff_mode`: Returns fill, `streamOut`, and trim parameters.
- `add_metal_fill_signoff`: Runs the signoff metal fill flow to generate HMF.

Following are the HMF-related parameters of these commands:

Parameter	Description
`-fill`	Runs signoff metal fill on the current database view and generates HMF.
`-incremental`	Runs incremental metal fill post ECO and updates HMF.
`-trim`	Trims metal fill for the specified nets or slack, and updates HMF.
`-delete_fill`	Selectively deletes metal fill and updates HMF.
`-flatten`	Selectively flattens metal fill into Innovus in editable mode for DRC/Density/Timing.
`-merge_fill_edits`	Merges metal fill edits into HMF.
`-view_fills`	Allows viewing metal fill in Innovus in the read-only mode.

Stream-out and Extraction

- `streamOut -pvs_fill`: Merges the HMF use-mode FILLONLY data attached to the current database into the output GDS file and instantiates the PVS top fill cell inside the top design cell.
- `setExtractRCMode -pvs_fill true`: Reads-in HMF during extraction/timing.

Save and Restore

- `saveDesign`: Saves the Innovus database with HMF under `*.enc.dat/`.
- `restoreDesign`: Restores the Innovus database and is aware of HMF, if it exists.

Running -fill in Background Mode/Launching Fill Job on LSF

To launch fill job on LSF and run `-fill` in the background mode without locking the Innovus license, use the `-bg` parameter as shown below:

```
add_metal_fill_signoff -fill -bg -temp_working_dir working_dir
```

To check the status of `-bg`, run:

```
add_metal_fill_signoff -check_bg_run working_dir
```

To restore the successfully completed `-bg`, run:

```
add_metal_fill_signoff -load_fill_db working_dir
```

To kill `-bg`, run:

```
add_metal_fill_signoff -kill_bg_run working_dir
```

To submit a background job on LSF, set resource string before launching Innovus by running the following commands:

```
setenv DP_LSF_RESOURCE ‘resource_string_for_slave’
setenv MASTER_LSF_RESOURCE ‘resource_string_for_master’
add_metal_fill_signoff -fill -bg  -lsf -temp_working_dir work_dir
```

Or:

```
add_metal_fill_signoff -fill -bg -master_lsf -lsf -temp_working_dir work_dir
```

For detailed description about HMF, its usage, syntax, examples, and Stylus mode, see *Cadence Pegasus User Guide*.

Achieving Gradient Density with Preferred Density Setting

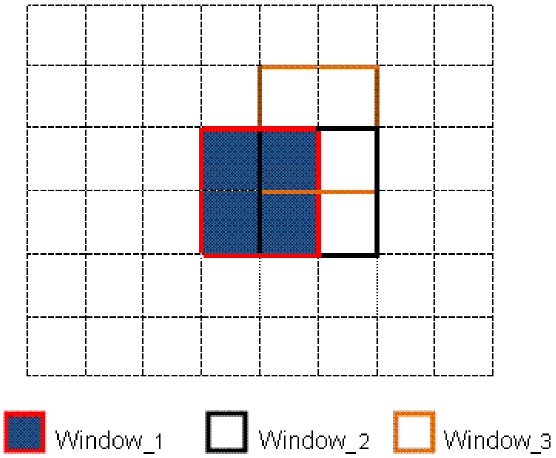
To prevent density in neighboring regions from varying too much, the `addMetalFill` targets a preferred density. This minimizes the variation in density from window to window. You can set the parameters as follows:

```
-minDensity 15 -maxDensity 85 -preferredDensity 35
```

```
addMetalFill -layer {Metal1 Metal2 Metal3}
```

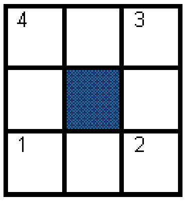
The metal fills are inserted into white space to meet the preferred density. When the metal density in a window is less than the minimum metal fill density value, `addMetalFill` adds metal fill to achieve a density slightly above the preferred density, if possible. If the density is larger than maximum 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.

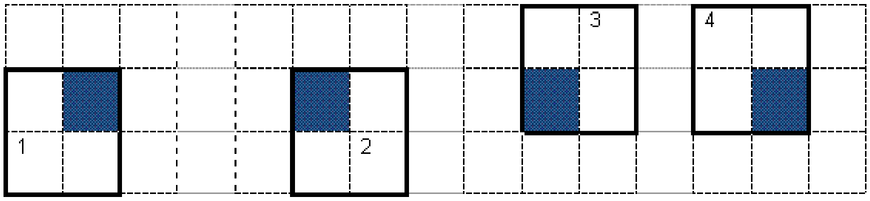


When add metal commands are applied, the engine calculates the Window_1 density and tries to insert metal fill until the window density reaches the preferred density target. When Window_1 is finished, the engine moves to the next window. The window step is specified in `setMetalFill` command. Note that 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	∞	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	∞	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	∞	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
```

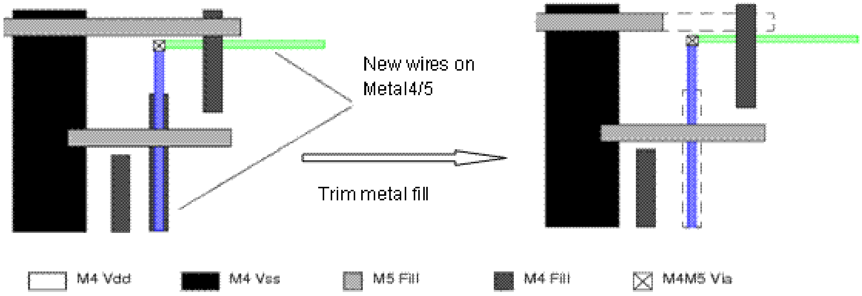
When required, you can use the `getMetalFillSpacingTable` command to print specified or all spacing tables used by `trimMetalFill`.

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 [-deleteViol] [-ignoreSpecialNet]
  ```

  This command repairs violations caused by the metal fill shapes. If the metal density drops below the target after trimming the metal fill, re-run the `addMetalFill` command.

  The `trimMetalFill` command trims metal and via fill shapes based on the following spacing rules:

- Between `FILLWIRE` and `FILLWIREOPC` shapes, the active spacing value or minimum spacing based on DRC rules, whichever is larger, is required.

- Between `FILLWIRE` shapes, the gap spacing value or minimum spacing, whichever is larger, is required.
- Between `FILLWIREOPC` and active shapes, the OPC active spacing value or minimum spacing, whichever is larger, is required.
- Between `FILLWIREOPC` shapes, minimum spacing is required.
  For more information, see `trimMetalFill`.
- To specify the layers that you want to trim to fix DRC violations, use the `-layer` parameter of the `trimMetalFill` command. For example, to trim metal fill in METAL2 and METAL3 layers, use the following command:

  ```
  trimMetalFill -layer {METAL2 METAL3}
  ```

  Note: This option is recommended for use with only floating metal fill. If you use `trimMetalFill -layer` on tied-off fill shapes, some of the shapes may become isolated from the Power/Ground network.
- To limit the area that in which metal fill is trimmed, use the `-area` parameter of the `trimMetalFill` command. This option is recommended for use with only floating metal fill. If you use `trimMetalFill -area` on tied-off fill shapes, some of the shapes may become isolated from the Power/Ground network.
- To remove connected metal fill, complete the following steps:

  a. Trim metal fill.

  b. Fix isolated fill issues with `fixOpenFill`. You can choose to either change isolated PG fills to floating fill or remove isolate fills (`fixOpenFill -remove`). If you choose to remove the isolated fills, you can then add metal fill incrementally to see if any of those islands can be tied either to the same or another PG rail.

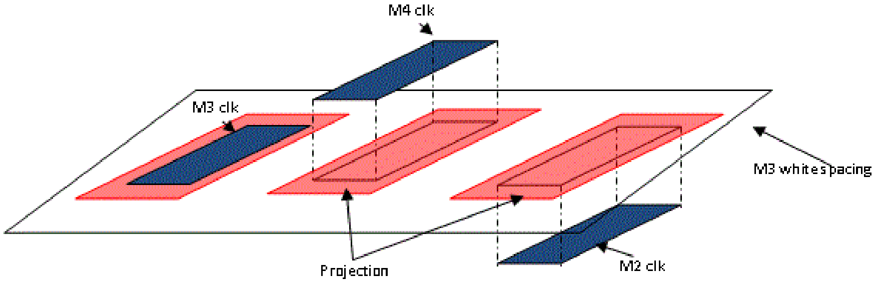
For information on `FILLWIRE` and `FILLWIREOPC`, see 'Shape' in the DEF Syntax chapter of the *LEF/DEF Language Reference*.

(FILLWIREOPC is not supported by LEF 5.6.)

Trimming Metal Fill for Timing Closure

The metal fill (FILLWIRE and FILLWIREOPC) shapes potentially impact the timing if the metal fills are close to the critical nets. Although Innovus provides timing-aware metal fill solution, you could also use post-fill trimming to improve the timing result. If you load third-party metal fill in Innovus with defIn, you could rely on the post-fill trimming for timing closure.

In the following diagram, if you insert metal fill in the red area to meet the metal density requirements, it may impact timing. Use trimMetalFillNearNet to trim the metal fill if the timing impact is high.



To specify critical nets:

- Specify the net list with -net.
- Use -clock to specify that metal fill should be trimmed around all clock nets.
- Use -slackThreshold to specify the slack threshold. All the nets whose slack value is less than the specified threshold are identified as critical nets.

To specify the spacing for trimming:

- Use -spacing to specify the distance to be kept around critical nets in the same layer when trimming fill.
- Use -spacingAbove to specify the distance to be kept around critical nets for the top layer.
- Use -spacingBelow to specify the distance to be kept around critical nets for the bottom layer.

To prevent incremental metal fill close to the critical nets:

- Use -createFillBlockage to create fill blockage after trimming metal fill. The fill blockage prevent metal fill in incremental steps.

To prevent minimum density issues when trimming:

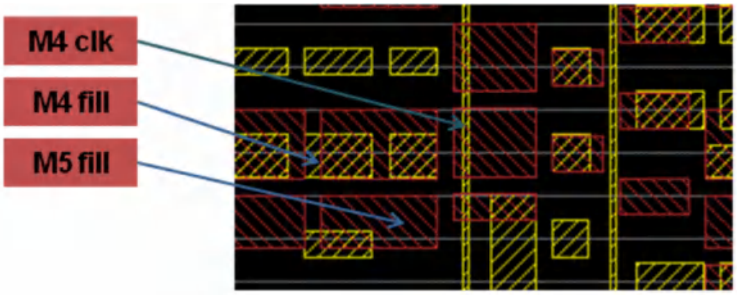
- Set density parameters before running the trimMetalFillNearNet command.
  Use the -minTrimDensity parameter to specify the minimum density value. Innovus calculates the metal density while trimming metal fill. If the metal density is less the minimum density, trimming stops.

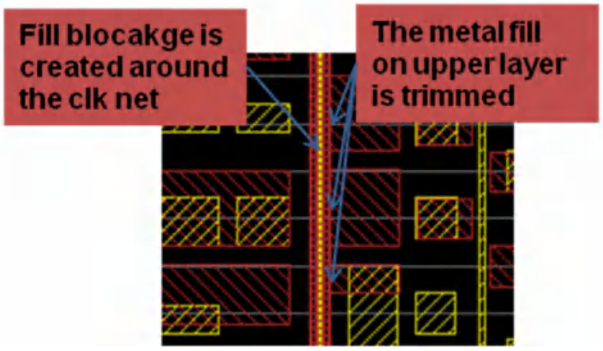
The following example illustrates how to use the trimMetalFillNearNet command.

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.




Fill blocakge is created around the clk net
The metal fill on upper layer is trimmed

Verifying Metal Density

After adding or trimming metal fill, use the Verify Metal Density and Verify DRC features to verify that the metal fill has been added correctly.

Ensure that the minimum, preferred, and maximum density values and window size and step are defined in the `default` iteration name. `verifyMetalDensity` uses the `setMetalFill` settings from the `default` iteration name. The default iteration name settings are the settings used when `setMetalFill` is run either without the `-iterationName` parameter or with `-iterationName default`. If these settings are not available, `verifyMetalDensity` uses the LEF settings. If the LEF settings are not available, `verifyMetalDensity` uses the internal default values for verifying density.

For more information, see the "Verify Commands" chapter of the *Innovus Text Command Reference*.

Adding Metal Fill Using the GUI

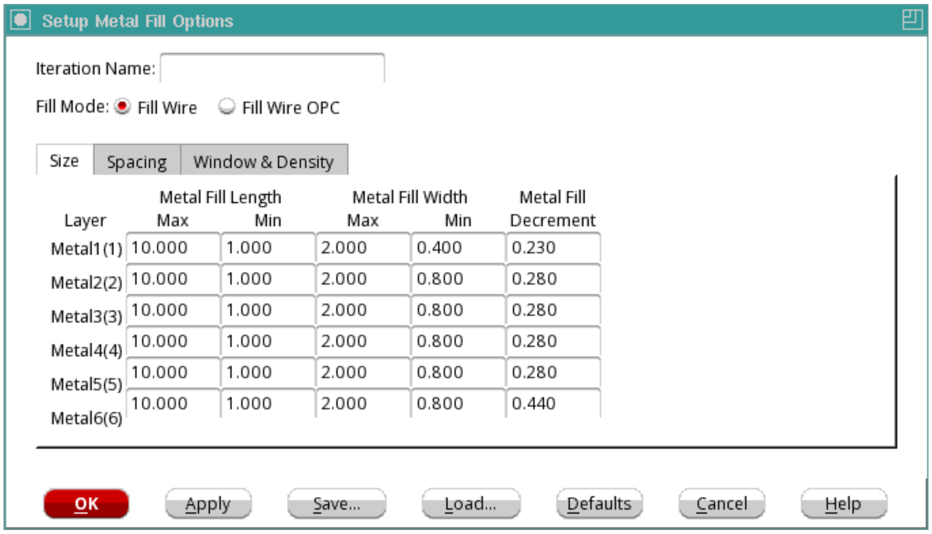
1. Determine the minimum and maximum size for metal fill shapes for each layer, then set these values on the *Size & Spacing* page of the Setup Metal Fill form.
   - If you are using rectangular metal fill, use the *Rectangle Length* and *Metal Fill Width* values.
   - If you are using square metal fill, use the *Metal Fill Width* and *Square Decrement* values.
2. Determine the spacing around metal fill shapes for each layer, then set the value on the *Size & Spacing* page of the Setup Metal Fill form. You must set two types of spacing values:
   - Spacing between a metal fill shape and an active metal shape. An active metal shape can be a signal wire, a power wire, a cell, a pin, or any other structure that is not classified as a fillwire.
   - Spacing between a metal fill shape and another metal fill shape.
3. Determine the minimum, maximum, preferred, and external metal density for each layer, then set these values on the *Window & Density* page of the Setup Metal Fill form.
4. Use the Verify Metal Density form to create a `Verify Density` report.
5. Locate an area in the design for which metal density is too low, then select that area on the Add Metal Fill form.
6. Determine whether you want metal fill to be square or rectangular, then choose the appropriate value on the Add Metal Fill form.
7. Click *OK* or *Apply* on the Add Metal Fill form to add metal fill shapes to the area that you specified.

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*.



Iteration Name:

Fill Mode: ● Fill Wire ○ Fill Wire OPC

	Metal Fill Length		Metal Fill Width		Metal Fill
Layer	Max	Min	Max	Min	Decrement
Metal1(1)	10.000	1.000	2.000	0.400	0.230
Metal2(2)	10.000	1.000	2.000	0.800	0.280
Metal3(3)	10.000	1.000	2.000	0.800	0.280
Metal4(4)	10.000	1.000	2.000	0.800	0.280
Metal5(5)	10.000	1.000	2.000	0.800	0.280
Metal6(6)	10.000	1.000	2.000	0.800	0.440

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
```

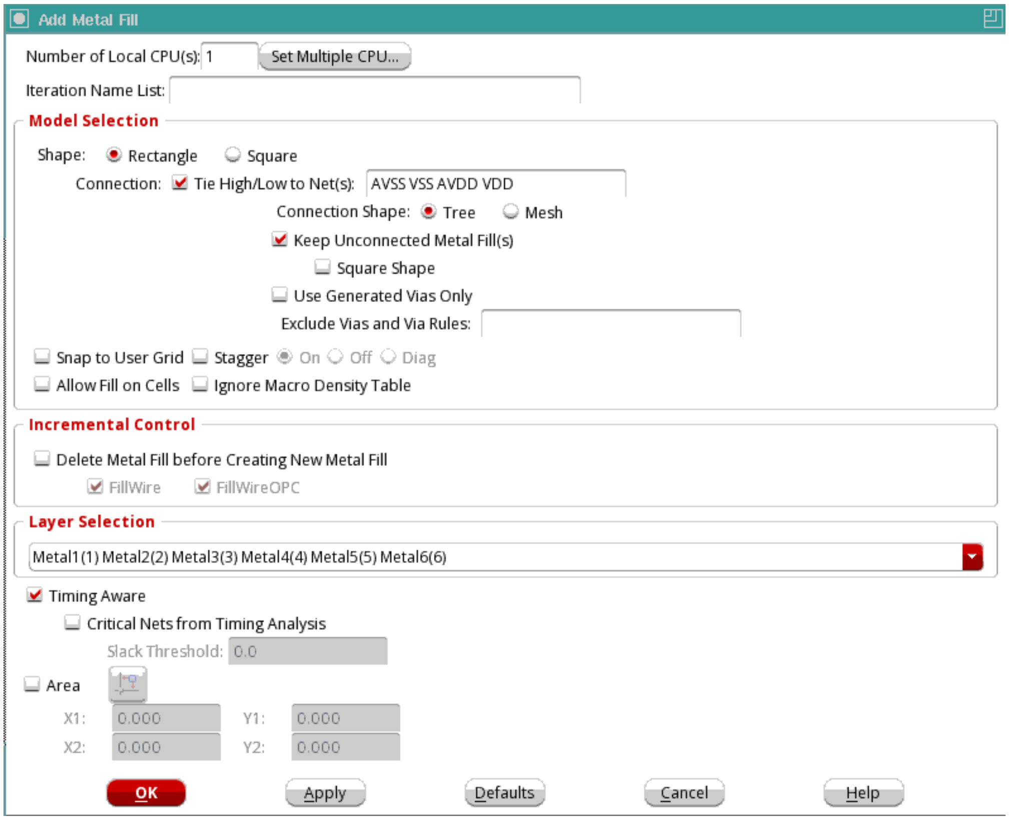
If you want to specify different window size and step when adding metal fill, you need to run `addMetalFill` in separate steps. In the following example, the specified values for `-windowSize` and `-windowStep` in *`step1`* , *`step2`* , and *`step3`* are different:

```
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.



Number of Local CPU(s): 1

Iteration Name List:

Model Selection

Shape: ● Rectangle ○ Square

Connection: ☑ Tie High/Low to Net(s): AVSS VSS AVDD VDD

Connection Shape: ● Tree ○ Mesh

☑ Keep Unconnected Metal Fill(s)

□ Square Shape

□ Use Generated Vias Only

Exclude Vias and Via Rules:

□ Snap to User Grid □ Stagger ● On ○ Off ○ Diag

□ Allow Fill on Cells □ Ignore Macro Density Table

Incremental Control

□ Delete Metal Fill before Creating New Metal Fill

☑ FillWire ☑ FillWireOPC

Layer Selection

Metal1(1) Metal2(2) Metal3(3) Metal4(4) Metal5(5) Metal6(6)

☑ Timing Aware

□ Critical Nets from Timing Analysis

Slack Threshold: 0.0

□ Area

X1: 0.000 Y1: 0.000

X2: 0.000 Y2: 0.000

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.

Viewing Metal Density Map in the GUI

You can generate the metal density report file containing the metal density violation information and view it in the GUI. For this, you need to use the `-report` parameter of the `verifyMetalDensity` command. For example, the content of the metal density report file is:

Metal	Density	Window Size
M1	11.14	(0 0) (10 10)
M1	8.96	(0 10) (10 20)
M1	8.96	(10 0) (20 10)
M1	8.96	(10 10) (20 20)

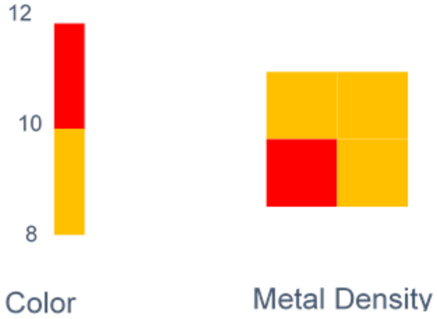
Run the following commands to check the metal density of all layers and view the metal density map in the GUI:

```
verifyMetalDensity -saveToDB
```

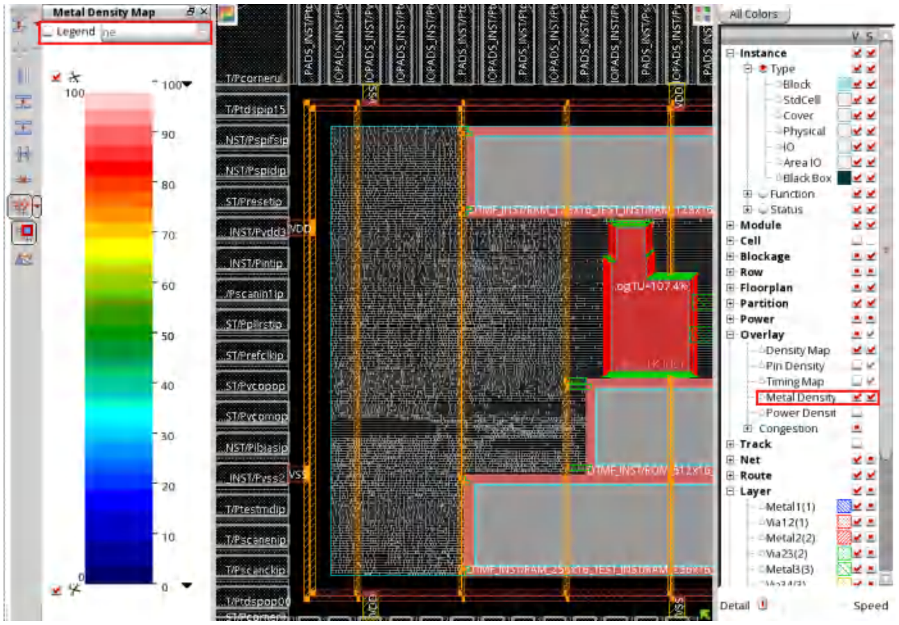
```
verifyMetalDensity -report reportName.rpt
```

Note: Before running the above commands, you need to set the same values for `setMetalFill -windowSize` and `setMetalFill -windowStep`. Otherwise, the display of metal density is overlapped.

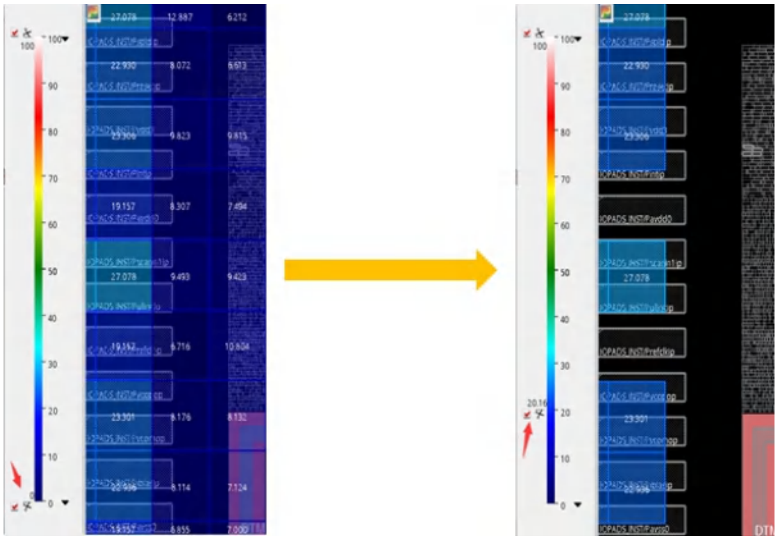
The output of the above commands is:



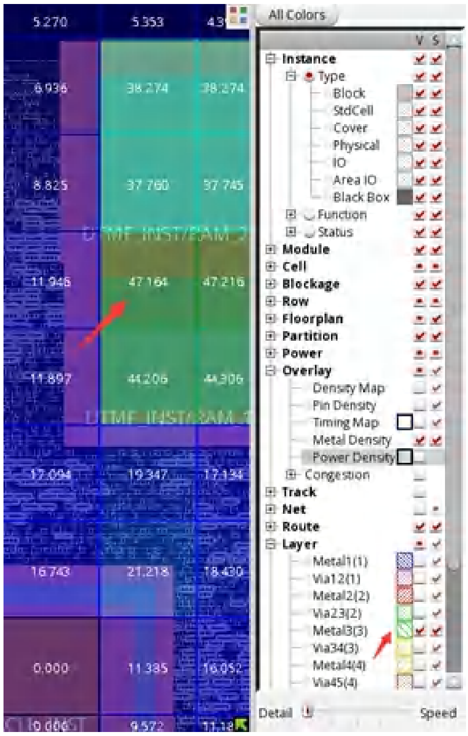
Additionally, the metal density map is displayed in the GUI with a layer control on the color panel, and the Legend check box on the left, as shown below:



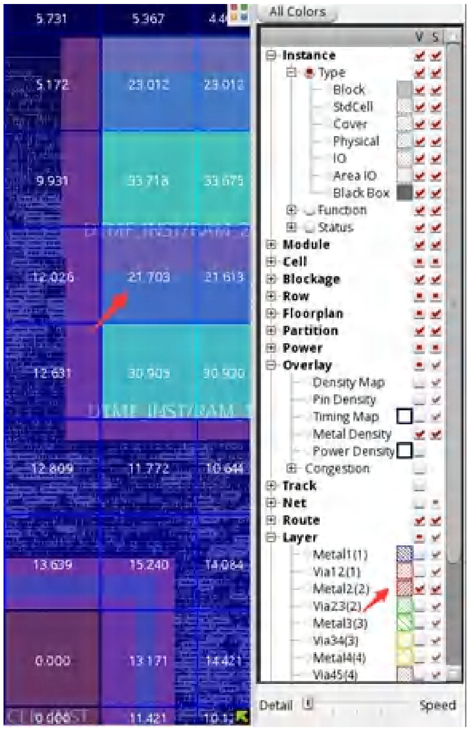
To modify the color range, slide up or down the color range slider, as highlighted below:



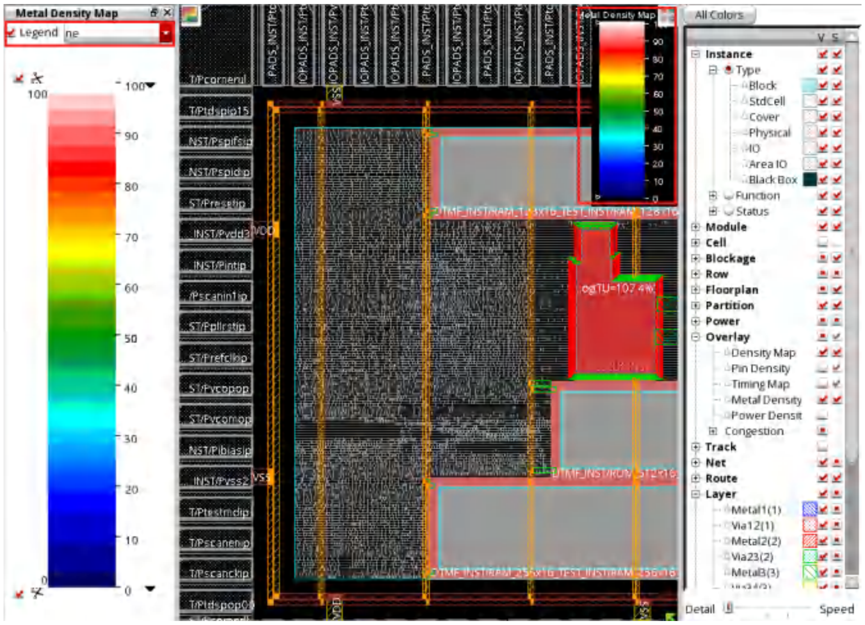
The following figure depicts the metal density map for Metal Layer 3:



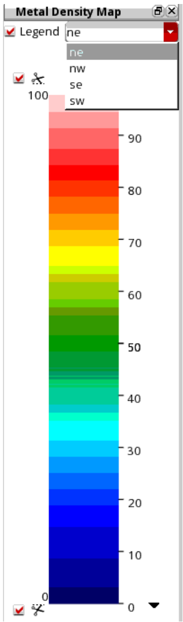
The following figure depicts the metal density map for Metal Layer 2:



Select the *Legend* check box on the Metal Density Map bar to see the legend on the layout.



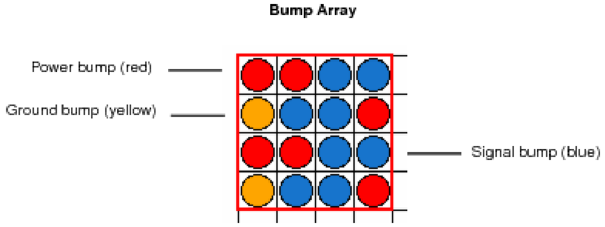
Use the drop-down menu next to *Legend* to change the location of the legend. Here, *ne* means north east, *nw* means north west and so on.



Flip Chip Methodologies

Overview

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



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

Related Packaging Tools

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

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

Before You Begin

Before using flip chip, the following information is required:

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

Using This Chapter

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

- For general flip chip flow information, see Flip Chip Flow in Innovus.
- For information on a specific type of flow, see one of the following sections:
    - SiP Bump Flow
    - Area I/O Flow
    - Peripheral I/O Flow

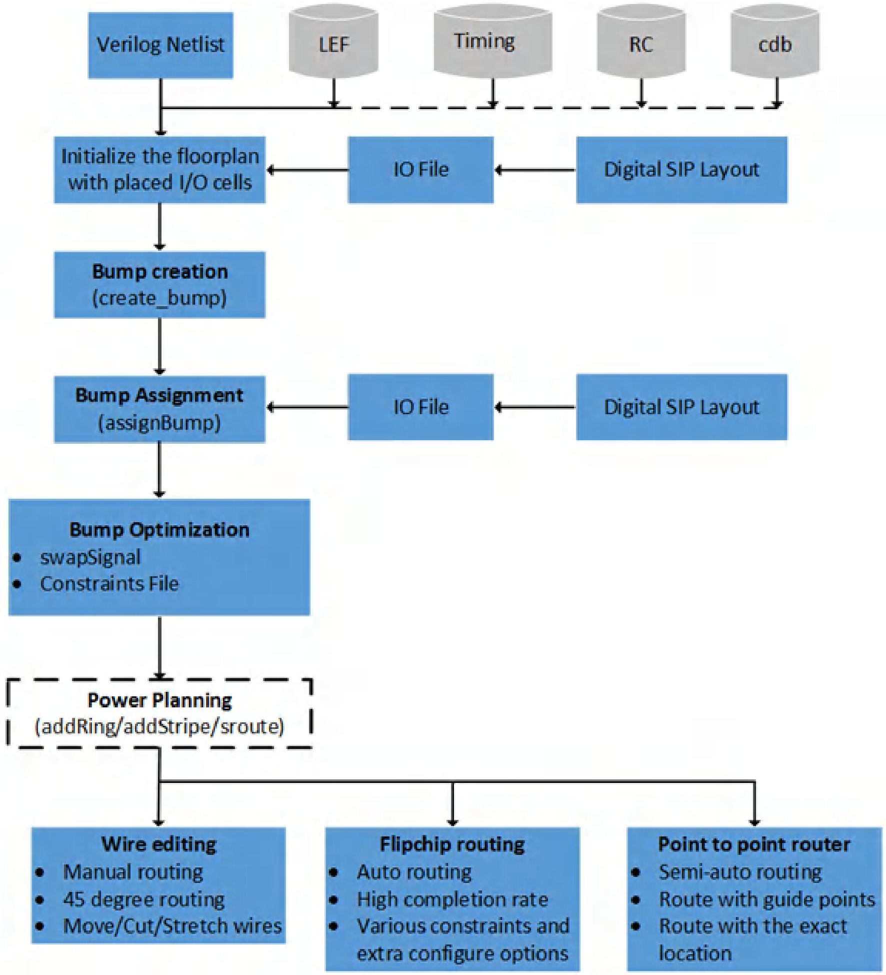
In addition to the above, Innovus also supports a mix of Peripheral I/O and Area I/O styles.

Related Flip Chip Information

- Text commands
  For information on the flip chip commands, see the Flip Chip Commands and Global Variables chapter of the *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 *Menu Reference*.

Flip Chip Flow in Innovus

The following figure shows the general Innovus flip chip flow.



Introduction to Flip Chip Methodology

This section describes the various implementation and flow methodologies for flip chip.

Flip Chip supports the following implementation methodologies:

- SiP Bump Flow
- Area I/O Flow
- Peripheral I/O Flow

SiP Bump Flow

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

Reducing Data Size for SiP Import (Bypass Flow)

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

```
defOut -noCoreCells
```

This flow bypasses the bump flow (see Flip Chip Flow in Innovus).

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

Testing the Package Routing Feasibility

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

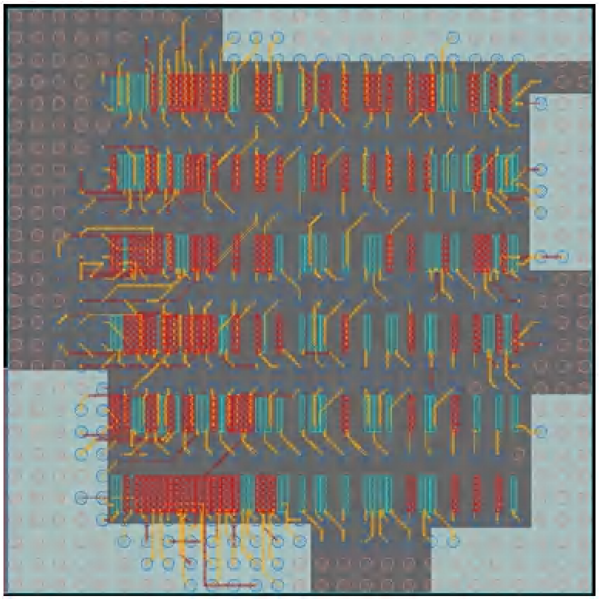
For more information, see the *Cadence Chip I/O Planner User Guide* or the *SiP Digital Architect/Layout User Guide* on SourceLink.

Area I/O Flow

In Area I/O (AIO) designs, the IO PADs are placed in the core area. You can define IO pad row clusters in the core, where these IO pads will be placed, and from where they will be routed. With this implementation, routing is much less constrained. As a result, routing congestion issues arise rarely. The bumps can be defined and placed close to the IO pads shortening the net length.

The disadvantage of this methodology is that the IO pad placement affects the standard cell placement and therefore the full timing closure flow. Power routing is also more demanding in this implementation as dedicated power stripes must be routed to feed the power requirement for the IOs.

In general, this implementation style is more complex but offers much less net delay from IO pad to the bump. In addition, the SI effect is greatly reduced as the net length is much shorter.



To create a flip chip design, complete the following steps:

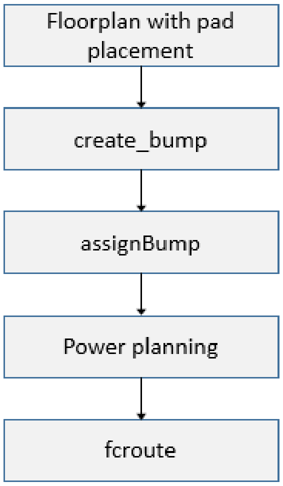
1. Load the floorplan with I/O pad placement.
2. Define the bumps.
   Use the `create_bump` command or the Create Bump Array form to set up the bump array.
3. Assign signals, power, and ground to the bumps.
   - Use the *Bump Assignment* tab of the Flip Chip form to assign the signals, power and ground to bumps. Signal bumps are blue-filled squares. Power bumps are red-filled squares. Ground bumps are yellow-filled squares.
4. Create power rings and stripes.
   - Use the Add Rings form to create rings around the core area and around the power and ground bumps.
   - Use the Add Stripes form to create stripes that connect to the power and ground bumps.
5. Connect power, from bumps to I/O cells or from bumps to rings/stripes.
   - Use the *RDL Routing* tab of the Flip Chip form to establish the power connections.

For more information on the creating power rings form, see Add Rings in the Power Menu chapter in the *Menu Reference*. For more information on creating stripes, see Add Stripes in the Power Menu chapter in the *Menu Reference*.

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

AIO Command Flow

The AIO command flow can be depicted as follows:



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

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

The text command for pushing down the data is:

```
handlePtnAreaIo -insertBuffer buffer_name
```

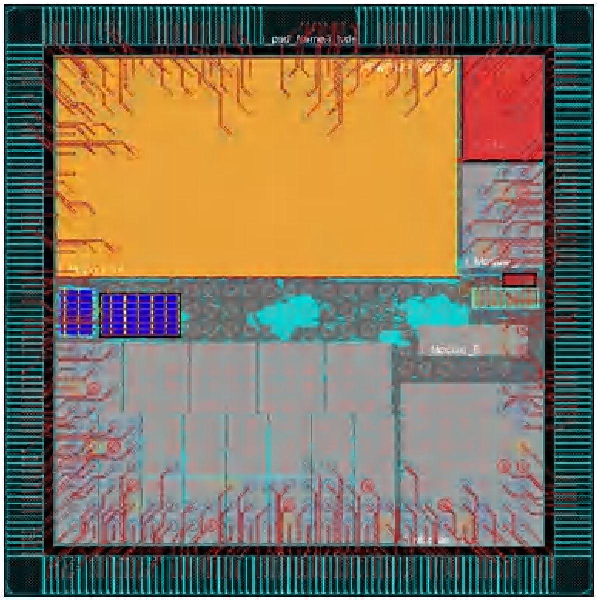
This command pushes down data in the partition as follows:

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

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

Peripheral I/O Flow

IO PADs are placed on the periphery of the die. If the design is severely pad limited, it could have multiple IO rings around the core area. The pin of the pad is routed to the bumps using the redistribution layer (RDL). These bumps are located in the core area of the chip. The diagram below depicts a typical peripheral design.



The recommendation is to floorplan the design, including the bump location, assignment, and RDL routing. The benefit is that you can take the advantage of having more freedom when moving IO pads and bumps. Moving IO pads that are not related to analog blocks is feasible at this stage. Bump movement needs to be verified for routing purposes in SiP. SiP verifies that there is no routing congestion between bumps and package balls using the current bump assignment.

It is not recommended to implement the flip chip features as a post process after design closure. During full chip implementation, you could choose not to use the RDL layer for signal or power routing and reserve this layer for the flip chip router. In this case, the bump assignment and routing can be performed in parallel to the implementation flow or as a post step after final timing signoff. This methodology restricts the movement of the IO pads as they are fixed after implementing the design closure flow. In addition, the package tool (SiP) can limit the bump location. In this case, it is common to face routing congestion and hotspots.

A peripheral I/O (PIO) design has no impact on the default timing/area/power/DFM-DFY closure implementation flow.

PIO Flow Steps

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

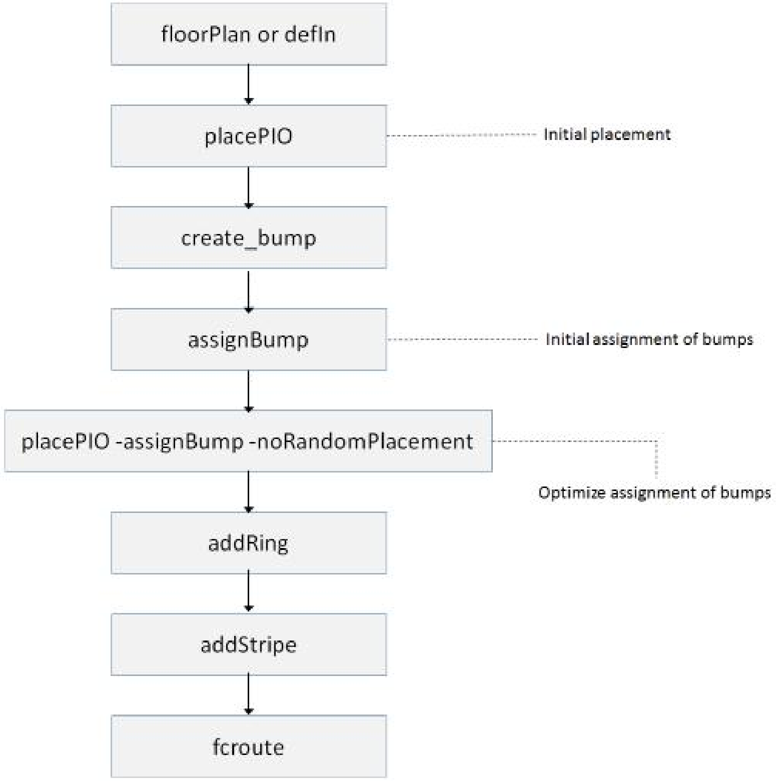
There are four major elements of the flow:

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

for timing and SI analysis with the RDL effects.

PIO Command Flow

The PIO command flow can be depicted as follows:


floorPlan or defIn
placePIO
Initial placement
create_bump
assignBump
Initial assignment of bumps
placePIO -assignBump -noRandomPlacement
Optimize assignment of bumps
addRing
addStripe
fcroute

Flow Methodologies

The design of an IC and its package/carrier has traditionally been two separate development processes done in succession (Serial Design Flow), driven from a common specification.

Innovus-driven Floorplaning

The digital implementation engineer has an initial and rough floorplan to start with. For example:

- The size of the chip might be given as a constraint from marketing or from the customer.
- The location and ordering of some or all the I/Os may already be known. This information may come from the PCB or package designer, or there may be some inherent placement requirements imposed by an analog block in the core.

All this information can be fed into Innovus during the implementation. In this case, Innovus can drive the flip chip implementation by placing these PADs and then creating and assigning the bumps. After this implementation, the design needs to go to SiP Layout (Cadence's package implementation tool) to verify that the bump placement can be routed to the package balls. This is becoming very critical as users are trying to reduce the size of the package which is limiting the routing resources.

Package-driven Floorplaning

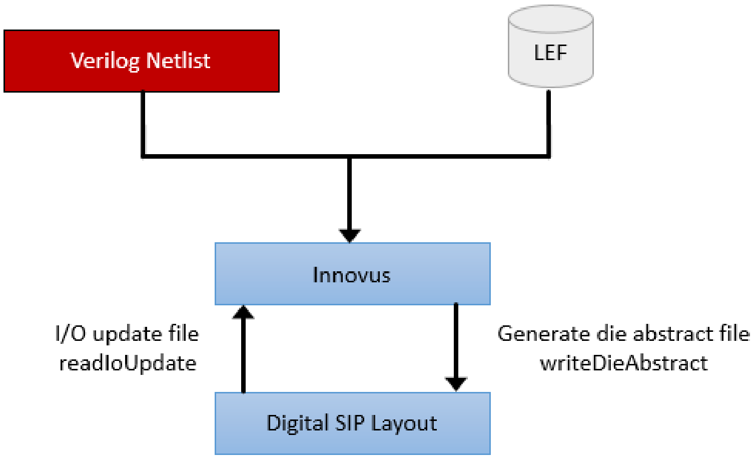
In this case the bump creation and assignment are driven by the package tool and by the constraints from the PCB. This information is fed into Innovus to drive the I/O pad placement. In this flow, the I/O pad and bump planning needs to be done before much of the digital implementation steps, as the bump assignment is driven by package requirements. Once the placement of these I/Os is fixed, the digital implementation in Innovus can start. There will be no need to come back to SiP to check the bump assignment as these should not have been modified during the design closure steps. The downside of this approach is that the package engineer does not have knowledge of the limitations and constraints coming from the logic in the design. It is probable that analog blocks have specific placement constraints, which drive the pad placement and therefore the bump assignment.

Co-design-driven Floorplaning

This is the best method to achieve a quick compromise between digital implementation and the package/PCB board design.

The advantage of this method is that you can move between digital implementation and the package implementation by using SiP Layout and Innovus. This methodology allows you to see IO pads and bumps by die abstract file in Sip Layout System and by I/O file in Innovus, which can help in achieving closure on floorplaning faster.

High-level flow diagram for co-design



In the above figure, the starting point in the flow is Innovus. However, the methodology is flexible enough for the engineer to choose either of the tools, Innovus or SiP Layout, as the starting point. Once the compromise between the two design domains is achieved in terms of routing feasibility, designers in the two domains can work on their own design issues separately as well as in parallel.

The following command in Innovus enables reading of I/O and bump information (placement and assignment information) from SiP layout into Innovus:

- `readIoUpdate`

The package balls in the package file dumped out by the SiP layout in XML format can be correctly displayed in Innovus even when the design is a flip chip design.

After saving the package XML file in the SiP Layout, you can load the package data in the Innovus floorplan view using the `readPackage` command.

Data Preparation

This section describes the data preparation required for a flip chip design.

LEF

Innovus relies on the LEF files to identify the bumps cells and flip chip pads. The cells involved in a flip chip design must have the corresponding keywords.

RDL Layer

The definition of the RDL layer for the flip chip flow is similar to that of other layers in LEF. Here's an example:

```
LAYER metalRDL
    TYPE ROUTING ;
    DIRECTION VERTICAL ;
    PITCH 0.800 ;
    OFFSET 0.100 ;
    HEIGHT 3.7350 ;
    THICKNESS 0.9000 ;
    MINSTEP 0.400 ;
    FILLACTIVESPACING 0.600 ;
    WIDTH 0.400 ;
    MAXWIDTH 12.0 ;
    SPACINGTABLE
    PARALLELRUNLENGTH     0.00    1.50    4.50
    WIDTH    0.00         0.40    0.40    0.40
    WIDTH    1.50         0.40    0.50    0.50
    WIDTH    4.50         0.40    0.50    1.50 ;
    AREA 0.565 ;
MINENCLOSEDAREA 0.565 ;
…
END metalRDL
```

BUMP

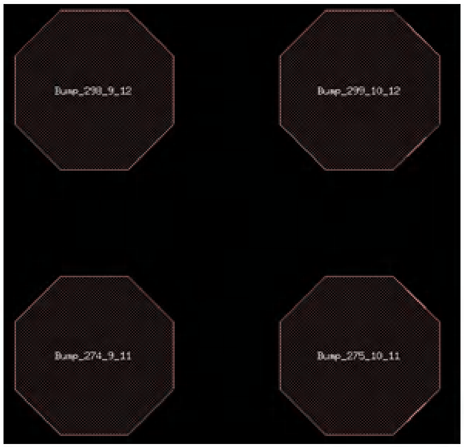
The macro type of BUMP cells need to be `CLASS COVER BUMP`. Here is an example of a bump cell:

```
MACRO BUMPCELL
      CLASS COVER BUMP ;
      ORIGIN 0 0 ;
      SIZE 60.0 BY 60.0 ;
      SYMMETRY X Y ;
      PIN PAD
         DIRECTION INPUT ;
         USE SIGNAL ;
         PORT
             LAYER M9 ;
             POLYGON 17.25 0.0  42.75 0.0  60.0 17.25  60.0 42.75  42.75 60.0  17.25 60.0  0.0 42.75  0.0 17.25 ;
             END
      END PAD

      OBS
            LAYER via89 ;
            POLYGON 17.25 0.0  42.75 0.0  60.0 17.25  60.0 42.75  42.75 60.0  17.25 60.0  0.0 42.75  0.0 17.25 ;
      END
END BUMPCELL
```

In the example above:

- The bump has an octagonal shape of RDL layer. Innovus also supports rectangular bumps.
- OBS can also be defined on the macro of BUMP cells and is honored by the flip chip router (`fcroute`).
- Innovus will display the BUMP shape in GUI.

The following figure shows how bumps are displayed in Innovus.



Note: Polygon shapes are supported from LEF 5.6.

I/O Pad

The macro type of I/O pads need to be `CLASS PAD AREAIO`. Here is an example,

```
MACRO iopad
    CLASS PAD AREAIO ;
    ORIGIN 0.000 0.000 ;

    SIZE 35.000 BY 246.000 ;

    SYMMETRY x y r90 ;

    SITE pad ;

    PIN PAD

      PORT

        CLASS BUMP ;

        ......
```

In the example above:

- The port of PIN PAD has `CLASS BUMP` attribute, which is an optional attribute for I/O pads with `CLASS PAD AREAIO`.

The flip chip flow supports CLASS PAD I/O cell by default.

Note: More information about `CLASS BUMP` is in the section of `CLASS BUMP`.

Hard Macro

If the design has hard macros that have pins to be connected to bumps directly at the top level, these hard macros will keep their original `CLASS BLOCK` and the `PORTS` definition will be enhanced to have a `CLASS BUMP` associated to them. This is an example of how the LEF will look:

```
MACRO Dummy_HM

     CLASS BLOCK ;

     SIZE 2661.1200 BY 696.6000 ;

    ORIGIN 0 0 ;

    SYMMETRY X Y R90 ;

    PIN A1

       DIRECTION OUTPUT ;

       USE SIGNAL ;

       PORT

             CLASS BUMP ;

              LAYER top_layer ;

             RECT 2469.1800 0.0000 2490.1800 83.0000 ;

       END

       PORT

             ... ...

END Dummy_HM
```

In the example above:

- If the hard macro does not have any port with `CLASS BUMP`, it will not be considered for flip chip flow even if the pin A1 is on an I/O net.

Note: More information about `CLASS BUMP` is in the section of `CLASS BUMP`.

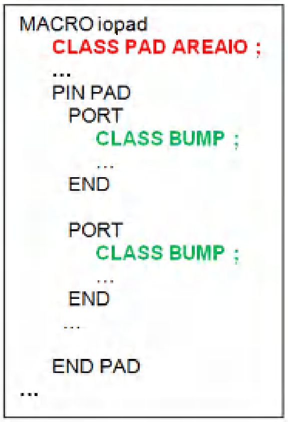
CLASS BUMP Attribute

`CLASS BUMP` is one type of the port. It explicitly indicates that the port is a bump connection point and it can help you to distinguish which ports in a pin are for flip chip flow.

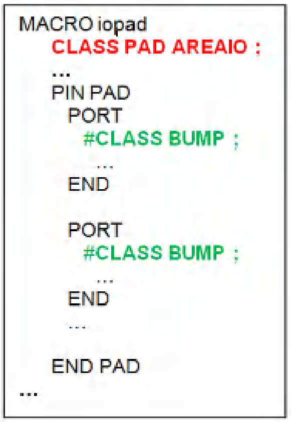
Only `CLASS PAD AREAIO` and `CLASS BLOCK` can have the `CLASS BUMP` attribute.

Note: The flip chip flow supports CLASS PAD I/O cell by default. So the `CLASS BUMP` attribute is allowed to add to CLASS PAD cells and the behavior of a `CLASS PAD` cell with `CLASS BUMP` is the same as that of `CLASS PAD AREAIO` with `CLASS BUMP`.

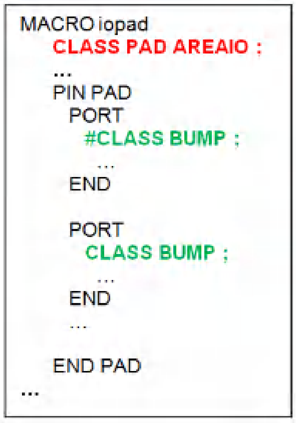
- For `CLASS PAD AREAIO`, this attribute is optional. Following definitions are correct for flip chip flow.



```
MACRO iopad
    CLASS PAD AREAIO ;
    ...
    PIN PAD
      PORT
         CLASS BUMP ;
         ...
      END

      PORT
         CLASS BUMP ;
         ...
      END
      ...

    END PAD
...
```



```
MACRO iopad
    CLASS PAD AREAIO ;
    ...
    PIN PAD
      PORT
        #CLASS BUMP ;
         ...
      END

      PORT
        #CLASS BUMP ;
         ...
      END
      ...

    END PAD
...
```



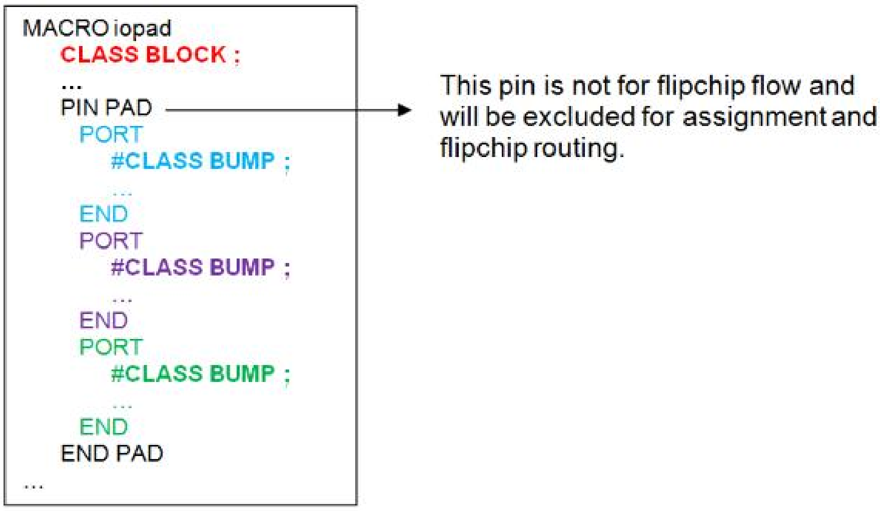
```
MACRO iopad
    CLASS PAD AREAIO ;
    ...
    PIN PAD
      PORT
        #CLASS BUMP ;
         ...
      END

      PORT
         CLASS BUMP ;
         ...
      END
      ...

    END PAD
...
```

- For `CLASS BLOCK`, if the `CLASS BUMP` attribute is not specified, the macro will not be considered for flip chip flow even if there are I/O pins connected to them.

`CLASS BUMP` will affect assignment and routing results.

From PORT Level

- If none of the ports in a pin has the `CLASS BUMP` attribute:
  - For CLASS BLOCK macro, this pin will be excluded from flip chip flow.

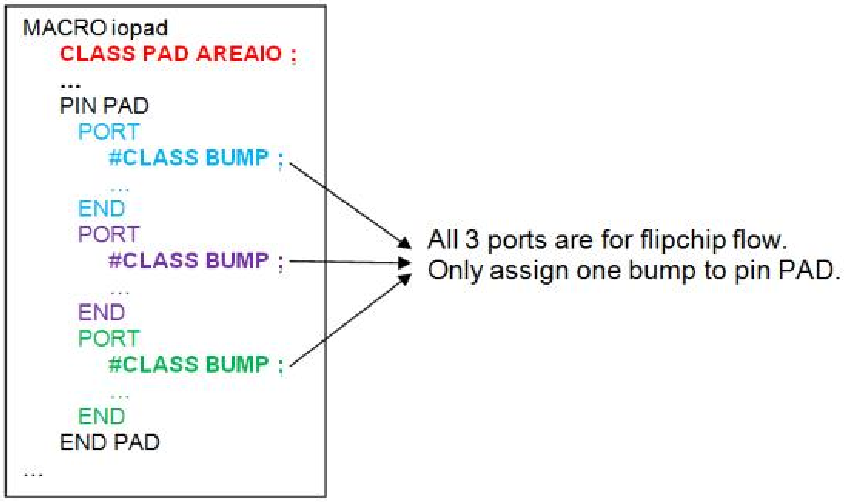


```
MACRO iopad
    CLASS BLOCK ;
    ...
    PIN PAD
      PORT
         #CLASS BUMP ;
         ...
      END
      PORT
         #CLASS BUMP ;
         ...
      END
      PORT
         #CLASS BUMP ;
         ...
      END
    END PAD
...
```

This pin is not for flipchip flow and will be excluded for assignment and flipchip routing.

  - For `CLASS PAD AREAIO` macro, all ports will be considered as one object for assignment and flip chip routing.

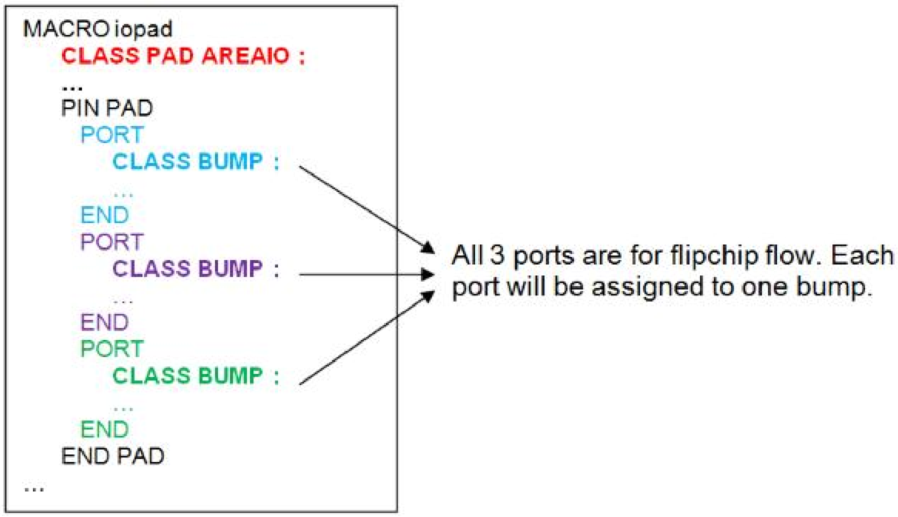
    - Only one bump is assigned to the pin.
    - The flip chip router will pick one port for routing based on its intelligence.



```
MACRO iopad
    CLASS PAD AREAIO ;
    ...
    PIN PAD
      PORT
        #CLASS BUMP ;
        ...
      END
      PORT
        #CLASS BUMP ;
        ...
      END
      PORT
        #CLASS BUMP ;
        ...
      END
    END PAD
...
```

All 3 ports are for flipchip flow.
Only assign one bump to pin PAD.

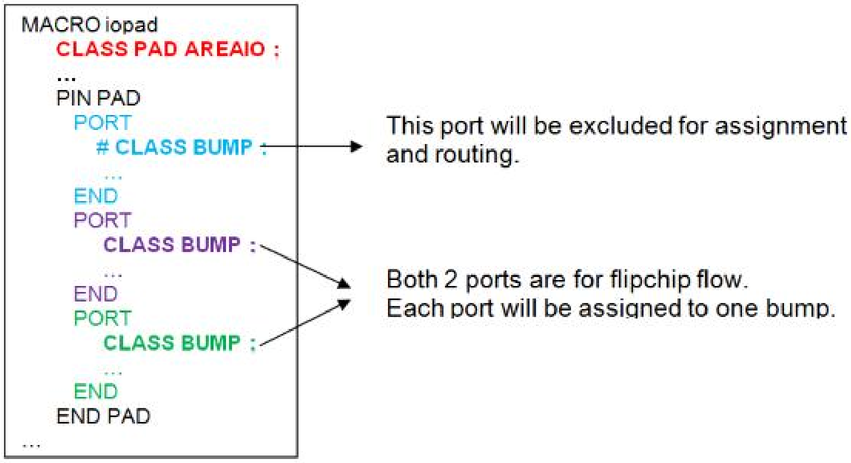
- If all ports with the `CLASS BUMP` attribute are equal in one pin:
  - All ports with CLASS BUMP are applied to both `CLASS PAD AREAIO` and `CLASS BLOCK` macros in terms of assignment and routing.
  - Each port will be assigned to one bump.
  - Every port can be routed to one or multiple bump, which depends on the setup of `fcroute`. You can control the pairing of ports and bumps by adding bump connect target property.



```
MACRO iopad
    CLASS PAD AREAIO ;
    ...
    PIN PAD
      PORT
        CLASS BUMP ;
        ...
      END
      PORT
        CLASS BUMP ;
        ...
      END
      PORT
        CLASS BUMP ;
        ...
      END
    END PAD
...
```

All 3 ports are for flipchip flow. Each
port will be assigned to one bump.

- If some ports in the same pin have the `CLASS BUMP` attribute while some do not:
  - The ports without the `CLASS BUMP` attribute will be excluded for assignment and flip chip routing. This applies to both `CLASS PAD AREAIO` and `CLASS BLOCK` macros. The example below depicts what happens when some ports have the `CLASS BUMP` attribute and some do not.

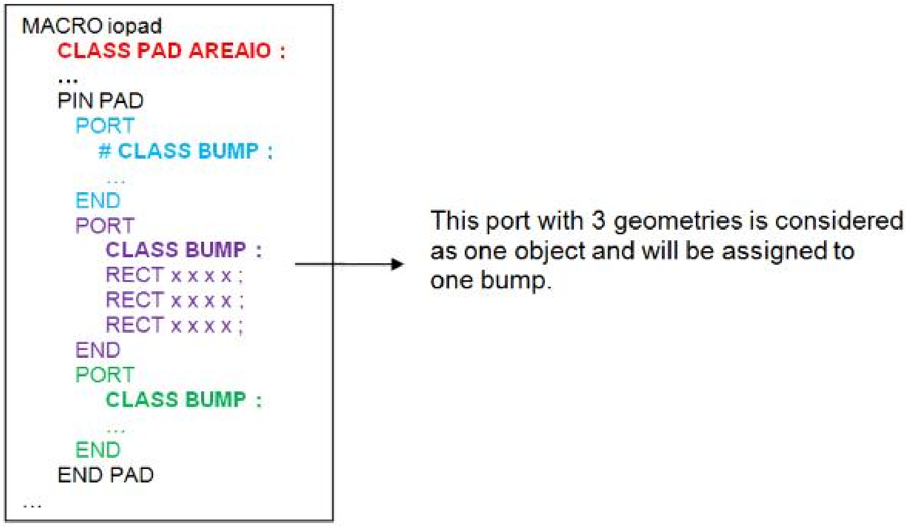


```
MACRO iopad
   CLASS PAD AREAIO ;
   ...
   PIN PAD
     PORT
       # CLASS BUMP ;
       ...
     END
     PORT
        CLASS BUMP ;
        ...
     END
     PORT
        CLASS BUMP ;
        ...
     END
   END PAD
...
```

This port will be excluded for assignment and routing.

Both 2 ports are for flipchip flow.
Each port will be assigned to one bump.

- If a port with the `CLASS BUMP` attribute has multiple geometries or port shapes:
  - It is considered as one object for assignment and flip chip routing. This applies to both `CLASS PAD AREAIO` and `CLASS BLOCK` macros in terms of assignment and routing.
  - The flip chip router will pick one geometry for routing based on its intelligence because there is no mechanism to link a specific geometry in one port to a specific bump. In case of very close geometries, `fcroute` will merge some geometries then choose better one for routing.



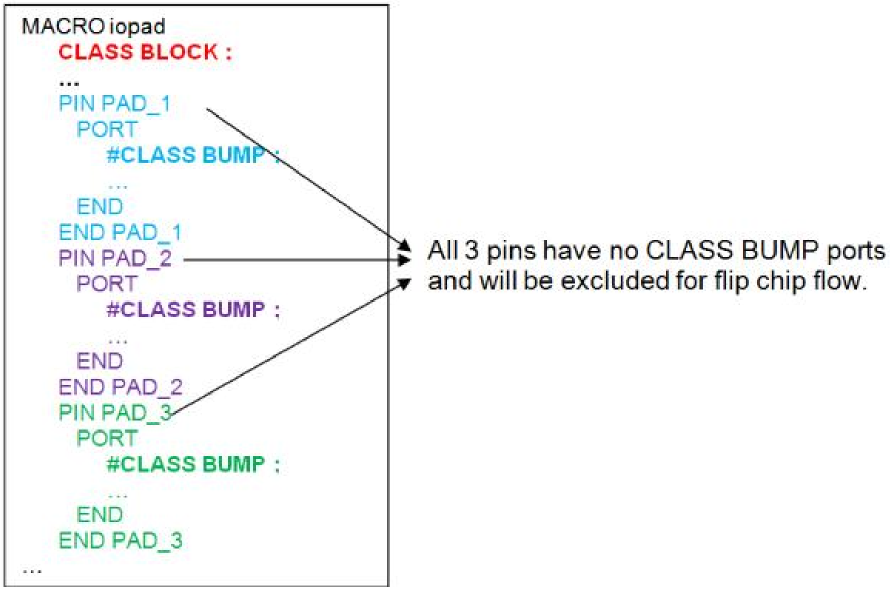
```
MACRO iopad
   CLASS PAD AREAIO ;
   ...
   PIN PAD
     PORT
       # CLASS BUMP ;
       ...
     END
     PORT
        CLASS BUMP ;
        RECT x x x x ;
        RECT x x x x ;
        RECT x x x x ;
     END
     PORT
        CLASS BUMP ;
        ...
     END
   END PAD
...
```

This port with 3 geometries is considered as one object and will be assigned to one bump.

From PIN Level

From PIN level, assume all pins are correctly defined as I/O port in the netlist and need to be connected to bumps.

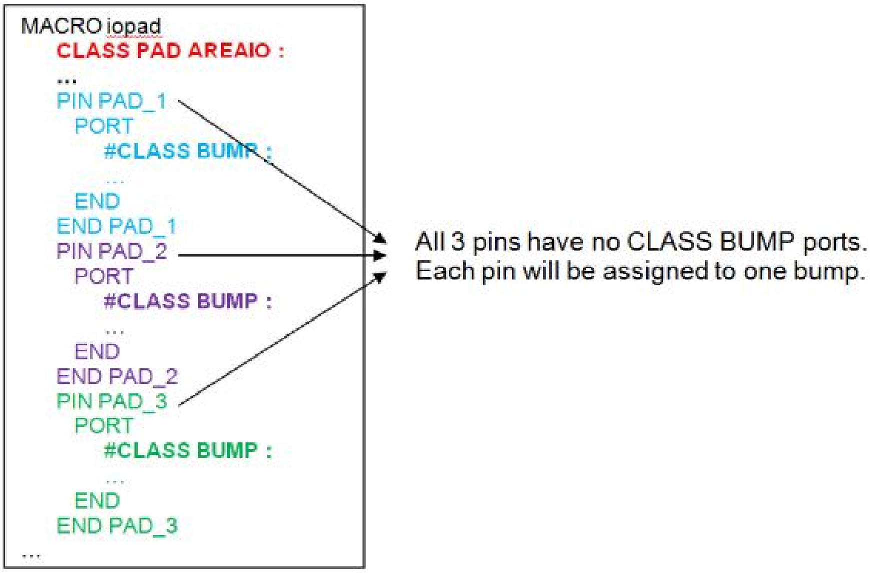
- If none of pins have `CLASS BUMP` ports:
  - For the `CLASS BLOCK` macro, these pins will be excluded from flip chip flow.



```
MACRO iopad
    CLASS BLOCK :
    ...
    PIN PAD_1
      PORT
          #CLASS BUMP :
          ...
      END
    END PAD_1
    PIN PAD_2
      PORT
          #CLASS BUMP :
          ...
      END
    END PAD_2
    PIN PAD_3
      PORT
          #CLASS BUMP :
          ...
      END
    END PAD_3
...
```

All 3 pins have no CLASS BUMP ports and will be excluded for flip chip flow.

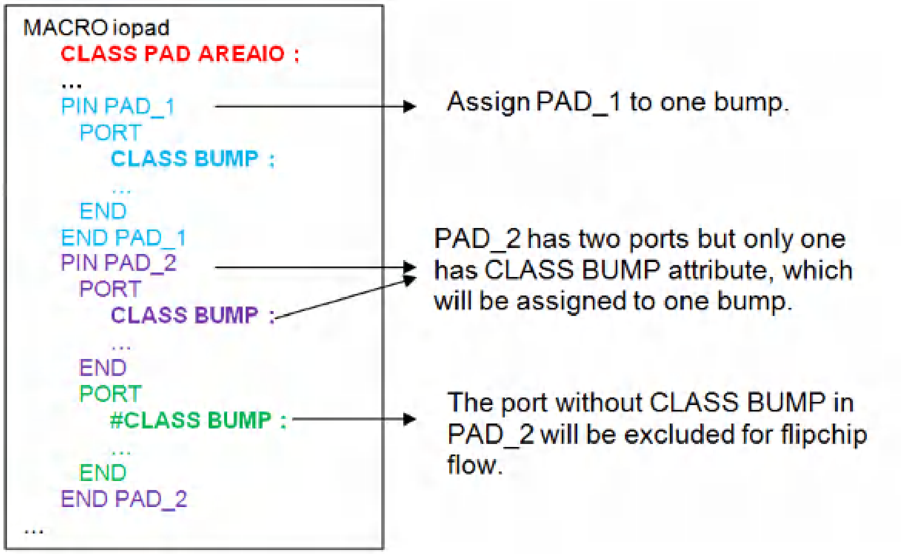
- For the `CLASS PAD AREAIO` macro, each pin will be assigned to one bump.



```
MACRO iopad
    CLASS PAD AREAIO :
    ...
    PIN PAD_1
      PORT
          #CLASS BUMP :
          ...
      END
    END PAD_1
    PIN PAD_2
      PORT
          #CLASS BUMP :
          ...
      END
    END PAD_2
    PIN PAD_3
      PORT
          #CLASS BUMP :
          ...
      END
    END PAD_3
...
```

All 3 pins have no CLASS BUMP ports.
Each pin will be assigned to one bump.

- If all pins with `CLASS BUMP` ports are equal in one macro:
  - All ports with `CLASS BUMP` are applied to both `CLASS PAD AREAIO` and `CLASS BLOCK` macros in terms of assignment and routing.
  - Each port with `CLASS BUMP` in one pin will be assigned to one bump.
  - Every port with `CLASS BUMP` in one pin can be routed to one or multiple bump.



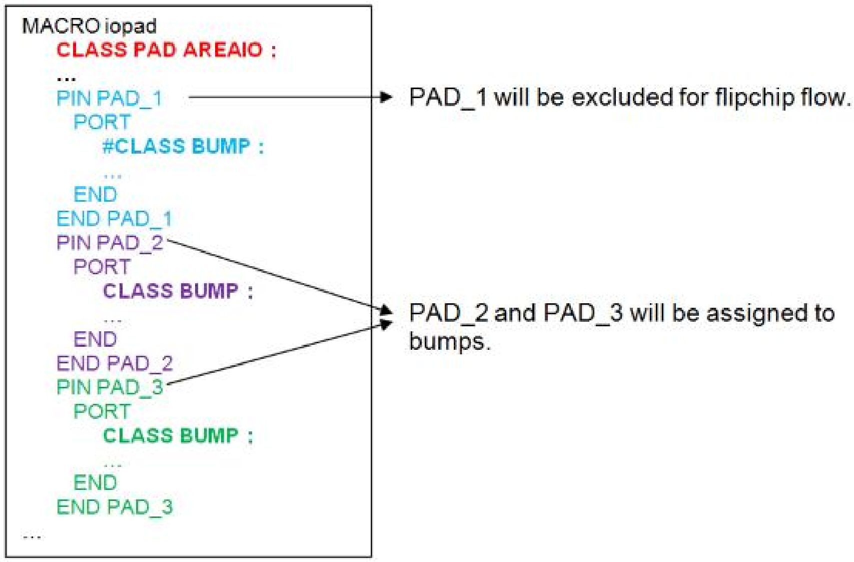
```
MACRO iopad
   CLASS PAD AREAIO :
   ...
   PIN PAD_1
     PORT
        CLASS BUMP :
        ...
     END
   END PAD_1
   PIN PAD_2
     PORT
        CLASS BUMP :
        ...
     END
     PORT
        #CLASS BUMP :
        ...
     END
   END PAD_2
...
```

Assign PAD_1 to one bump.

PAD_2 has two ports but only one has CLASS BUMP attribute, which will be assigned to one bump.

The port without CLASS BUMP in PAD_2 will be excluded for flipchip flow.

- If some pins in a macro have `CLASS BUMP` ports but others do not:
    - The pins in the macro without `CLASS BUMP` ports will be excluded for assignment and flip chip routing. This applies to both `CLASS PAD AREAIO` and `CLASS BLOCK` macros in terms of assignment and routing.



```
MACRO iopad
   CLASS PAD AREAIO :
   ...
   PIN PAD_1
     PORT
        #CLASS BUMP :
        ...
     END
   END PAD_1
   PIN PAD_2
     PORT
        CLASS BUMP :
        ...
     END
   END PAD_2
   PIN PAD_3
     PORT
        CLASS BUMP :
        ...
     END
   END PAD_3
...
```

PAD_1 will be excluded for flipchip flow.

PAD_2 and PAD_3 will be assigned to bumps.

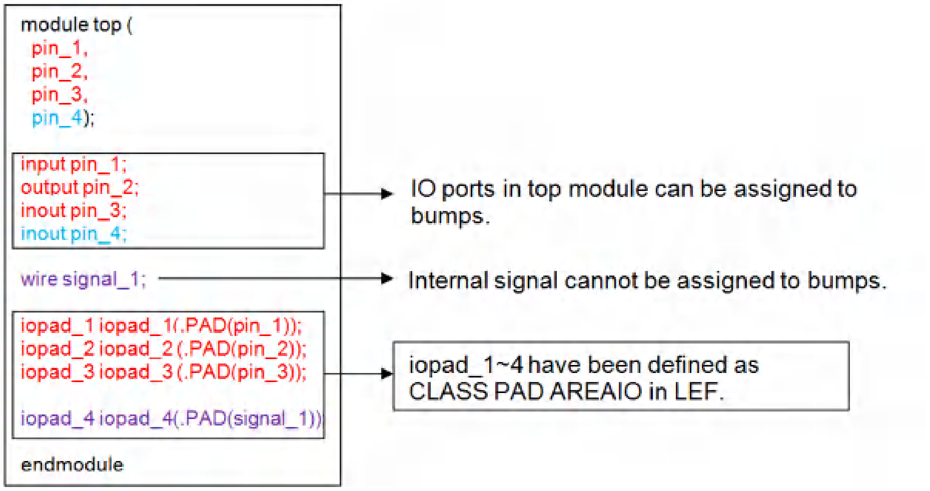
NETLIST

A bump is physical cell, which must not be defined in the netlist. The relationship between bump and pad is constructed during bump assignment which is based on the shortest distance. Then, the flip chip router is used to connect the IO pad to its assigned bump. Defining bumps in the netlist and assigning nets to these bumps is not compatible with Innovus and cannot be handled by Innovus.

IO pads must be defined in netlist.

Signal Pads

To be able to assign signal pads to bumps and connect them to their assigned bumps, the netlist needs to have their connection. This is an example:



```
module top (
  pin_1,
  pin_2,
  pin_3,
  pin_4);

input pin_1;
output pin_2;
inout pin_3;
inout pin_4;

wire signal_1;

iopad_1 iopad_1(.PAD(pin_1));
iopad_2 iopad_2 (.PAD(pin_2));
iopad_3 iopad_3 (.PAD(pin_3));

iopad_4 iopad_4(.PAD(signal_1))

endmodule
```

IO ports in top module can be assigned to bumps.

Internal signal cannot be assigned to bumps.

iopad_1~4 have been defined as CLASS PAD AREAIO in LEF.

In the example above:

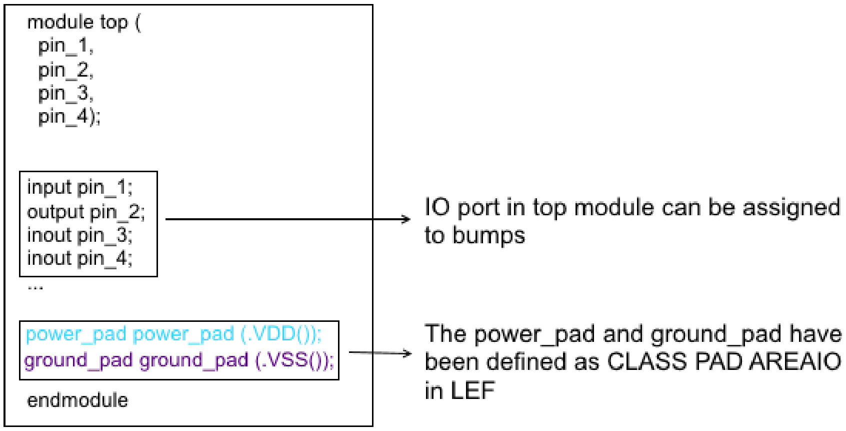
- IO ports pin_1~3 have related IO pads iopad_1~3 with which they can be connected. Therefore, the signal pads iopad_1~3 can be connected to their assigned bumps by the flip chip router.
- IO port pin_4 does not have an IO pad with which to be connected. You may get the following WARNING message after assignment and the flip chip router will not route net pin_4:

  ```
  **WARN: (ENCSP-6014):   I/O pin 'pin_4' does not connect to placed Area I/O instance or hard macro.
  ```

- Internal wire signal_1 cannot be assigned to bumps, so iopad_4 with signal_1 does not have assigned bumps and the flip chip router will not route the net signal_1.

Power and Ground Pads

To be able to assign PG pads to bumps and connect them to their assigned bumps, the netlist could have their connection. This is an example:



```
module top (
  pin_1,
  pin_2,
  pin_3,
  pin_4);

input pin_1;
output pin_2;
inout pin_3;
inout pin_4;
...

power_pad power_pad (.VDD());
ground_pad ground_pad (.VSS());

endmodule
```

IO port in top module can be assigned to bumps

The power_pad and ground_pad have been defined as CLASS PAD AREAIO in LEF

In the example above, you can find the pin VDD of power_pad or the pin VSS of ground_pad does not have any related IO port.

If the initial netlist does not have physical IO pads (for example, power and ground IO pads), you have the option of adding them during floorplanning with `addIoInstance`.

To create the PG connections:

1. Define the power and ground (PG) net names in the power and ground fields in the Innovus `.globals` file:

   `setUserDataValue init_pwr_net {VDD}`

   `setUserDataValue init_gnd_net {VSS}` **Note**: PG nets definition is variable. Some users define these nets as logical nets in the netlist without declaring them in the *.globals file. In such a case, Innovus cannot correctly recognize these nets as special nets and therefore some feature may not work properly.
2. Run the `globalNetConnect` command or read the CPF file (`read_power_intent -cpf` and `commit_power_intent`).

After the PG connection creation, you can assign PG pads to bumps and connect them to their assigned bumps by using the flip chip router.

If the PG connection is not created, the flip chip router will not route these PG nets.

Flip Chip Floorplanning

Bump Creation and Assignment

The flip chip die requires solder bumps to be attached to the package substrate. Bump generation is typically a two-step process -- placement and signal assignment.

Bump Creation

There are multiple ways to create bumps in Cadence tools. You can create a single bump or a bump pattern by using the `create_bump` command.

Usually, bumps are created in a regular pattern with fixed pitch. The `create_bump` command can support the following bump patterns in the chip:

- `-pattern_full_chip`
- `-pattern_side {`*`side width`*`}`
- `-pattern_array {`*`row column`*`}`
- `-pattern_ring` *`width`*
- `-pattern_center {`*`row column`*`}`

Note: All these different bump patterns can coexist in the same floorplan.

During bump creation, the tool will issue a warning and will not create bumps where there are overlaps with other bumps based on bump geometry. But you could specify the `-allow_overlap_control` option to create bumps with overlapping.

The bump creation process does not look into any type of routing blockages or obstruction in hard macro LEFs. The `deleteBumps` command has the options `-overlap_blockages` and `-overlap_macro`, which can be used to clean up the placement of the bumps before signal assignment.

Currently, `verify_drc` will not highlight overlaps between bumps and routing blockages. The bumps need to be assigned (committed) in order for `verify_drc` to flag the short violation. It is a normal procedure to delete the unassigned bumps after a flip chip design implementation.

A bump has four location types as specified below. `create_bump` provides you the capability to specify which location type is used for bump creation:

- Bump cell center
- Bump cell lower left location - This location can be obtained from bump attribute editor or by using the following command:

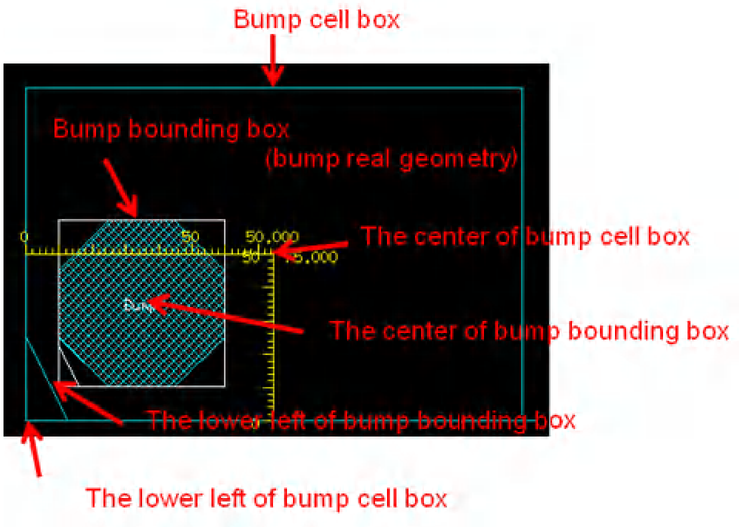
  ```
  dbGet [dbGet top.bumps.name $Bump_name -p].pt
  ```

- Bump geometry (bounding box) center - This location can be obtained with the following command:

  ```
  dbGet top.bumps.bump_shape_center
  ```

- Bump geometry (Bounding box) lower left location - This location can be obtained with the following command:

  ```
  dbGet top.bumps.bump_shape_bbox
  ```


Bump cell box
Bump bounding box
(bump real geometry)
0
50
50,000
50
5,000
The center of bump cell box
Bump
The center of bump bounding box
The lower left of bump bounding box
The lower left of bump cell box

Bump placement supports undo. During bump placement trials, if you click the *Undo* button (or use the `undo` command) after running `create_bump`, the bump floorplan will return to the state it was in before `create_bump`. If you then click the *Redo* button (or use the `redo` command), the bump floorplan will reapply the changes made by `create_bump`.

Similarly, if you click *Undo* after running `deleteBumps`, the changes made by `deleteBumps` are cancelled out and the following bump properties are recovered:

- Name
- Location
- Port number properties
- Fixed status
- Placement status

On clicking *Redo*, the bump floorplan will revert to the status before undo.

Bumps can be created relative to an existing object in the design. To specify the type of the relative object, use the `-relative_type` parameter of `create_bump`. The relative object can be an embedded bump, an inst_pin_port, or a block.

For details, refer to the `-relative_type` parameter in the `create_bump` command description.

Note that when the relative object is a block, you must use the `-relative_block` parameter to specify the block list for creating bumps. `create_bump` then automatically creates bumps at the same location as that of the specified block pin, which must be on the top routing layer of the design.

In addition, you can also use the `-relative_block_constraint` parameter to specify the filename of the block constraint to be used for creating bumps based on block pins. `create_bump` parses the constraint file specified with `-relative_block_constraint` and creates bumps based on the instance pins specified with `-relative_block`. The constraint format for bumps based on a block pin is as follows:

```
BUMP_ON_BLOCK_PIN

    MACRO marco_name_list

    PIN [pin_name_list] [SINGLE|MULT]

    PORT [SINGLE|MULT]

        GEOMETRY_SHAPE RECT/POLY

        GEOMETRY_SHORT_EDGE min_value [:max_value]

        GEOMETRY_LONG_EDGE min_value [:max_value]

    NET net_list_name

END BUMP_ON_BLOCK_PIN
```

Here:

- `MACRO` *`macro_name_list`*
  Specifies the list of the constraint macros (blocks) whose pins are connected to the bump. This parameter is optional. If not specified, the constraint works on all available macros. This parameter supports wildcard matching.
- `PIN [`*`pin_name_list`*`] [SINGLE|MULT]`
  Specifies the list of the constraint pins connected to the bump. This parameter is optional. If not specified, the constraint works on all available pins of available macros. `[SINGLE|MULT]` is optional. `SINGLE` is the default value and indicates that only one bump will be created for each available pin. `MULT` means bumps will be created on every available port of each available pin.
- `PORT [SINGLE|MULT]`
  Specifies the constraint ports connected to the bump. This parameter is optional. If not specified, the constraint works on one available port. `SINGLE` is the default value and indicates that only one bump will be created for each available port. `MULT` means bumps will be created on every available geometry of each available port.
- `GEOMETRY_SHAPE RECT/POLY`
  Specifies the shape of the constraint pin port geometries. This parameter is optional. If not specified, the constraint works on all shapes of pin port geometries.
- `GEOMETRY_SHORT_EDGE` *`min_value`* `[:`*`max_value`*`]`
  Specifies the value or the region of the short edge length of the pin port geometry. If only one value is specified, it means the expected short edge length of the pin geometry equals the specified value. If two values are specified, the expected short edge length is between the specified *`min_value`* and *`max_value`*. The unit is micron. `GEOMETRY_SHORT_EDGE` means the length of the short edge. If the geometry is a polygon, the tool uses the short edge of its bounding box as `GEOMETRY_SHORT_EDGE`.
- `GEOMETRY_LONG_EDGE` *`min_value`* `[:`*`max_value`*`]`
  Specifies the value or the region of the long edge length of the pin port geometry. If only one value is specified, it means the expected long edge length of the pin port geometry equals the specified value. If two values are specified, the expected long edge length is between the specified *`min_value`* and *`max_value`*. The unit is micron. `GEOMETRY_LONG_EDGE` means the length of the long edge. If the geometry is a polygon, the tool uses the long edge of its bounding box as `GEOMETRY_LONG_EDGE`. This parameter is optional. If not specified, the constraints work on all available geometries of pin port.
- `NET` *`net_name_list`*
  Specifies the list of the net names of the constraint pins connected to the bump. This parameter is optional. If not specified, the constraint works on all nets of pins. This parameter supports wildcard matching and supports `@signal`, `@power`, and `@ground`.

Note that:

- `GEOMETRY_SHAPE`, `GEOMETRY_SHORT_EDGE`, and `GEOMETRY_LONG_EDGE` are sub-constraints of the `PORT` constraint.
- In one `BUMP_ON_BLOCK_PIN` constraint, only one `PORT` can be specified.

- You can specify multiple `BUMP_ON_BLOCK_PIN` constraints to set different constraint for different pins. Note that there is no specific priority order among the various constraints defined in a constraint file. There is an OR relationship between the various constraints defined in a file, which means that if the specified constraint file contains multiple `BUMP_ON_BLOCK_PIN` constraints, the tool creates bumps on the block pins that meet any of the constraint in the constraint file.

Here are some examples illustrating the use of the `BUMP_ON_BLOCK_PIN` constraint:

- Example 1: `create_bump -relative_type block` is run without specifying `-relative_block_constraint`. The tool creates a single bump for each block pin on the top layer for the specified blocks.
- Example 2: `create_bump -relative_type block` is run with `-relative_block_constraint 1.const`. The specified constraint file, `1.const`, contains a specific `BUMP_ON_BLOCK_PIN` constraint. The tool parses the `1.const` constraint file and creates bumps only on the block pins that are specified in the constraint file.
- Example 3: `create_bump -relative_type block` is run with `-relative_block_constraint 2.const`. However, the specified constraint file, `2.const`, is empty. In this case,the tool does not create any bumps because no block pin is specified in the constraint file.
- Example 4: `create_bump -relative_type block` is run with `-relative_block_constraint 3.const`. However, the specified constraint file, `2.const`, contains only the constraint start and end statements as given below:

```
BUMP_ON_BLOCK_PIN

END BUMP_ON_BLOCK_PIN
```

  In this case, the tool creates bumps for all block pins on the top layer for the specified blocks. This is because all parameters in the constraint file are assumed to be set at their default values, which means it is equivalent to the following constraint:

```
BUMP_ON_BLOCK_PIN

   MACRO *

   PIN SINGLE

   PORT SINGLE

   NET *

END BUMP_ON_BLOCK_PIN
```

  This is also the same behavior as running `create_bump -relative_type block` without specifying `-relative_block_constraint`.

- Example 5: The following `BUMP_ON_BLOCK_PIN` constraints are defined in the constraint file:

```
BUMP_ON_BLOCK_PIN

  BLOCK abc

  PIN MULT

  PORT MULT

  NET *

END BUMP_ON_BLOCK_PIN


BUMP_ON_BLOCK_PIN

  BLOCK abc

  PIN MULT

  PORT MULT
  GEOMETRY_SHAPE POLY

  NET @POWER @GROUND

END BUMP_ON_BLOCK_PIN
```

The tool creates bumps on all the pin ports because all pin ports are specified in first constraint.

- Example 6: The following `BUMP_ON_BLOCK_PIN` constraints are defined in the constraint file:

```
BUMP_ON_BLOCK_PIN

  BLOCK abc

  PIN MULT

  PORT MULT

  NET @SIGNAL

END BUMP_ON_BLOCK_PIN


BUMP_ON_BLOCK_PIN

  BLOCK abc

  PIN MULT

  PORT MULT

  GEOMETRY_SHAPE POLY

  NET @POWER @GROUND

END BUMP_ON_BLOCK_PIN
```

The tool creates bumps on all signal pin ports and on all PG polygon pin ports.

- Example 7: There are multiple ports of pin *`pinName1`* with rectangle or polygon shapes. If the following constraint is defined in the constraint file, the tool creates top bumps on only the polygon shapes and not on the rectangle shapes:

```
BUMP_ON_BLOCK_PIN

   PIN pinName1 MULT

   PORT MULT

        GEOMETRY_SHAPE POLY

END BUMP_ON_BLOCK_PIN
```

- Example 8: There is a single port of pin *`pinName2`* with multiple rectangle or polygon shapes. If the following constraint is defined in the constraint file, the tool creates top bumps on all polygon and rectangle shapes:

```
BUMP_ON_BLOCK_PIN

   PIN pinName2

   PORT MULT

END BUMP_ON_BLOCK_PIN
```

- Example 9: There is a single port with a single small rectangle. If the following constraint is defined in the constraint file, the tool creates a top bump on the small rectangle:

```
BUMP_ON_BLOCK_PIN

   PIN pinName_list

END BUMP_ON_BLOCK_PIN
```

- Example 10: The constraint file defines the following constraints, one on 10x10 rectangle ports and other on 20x20 polygon ports:

```
BUMP_ON_BLOCK_PIN
   PIN MULT
   PORT MULT
        GEOMETRY_SHAPE RECT
        GEOMETRY_SHORT_EDGE 10
        GEOMETRY_LONG_EDGE 10
END BUMP_ON_BLOCK_PIN


BUMP_ON_BLOCK_PIN
   PIN MULT
   PORT MULT
        GEOMETRY_SHAPE POLY
        GEOMETRY_SHORT_EDGE 20
        GEOMETRY_LONG_EDGE 20
END BUMP_ON_BLOCK_PIN
```

The tool creates bumps on all rectangle shapes within the specified 10x10 constraint and all polygon shapes within the 20x20 constraint.

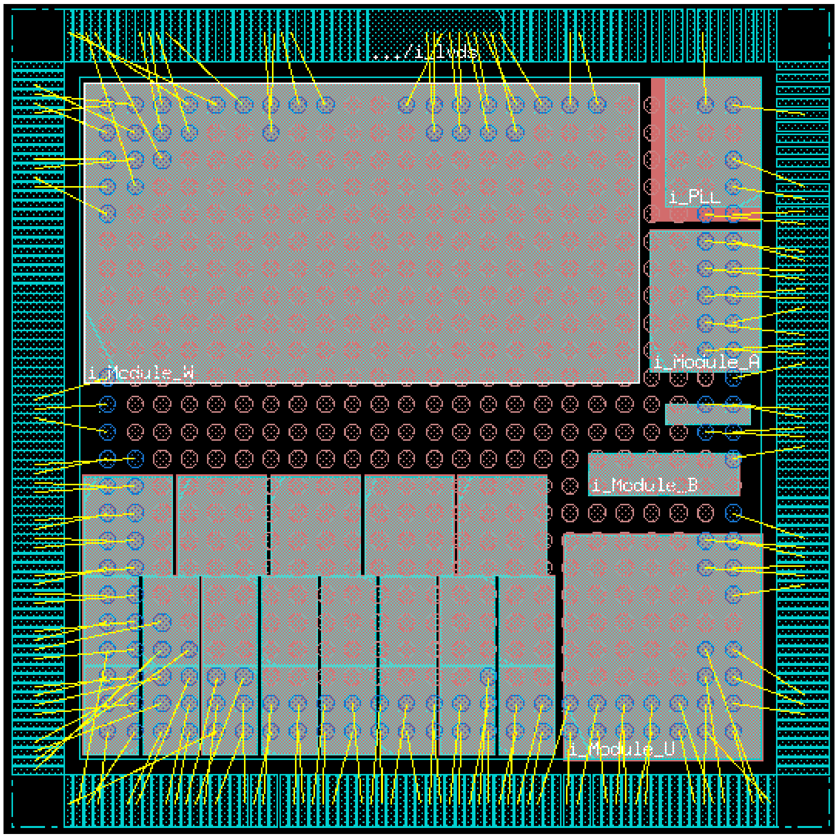
Bump Assignment

After creating bumps, the user can now assign signal and PG bumps.

Signal Assignment

For signal assignment, you can automatically assign the signal bumps to the closest pad IO. This normally gives a suboptimal assignment for routing.

- Automatic signal assignment
  You can use the `assignBump` command for signal assignment. This is fully automatic assignment and it will assign all available signals for flip chip to bumps based on the shortest distance.
- Manual signal assignment with the `assignSigToBump` command.

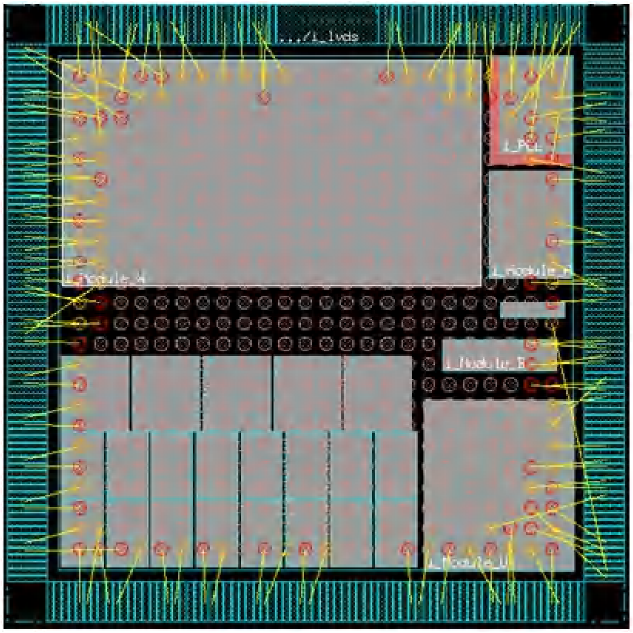


P/G Assignment

For PG assignment, the tool accepts the possibility of associating PG pads to specific bumps along with signal assignment. You can choose to assign PG and signal pads to bumps together or just do the assignment separately .

- Assign PG and signal pads to bump together
  The assignment can be done automatically by using the command `assignBump -pgnet {net_list}`. With this usage, the tool will consider PG nets and signal nets together and distribute appropriate bump resource from the global view based on the shortest distance.
- Assign PG nets only
  The tool also allows you to assign only PG nets to bumps automatically by using the command `assignBump -pgonly -pgnet {net_list}`.

  If you want to have more control over the assignment of the power and ground nets, the tool offers a manual assignment method by using the `assignPGBumps` and `assignSigToBump` command.



Notes :

- Notice how the bumps change color once they are assigned. By default, blue for an assigned signal bump, red for a power bump, and yellow for a ground bump.
- You can change the color of the assignment by supplying the net name and a valid color, which can be obtained from the link http://www.w3.org/TR/SVG/types.html#ColorKeywords.

  This is useful when you want to track the assignment of very specific critical nets. The command to read this text file containing color mapping for bumps is `ciopLoadBumpColorMapFile`. This is an example of the file.

  ```
  Clk green
  ```

  ```
  Address* magenta
  ```

  It can support wildcards for ease of use.
- After assignment, you can use the `viewBumpConnection` command, which can display the connection as a flight line between IO pads and bumps.

  The flight lines of the `viewBumpConnection` command can be automatically redrawn after bump manipulations.

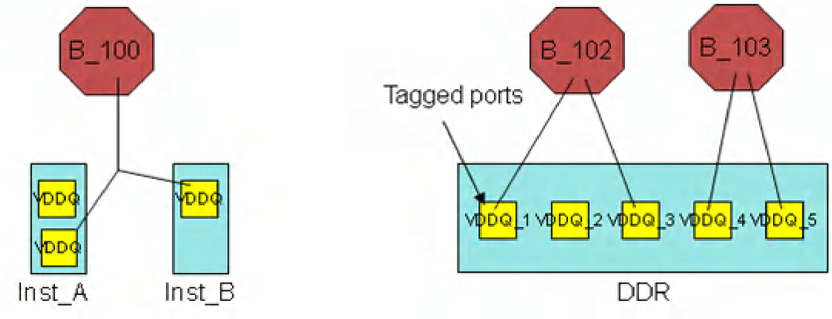
  You can use the `viewBumpConnection -bumpType power` to display the flight lines of only PG connections.

  You can use the `viewBumpConnection -multiBumpsToPad` or `viewBumpConnection -multiPadsToBumps` command to turn on multiple connections. Without these two options, this command will only display one-to-one connections.

Port Numbering Approach in Assignment

For signal assignment, one net is usually related to one bump and the flip chip router can find the connection based on nets; however, for PG assignment, many PG pads are connected to the same net. Therefore, you need to explicitly specify the pairing for PG connection. This means that for a customized pattern of multiple pads to multiple bumps, `fcroute` needs to know exactly which pin or ports need to be routed to which bump.

The port numbering approach allows you to specify the connection between pads to bumps explicitly during the assignment stage.



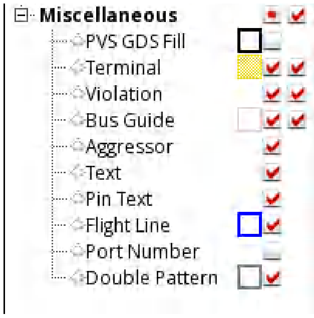
The port numbering feature allows you to specify explicitly:

- Which port is to be connected to a bump
  For example: Bump B_100, Inst_A and Inst_B
- Which pad is to be connected to which bump in case of multiple pads to multiple bumps
  For example: Bump B_102, B_103 and DDR

You can add the CLASS BUMP attribute in LEF onto pins/ports to specify explicitly which pin/port is for flip chip assignment and routing. This attribute has been introduced in the section `CLASS BUMP`.

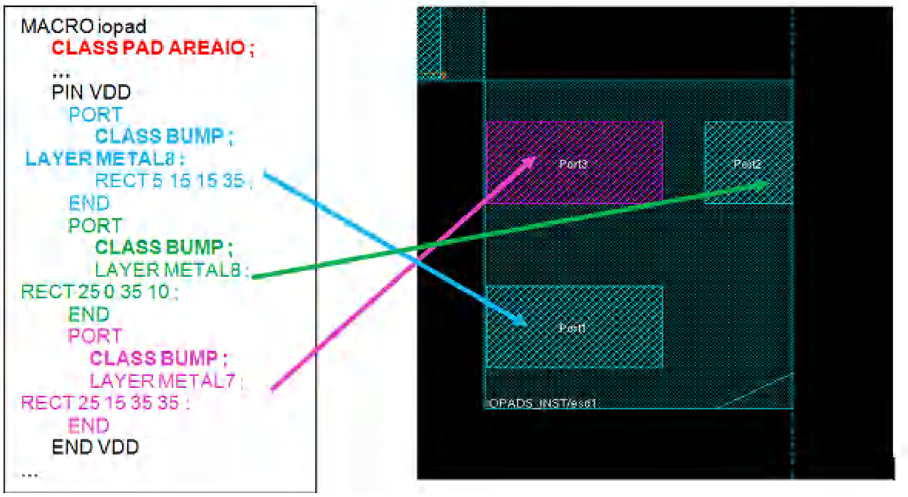
PORTs are numbered uniquely per cell in the Innovus database so that they can be referred during assignment and routing based on instance.

- Numbers are references for ports.
- Regardless of whether or not the CLASS BUMP attribute is specified in LEF, ports will be numbered.
- You can instantly view port numbers by selecting *Miscellaneous* -> *Port Number* on the *Layer Control* bar as shown below:



Miscellaneous		●	✓
PVS GDS Fill	□	□	
Terminal	▨	✓	✓
Violation		✓	✓
Bus Guide	□	✓	✓
Aggressor		✓	
Text		✓	
Pin Text		✓	
Flight Line	□	✓	
Port Number		□	
Double Pattern	□	✓	

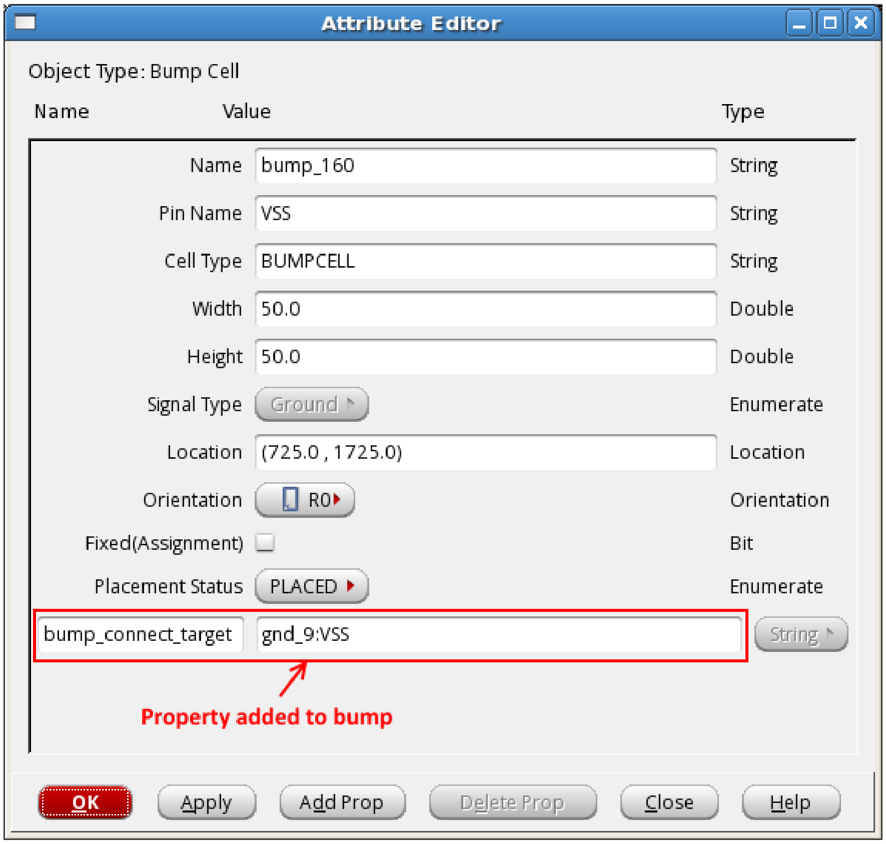
The figure below shows how the tool displays port number based on the LEF definition.



```
MACRO iopad
   CLASS PAD AREAIO ;
   ...
   PIN VDD
     PORT
       CLASS BUMP ;
 LAYER METAL8 ;
       RECT 5 15 15 35 ;
     END
     PORT
       CLASS BUMP ;
       LAYER METAL8 ;
RECT 25 0 35 10 ;
     END
     PORT
      CLASS BUMP ;
      LAYER METAL7 ;
RECT 25 15 35 35 ;
     END
   END VDD
...
```

The assignBump command adds the bump connection target property to bumps by default during bump assignment. You can open the Attribute Editor for a bump to see the added property values after assignment. The value takes the following format:

Property name	`bump_connect_target`
Type	string
Property string format	or *`:pin_name`* or *`:pin_name:port_num`*



Attribute Editor

Object Type: Bump Cell

Name	Value	Type
Name	bump_160	String
Pin Name	VSS	String
Cell Type	BUMPCELL	String
Width	50.0	Double
Height	50.0	Double
Signal Type	Ground	Enumerate
Location	(725.0 , 1725.0)	Location
Orientation	R0	Orientation
Fixed(Assignment)	□	Bit
Placement Status	PLACED	Enumerate
bump_connect_target	gnd_9:VSS	String

Property added to bump

Use the commands listed in the table below for manipulating and saving or restoring properties.

Assignment	Property Manipulation	Property Save/restore	Routing
assignBump	addBumpConnectTargetConstraint	writeFlipChipProperty	fcroute
unassignBump	editBumpConnectTargetConstraint	readFlipChipProperty	
swapSignal	deleteBumpConnectTargetConstraint		
viewBumpConnection	findPinPortNumber		

Notes

- Add `srouteFcroutePadPinTagging TRUE` in the extra configuration file to turn on the port numbering feature for flip chip router (`fcroute`).
- The pin port numbers can be obtained using the `dbGet` command.

```
> dbGet selected.instTerms.cellTerm.pins.?
pin: allShapes class layerShapeShapes objType portNumber shapeViaShapes


> dbGet selected.instTerms.cellTerm.pins.portNumber
2 1 1


> dbGet selected.instTerms.cellTerm.pins.??

allShapes: 0x2adbb65bd330
class: undefined
layerShapeShapes: 0x2adbb65bd330
objType: pin
portNumber: 2
shapeViaShapes: 0x0

allShapes: 0x2adbb65bd308
class: undefined
layerShapeShapes: 0x2adbb65bd308
objType: pin
portNumber: 1
shapeViaShapes: 0x0

allShapes: 0x2adbb65bd420 0x2adbb65bd3f8 0x2adbb65bd3d0 0x2adbb65bd3a8 0x2adbb65bd380 0x2adbb65bd358
class: undefined
layerShapeShapes: 0x2adbb65bd420 0x2adbb65bd3f8 0x2adbb65bd3d0 0x2adbb65bd3a8 0x2adbb65bd380 0x2adbb65bd358
objType: pin
portNumber: 1
shapeViaShapes: 0x0
```

- The pin port numbers can be obtained using the `get_db` command.

```
> get_db selected .pins.base_pin.physical_pins. < Tab>
class  layer_shapes  obj_type  shape_vias

> eval_legacy {dbGet selected.instTerms.cellTerm.pins.portNumber}
2 1 1
> get_db selected .pins.base_pin.physical_pins.*

Object: physical_pin:0x2b9c3f17d6e0
class: undefined
```

```
layer_shapes: layer_shape:0x2b9c43079330
obj_type: physical_pin
shape_vias:

Object: physical_pin:0x2b9c3f17d670
class: undefined
layer_shapes: layer_shape:0x2b9c43079308
obj_type: physical_pin
shape_vias:

Object: physical_pin:0x2b9c3f17d750
class: undefined
layer_shapes: layer_shape:0x2b9c43079420 layer_shape:0x2b9c430793f8
layer_shape:0x2b9c430793d0 layer_shape:0x2b9c430793a8
layer_shape:0x2b9c43079380 layer_shape:0x2b9c43079358
obj_type: physical_pin
shape_vias:
```

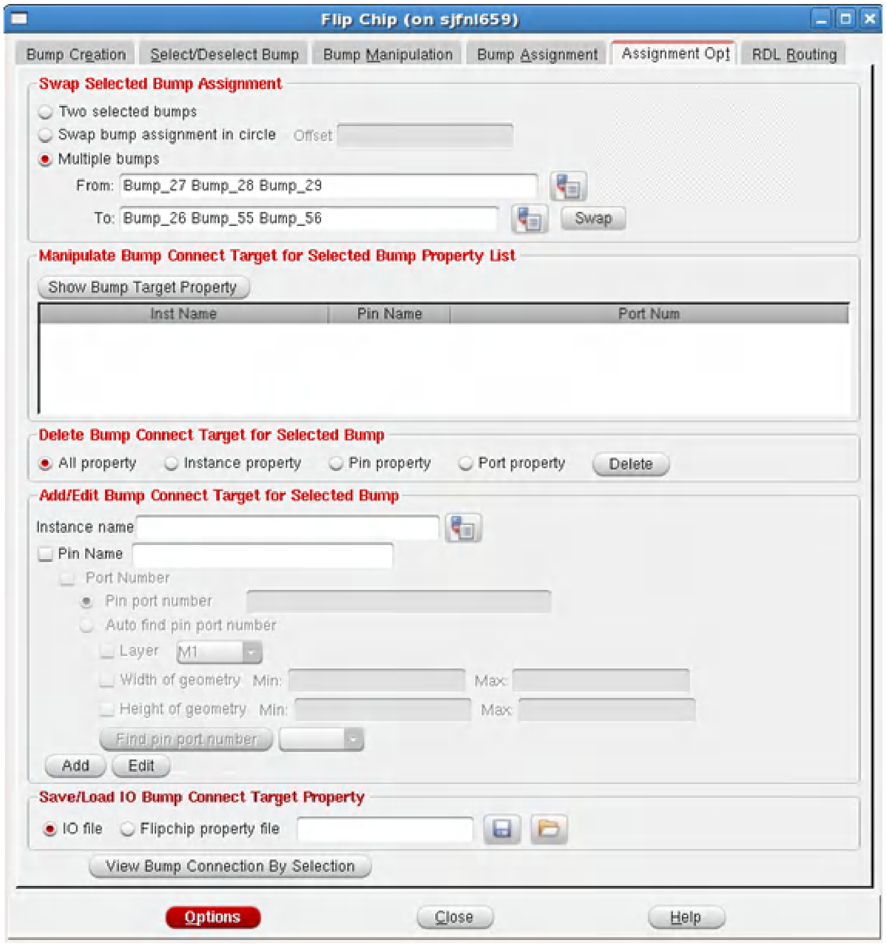
Bump Assignment Optimization

The tool offers two methods for bump assignment optimization.

- Manual optimization
- Use bump assignment constraints

Manual Bump Assignment Optimization

If you need to perform minor ECOs on the assigned bumps, use the `swapSignal` command or choose *Tools ->Flip Chip* and then click the *Assignment Opt* tab in the Flip Chip form.



Swap Selected Bump Assignment

○ Two selected bumps
○ Swap bump assignment in circle Offset
● Multiple bumps
From: Bump_27 Bump_28 Bump_29
To: Bump_26 Bump_55 Bump_56

Manipulate Bump Connect Target for Selected Bump Property List

Delete Bump Connect Target for Selected Bump

● All property ○ Instance property ○ Pin property ○ Port property

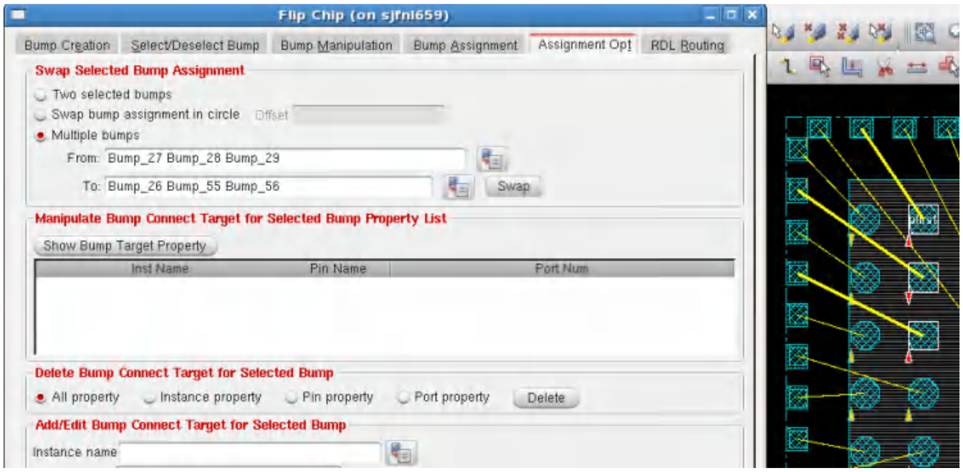
Add/Edit Bump Connect Target for Selected Bump

Instance name
□ Pin Name
□ Port Number
● Pin port number
○ Auto find pin port number
□ Layer M1
□ Width of geometry Min: Max:
□ Height of geometry Min: Max:

Save/Load IO Bump Connect Target Property

● IO file ○ Flipchip property file

The following figure shows the signals before swapping.



Swap Selected Bump Assignment

○ Two selected bumps
○ Swap bump assignment in circle Offset
● Multiple bumps
From: Bump_27 Bump_28 Bump_29
To: Bump_26 Bump_55 Bump_56

Manipulate Bump Connect Target for Selected Bump Property List

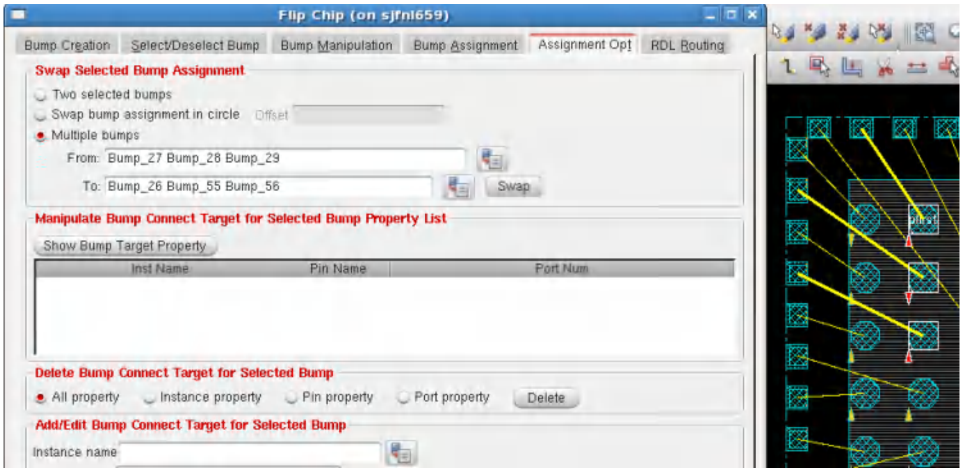
Delete Bump Connect Target for Selected Bump

● All property ○ Instance property ○ Pin property ○ Port property

Add/Edit Bump Connect Target for Selected Bump

Instance name

The following figure shows the signals after swapping.


Flip Chip (on sjfnl659)
Bump Creation
Select/Deselect Bump
Bump Manipulation
Bump Assignment
Assignment Opt
RDL Routing
Swap Selected Bump Assignment
○ Two selected bumps
○ Swap bump assignment in circle
Offset
Show Bump Target Property
Inst Name
Pin Name
Port Num
Delete Bump Connect Target for Selected Bump
○ Instance property
○ Pin property
○ Port property
Delete
Add/Edit Bump Connect Target for Selected Bump
Instance name
Swap
Manipulate Bump Connect Target for Selected Bump Property List

● Multiple bumps

From: Bump_27 Bump_28 Bump_29

To: Bump_26 Bump_55 Bump_56

● All property

Using Bump Assignment Constraints

If you have any constraints for bump assignment, such as constraints to filter ports of pad pin, you can specify these in a constraint file. The bump assignment constraint file is loaded using the `assignBump -constraint_file` option.

At present, the following types of bump assignment constraints are supported:

- `SHARE_FIND_PORT` constraint to filter unnecessary ports
- `ASSIGN_ANALOG_PG_NETS` constraint to specify which signal nets are analog PG nets
- `SHARE_IGNORE_*` and `ASSIGN_IGNORE_*` constraints to exclude instances, macros, pins, or nets for assignment.
- `ASSIGN_PAD2BUMP_RATIO` constraint to specify the pad to bump ratio per net, macro, or instance.

SHARE_FIND_PORT Constraint

The syntax for the `SHARE_FIND_PORT` constraint is as follows:

```
SHARE_FIND_PORT

PIN pin_name_list

MACRO macro_name_list

            LAYERS top_layer[: bottom_layer]

            GEOMETRY_SHORT_EDGE min_value [:max_value]

            GEOMETRY_LONG_EDGE min_value [: max_value]

net_name_list

END SHARE_FIND _PORT
```

The `SHARE_FIND_PORT` constraint can help you find a specific port. However, the property of `CLASS BUMP` port in the LEF file is higher priority than this constraint, which means the `SHARE_FIND_PORT` constraint cannot filter the port with `CLASS BUMP` in the LEF file.

Parameters

- Net name and at least one of the parameters - `LAYERS`, `GEOMETRY_SHORT_EDGE`, and `GEOMETRY_LONG_EDGE` - must be specified to filter unnecessary pin ports.

- `PIN` *`pin_name_list`*
  - Specifies the list of the constraint pins connected with the specified net.
  - Is optional. If not specified, the constraints for the specified net work on all available pin ports for the flip chip design.
  - Supports wildcard matching.
- `MACRO` *`macro_name_list`*
  - Specifies the list of the constraint MACROs whose pins are connected with the specified net.
  - Is optional. If not specified, the constraints for the specified net work on all available MACROs for the flip chip design.
  - Supports wildcard matching.
- `LAYERS` *`top_layer`*`[:` *`bottom_layer`*`]`
  - Specifies the layer or the layer region on which the pin connected with the specified net is located.
  - The name of the layer must support layer ID and layer name in LEF.
- `GEOMETRY_SHORT_EDGE` *`min_value`* `[:`*`max_value`*`]`
  - Specifies the value or the region of the short edge length of the pin geometry connected to the specified net. If only one value is specified, it means the expected short edge length of pin geometry equals the value. Otherwise, the expected short edge length is between *`min_value`* and *`max_value`* .
  - The unit is micron.
  - `GEOMETRY_SHORT_EDGE` means the length of the short edge.
  - `assignBump` ignores the geometries with a short edge length that does not meet the rule.
    For example, pin `VDD` has following different ports:
    - `port1: 5x25`
    - `port2: 15x10`
    - `port3: 5x5`
    - `port4: 10x20`
    - `port5: 15x25`

    If the constraint file specifies `GEOMETRY_SHORT_EDGE 10:20`, then `port2`, `port4` and `port5` can become candidates for assignment.
    On the other hand, if constraint file specifies `GEOMETRY_SHORT_EDGE 5`, `port1` and `port3` can become candidates for assignment.
- `GEOMETRY_LONG_EDGE` *`min_value`* `[:` *`max_value`*`]`
  - Specifies the value or the region of the long edge length of the pin geometry connected to the specified net. If only one value is specified, it means the expected long edge length of pin geometry equals the value. Otherwise, the expected long edge length is between *`min_value`* and *`max_value`* .
  - The unit is micron.
  - `GEOMETRY_LONG_EDGE` means the length of the long edge.
  - `assignBump` ignores those geometries with a long edge length that does not meet the rule
- *`net_name_list`*

  - Specifies the list of the nets. This is a mandatory parameter.
  - Supports wildcard matching and also `@signal`, `@power` and `@ground`.

Here's an example of the `SHARE_FIND_PORT` constraint:

```
SHARE_FIND_PORT

            PIN VDD

MACRO VDD_PAD DDR_VDD_PAD

            LAYERS metalALP

            GEOMETRY_SHORT_EDGE 20:25

            GEOMETRY_LONG_EDGE 15

@power

END SHARE_FIND_PORT
```

If this constraint is specified in the `assignBump` constraint file, only those pins that meet the following requirements are available for bump assignment. `assignBump` ignores all bumps that do not meet these requirements:

- To be connected to the `power` net in the design
- The name in LEF is `VDD`
- Belongs to the macros `VDD_PAD` or `DDR_VDD_PAD`
- On the layer `metalALP`
- Has short edge between `20` and `25` microns and long edge equals `15` microns

ASSIGN_ANALOG_PG_NETS Constraint

The syntax for the `ASSIGN_ANALOG_PG_NETS` constraint is as follows:

```
ASSIGN_ANALOG_PG_NETS

net_name_list

END ASSIGN_ANALOG_PG_NETS
```

Parameters

- *`net_name_list`*
  - Specifies the list of nets that are defined as signal nets in LEF or netlist but are actually analog PG nets.
  - Supports wildcard matching.

Notes:

- The nets specified with the `ASSIGN_ANALOG_PG_NETS` constraint are still signal nets but are used for auto power/ground assignment. `assignBump` treats these nets in the same way as other normal power/ground nets.
  a. These nets can be specified as arguments of the `assignBump -pgnet {net_list}` or `-exclude_pgnet {net_list}` options.
  b. These nets are considered when the `assignBump -pginst {instance_list}` is used.
  c. These nets are excluded from signal assignment of `assignBump`.
- `assignBump` does not modify the property of specified nets in `ASSIGN_ANALOG_PG_NETS`. Therefore, after assignment, the bumps

that are assigned with the specified nets in `ASSIGN_ANALOG_PG_NETS` must be signal bumps instead of power/ground ones.

SHARE_IGNORE_* and ASSIGN_IGNORE_* Constraints

The `SHARE_IGNORE_*` and `ASSIGN_IGNORE_*` constraints can be used to exclude instances, macros, pins, or nets for assignment. The syntax of the the constraint varies depending on what is being excluded.

Excluding Instances

Use the following syntax to specify list of instances to be excluded from assignment:

```
SHARE_IGNORE_INSTANCE

instance_name_list

END SHARE_IGNORE_INSTANCE
```

Here, *`instance_name_list`* specifies the list of instances that are to be excluded during assignment. It supports wildcards.

Excluding Macros

Use the following syntax to specify list of macros to be excluded from assignment:

```
SHARE_IGNORE_MACRO

macro_name_list

END SHARE_IGNORE_MACRO
```

Here, *`macro_name_list`* specifies the list of macros that are to be excluded during assignment. It supports wildcards.

Excluding Instance Pins

```
ASSIGN_IGNORE_INSTANCE_PIN

instance_name:pin_name

...

END ASSIGN_IGNORE_INSTANCE_PIN
```

Here *`instance_name:pin_name`* :

- Specifies which pin of the specified instance is to be excluded during assignment.
- Supports multiple parameters.
- Supports wildcards.
- Does not allow any blank spaces before or after "`:`"

Excluding Macro Pins

```
ASSIGN_IGNORE_MACRO_PIN

macro_name:pin_name

...

END ASSIGN_IGNORE_MACRO_PIN
```

Here *`macro_name:pin_name`* :

- Specifies which pin of the specified macro is to be excluded during assignment.
- Supports multiple parameters.
- Supports wildcards.
- Does not allow any blank spaces before or after ":"

Excluding Nets

```
ASSIGN_IGNORE_NET

net_name_list

END ASSIGN_IGNORE_NET
```

Here, *`net_name_list`* specifies the list of nets that are to be excluded during assignment. It supports wildcards.

ASSIGN_PAD2BUMP_RATIO Constraint

The syntax for the `ASSIGN_PAD2BUMP_RATIO` constraint is as follows:

```
ASSIGN_PAD2BUMP_RATIO
TOLERANCE net_name integer
PGNET net_name ratio
PGMACRO macro:pin ratio
PGINST  inst:pin ratio
END ASSIGN_PAD2BUMP_RATIO
```

The `ASSIGN_PAD2BUMP_RATIO` constraint specifies the pad to bump ratio per net, macro, or instance.

Parameters

Parameters for Ratio Greater than 1

The following parameters work with a pad to bump ratio greater than 1.

- `TOLERANCE` *`net_name integer`*
- `PGNET` *`net_name ratio`*

When the pad to bump ratio is greater than 1, `assignBump` calculates the number of groups of multiple PG ports to multiple bumps and the maximum number of ports in each group based on the specified ratio per net.

Assume the total number of ports is *`total_port`*, and the ratio is *`port_num:bump_num`*.

- If *`total_port`* is an integer multiple of the ratio:

  ```
  Number of groups = total_port÷(port_num/bump_num) + TOLERANCE
  ```

  Else:

  ```
  Number of groups = [total_port÷(port_num/bump_num)] + 1 + TOLERANCE
  ```

- If *`port_num`* is an integer multiple of *`bump_num`*:

  ```
  Maximum number of ports in each group = port_num ÷ bump_num
  ```

  Else:

  ```
  Maximum number of ports in each group = [port_num ÷ bump_num] + 1
  ```

For example, if there are 5 pads with VDD and the VDD pin of each pad has only one port and the pad to bump ratio is 3:2. Then:

```
Number of groups = Round off {5÷ (3/2) + 0} = 4
```

```
Maximum number of ports in each group = Round off {3/2} = 2
```

This feature tries to provide the best assignment result in following conditions:

- The number of groups remains unchanged.
- In each group, the number of ports must be no more than the maximum number of ports.

Following is the description of the parameters for ratio greater than 1:

- `TOLERANCE` *`net_name integer`*
    - Specifies the tolerance value for specific nets. It only supports positive integers. With this option, the number of pad/port groups will be counted by the tolerance value.
    - The default value is zero.
    - It works only for `PGNET` *`net_name ratio`*
    - *`net_name`* supports wildcard matching.
- `PGNET` *`net_name ratio`*
    - Specifies the ratio for specific nets.
    - *`net_name`* supports wildcard matching.
    - The ratio must be more than 1. If you specify a ratio less than 1, `assignBump` issues an ERROR message and ignores the ratio.
    - If the ratio is not in the simplest form, the tool will simplify it and give a WARNING.
    - For different PG nets, define the ratio per net separately. If you specify more than 1 ratio for the same net, `assignBump` issues a WARNING message and chooses the last specified ratio for the net.

    Example1
    If the ratio for both VDD and VDD0 is 2:1 and the ratio for VSS is 3:2, the syntax is as below:

    ```
    PGNET VDD 2:1    #The ratio for VDD is 2:1
    PGNET VDDO 2:1   #The ratio for VDDO is 2:1
    PGNET VSS 3:2    #The ratio for VSS is 3:2
    PGNET VSS 3:2    #As this is the second and last ratio for VSS, assignBump     #issues a WARNING message
    and use it as the ratio of     #net VSS.
    PGNET VSSO 1:2   #As the ratio for VSSO is less than 1, assignBump issues
                     #an ERROR message and ignores it.
    ```

    Example 2

    ```
    PGNET * 2:1      #The ratio for all pg nets specified in the assignBump command is 2:1
    ```

    Example 3
    In the ratio.const constraint file, following is specified:

    ```
    PGNET VDD 4:2
    PGNET VSS 4:2
    ```

    Run the following command:

    ```
    assignBump -constraint_file ratio.const -pgnet {VDD VSS}
    ```

    a. The tool first simplifies the ratio to 2:1.
    b. 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:

  `Number of groups = [`*`total_port÷(port_num/bump_num)`*`]+1 +0 = [5÷(2/1)]+1 = [2.5] + 1 = 2 + 1 = 3`

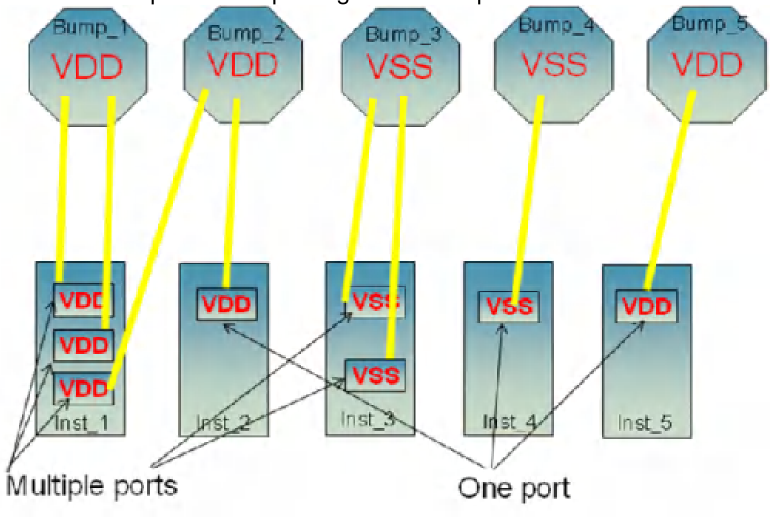
- For VSS, as the total number of ports with VSS (3) is not an integer multiple of *port_num* (2), therefore:

  `Number of groups = [`*`total_port÷(port_num/bump_num)`*`]+1 +0 = [3÷(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÷bump_num`*` = 2/1 = 2`

  The multi-PG pad to bump assignment is depicted below.



Parameters for Ratio Less than 1

The following parameters work with a ratio less than 1.

- `PGMACRO` *`macro:pin ratio`*
  - Specifies the ratio for the specific MACRO cell.
  - *`macro:pin`* supports wildcard.
  - The ratio must be less than 1. Otherwise, `assignBump` issues an ERROR message and ignores it.
  - If you specify more than 1 ratio for the same pin, `assignBump` issues a WARNING message and chooses the last specified ratio for it.
  - Choose the out-most geometry as the target and record the port number into bump_connect_target.

    Example 1

    ```
    PGMACRO DDR1:VDD   1:2   #The ratio for VDD pin in DDR1 is 1:2
    PGMACRO DDR2:VDD   1:2   #The ratio for VDD pin in DDR2 is 1:2
    PGMACRO DDR2:VDD   1:3   #As this is the second ratio and last ratio for
                             #VDD pin in DDR2, assignBump issues a WARNING
                             #message and uses it as the ratio of VDD pin in DDR2.
    PGMACRO DDR3:VSS   2:1   #As the ratio is more than 1,
                             #assignBump issues an ERROR message and ignores it.
    ```

    Example 2

    ```
    PGMACRO * 1:2            #The ratio for all pg pins in all macros of the
    ```

```
                        #instances specified in the assignBump command
                        #is 1:2.
PGMACRO DDR1:* 1:2      #The ratio for all pg pins in DDR1 is 1:2.
PGMACRO *:VDD 1:2       #The ratio for VDD pin in all macros of the
                        #instances specified in the assignBump command
                        #is 1:2.
```

- `PGINST` *`inst:pin ratio`*
    - Specifies the ratio for the specific instance.
    - *`inst:pin`* supports wildcard.
    - The ratio must be less than 1. Otherwise, `assignBump` issues an ERROR message and ignores it
    - If you specify more than 1 ratio for the same pin, `assignBump` issues a WARNING message and chooses the last specified ratio for it.
    - Choose the out-most geometry as the target and record the port number into bump_connect_target.

      Example 1

      ```
      PGINST inst1:VDD   1:2   #The ratio for VDD pin in inst1 is 1:2
      PGINST inst2:VDD   1:2   #The ratio for VDD pin in inst2 is 1:2
      PGINST inst2:VDD   1:3   #As this is the second and last ratio for VDD pin
                               #in inst2, assignBump issues a WARNING message
                                #and uses it as the ratio of VDD pin in inst2.
      PGINST inst3:VSS   2:1   #As the ratio is more than 1, assignBump issues
                              #an ERROR message and ignores it.
      ```

      Example 2

      ```
      PGINST * 1:2             #The ratio for all pg pins of all instances
                              #specified in the assignBump command is 1:2.
      PGINST inst1:* 1:2       #The ratio for all pg pins in inst1 is 1:2.
      PGINST *:VDD 1:2         #The ratio for VDD pin of all instances
                              #specified in the assignBump command is 1:2.
      ```

If there is some conflict over ratio, the priority order is `PGINST > PGMACRO > PGNET` as shown in the following example.

```
ASSIGN_PAD2BUMP_RATIO
PGNET VDD 2:1           #The ratio for VDD is 2:1.
PGMACRO DDR:VDD 1:2      #If the VDD pin in DDR is connected to VDD,
                        #the ratio for VDD pin in DDR is 1:2 as the priority
            #of PGMACRO is higher than PGNET.
PGINST  inst:VDD 1:3     #The MACRO cell of inst is DDR. As the priority of PGINST
#is higher than PGMACRO, the ratio for VDD pin in inst
#is 1:3
PGNET VSS 3:1PGINST  inst1:VSS 1:3   #If the VSS pin in inst1 is connected to VSS, the ratio
#for VSS pin in inst1 is 1:3 as the priority of
#PGINST is higher than PGNET.
END ASSIGN_PAD2BUMP_RATIO
```

The usage of `ASSIGN_PAD2BUMP_RATIO` is as below:

- It turns on the feature of pads to bumps assignment by ratio.
- If used with `-pgnet`, `ASSIGN_PAD2BUMP_RATIO` works on only the specified PG nets.

- If used with `-exclude_pgnet`, `ASSIGN_PAD2BUMP_RATIO` ignores the constraints for the specified excluded PG nets.
- If used with `-pginst`, `ASSIGN_PAD2BUMP_RATIO` works on only the instances specified by `PGMACRO` and `PGINST`.
- If used with `-pgnet` and `-pginst`, `ASSIGN_PAD2BUMP_RATIO` works on only the specified PG nets of the instances specified by `PGMACRO` and `PGINST`.
- If used with `-exclude_pgnet` and `-pginst`, `ASSIGN_PAD2BUMP_RATIO` works on the PG nets of the instances specified by `PGMACRO` and `PGINST`, with the exception of the specified excluded PG nets.
- If the objects specified by `-pgnet`, `-exclude_pgnet` and `-pginst` are not included in `ASSIGN_PAD2BUMP_RATIO`, `assingBump` issues a warning message and uses 1 as the ratio value.
- If not used with `-pgnet`, `-exclude_pgnet` or `-pginst`, `ASSIGN_PAD2BUMP_RATIO` ignores all the defined constraints.

Viewing Flip Chip Flightlines

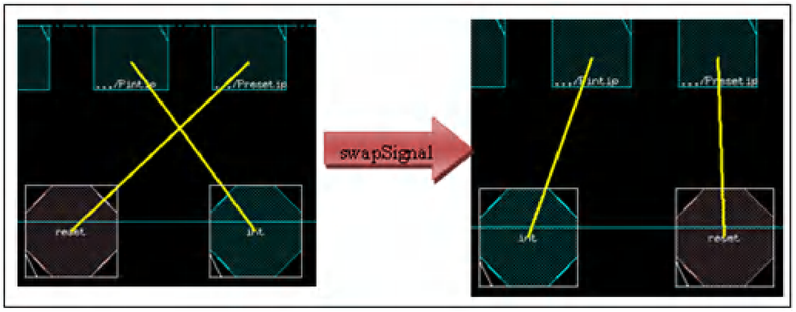
If In flip chip designs, flightlines are used extensively to interact with the design. Flip chip flightlines are different from the normal blue flightlines in Innovus and can be displayed using the `viewBumpConnection` command.

`viewBumpConnection` provides the following features to make it easy for you to use flightlines:

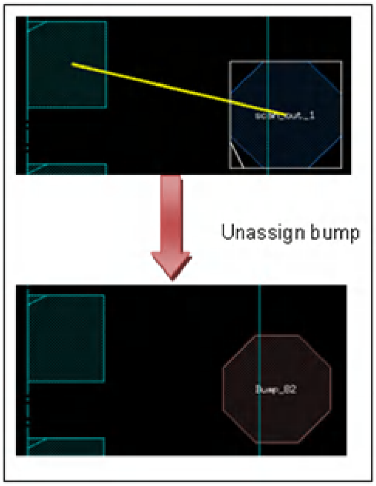
Automatic Redraw Feature

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

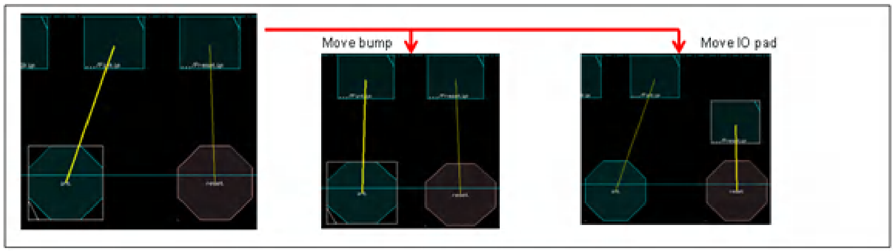
- Bump assignments are swapped using `swapSignal`: Flightlines of the selected bumps are swapped to reflect the manipulation.



- Bumps are unassigned using `unassignBump`: Flightlines of specified bumps are removed.



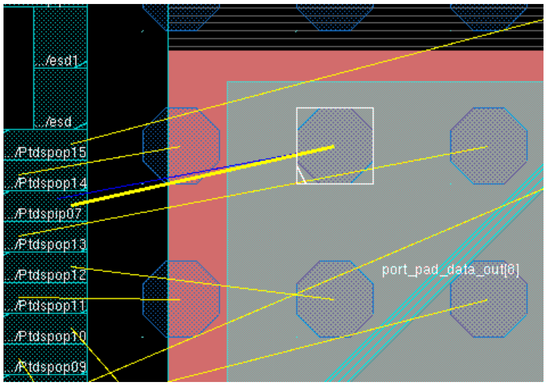
- A bump, IO pad, or block is moved: Flightlines are redrawn to reflect the new location.



Selection-Based Highlighting

When you select an object, the corresponding flightlines are highlighted in bold.

- When a bump or IO pad is selected, its corresponding flightline is highlighted in bold.
- When multiple bumps/IO pads are selected, all their flightlines are highlighted in bold.
- If a block with multiple IO pins is selected, all its flightlines are highlighted in bold.
- When the objects are deselected, the corresponding flightlines return to non-bold status.

1. Run `viewBumpConnection` to display all flip chip flightlines. 2. Click on an object to highlight its flightline in bold.	

Colored Flightlines

By default, all flip chip flightlines are displayed in yellow. You can use the `viewBumpConnection -honor_color` option to color these flightlines based on either bump type or the nets to which the bumps are assigned:

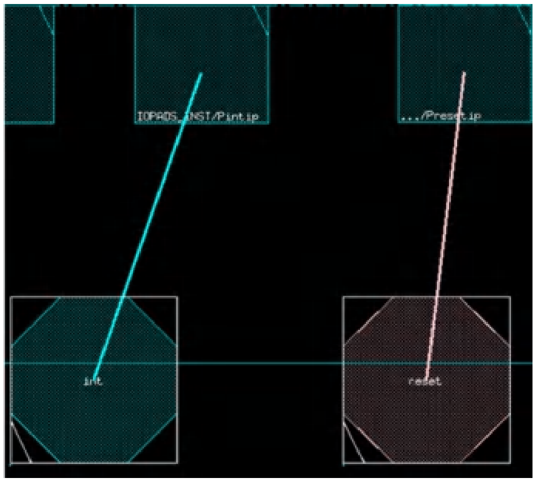
- To color flightlines by bump type, simply run `viewBumpConnection -honor_color`. The tool displays flightlines using the default colors of the bumps:
  - Blue for signal bumps
  - Red for power bumps
  - Yellow for ground bumps
- To color flightlines based on the nets to which they are assigned, you must:
  1. Define bump color settings in a bump color map file using the following format:

     ```
     net_name color_name
     ```

     Example:

     ```
     int cyan
     reset pink
     ```

  2. Load the bump color file using the `ciopLoadBumpColorMapFile` command.
  3. Run `viewBumpConnection -honor_color`.



For bumps whose nets are not defined in the bump color file, the default colors are used as follows – blue for signal bumps, red for power bumps, and yellow for ground bumps. A flightline has the same color as its bump.

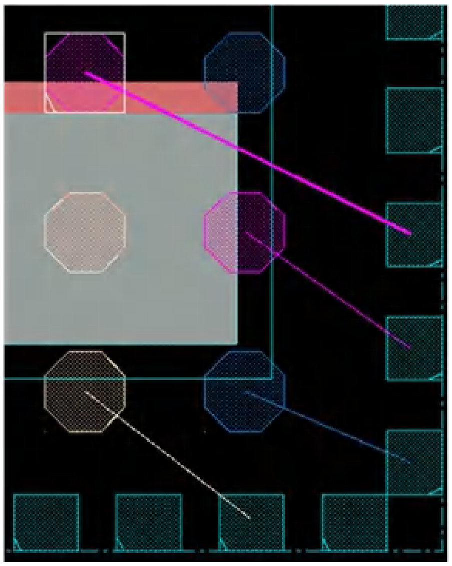
Object-Specific Flightlines

You can easily view connections for specific objects, such as bumps, nets, and IO instances, using the following `viewBumpConnection` parameters:

- `-bump {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.
- `-net {net_list}:` Use this parameter to view connections of specified nets.
- `-selected:` Use this parameter to view connections of selected bumps or IO pads in bold. If a block with multiple IO pins is selected, all its flightlines are displayed in bold.

For example, the following command displays the flightlines for the `port_pad_data_out[10]` net, the `Bump_29` bump, and the `IOPADS_INST/Ptdspop07` instance. It also displays in bold the flightline for the selected bump:

```
viewBumpConnection \
-net {port_pad_data_out[10]} \
-bump Bump_29 \
-io_inst IOPADS_INST/Ptdspop07 \
-selected \
-honor_color
```



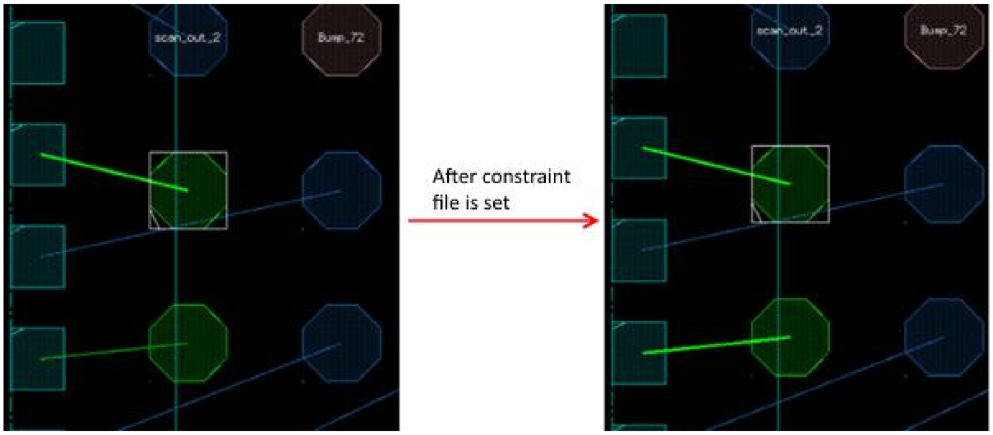
DIFFPAIR-Based Highlighting

Flip chip flightlines now honor the DIFFPAIR constraints specified in the flip chip router constraint file. This means that when you select any one bump or IO pad that is part of a DIFFPAIR constraint, the tool highlights all flightlines of that DIFFPAIR in bold.

For example, suppose the flip chip router constraint file, `diffpair.const`, has the following setting:

```
SHARE_DIFFPAIR

    port_pad_data_in[15]

    port_pad_data_in[13]

END SHARE_DIFFPAIR
```

Now after `setFlipChipMode -constraintFile diffpair.const` is set, the flightlines for the DIFFPAIR are highlighted in bold when any one bump or IO pad of the DIFFPAIR is selected:



Currently, you cannot turn off normal flightlines to focus on DIFFPAIR flightlines. However, you can use `viewBumpConnection -net` *`net_list`* as a workaround. Here, *`net_list`* specifies nets of the DIFFPAIR. This way, you can display only the flightlines for the DIFFPAIR and turn off all other flightlines.

viewBumpConnection Display Rules

A PAIR constraint is frequently used in the constraint file to define the connection between bump and IO pad for power/ground net explicitly. The flip chip flightlines, viewed using `viewBumpConnection`, honor the `SHARE_PAIR` constraint and display the connection between bump and IO pad for power/ground net.

If you want flip chip flightlines to honor the `SHARE_PAIR` constraint:

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

   ```
   SHARE_PAIR

   net_name pad_name_list bump_name_list

   END SHARE_PAIR
   ```

2. Specify the constraint file using the following command:

   ```
   setFlipChipMode -constraintFile file_name
   ```

`viewBumpConnection` displays the connection between bump and IO pad based on the net. The following examples use net `VDD` to explain the `viewBumpConnection` display rules:

Example 1: Design has only one bump, `bump_vdd`, for `VDD`

- If `bump_vdd` does not have the port number property or the PAIR constraint, `viewBumpConnection` auto-pairs it and displays the connections for `VDD`.
- If `bump_vdd` has only the `SHARE_PAIR` constraint, `viewBumpConnection` displays the connection specified by the `SHARE_PAIR` constraint.
- If `bump_vdd` has only the port number property, `viewBumpConnection` displays the connection specified by the port number.
- If `bump_vdd` has both the port number property and the `SHARE_PAIR` constraint, `viewBumpConnection` displays only the connection specified by port number.

Example 2: Design has multiple bumps for `VDD`

- If none of the bumps has either the port number property or `SHARE_PAIR` constraint, `viewBumpConnection` auto-pairs them and displays the connections for `VDD`.
- If all of the bumps have only the `SHARE_PAIR` constraint, `viewBumpConnection` displays the connections specified by the `SHARE_PAIR` constraint.
- If all of the bumps have only the port number property, `viewBumpConnection` displays the connections specified by the port number.
- If one bump has both the port number property and the `SHARE_PAIR` constraint as in the following example:
  1. `Bump_1`: Does not have either the port number property or the `SHARE_PAIR` constraint.
  2. `Bump_2`: Only has a `SHARE_PAIR` constraint:
     ```
     SHARE_PAIR
     VDD pad_2 Bump_2
     END SHARE_PAIR
     ```
  3. `Bump_`3: Only has the port number property.
  4. `Bump_4`: Has both the `SHARE_PAIR` constraint and the port number property:
     ```
     SHARE_PAIR
     VDD pad_4 Bump_4
     END SHARE_PAIR
     ```
  5. Bump_5 and Bump_6: Only have a `SHARE_PAIR` constraint:
     ```
     SHARE_PAIR
     VDD pad_1 pad_5 pad_6 Bump_5 Bump_6
     END  SHARE_PAIR
     ```
  6. `Bump_7` and `Bump_8`: Have a `SHARE_PAIR` constraint as below and `Bump_8` also has the port number property:
     ```
     SHARE_PAIR
     VDD pad_7 pad_8 Bump_7 Bump_8
     END SHARE_PAIR
     ```

  In this case, `viewBumpConnection` displays flightlines as follows:

  - Does not display the connection of `Bump_1`.
  - Displays the connection between `pad_2` and `Bump_2` based on the `SHARE_PAIR` constraint.

- Displays the connection of `Bump_3` based on the port number property.
- Displays the connection of `Bump_4` based on only the port number property.
- Displays the connections based on `viewBumpConnection` auto-pairing results for `pad_1`, `pad_5` and `pad_6` with `Bump_5` and `Bump_6`.
- Displays the connection of `Bump_8` based on only the port number property. For `Bump_7`, ignores the `SHARE_PAIR` constraint and does not display the connection of `Bump_7`. This means if one or more bumps in a `SHARE_PAIR` constraint have the port number property, the other bumps without port number property in this `SHARE_PAIR` constraint are ignored and their connections are not displayed.

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

Scenario #1

```
SHARE_PAIR VDD pad_1 Bump_1
END SHARE_PAIR
```

```
SHARE_PAIR
VDD pad_1 Bump_2 # This constraint works
END SHARE_PAIR
```

If you want to pair pad_1 to Bump_1 and Bump_2, use the following syntax:

```
SHARE_PAIR
VDD pad_1 Bump_1 Bump_2
END SHARE_PAIR
```

Scenario #2

```
SHARE_PAIR
VDD pad_1 Bump_1
END SHARE_PAIR
```

```
SHARE_PAIR
VDD pad_2 Bump_1 # This constraint works
END SHARE_PAIR
```

If you want to pair `pad_1` and `pad_2` to `Bump_1`, use the following syntax:

```
SHARE_PAIR
VDD pad_1 pad_2 Bump_1
END SHARE_PAIR
```

Long Pin Connection Display

Normally, `viewBumpConnection` uses the center of the pin of an IO instance for the connection display between the IO instance and a bump. However, in case of a long pin with multiple connections, using the the center of the long pin geometry for the connection display may be confusing when you check the connection. By default, `viewBumpConnection` displays a long pin connection with the correct location, instead of the center of the pin, for both signal and power pins.

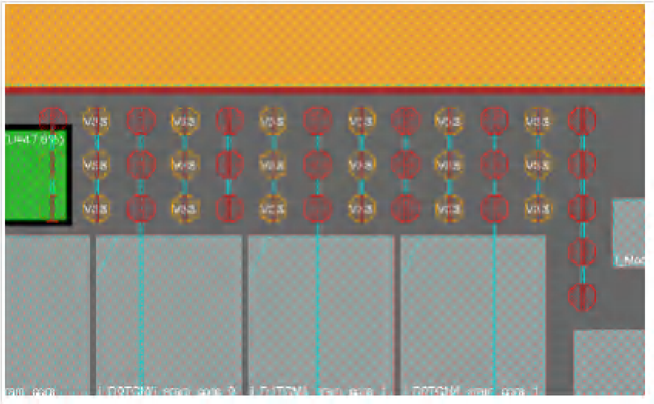
Power Planning in Flip Chip Design

The general power planning step does not differ from a non-flip-chip design. However, there are flow recommendations worth mentioning.

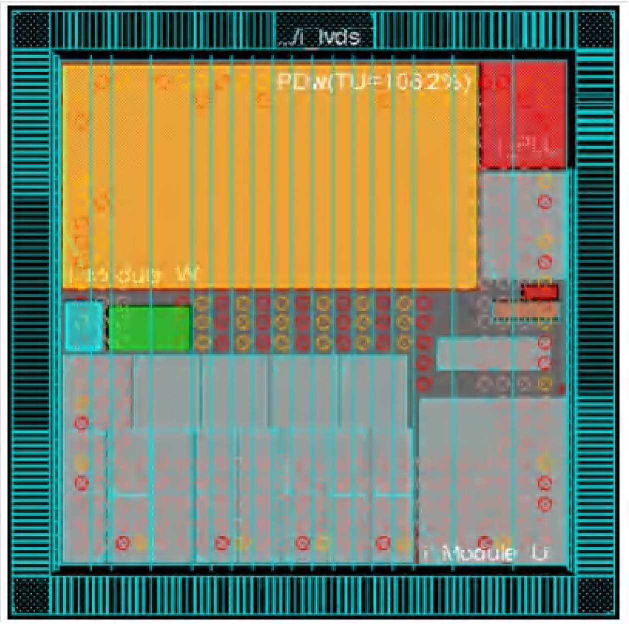
It is advisable NOT to use the RDL layer before pad to bump routing, as the flip chip router is very sensitive to routing obstructions. The power routing, i.e. stripe generation, with the RDL layer can be done after routing of I/O pads to bumps.

The `addStripe` command provides two options for easier flip chip design support. These options are:

- `-over_bumps`
  - This option will create a power stripe over PG bumps.
  - The stripe generation will STOP at the end of a valid PG bump.
  - The image below shows power stripes in metal 8 generated over bumps in metal 9. The command will automatically drop the generated vias if bumps do not have OBS to prevent via under bump as defined in LEF.



- `-between_bumps`
  - This option will create a power stripe between PG bumps.
  - The stripes will NOT be created in areas where there are no PG bumps.
  - The image below shows the result of running the `addStripe` command with the `-between_bumps` option. Notice the missing stripes where there are no PG bump.



These options allow for the possibility of providing an area (rectangular or rectilinear) for stripe routing. This is very flexible and improves the usability of the command when there are multiple power domains in your design.

At this stage, it is not recommended to route/create the connection between power bumps and power routing/power pads using the RDL layer. This step will be performed during or as a post step of the signal routing.

Notes:

- The `addStripe` command creates stripes over unassigned bumps. It is recommended that after pad and bump optimization, you delete all the spare bumps. These shorts will not be highlighted as violations during verification as the bump is not assigned.
- If you are planning to create power/ground stripes in the same layer as the RDL routing, it is recommended to do this after the signal routing by `fcroute`.

RDL Routing

Introduction

The flip chip router (`fcroute`) is mainly used for routing the nets between bumps and I/O PADs.

The `fcroute` command supports two types for routing:

- `fcroute -type power`
  - The `power` type can connect PG bumps to rings or stripes.
  - It supports only the `m anhattan` routing style.
  - It does not honor the settings specified with `setFlipChipMode`.
- `fcroute -type signal`
  - The `signal` type can connect bumps to I/O pads.

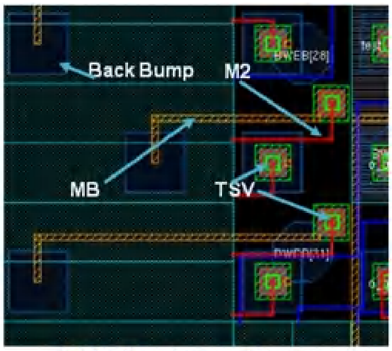
  - The router under this type supports two routing styles, `manhattan` and `45DegreeRoute`. `45DegreeRoute` is the default routing style.
  - It honors all the settings specified with `setFlipChipMode`.
  - Used with the command `setFlipChipMode -connectPowerCellToBump true`, it can route PG bumps to I/O pads.
  - Used with the command `setFlipChipMode -prevent_via_under_bump true`, it can prevent via generated on the bump.

The command `fcroute -type signal` accepts two design styles, peripheral IO (pio) and area IO (aio). However, it is not limited by design styles, which means that you can use the aio mode for some specific purpose in a pio design.

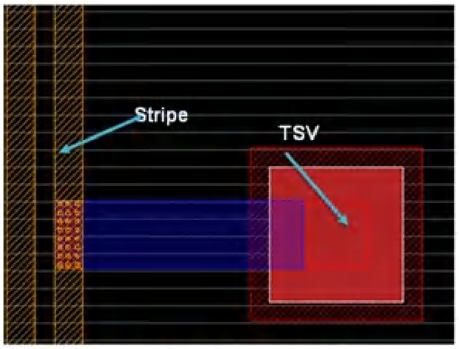
Note: In this document, the aio or pio mode is used to describe which design style is specified for `fcroute`.

Here are some examples of `fcroute` usage:

- `fcroute -type signal -designStyle aio`
  This command will use only the detail router to finish the connections between bumps to IO pads and it routes nets one by one, so it is incremental.
- `fcroute -type signal -designStyle pio`
  This command will first use the global router to distribute all routing resource and then use the detail router to finish the connections between bumps to IO pads followed by a post processing step to finalize routing.
- `fcroute -type signal -designStyle pio -area {`*`x1 y1 x2 y2`*`} -incremental`
  `fcroute` partially supports area-based routing in the specified coordinates for the area in which the net is routed. The `-incremental` option is required for area-based `pio` mode routing.
- `fcroute` can support TSV routing, which can route TSV to bumps/stripes/instance pins. The related commands are listed as below:
  - Connect TSV to bumps: `fcroute -type signal -connectTsvToBump`
    As TSV has front-side and back-side bumps, `fcroute` should route them separately.
    Add `srouteExcludeBumpType bump_cell_name` in the extra configuration file to specify which bumps should be excluded for routing by `fcroute`.
    - When connecting TSV to the back-side bump, use the extra configuration file to exclude the front bump.
    - When connecting TSV to the front-side bump, exclude the back-side bump.
  - Connect TSV to I/O pads: `fcroute -type signal -connectTsvToPad`
  - Connect power TSV to stripes: `fcroute -type power -connectTsvToRingStripe`
    The TSV routing result is shown as below:



TSV to IO and back side bump routing



TSV to power stripe bump routing

The flip chip router (`fcroute`) is an intelligent and predictable RDL router and it can help you evaluate floorplan change based on the quick flip chip routing result. In the initial floorplan stage, you can use the following general setting to get a quick routing result:

To set routing layers

- Use the `setFlipChipMode` command to set routing layers:

  ```
  setFlipChipMode -layerChangeBotLayer bot_layer_name -layerChangeTopLayer top_layer_name
  ```

- Use the `fcroute` command to set.

  ```
  fcroute -layerChangeBotLayer bot_layer_name -layerChangeTopLayer top_layer_name
  ```

To set routing width

- Use the `setFlipChipMode` command to set.

  ```
  setFlipChipMode -routeWidth value
  ```

- Use the `fcroute` command to set.

  ```
  fcroute -routeWidth value
  ```

You can use the following commands with the above setting:

- Use `setFlipChipMode` for settings and then run `fcroute` for routing:
  a. Specify the routing setting for `fcroute` as follows:

     ```
     setFlipChipMode -layerChangeBotLayer bot_layer_name -layerChangeTopLayer top_layer_name -routeWidth value
     ```

  b. Get the routing setting for `fcroute` to make sure all the settings are as expected:

     ```
     getFlipChipMode
     ```

  c. Run `fcroute` in the `pio` mode as an example:

     ```
     fcroute -type signal -designStyle pio
     ```

- Use `fcroute` for both setting and routing:
  a. Get routing setting before routing to make sure all the settings are as expected:

     ```
     getFlipChipMode
     ```

  b. Run `fcroute` in the `pio` mode with general settings as an example:

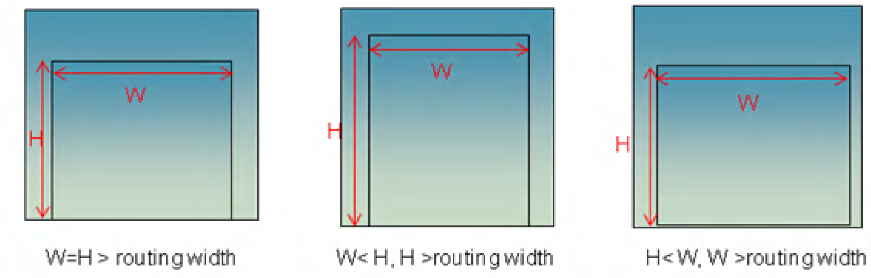
     ```
     fcroute -type signal -designStyle pio -routeWidth value -layerChangeBotLayer bot_layer_name -
     layerChangeTopLayer top_layer_nam
     ```

Note: The differences between the settings for `setFlipChipMode` and `fcroute` are:

- All the settings set by `setFlipChipMode` can be saved and restored by `saveDesign`/`restoreDesign`. These settings can work on `fcroute` if they are not reset or overwrite by the options of `fcroute`.
- All the settings set by `fcroute` cannot be saved after running `fcroute`. In addition, these settings work only when they are used in `fcroute`.

Generally, `fcroute` can well handle typical pin shape shown as below:

- Only one geometry for `fcroute`
- Width/height of pin geometry will not be very different from each other.
- At least one of width/height is larger than the routing width.



The above is the basic function of `fcroute`, and it is greatly extended by constraints and extra configuration options for customized requirements.

- All constraints are specified in a text file, which can be specified as input to the command `fcroute -constraintFile` or `setFlipChipMode -constraintFile`.
- All extra configuration options are specified in a text file, which can be specified as input to the command `fcroute -extraConfig` or `setFlipChipMode -extraConfig`.

Some useful constraints and extra configuration options for `fcroute` will be described in detail in the next two sections.

Useful Constraints for Flip Chip Routing

The flip chip router (`fcroute`) provides you the capability to specify routing constraints in a text file, which can be specified as input to the command `fcroute -constraintFile` or `setFlipChipMode -constraintFile`.

Note:

- All length constraints take micron as the unit.
- All resistance constraints take ohm as the unit.
- All values can of floating type.

The routing constraints supported by `fcroute` are as follows:

Global Constraints

You can specify general constraints affecting all the nets in the design. These general constraints need to be declared at the top of the file. The accepted constraints are as follows:

- `WIDTH` *`value`*
  - It specifies the global width for all the nets.
  - The unit is micron.
  - Both the `aio` and `pio` modes support this constraint.
  - Both `manhattan` and `45DegreeRoute` support this constraint.
  - If you also specify routing width in the command `setFlipChipMode` or `fcroute`, `fcroute` will use the width specified in the command.
  - It can also be applied into the `NETS` constraint, which is a local constraint. In this case, the specified width value only works on the associated nets in the `NETS` constraint.
- `WIDTHRANGE` *`min_value:max_value`*
  - It works only for the `pio` mode.

  - It specifies a width range and allows `fcroute` to optimize wire width by considering `MAXRES` constraints.
  - The unit is micron.
  - It can also be applied into the `NETS` constraint, which is a local constraint. In this case, the specified width range only works on the associated nets in the `NETS` constraint.
- `MAXRES` *`value`*
  - It specifies the maximum resistance allowed to all the specified nets.
  - The unit is ohm.
  - Only the `pio` mode supports this constraint.
  - Both `manhattan` and `45DegreeRoute` support this constraint.
  - The command `fcroute` can report the resistance with the setting `srouteFCReport` *`res_file_name`* in the extra configuration file.
  - It can also be applied into the `NETS` constraint, which is a local constraint and the specified resistance only works on the associated nets in the `NETS` constraint.
- `SPACING` *`value`*
  - It specifies the distance in microns between the net routed by the flip chip router and all other routes.
  - The unit is micron.
  - Both the aio and pio modes support this constraint.
  - Both `manhattan` and `45DegreeRoute` support this constraint.
  - The router uses the spacing value to limit the effect of coupling capacitance on the total capacitance of the net.
  - It can also be applied into the `NETS` constraint, which is a local constraint and the specified spacing only works on the associated nets in the `NETS` constraint.
- `PIOLAYERCHANGE PAD`
  - It turns on the layer change feature.
  - Only the `pio` mode supports this constraint.
  - Both `manhattan` and `45DegreeRoute` support this constraint.
  - Different setting for routing layers will have the different behaviors as shown in the following examples. Assume the top RDL is `TOP_RDL` and the second top RDL is `2nd_RDL`.
    - Use `TOP_RDL` as much as possible and `2nd_RDL` is used only when a single layer cannot finish routing in case of cross-over.
      ```
      -layerChangeBotLayer TOP_RDL -layerChangeTopLayer TOP_RDL
      ```
    - Freely change layers so that the tool will use the layer resources by its intelligence.
      ```
      -layerChangeBotLayer 2nd_RDL -layerChangeTopLayer TOP_RDL
      ```

SPLIT Constraint

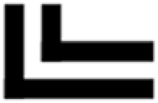
The `SPLIT` constraint is used to split wires when the routing width is larger than `MAXWIDTH` defined in LEF or the value specified by the user. The `SPLIT` constraint works for the `aio` and pio modes. In addition, it supports both both `manhattan` and `45DegreeRoute`. The syntax of the `SPLIT` constraint is as follows:

```
SPLITSTYLE RIVER|MESH

SPLITWIDTH value

SPLITGAP value

SPLITKEEPTOTALWIDTH TRUE|FALSE
```

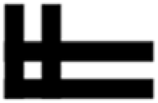
Parameters

`SPLITSTYLE RIVER|MESH`

- It specifies the interleaving style used for the split wires.
- The default value is `RIVER`.
- `RIVER`: The split wires do not have an interleaving pattern.



- `MESH`: The split wires interleave with one another, as shown in the following illustration:



`SPLITWIDTH` *value*

- It specifies the width of the split wire segment. If routing width is larger than this value, `fcroute` will auto split them.
- The unit is micron.
- Default value: `MAXWIDTH` defined in LEF.

`SPLITGAP` *value*

- It specifies the gap between split wire segments.
- The unit is micron.
- If you specify this constraint, ensure that the distance between the split wire segments must be greater than the specified gap value.
- If you do not specify this constraint, the distance between split wire segments is the default minimum spacing value that does not cause DRC violations.

`SPLITKEEPTOTALWIDTH TRUE|FALSE`

- It is used to control different formula to compute the width of splitting wires.
- If the value is `FALSE`:
  - This is the default value.
  - The splitting wire width is calculated using the following formula:
    `Total_wire_width = split_wire_width x n + split_gap x (n - 1)`
  - For example, routing width=13, MAXWIDTH in LEF=10, split_gap=1.5.
    Therefore, `split_wire_wdith=(13-1.5)/2=5.75`

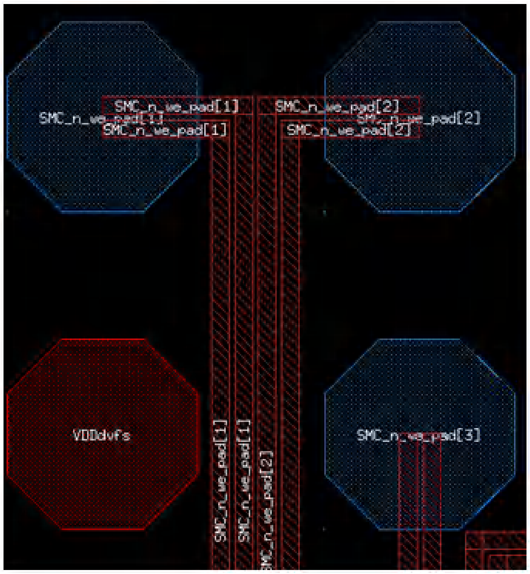
- If the value is `TRUE`:
  - The splitting wire width is calculated using the following formula:
    `Total_wire_width = split_wire_width X n`
  - For example, routing width=13, MAXWIDTH in LEF=10, split_gap=1.5.
    Therefore, `split_wire_wdith=13/2=6.5`
  - The difference from `FALSE` is to take the split gap out of the wire width calculation so that splitting wire width can be controlled by the user.

It can also be applied into the `NETS` constraint, which is a local constraint. In this case, the `SPLIT` feature works only on the associated nets in the `NETS` constraint.

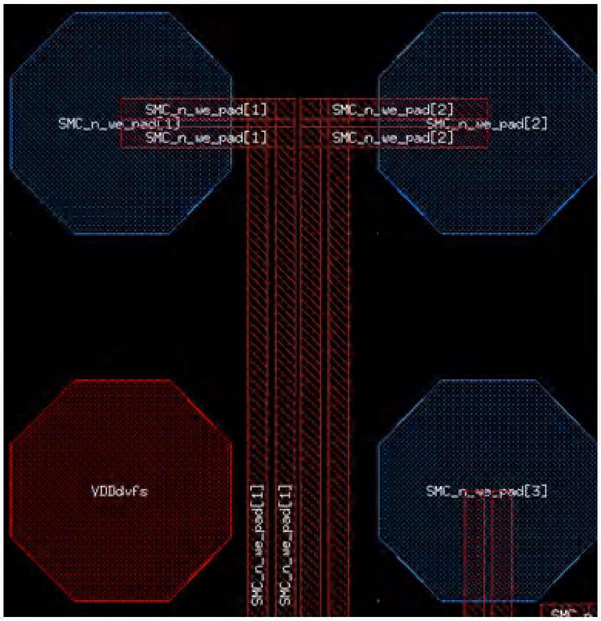
Examples

Examples of different split styles are given below:

- `SPLITSTYLE RIVER`



- `SPLITSTYLE MESH`



NETS Constraint

You can use the `NETS` constraint to set some specific constraint for nets. It can work for the aio and pio modes. It supports both `manhattan` and `45DegreeRoute`.

The syntax of the `NETS` constraint is as follows:

```
NETS

     WIDTH value

     ROUTELAYERS bottom_layer:top_layer

     SPACING value

     net_name_list

END NETS
```

Parameters

- `WIDTH value` and `SPACING value` are the same as the one for Global Constraints.
- `ROUTELAYERS` *`bottom_layer:top_layer`*
  - It specifies a layer range for routing.
  - The name of layer could be specified the layer number or the layer name in LEF.
    For example, `bottom_layer` is `metal7` (`METG2` in `LEF`) and top_layer is `metal8` (`METTOP` in `LEF`). Following usage is acceptable for `fcroute`.

    `ROUTELAYERS 7:8;`

    `ROUTELAYERS metal7:metal8;`

    `ROUTELAYERS METG2:METTOP`.// This is recommended usage.
- `<net_name_list>`

- It specifies the names of nets.
- It supports general matching methods as below:
- It supports “~”, which negates the specified net.
- It supports wildcard matching(*)
- It supports `@SIGNAL, @POWER, @GROUND`:
  - `@SIGNAL` means all signal nets;
  - `@POWER` means all power nets;
  - `@GROUND` means all ground nets.

Constraint for Changing Pin Access Direction

You can use the `PADACCESSDIR` soft constraint under `NETS` in the routing constraint file to change pin access direction. This is a soft constraint:

```
PADACCESSDIR { FROMCORE | FROMDIEBOUNDARY | EAST | WEST | SOUTH | NORTH }
```

You can set `FROMCORE` or `FROMDIEBOUNDARY` exclusively to set the preferred pin access direction from the core and from the die boundary, respectively. For example, `FROMCORE` means from the West side for I/O pad pins on the right side.
Similarly, `FROMDIEBOUNDARY` means from North for the top side.

You can also set one of `EAST | WEST | SOUTH | NORTH` directions. `EAST, WEST, SOUTH and NORTH` directions represent the 'from' direction to the pin. For example, if you want `fcroute` to access pins of I/O pads on the West side (left side) from the center, you can set `PADACCESSDIR` to either `FROMCORE` or `EAST`.

Here is an example of using `PADACCESSDIR`:

```
NETS

PADACCESSDIR EAST

    tdigit[5]

END NETS
```

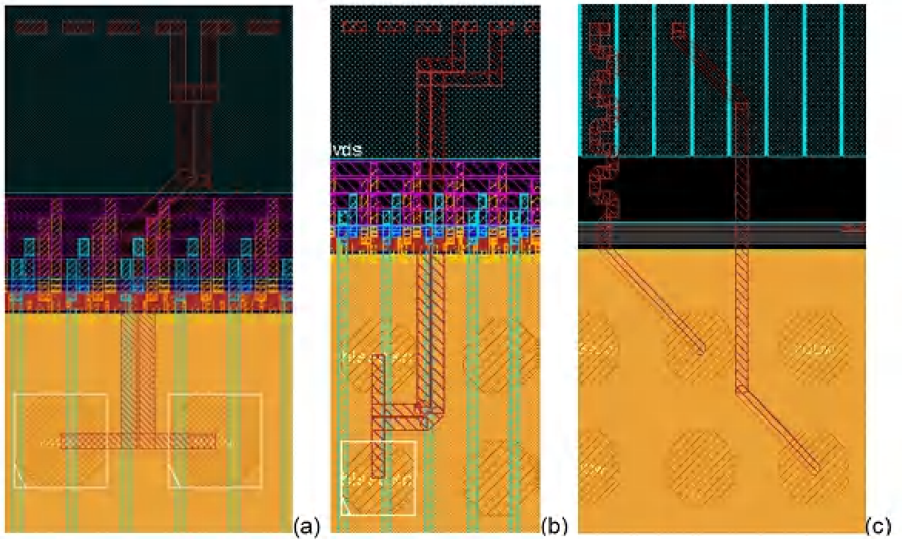
Note: This is a soft constraint.

Differential Routing Constraint

Differential routing is aimed at providing similar net delays and it applies to pairs of nets. It works only for the `aio` mode and also it supports both `manhattan` and `45DegreeRoute`.

To route these nets, `fcroute` will try,

1. Balance pair routing (matching routing length), if it fails then
2. Topological pair routing, if it fails then
3. Match L/W routing



The same constraints have been used to create the above three routing results, the differences were on the nets selected for routing. In these routing examples:

- Picture (a) has balanced routing matching the length and topology of both nets.
- Picture (b) cannot match the length but keeps the topological matching.
- Picture (c) cannot match the topological so `fcroute` matches the length and the width of both nets.

All these methodologies guarantee a similar net delay.

Syntax

```
SHARE_DIFFPAIR

     REL_DIFFERENCE value

     ABS_DIFFERENCE value

     DPAIRGAP value

     net_1 net_2

END SHARE_DIFFPAIR
```

Here:

- `SHARE_DIFFPAIR` in constraint file is supported in both the `aio` and `pio` modes by default.
- `REL_DIFFERENCE` *`value`* is used to check whether the relative difference in length of the two nets in the differential after routing meets the specified constraint value.
  - The default value is `0.2`.
  - The relative difference is calculated by using the following formula:

    ```
    REL_DIFFERENCE = 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 24 microns and 20 microns after routing, the relative difference is calculated as (24 - 20) / 20 = 0.2.
- `ABS_DIFFERENCE` *`value`* is used to check whether the absolute difference in length of the two nets in the differential pair after

routing meets the constraint.

  - The default value is `40`
  - The unit is microns.
  - The absolute difference is calculated by following formula:
    `ABS_DIFFERENCE = Length of the longer net - Length of the shorter net`
    For example, if the lengths of the constrained nets are 24 microns and 20 microns after routing, the absolute difference would be calculated as (24 – 20) = 4.

- `DPAIRGAP` *`value`* is used to control the spacing between the two nets in the differential pair.
  - The default value is the minimum spacing value in the LEF file.
  - The unit is microns.

Any violation in `REL_DIFFERENCE` or `ABS_DIFFERENCE` is reported in `srouteFcReport`.

The `SHARE_ DIFFPAIR` constraint can be used with resistance constraints if both nets in the `SHARE_DIFFPAIR` are restrained by the same `MAXRES` and `WIDTHRANGE` constraints.

```
NETS
    MAXRES value
    WIDTHRANGE min_width: max_width
    net_list
END NETS
SHARE_DIFFPAIR
    REL_DIFFERENCE value
    ABS_DIFFERENCE value
    DPAIRGAP value
    net_1 net_2
END SHARE_DIFFPAIR
```

If the two nets in a `SHARE_DIFFPAIR` are restrained by different `MAXRES` and `WIDTHRANGE` constraints, an error occurs and only the `SHARE_ DIFFPAIR` constraint is applied on those two nets.

Note: The two nets in a `SHARE_DIFFPAIR` have the same width after the resistance-driven routing feature is applied.

Example

```
SHARE_DIFFPAIR
    REL_DIFFERENCE 0.2
    ABS_DIFFERENCE 4
    port_pad_data_out[7]
    port_pad_data_out[8]
END SHARE_DIFFPAIR
```

Match Routing Constraint

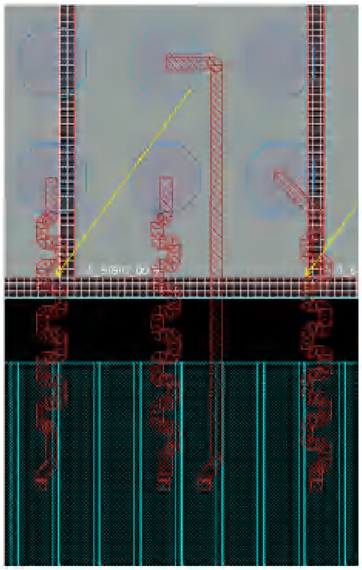
For a group of nets (more than two), the router tries to match routing length. It only works for the `aio` mode. It also supports both `manhattan` and `45DegreeRoute`.

The syntax of `MATCH` constraint is as follows:

```
MATCH

    TOLERANCE value

    <2 or more nets>

END MATCH
```

Here, `TOLERANCE` specifies the tolerance value for differential routing. `<2 or more nets>` specifies the nets for which differential routing is done.

In the picture below, the user has selected four nets to MATCH the length of the routing.



Shielding Routing Constraint

The flip chip router supports the capability of shielding nets during routing. To enable the feature, the recommendation is to use the constraint file. Both the `aio` and `pio` modes support this feature. In addition, both `manhattan` and `45DegreeRoute` support it.

The syntax of SHIELDING constraint is as below:

```
SHIELDING

    SHIELDBUMP TRUE|FALSE

    SHIELDWIDTH value

    SHIELDGAP value

    SHIELDSTYLE a|b|c

    SHIELDNET net_name

    net_name_list

END SHIELDING
```

Parameters

`SHIELDBUMP TRUE|FALSE`

- If the value is `TRUE`
  - Shields bump with the specified shield net.
  - It only works in the `aio` mode and `manhattan` routing style.
- If the value is `FALSE`
  - Does not shield bump.
  - It is the default value for `SHIELDBUMP`.

`SHIELDWIDTH` *`value`*

- It specifies the width of the *Shield Net*, measured in microns.

`SHIELDGAP` *`value`*

- It specifies the distance in microns between the shield (the special net) and the shielded net (the signal net).

`SHIELDSTYLE` *`a|b|c`*

- It specifies where you want the shield to be placed, *Above* or *Below* or on the *Common* layer.

  a = above the layer containing bumps

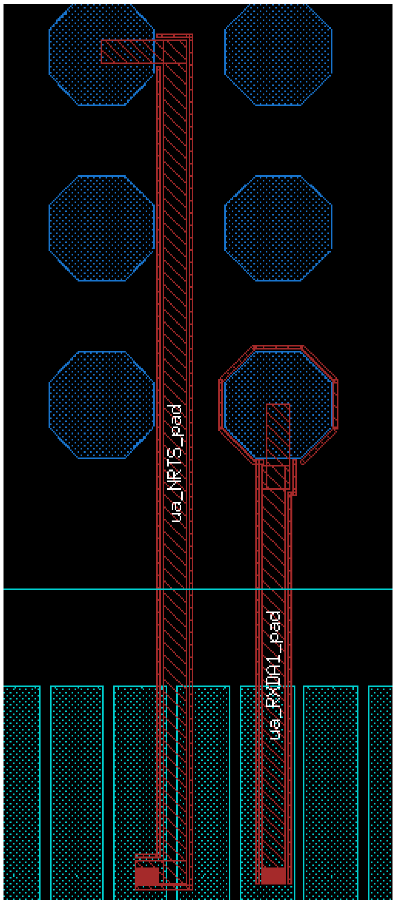
  b = *below* the layer containing bumps

  c = on the layer containing bumps ("*common* layer")

`SHIELDNET` *`net_name`*

- It specifies a special net (typically VSS) used to shield the net.
  net_name_list
- It specifies the shielded nets.

In the picture below, the left net does not have `SHIELDBUMP` and the right net can be achieved by using `SHIELDBUMP TRUE`.



Notes

- The shielding created will be floating. You will need to connect the shielding to the correct power/ground supply by using the `editPowerVia` command.
- `defOut` can mark the shielded nets as `SHIELD`, while displaying the `SHAPE` and `ROUTED` status of the metal shield wire. See the following example.

Example

Consider the following example in which the `fcroute` command connects signal bumps to I/O cells using 90-degree signal routing for aio mode. The command adds a side shield (VSS) on both sides of the signal route.

```
setFlipChipMode -route_style manhattan
```

```
fcroute -type signal -designStyle aio -layerChangeTopLayer 8 -layerChangeBotLayer 7 -routeWidth 8 -constraintFile
CFG/aio.constr
```

Constraint File CFG/aio.constr: Shield Net Description

```
SHIELDING

    SHIELDBUMP true
    SHIELDWIDTH 0.4
    SHIELDLAYERS abc
    SHIELDNET VSS
    scan_out_2
```

```
   port_pad_data_out[15]

END SHIELDING
```

DEF Syntax

`defOut` contains the `SHIELD` syntax as follows:

```
-scan_out_2 ( Bump_27_6_2 PAD ) (IOPADS_INST/Pscanout2op PAD)

+ ROUTED METAL8 16000 + SHAPE IOWIRE (1255310 541920 ) ( 1369310 *)

NEW METAL8 16000 + SHAPE IOWIRE (1263310 533920 ) ( * 695760 )

+ PROPERTY BUMP_ASSIGNMENT "ASSIGNED"

;

-VSS ( * VSS )

  + SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275310 554320 ) ( 1315310 * )

  NET METAL7 16000 + SHAPE IOWIRE ( 1255310 541920 ) ( 1315310 * )

  NET METAL8 800 + SHAPE IOWIRE ( 1250510 529520 ) ( 1315310 * )

  + ROUTED METAL6 16000 + SHAPE STRIPE ( 1553200 109600 ) ( * 186800 )

  NET METAL6 16000 + SHAPE STRIPE ( 1753200 109600 ) ( * 186800 )

  + SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275710 553920 ) ( * 675760 )

  METAL7 16000 + SHAPE IOWIRE ( 1263310 533920 ) ( * 695760 )

  METAL8 800 + SHAPE IOWIRE ( 1250910 529120 ) ( * 675760 )
```

PAIR Constraint

The `SHARE_PAIR` constraint is used to control the pairing between bumps and pads, especially for PG connection because it is a common case for PG bumps and pads to have multiple bumps to multiple pads connection requirements and the user needs to define the pairing. Both `aio` and `pio` modes support this constraint. In addition, both `manhattan` and `45DegreeRoute` routing styles support it.

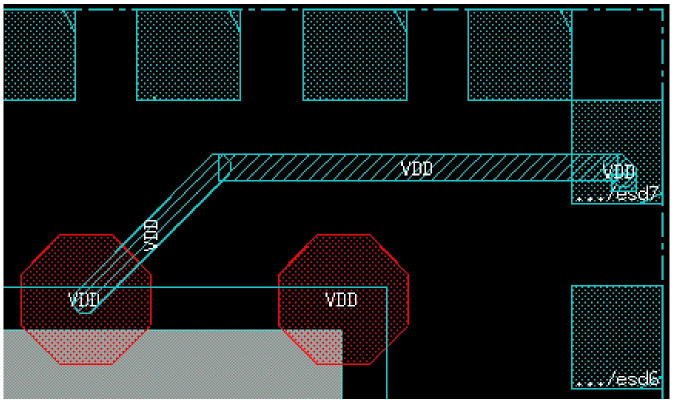
The syntax of the `SHARE_PAIR` constraint is as follows:

```
SHARE_PAIR

net_name pad_name_list bump_name_list

END SHARE_PAIR
```

Parameters

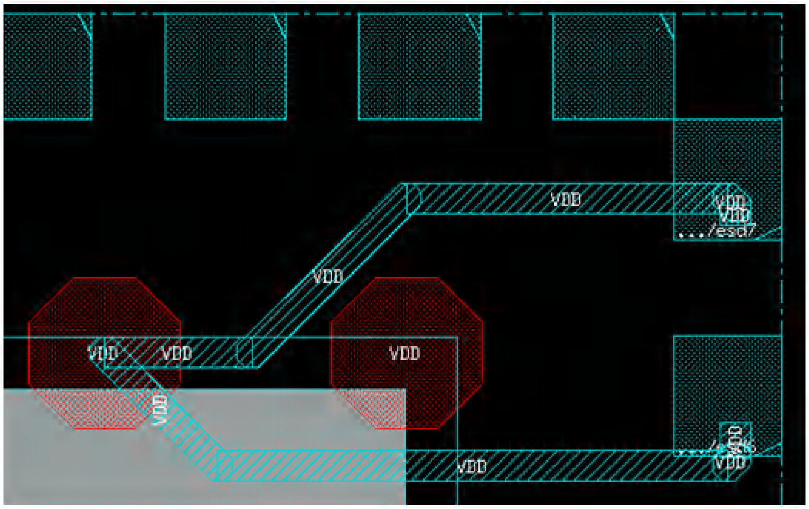
This constraint supports the following pairing pattern.

- One pad to one bump pairing



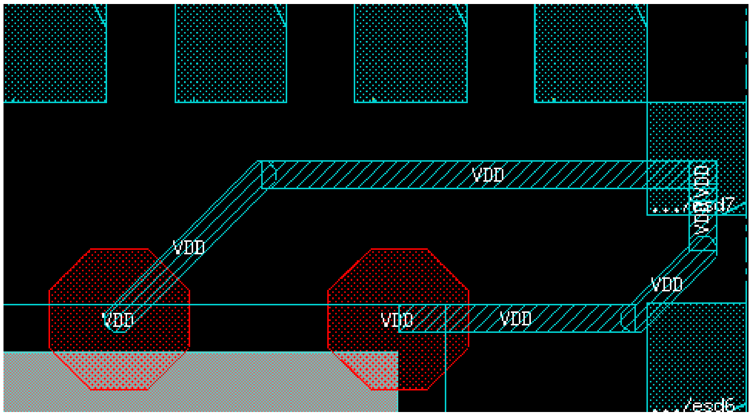
- Multiple pads to one bump pairing

  Turn on this feature with the option `-multipleConnection multiPadsToBump` in `setFlipChipMode`.



- Multiple bumps to one pad pairing

  Turn on this feature with the option `-multipleConnection multiBumpsToPad` in `setFlipChipMode`.



If the specified bump is not assigned or assigned to another net which is different from the specified net, `fcroute` will ignore it.

For those I/O pads or bumps not in the list but needed to be connected to the specified net, `fcroute` will ignore them and only route the specified ones by PAIR constraint for the same net.

Resistance Driven Constraint

You may constrain the net resistance during routing. Both the `aio` and `pio` modes support this feature. In addition, both `manhattan` and `45DegreeRoute` routing styles support it.

The syntax of Resistance Driven Constraint is as follows:

```
NETS

          MAXRES value

          WIDTHRANGE min_value: max_value

          net_name_list

END NETS
```

- `MAXRES` is the maximum resistance value allowed for the nets defined in the constraint.
- `WIDTHRANGE` specifies a variable width. The router is allowed to use any width value between these two limits to avoid violating the max resistance.

It is recommended to add `srouteFCReport` *`res_file_name`* in the extra configuration file.

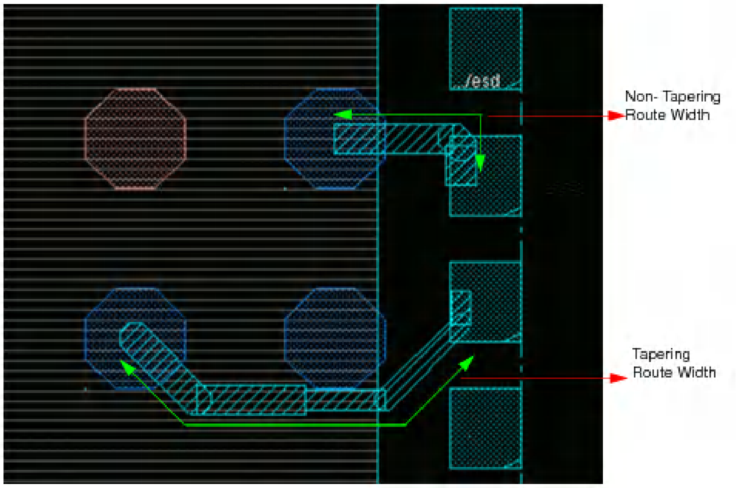
The resistance values achieved after routing will be written to this file. This file will report the resistances in a table as follows:

```
#================================

#........Resistance Table........

#  resSQ: 0.100 for METAL8

#================================

'int' 0.875

...

'port_pad_data_out[15]' Actual:3.159 Constraint:3.000
```

This information can be useful for debugging purposes. For accurate values of resistance it is recommended to run Quantus (Cadence Signoff Extraction Tool). Quantus can be run from Innovus.

Tapering Feature

The Tapering feature is enabled in the `aio` mode, wherein `fcroute` uses a thin routing width on I/O pins and wide routing width on the bumps.



You can specify the Tapering constraint in the `fcroute` constraint file. The constraint syntax is as follows:

Syntax

```
NETS

     TAPERSTEP step_value
     TAPERWIDTH width_value
     <nets>

END NETS
```

Here:

- `TAPERSTEP` is the constraint to turn on tapering feature. It takes two values: 0 and 1.
  - 0: There is no tapering needed.
  - 1: Allow router taper once from normal routing width to the taper width at somewhere on the routing wires. The tapering point is determined by the tool and user cannot control it.
- `TAPERWIDTH` defines the wire width after tapering. It takes floating value and should be in the range of [`MINWIDTH, normal_width`].
  - `MINWIDTH` is the minimum width allowed for the metal layer and has been defined in technology LEF file.
  - `normal_width` is defined by the `-routeWidth` option in `fcroute` or routing width constraint specified in constraint file.

Example

```
NETS
    TAPERSTEP 1         # 1 enables tapering, 0 disables tapering
    TAPERWIDTH 10       # After tapering, the width would be 10
    vssx_0              # Specifies the net name
    vddcx_1             # Specifies the net name
END NETS
```

SHARE_FIND_PORT Constraint

`fcroute` supports the `SHARE_FIND_PORT` constraint, which is used to find suitable ports based on specified parameters. All ports meeting the specified constraint are used by both `assignBump` and `fcroute`.

For details of the syntax and description of the parameters, see `SHARE_FIND_PORT` Syntax.

With `fcroute` support for `SHARE_FIND_PORT`, the following two global constraints are being made obsolete:

- `FINDPINLAYERS` *`layer_name`* - Used to specify the layer of the pin geometry for connection.
  - If both `SHARE_FIND_PORT` and `FINDPINLAYERS` are specified, the tool honors `F INDPINLAYERS` but gives a warning message that it is now obsolete.
  - If only `FINDPINLAYERS` is specified, the tool follows the behavior of previous releases for `FINDPINLAYERS`but gives a warning message that it is now obsolete and recommends you to use `SHARE_FIND_PORT`.
  - If only `SHARE_FIND_PORT` is specified, the tool honors`SHARE_FIND_PORT`. Behavior depends on the values specified for `LAYERS`:
    - If one value is specified for `LAYERS`, the tool chooses port geometries on the specified layer.
    - If two values are specified for `LAYERS`, the tool chooses port geometries on layer ranges `[`*`bottom_layer, top_layer`*`]`.
- `MINPINSIZE` *`width height`* - Used to specify the minimum geometry of the pin for connection.
  - If both `SHARE_FIND_PORT` and `MINPINSIZE` are specified, the tool honors `MINPINSIZE` but gives a warning message that it is now obsolete.
  - If only `MINPINSIZE` is specified, the tool follows the behavior of previous releases for `MINPINSIZE` but gives a warning message that it is now obsolete and recommends you to use `SHARE_FIND_PORT`.
  - If only `SHARE_FIND_PORT` is specified, the tool honors `SHARE_FIND_PORT`. Behavior depends on the values specified for `GEOMETRY_SHORT_EDGE/GEOMETRY_LONG_EDGE`:
    - If one value is specified, the expected edge length is equal to the specified value.
    - If two values are specified, the expected edge length is in the range `[`*`min_value, max_value`*`]`.

Examples

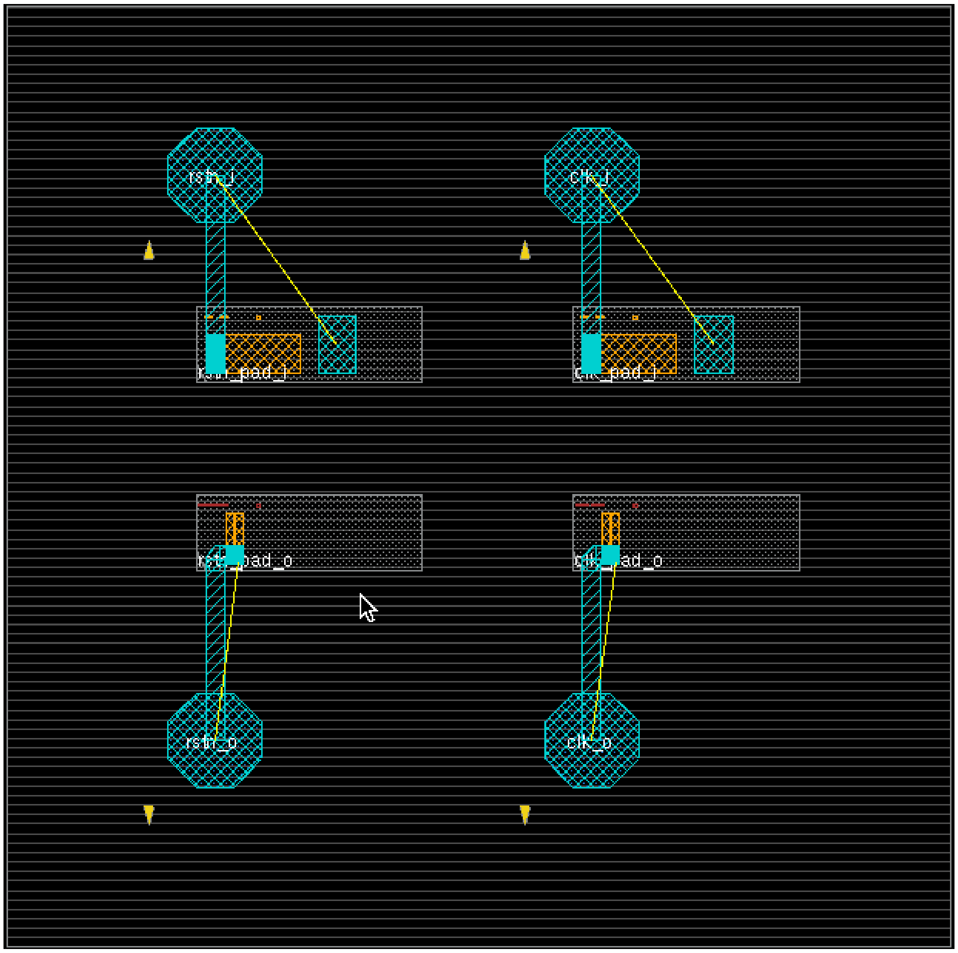
Some examples of `SHARE_FIND_PORT` are given below:

share_find_port_1.constr

```
SHARE_FIND_PORT

    PIN PAD
    MACRO LONG_INPAD
    LAYERS METAL6
    GEOMETRY_SHORT_EDGE 10:20
    GEOMETRY_LONG_EDGE 20:60
    @signal

END SHARE_FIND_PORT
```

This constraint constrains the selection of `LONG_INPAD`’s `PAD` pin: the pin should be on `METAL6`, the short edge should be between 10um~20um, and the long edge should be between 20um~60um. This constraint applies to all signal nets.

The routed result is as follows:

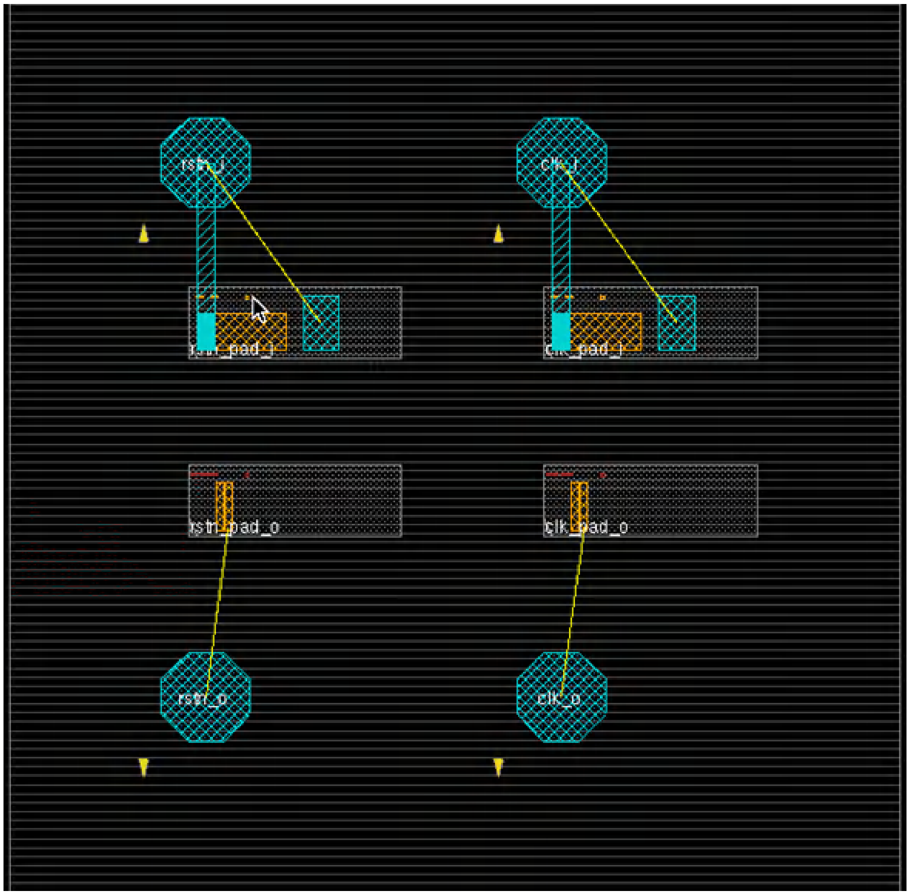


share_find_port_2.constr

```
SHARE_FIND_PORT

    LAYERS METAL6
    GEOMETRY_SHORT_EDGE 10:20
    GEOMETRY_LONG_EDGE 20:60
    @signal

END SHARE_FIND_PORT
```

This constraint constrains the selection of pins: the pin should be on `METAL6`, the short edge should be between 10um~20um, and the long edge should be between 20um~60um. This constraint applies to all signal nets.

The routed result is as follows. The pins on `rstn_pad_o` and `clk_pad_o` do not satisfy the size restrictions specified with `SHARE_FIND_PORT` constraint and are therefore not routed:

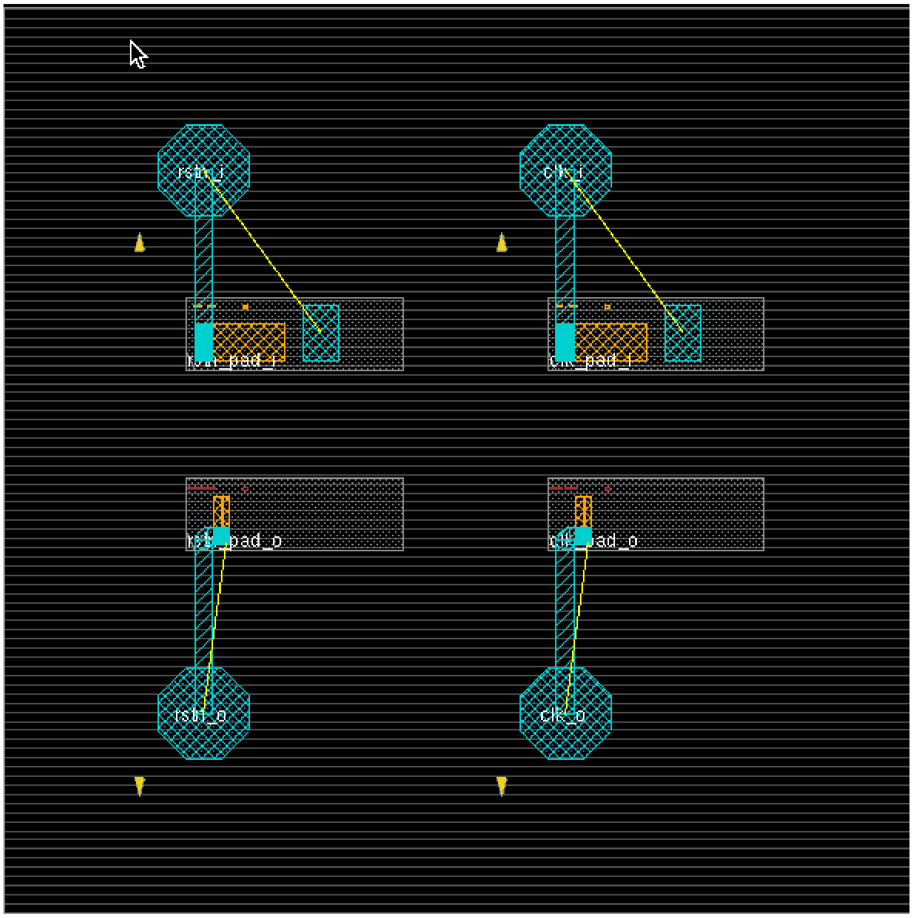


share_find_port_3.constr

```
SHARE_FIND_PORT

    GEOMETRY_SHORT_EDGE 10:20
    GEOMETRY_LONG_EDGE 40:60
    rstn_1 clk_i

END SHARE_FIND_PORT
```

This constraint applies to only `rstn_1` and `clk_1` pins. The pin's short edge should be between 10um~20um, and the long edge should be between 40um~60um.

The routed result is as follows. For `rstn_i` and `clk_i`, the `METAL8` pin is of size 20umx30um, and the `METAL6` pin is of size 20umx50um, therefore `fcroute` chooses the `METAL6` pin shape:



SHARE_IGNORE_* Constraints

The `SHARE_IGNORE_*` constraints can be used to exclude instances and macros from routing.

Excluding Instances

Use the following syntax to specify list of instances to be excluded from routing:

```
SHARE_IGNORE_INSTANCE

instance_name_list

END SHARE_IGNORE_INSTANCE
```

Here, *`instance_name_list`* specifies the list of instances that are to be excluded during flip chip routing. It supports wildcards.

Excluding Macros

Use the following syntax to specify list of macros to be excluded from routing:

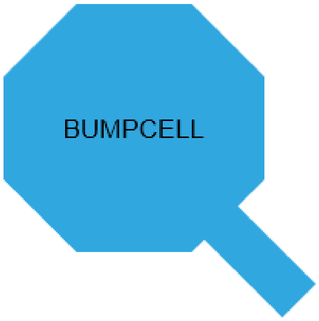
```
SHARE_IGNORE_MACRO

macro_name_list

END SHARE_IGNORE_MACRO
```

Here, *`macro_name_list`* specifies the list of macros that are to be excluded during during flip chip routing. It supports wildcards.

Constraint for Restricting the Bump Escape Direction

The `BUMP_ESCAPE_DIRECTION` constraint can be used to restrict the escape direction in bumps with pre-drawing wires for outside access.

`fcroute` supports octagonal bump pad structures with pre-drawing wires for outside access:



The pre-drawing wire is a pin in bump LEF. Use a constraint file to guide `fcroute` to connect the bump pre-drawing wire. You can specify the bump escape direction as follows in the constraint file:

```
BUMP_ESCAPE_DIRECTION
BUMPCELLNAME directions
END BUMP_ESCAPE_DIRECTION
```

One bump cell can have multiple escape directions. Supported direction keywords are `east`, `west`, `north`, `south`, `northeast`, `northwest`, `southeast`, and `southwest`. Other keywords are not allowed.

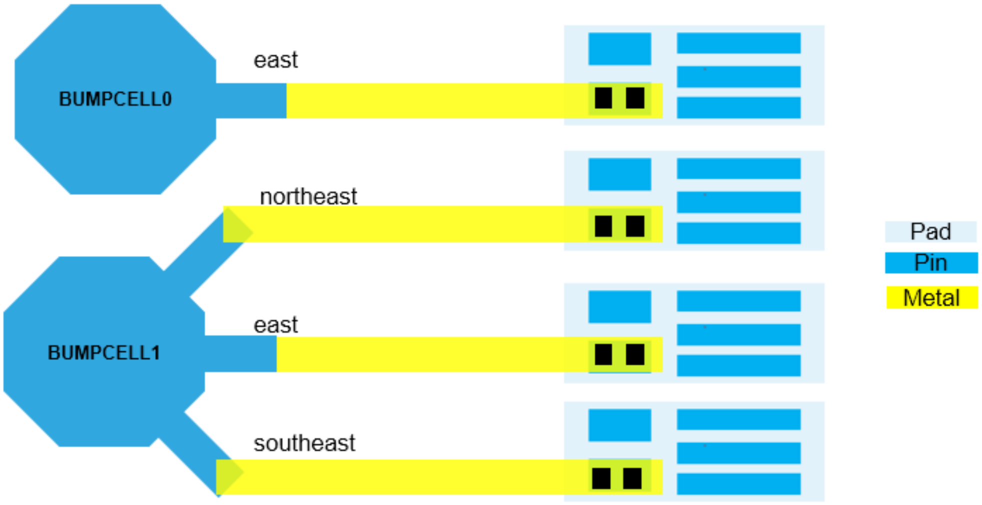
Note that this constraint file is used only for `fcroute -type signal`.

The following example shows how `fcroute` handles the such bump pad structures:

```
> fcroute -type signal -designStyle pio -constraintFile bumpEscapeDirection.const
```

In the constraint file:

```
BUMP_ESCAPE_DIRECTION
BUMPCELL0 east
BUMPCELL1 east northeast southeast
END BUMP_ESCAPE_DIRECTION
```



Useful Extra Configurations for Flip Chip Routing

The flip chip router (`fcroute`) provides you the capability to specify extra options into a text file, which can be given as an input to the command `fcroute -extraConfig` or `setFlipChipMode -extraConfig`. Because of the rich patterns in flip chip routing and extensive customization of RDL routing, this file defines extra options in addition to general settings that you may need for RDL routing. You can choose the variables that can meet your requirements.

> ⓘ ***You should only use the extra configuration file if you are familiar with its use.***

The following variables can be used in the extra configuration file:

- `srouteBumpToBumpRoute [TRUE | FALSE]`
  Specifies bump to bump routing. The default is `TRUE`.
- `srouteBumpToWireShape BLOCKWIRE`
  Connects the net to block wires
- `srouteConnectToAnyOfBump [TRUE | FALSE]`
  - If it is set to `TRUE`, it means that it is unnecessary for `fcroute` to route to the center of bump but it is acceptable to touch the bump geometry anywhere.
  - The default value is `FALSE`.
- `srouteConnectToCenterOfPin [TRUE | FALSE]`
  - If it is set to `TRUE`, `fcroute` can connect to the center of any pin except bump.
  - The default value is `FALSE`.
  - It is usually used when the width of the wire is less than the width of the pin.
- `srouteConnectToEdgeOfBump [TRUE | FALSE]`
  - The default is `FALSE`.
  - Connects a wire to the edge of the bump if it is set to `TRUE`.

- `srouteDifferentialRouteTolerance` *`value`* The default value is 2.

  If the actual difference between the length/width ratios is less than the specified value, the software omits differential routing. If the actual difference is greater than the specified value, the software attempts differential routing, but if the software cannot meet the specified value, the software displays a warning message.

- `srouteEcoMode [TRUE | FALSE]`

  The default is `FALSE`. When you set `fcroute -eco` to enable the ECO mode for Flip Chip route, `srouteEcoMode` is set to `TRUE` in the extra configuration file.

- `srouteExcludeBumpType` *`bumpCellName`* Specifies which bump cell will be excluded during routing.

  Note: Currently, `srouteExcludeBumpType` supports both power and signal bumps. Use `fcroute -type power ... -extraConfig` *`config_file_name`* to route certain types of bumps. The excluded types of bumps are defined with `srouteExcludeBumpType` in the config file.

- `srouteExcludeRegion` *`"llx1 lly1 urx1 ury1 llx2 lly2 urx2 ury2 …"`*
  - It specifies the region to exclude bumps and pads from `fcroute`.
  - It takes a string quoted by "" and in the string, multiple rectangular shapes are defined even they are disjointed.
  - The format of the string is like below,
    "`rect_ll_x rect_ll_y rect_ur_x rect_ur_y [more_rects]`"
  - The unit is DB unit.

- `srouteFcCompaction [TRUE | FALSE]`
  - The default is `FALSE`.
  - Turns on compaction routing, when set to `TRUE` . In the flip chip post-route stage, the compaction function pushes the routing as close as possible to bumps in order to leave more routing resource for subsequent routing steps, such as P/G bump connections or general power routing.
    Note: This feature cannot be used with resistance driven and diffpair routings because the compaction action may change the wire length and routing pattern. This feature does not support two-layer RDL routing.

- `srouteFcDieAreaOffset` *`"left bottom right top"`*
  - Specifies the distance (in microns) to which the expanded area should extend from the edges of the chip. If specified, `fcroute` can finish the routing in this expanded die area instead of the actual die area. The default value is " `0 0 0 0` ". This option does not support negative values.
    Note: Routing is not allowed outside the expanded die area. If `fcroute` cannot finish the routing to specific bumps in the expanded die area, it leaves these bumps open.

- `srouteFCReport` *`file_name`*
  - Writes the resistance report into the specified file using the resistance constraint MAXRES and the inputs. It also reports the width and length of special nets routed by `fcroute`.

- `srouteFcrMazeRoute45 [TRUE | FALSE]`

  Enables maze routing for 45 degree style routing. Maze routing increases the final routing completion rate for `fcroute` and improves results for nets with complex patterns and long routing path.

- `srouteFcrouteAllowOverCongestion [TRUE | FALSE]`

  Specifies that all nets are routed even if there is not enough spacing in some area. The default is `FALSE`.

- `srouteFCrouteLayerIsPreferred.` *`n`* `[TRUE | FALSE]`

  Allows you to specify layer priorities for routing in a multi-layer design. For example, if you want M9 to be used only in very congested area, set `srouteFCrouteLayerIsPreferred.9` to `FALSE` to indicate that M9 is not a preferred layer for routing. Without any setting, every layer is considered a preferred layer, and the routing is distributed evenly.

> **Note:** Currently, layer priority can be specified only in the `pio` flow.

- `srouteFcroutePadPinTagging [TRUE | FALSE]`
    - It enables global router to honor port numbering if it is set to `TRUE`.
    - The default value is `FALSE`.
- `srouteGrouteClusterRegions` *`x1 y1 x2 y2 x3 y3 x4 y4 ...`*
  Allows you to specify the area I/O cluster regions in microns. Here, *`x1 y1 x2 y2`* coordinates are equivalent to the *`llx`* , *`lly`* , *`urx`* , and *`ury`* coordinates of the first area I/O cluster, *`x3 y3 x4 y4`* represent the second, and so on.

  Default: " "
- `srouteGrouteIncremental {TRUE | FALSE}`
  The default is FALSE.
- `srouteGrouteLengthDriven [TRUE | FALSE]`
  The default is `FALSE`. When optimizing placement and bump assignment, bias to vertical / horizontal connection instead of 45-degree connection.
- `srouteGrouteMaxPathPerBump` *`num_of_routes`*
  Specifies the maximum number of routes coming from a single bump. Use along with `srouteGrouteMaxPathPerPad` to control the maximum number of connections to a pin port for multiple bump routing. Here, *`num_of_routes`* has a range of `1` to `15`.
- `srouteGrouteMaxPathPerPad` *`num_of_routes`*
  Specifies the maximum number of routes to a single pad pin geometry. Use along with `srouteGrouteMaxPathPerBump` to control the maximum number of connections to a pin port for multiple bump routing. Here, *`num_of_routes`* has a range of `1` to `15`.
- `srouteGrouteOptimizeWidth [TRUE | FALSE]`
  The default is `FALSE`.
  Note: You must use this option in conjunction with the `srouteGrouteOptimizeSpacing` option and they cannot be set to `true` at the same time.
  Automatically adjusts the width of the nets specified in the constraint file within the max and min constraint. When set to `true`, this configuration file option calls the global router to get the optimized width for each net and writes the results to a constraint file called `width.cons`. If it is an `fcroute -designStyle pio`, the router routes the given nets as wide as possible (still within the width range) while maintaining routability.
  You can create your own constraint file to specify how the width is calculated. For example, if you have 10 nets in the design (`n1` through `n10`), you can create the following constraint file:

```
NETS
MINWIDTH 2.0
MAXWIDTH 5.0
WIDTHSTEP 0.1
n1 n2 n3
END NETS

NETS
MINWIDTH 5.0
MAXWIDTH 10.0
WIDTHSTEP 0.5
n4 n5 n6
END NETS

NETS
```

```
WIDTH 3.0
n7 n8
END NETS


NETS
WIDTH 12
n9 n10
END NETS
```

where:

n1, n2, and n3	Have a width between 2 and 5, and an incremental size of 0.1.
n4, n5, and n6	Have a width between 5 and 10, and an incremental size of 0.5.
n7 and n8	Have a fixed width of 3.0.
n9 and n10	Have a fixed width of 12.0.

Note: All the nets within the same NET group will use the same final width. If you allow the nets to have different widths, while satisfying the min and max value, you can use the following configuration file option:
`srouteGrouteUniformWidth FALSE`

- `srouteGrouteOptimizeSpacing [TRUE | FALSE]`

  The default is `FALSE`.

  > **Note:** You must use this option in conjunction with the `srouteGrouteOptimizeWidth` option and they cannot be set to `true` at the same time.
  > Automatically adjusts the spacing within the max and min constraint. When set to `true`, `fcroute` calculates the maximum allowable spacing (for the given net width) and writes it to the log file. All spacing values are the same.

- `srouteGrouteSerialBumpRouting [TRUE | FALSE]`

  Allows bumps of the same net to be connected together. The default is `FALSE`.

- `srouteGrouteUniformWidth [TRUE | FALSE]`

  All nets in the same width group have uniform width. The default is `TRUE`.

- `srouteJogControl [TRUE | FALSE]`

  Allows jogs during routing to avoid DRC violations. The default is `FALSE`.

- `srouteLayerChangeExcludeRegion "`*`llx1 lly1 urx1 ury1 llx2 lly2 urx2 ury2 …`*` "`

  Specifies the region to be excluded from layer change initiated by `fcroute`. In the argument string for this option, you can define one or more rectangular shapes even if they are disjointed. You can also define a rectilinear shape. The format of the argument string is as follows:

  - `"`*`rect_ll_x rect_ll_y rect_ur_x rect_ur_y`*` [`*`more_rects`*`]"`
    or
  - *`"rectilinear_x1 rectilinear_y1 rectilinear_x2 rectilinear_y2 …"`*

  `srouteLayerChangeExcludeRegion "0 0 0 0"`means layer change is allowed on the whole chip. `srouteLayerChangeExcludeRegion` must be used with `PIOLAYERCHANGE PAD` in the constraint file. Only the `pio` mode supports this constraint.

- `srouteLengthLimit [integer value]`

  Specifies the maximum routing wire length for each flip chip net. The default is `0`.

- `srouteMinLength [integer value]`
  Specifies the minimum segment length. The default is `0`.
- `srouteOutputFailedResistanceGroute [TRUE | FALSE]`
  Displays nets that failed maximum resistance with different colors. The default is `FALSE`.
- `sroutePinSpacing` *`dbUnitValue`*
  Defines the I/O pad pin spacing.
- `sroutePioBusRoute [TRUE | FALSE]`
  Routes only the nets and bumps defined in the `NETGROUP` constraint, when set to `TRUE`. The default is `FALSE`.
- `sroutePioDiffPair [TRUE | FALSE]`
  Supports differential pair (DIFF PAIR) routing in the `pio` mode. The default is `FALSE`.
- `sroutePowerBumpAllDir [TRUE | FALSE]`
  Connects power bump to power stripes on all directions. The default is `FALSE`.
- `sroutePreferSameLayerJog [TRUE | FALSE]`
  Prefers a jog in the same layer. The default is `FALSE`.
- `sroutePrevent45ForLowerLayer [TRUE | FALSE]`
  - It can prevent 45-degree routing for lower layer if it is set to `TRUE`.
  - The default value is `FALSE`.
- `sroutePreventViaUnderBump [TRUE | FALSE]`
  Prevents via under a bump. The default is `FALSE`.
- `sroutePushAndShove [TRUE | FALSE]`
  If `TRUE`, detail routing gets ripped and rerouted, to route open nets. The default is `TRUE`.
- `sroutePushAndShoveVerbose [TRUE | FALSE]`
  If `TRUE`, displays debugging messages during `push_and_shove`. The default is `FALSE`.
- `srouteReduceLayerChanges` *`integer`*
  The default is `0`.
- `srouteRouteSpacing [integer value]`
  Specifies the default routing space for each flip chip net. The default is 0.
- `srouteRouteWidthForLowerLayer` *`dbUnit`*
  - Indicates that `fcroute` should use *`dbUnit`* as width for the lower layer.
  - For example, if DB unit is 2000 and the width for lower layer is 8 micro, then this value should be set as 2000*8=16000.
- `srouteSpreadWiresFactor` *`float`*
  Specifies automatic spreading of wires during bump routing in order to prevent any SI violations. Here, *`float`* is applied as a multiple of the minimum spacing that is required.
- `srouteStraightRouteOnly [TRUE | FALSE]`
  Specifies straight routing connections between targets. The default is `TRUE`.
- `srouteUseSpecifiedWidthForTopLayer {TRUE | FALSE}`
  If `TRUE`, forces `fcroute` to use user-defined width for the top layer connecting to the bump. The default is `FALSE`.

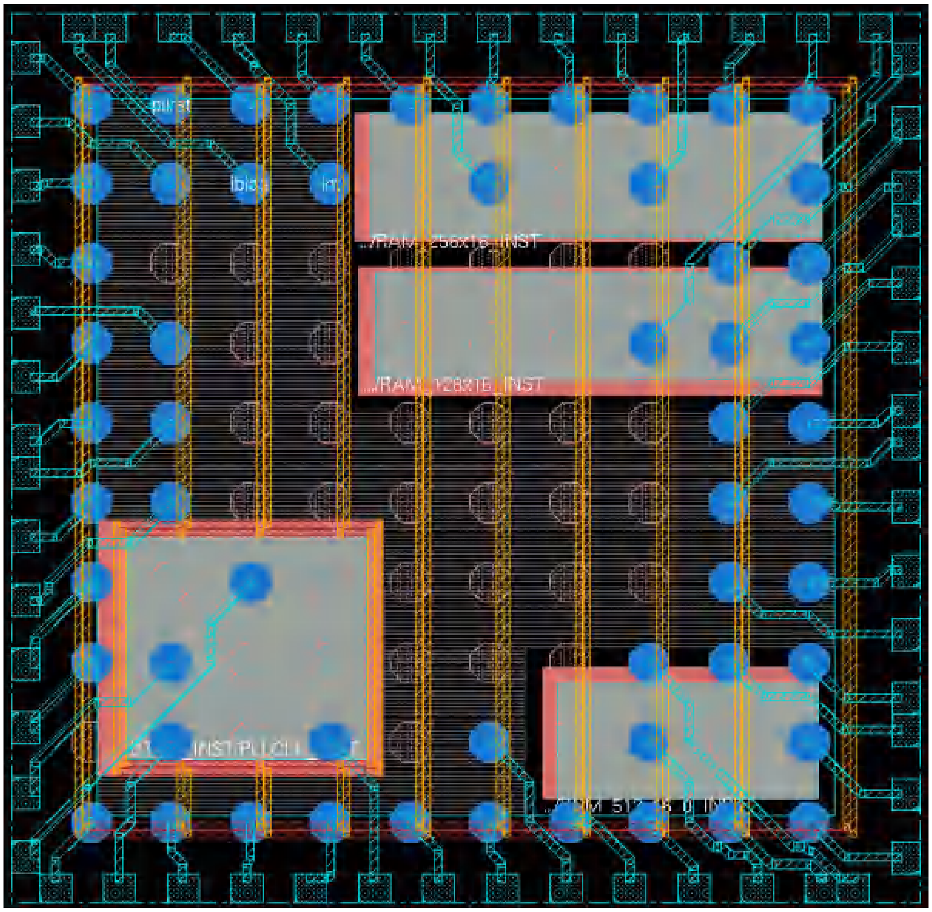
Power Routing

There are two methodologies for connecting the power and ground bumps that can coexist in the same design:

- Connect PG bumps to the I/O pads.
- Connect PG bumps to rings or stripes.

Connect PG Bumps to I/O Pads

It is recommended that these connections are done at the same time as the signal routing. First use the command `setFlipChipMode -connectPowerCellToBump true` to enable connecting power cells to bumps, and then use the command `fcroute -type signal` to route power cells to bumps.



Note: In a design, some Power/Ground (PG) bumps may be already connected to PG stripes before flip chip routing is done. `fcroute` first checks whether port number is defined in bumps. If yes, `fcroute` routes the bumps to IO even if they are already routed by PG stripes.

The command `fcroute` will automatically pair PG bumps with PG pads based on routability with some intelligence if there are no specific pairing constraints by the user. Also `fcroute` will choose the suitable pin for routing based on its intelligence if there is no constraint by the user.

For more control of the connection between PG bumps and PG pads, the designer can use the `SHARE_PAIR` constraint in the constraint file or use port numbering feature.

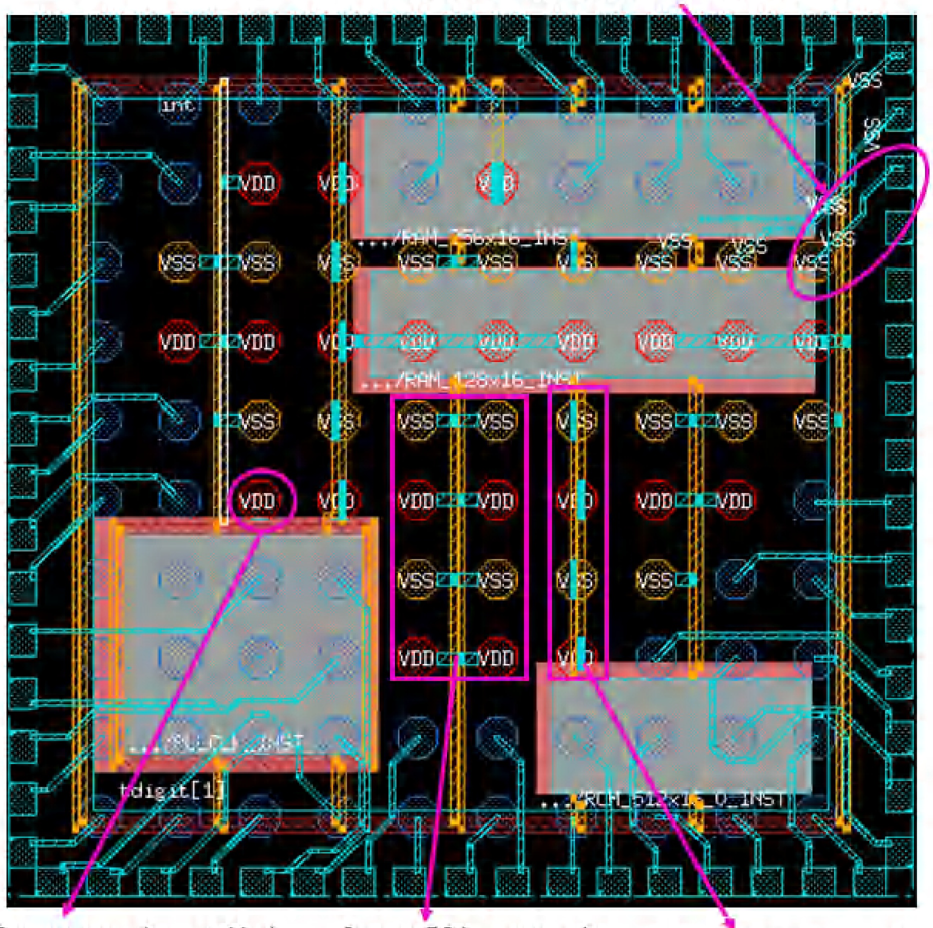
PG Bumps Connect to Rings or Stripes

The command `fcroute -type power` can connect PG bumps directly to rings or stripes.

You can create some extra power and ground bumps to reduce the IR drop. This is one of the advantages of the flip chip design.

The image below shows the difference between `fcroute -type signal -designStyle pio`, `fcroute -type power`, and `addStripe`.

Connect ground bump to IO pad with "setFlipChipMode -connectPowerCellToBump true" and "fcroute -type signal -designStyle pio"

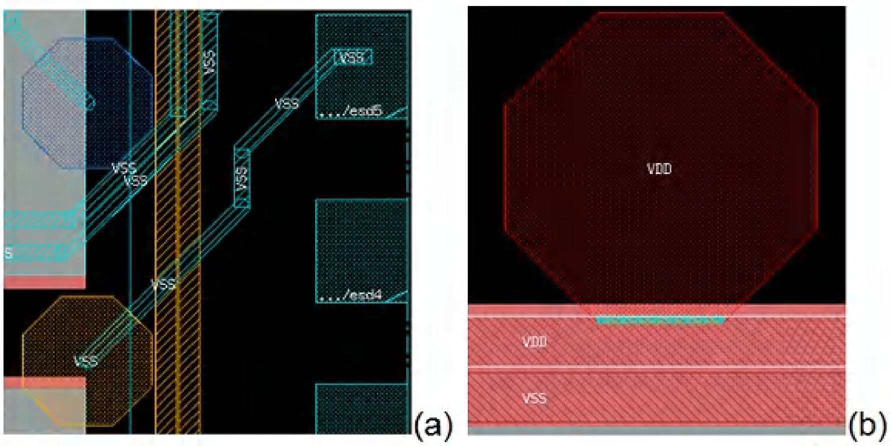


Connect power bump to block ring with "fcroute -type power"

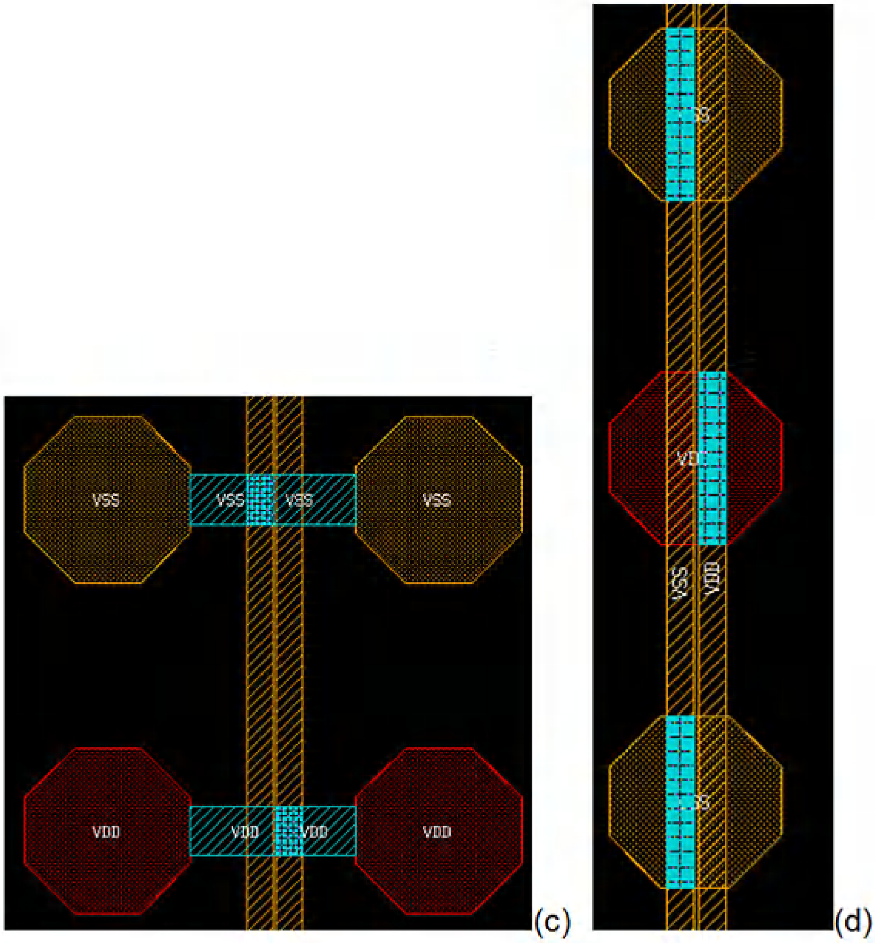
Connect PG bumps to stripe with "fcroute -type power"

Connect PG bumps to stripe with "addStripe"

- Connect ground bump to I/O pad with `setFlipChipMode -connectPowerCellToBump true` and `fcroute -type signal -designStyle pio`, as shown in (a).
- Connect power bump to block ring with `fcroute -type power`, as shown in (b).



- Connect PG bumps to stripe with `fcroute -type power`, as shown in (c).
- Connect PG bumps to stripe with `addStripe`, as shown in (d).



ECO Routing

The `fcroute` command detects ECO changes and performs ECO routing automatically. As a result, you do not need to modify routing manually to complete ECO changes. To enable ECO mode flip chip routing, use the `-eco` parameter of the `fcroute` command. When you specify the `-eco` parameter, `fcroute` automatically performs typical ECO routing steps, such as deleting existing routing results or re-routing affected nets, whenever there is an ECO change.

Typical ECO routing working modes are:

- Working Mode I:
  - Delete existing routing results
  - Re-route affected nets
- Working Mode II
  - Re-route affected nets
- Working Mode III
  - Delete existing routing results

Let's see how typical ECO changes impact routing:

- IO-related ECO changes

ECO Change	**Impact on Routing**
Change in IO pad location	Invokes `fcroute -eco` (Working Mode I)
Change in IO pad orientation	Invokes `fcroute -eco` (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 `fcroute -eco` (Working Mode II)
Delete an IO pad	Invokes `fcroute -eco` (Working Mode III)

- Bump-related ECO changes

ECO Change	**Impact on Routing**
Change in bump location	Invokes `fcroute -eco` (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 `fcroute -eco` (Working Mode I)
Add a new bump	Invokes `fcroute -eco` (Working Mode II)
Delete a bump	Invokes `fcroute -eco` (Working Mode III)

Change the characters of an existing bump array	No ECO routing needed
Add a new bump array	No ECO routing needed
Delete a bump array	No ECO routing needed

- Routing ECO change

ECO Change	Impact on Routing
Routing is partially deleted	Invokes `fcroute -eco` (Working Mode II) **Note:** The rerouting should preserve undeleted routings and do reroute on deleted routing only.
Extra routing wires added but they cannot form a complete path from pin to bump	No ECO routing needed
Extra routing wires added and they can form a complete path from pin to bump	Invokes `fcroute -eco` (Working Mode III) and keeps new wires

- Netlist ECO changes
  - Bumps are not in netlist.
  - For netlist ECO changes related with IOs, the tool performs routing as for IO-related ECO changes.

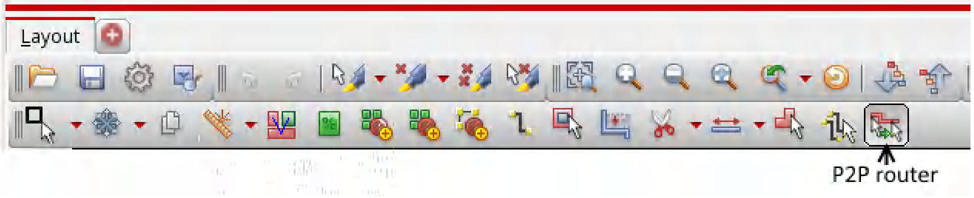
Note: The optimization of IO pad location and/or bump assignment after IO ECO is not supported. You can use the exclude region constraint to accomplish the task. For instance, in the example below, use `srouteExcludeRegion` in the `etr.cfg` file to exclude unaffected areas so that the optimization is done only on affected components:

```
placePIO -assignBump -extraConfig etr.cfg
```

`assignBump` issues an ERROR message and ignores it.

P2P Router

Innovus supports a semi-automatic point-to-point (P2P) router. The P2P router can be accessed as follows:

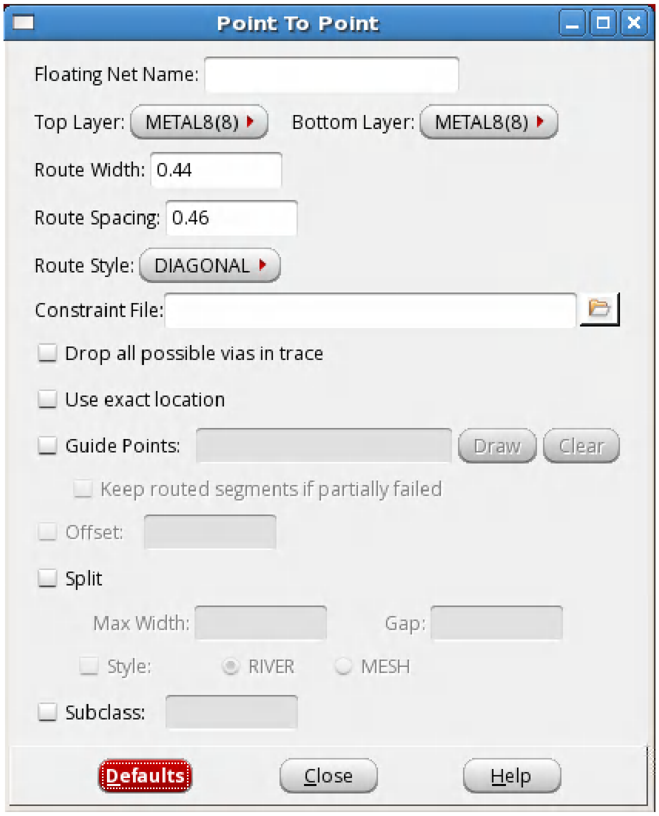


The P2P router can be used for customized and special routing pattern generation. To use the P2P router:

1. Select the P2P router button from the toolbar on the main window.
2. Press `F3` to set up the P2P router.
3. Click the object that will be the source.
4. Click another object to set it as the target.

Setting Up the P2P Router

When you press `F3` for P2P setting, the following GUI form opens:



Point To Point

Floating Net Name:

Top Layer: METAL8(8) Bottom Layer: METAL8(8)

Route Width: 0.44

Route Spacing: 0.46

Route Style: DIAGONAL

Constraint File:

□ Drop all possible vias in trace

□ Use exact location

□ Guide Points: Draw Clear

□ Keep routed segments if partially failed

□ Offset:

□ Split

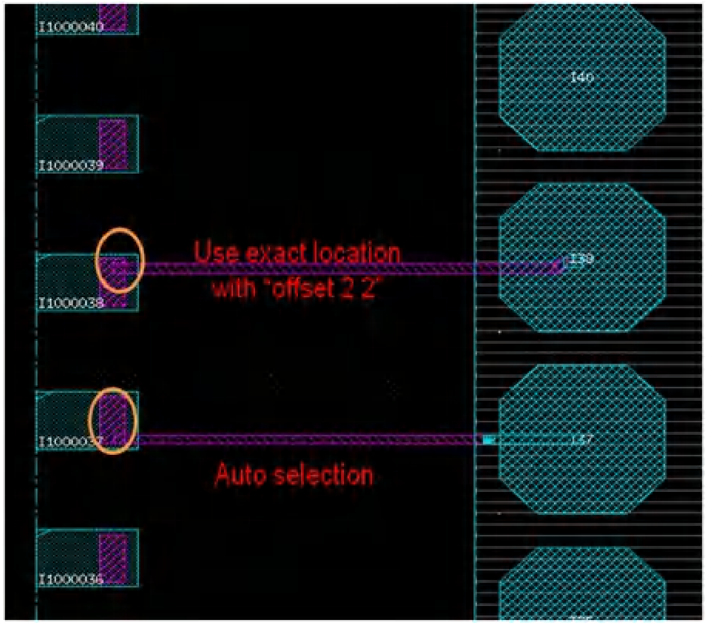
Max Width: Gap:

□ Style: ● RIVER ○ MESH

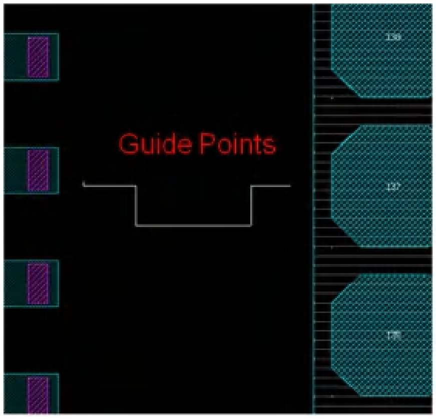
□ Subclass:

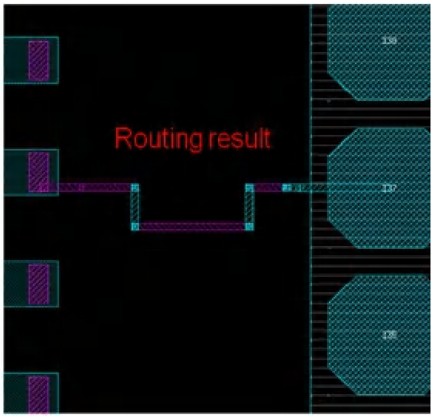
- If net name is not set, the P2P router automatically picks up the net name based on the objects selected in the GUI.
- If you do not select the *Use exact location* check box in the Point To Point form, the P2P router performs an auto search to find an access point. Note that:
  - *Use exact location* specifies that the P2P router should use the exact location that you have clicked.
  - You can additionally select the *Offset* check box and specify offset values to define the maximum search offset from the original click location.

  The picture below depicts the results of auto selection versus using *Use exact location*:



- You can specify *Guide Points* in the Point to Point form to define a customized routing pattern. You can use this feature in following ways:
  - Click the *Draw* button in the form and then click the required guide points in the main window. Press the `Esc` key to return the location of selected guide points to the *Guide Points* text box. Click two objects for the source and target.
    Or
  - Press `Shift` to turn on *Guide Points*. Then, click the required guide points in the main window. Click two objects for the source and target.
    Or
  - Directly enter the location of the guide points in the *Guide Points* text box in the Point to Point form. Click two objects for the source and target.





Handling Flip Chip Designs with Complex Floorplans

If you have a flip chip design with a complex floorplan, you might get the following error:

```
**ERROR: Exceeding maximum number of 32 rows per cluster in cluster 0. Quit
```

The reason for this error is that if a design has a floorplan with IO pads of many different heights, the flip chip router may not be able to generate that many IO rows of different heights, internally. So, the flip chip router cannot proceed.

To solve this issue, you should do the routing in parts. You can try to unplace some IO pads (with CLASS PAD AREAIO or CLASS BLOCK) or filler cells (with CLASS PAD SPACER or CLASS PAD AREAIO) that may not need to be routed. With a simplified floorplan, do flip chip routing. After routing, you can load the IO pads and fillers back using the saved floorplan. Then, unplace another part and do routing.

Flip Chip Router Report

You can use the `srouteFCReport` *`file_name`* option in the extra configuration file to report width, length, and resistance of the special nets routed by `fcroute`. The report file generated by this option has the following format:

```
####################################################
##########        fcroute report        ############
####################################################
NET net_name bump_name:pad_name:pin_name:port_num
         STATUS open/resistance violation/routed
         Layer#: to be split/tapering/widthOpt
              Path: Width xx Length xx Resistance xx
              Path: Width xx Length xx Resistance xx
              …
         Layer#: to be split/tapering/widthOpt
              Path: Width xx Length xx Resistance xx
              Path: Width xx Length xx Resistance xx
              …
         Total length:xx
         Total resistance:xx (Constraint:xx)
END net_name
```

Format Definitions

- *`net_name bump_name:pad_name:pin_name:port_num`*
  Specifies the connection between bump and pad based on net. Incomplete connection specifications are also supported:
  - If the bump does not have the port number property, it outputs *`bump_name:pad_name`*
  - If the port number property does not include *`port_num`* , it outputs *`bump_name:pad_name:pin_name`*
- `STATUS  open/resistance violation/routed`
  Reports the status of the net as one of the following:

Status	Meaning
`open`	If the net is not routed, its status is reported as open. No more information is output for this net.

`resistance violation`	If `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:

      $$R = \sum_{i=1}^{n} \rho Li/Wi$$

      where Li is the length of center line of i^{th} wire segment, Wi is the width of the i^{th} wire segment and ρ 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#`	Indicates that no special feature is applied on this net.
	For example, suppose net `VSS` is assigned to `bump_vss` with incomplete port number property.
	`NET VSS bump_vss:ground_pad:vss`
	`      STATUS resistance violation`
	`      Layer TOP_RDL`
	`           Path: Width 25 Length 200 Resistance 2.5`
	`      Layer 2nd_RDL`
	`           Path: Width 25 Length 50 Resistance 0.8`
	`      Total length: 250`
	`      Total resistance: 3.3 (Constraint:3)`
	`END VSS`

Layer# to be split	Indicates that the splitting feature will be applied on this net. Note that splitting happens after the report is output. For example, suppose net VSS is assigned to bump_vss with incomplete port number property. NET *VSS bump_vss:ground_pad:vss* STATUS routed Layer TOP_RDL Path: Width 25 Length 200 Resistance 2.5 Layer 2nd_RDL **to be split** Path: Width 25 Length 50 Resistance 0.8 Total length: 250 Total resistance: 3.3 END *VSS*
Layer# tapering	Indicates that the tapering feature is applied on this net. For example, suppose net VSS is assigned to bump_vss with incomplete port number property. NET *VSS bump_vss:ground_pad:vss* STATUS routed Layer TOP_RDL **tapering** Path: Width 25 Length 200 Resistance 2.5 Path: Width 15 Length 50 Resistance 1 Total length: 250 Total resistance: 3.5 END *VSS*
Layer# widthOpt	Indicates that width optimization feature is applied on this net. For example, suppose net VSS is assigned to bump_vss with incomplete port number property. NET *VSS bump_vss:ground_pad:vss* STATUS routed Layer TOP_RDL **widthOpt** Path: Width 25 Length 250 Resistance 3 Total length: 250 Total resistance: 3 END *VSS*

Advanced Flip Chip Features

Two-Layer RDL Routing

As flip chip design become more and more complex, one layer may not be sufficient for completing RDL routing. Innovus supports two-layer RDL routing for complex designs. However, in most flip chip designs, complete two-layer routing may not be required. Instead, you may need to use two layers only in the IO area and one layer in the core area to optimize routing resources. In these cases, you can add routing blockages to control where two layers are used and where only only 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` *`"llx1 lly1 urx1 ury1 llx2 lly2 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 `45DegreeRoute` styles support this constraint.
        - By default, this constraint uses the region defined by `srouteLayerChangeExcludeRegion` *`"llx1 lly1 urx1 ury1 llx2 lly2 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` *`"llx1 lly1 urx1 ury1 llx2 lly2 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 `45DegreeRoute` styles support this constraint.
        - In the argument string for this option, you can define one or more rectangular shapes even if they are disjointed. You can also define a rectilinear shape.
        - The format of the argument string is as follows:
          `"`*`rect_ll_x rect_ll_y rect_ur_x rect_ur_y`* `[`*`more_rects`*`]"` or
          `"`*`rectilinear_x1 rectilinear_y1 rectilinear_x2 rectilinear_y2`* `..."`
        - The default region is the core region of the chip. If the core region is the same as the whole chip area, `fcroute` ignores the `PIOLAYERCHANGE PAD` setting in the constraint file.
        - This option prevents layer change in the specified region. However, multiple layers can be allowed in the region. This option places a virtual routing blockage on the cut layer. To prevent wires on a certain layer, a routing blockage is still needed.
        - This option is a soft constraint for `fcroute`.

    If the setting for routing layers is `-layerChangeBotLayer TOP_RDL -layerChangeTopLayer TOP_RDL`, `fcroute` implements the above settings as follows:

- Uses TOP_RDL as much as possible. It uses the second redistribution layer (RDL) only when a single layer cannot finish routing in case of cross-over.
- Honors the settings defined by `srouteLayerChangeExcludeRegion` as a soft constraint.

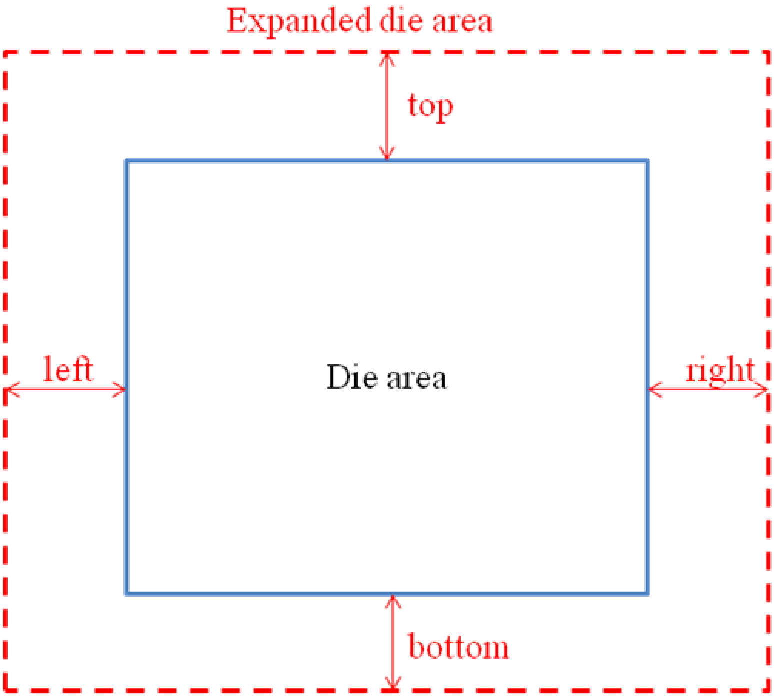
2. Add routing blockage to control where to make layer change.
   `fcroute` strictly honors routing blockages. If you use a routing blockage with other routing constraints, `fcroute` honors the mixed usage as well.

Routing Bumps in the eWLB Process

In embedded wafer leaver ball (eWLB) grid array process, some bumps are placed out of the chip and are required to connect to the IO pin in the die. However, out-of-die routing is not allowed by default. You can use the `srouteFcDieAreaOffset` option in the `fcroute` extra configuration file to specify the expanded area in which routing should be allowed:

```
srouteFcDieAreaOffset "left bottom right top"
```

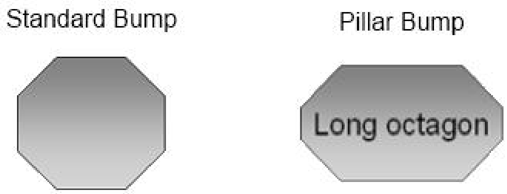
Here, *`left`* specifies the distance (in microns) to which the expanded area should extend from the left edge of the chip. Similarly, *`bottom`* , *`right`* , and *`top`* specify the distance from the bottom, right, and top edges of the chip, respectively. `f croute` can finish the routing in this expanded die area instead of the actual die area.



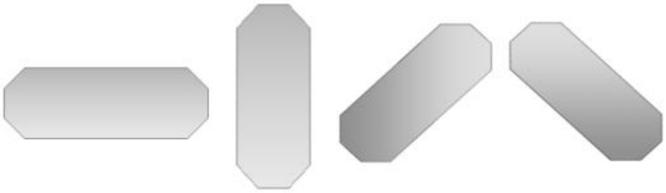
Note: Routing is not allowed outside the expanded die area. If `fcroute` cannot finish the routing to specific bumps in the expanded die area, it leaves these bumps open.

Pillar Bump Support

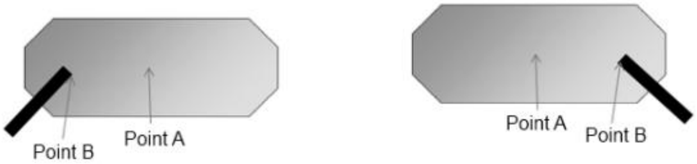
Flip chip development is driven by device performance and package miniaturization trends. Higher device performance leads to more input/output connections per IC. At the same time, miniaturization requires smaller/thinner packaging, leading to smaller, closer-spaced connections. Fine-pitch flip chip pillar bumps reduce size while meeting the challenges of thinner ICs and maintaining robust IC and package reliability. As a result, pillar bumps are being used more and more in flip chip designs. With pillar bump, the shape of a bump changes from octagon to a long octagon as shown below:



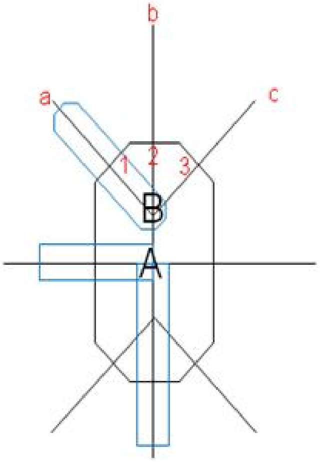
`fcroute` supports horizontal and vertical pillar bumps. In addition, it also supports 45/135-degree pillar bumps, which are already defined in LEF.



`fcroute` connects to bump center, point A for long-side or short-side entry, or point B for diagonal-side entry



Here, Point B is the intersection of lines a, b and c, which are the perpendicular bisectors of sides 1, 2 and 3 as shown below.



Sometimes, if `fcroute` cannot complete the routing when connecting to the center of bump, it may adjust the connection location with its intelligence and without causing any DRC violations.

You can use the `create_bump -orientation` option to specify the orientation of a pillar bump. When required, you can use this option to rotate a pillar bump by 90, 180, or 270 degrees to alleviate package and chip routing problems.

Note: Innovus does not rotate a bump by 45 degrees, but it can place and route 45-degree pillar bumps defined in LEF. Instead of rotating the pillar bump, you can change the pillar bump master for correct orientation.

fcroute Bus Routing for DDR3

A long routing path in Double Data Rate 3 (DDR3) may cause crosstalk between signals. To prevent crosstalk, you must add P/G nets between signal nets and route these nets together with the same pattern like bus routing and specify different width and spacing per net.

In the bus routing pattern for DDR3:

- Route signal and P/G nets together with the same pattern in a long path.
- The shielding P/G net is floating, not connected to bump or pad pin.

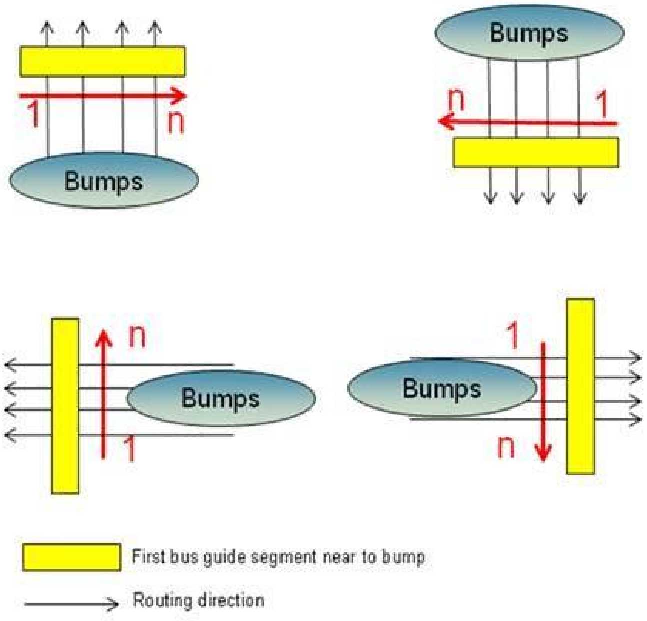
- You can define the routing width and spacing for each net.

To support this bus routing pattern, you need to add the `NETGROUP` constraint in the constraint file. The `NETGROUP` constraint has the following format:

```
NETGROUP

       BUSGUIDE net_group_name

       SHIELDNET/BUMP name WIDTH value SPACING value

       SHIELDNET/BUMP name WIDTH value SPACING value

       ...

       SHIELDNET/BUMP name WIDTH value SPACING value

END NETGROUP
```

Here:

- `BUSGUIDE` *`net_group_name`* guides the global router as a hard constraint. You must create the net group with `createNetGroup` . `fcroute` will honor the net order specified in NETGROUP in the constraint file.
- `SHIELDNET/BUMP` *`name`* specifies the shield net or bump name and specifies the order for routing. The tool finishes routing in the specified order.
  - Shield net will be floating after routing. The start/end point of shield net is controlled by the bus guide segments.
  - Define the net from left to right of routing accessing the first bus guide segment near to the bump as shown below. `fcroute` honors this order to complete the routing.



- `WIDTH` specifies the width for routing and overwrites the width by other options. It is optional.
- `SPACING` specifies the spacing for routing. It is optional. If it is not specified, min spacing in LEF is used.

After adding the `NETGROUP` constraint in the constraint file, use the `sroutePioBusRoute` configuration variable in the `fcroute` extra config file to control it. The use model is as follows:

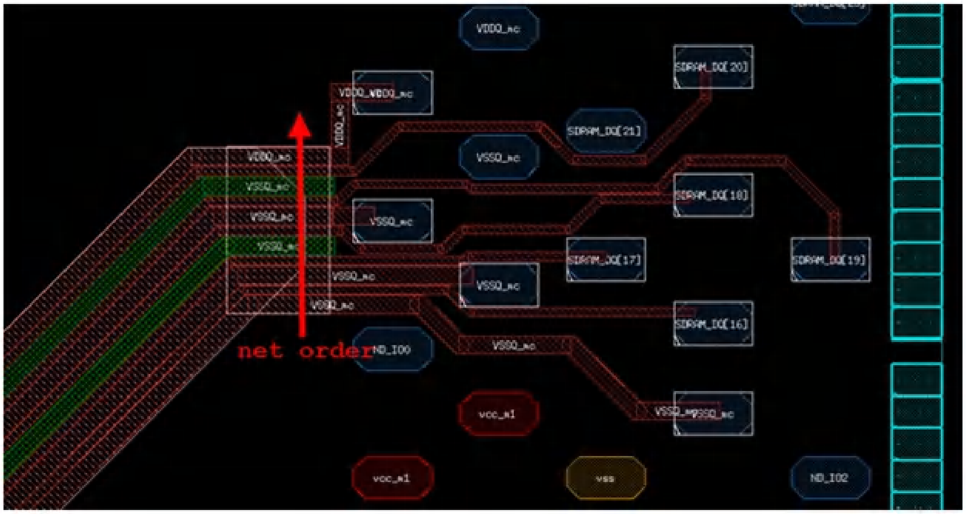
1. Define the `NETGROUP` constraint in the constraint file.
2. Add `sroutePioBusRoute true` in the `fcroute` extra config file. With this setting, `fcroute` will route only the nets and bumps

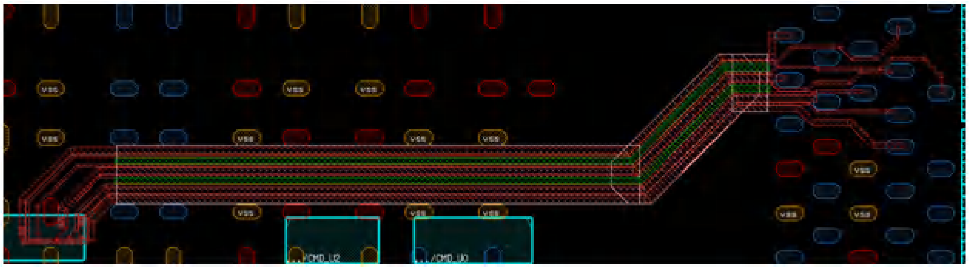
defined in `NETGROUP`.

3. Create a net group with `createNetGroup`.
4. Add the required nets to the net group with `addNetToNetGroup` .
5. Create a bus guide for the net group. You must use orthogonal bus guide segments for the start and end points. Others segments can be either orthogonal or 45-degree.
6. Run `fcroute` with `-selected_bump`, which specifies the bumps to be routed for `NETGROUP`.

Here is an example of how `NETGROUP` constraint can be used:

```
NETGROUP
    BUSGUIDE     ddr_1
         BUMP              Bump_598                    WIDTH 28
         BUMP              Bump_601                    WIDTH 13.2
         BUMP              Bump_602                    WIDTH 28
         BUMP              Bump_606                    WIDTH 13.2
         SHIELDNET         VSSQ_mc                     WIDTH 28
         BUMP              Bump_607                    WIDTH 13.2
         BUMP              Bump_612                    WIDTH 28
         BUMP              Bump_605                    WIDTH 13.2
         SHIELDNET         VSSQ_mc                     WIDTH 28
         BUMP              Bump_610                    WIDTH 13.2
         BUMP              Bump_613                    WIDTH 28
END NETGROUP
```





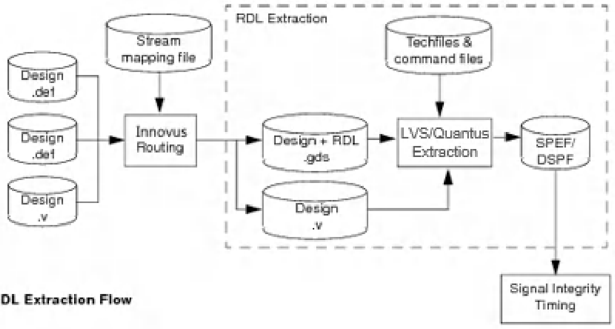
RDL Extraction

In the RDL extraction flow for designs using peripheral I/O methodology, Innovus outputs the design with the RDL routing into a GDS file that is fed into Quantus extraction engine for parasitic extraction at the cell-level. Quantus generates a cell-level SPEF/DSPF file that is used for timing and signal integrity analysis.

There are two steps in parasitic extraction with Quantus.

- LVS is run to perform connectivity extraction.
- Quantus is run to perform parasitic extraction.

The following diagram illustrates this flow.



RDL Extraction Flow

Inputs to Extraction

- Verilog netlist for annotation, generated by Innovus
- GDS of design with RDL, generated by Innovus
- qrcTechFiles

Outputs from Extraction

- Cell-level SPEF/DSPF for SI/Timing analysis, including 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 quantus_coupled.spef`

Make sure all the parasitics of the SPEF are back annotated in Innovus. If all the nets are back annotated, Innovus displays the following message:

```
0 nets are missing in SPEF file.
```

3. Perform timing analysis in Innovus by using the `timeDesign` command.

   ```
   timeDesign -postRoute -reportOnly
   ```

   This command reports worst and total negative slack as well as `register-to-register`, `default`, `register-to-clock-gating`, and `input-to-output` port slacks.

4. Analyze signal integrity by performing SI analysis in Innovus. The SI engine analyzes the design for glitch and SI violations.

   ```
   timeDesign -postRoute
   ```

   This command analyzes the design for SI, creates the analysis report, and reports the worst negative slack path with SI-induced delay.

The following listing is a sample script for signal integrity and timing analysis in Innovus.

```
timeDesign -postRoute -reportOnly
```

6

Hierarchical Flow Capabilities

Partitioning the Design

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.

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.

Flow Methodologies

This chapter covers the following methodologies in the partitioning area:

- Top-down Methodology
- Bottom-up Methodology

Top-down Methodology

The top-down methodology usually consists of top-down planning, implementation, and chip assembly stages. Use this methodology to create a top-level or hierarchical floorplan from a flat floorplan based on fenced modules. In this approach, the die size, shape, and I/O pads locations will drive block and partition placement. Block-level design size and pins will be generated based on the top-level floorplan.

Chip Planning

The following steps describe the most common flow for chip planning, which includes specifying partitions and blackboxes:

1. Import the entire design to be partitioned. Import the design into the Innovus Implementation System (Innovus) environment. You can also include blackboxes.
2. (Optional) Define the blackboxes. If your design has blackboxes that are not specified in step 1, you can define them after reading in the netlist. You can also adjust the size of the blackboxes. For more information, see Saving Blackboxes.
3. Lay out the floorplan. Manually pre-place all modules that will become partitions or blackboxes. You can also use the `proto_design` command for module placement and use the `place_design -concurrent_macros` command for macro placement that places blocks and std_cells together.
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 Early Global Route before this step if you want to run route-based feedthrough insertion. You must also run Early Global 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 Early Global Route. Depending on what stage of the design is in, such as prototyping, intermediate, tapeout, use the appropriate option of the `setRouteMode` command. For example, the `-earlyGlobalRoutePartitionHonorFence` *`list_of_ptn_cell_names`* parameter defines the partition cells in which Early Global Route honors the fence constraints. The `-earlyGlobalRoutePartitionHonorPin` *`list_of_ptn_cell_names`* defines the partition cells in which Early Global Route honors partition fences with single-entry constraints and pre-assigned pins (pins marked `FIXED`) and assigned pins (pins marked `PLACED`).
9. Assign partition pins and blackbox pins using the `assignPtnPin` command.
10. Regenerate the routes that follow assign pins using the `setRouteMode -`

`earlyGlobalRoutePartitionHonorPin` 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 using the `savePartition` command. This creates a directory for each block, and saves its netlist, floorplan, and budgeted constraints to this directory. For top-level designs, this also creates a directory containing the top-level netlist, floorplan, simple timing model, and physical abstract for each partition block or blackbox. Subsequent work should be done in these block-level and top-level directories for implementing the block-level and top-level designs, respectively.
    Tip: You should do all design work in each saved partition directory, including the top-level directory.

    Note: For Integrated Hierarchical Database (iHDB), save partitions with a specified module tag (`savePartition -module_model_tag`). This creates pnr and lef module models for each partition block. For top-level design, this also creates a pnr module model. These block-level and top-level pnr models should be used for implementing the block-level and top-level designs, respectively. For example if the design dtmf has the ptn_wrapper partition and the default global data directory is DATA2. With `savePartition -module_model_tag init`, a pnr model will be created for top-level design in the following directory: DATA2/dtmf/init/pnr.
    For partition ptn_wrapper, a pnr and lef models will be created in the DATA2/ptn_wrapper/init/pnr and DATA2/ptn_wrapper/init/lef directories.

Tip: You should do all design work in each saved partition directory, including the top-level directory.

Implementation

After the chip planning is complete, the next stage is to implement the individual blocks. The detail of each block is implemented using the constraints for timing, size, and pin assignment determined during the planning stage. Block implementation must be done in the block directory 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.

Note: For Integrated Hierarchical Database (iHDB), block implementation must be done with the pnr model that was generated by the `savePartition` step. To restore a block or top-level design, `restore_module_model -cell cell_name -tag` *`tagName`* should be used. At the completion of this step, all needed models such as 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.

⚠ For information on chip assembly in the Integrated Hierarchical Database (iHDB) flow, see Chip Assembly in iHDB Flow.

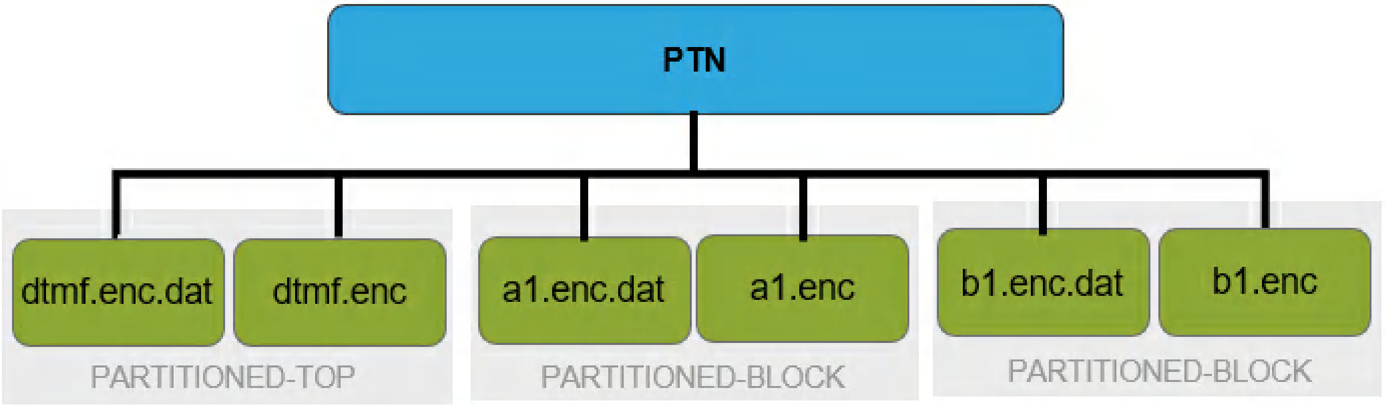
Note: Before using the `assembleDesign` command, for each design, save the top-level and the block-level designs using the `saveDesign -def` command. It is recommended that you save the design with a def file so that it can be used for reassembling the design using the DEF merge capability for a fast turnaround time.

As an example, consider a design called dtmf that has two partitions: a1 and b1. After running the `partition` command, the partition directories are saved under the PTN directory. You would, therefore, implement the following:

- top-level design dtmf
- a1 block

- b1 block

The design files are a1.enc and a1.enc.dat for a1 block and b1.enc and b1.enc.dat for b1 block. The following figure shows the directory structure:



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

- Concatenates the Verilog netlist files from the partitions back to the top level
  Note: The partition netlists and top-level netlist are changed from the time the save partition step was performed.
- Merges the design data with the original top design level. By default, data from DEF files is used. However, you can use the `-fe` parameter to specify that Innovus data should be used. You can also use data in the OpenAccess database format.
- Rows at top-level design will be cut, and the rows at block-level design will be brought back
- Preserves scan chain information at partition block-level design, therefore minimizing the floorplan data loss during partition and assemble design cycle. The start and stop scan chain points at partition block I/O pins are adjusted back to instances that connect to scan chain points. Top-level scan chains are not connected to block-level scan chains.

Note: You must run the `assembleDesign` command from the directory that contains the full chip-level floorplan for the top-down hierarchical flow.

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

```
assembleDesign -topDir PTN/dtmf.enc.dat -blockDir PTN/a1.enc.dat -blockDir
PTN/b1.enc.dat
```

This assembles the entire design. You can also use the `assembleDesign` command to bring back specified block data from OpenAccess database. Here is an example:

```
assembleDesign -topDesign testOALib DTMF layout -block testOALib ptn1 layout -block
testOALib ptn2 layout
```

In this example, the OpenAccess database top-level library is testOALib, the top-level cell name is DTMF, and the top-level view is layout. Two blocks, ptn1 and ptn2, have been specified.

Note: The `assembleDesign` command supports rectilinear partitions. It also supports nested blackboxes for the place-and-route data (`-fe` parameter) and the OpenAccess database. However, because blackbox information cannot be specified in a block-level DEF file, nested blackboxes are not supported for the DEF flow.

Chip Assemby for iHDB Flow

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 for the Integrated Hierarchical Database (iHDB) flow can be done with the following steps:

1. Restore the top-level design using `read_module_model` command.
2. Specify the block-level designs with pnr type model that will be assembled with the top-level design using `set_module_model` command. Wildcard can be used to specify all blocks at top-level.
3. Run `commit_module_model -mmmc_file` *`fullChipViewDefinitionFile`* command to load all specified pnr models and merge them with top-level design.

Once the top and block level designs are assembled together:

- Rows at top-level design will be cut, and the rows at block-level design will be brought back.
- Scan chain information at partition block-level design is preserved, therefore minimizing the floorplan data loss during partition and assemble design cycle. The start and stop scan chain points at partition block I/O pins are adjusted back to instances that connect to scan chain points. Top-level scan chains are not connected to block- level scan chains.

Note: Before using the iHDB flow, for each design, save the top-level and the block-level designs using the `create_module_model -tag` *`tag_name`* command.

For example, consider a design called dtmf that has two partitions: a1 and b1 and the global data directory is DATA (can be set using the `set_module_model -default_dir` command). After running the `partition`/`savePartition -module_model_tag init` command, the partition pnr models are saved under the DATA/<cellName>/init/pnr directory.

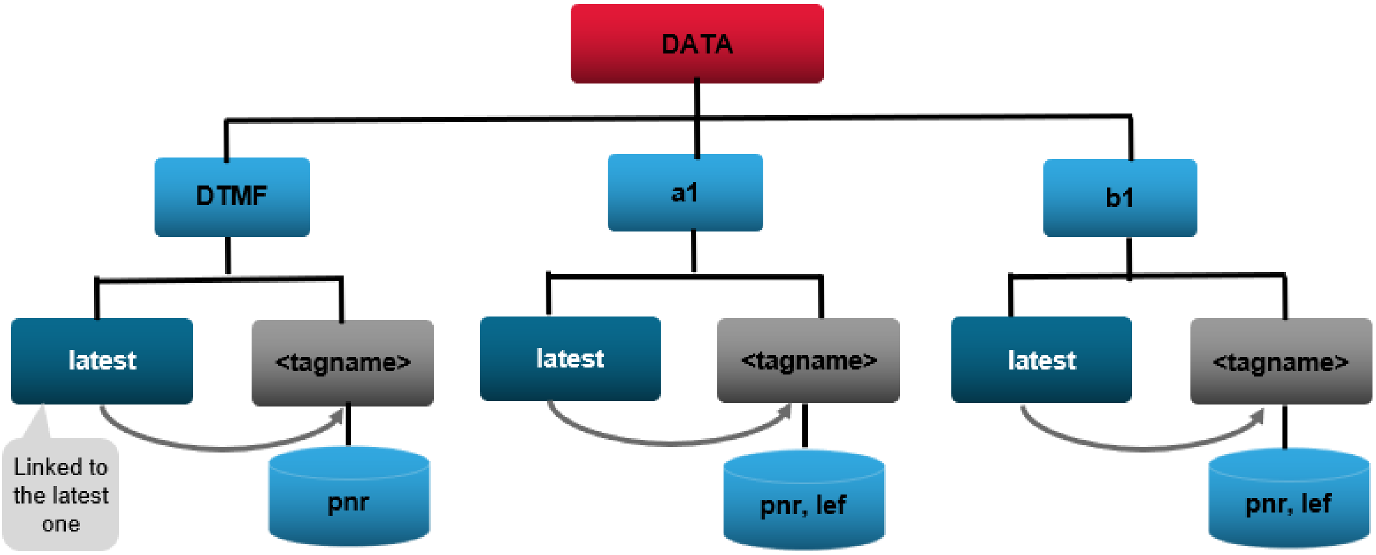
You would, therefore, implement the following:

- top-level design dtmf
- a1 block
- b1 block
- Then assemble top and block design

Example of a chip assemble run script:

```
set_module_model -default_dir /myproject/DATA
read_module_model top -tag tag_name
set_module_model -cell * -tag tag_name -type pnr
commit_module_model -mmmc_file fullChipViewDefinititionFile
```

The following figure shows the directory structure:



⚠ **Note:** Chip assembly with Integrated Hierarchical Database does not support the OpenAccess database yet.

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 following is a sample report that displays if the design has overlap and Master-Clone alignment violations:

The size of a block-level design can be derived or adjusted using the *Floorplan - Specify Floorplan* menu command or the `floorPlan` command. The Innovus software can support a rectilinear block level design.

You can run the `assignIoPins` command to assign I/O pins based on placement information. You can specify initial I/O pin placement in an I/O constraint file. For more information, see the Generating the I/O assignment File section in the "Data Preparation" chapter of the *User Guide*. You can read in the I/O constraint file into the Innovus environment during the design import step, or use the `loadIoFile` command after reading in the netlist. If an I/O constraint file does not exist, an initial I/O pin placement can be derived from cell placement. After placing macros and standard cells, the placer can internally call the `assignIoPins` command to place I/O pins based on current cell placement. By default, pins are placed under power areas on different layers.

Note: The placer internally calls the `assignIoPins` only when you specify the `setPlaceMode -place_global_place_io_pins true` command.

Note: Use the `setPlaceMode -place_global_place_io_pins` {true | false} command to disable I/O pin assignment during placement.

After I/O pins have been assigned, you can further refine the current I/O pin assignment by doing either of the following:

- Adjust pins (using the *Pin Editor* or the `editPin` command). You can also use direct pin manipulation to manually move selected pins to different locations.
- Run incremental pin assignment by running the `assignIoPins` command. This command honors fixed pins and re-assigns only the ones that have a placed or unplaced status.

Note: The `loadIoFile` command automatically sets the I/O pin placement status to fixed. For the pins that need to be re-assigned, you must change their pin placement status.

You can use the `legalizePin` command to resolve pin overlaps or pins off-grid.

Top-level Implementation

The following is a sample report that displays if the design has overlap and Master-Clone alignment violations:

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.

Note: You can use the following command to generate the design abstract (LEF) information for the block-level design:

- `write_lef_abstract` command
- `create_module_model -type lef` command (for iHDB flow)

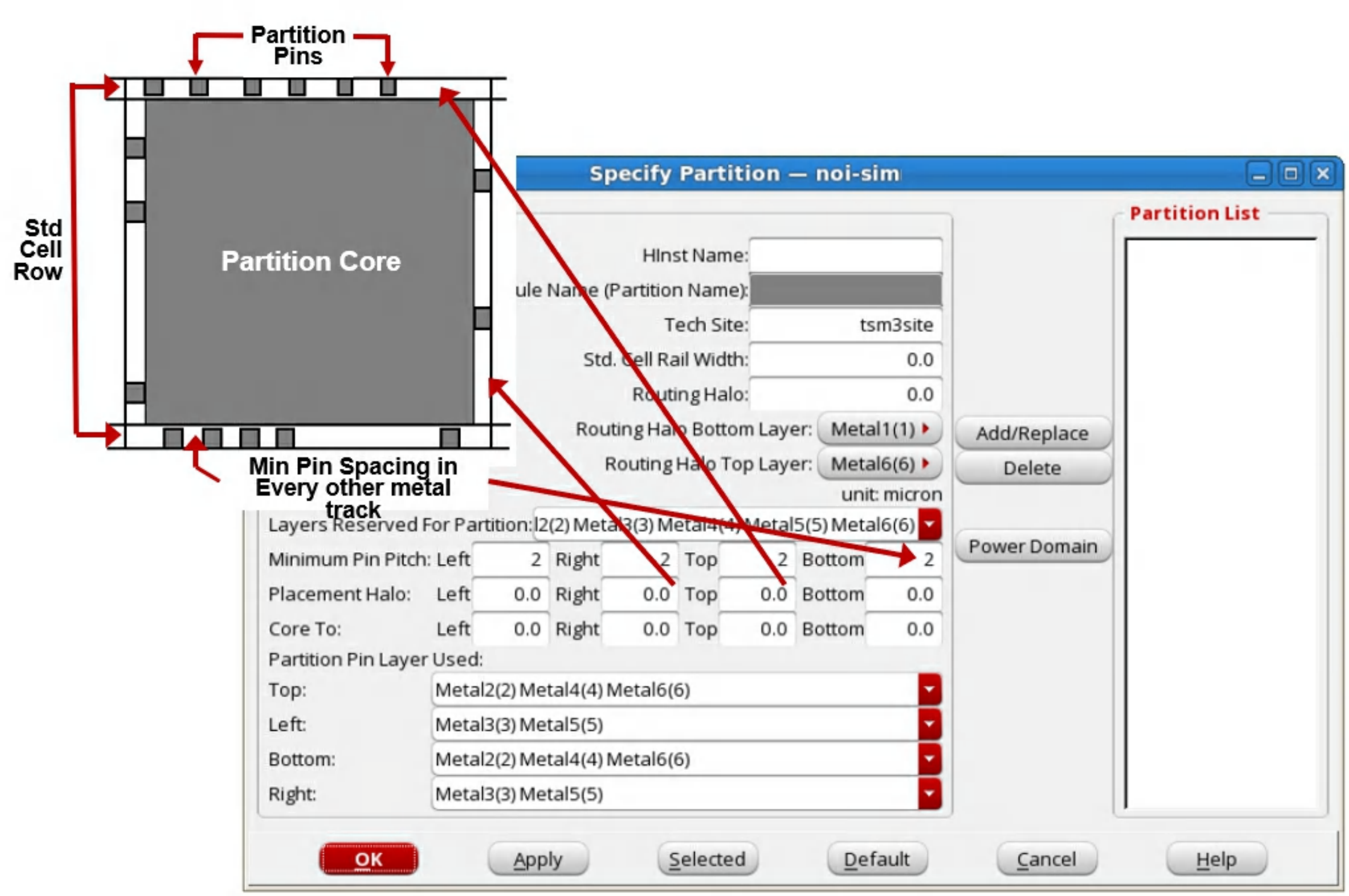
Chip Assembly

For the bottom-up approach, see Chip Assembly, to bring together all the top-level and block-level netlists and routing information.

Specifying Partitions and Blackboxes

Defining Partitions

To designate partitions, use the `definePartition` command and the *Specify Partition* form. The following figure shows an example of how some of the fields in the *Specify Partition* form relate to the partition. For a description of all the fields, see the Partition Menu in the *Menu Reference*.



HInst Name:
ule Name (Partition Name):
Tech Site: tsm3site
Std. Cell Rail Width: 0.0
Routing Halo: 0.0
Routing Halo Bottom Layer: Metal1(1)
Routing Halo Top Layer: Metal6(6)
unit: micron
Layers Reserved For Partition: l2(2) Metal3(3) Metal4(4) Metal5(5) Metal6(6)

Minimum Pin Pitch:	Left	2	Right	2	Top	2	Bottom	2
Placement Halo:	Left	0.0	Right	0.0	Top	0.0	Bottom	0.0
Core To:	Left	0.0	Right	0.0	Top	0.0	Bottom	0.0

Partition Pin Layer Used:
Top: Metal2(2) Metal4(4) Metal6(6)
Left: Metal3(3) Metal5(5)
Bottom: Metal2(2) Metal4(4) Metal6(6)
Right: Metal3(3) Metal5(5)

To specify a module as a partition, complete the following steps:

1. Move the module inside the core area. You can manually move a module, or use the `setObjFPlanBox` command, to define a new module boundary with its coordinates in the core area.
   Note: A blackbox is a special partition where this restriction does not apply.
   Note: You cannot create donut shaped objects during the partition flow.
2. Specify the name of the partition.
3. Specify the instance name of a module that is to become a partition.
   Note: For cases 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. 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 Metal1,Metal2, Metal3, Metal4, and Metal5 selected, and Metal6 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. 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.
10. Add the partition information to the *Partition List* field.

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 -keepEmptyModule true` command before importing a design.
- Once the design is imported, specify a module or hard macro as blackbox using the `specifyBlackBox` command or the *Specify Black Box* form.

  Note: Converting a hard macro into a blackbox will not update the blockage definitions when you change the blackbox size.

- Define LEF abstracts for blackboxes. You can specify a blackbox library in the LEF Files field of the *Design Import* form. If a blackbox LEF abstract is specified in the LEF Files field, the LEF abstract should have CLASS type as BLOCK BLACKBOX to indicate it is a blackbox. The following is an example of a blackbox LEF abstract:

  ```
  MACRO amba_dsp

  CLASS BLOCK BLACKBOX ;

  ORIGIN 0 0 ;

  SIZE 4411.8600 BY 5697.3600 ;

  END amba_dsp
  ```

After defining a blackbox with any of the above methods, you can further modify an existing blackbox size with the `specifyBlackBox` command.

Note: You can use the `getBlackBoxArea` command to retrieve the standard cell area, macro area, and cell utilization value for the specified blackbox.

Warning: If you convert a hard macro into a blackbox or define a blackbox with a LEF abstract that has obstructions, the obstructions size will not be updated with a new blackbox size. Due to this limitation, obstructions may be intruded outside of the new blackbox boundary.

Blackbox Flow

Note: Even though there are more than one ways to define a black box, it is recommended that you define a black box by using the `specifyBlackBox` command.

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

1. Import the design. By default, the Innovus software keeps empty modules (`setImportMode -keepEmptyModule true`)
2. Specify the blackboxes or load a floorplan file with blackbox information.
3. Floorplan the design.
4. (Optional) Save the design using the `saveDesign` command. This saves the blackbox information. For the iHDB flow, save the design by running `create_module_model -type pnr` command.
5. Run placement.
6. (Optional) Run Early Global Route.

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 Early Global Route, use the `assignPtnPin` command to finally place blackbox pins to honor user-specified constrains. When you partition the design, blackboxes as well as regular partitions are committed. Blackboxes get converted to hard macros at top-level design that display as a Block object in the *Attribute Editor.* The following flow is an ECO flow where the contents of the black box are now well defined.

1. Restore the design (or import the design and load a floorplan with the black box information) using the `restoreDesign` command. For the iHDB flow, use the `restore_module_model` command.
2. Run the `loadBlackBoxNetlist` command to incrementally load the netlist for the blackbox. You can run this command without exiting the current session of the Innovus software.
3. Run the `convertBlackBoxToFence` command to convert the blackbox to a fence.
   Note: To convert the fence back to a blackbox, run the `convertFenceToBlackBox` command.
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. For the iHDB flow, use the `-type pnr` command.

Reshaping Blackboxes

The following is a sample report that displays if the design has overlap and Master-Clone alignment violations:

During `proto_design`, 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.

Removing Blackbox Specifications

The following is a sample report that displays if the design has overlap and Master-Clone alignment violations:

A blackbox can be unspecified by using the `unspecifyBlackBox` command. If the blackbox is an empty module in the netlist, then you can also convert it to a partition fence using the `convertBlackBoxToFence` command.

Tip: You should not delete a blackbox that was originally defined as a macro in the technology file; otherwise, you might have problems with loosely integrated applications because these application interfaces automatically generate only macro definitions for blackboxes. You should only use the delete capability to try out different floorplan.

Handling of Blackboxes with Non-R0 Orientation

The partitioning- and blackbox-related commands in Innovus support only those blackboxes whose master instances have an R0 orientation. Clones with a non-R0 orientation clones are, however, supported. Partitioning-related commands such as `assignPtnPin`, `partition`, `assembleDesign`, `flattenPartition`, `convertBlackBoxToFence`, and `editPin` work only with those blackboxes whose master instances have an R0 orientation. Several commands in the Innovus software automatically convert the orientation of master blackboxes to R0. In addition, you can also run the `changeBBoxMasterToR0` command to convert the orientation of the master blackboxes to R0. This would be useful for example, you restore a design and want to convert the orientation of all the master blackboxes to R0.

Note: You can check the orientation of partitions and blackboxes in a design by right-clicking a partition/blackbox and selecting the *Show Partition Orientation* option from the context menu.

Note: You can use the `changeBBoxMasterFromR0` command to change the orientation of the master blackboxes from R0 to a different orientation.

The following sections provide additional 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`
- `placeInstance`
- `proto_design`

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.

The conversion includes the following:

- Cell blackbox geometries (PORT, OBS, and so on) are transformed.
- Master instances are converted to R0 orientation. The clone instances are oriented accordingly.
  Note: The placement location remains unchanged.
- Any pin guides, pin blockages, and pin constraints associated with transformed blackboxes are deleted.
  Note: There is no change in the design physically as a result of these transformations. Only the cell orientation and the instance representation are modified.

As an example, if the blackbox master instance is MX, then after the transformation:

- Cell geometries are transformed to MX
- The orientation of the master instance is changed to R0.

Performing R0 Transformation

For designs that contain blackboxes whose master instances have a non-R0 orientation, you can use the `changeBBoxMasterToR0` command to convert the orientation of the master blackboxes to R0.

The syntax of the command is as follows:

```
changeBBoxMasterToR0 [-checkOnly] [{cellName | cellNameList}]
```

If `cellName`, or `cellNameList`, is not specified, the command converts the orientation of all the non-

R0 master blackboxes to R0. If the `-checkOnly` parameter is specified, the command does not actually convert the orientation of any master blackbox; it only displays the number of master blackboxes whose orientation would have been changed had the command been run without the `-checkOnly` parameter. When you are ready to run a loosely integrated application, complete the following steps:

- Run the `saveDesign` command to make sure that you have updated the size and pin information. For the iHDB flow, run the `create_module_model -tag` *`tag_name`* command.
- Exit the Innovus software.
- Rerun the Innovus software with the updated macro information.

To delete all the blackboxes in the design, use the `unspecifyBlackBox -all` command.

Specifying Multiple Instantiated Partitions and Blackboxes

When a module with multiple instantiations (also known as repeated partitions) of the same cell type is assigned to become a partition, you can specify either one of the multiple instantiated hierarchical instances to be partitions. The name of a hierarchical instance used for partition specification becomes the master partition, and the other instantiations are clones of this master partition.

When working with repeated partitions, you should be aware of the following:

- You can only specify one instance as a master partition. The Innovus software will treat the other instances are partition clones.
- For the bottom-up hierarchical flow, where the block is implemented first, make sure all the non-uniquified instances are placed inside the core before you specify the partition.
- For non-uniquified blackboxes, the Innovus software automatically converts all hierarchical instances of a same module as repeated blackboxes. The hierarchical instance that is first instantiated in the netlist is treated as the master blackbox.
- Partition and blackbox clones can be rotated and flipped even if the vias used in the design are not square. The `assembleDesign` command will create the required symmetry of the via if its definition is missing in the LEF.
  Note: For the iHDB flow, use the following commands to create the required symmetry:

  ```
  set_module_model -cell cell_name -type pnr
  ```

  ```
  commit_module_model
  ```

- Partition clones share the same pin assignment and pushed-down data as their partition master, you must run the `alignPtnClone` command before the commit partition step to make sure all the partition clones are well aligned with the master on power mesh so you do not run into problems when flattening the partitions.

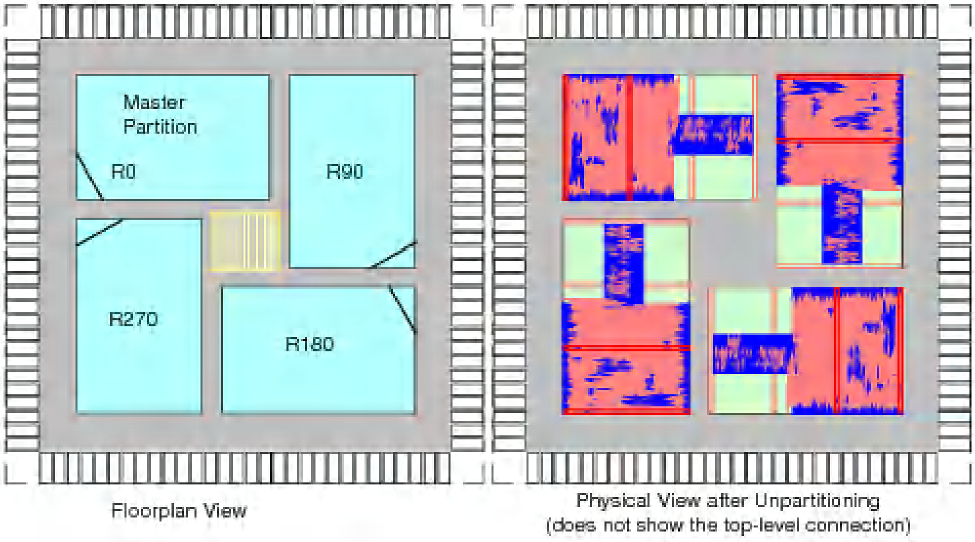
- For master and clones partitions, the Innovus software automatically snaps the clone partitions such that clones will have the same row structure and pattern as their master. To disable this snapping capability, use the `-noEqualizePtnHInst` option of the `loadFPlan` command.

Changing Partition Clone Orientation

After specifying the partition, you can change the partition clones' orientation by using the `setClonePtnOrient` command or through *Attribute Editor* during floorplanning. Use the `getClonePtnOrient` command to retrieve orientation information of a specific partition clone. For routing purposes, the Innovus software automatically stitches regular wires and rotates vias correctly for non-R0 orientations, such as MX, MY, R180, and R270. For example, there is a case where some of the clones follow the orientation of the master instance (R0), and some are placed with R180 orientation. After chip assembly, the Innovus software flips and places the clone instances' standard cells to match the R180 clone orientation, and repositions the routing according to the R180 orientation.

Note: The R90, MX90 ,MY90 and R270 orientation clones are not supported because they have have vertical rows.

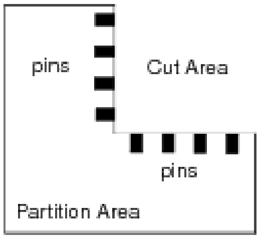
The following example shows a design that has R90, R180, and R270 orientation clones:


Master
Partition
R0
R90
R270
R180
Floorplan View
Physical View after Unpartitioning
(does not show the top-level connection)

Note: The illustration above only shows the wire information inside the partition, and does not include the top-level connection.

Specifying Rectilinear Partitions and Blackboxes

You can specify a rectilinear (non-rectangular) partition shape by adding a cut area. The partition's cut area will have no cell placement and no routing. Pins are assigned to the rectilinear partition edges, as shown in the following figure:



The rectilinear pin assignment recognizes the rectilinear edges when assigning pins, and supports any rectilinear shape. See Assigning Pins on Rectilinear Edges for more information.

To add a cut area to the partition or blackbox, complete the following steps:

- Click on the *Cut Rectilinear* widget from the *Tools* area.
- Move the mouse to an edge or corner of the partition or blackbox.
- Left click and drag over the area.
- Left click again to complete the cut.

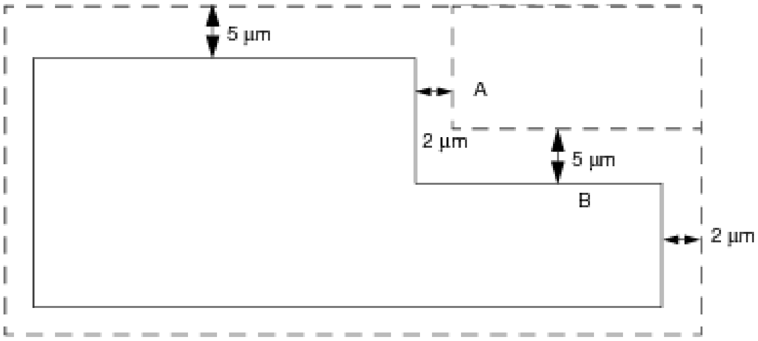
For the top-level partition, the cut area allows block or cell placements. The equivalent text command is `setObjFPlanBoxList` with the Module object type. For backward compatibility, you can also use the `createPtnCut` command. You should specify a module as a partition before using the `createPtnCut` command. 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/partition that has pins assigned to it, the affected pins are automatically moved to the new edge boundary created by the cut.

Specifying Core-to-I/O Distance for Partition Cuts

Core-to-I/O distance is specified in the *Specify Partition* form. If the partition has a partition cut, core-to-I/O distance is honored where the cut is specified. The specified top, bottom, left, and right core-to-I/O distances is automatically assigned for the cutting edges that face the north, south, west, and east side, respectively.

For example, if you specify a core-to-I/O distance of 5 µm for the top and bottom, and 2 µm for left and right sides:



The core to I/O distance for the edge A (facing east) should be 2 µm. The core to I/O distance for the edge B (faced to north) should be 5 µm, same as the top side.

Displaying All Partitions

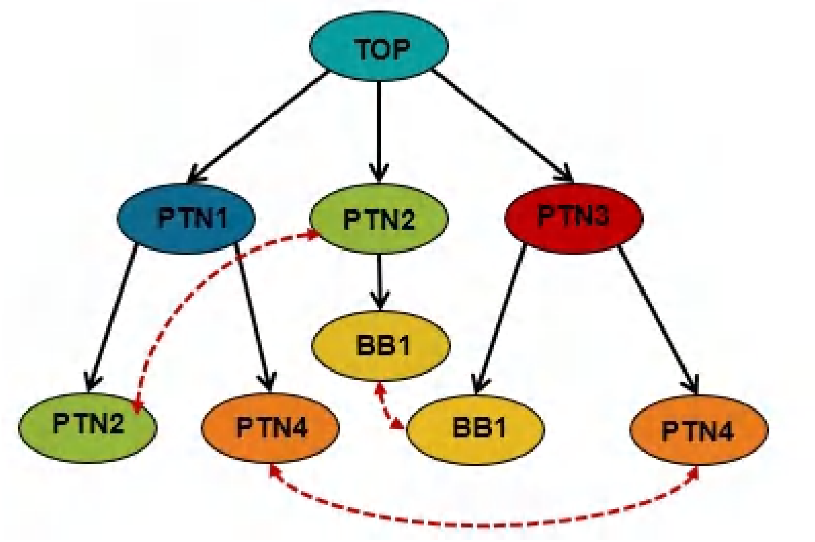
You can use the *Show All Partition* context-menu to display all the partitions in the design including any hidden ones.

Working with Nested Partitions

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.

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
```

or

```
definePartition -hinst PTN2
definePartition -hinst PTN1
```

Inserting Feedthroughs in Nested Partitions

The `insertPtnFeedthrough` and `showPtnWireX` commands support nested partitions. The `insertPtnFeedthrough` command removes all parent level partitions and does feedthrough on the bottom most level of partitions as if in a single level of partitions. For inserting feedthrough buffers into nested partitions, the `insertPtnFeedthrough` command, first, automatically deletes the parent partitions and keeps only the bottom most partition. It then does a single level feedthrough insertion and then automatically restores the parent partitions and brings the nested partition definition back as it is.

Note: In nested designs, the `insertPtnFeedthrough` command only puts buffers and ports in the bottom most level of partitions.

Pin Assignment Across Nested Partitions

The `assignPtnPinassign_partition_pins` 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 `editPinedit_pin` command is used for manual pin assignment in nested partitions. It places pins of nested partitions similar to single level partitions. Early Global 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 `setRouteModeset_db -earlyGlobalRoutePartitionHonorFence` command 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.

For example (iHDB Flow):

```
restore_module_model
# child cells model need to explicitly set in nested design
foreach k {parent child} {set_module_model -cell $k -type pnr}
commit_module_model
```

Assigning Pins

You can optimize partition and blackbox pins in the Innovus environment based on routing or placement information. You can assign the pins or ports to a location on a partition, and set various constraints as per your requirements on pin assignment, for example, you can create pin blockages on specified areas. Run the *Check Pin Assignment* menu command of the *Partition Menu* or the `checkPinAssignment` command after pin assignment to make sure that all pins are assigned, are placed on routing grids, and are not overlapping. Blackbox pins are assigned in the same way as partition pins. Pin assignment supports the following:

- Rectilinear partitions and black boxes
- Repeated partitions and black boxes. Both master and clones are considered when assigning their pins.
- Designs with an arbitrary origin.
- Non-uniform tracks.
- Automatic pin alignment across feedthrough path.

Note: Pin assignment assigns only signal pins but it does honor power/ground stripes and follow pins. Power and ground pins are created when the design is partitioned.

Note: The pin assignment commands have been updated to honor the preferred routing layer attributes as soft constraint during pin assignment. Pin assignment commands try to honor the `setAttribute -top_preferred_routing_layer` and `-bottom_preferred_routing_layer` attributes. However in case they cannot be honored, the pin is assigned to any allowed layer.

The following sections describe pin assignment in Innovus:

- Checking the Feasibility of Pin Assignment
- 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
- Assigning Pins for Bus Guides

Checking the Feasibility of Pin Assignment

Before assigning pins to the design, you can check the design to identify potential issues that may arise while assigning pins. You can use the `-preCheck` option of the `checkPinAssignment` command to check for the feasibility of pin assignment mainly to catch the reason for unplaced pins in fully abutted design.

Note: No other `checkPinAssignment` command parameter can be used if the `-preCheck` option is being used.

The `-preCheck` parameter checks for the following:

- Partition Fence Overlap Violation: It checks for any partition fences which are overlapping and reports the violation.
- Layer Violation: It checks for allowed layers on abutted edges of partition.
- Spacing Violation: it checks for spacing on abutted edges.
- Block Pin Violation: It checks if the block pin is placed on an abutted edge with its connected partition.
- Multi Partition Pin Net Violation: It checks for multi partition pin nets in abutted design.
- Master-Clone Pin Connection Violation: It checks if symmetric abutted designs have each master-clone pair connected to same pin.
- Master-Clone Orientation Violation: Its check for the clone inside clone orientation. The clone inside a clone should have a relative orientation with its parent which is checked with master/clone inside master.
- Master-Clone Net-Group Violation: It checks the net group ordering in master and clone partitions.
- Master-Clone Symmetric Violation: It checks if the clone to clone offset (relative distance on abutted edges) is same as in the master to master/clone connection pair. This is required so that the pins in each clone-clone and master-master/clone pair is the same.
- Master-Clone alignment Violation: It checks for routing track offset in Master-Clone design to

avoid off grid pins.

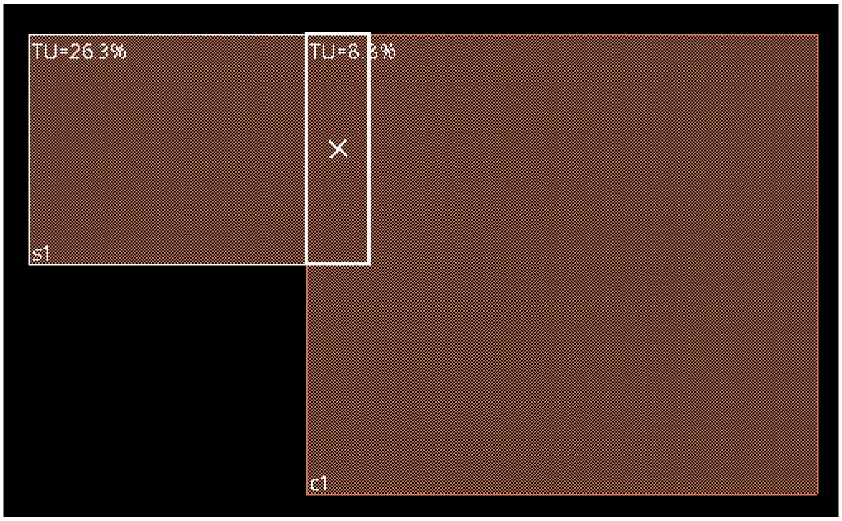
- Master-Clone size Violation: It checks the master-clone size. This is required to check the number of poly edges of clones with master and check each edge size for any mismatch.
- Master-Clone align-zone Violation: It checks for the common alignment zone between master-clone and their connected partitions.

Example

The following command checks the feasibility of assigning the pins in the design:

```
checkPinAssignment -preCheck#% Begin checkPinAssignment (date=02/26 13:51:22,
mem=737.7M)
                     There are overlap violations found. Correct it first. All other
violation results will not be correct with overlap violation. Check detail report of
overlap violation in file test_preCheck.rpt

--------------------------------------------------------------------
                          Summary of pin pre checks
--------------------------------------------------------------------
Partition Fence Overlap Violation              |     1 |
Layer Violation                                |     0 |
Spacing Violation                              |     0 |
Block Pin Violation                            |     0 |
Multi Partition Pin Net Violation              |     0 |
Master-Clone Pin Connection Violation          |     0 |
Master-Clone Orientation Violation             |     0 |
Master-Clone Net-Group Violation               |     0 |
Master-Clone Symmetric Violation               |     0 |
Master-Clone alignment Violation               |     0 |
Master-Clone size Violation                    |     0 |
Master-Clone align-zone Violation              |     0 |
---------------------------------------------------------------------
#% End checkPinAssignment (date=02/26 13:51:22, total cpu=0:00:00.0, real=0:00:00.0,
peak res=737.9M, current mem=737.9M)
```

The violation marker is also displayed in the design:



The detailed report can be used to fix the violations:

```
Partition Fence Overlap Violation : Fence of hierarchical Instance 'c1' is overlapping with fence of
hierarchical instance 's1'. Recommended to change the area/position of fence to avoid this overlap
```

All the violations must be fixed before pin assignment is carried out.

Fixing PreCheck Issues

You can reduce the issues reported by the `checkPinAssignment -preCheck` command and improve the pin QOR by automatically by using the `alignPtnClone` command to doing the following:

- Fix the fence overlaps for partitions
- Fix the clone orientation to make designs symmetrical
- Fix the relative location and orientation of a set of target partitions based on reference partitions.

Fixing Partition Fence Overlap Violation

The `checkPinAssignment -preCheck` command reports partition fence overlap violations. The `-snapAllCorners` parameter of the `alignPtnClone` command can be used to fix partition overlaps.

Fixing Master-Clone Symmetric Violation

The `checkPinAssignment -preCheck` command checks if the clone to clone orientation and offset (relative distance on abutted edges) is same as in the master to master/clone connection pair for maintaining the symmetry . This is required so that the pins in each clone-clone and master-master/clone pair are the same. The `-symmetricOrientation` parameter of the `alignPtnClone` command can be used to check the orientation between each pair of partitions, and fix the symmetry by finding the optimum symmetric orientation of clones.

Note: These symmetry violations are reported under the "Master-Clone Symmetric Violation" section of the report.

Fixing the relative location and orientation of a set of target partitions based on reference partitions

The `alignPtnClone` command can be used to fix the relative location and orientation of the target partitions based on the reference partitions using the following parameters:

- `-symmetryPatternReference` *`string`*

  Specifies the names of the partitions that are used as a reference set to maintain a symmetrical pattern for other set of clone partitions in the group. The first partition in the list is considered as the seed partition from which all the relative locations and orientations are calculated for the rest of the partitions in the list.

- `-symmetryPatternTarget` *`string`*

  Specifies the names of the partitions that are used as a target set to maintain a symmetrical pattern relative to the reference set of partitions in the group.

  Note:

  - The number of partitions and their order must be the same as that specified with the `-symmetryPatternReference` parameter.
  - The current location and orientation of the seed partition is used to relatively move the other partitions in the list. The first partition in the list is considered as the seed partition from which all the relative locations and orientations are calculated for the rest of the partitions in the list.
  - You can specify master partitions in the target partitions set list. However, master partitions can only have R0 orientation.

- `-targetOrientation {R0 R90 R180 R270 MX MX90 MY MY90}`

  Specifies the new orientation of the seed partition of the target partition set (`-symmetryPatternTarget`). The specified orientation overrides the current orientation of the seed partition and enables you to maintain the relative orientation of the different pairs of clones.

  Notes:

  - The seed partition is the first partition specified with the (`-symmetryPatternTarget`) parameter from which all the relative locations and orientations are calculated for the rest of the partitions in the list.

- The orientation of the master partitions cannot be changed.

- `-targetOrigin {x y}`

  Specifies the new origin of the seed partition of the target partition set (`-symmetryPatternTarget`). The specified location overrides the current location of the target seed partition.

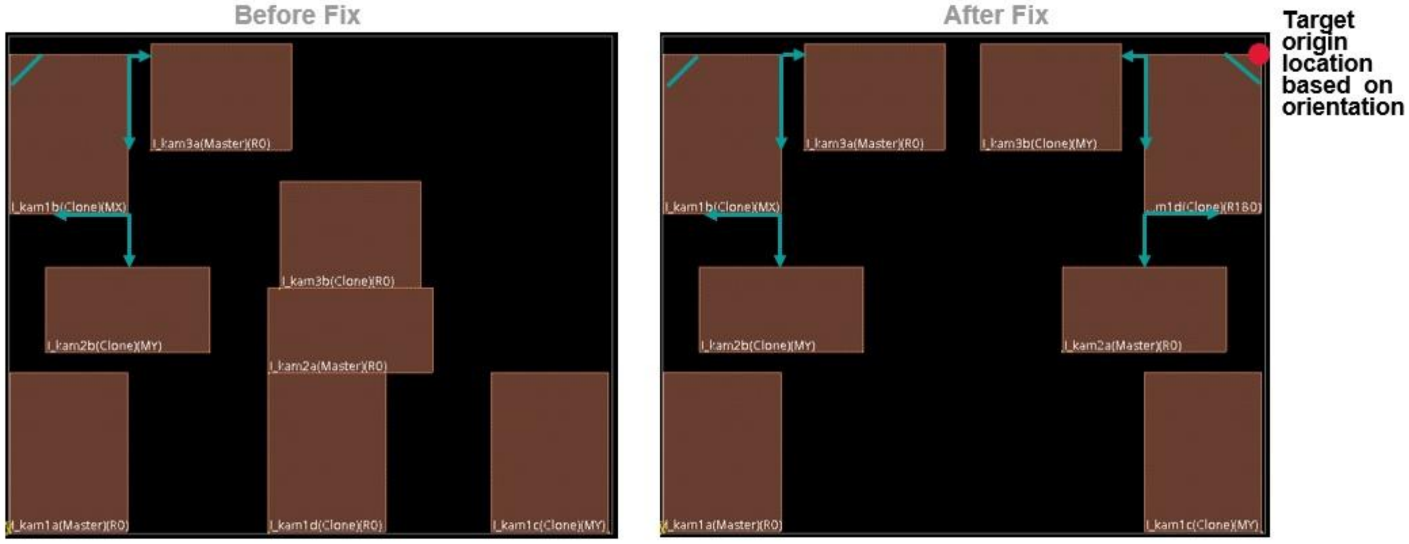
  Note:

  - It does not let the target seed partition go outside the core boundary.
  - The seed partition is the first partition specified with the (`-symmetryPatternTarget`) parameter from which all the relative locations and orientations are calculated for the rest of the partitions in the list.

Example:

The following command fixes the relative location and orientation of the target partitions based on the reference partitions:

```
alignPtnClone -symmetryPatternReference {I_kam1b I_kam3a I_kam2b} (-
symmetryPatternTarget) {I_kam1d I_kam3b I_kam2a} -targetOrigin {51.1 42.845} -
targetOrientation R180
```



Assigning Partition and Blackbox Pins

Assigning pins for partitions and blackboxes includes the following steps:

- Setting Pin Constraints
- Assigning Pins
- Validating Pin Placement Results
- Refining Pin Assignment and Fixing Pin Violations
- ECO Pin Assignment

Setting Pin Constraints

The Innovus software provides a number of constraints to control or guide partition, blackbox, or I/O pin assignment:

- Pin Group
- Net Group
- Pin Guides
- Pin Size (Width and Height)
- Pin Spacing
- Pin Layers
- Pin-to-corner distance
- Pin Blockage

Pin Group

While assigning bus pins or signal pins that you want to be placed together, you can specify a constraint for these pins by creating a cell pin group. You can create a cell pin group with the `createPinGroup` command or by using the *Edit Pin Group* form (*Edit - Edit Pin Group*). You can add pins to a cell pin group with the `createPinGroup` command or with the `addPinToPinGroup` command. Cell pin groups do not have to be associated with a partition pin guide because a pin group is not a constraint on any partition edge. In this case, the pin assignment program can freely place this group of pins on any edge of the partition. However, pins that belong to this pin group are still placed together in adjacent locations. With a pin group you can:

- Optimize the order of pins within a cell pin group to improve wire length using the `-optimizeOrder` option of the `createPinGroup` command. If this option is not specified, the pin order is exactly as specified in the pin group.

- Specify pin spacing. The default minimum pin spacing between pins of a cell pin group is two tracks.

The following commands create a pin group `pGroup1` that consists of `3 INT` bus bit pins of the module `ALU`. These pins can be optimized within the pin group:

-
  ```
  createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -optimizeOrder
  ```
  or

-
  ```
  createPinGroup pGroup1 -cell ALU -optimizeOrder
  addPinToPinGroup -cell ALU -pinGroup pGroup1 -pin {INT[0] INT[2] INT[3]}
  ```

Use the `deletePinGroup` command to delete a pin group or all pin groups and use the `deletePinFromPinGroup` command to delete a pin from a pin group.

Net Group

You can create a net group using the `createNetGroup` command or by using the *Edit Net Group* form (*Edit - Edit Net Group*). You can specify net members when creating a net group or add them later using the `addNetToNetGroup` command. To be honored by pin assignment, net groups must be used in conjunction with a pin guide. As for a pin group, you can optimize the net pin order, alternate the pin layers, and specify pin spacing for a net group. The following commands create a net group `nGroup1` that has two nets `NET1` and `NET2` with minimum pin spacing of 2 tracks.

-
  ```
  createNetGroup nGroup1 -net {NET1 NET2} -spacing 2
  Or
  ```

-
  ```
  createNetGroup nGroup1 -spacing 2
  addNetToNetGroup nGroup1 -net {NET1 NET2}
  ```

-
  ```
  create_net_group -name nGroup1 -nets {NET1 NET2} -spacing 2
  Or
  ```

-
  ```
  create_net_group -name nGroup1 -spacing 2
  update_net_group -name nGroup1 -add_nets {NET1 NET2}
  ```

Use the `delete_net_group` command to delete a net group or all net groups and use the `delete_nets_from_net_group` command to remove a net from a net group.

Note: When you delete a net group, any bus guide associated with the net group also gets deleted.

Pin Guides

You can create a pin guide to constrain a bus, net, pin, net group, or pin group to be placed in specific areas. A pin guide is used for specifying a physical guided area where pins belonging to the pin guide will be placed. A pin guide can support multiple constraint pin layers. In addition, any bus, net, pin, net group, or pin group can be assigned to multiple pin guides. You can create a pin guide using the *Create Pin Guide* widget from the GUI or through the `createPinGuide` command.

Note: While creating a pin guide, you cannot optimize the order of pin members or specify minimum spacing. If you want to control the pin order and the pin spacing of the members of a pin guide, first create a net group or a pin group and associate this net group or pin group with a pin guide.

A physical location constraint can be specified either as a rectangular area or as an edge constraint. If you specify a physical location constraint as an edge constraint, you will also need to specify the partition/black box cell name.

Examples

Here are a few examples of using the `createPinGuide` command to create a pin guide.

- The following command creates a pin guide for a net group nGroup1. The pin order within this net group will be optimized. The pin members of this pin guide can be placed on the top edge of the cell ALU. Pins will be placed on Metal2 or Metal4 layers:

  ```
  createNetGroup nGroup1 -net {NET1 NET2} -optimizeOrder
  createPinGuide -netGroup nGroup1 -edge 1 -cell ALU -layer {Metal2 Metal4}
  ```

- The following command creates a pin guide for a pin group pGroup1 of cell/module ALU. Pins of this pin guide will have a minimum spacing of 2 tracks:

  ```
  createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -spacing 2
  createPinGuide -area 678.52 371.25 778.53 787.33 -pinGroup pGroup1 -cell ALU
  ```

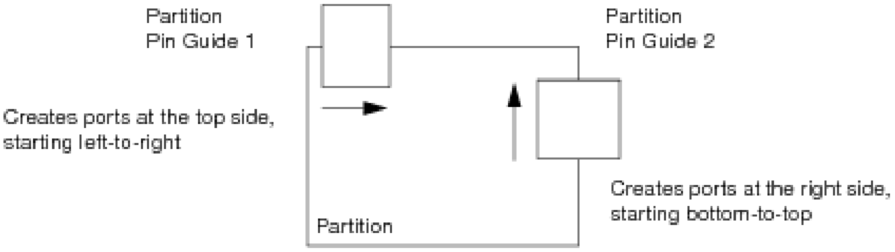
  The pins will be assigned on the preferred layers.

- The following command creates a pin group pGroup2. This pin group can be placed on the top edge or the right edge of the cell TDSP. For top edge, pins can be assigned on the Metal4 or Metal6 layers. For right edge, pins can be assigned on the Metal5 layer.

  ```
  createPinGroup pGroup2 -cell TDSP -pin p_addr* -optimizeOrder
  createPinGuide -edge 1 -pinGroup pGroup2 -cell TDSP -layer {Metal4 Metal6}
  createPinGuide -edge 2 -pinGroup pGroup2 -cell TDSP -layer Metal5
  ```

You can also use the GUI to create a partition pin guide. After you have determined a pin guide location in the design display area, you can create a partition port for a net or bus name and add a partition pin guide. To add a partition pin guide through the GUI, complete the following steps:

- 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.
- 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:


Partition
Pin Guide 1
Partition
Pin Guide 2
Creates ports at the top side, starting left-to-right
Creates ports at the right side, starting bottom-to-top
Partition

  The default pin spacing is 2, which places one pin for every two metal tracks.You can change the pin spacing with the *Minimum Pin Pitch* field in the Specify Partition form, or by changing spacing of the associated pin group or net group.
  You can use the Move/Resize/Reshape tool to modify the pin guide location.

  Note: For a partition that has a rectangular cut, the partition pin guide must be placed on the edge of the cut. You can also use a pin guide to assign pins, net group, or a pin group to a specific side without specifying a pin guide area by using the `createPinGuide` command.

- 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.

Handling Overlapping Pin Guides

Innovus supports overlapping pin guides on the same layer of a partition, however it may not create a good pin QoR due to pinGroup/netGroup ordering issue. To resolve the ordering issue, you may need to perform pin assignment multiple times:

- Assign only 100 pins in middle of the side using selected pin assignment (`-pin_file` *`filename`*).
- Assign the remaining unplaced pins using the `assignPtnPin -unplaceOnly` command.

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.

Examples

- The following commands set the pin width and depth of layer Metal2 for partition cell ALU to 0.4 and 0.6 respectively.

  ```
  setPinConstraint -cell ALU -layer Metal2 -width 0.4
  ```

  ```
  setPinConstraint -cell ALU -layer Metal2 -depth 0.6
  ```

- The following commands set the pin width of pin pGroup1 to 0.3 and pin depth of pin pGroup1 to default.

  ```
  setPinConstraint -cell ALU -pin pGroup1 -width 0.3
  ```

  ```
  setPinConstraint -cell ALU -pin pGroup1 -depth default
  ```

  With this example, all the pins of pGroup1 will have the width 0.3 and the default depth.

Pin Spacing

You can set minimum pin spacing in terms of track number using the *Specify Partition* form (*Partition - Specify Partition*). The default pin spacing is 2, which places a pin for every two metal tracks. You can modify the pin spacing in the following ways:

- Global pin spacing at design level
  Use the `-global` and `-spacing` parameters with the `setPinConstraint` command to set global pin spacing. This spacing value will be applied to all partition/black box pins of the design.
- Partition/black box level
  Use the `definePartition` command with `-minPitchTop`, `-minPitchBottom`, `-minPitchLeft`, and `-minPitchRight` parameters to specify minimum pin spacing for a partition. 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` command 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` can take the following values:

- N, S, W, E (supports both upper and lower case)
- T, B, L, R (supports both upper and lower case)
- dbcN, dbcS, dbcE, dbcW
- The following commands set the minimum pin spacing for top and bottom edge of partition cell ALU to 1 track.

  ```
  setPinConstraint -cell ALU -edge T -spacing 1
  setPinConstraint -cell ALU -edge B -spacing 1
  ```

- The following command sets minimum pin spacing for all edges of partition cell TDSP to 3 tracks

  ```
  setPinConstraint -cell TDSP -all -spacing 3
  ```

  Note: Use the `unsetPinConstraint` command 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 constraints can be specified at more than one level, pin assignment honors the spacing constraints in the following order (with appropriate command):

- Pin spacing (`setPinConstraint -cell A -pin * -spacing 3`)
- Net group or pin group spacing (`createPinGroup GRP -cell A -pin * -spacing 4`)
- Partition/black box spacing on a particular edge (`setPinConstraint -cell A -area {2 3 4 5} -spacing 5`)
- Partition/black box spacing (`setPinConstraint -cell A -side {T B} -spacing 6`)
- Global spacing (`setPinConstraint -global -spacing 8`)

Pin Layers

Specify pin layers that will be used for placing pins on a specific partition side using the *Specify Partition* form (*Partition - Specify Partition*). Alternatively, you can use the `setPinConstraint -layer -edge` command.

Note: Using the `setAttribute -top_preferred_routing_layer` and `-bottom_preferred_routing_layer` parameters, the preferred routing layer attributes are attached to nets. These attributes are honored throughout the routing process. The pin assignment commands honor these preferred routing layer attributes as soft constraint during pin assignment. This provides a better co-relation with the NanoRoute Router as it considers these as soft constraints. If a pin cannot be placed on these layers, the pin will not be unassigned. Pin assignment commands try to honor these attributes however in case they cannot be honored, the pin is assigned to any allowed layer.
If only one of the attributes is given, the other is assumed from the lowest or the highest value of the allowed layer list. If `-top_preferred_routing_layer` is applied, then `-bottom_preferred_routing_layer` is assumed to be the lowest layer of allowed layer list. Alternatively, if `-bottom_preferred_routing_layer` is applied, then `-top_preferred_routing_layer` is assumed to be the highest layer of allowed layer list.

You can specify layer constraints at partition level, pin guide level, or pin level.

- Partition level
  Layer constraint per edge can be specified at partition level using either:
  - The *Specify Partition* form (*Partition - Specify Partition*), or
  - The `definePartition` command with `-pinLayerTop`, `-pinLayerBottom`, `-pinLayerLeft`, and `-pinLayerRight` parameters. These layer constraints will be applied to all pins on a particular edge of the specified partition.
  - The `setPinConstraint` command with the `-layer` and `-edge` options. This command sets the pin layers for a specified edge or for all edges.
- Pin guide level
  Use the `-layer` parameter of the `createPinGuidecreate_pin_guide` 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.

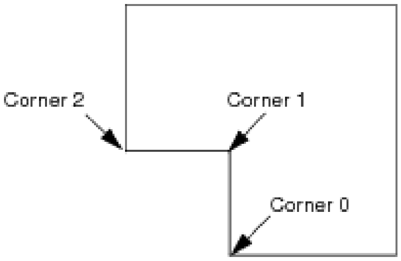
- If a layer constraint is applied to a pin that also belongs to a pin guide, the pin guide layer constraint will have higher precedence.
- If a layer constraint is being applied to a pin that already belongs to a pin group a or net group, the layer constraint will not be applied. To apply layer constraint for this pin, first remove this pin from the pin group or net group, and then apply the pin layer constraint.

Pin-to-corner Distance

To keep pins away from partition/black box corners, you can set the pin-to-corner distance constraint.

- Use the `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 `setPinConstraint -global -corner_to_pin_distance` command 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: Early Global Route does not honor the partition pin blockage.

To create the partition pin blockage with the *Create Pin Blockage* widget, complete the following steps:

- Click the *Create Pin Blockage* widget from the *Toolbox*. Alternatively, select *Floorplan - Edit Floorplan - Create Pin Blockage*.
- Left-click and drag over a partition fence overlap.
- Right-click the partition pin blockage and select *Attribute Editor* and then specify the metal layers to block.

The following command creates a pin blockage for the entire left edge of cell TDSP on layer Metal5.

```
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.

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 `create_pin_blockage` command.

Note: Early Global Route does not honor the partition pin blockage.

To create the partition pin blockage with the *Create Pin Blockage* widget, complete the following steps:

- Click the *Create Pin Blockage* widget from the *Toolbox*. Alternatively, select *Floorplan - Edit Floorplan - Create Pin Blockage*.
- Left-click and drag over a partition fence overlap.

- Right-click the partition pin blockage and select *Edit Pin Blockage* and then specify the metal layers to block.

The following command creates a pin blockage for the entire left edge of cell TDSP on layer Metal5.

```
create_pin_blockage -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 `delete_pin_blockages` command to delete a pin blockage or all pin blockages (`delete_pin_blockages -all`).

Note: You can create pin blockage on master or clone instances. Pin blockages are transformed on both master and clones enabling you to visually see the pin blockage on both master and clone partitions. If you delete or modify any of such pin blockages, all its variants with same corresponding box on master/clones will be deleted.

Performing Pin Pre-Assignment

You can pre-assign a pin before pin assignment using the *Pin Editor* or the `editPin` command. These pre-assigned pins will have fixed placement status so pin optimizers will honor them. For more details, see the *Pin Editor* section in the Edit Menu chapter of the *Menu Reference*.

Setting Constraints on a Specific Pin

Use the `setPinConstraint` command to specify the following constraints for a particular pin:

- Physical location
  A pin can be constrained by specifying its coordinate (x, y) location and its preferred routing layer. If specified location is not on valid track, the pin will be snapped to the closest location. To keep the pin on non-preferred routing layer or to not snap the pin, use the `editPin` command instead. Besides an actual physical location, a pin can also be constrained to a particular edge.
- Layer
- Spacing

The following command specifies that the pin reset of partition cell mult_32 should be placed on top edge with either Metal5 or Metal7 routing layer.

```
setPinConstraint -cell mult_32 -pin reset -edge 1 -layer {Metal5 Metal7}
```

For setting pin size constraint for a specific pin use the `setPinConstraint` command with the `-pin`

`-width -depth` parameters. The following salient points apply to setting the pin constraints for a specific pin:

- If constraints are applied to a pin that also belongs to a pin guide, the pin guide constraint will have higher precedence.
- If a location and/or layer constraint is being applied to a pin that already belongs to a pin group or a net group, the constraint will not be applied. To apply location and/or layer constraint for this pin, first remove this pin from the pin group or net group, and then apply the pin constraint(s).
- If a pin with layer constraints defined is added to a net group or pin group, the pin cannot be added to a pin group or a net group with the `createPinGroup`, `createNetGroup`, `addPinToPinGroup`, or `addNetToNetGroup` commands because the pin has already been constrained. To add this pin to a pin group or net group remove the constraints first (using the `unPinConstraint` command).
- If the following constraints cannot be met during pin assignment, the Innovus software will issue a warning message and the constrained pins will be placed at the lower-left corner of the partition/black box with unplaced placement status:
  - Pin constraint
  - Pin group constraint
  - Net group constraint

Use the `unPinConstraint` 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` 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 follow pins. Power and ground pins will be created during the partition step.

Note: Even though unconnected (floating) partition pins can be assigned randomly on any open space on any edge, the `assignPtnPin` command tries not to club all the floating pins on one edge.

However, in case of floating buses, it assigns all the unconnected bus pins of a bus close to the connected pins of the same bus. In case all the bus pins are unconnected, then they are assigned close to the last assigned bus. The benefit of this feature is that if the pins need to be connected later on then there is no need to re-open the block.

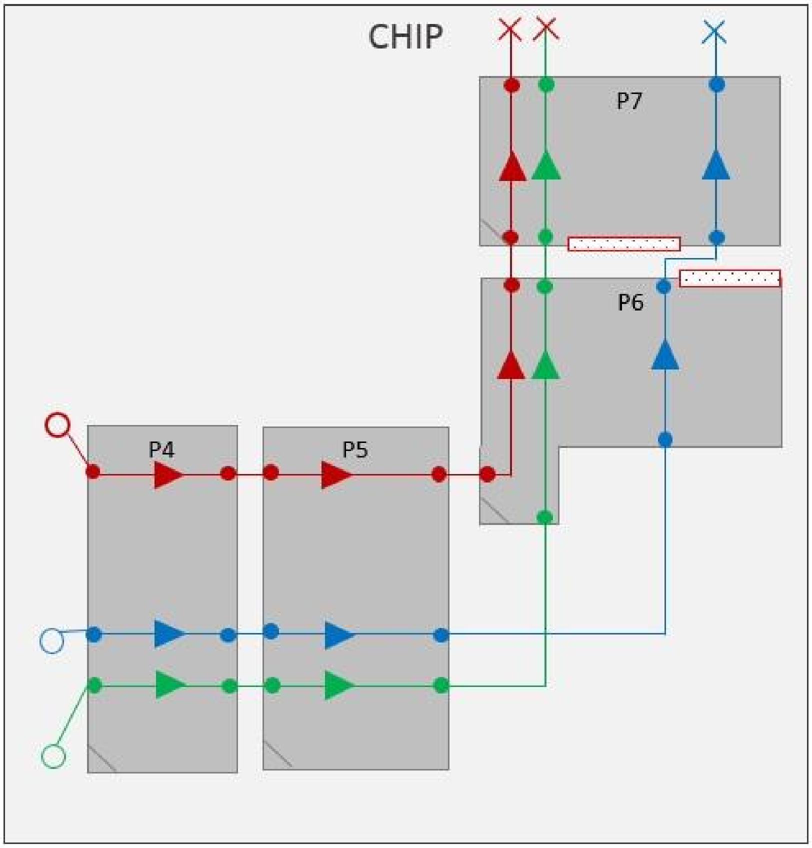
Note: The `assignPtnPin` command internally calls the `reportUnalignedNets` command to generate the summary report at the command line.

Aligning Partition Pins Across Feedthrough Buffers

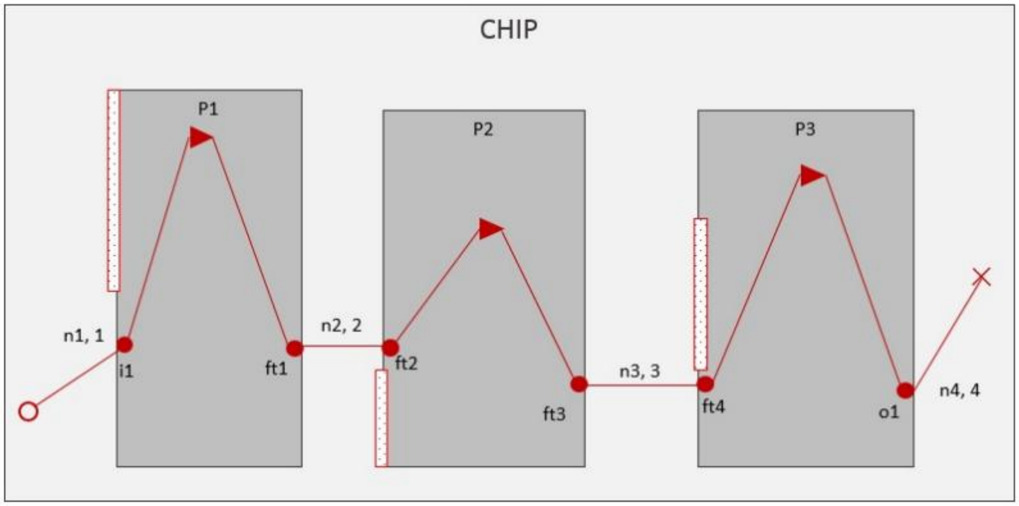
The `assignPtnPin` command, by default, places pins such that they are aligned on different partitions across nets through buffer chains. The re-placement of buffers aligned to a new pin location gives better congestion and net length. The following consideration as made while aligning feedthrough pins:

- Pins across nets on different partitions are aligned.
- The pin for 1-pin nets (partitions pins connecting with the top-level object) are assigned near the source/sink. Here, the top-level object can be a pin of the pre-placed standard cell or macro/block or IO pins.
- Maximum alignment is attempted on buffer chain nets (for Z, L and other shape of partition feedthroughs)
- Maximum alignment is attempted on buffer chain nets if encountered with blocked slots.
- The order of different buffer chain nets is not maintained.
- Nets can choose different edges or sides

The following is an example of default feedthrough pin assignment. Here, despite blockages, there is maximum alignment in the red buffer and the blue buffer chain nets.



In the following example, in spite of pin blockages, the `assignPtnPin` command has attempted maximum alignment on the buffer chain nets.



After pin placement, as shown above, the blocks place these buffers again for better congestion and reduced net length results.

Placement-based Pin Assignment

Pin assignment is based on connectivity. 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. Default Early Global 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 `earlyGlobalRoute` to route to black box pins
- Removes temporary blockages

For channel-less designs use default early global route:

```
earlyGlobalRoute
```

For channel-based designs that have thick channels, use:

```
setRouteMode -earlyGlobalRoutePartitionHonorFence list_of_ptn_cell_names
earlyGlobalRoute
```

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 (`setRouteMode -earlyGlobalRoutePartitionHonorFence`) 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:

  ```
  setRouteMode -earlyGlobalRoutePartitionHonorFence list_of_ptn_cell_names
  earlyGlobalRoute
  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 the `earlyGlobalRoute` command 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.

- `-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
- 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

Checking the Pin Legality

The `checkPinAssignment` command checks the generated partition and blocks-box pins and I/O pins and generates a summary report of the violations found. Use the `checkPinAssignment` command to make sure that pins are legalized (for example, the pins snap to routing grid, are on reserved routing layers, honor user-specified constraints, do not cause any DRC violations, and so on).

You can check:

- All partition/black box pins
  Example: The following command checks all partition/black box pins in the current design and saves the result into the output file `pinLegality.rpt`.

  ```
  checkPinAssignment -outFile pinLegality.rpt
  ```

- All pins of a specific partition
  Example: The following command checks all pins of the partition TDSP_CORE

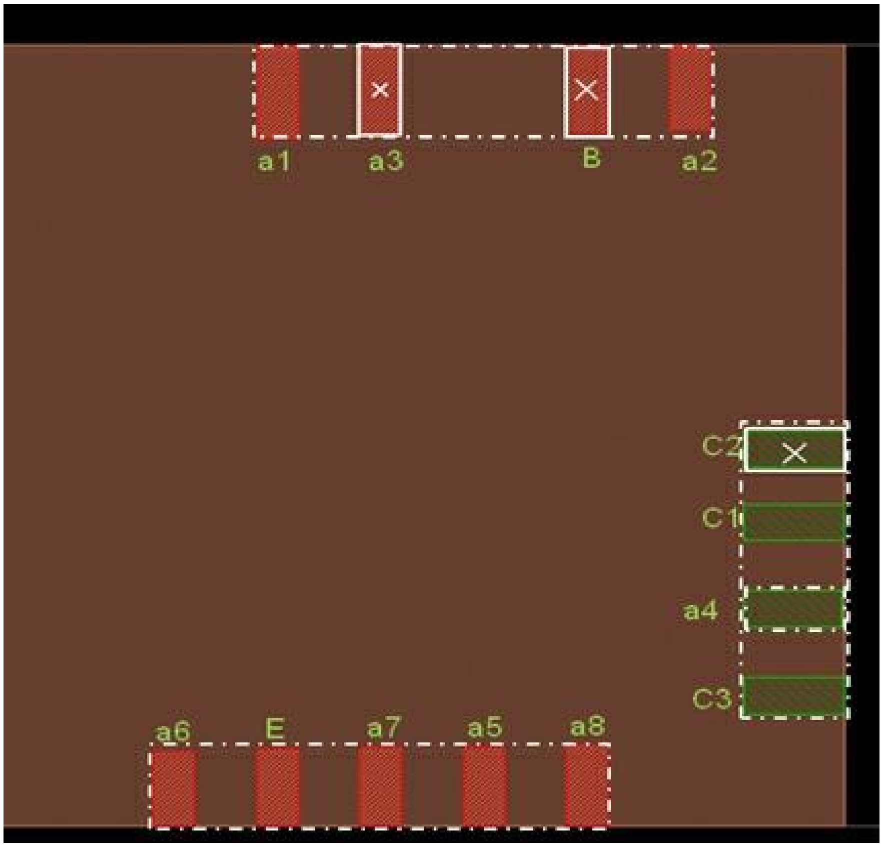
  ```
  checkPinAssignment -ptn TDSP_CORE -pin *
  ```

- Specific partition pins
  Example: The following command checks all bus pins `p_addrs` and `rom_data` of the partition `TDSP_CORE`

  ```
  checkPinAssignment -ptn TDSP_CORE -pin {p_addrs* rom_data*}
  ```

The `checkPinAssignment` command highlights violations on individual pins by marker that covers the full pin shape. It checks violations for `pin_abutment`, `pin_depth`, `pin_layer`, `pin_min_area`, `pin_on_fence`, `pin_on_track`, `pin_spacing`, `pin_width`, and `logical_pin`. If there is a marker on the master pin, marker on clone pins is not highlighted. The `checkPinAssignment` command will check for individual violations, which are different than master, on clone pins as well. For instance, spacing to an object near clone. The markers that are specific to clones will be reported by `checkPinAssignment`.

- For violations of guide, such as `bus_guide` and `pin_guide`, if there is violation in individual pins, markers will be on individual pins only.
- For violations of group, `pin_group` and `net_group`, the first pin offending the group order will be highlighted as violation and all the pins from first pin to the last pin of the group will be highlighted as aggressor. If the pin group is traversing multiple edges then multiple aggressor type violations will be generated. The graphic below shows an example of violations marked for pins.



In this example, there is a pin group a1, a2, a3, a4, a5, a6, a7, a8. After the placement, the order of the pins is changed to a1, a3, a2, a4, a8, a5, a7, a6. In this case, the first pin violating the order is a3, and therefore will be marked by a violation marker. Also, to reduce the number of victims, entire group from the first pin to the last pin is highlighted. Here, the pin group is on three edges.

- On edge 1, first pin of the group is a1 and the last pin is a2.
- On edge 2, there is only one pin a4.
- On edge 3, the first pin is a8 and the last pin is a6.

Therefore, three victim markers will be drawn from first to last pin on each edge. These victim markers may cover other pins which fall between the first and the last pin, such as pin B and pin E on edge1 and edge 3, respectively.

Note: The `checkPinAssignment` command enables you to query results of the command report and find more details of the violations. Each time the `checkPinAssignment` command is run, it automatically refreshes the properties of pins. It clears all properties on all pins and populates fresh properties on each pin based on the latest the `checkPinAssignment` command results. These properties are persistent in save/restore DB.

You can query results of the pin violations report generated by the `checkPinAssignment` command to automate the flow. For example:

- To query for all partition pins with depth violation, you can use the following query:

  ```
  get_db [get_db partitions .master -if {.obj_type == hinst}] .local_hpins -if
  {.vio_depth == 1}
  ```

- To query for all pins (partition pins, bbox pins and IO pins) with depth violation, you can use the following query:

  ```
  puts [concat \
  [get_db ports -if {.vio_depth == 1}] \
  [get_db [get_db partitions .master -if {.obj_type == inst}] .pins -if {.vio_depth
  == 1}] \
  [get_db [get_db partitions .master -if {.obj_type == hinst}] .local_hpins -if
  {.vio_depth == 1}] \
  ]
  ```

- To query for all pins in a master partition having pin depth violations, you can use the following query:

  ```
  get_db [get_db partitions .master -if {.obj_type == hinst && .name == “S”}]
  .local_hpins -if {.vio_depth == 1}
  ```

Refining Pin Assignment and Fixing Pin Violations

After assigning partition or blackbox pins, you can further refine the current pin assignment and fix any pin violations using one or more of the following methods:

- Adjusting Pins
- Aligning Partition Pins
- Running Incremental Pin Assignment
- Adjusting Floorplan or Floorplanning the Design Again
- Performing Pin Assignment Again

These steps are explained in the following sections.

Adjusting Pins

You can Adjust pins using the *Pin Editor* or the `editPin` text command. You can also use direct pin manipulation to manually move selected pins to different locations.

Aligning Partition Pins

You can align partition pins with other block pins (using the Pin Editor or the `pinAlignment` text command). The `pinAlignment` command can be used to align partition/black box pins with or without specified reference object(s). Reference objects can be hard macros, blackboxes, I/O pads, standard cells, and ptn pin.

You can use the `pinAlignment` command in different ways to align pins:

- Align specific pins with the specified referenced object

  ```
  pinAlignment -refObj refInstName -ptnInst instName -pinNames pinList
  ```

- Align all pins of specified partition/blackbox instance that connect with the specified reference object

  ```
  pinAlignment -refObj refInstName -ptnInst instName
  ```

- Align all pins of every partition/blackbox that connects with the specified reference object

  ```
  pinAlignment -refObj refInstName
  ```

- Align specific pins of specified partition/blackbox instance

  ```
  pinAlignment -ptnInst instName -pinNames pinList
  ```

- Align all pins of specified partition/blackbox

  `pinAlignment -ptnInst *instName*`

- Align all possible partition/blackbox pins

  `pinAlignment`

If the referenced object is not specified, the `pinAlignment` command will automatically derive referenced object based on connectivity information. If the aligned pin has multiple connections, the referenced object will be derived based on the following priority:

- Hard macro pin
- I/O pad pin or I/O pin
- Partition pin
- Standard cell pin

By default an aligned pin will:

- Be snapped to routing grid. Use `-noSnap` option if you want that pins should not be snapped.
- Have the same layer routing with the referenced pin. Use the `-keepLayer` option to keep existing aligned pin layer. Use the -newLayer option to assign new layer for aligned pin.
- Not be legalized. Use the `-legalizePin` option to legalize aligned pin(s).
- Have a fixed pin status. Use the `-markPlaced` option to assign placed status to aligned pin(s).

Running Incremental Pin Assignment

Based on the current pin assignment result, re-run pin assignment. You can specify pin constraints to further guide new pin placement. If you want to reoptimize only a few specific pins, use the `-ptn` and the `-pin` options of the `assignPtnPin` command to specify the list of pins that will be reassigned.

Example: The following command reoptimizes address bus bit pins, rom_data bus bit pins of partition ALU, and `reset` pin of partition ARB:

```
assignPtnPin -ptn ALU -pin {address* rom_data[*]} -ptn ARB -pin reset
```

By default, `-ptn` and `-pin` options will not reassign specified pins if they have fixed status. Use the `-moveFixedPin` option with the `-ptn` and `-pin` options to force specified fixed pins to be reassigned. However if you want to keep only a few existing pins and re-optimize the rest of the pins, instead of specifying many pins, you can mark these existing pins to fixed placement status using the `setPtnPinStatus` command and then re-run pin assignment:

```
setPtnPinStatus -cell * -pin * -status fixed
assignPtnPin
```

Adjusting Floorplan or Floorplanning the Design Again

Sometimes a sub-optimal floorplan can also lead to a bad pin placement result. If this is the case, re-adjust the floorplan and run pin assignment again.

Performing Pin Assignment Again

Perform pin assignment again. If the partitions or blackboxes have been committed, use the `flattenPartition` command to unassign the pins. If the partitions or blackboxes are not yet committed, use the `setPtnPinStatus` command to unplace the pins.

ECO Pin Assignment

The Innovus software provides incremental or ECO pin assignment capability. This capability can be used in the ECO flow where partition/black box ports in the original netlist get modified (added/deleted). In this flow, you can preserve most of the existing partition/black box pin locations and let the software assign the newly added pins.

General Flow

1. Import design.
2. Floorplan design (specify partition/black box information in this step).
3. Run placement.
4. Route design.
5. Save full chip floorplan and/or save design for later use.
6. Assign pins (`assignPtnPin`)
7. Save partition/black box pin information into a partition file `(savePtnPin).` For iHDB flow, use `savePtnPin -module_model_tag.`
8. Route design to connect to new partition/black box pins.
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:

1. Import design with new modified netlist.

2. Load full-chip floorplan that saved in the previous step 5.

3. Place and route the design. Placement and routing information that are saved in the step 5 can be restored if still applicable.

4. 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.

5. 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 -file ALU.ptn
  ```

- Block-level DEF file
  Example: Load pin locations of partition ALU from ALU DEF file

  ```
  loadPtnPin -ptnName ALU -def ALU.def
  ```

Assigning I/O Pins

For a top-down hierarchical flow, I/O pins of a block-level design will normally be assigned during the full-chip pin assignment. This pin placement information is saved in a block-level floorplan partition file (`*floorplan*.fp.ptn`) or a DEF file that is generated by the `savePartition` command. You must explicitly run I/O pin assignment with the `assignIoPins` command.

Note: In the iHDB flow, the DEF file is generated by using the `savePartition -module_model_tag` command.

This section covers the following topics:

- Setting Pin Constraints
- Performing Initial Pin Assignment
- Validating Pin Placement

Setting Pin Constraints

The Innovus software provides a number of pin constraint commands to control or guide I/O pin assignment. The same set of pin constraint commands that are used for setting constraints for partition/blackbox pins can also 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 placing the design. After importing a design and floorplanning it, you should place the design.

Note: Set the `setPlaceMode -place_global_place_io_pins` command to true if you want to enable I/O pin assignment during the placement step.

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.

  For example, the following command re-assigns all p_address bus pins, int, and bio I/O pins of the design tdsp_core. It optimizes 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.

  For example, the following command preserves the `port_pad_data_in` and `port_pad_data_out` buses and clock pins, and re-optimizes the rest.

  ```
  setPtnPinStatus -cell tdsp_core -pin port_pad_data* -status fixed
  setPtnPinStatus -cell tdsp_core -pin clk -status fixed
  assignIoPins
  ```

Validating Pin Placement

After assigning I/O pins, it is recommended that you check for I/O legalization. Use the `checkPinAssignment` command to make sure that pins are legalized (such as they snap to routing grid, are on reserved routing layers, honor user-specified constraints, not cause any DRC violations, and so on).

You can check:

- All I/O pins
  Example: Verify all I/O pins of the current design and output the result into the output file `pinLegality.rpt.`

  ```
  checkPinAssignment -outFile pinLegality.rpt
  ```

- Specific I/O pins
  Example: Verify all bus pins `BG_scan_in`, `BG_scan_out`, and the `write` pin of the design

  ```
  checkPinAssignment -pin {BG_scan* write}
  ```

If any pin violation is detected, you can:

- Manually adjust pins by direct pin manipulation or using pin editor.
- Run the `legalizePin` command to automatically legalize pins. You can legalize all I/O pins or specific I/O pins of the design. Fixed pins will not be adjusted unless the `-moveFixedPin` option is specified.

Examples

- The following command legalizes 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.

  ```
  legalizePin
  ```

- The follwoing command legalizes all I/O pins. Fixed pins will also be adjusted because the option `-moveFixedPin` has been specified.

  ```
  legalizePin -pin * -moveFixedPin
  ```

- The follwoing command legalizes the clock, reset, and all rom_data bus bit pins of the design. Pins with fixed status will not be moved.

  ```
  legalizePin -pin {clock reset rom_data*}
  ```

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.

For channel-based designs, use:

```
setRouteMode –earlyGlobalRoutePartitionHonorFence list_of_ptn_cell_names
earlyGlobalRoute
```

Note: The iHDB flow 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.

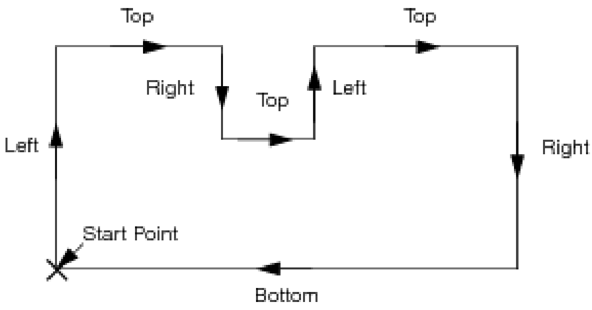
Salient Points About Congestion-aware Pin Assignment

The following points apply to the behavior and usage of the congestion-aware pin assignment feature:

- 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 earlyGlobalRoute.
- 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.

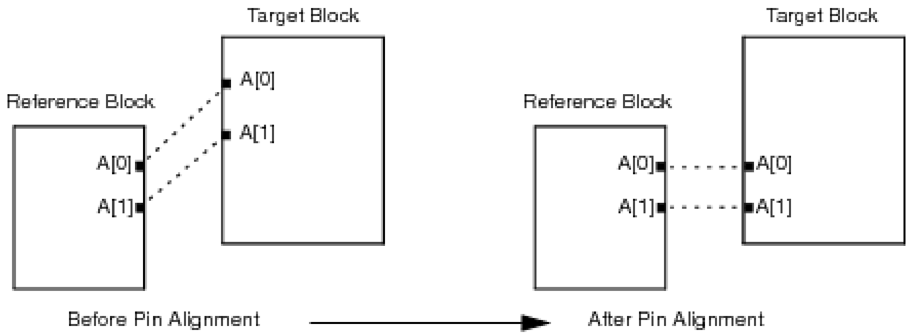
Swapping Partition Pins

- Select two pins of the same partition.
- With the cursor over one of the selected pins, click and hold the middle mouse button to bring up the context pop-up menu.
- 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 -ptnInst B -refObj blockA -pinNames {A0 A1}
```



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. 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 groups. The check pin assignment, pin legalization and pin refinement features also support bus guides.

Info: The bus guide feature is intended to guide partition pins and blackbox pins and *not* I/O pins. The I/O pin assignment feature (`assignIoPins` command) does not, therefore, take bus guides into account.

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.

Info: Before inserting feedthroughs, you should determine what stage the design is in, such as prototyping, intermediate, tapeout, and set the appropriate global options by running commands such as `setPlaceMode`, `setExtractRCMode` `etc`. For example, when inserting feedthroughs during prototyping, you could set modes with the following commands:

```
setPlaceMode -place_design_floorplan_mode true setExtractRCMode -engine preRoute
```

Inserting feedthrough buffers allows a netlist change, whereas inserting routing feedthroughs does not. The differences between how these two commands affect the design are as follows:

- **Feedthrough Buffers for Partitions**
  The `insertPtnFeedthrough` text command inserts feedthrough buffers into the partitions to change the partition netlists, and avoids routing nets over partition areas. Alternatively, you can use the *Browse/Plan Partition Feedthroughs* form. Inserting feedthrough buffers for partitions affect the design in the following areas:
  - Changes both the top-level and partition-level netlists.

- After inserting buffers, it automatically calls the `ecoPlace` command 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 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 early global 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 `-selectNet` option, the Innovus software does not necessarily insert a feedthrough.
- Feedthrough insertion is driven by connectivity when Early Global 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.
- 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.

- For inserting feedthrough buffers into nested partitions, the `insertPtnFeedthrough` command, first, automatically deletes the parent partitions and keeps only the bottom most partition. It then does a single level feedthrough insertion and then automatically restores the parent partitions and brings the nested partition definition back as it is. In nested designs, the `insertPtnFeedthrough` command only puts buffers and ports in the bottom most level of partitions.

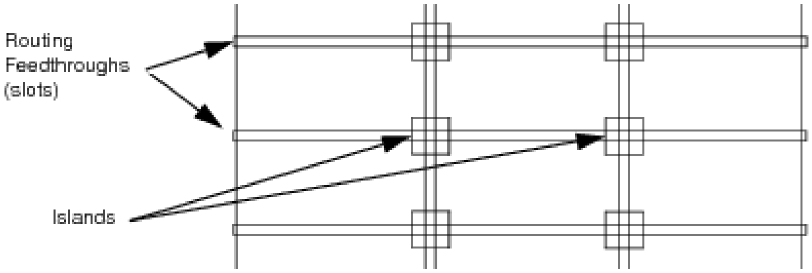
- **Routing Feedthroughs**
  The `createPtnFeedthrough` text command inserts routing feedthroughs into the partitions without changing the design netlist. Alternatively, you can use the *Create Physical Feedthrough* form. Inserting routing feedthroughs affects the design in the following areas:
  - Manages only the physical aspect of a partition, not the logical aspect.
  - No new ports are added to a partition and no buffers are added to the partition netlist.
  - For channel feedthroughs, this creates channels for over-the-block routing on specified layers at the top-level design. These channels are pushed down as routing blockages on the correct routing layers at the partition level during Partitioning.
  - For placement island feedthroughs, the Innovus software reserves these areas for inserting buffers at the top-level design after running the `addRepeaterByRule` 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 Routing Feedthroughs

Routing feedthroughs and hole punch buffers reserve a portion of the partition area for top-level use. Because the partition's netlist does not change, no new ports are created for the partition. Buffers are inserted in top-level netlist but occupy space within the partition's fence. Partition feedthroughs are used to indicate the top-level partition's concession within the partition fence. Partition feedthroughs should be defined before running the Partition program, which automatically generates appropriate placement and routing blockages within the partition and in top-level view to reflect the real estate ownership scheme. For example, a routing feedthrough with *Metal6* will generate a *Metal6* routing blockage for the partition, and an opening in the *Metal6* blockage in the top level.

Note: The partition feedthrough discussed in this section is a floorplan object. It affects a partition only physically (not logically) and does not affect partition feedthrough buffer cells.

A routing feedthrough (slot) within the partition's fence is used by the top-level partition's routing, and an *island* within the partition's fence can be used by the top-level partition's placement, as shown in the following figure:

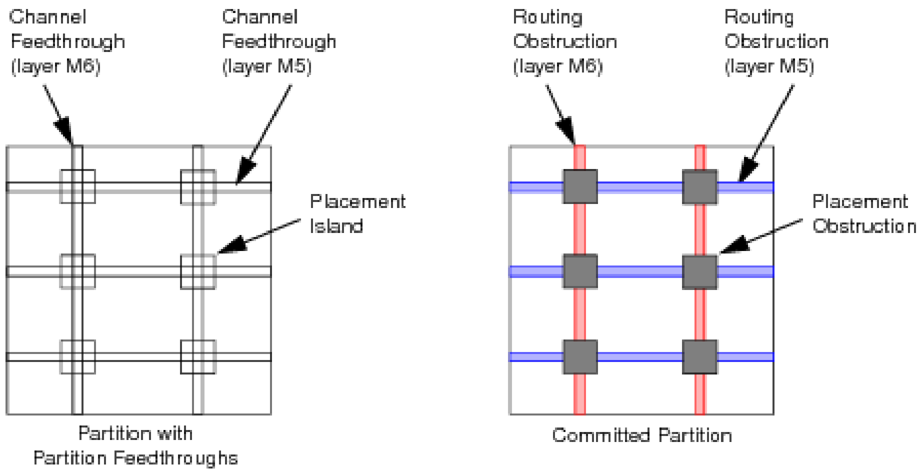


Note: Routing feedthroughs can be used without placement islands.

To create a channel-type feedthrough, use the *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.

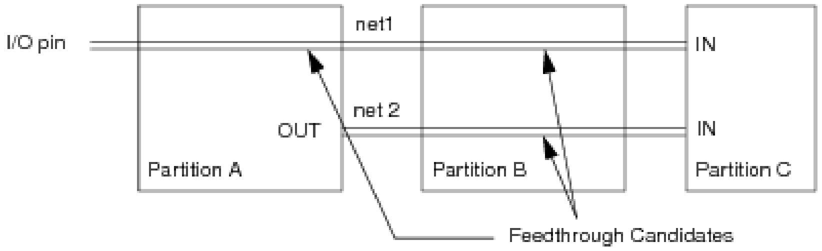
To insert routing feedthroughs and hole punch buffers, complete the following steps:

- 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*.
- (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.
- Run Partition. This automatically creates routing blockages for the channel feedthroughs, and placement blockages for the placement island, as shown in the following figure:


Channel Feedthrough (layer M6)
Channel Feedthrough (layer M5)
Placement Island
Partition with Partition Feedthroughs
Routing Obstruction (layer M6)
Routing Obstruction (layer M5)
Placement Obstruction
Committed Partition

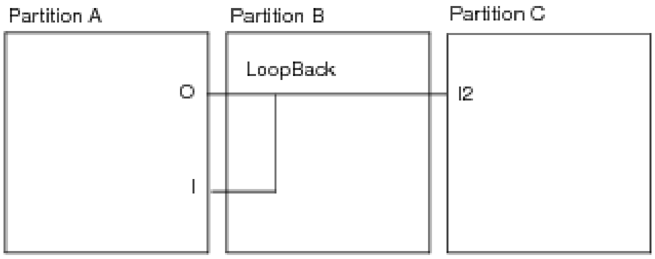
Inserting Feedthrough Buffers

Partition feedthrough insertion manages partitioned designs that have nets that need to be pushed down to become a component of each partition design. That is, each feedthrough buffer must be added to the partitioned design, which changes the partition's netlist. This approach is typically used in channelless designs and in designs with limited channel resources. A pure channelless design has no channel routing resource – connections among partitions are always done by means of module abutment and pin alignment. A mixed or partially channelless design has limited routing resource in the channels; therefore, abutment and pin alignment is only performed on selected nets. The following example shows how nets are selected for feedthrough buffers:

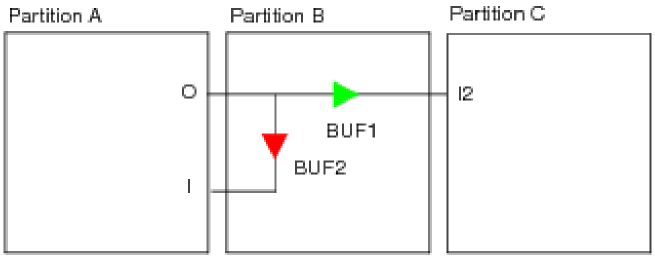


Inserting Feedback Buffers

You can insert a feedthrough buffer to a net that loops back to an original partition to avoid the net routing over a partition area using the `insertPtnFeedthrough` command or the *Browse/Plan Partition Feedthroughs* GUI form (*Partition - Feedthrough Ports*). The following example shows a situation where net `LoopBack` connects to output pin `O` and input pin `I` of Partition A, and input `I2` of Partition C.



By inserting a feedthrough buffer (BUF1) and a feedback buffer (BUF2) with the `insertPtnFeedthrough` command, LoopBack now connects to the input pins of BUF1 and BUF2, as shown in the following figure:



Limitations

- Each partition must be intact. A non-child instance cannot be preplaced in another partition. This would present a top-level net connection problem.
- Partition pin guides cannot be used during feedthrough insertion.
- The Unpartition program cannot remove the inserted buffers for the feedthrough nets.
- Does not handle blackboxes.
- It might not handle clock nets efficiently because the `insertPtnFeedthrough` 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.
- For channel based methodology, you should exclude high fanout nets and clock nets from feedthrough insertion for better results.
- The `insertPtnFeedthrough` command does not support master-clone designs with power domains.
- In nested designs, the `insertPtnFeedthrough` command only puts buffers and ports in the bottom most level of partitions.

Procedure

1. Design the top-level floorplan for the partition design.
2. Run Placement.
3. (Optional) Run Early Global Route.
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 `insertPtnFeedthrough -selectNet` command with the created net file. alternatively, use the *Browse/Plan Partition Feedthroughs* form.
6. Run Early Global 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.
9. Run the `savePartition` command. This saves the design and generates a top-level directory and partition directories.
   Note: For the iHDB flow, use the `savePartition -module_model_tag` command.

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. The topology file can be created using the following methods:

- By manually writing the topology structure in a file. This structure must be written using a specific syntax. You can manually create this file and subsequently edit it.
- By using the *Browse/Plan Partition Feedthroughs* GUI form. For more information see the Creating a Topology File Using the GUI section.

Topology File Structure Guidelines

The topology file consists of multiple sections and must follows the following guidelines:

- Each section contains a header which specifies whether the section specifies the topology for a net, netgroup or a bus.
- The section specifies the topology tree in terms of the edges between two nodes.
- The root node of the tree must be the driver of the net.
- The root node does not have any parent node.
- Each other node in the tree other than the root node must have a unique parent node.
- The tree must be complete. The path from the driver to each sink should be traceable in the tree.

Topology File Versions

There are different versions of the topology file:

- Version 2.0
- Version 1.0
- Version 0.5 (Used with `write_generalized_feedthru_path` command ONLY)

In Version 2.0 of the topology file:

- You can specify exactly how the reuse of ports takes place in master clone scenarios.
- You have more control over the feedthrough topology.

The main difference from version 1.0 is that the topology needs to be specified in terms of the port names of the hierarchical modules rather than the names of the hierarchical modules. The version 2.0 of the topology file will *not* obsolete the version 1.0.

Version 2.0 of the Topology File - Syntax and Examples

The syntax of the topology information in the file is as follows.

```
# Comment line

  version version_string;
    nametype netname
        fromtype-totype from_name to_name;
        fromtype-totype from_name to_name;
        ...
    end nametype
    nametype netname
        fromtype-totype from_name to_name;
        fromtype-totype from_name to_name;
        ...
    end nametype
        ...
```

The description of the syntax is as follows

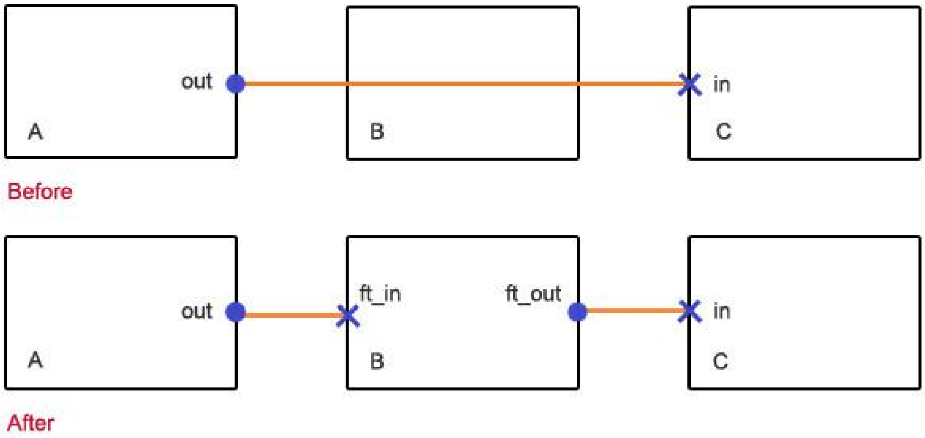
`nametype`	Can only be `net`. It cannot be `bus` or `netgroup`.
`netname`	The name of the net. Wildcards (* or ?) cannot be used for net names.
`fromtype`	Can have one of the following values: • `hterm`: The `from_name` is to be specified as hinstname/portname. If the port names specified don't exist already then they will be created. • `instterm`: ○ For top level pins the `from_name` is specified as instname/termname. ○ For multi-fanout nets originating from a hierarchical module which is a partition or a clone the topology must originate from the driver term. Instead of specifying the complete driver name such as instname/termname one can use the shorthand `driver_term`. Refer to examples 2,4,and 6 given below. • `io`: For chip level I/O pins. **Note**: The `hinst` keyword may not be used as a `fromtype` in version 2.0.
`totype`	Can have one of the following values: • `hterm`: The `to_ name` is to be specified as hinstname/portname. If the port names specified don't exist already then they will be created. • `instterm`: For top level instance pins. The `to_name` is specified as instname/termname. • `io`: For chip level I/O pins. **Note** : The `hinst` keyword may not be used as a `totype` in version 2.0.

`version`	The version is 2.0. The topology file is saved in this format when the `-repeatedSymmetricAbuttedFPlan` option is used with the `-saveTopoFile` *`filename`* option of the `insertPtnFeedthrough` command.

Examples of usage of topology file version 2.0

The following examples illustrate topology file usage:

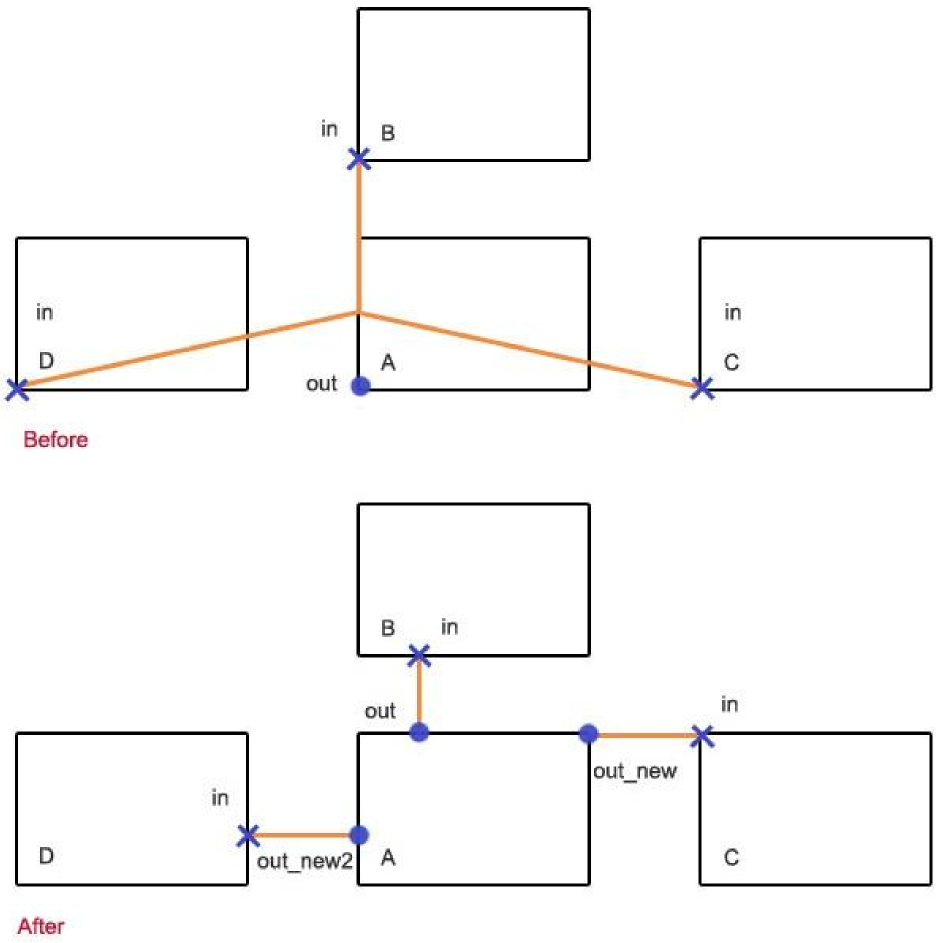
- **Example 1: Simple Case**
  The following image shows a two pin net before and after feedthrough insertion. It has an output port on partition A and an input port on partition C.



The following topology syntax is used:

```
version 2.0;
net net
hterm-hterm A/out B/ft_in;
hterm-hterm B/ft_in B/ft_out;
hterm-hterm B/ft_out C/in;
end net
```

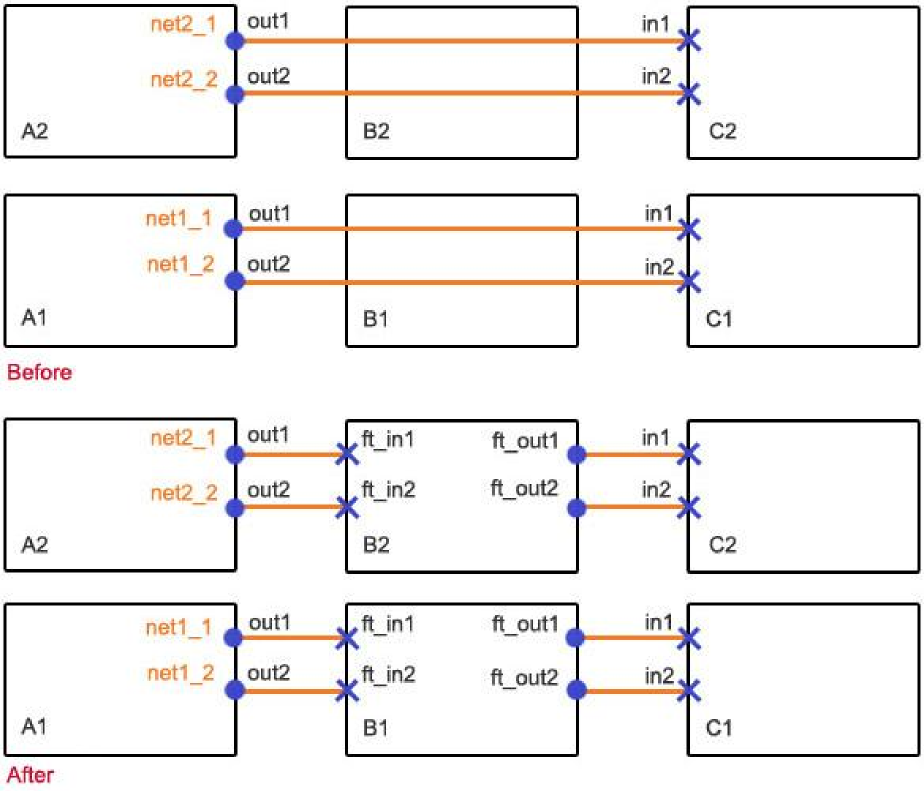
- **Example 2: Multi-fanout Case**
  The following is a before and after feedthrough insertion image of a multi-fanout net with three sink partitions. It has output port on partition A and input ports on partitions B, C, and D.



The following topology file is used

```
version 2.0;
net net
instterm-hterm driver_term A/out;
instterm-hterm driver_term A/out_new;
instterm-hterm driver_term A/out_new2;
hterm-hterm A/out B/in;
hterm-hterm A/out_new C/in;
hterm-hterm A/out_new2 D/in;
end net
```

- **Example 3: Reuse for simple case of Example 1**
  The following is a before and after feedthrough insertion image of four nets with two pins.

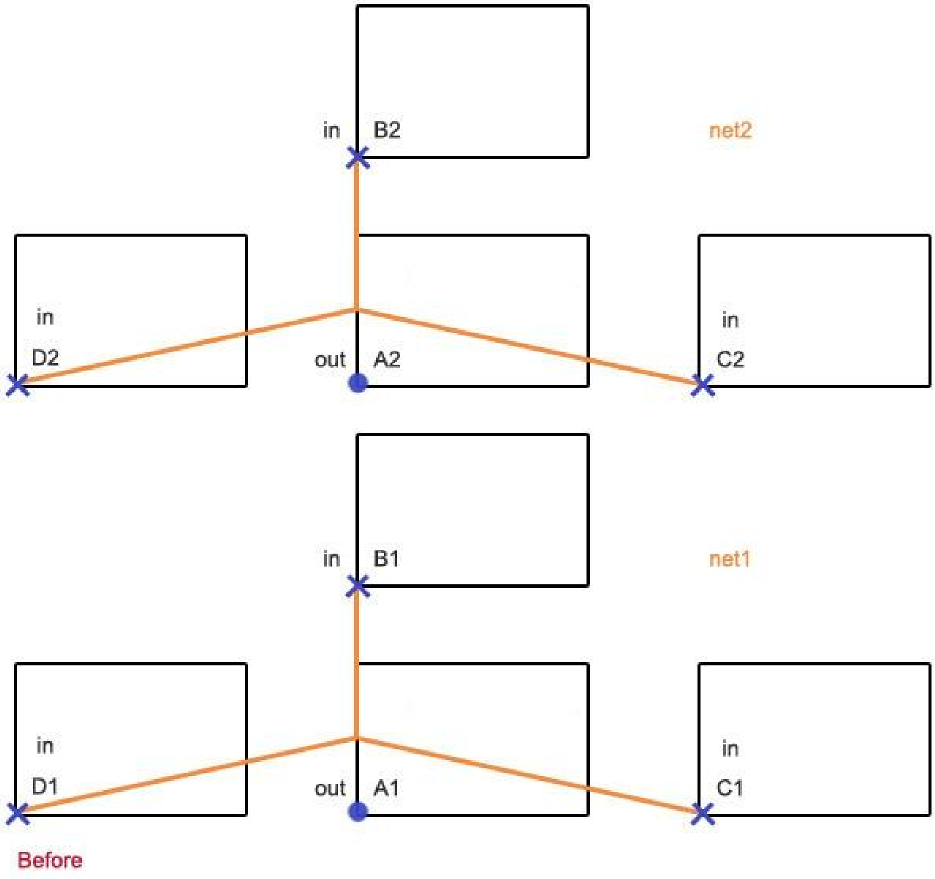


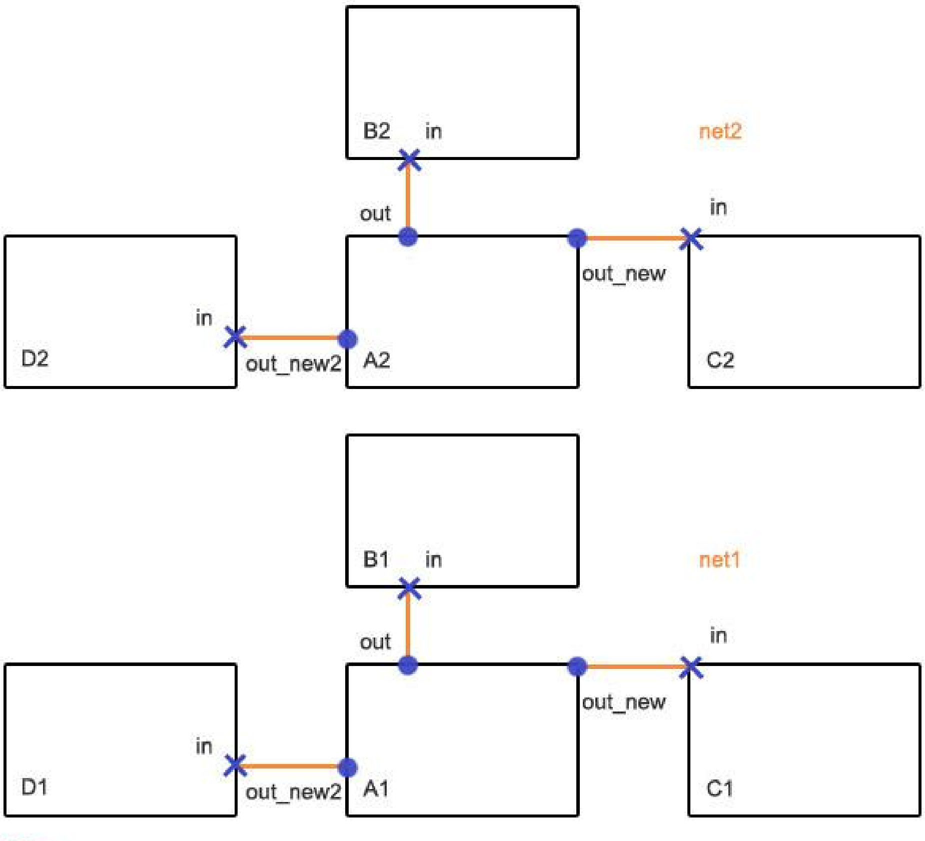
The following topology file is used:

```
version 2.0;
net net1_1
hterm-hterm A1/out1 B1/ft_in1;
hterm-hterm B1/ft_in1 B1/ft_out1;
hterm-hterm B1/ft_out1 C1/in1;
end net

net net1_2
hterm-hterm A1/out2 B1/ft_in2;
hterm-hterm B1/ft_in2 B1/ft_out2;
hterm-hterm B1/ft_out2 C1/in2;
end net

net net2_1
hterm-hterm A2/out1 B2/ft_in1;
hterm-hterm B2/ft_in1 B2/ft_out1;
hterm-hterm B2/ft_out1 C2/in1;
end net
```

```
net net2_2
hterm-hterm A2/out2 B2/ft_in2;
hterm-hterm B2/ft_in2 B2/ft_out2;
hterm-hterm B2/ft_out2 C2/in2;
end net
```

- **Example 4 : Reuse case for multi-fanout case of Example 2**
  The following is a before and after feedthrough insertion image of two multi-fanout nets with three sink partitions each.





After

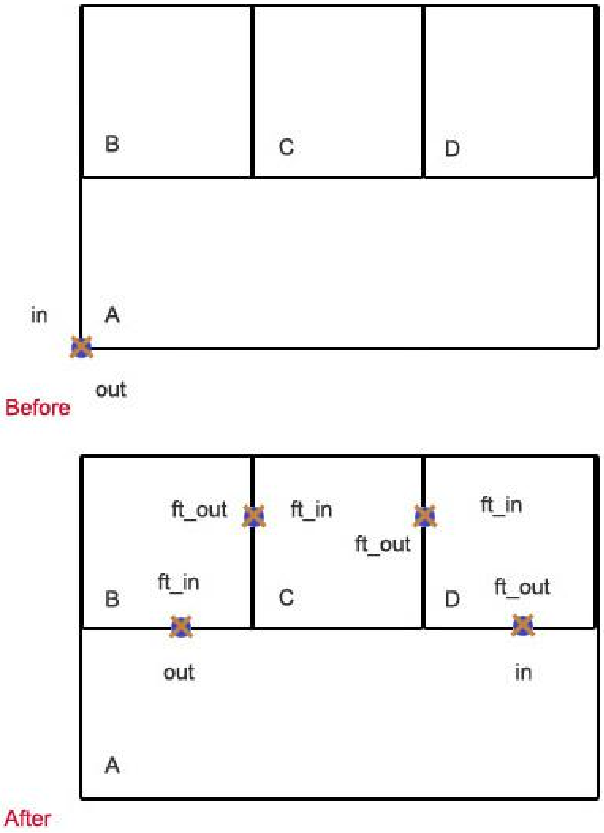
The following topology file is used:

```
version 2.0;
net net1
instterm-hterm driver_term A1/out;
instterm-hterm driver_term A1/out_new;
instterm-hterm driver_term A1/out_new2;
hterm-hterm A1/out B1/in;
hterm-hterm A1/out_new C1/in;
hterm-hterm A1/out_new2 D1/in;
end net

net net2
instterm-hterm driver_term A2/out;
instterm-hterm driver_term A2/out_new;
instterm-hterm driver_term A2/out_new2;
hterm-hterm A2/out B2/in;
hterm-hterm A2/out_new C2/in;
```

```
hterm-hterm A2/out_new2 D2/in;
end net
```

- **Example 5: Complex case for loopback net**

  The following is a before and after feedthrough insertion image of a loopback net.

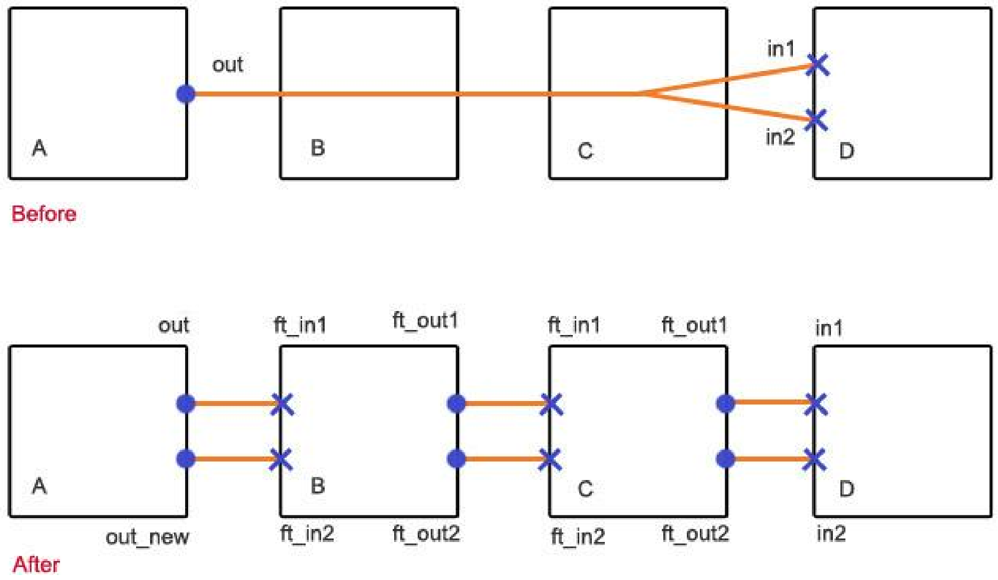


The following topology file is used:

```
version 2.0;
net loopback
hterm-hterm A/out B/ft_in;
hterm-hterm B/ft_in B/ft_out;
hterm-hterm B/ft_out C/ft_in;
hterm-hterm C/ft_in C/ft_out;
hterm-hterm C/ft_out D/ft_in;
hterm-hterm D/ft_in D/ft_out;
hterm-hterm D/ft_out A/in;
end net
```

- **Example 6: Complex case for multi-fanout net**

  The following is a before and after feedthrough insertion image of a complex multi-fanout net.



The following topology file is used:

```
version 2.0;
net mfanout
instterm-hterm driver_term A/out;
instterm-hterm driver_term A/out_new;
hterm-hterm A/out B/ft_in1;
hterm-hterm B/ft_in1 B/ft_out1;
hterm-hterm B/ft_out1 C/ft_in1;
hterm-hterm C/ft_in1 C/ft_out1;
hterm-hterm C/ft_out1 D/in1;
hterm-hterm A/out_new B/ft_in2;
hterm-hterm B/ft_in2 B/ft_out2;
hterm-hterm B/ft_out2 C/ft_in2;
hterm-hterm C/ft_in2 C/ft_out2;
hterm-hterm C/ft_out2 D/in2;
end net
```

Version 1.0 of the Topology File - Syntax and Examples

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
     ...
```

Note: The syntax is case sensitive.

The description of the syntax is as follows:

nametype	Can be `net`, `bus`, or `netgroup`. The value `netgroup` represents all nets in the net group. You should update the net group after feedthrough insertion step. Here are some examples of nametype: ● `bus myBus[0:1]` specifies bus bits ● `net myBus[0:1]` specifies a scalar net. ● `bus myBus[1]` specifies a bus bit ● `net mybus[1]` specifies a scalar net or a bus bit. In case both exist in the design, use the Verilog escape name and use the `dbgIsBackSlashInNamesHiddenFlag` variable to resolve correctly.

netname	Can be a net name, bus name, or a net group name. Wildcards (* or ?) can be used for net name, bus name, or net group name. If more than one net group is matched with wildcard, the `insertPtnFeedthrough` command will issue a warning message and: • use only the first matched net group • ignore the other ones. Wild cards can only be used for a bus name BUT not bus range. For example, you cannot specify bus busname[1:*]. *Specifying bus entries* : If a bus named `databus` has 32 bits (from 0 to 31), its r bus entries are specified as follows: • `bus databus` specifies all 32 bits from 0 to 31 • `bus databus[13:23]` specifies databus[13] to databus[23] • `bus databus[13]` specifies only the bit 13 of databus You cannot provide any net-specific entries for multiple bus bits, net groups, or wildcard nets. Hence, bus topologies cannot be specified for bus nets connected to top-level instance pins or to I/O pins. *Using escape mechanism for special characters:* The following escape mechanisms remove all restrictions on characters: • `\\` for the backslash character (\) itself • `\b` for blank • `\t` for tab • `\n` for new line • `\0` for null • `\s` for semicolon (semicolon (;) is the path statement terminator). Any other character which follows a backslash (\) is taken literally. For example, `\a` is considered as `a`. If one wants to use *,? literally then must use escaping as these are used for wildcards.
	Note: If a net appears twice in any form, the first entry corresponding to the net is used. The subsequent entries generate an error.

fromtype	Can have one of the following values: • `io` for I/O pins • `hinst` for hierarchical instance name of a partition or partition clone • `instterm` for top-level instance pins
totype	Can have one of the following values: • `io` for I/O pins • `hinst` for hierarchical instance name of a partition or partition clone • `instterm` for top-level instance pins • `hinstfb` for hierarchical instance name of a partition or partition clone. This can only be used as part of the combination `hinst-hinstfb`, which specifies a feedback buffer path.
`version`	Version is the format version. The format version for this release is 1.0.
`route_data`	Optional field that specifies routing information. This is not a user-specified field. This field is created when the `insertPtnFeedthrough` command is run with the `-saveTopoFile` parameter. This field is used only for ECO purposes. The `route_data` parameter is not available if the *totype* is `hinstfb`.

All version- and topology-statements in the topology file end with a semicolon (;). Any extra spaces are ignored. Here is an example of a topology file:

```
############################
version 1.0;
net net1
io-hinst net1 i_b;
hinst-instterm i_b inst_c/net1;
end net

net clk*
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end net
```

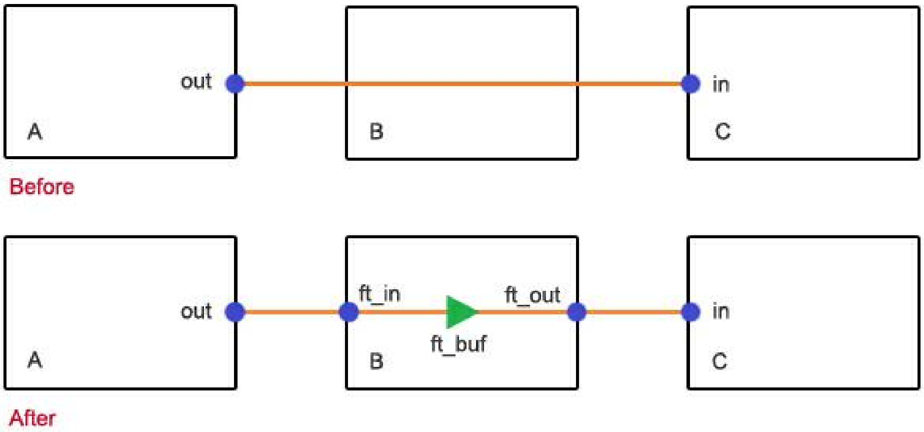
```
netgroup group_a
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end netgroup
```

```
bus databus[0:31]
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end bus
############################
```

Examples of usage of topology file version 1.0

The following examples illustrate topology file usage:

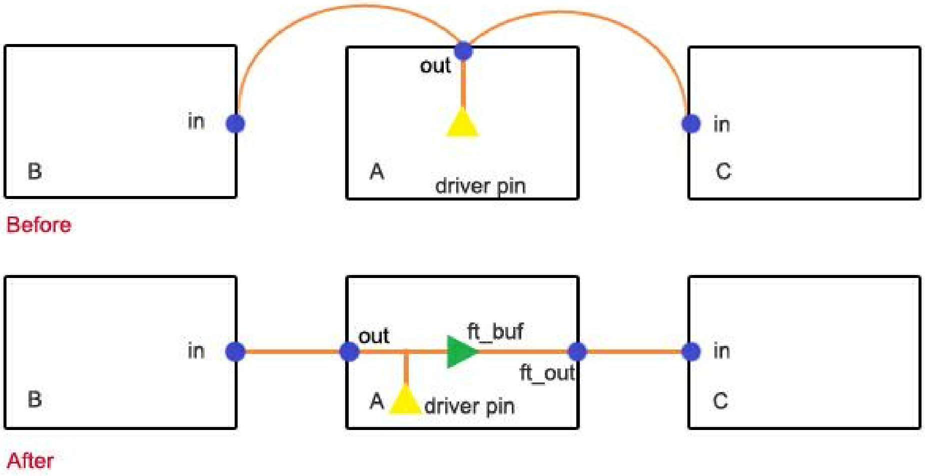
- **Example 1: Two pin net**
  The following image shows a two pin net before and after feedthrough insertion. It has an output port on partition A and an input port on partition C.



  In order to insert a buffer in partition B, the following topology syntax is used:

```
version 1.0;
net netname
hinst-hinst A B;
hinst-hinst B C;
end net
```

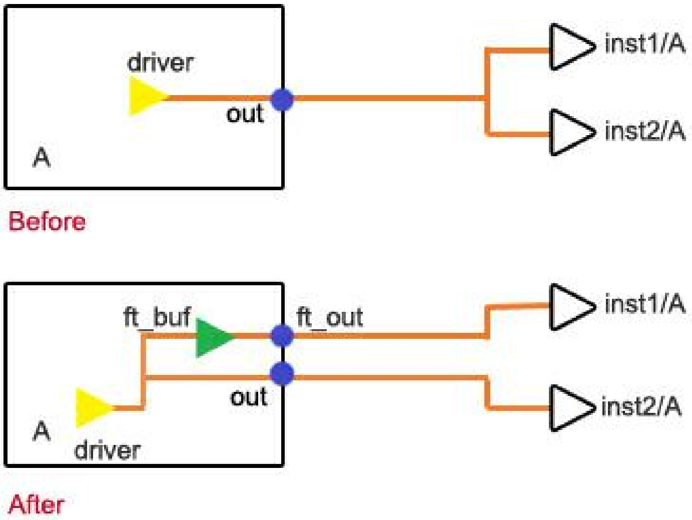
- **Example 2: Multi-fanout net with two sink partitions**
  The following is a before and after feedthrough insertion image of multi-fanout net with two sink partitions. It has output port on partition A and input ports on partitions B and C.



The following topology file is used

```
version 1.0;
net netname
hinst-hinst A B;
hinst-hinst A C;
end net
```

- **Example 3: Multi-fanout net with 2 sinks which are standard cells**
  The following is a before and after feedthrough insertion image of a multi-fanout net with 2 sinks which are standard cells.

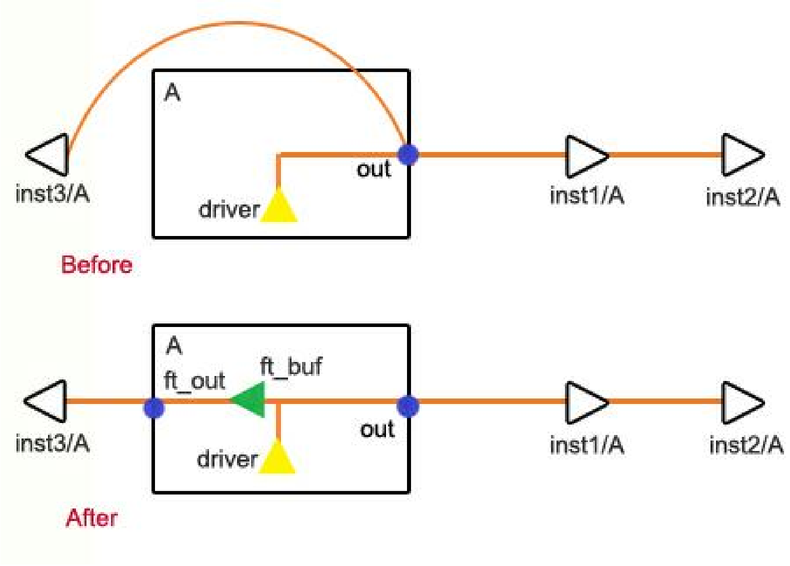


The following topology file is used:

```
version 1.0;
```

```
net netname
hinst-instterm A inst1/A;
hinst-instterm A inst2/A;
end net
```

- **Example 4 : Multi-fanout net with 3 sinks which are standard cells**
  The following is a before and after feedthrough insertion image of a multi-fanout net with 3 sinks which are standard cells. It is not desired to insert separate buffers for inst1 and inst2.



The following topology file is used:

```
version 1.0;
net netname
hinst-instterm A inst1/A;
hinst-instterm A inst3/A;
instterm-instterm inst1/A inst2/A;
end net
```

Version 0.5 of the Topology File - Syntax and Examples

The Version 0.5 of the Topology file is used only while using the Generalized Feedthrough Paths capability. This topology file version can be written by the user and then be used by the write_generalized_feedthru_paths command. Alternatively, you can use the *Path Guidance* page of the *Browse/Plan Partition Feedthroughs* GUI form to generate the topology file in version 0.5 format that contains a path for one master and clone chain to derive path of all master and clone chains.

The syntax of the topology information in the version 0.5 of the topology file file is as follows.

```
# Comment line

  version version_string;
     path_nets*
          fromtype-totype from_name to_name;
          fromtype-totype from_name to_name;
          ...
    end path_nets
```

Example of Topology File Version 0.5:

```
version 0.5;
path_nets *
hinst-hinst iW1/it1/ia1 iW1/it1/ib1;
hinst-hinst iW1/it1/ib1 iW1/it1/ic1;
hinst-hinst iW1/it1/ic1 iW1/it1/id1;
end path_nets
```

The description of the syntax is as follows

`fromtype`	Can have one of the following values: • `hterm`: The `from_name` is to be specified as hinstname/portname. If the port names specified don't exist already then they will be created. • `instterm`: ◦ For top level pins the `from_name` is specified as instname/termname. ◦ For multi-fanout nets originating from a hierarchical module which is a partition or a clone the topology must originate from the driver term. Instead of specifying the complete driver name such as instname/termname one can use the shorthand `driver_term`. Refer to examples 2,4,and 6 given below. • `io`: For chip level I/O pins.
`totype`	Can have one of the following values: • `hterm`: The `to_ name` is to be specified as hinstname/portname. If the port names specified don't exist already then they will be created. • `instterm`: For top level instance pins. The `to_name` is specified as instname/termname. • `io`: For chip level I/O pins.
`path_nets`	Keyword for user specified set of paths for which generalized path will be derived. **Note:** It is necessary to use *
`version`	The version is 0.5. The topology file is used only by the write_generalized_feedthru_paths command.

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:

- Import a design.
- Perform floorplanning on the design.
- Perform feedthrough buffer insertion and save the feedthrough buffer topology tree information in a file (use the `-saveTopoFile` parameter of the `insertPtnFeedthrough` command).
- Import design with a new ECO netlist.
  Note: The ECO netlist should not contain the original inserted feedthrough buffers.
- 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.
- Repeat steps 4 and 5 for more ECO netlists, if required.

Reducing the Number of Buffers and Ports Added for Route-based Feedthrough Insertions

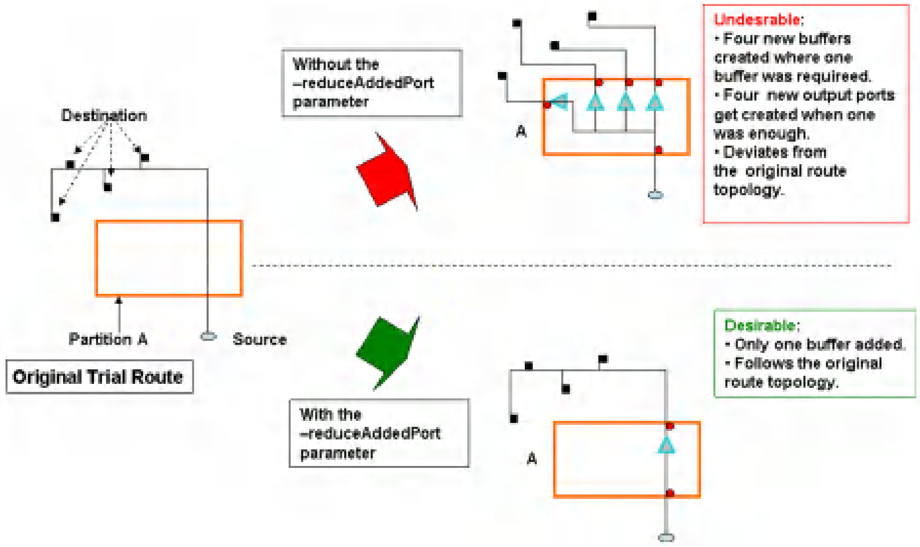
You can use the `-reduceAddedPort` parameter of the `insertPtnFeedthrough` command to specify that feedthrough insertion should follow the routing topology more closely. This can help reduce the number of added ports and buffers. The ports are created at the route crossing points. The status of the added ports is set to *Fixed* . Subsequent use of Early Global 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:

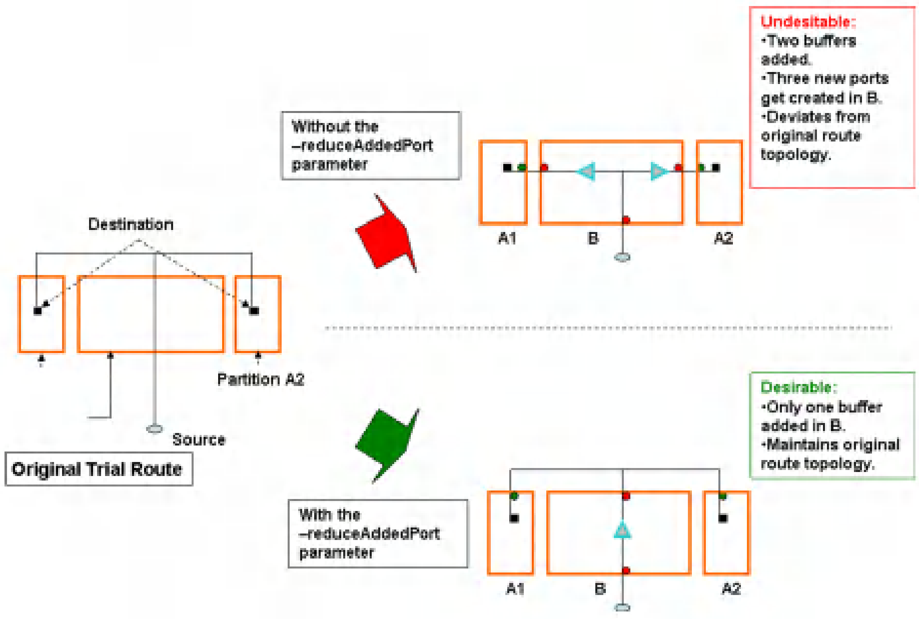
- **Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels**

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.


Destination
Partition A
Source
Original Trial Route
Without the
–reduceAddedPort
parameter
A
Undesrable:
• Four new buffers created where one buffer was requireed.
• Four new output ports get created when one was enough.
• Deviates from the original route topology.
Desirable:
• Only one buffer added.
• Follows the original route topology.
With the
–reduceAddedPort
parameter
A

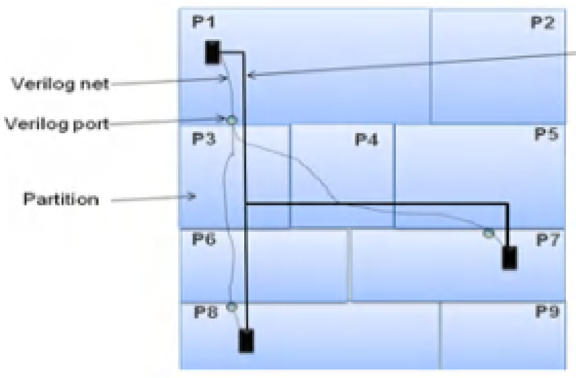
Net Connecting Through Adjoining Partition

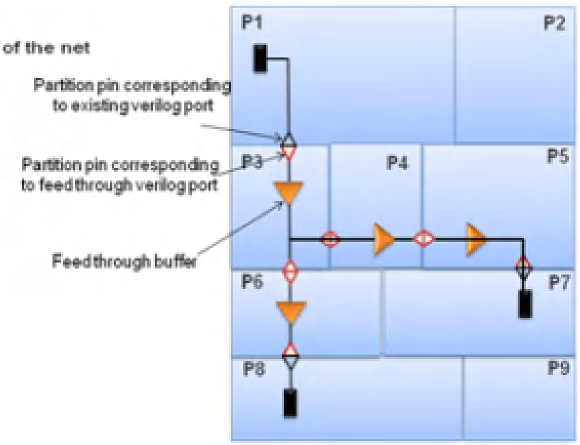
The following diagram illustrates the improvement in feedthrough insertion between partitions where there is another partition in between.


Undesirable:
•Two buffers added.
•Three new ports get created in B.
•Deviates from original route topology.
Without the –reduceAddedPort parameter
A1
B
A2
Destination
Partition A2
Source
Original Trial Route
Desirable:
•Only one buffer added in B.
•Maintains original route topology.
With the –reduceAddedPort parameter
A1
B
A2

Mentioning Some Verilog Modules as dont-add-ports

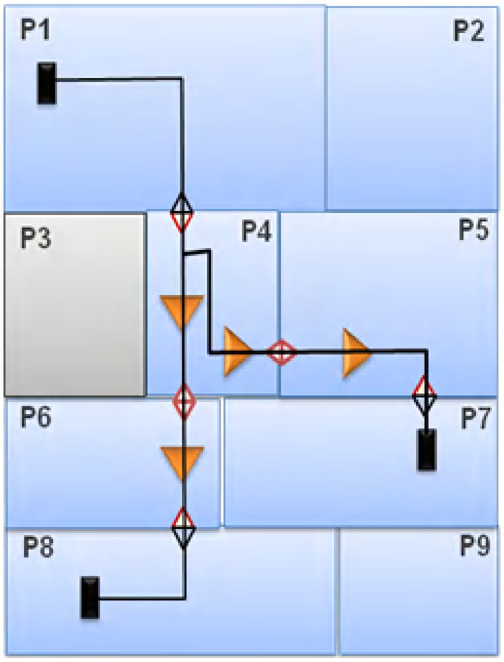
During `insertPtnFeedthrough`, if a new verilog port gets created in a module, then the tool does not create new ports in the module if it is listed in the `dont_add_ports_to_module` option. For example, without the `dont_add_ports_to_module` option, the feedthrough buffers and new verilog ports are added as shown in the figures below:




P1
P2
Partition pin corresponding to existing verilog port
Partition pin corresponding to feedthrough verilog port
P3
P4
P5
Feedthrough buffer
P6
P7
P8
P9

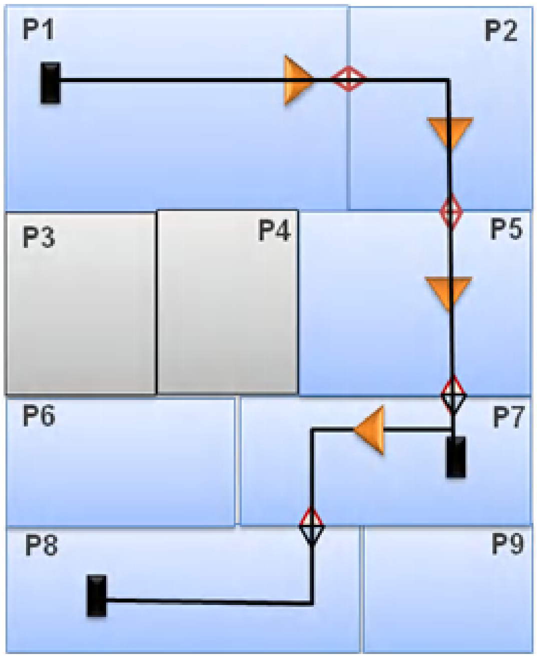
However, if some modules are specified in `dont_add_ports_to_module`, then new ports are not created in the specified modules. For the design scenario shown above, here are some examples of the usage of `dont_add_ports_to_module`.

- `insertPtnFeedthrough -dont_add_ports_to_module {P3_hi}`

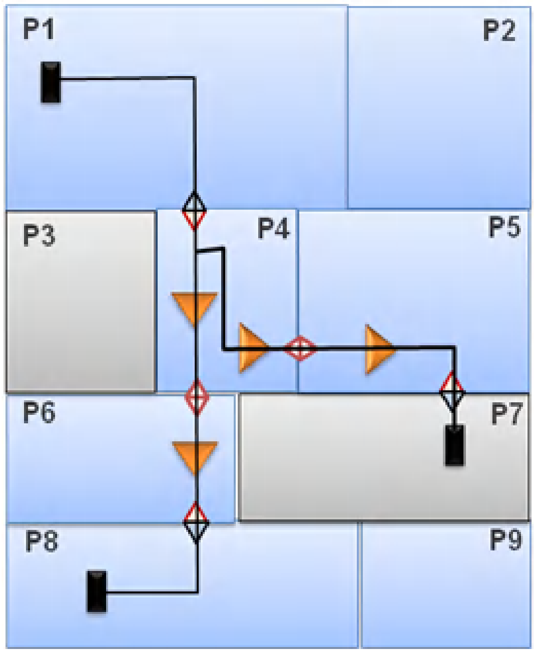


- `insertPtnFeedthrough -dont_add_ports_to_module {P3_hi P4_hi}`

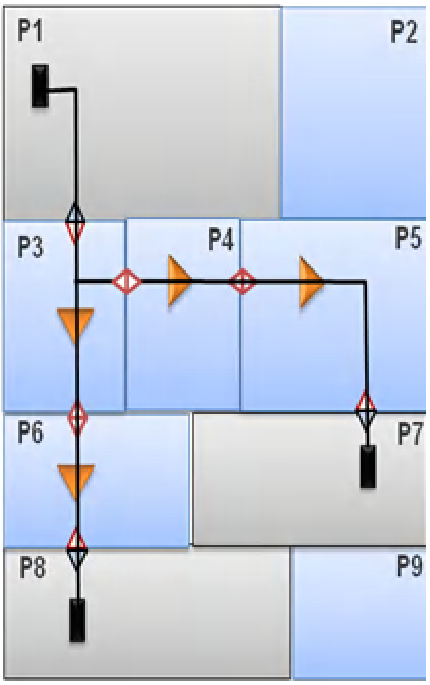
  An alternate path is created leaving the partitions mentioned in `dont_add_ports_to_module`.



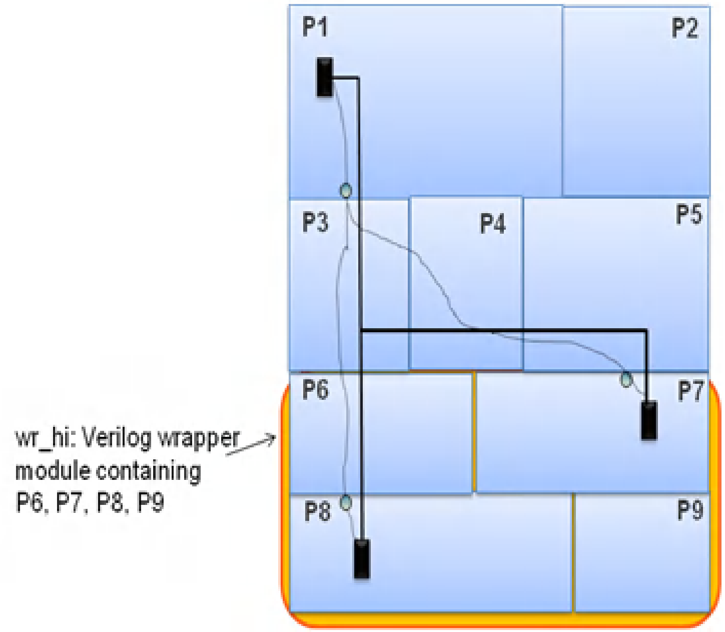
- `insertPtnFeedthrough -dont_add_ports_to_module {P3_hi P7_hi}`



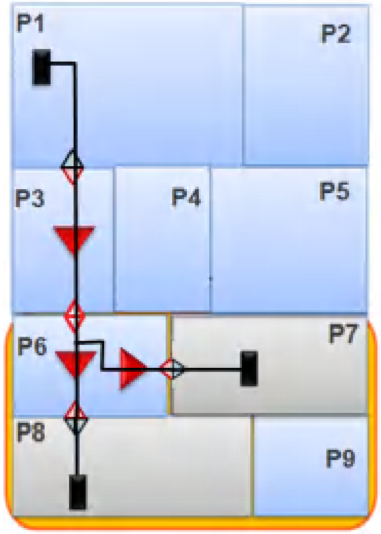
- `insertPtnFeedthrough -dont_add_ports_to_module {P1_hi P7_hi P8_hi }`



Consider another design scenario now, as shown below. Note that `wr_hi` is the verilog wrapper module containing `P6 (wr_hi/P6_hi)`, `P7 (wr_hi/P7_hi)`, `P8 (wr_hi/P8_hi)`, `P9 (wr_hi/P9_hi)`.



- `insertPtnFeedthrough -dont_add_ports_to_module {wr_hi wr_hi/P7_hi  wr_hi/P8_hi}`



Abbreviating Lengthy Feedthrough Net Names

You can abbreviate inserted feedthrough net names so that the net names will not extend too long if you run the `insertPtnFeedthrough` multiple times. With the `-useShortName` option, you can eliminate the use of the old net name and partition names, and instead use a running count for the new net names. For example, if a feedthrough net reset connects two partitions `tt_chiplet` and `video_chiplet`, the feedthrough net name is:

```
FE_FEEDX_NET_C__tt_chiplet_video_chiplet_reset
```

The net name abbreviation convention for feedthrough buffer insertion when using the `insertPtnFeedthrough -useShortName` command are:

Net Names	`FE_FTN_` *n* , where *n* is an integer
Buffer Names	`FE_FTB_` *n* , where *n* is an integer

The net name abbreviation convention for feedback buffer insertion when using the `insertPtnFeedthrough -useShortName` command are:

Net Names	`FE_FB_NET_` *n* , where *n* is an integer
Buffer Names	`FE_FB_BUF_` *n* , where *n* is an integer

Blocking Edges for Feedthrough Insertion

Automatic feedthrough of high fanout nets in abutted designs is a limitation. As a workaround, you can get a topology file for feedthrough of high fanout nets. The `-blockedEdgesFile` parameter of the `insertPtnFeedthrough` command accepts a file, which contains blocked edges to guide feedthrough. This is helpful in symmetric master and clone designs.

Use Model

Use the `insertPtnFeedthrough -blockedEdgesFile` command to provide a file that contains the names of the partitions followed by the edge numbers for the partition which are considered as blocked for the purpose of `insertPtnFeedthrough` command.

The syntax of the blocked partition edge file is as below:

```
<CellName> <edge_number_list>
```

For example,

```
Master1 0 1
Master2 0 1
```

Here, Master1 0 1 signifies that edges 0 and 1 for partition Master1 are blocked. Similarly,

Note:

- Both facing edges of two abutted partitions need to be blocked to ignore a path for feedthrough.
- Blocking one edge is no operation for feedthrough path finding. The edge order or partition

order does not matter.

- If a partition and its edge is repeated in file , sum total of all edges will be blocked
- A blocked pair of edge is considered blocked for all feedthrough paths for selected nets.

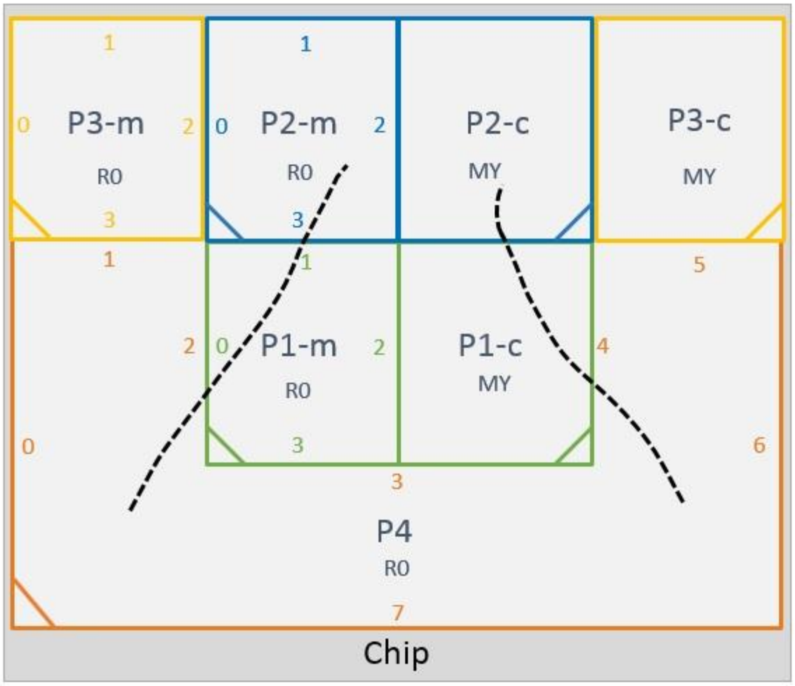
For example, the following command will run and commit the possible feethroughs.

```
insertPtnFeedthrough -blockedEdgesFile ft.blocked -saveTopoFile -checkOnly
```

Note: The topofile can be further edited for path corrections/alterations.

Examples

The following examples illustrate how blocked edges guide feedthrough insertion. The aim is to develop connectivity between partition P4 and P2-m and partition P4 and P2-c by doing feedthrough insertion.



Use the `insertPtnFeedthrough -blockedEdgesFile` command to provide the file containing the names of the partitions followed by the edge numbers for the partition which are considered as blocked for the purpose of `insertPtnFeedthrough` command.

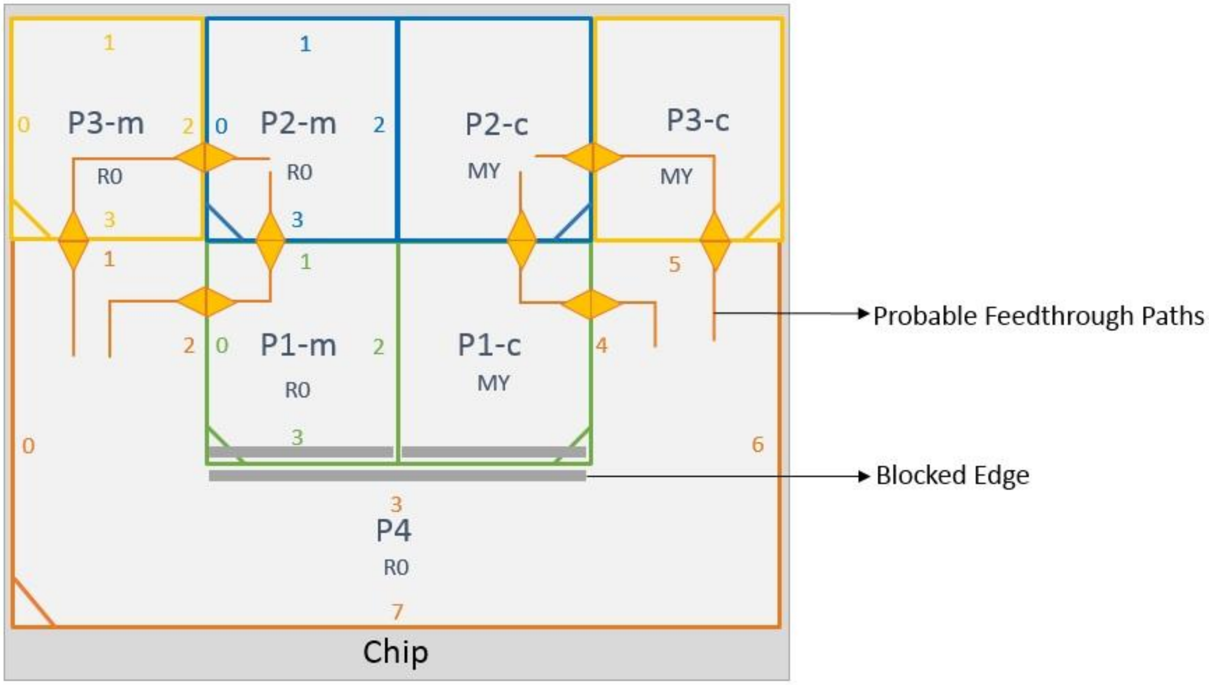
```
insertPtnFeedthrough -blockedEdgesFile ft.blocked
```

The *blockedEdgesFile* can have:

- **Single block edge per partition**

```
ft.blocked:         P4 3      or,
ft.blocked:         P1 3
```

The following illustration shows the blocked edges (as specified in the ft.blocked file) and the probable feedthrough paths.

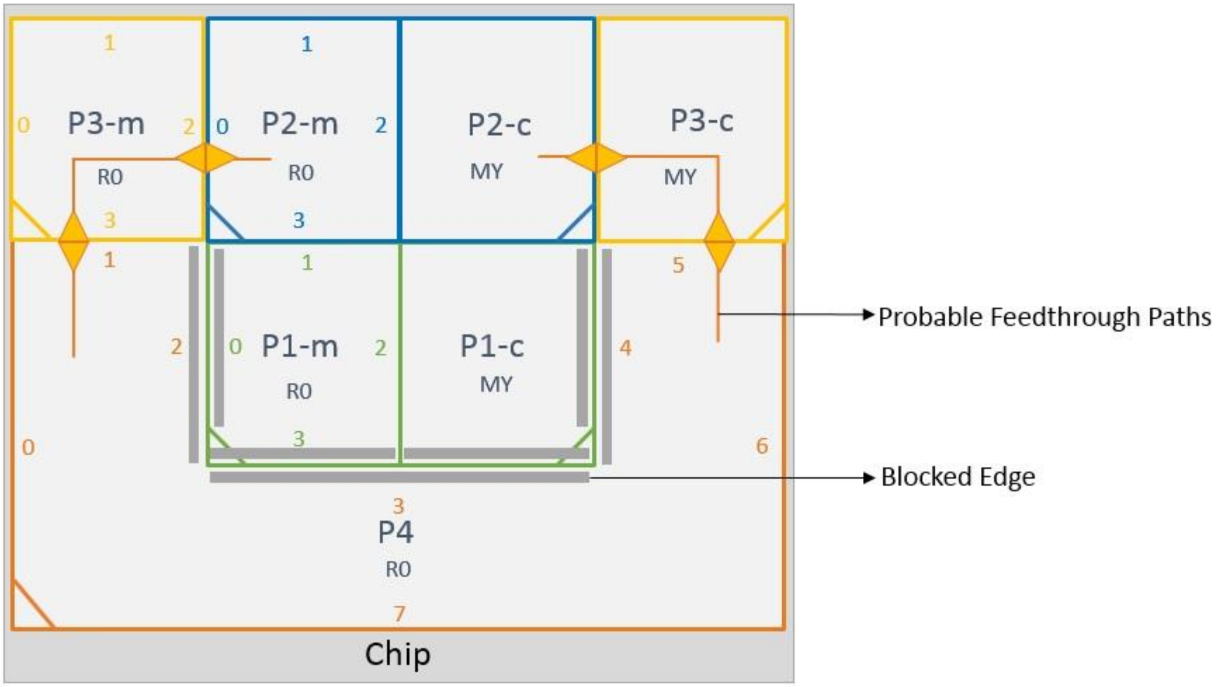


- **Multiple block edges per partition**

  ft.blocked: P4 2 3 4 or,

  ft.blocked: P1 0 3

  The following illustration shows the blocked edges (as specified in the ft.blocked file) and the probable feedthrough paths.



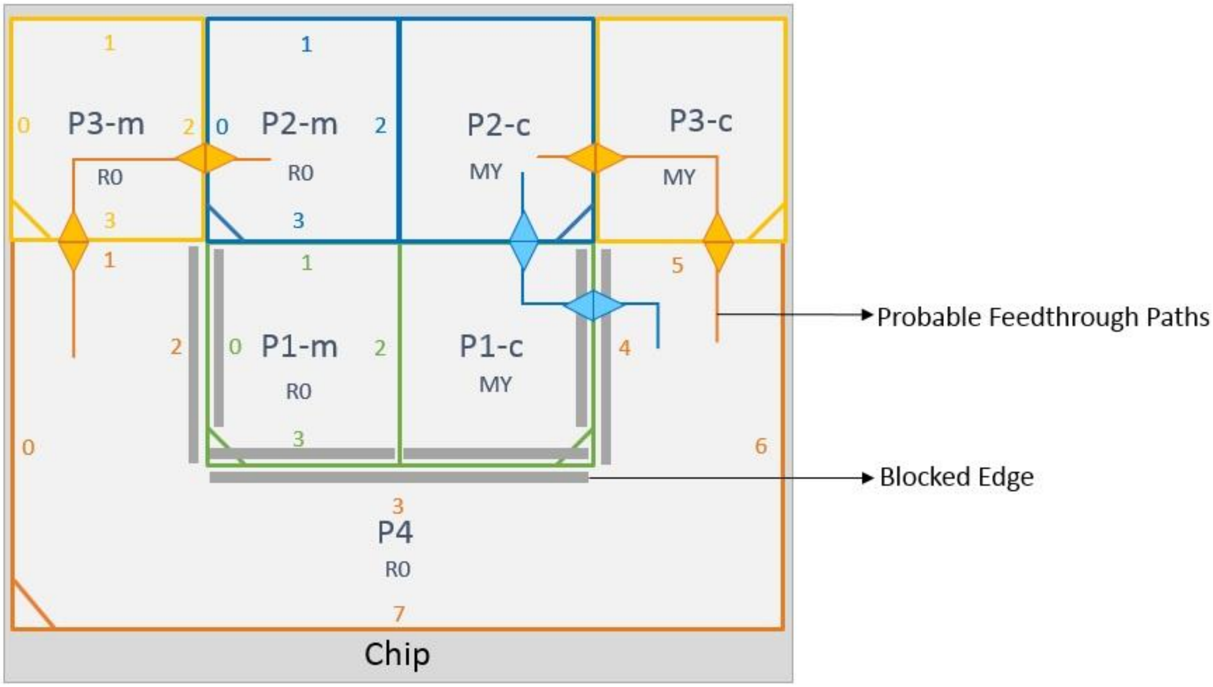
In the above example if we change one edge such that:

ft.blocked: P4 2

ft.blocked: P1 3

Then the feedthrough path from P4 to P2-c can go through P1-c, as shown below in the blue path, but no change will happen for path for P4 to P2-m. This makes the feedthrough asymmetric and port/buffer reuse difficult.

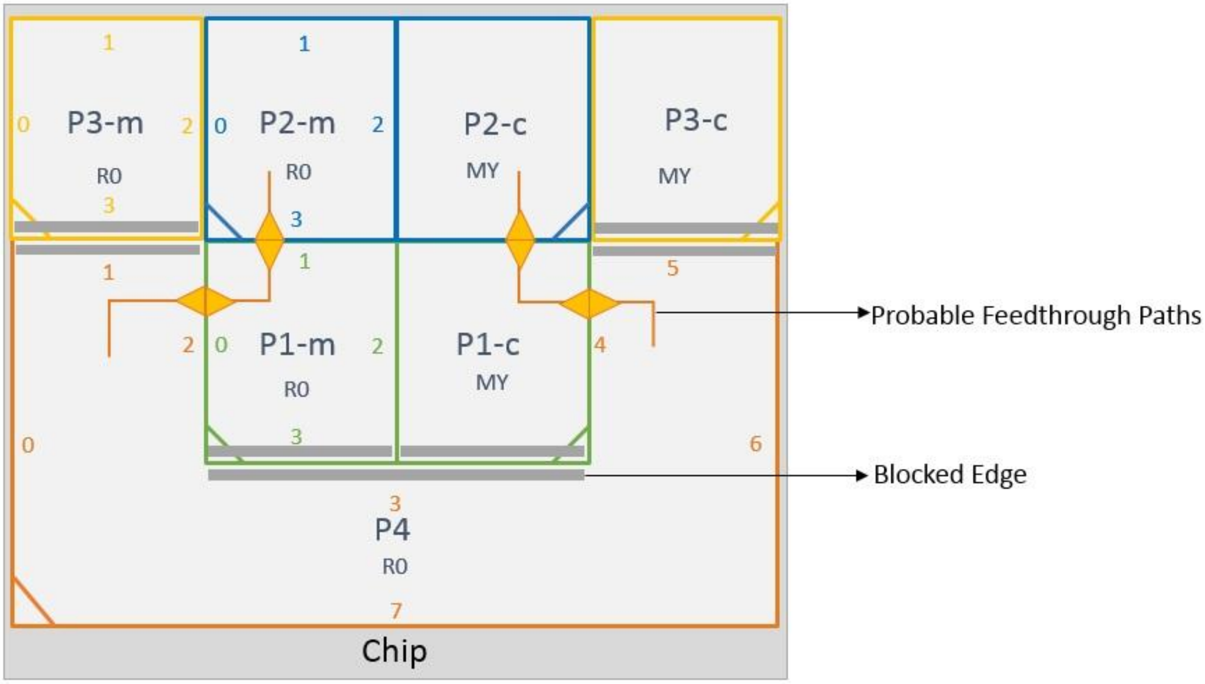
Note: It is better to block a master/clone edge than to block a unique partition edge.



- **A mix combination of block edges**

  ft.blocked: P4 1 3 5
  ft.blocked: P1 3
  P3 3

  The following illustration shows the blocked edges (as specified in the ft.blocked file) and the probable feedthrough paths.



Note: For any other connection, say between P1-m to P3-m and P1-c to P3-c, the same set of blocked edges will apply. The `insertPtnFeedthrough` command will chose a path which does not cross these edges.

Support for Blockage Lines

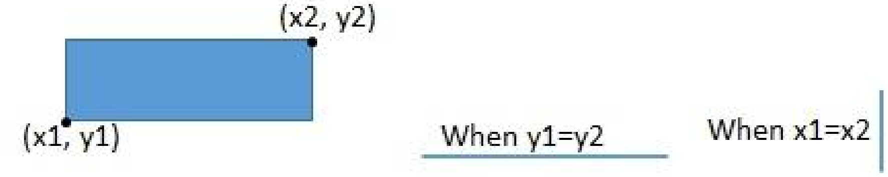
The `insertPtnFeedthrough` command enables you to block partial edges during the placement based feedthrough insertion. Using the `-blockageFile` parameter, you can specify the file that contains the coordinates of line segments that are considered as blockages for placement based feedthrough insertion.

Note: The blockages must be defined as lines that do not overlap with partitions or macros. These blockage lines are supported only with automatic feedthrough insertion or placement based feedthrough insertion.

The following is the format of the blockage file:

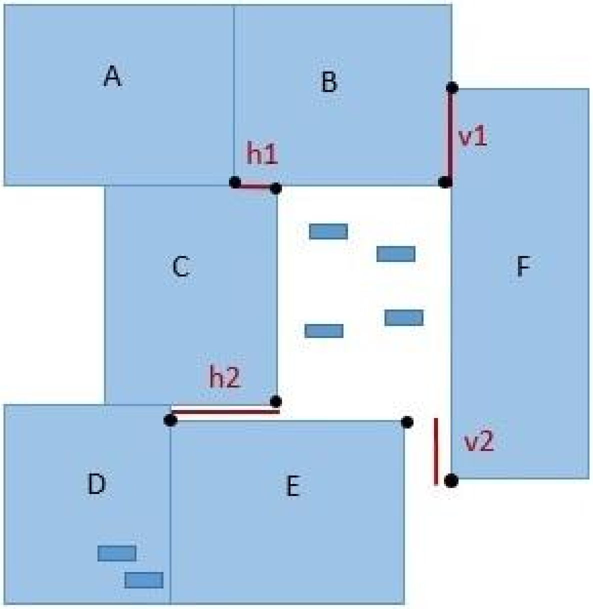
```
x1 y1 x2 y2
x1 y1 x2 y2
...
xn yn xm ym
```

The blockage lines are specified as coordinates of a box, however, the boxes should be of zero width or height.



For a vertical line `x1 = x2`
For a horizontal line `y1` = `y2`

In the following illustration, h1,h2,v1,and v2 are blockage lines.



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.

Using the Feedthrough Ports GUI Menu

> ⚠ The *Browse/Plan Partition Feedthroughs* form provides enhanced GUI support for Feedthrough Insertion and Debugging. This form is a limited-access feature in this release. It is enabled by a variable specified using 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.

You can use the *Feedthrough Ports* menu to inserts feedthrough buffers into the partitions to change the partition netlists, and avoid routing nets over partition areas. Additionally, you can use this menu for the following:

- Change both the top-level and partition-level netlists.
- Specify a file that contains net names for which to insert feedthrough buffers.
- Save feedthrough insertion buffer topology tree information in a file.

The *Feedthrough Ports* menu command opens the *Browse/Plan Partition Feedthroughs* form.

Note: For a description of all the fields of the *Browse/Plan Partition Feedthroughs* form, see the Partition Menu chapter in the *Menu Reference*.

Inserting Feedthrough Buffers Using the GUI

You can use the *Automatic* page of the *Browse/Plan Partition Feedthroughs* form to insert feedthrough buffers into the partitions to avoid routing a net over a block area. This automatically runs ECO placement and changes the original netlist.

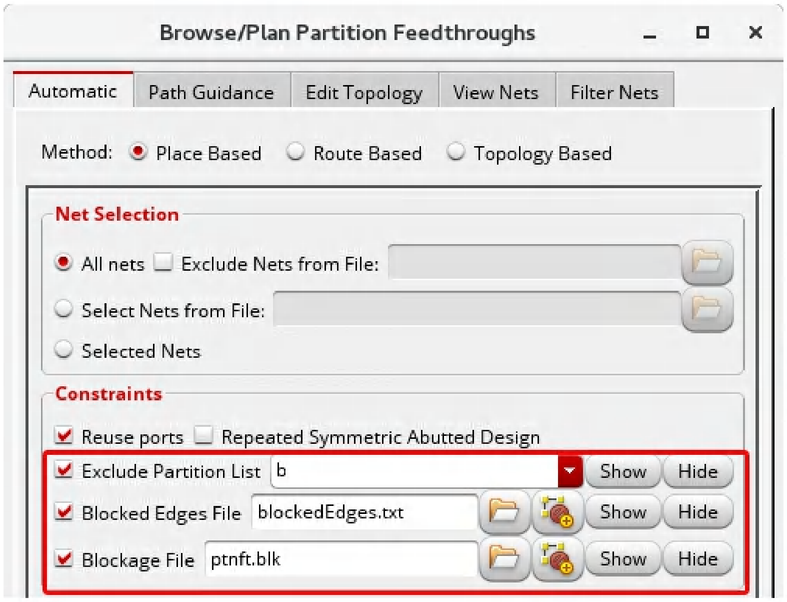
Use Model:

1. Select the method to use for defining feedthrough insertion. Based on your selection, the availability of different GUI items of the form changes.
    - To derive the feedthrough topology based on the placement and shape of the elements of the design irrespective of the routing, use *Place Based*.
    - To derive the feedthrough topology based on the routing results, use *Route Based*.
    - To derives the feedthrough topology based on the topology defined in the file, use *Topology Based.*

2. (*Place Based* and *Route Based*) Specify options for selecting nets for feedthrough insertion.
3. (*Topology Based*) Specify the file that has the topology tree information. This information is used to create feedthrough buffers for the netlist.
4. (Optional) Specify the different constraints on finding path.
5. (Optional) Specifies the different options for feedthrough insertion.
6. Click *OK* or *Apply* on the *Automatic* page of the *Browse/Plan Partition Feedthroughs* form to insert the feedthrough buffers.

Example: Place-based feedthrough insertion by interactive path constraining

The following image shows a place based feedthrough insertion.



Browse/Plan Partition Feedthroughs

Automatic | Path Guidance | Edit Topology | View Nets | Filter Nets

Method: ● Place Based ○ Route Based ○ Topology Based

Net Selection

● All nets □ Exclude Nets from File:

○ Select Nets from File:

○ Selected Nets

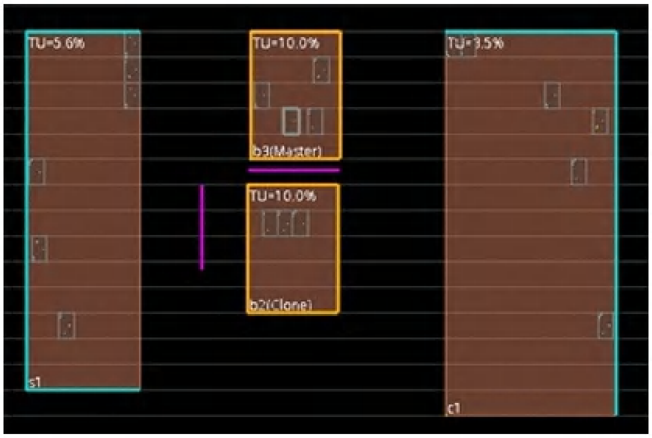
Constraints

☑ Reuse ports □ Repeated Symmetric Abutted Design

☑ Exclude Partition List b Show Hide

☑ Blocked Edges File blockedEdges.txt Show Hide

☑ Blockage File ptnft.blk Show Hide



Here, you can use the *Browse/Plan Partition Feedthroughs* form for interactive path constraining by using the *Show* and *Hide* button next to the different constraints on the form.

- Click *Show* or *Hide* to exclude partition *b* from feedthrough paths consideration.
- Click *Show* or *Hide* to show or hide certain edges of partition being blocked for feedthrough through the *blockedEdges.txt* file.
- Click *Show* or *Hide* to show or hide blockage lines for avoiding feedthrough path through the *ptnft.blk.* file. You can click Add to add blockage lines directly in design and automatically save them to the *ptnft.blk.* file.

Note: For a description of all the fields of the *Browse/Plan Partition Feedthroughs - Automatic* form, see the Partition Menu chapter in the *Menu Reference*.

Writing Generalized Feedthrough Paths

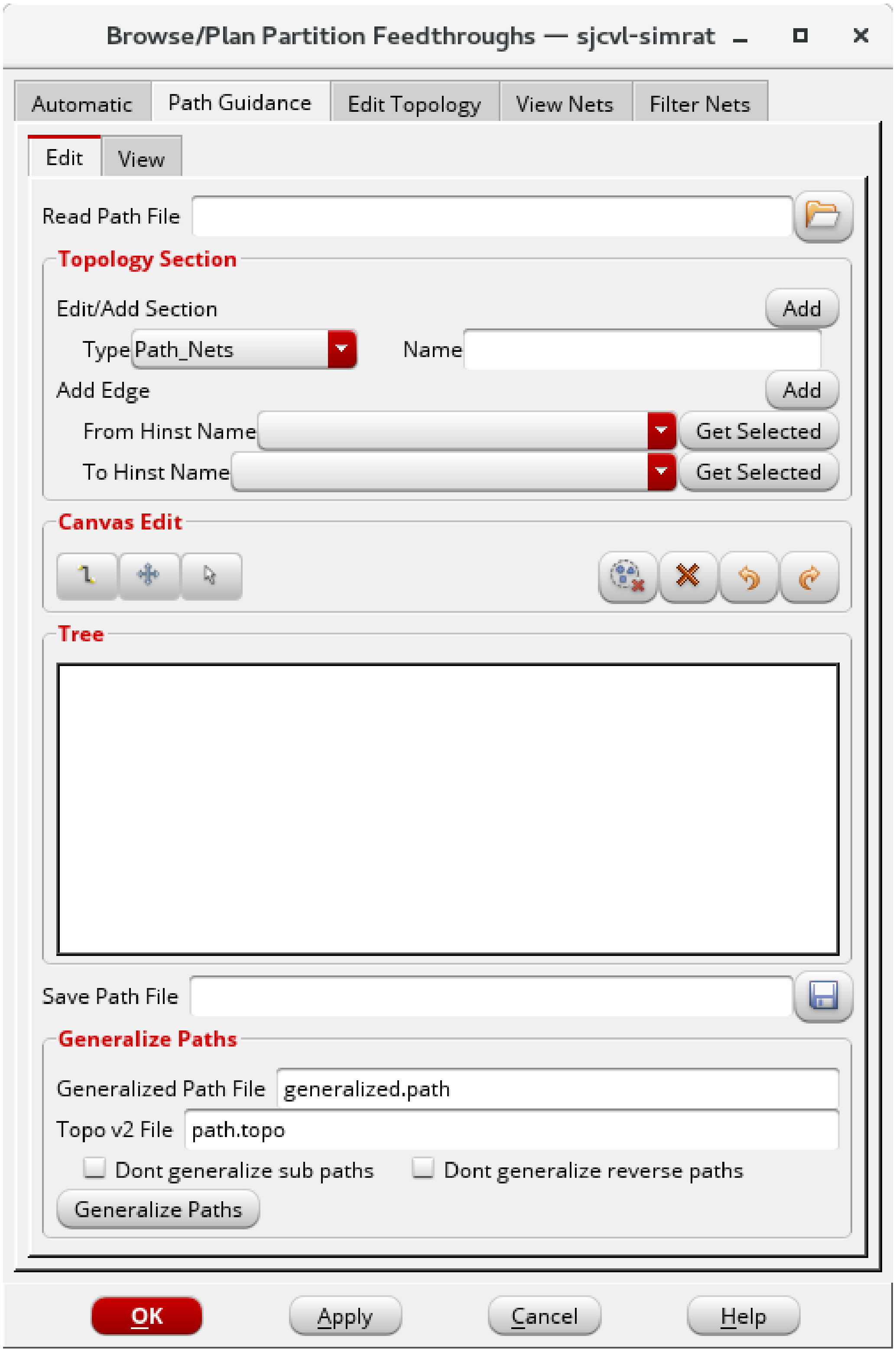
In any design, feedthroughs are required to make non-neighbor and multi-ptn-pin nets into 2-ptn-pin nets. With designs getting bigger and complex, the presence several partitions in designs results in hundreds of possible combinations for feedthrough paths based on the connectivity of different partitions. The path guidance based feedthrough capability provides a way to generalize and automate the feedthrough paths. You can use the *Path Guidance* page of the *Browse/Plan Partition Feedthroughs* form to derive feedthrough topological paths from the user specified paths to cover all the chain connectivity probabilities in a master and clone hinst scenario. It is used to create or view the topology file version 5.0 that contains user specified set of paths for which generalized paths will be derived. The topological file, in version 0.5 is further converted to version 2.0 topological file.

Alternatively, you can use the `write_generalized_feedthru_paths` command to read an input topological file, in version 0.5 and then create the version 2.0 topological file.

The cases where the tool can generalize the paths include:

1. Partition to Partition Generalization: Nets from one hisnt pair to another that are all on the same path. These can have one generalized path that applies for all nets from hisnt to hinst pair.
2. Partition Chain Generalization: The tool can automatically generalize the feedthrough path if the master and clone (source-sink) chain is the same. The user can specify one path and the tool can automatically derive the generalized path for all symmetrical master and clone chains.
3. Partition Sub chain Generalization: The tool can automatically generate sub chain paths if the master and clone chains are the same. The user can specify one path (TIP: Specify the longest chain path to get maximum generalization) and the tool can automatically derive the generalized sub chains.

⚠ The *Path Guidance* option is a limited-access feature in this release. It is enabled by a variable specified using 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.



Browse/Plan Partition Feedthroughs — sjcvl-simrat

Type: Path_Nets

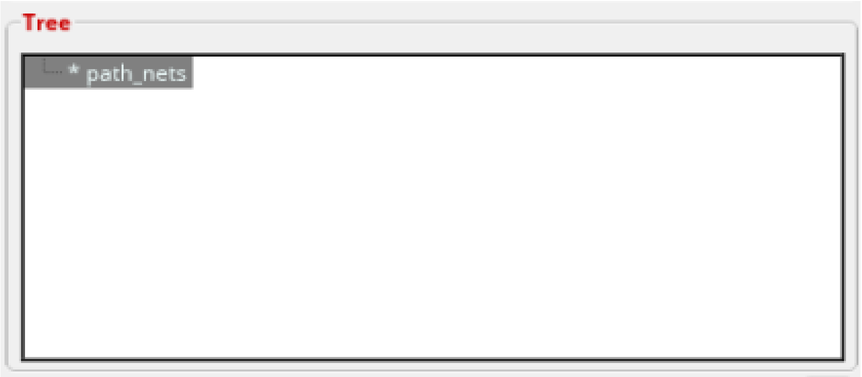
Generalized Path File: generalized.path

Topo v2 File: path.topo

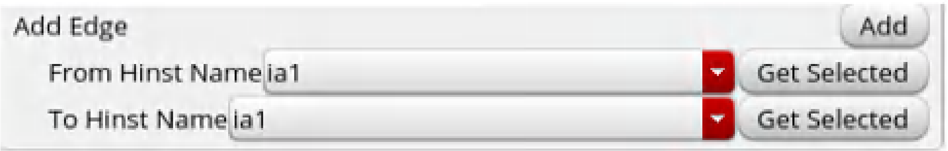
□ Dont generalize sub paths
□ Dont generalize reverse paths

Use Model:

1. Click *Edit* to create or edit a path guidance file (topology file version 0.5). You can click *View* to review the generalized and derived paths. To read and edit an already created user path file, enter the name in the *Read Path File* field.
2. In *Topology Section*, click Edit/*Add Section* and select the *Type* of entry.
   Note: Currently, `Path_Nets` is the only available option.
3. Enter the *Name* for the nets and click the *Add* button. This name displays in the *Tree* area.
   Note: Currently, this option is supported for all nets. You must only specify * as the *Name*.



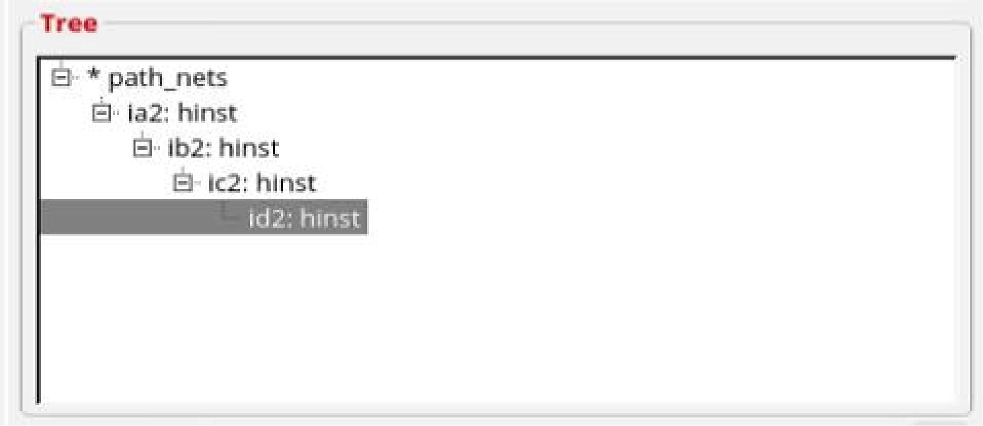
4. Add nodes to the *Path_Nets* section. You can add nodes using either the *Add Edge* area or the interactive *Canvas Edit* area.
   - Using the *Add Edge* area specify the start and end partitions of paths



   a. *From Hinst Name:* Specify the start partition. Alternatively, click *Get Selected* to control the list of partitions being populated by selecting the partitions in the GUI.

   b. *To Hinst Name:* Specify the end partition.

   c. Click the *Add* button to create an edge node for the object in the specified hierarchy. The new node is added to the bottom of the topology.



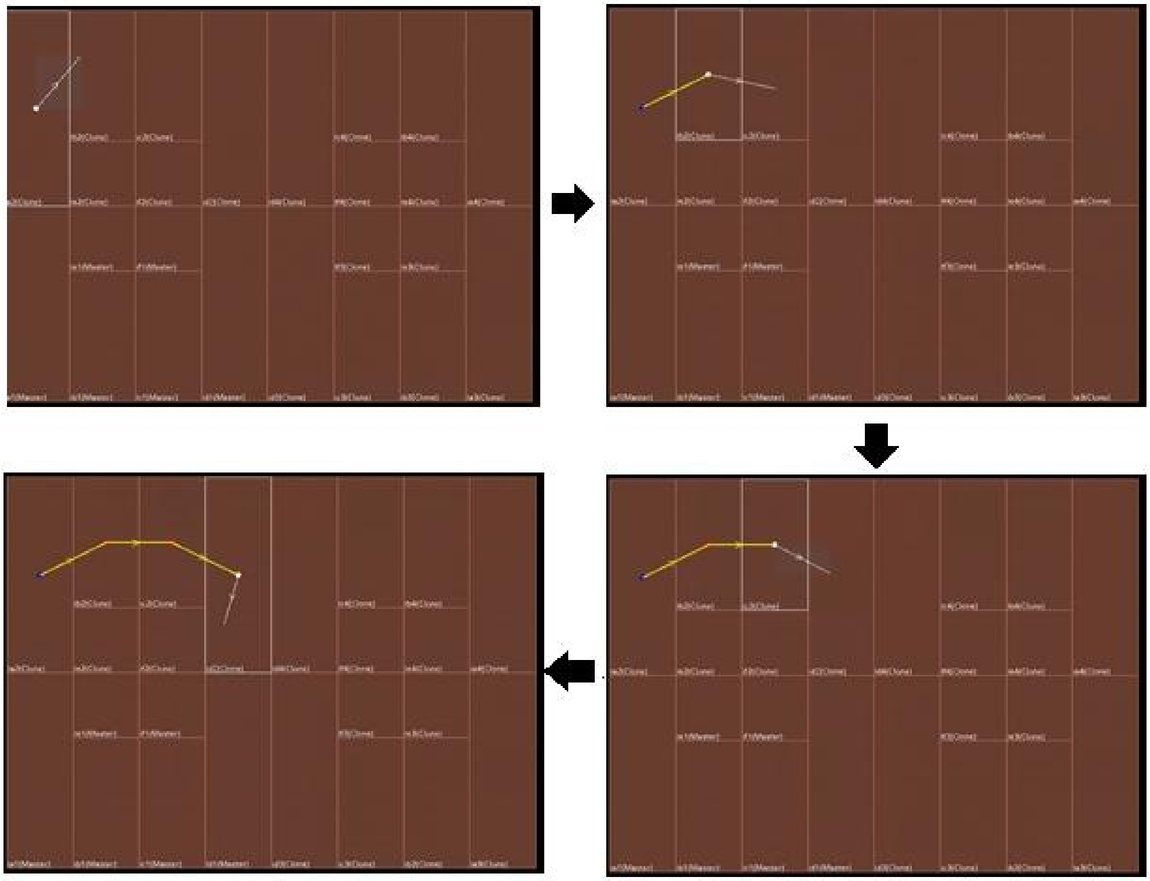
Tree

- * path_nets
  - ia2: hinst
    - ib2: hinst
      - ic2: hinst
        - id2: hinst

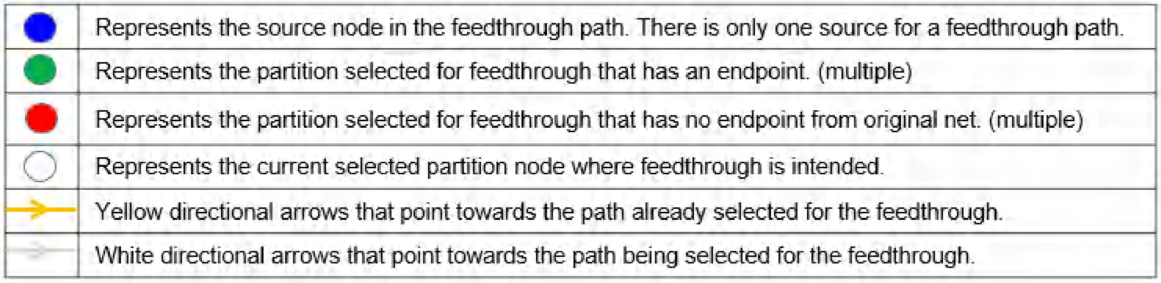
- Using the *Canvas Edit* area:

  a. Click the button in the *Canvas Edit* section to create a node. As soon as we click on a partition in GUI, a node is added to path in the tree.

  b. A node (● blue color) appears on the partition in the Innovus GUI canvas where the net source is. This node is also added and selected in the “Tree” section of form. Partition of the selected node is also selected in the Innovus canvas.



Note: The nodes and direction arrows of the feedthrough path follow a color-coded representation in the GUI for easy identification.



🔵	Represents the source node in the feedthrough path. There is only one source for a feedthrough path.
🟢	Represents the partition selected for feedthrough that has an endpoint. (multiple)
🔴	Represents the partition selected for feedthrough that has no endpoint from original net. (multiple)
○	Represents the current selected partition node where feedthrough is intended.
→	Yellow directional arrows that point towards the path already selected for the feedthrough.
→	White directional arrows that point towards the path being selected for the feedthrough.

Note: The dots for the nodes are always placed at the center of the partition, irrespective of where you click on the partition.

When the cursor is moved in the Innovus canvas, the node changes the color from blue to white. This signifies that the node is selected. Directional arrows showing all the possible endpoints of the net also appear. These arrows follow the cursor. **Note:** For a feedthrough path to be complete, the created path should include all these endpoints.

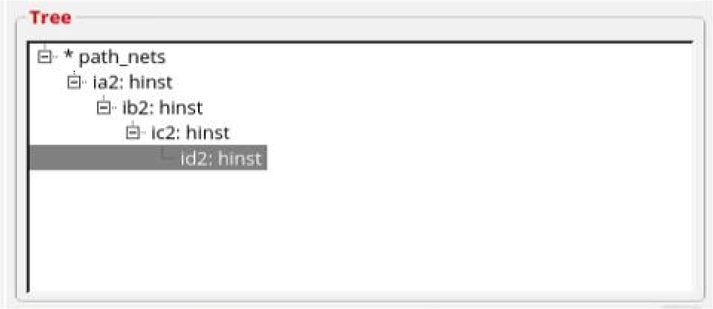
c. Click on any partition to create a feedthrough node. The new feedthrough node added to the path is now displayed in white color, indicating it is now the current selected node. Its partition gets selected in the Innovus canvas. This node is also added in the *Tree* area.
Note: The previous node now takes the original blue color that represents a source node of the net. The directional arrow from the previous node (blue, source node) to the current selected node, changes to yellow color indicating that the path has been already selected for the feedthrough. The white directional arrows now start from the current selected node (white) to the remaining endpoints. Subsequently, click on partitions to add feedthrough nodes and reach an endpoint. The previous node always changes to a different color and the current node becomes white. The nodes that are part of the feedthrough path are displayed in the *Tree* area. **Note:** You can click the *Undo* button to undo the last operation. Node tree is also updated in *Tree* section of form. Selected partition is also updated in Innovus canvas. This is a sequential operation. Alternatively, you can use the *Redo* button to redo the last undo operation.

d. To complete the feedthrough path, double-click on the current node or press the Esc key. The white current node takes the color of the node it represents.

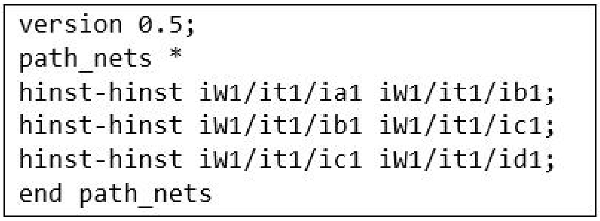
e. Finally, click the ☐ button to edit the Canvas Edit mode.

5. Click *Save Path File* and then specify the name of the topology version 0.5 path guidance file. Click the Save button to save the file.

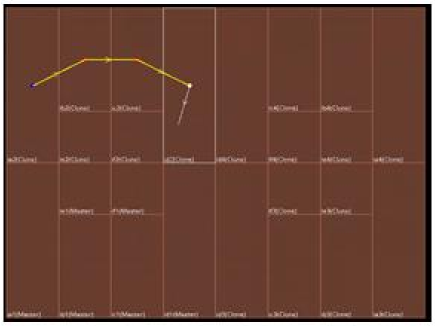
The following image shows the path guidance topology file (version 0.5) created using the *Browse/Plan Partition Feedthroughs* form.



Sample Topology File:



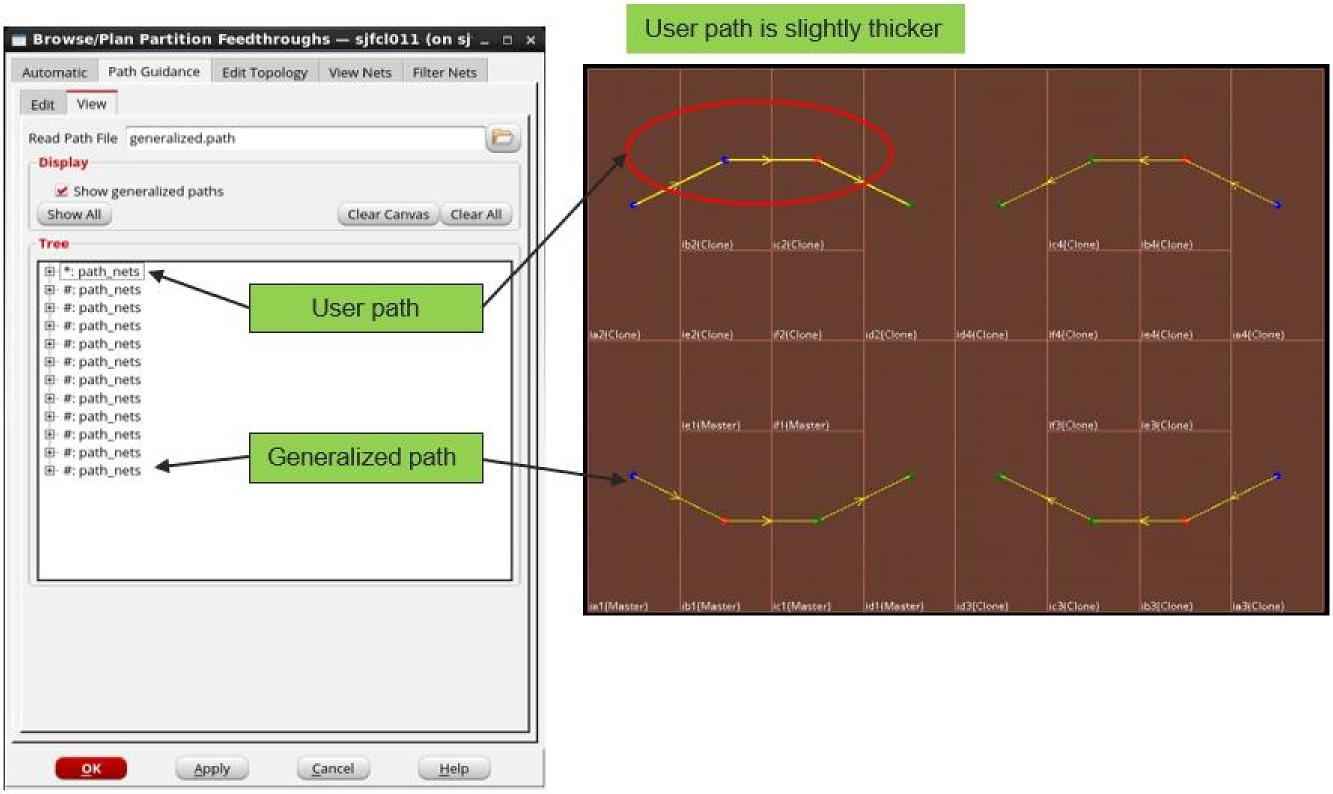
```
version 0.5;
path_nets *
hinst-hinst iW1/it1/ia1 iW1/it1/ib1;
hinst-hinst iW1/it1/ib1 iW1/it1/ic1;
hinst-hinst iW1/it1/ic1 iW1/it1/id1;
end path_nets
```



6. In the *Generalize Paths s*ection, click the Generalize Paths button to automatically derive feedthrough topological paths from the user specified paths to cover all the chain connectivity probabilities in a master and clone hinst scenario.
    a. The *Generalized Path File* field displays the name (`generalized.path`) of the intermediate file containing the generalized paths. It contains the user defined and derived paths.
    b. The *Topo v2 File field* displays the name (`path.topo`) output topological file with version 2.0 format.
    c. Select *Don't generalize sub path*s to make the generalization algorithm avoid generalizing reverse chain paths.
    d. Select *Don't generalize sub path*s to make the generalization algorithm avoid generalizing sub-paths.
7. Click the *View* tab to review generalized and derived paths.
    - The user and generalized paths are auto populated from internal files.
    - User paths depicted by "*"

  - Generalized paths depicted by "#"

8. Click Show Generalized Paths to display the paths in the GUI. You can click the *Show All* button to display all the generalized paths of click on any path listed in the tree area to see it in the GUI


User path is slightly thicker

User path

Generalized path

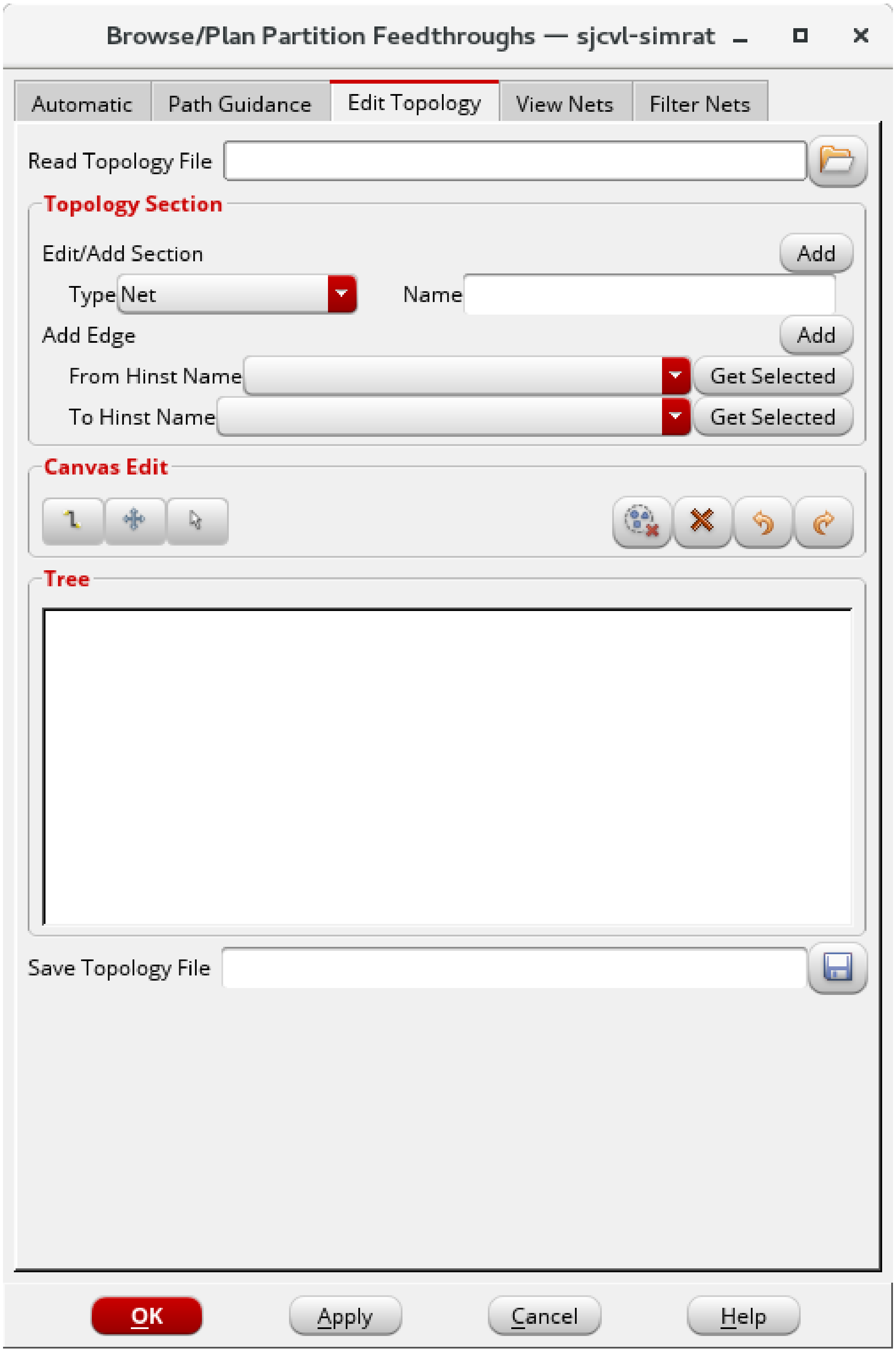
Read Path File generalized.path

☑ Show generalized paths

*: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets
#: path_nets

Creating a Topology File Using the GUI

Use the *Edit Topology* page of the *Browse/Plan Partition Feedthroughs* form to create a feedthrough path topology file.

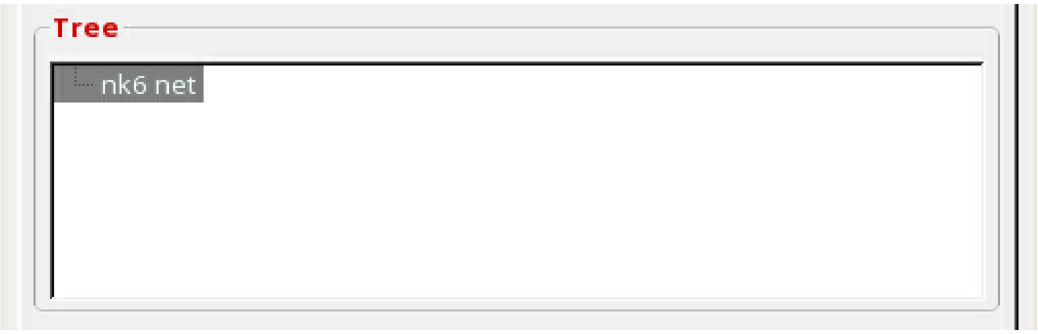


Type Net

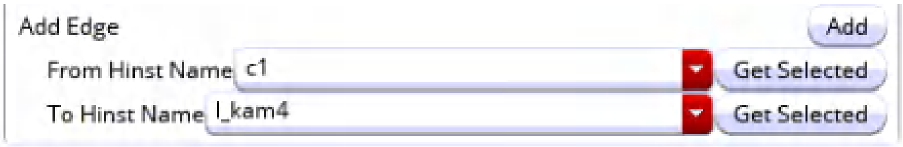
Note: For more information on the structure of the topology file, see Topology File Structure Guidelines.

Use Model:

1. In *Topology Section*, click Edit/*Add Section* and select the *Type* of entry. You can select between: Net, NetGroup, or Bus.
2. Enter the *Name* of the topology section entry and click the *Add* button. This name displays in the *Tree* area.



3. Add nodes to the selected topology section. You can add nodes using either the *Add Edge* area or the interactive *Canvas Edit* area.
   - Using the *Add Edge* area specify the start and end partitions.



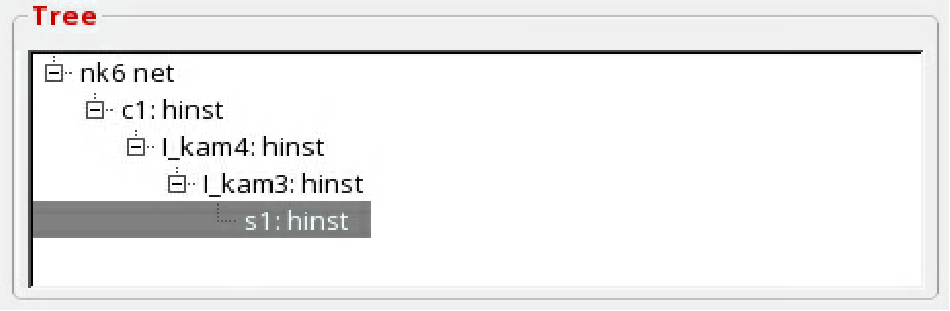
From Hinst Name: c1

To Hinst Name: I_kam4

   a. *From Hinst Name:* Specify the start partition. Alternatively, click *Get Selected* to control the list of partitions being populated by selecting the partitions in the GUI. By default, all partitions are populated in the list.

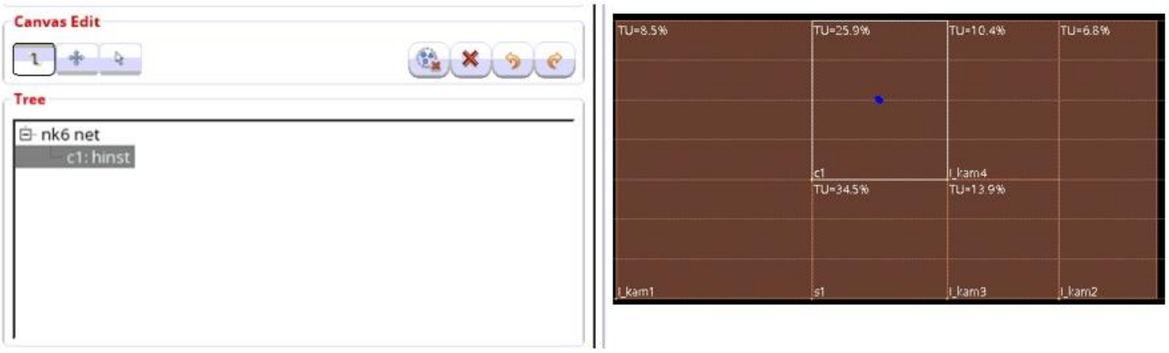
   b. *To Hinst Name:* Specify the end partition.

   c. Click the *Add* button to create a node for the object in the specified hierarchy. The new node is added to the bottom of the topology.

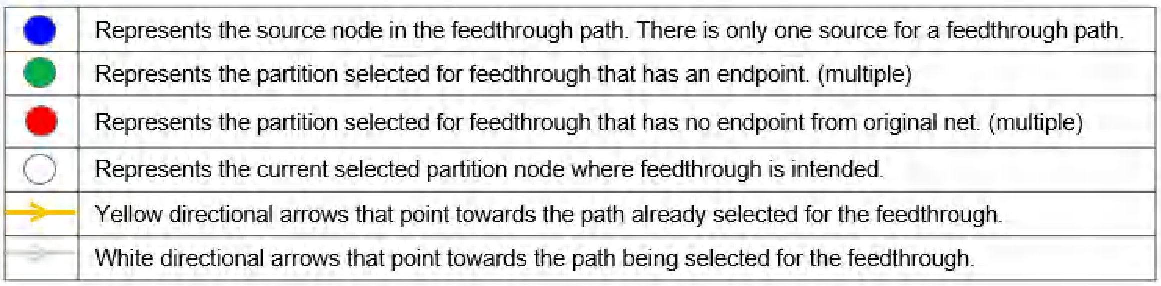


- Using the *Canvas Edit* area:

  a. Click the button in the *Canvas Edit* section to create a node. A node (● blue color) appears on the partition in the Innovus GUI canvas where the net source is. This node is also added and selected in the “Tree” section of form. Partition of the selected node is also selected in the Innovus canvas.



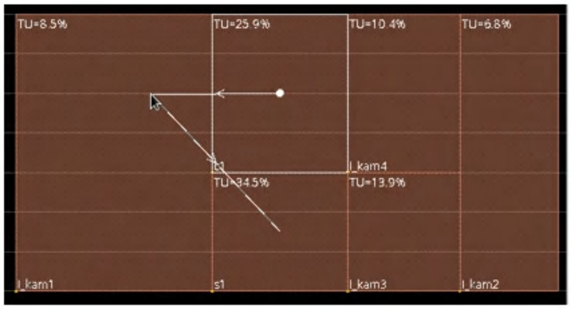
Note: The nodes and direction arrows of the feedthrough path follow a color-coded representation in the GUI for easy identification.

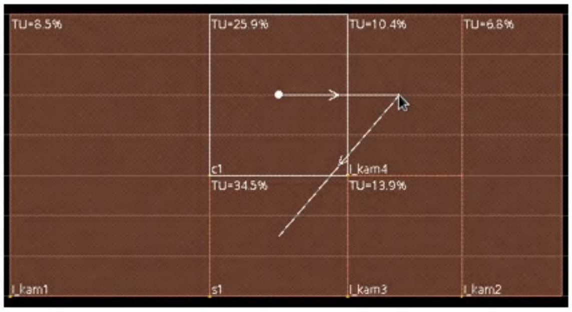


🔵	Represents the source node in the feedthrough path. There is only one source for a feedthrough path.
🟢	Represents the partition selected for feedthrough that has an endpoint. (multiple)
🔴	Represents the partition selected for feedthrough that has no endpoint from original net. (multiple)
○	Represents the current selected partition node where feedthrough is intended.
→	Yellow directional arrows that point towards the path already selected for the feedthrough.
→	White directional arrows that point towards the path being selected for the feedthrough.

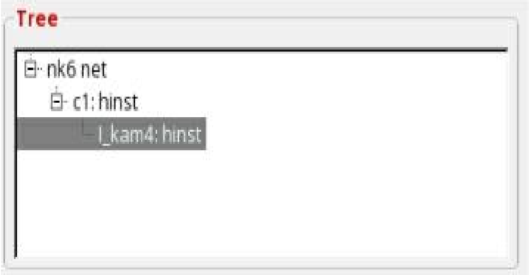
Note: The dots for the nodes are always placed at the center of the partition, irrespective of where you click on the partition.

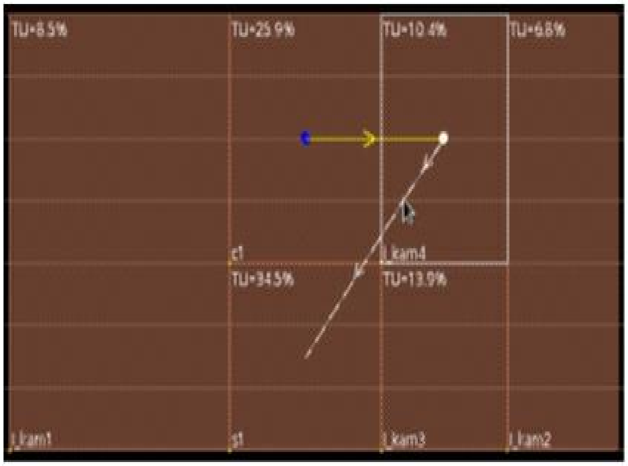
When the cursor is moved in the Innovus canvas, the node changes the color from blue to white. This signifies that the node is selected. Directional arrows showing all the possible endpoints of the net also appear. These arrows follow the cursor. **Note:** For a feedthrough path to be complete, the created path should include all these endpoints.





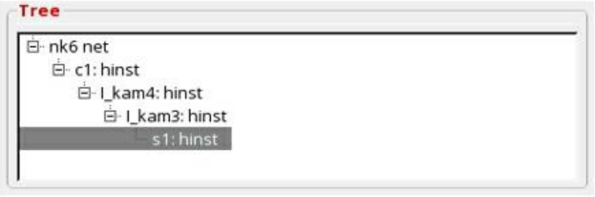
b. Click on any partition to create a feedthrough node. The new feedthrough node added to the path is now displayed in white color, indicating it is now the current selected node. Its partition gets selected in the Innovus canvas. This node is also added in the *Tree* area.
Note: The previous node now takes the original blue color that represents a source node of the net. The directional arrow from the previous node (blue, source node) to the current selected node, changes to yellow color indicating that the path has been already selected for the feedthrough. The white directional arrows now start from the current selected node (white) to the remaining endpoints.





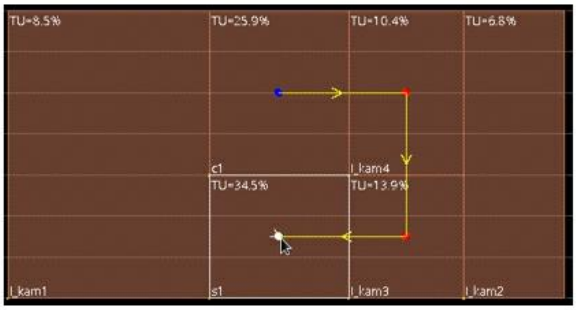
Subsequently, click on partitions to add feedthrough nodes and reach an endpoint. The previous node always changes to a different color and the current node becomes white. The nodes that are part of the feedthrough path are displayed in the *Tree* area.
In the following image, node I_kam4 (●red in color) represents a partition selected for feedthrough that has no endpoint from the original net.

Note: A feedthrough path can have multiple sink nodes.



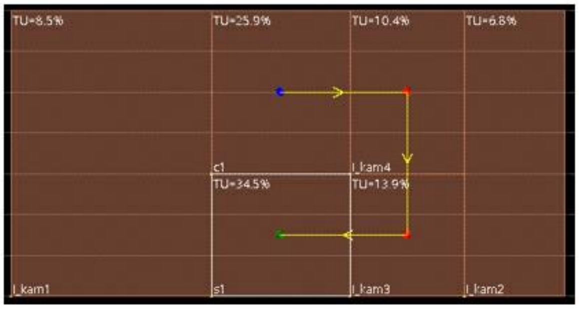
Tree

- nk6 net
  - c1: hinst
    - I_kam4: hinst
      - I_kam3: hinst
        - s1: hinst



Note: You can click the *Undo* button to undo the last operation. Node tree is also updated in *Tree* section of form. Selected partition is also updated in Innovus canvas. This is a sequential operation. Alternatively, you can use the *Redo* button to redo the last undo operation.

c. To complete the feedthrough path double-click on the current node or press the Esc key.The white current node takes the color of the node it represents. In the following image, node s1 (🟩green in color) represents a sink node partition selected for feedthrough that has an endpoint.



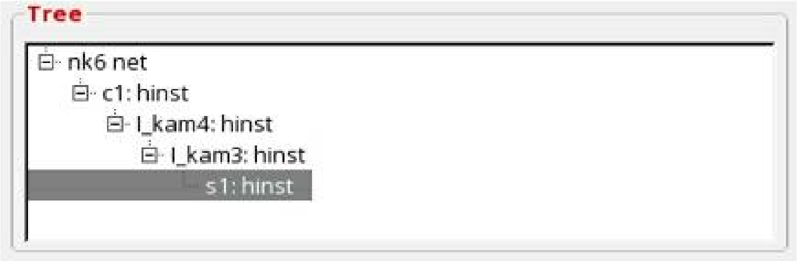
Note: Once complete feedthrough path, you can continue to add nodes to the tree structure. If required, you can split the nodes of the topology tree to branch out the topology tree structure.

d. Finally, click the button to edit the Canvas Edit mode. The Feedthrough Browser stops the suggestive directional white arrows.

4. Click *Write* and then specify the name of the topology file to *Save.* Click the Save button to save the file.

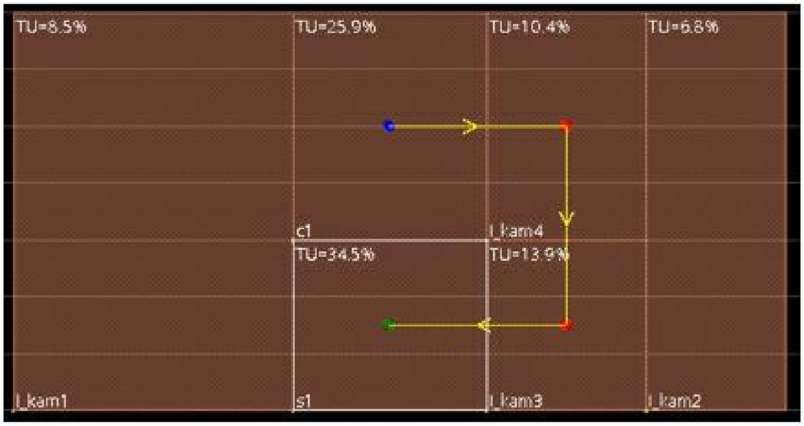
Example:

The following image shows the details of a topology file created using the *Browse/Plan Partition Feedthroughs* form.



Tree

```
nk6 net
    c1: hinst
        I_kam4: hinst
            I_kam3: hinst
                s1: hinst
```



Sample Topology File:

```
version 1.0;
net nk6
    hinst-hinst c1 I_kam4
    hinst-hinst I_kam4 I_kam3
    hinst-hinst I_kam3 s1
end net
```

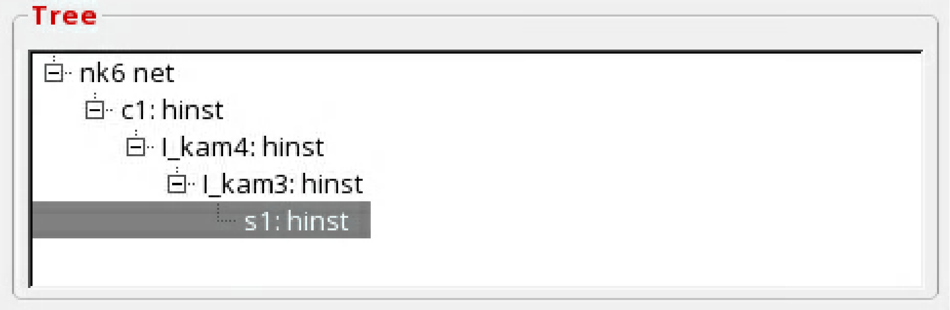
Editing a Topology File Using the GUI

Use the *Edit Topology* page of the *Browse/Plan Partition Feedthroughs* form to edit a feedthrough path topology file.

Note: For more information on the structure of the topology file, see Topology File Structure Guidelines.

Use Model:

1. Click Open (v.1) and then specify the name of the topology file to *Load*. Click *Apply* to load the file and populate the *Tree* area with the topology information.



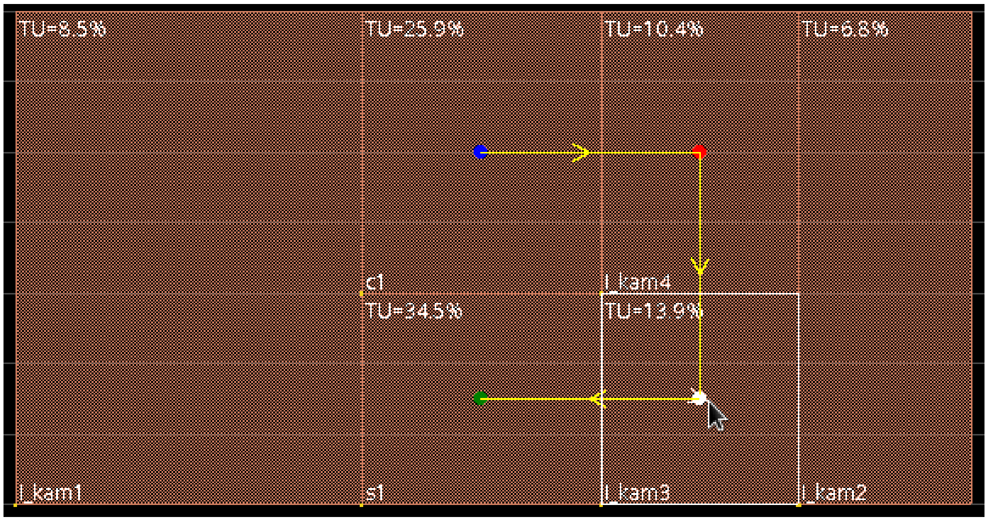
Tree

```
nk6 net
    c1: hinst
        I_kam4: hinst
            I_kam3: hinst
                s1: hinst
```

2. Add or delete topology sections or edges and then click *Apply* to make changes to the structure of the topology file.

- To add a section node in the tree area and click *Add* in the *Topology Section* area*. S*elect the *Type* of entry (Net, NetGroup, or Bus).
- To delete an entry from the *Tree* area, select the node in the Tree area and click the ✖ (*Delete selected node/section in the topology tree*) button in the *Canvas Edit* area. You can also click *Delete All* in the *Topology Selection* area to delete all entries from the topology tree.
- To delete an entry from the canvas GUI, enable the *Canvas Edit* mode, and then select a node (white color) and delete it.

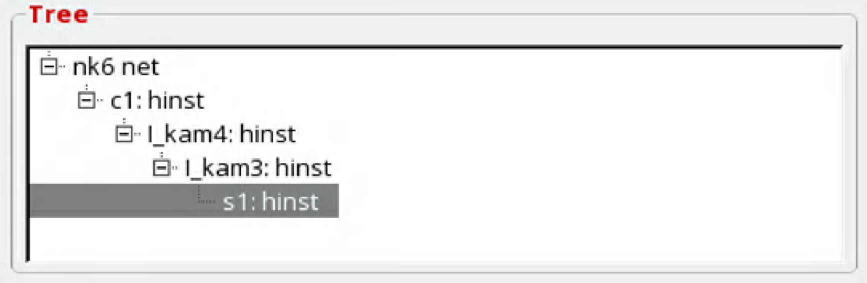
3. Add or delete topology edges and then to click *Apply* to make changes to the structure of the topology file.
    - To add a topology edge node, select the Topology Section node in the tree area. Click *Add* in the *Topology Edge* area and then specify the start and end partitions using *From Hinst Name* and *To Hinst Name* respectively*.*
    - To delete a topology edge node, select an edge node in the tree area and then click ✖ (*Delete selected node/section in the topology tree*) in the *Canvas Edit* area.
    - To delete an entry from the canvas GUI, enable the *Canvas Edit* mode, and then select a node (white color) and delete it.



4. Click *Save* and then specify the name of the topology file to *Save.* Click *Apply* to save the file.

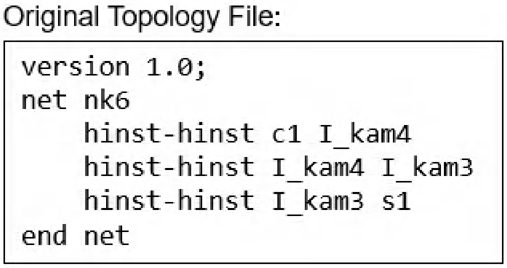
Example:

The following image shows the details of a topology file edited using the *Browse/Plan Partition Feedthroughs* form.



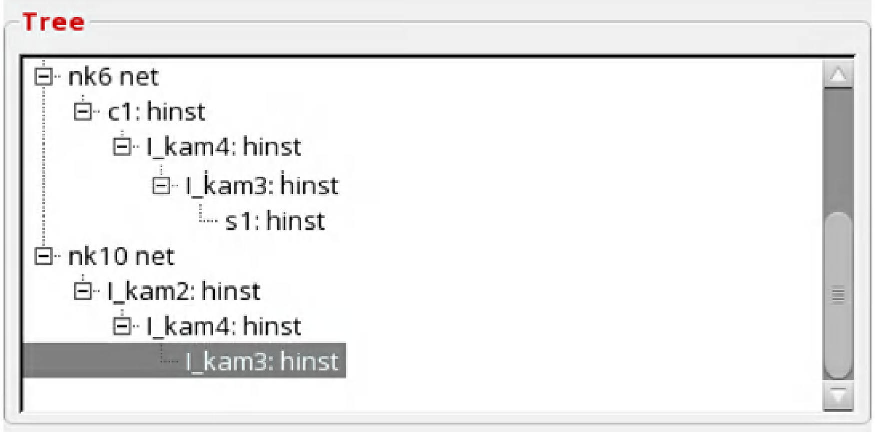
Tree

```
nk6 net
  c1: hinst
    I_kam4: hinst
      I_kam3: hinst
        s1: hinst
```



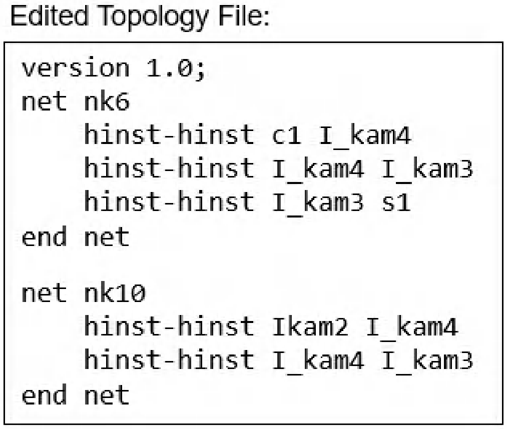
Original Topology File:

```
version 1.0;
net nk6
    hinst-hinst c1 I_kam4
    hinst-hinst I_kam4 I_kam3
    hinst-hinst I_kam3 s1
end net
```



Tree

```
nk6 net
  c1: hinst
    I_kam4: hinst
      I_kam3: hinst
        s1: hinst
nk10 net
  I_kam2: hinst
    I_kam4: hinst
      I_kam3: hinst
```



Edited Topology File:

```
version 1.0;
net nk6
    hinst-hinst c1 I_kam4
    hinst-hinst I_kam4 I_kam3
    hinst-hinst I_kam3 s1
end net

net nk10
    hinst-hinst Ikam2 I_kam4
    hinst-hinst I_kam4 I_kam3
end net
```

Browsing Nets Using the GUI

Use the *View Nets* page of the *Browse/Plan Partition Feedthroughs* form to analyze the feedthrough path and to highlight all nets for which feedthrough buffers were inserted or were considered. The highlighted feedthrough path consists of the nets, the terms, and the instances.

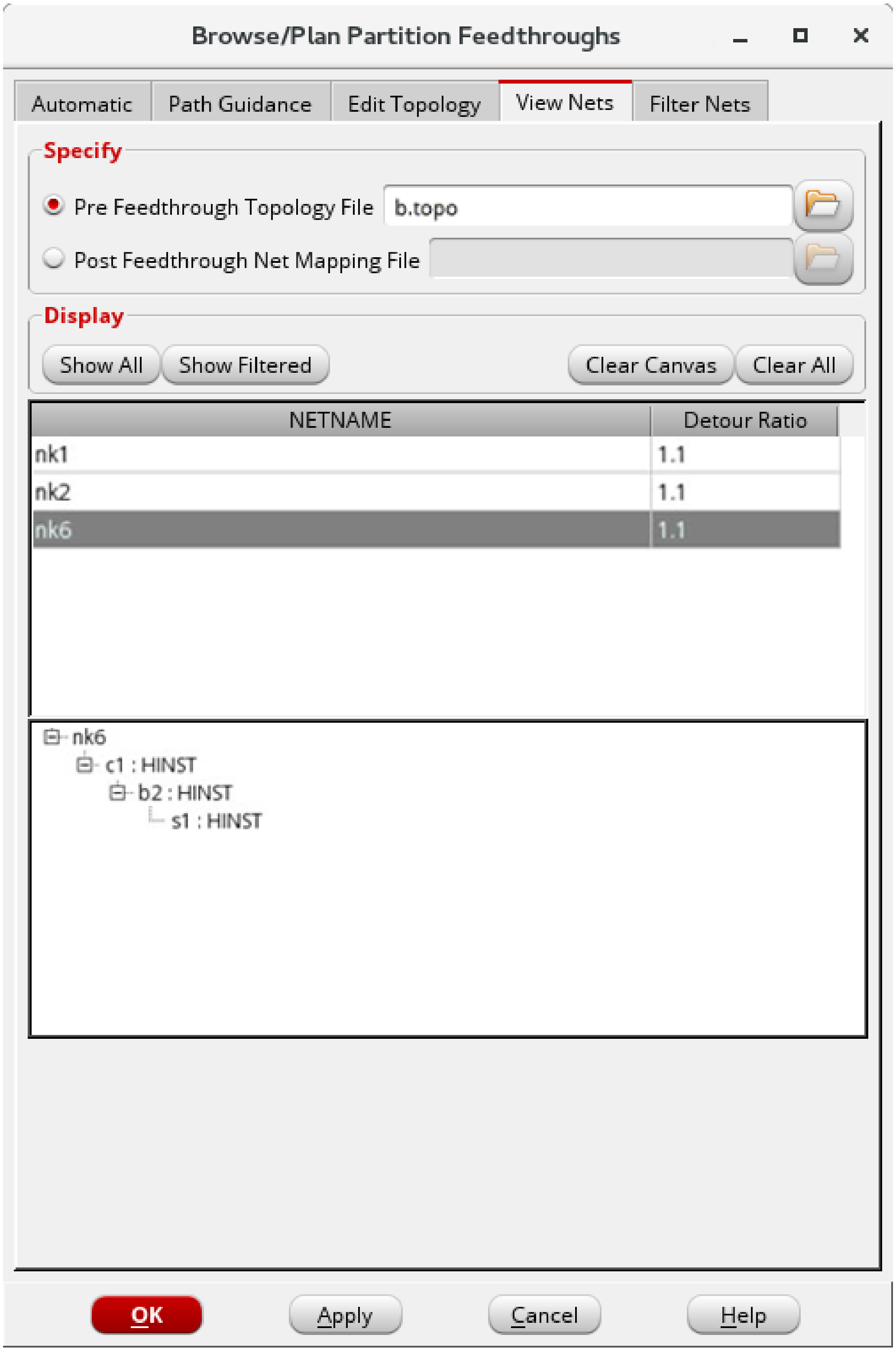
Debugging Nets for Feedthrough Path at Pre Feedthrough Stage

Use Model:

1. Load the pre-feedthrough topology file by selecting Pre Feedthrough Topology File and then specifying the name of the topology file.

   Note: If feedthrough was run using the *Automatic* Page of *Browse/Plan Partition Feedthroughs* form without mapping file, the topology file name is automatically populated.

   Note: Cadence recommends you to use the topology file for path analysis before inserting feedthroughs.



● Pre Feedthrough Topology File: b.topo

○ Post Feedthrough Net Mapping File:

NETNAME	Detour Ratio
nk1	1.1
nk2	1.1
nk6	1.1

```
⊟ nk6
    ⊟ c1 : HINST
        ⊟ b2 : HINST
            └ s1 : HINST
```

2. Click *Load* to populate the *Display* area with the net names and the detour ratios.

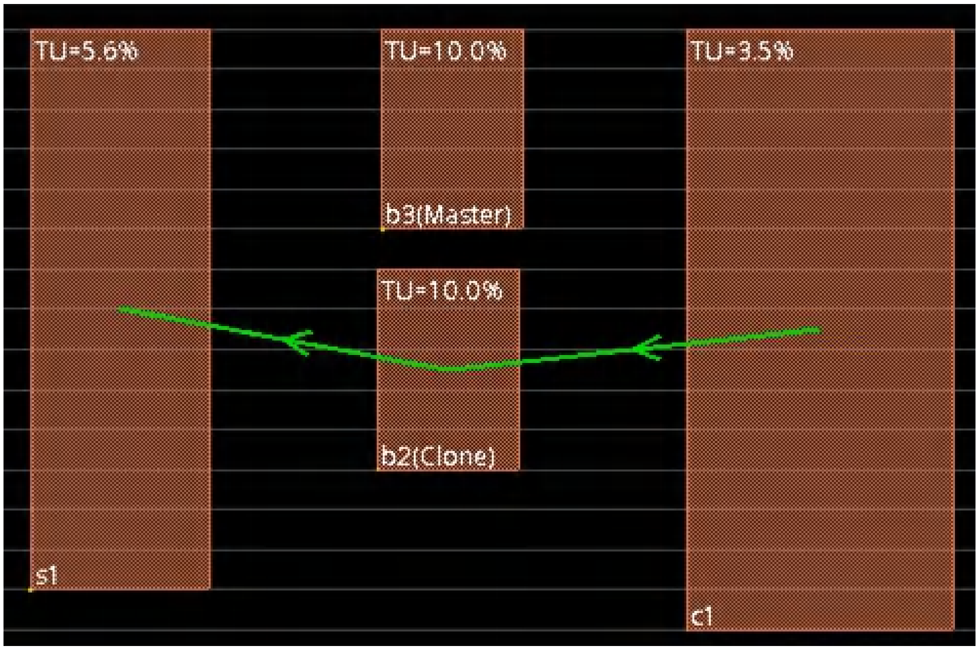
Tip: You can click the *NETNAME* header to sort the net names in ascending or descending alphabetical order. Alternatively, you can click the *Detour Ratio* header to sort the nets in ascending or descending order of their detour ratio.

3. Select a net to display its details and analyze its topology. A corresponding path is also drawn in the GUI.

   Tip: You can press the `shift` key and add more nets for analysis.

Example:

In the following image feedthrough path of net nk6 traverses through b2 (clone of cell b). When we load the topo file, cross probing is done for hinst to hinst (as is described in topology file).



Analyzing the Feedthrough Path After Committing Feedthrough

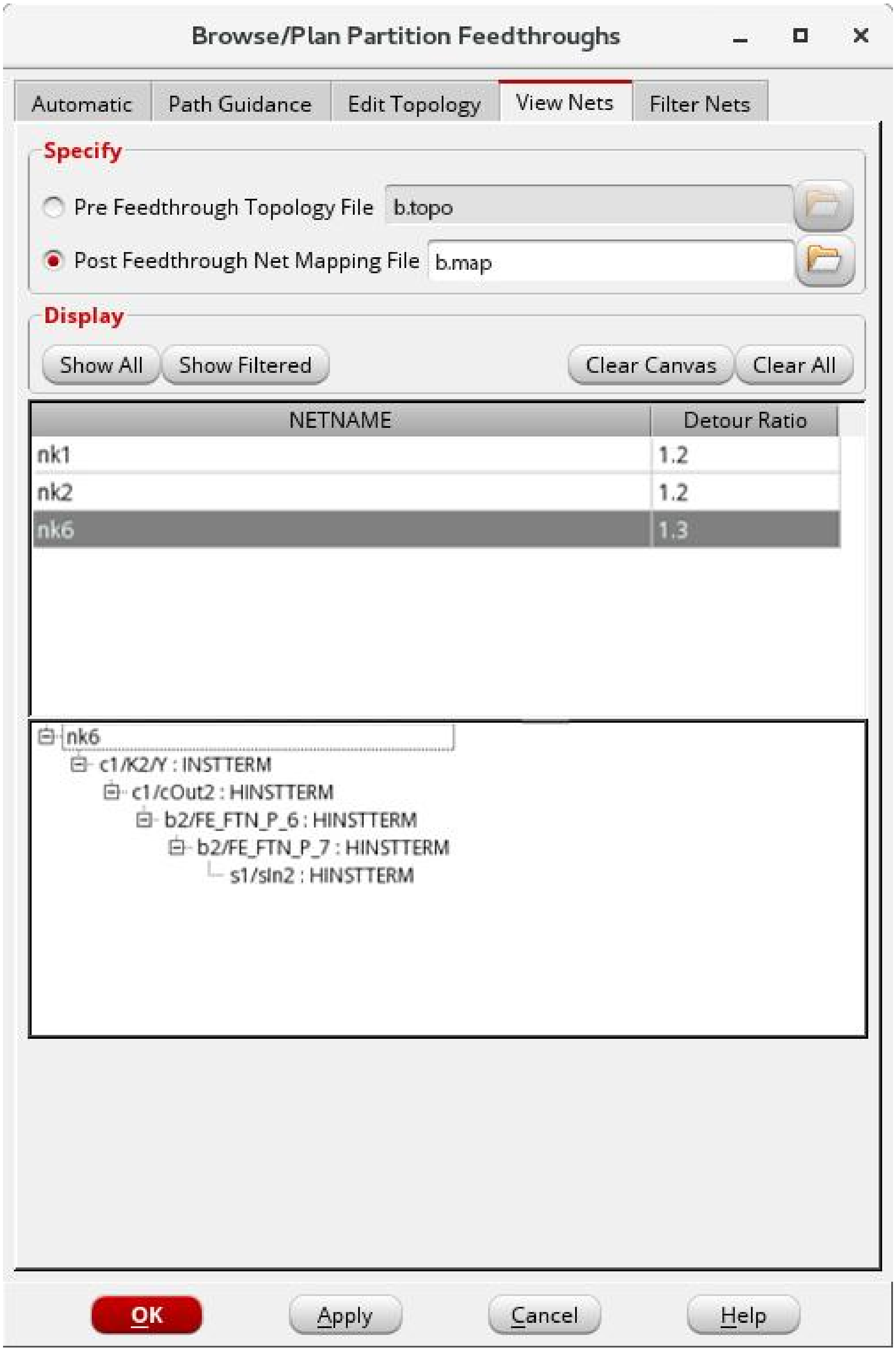
You can use the *View Nets* page of the *Browse/Plan Partition Feedthroughs* form to analyze the feedthrough path using the net mapping file after committing the feedthough.

Use Model:

1. Load the net mapping topology file by selecting *Post Feedthrough Net Mapping* and then specifying the name of the *Net Mapping File*.

   Note: If feedthrough was run using the *Automatic* Page of *Browse/Plan Partition Feedthroughs* form with mapping file, the net mapping file name is automatically populated.

   Note: Cadence recommends you to use the mapping file to analyze the path after feedthrough insertion.



○ Pre Feedthrough Topology File b.topo

● Post Feedthrough Net Mapping File b.map

NETNAME	Detour Ratio
nk1	1.2
nk2	1.2
nk6	1.3

```
nk6
  c1/K2/Y : INSTTERM
    c1/cOut2 : HINSTTERM
      b2/FE_FTN_P_6 : HINSTTERM
        b2/FE_FTN_P_7 : HINSTTERM
          s1/sIn2 : HINSTTERM
```

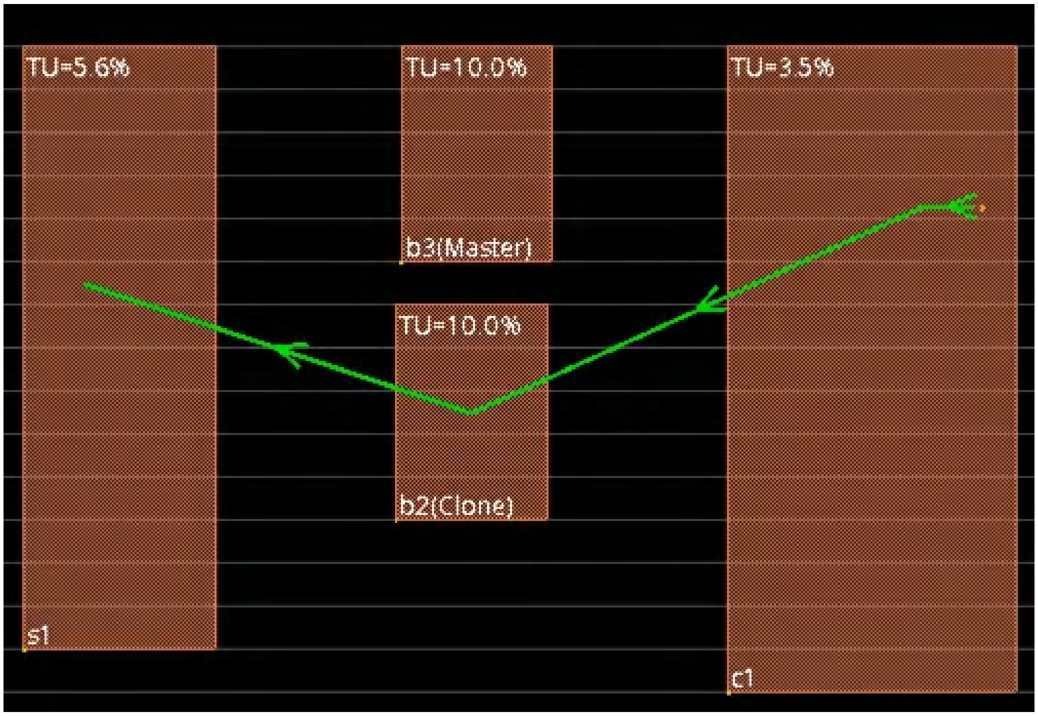
2. Click *Load* to populate the *Display* area with the net names and the detour ratios.

   Tip: You can click the *NETNAME* header to sort the net names in ascending or descending alphabetical order. Alternatively, you can click the *Detour Ratio* header to sort the nets in ascending or descending order of their detour ratio.

3. Select a net to display its details and analyze its topology. A corresponding path is also drawn in the GUI.

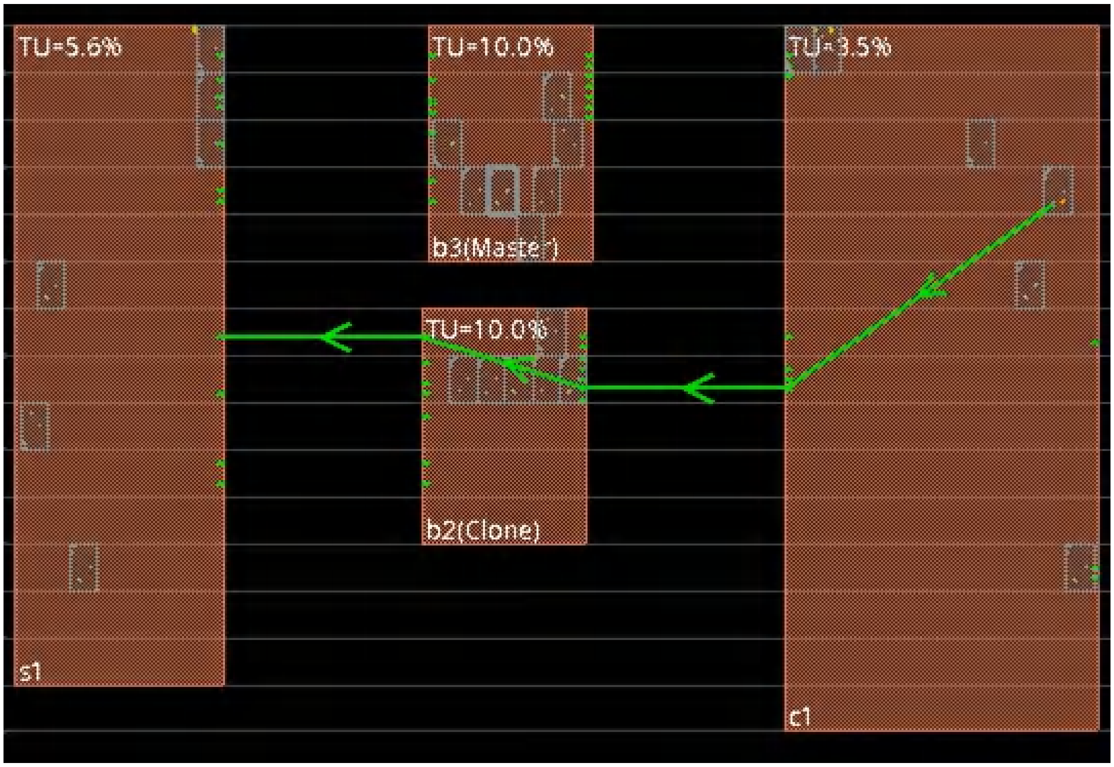
Example 1:

In the following image, feedthrough path of net nk6 traverses through b2 (clone of cell b). The path displayed is similar to the one displayed using the topology file, however with the use of mapping file additional details like start instterm and traversing intermediate hinst terms are displayed.



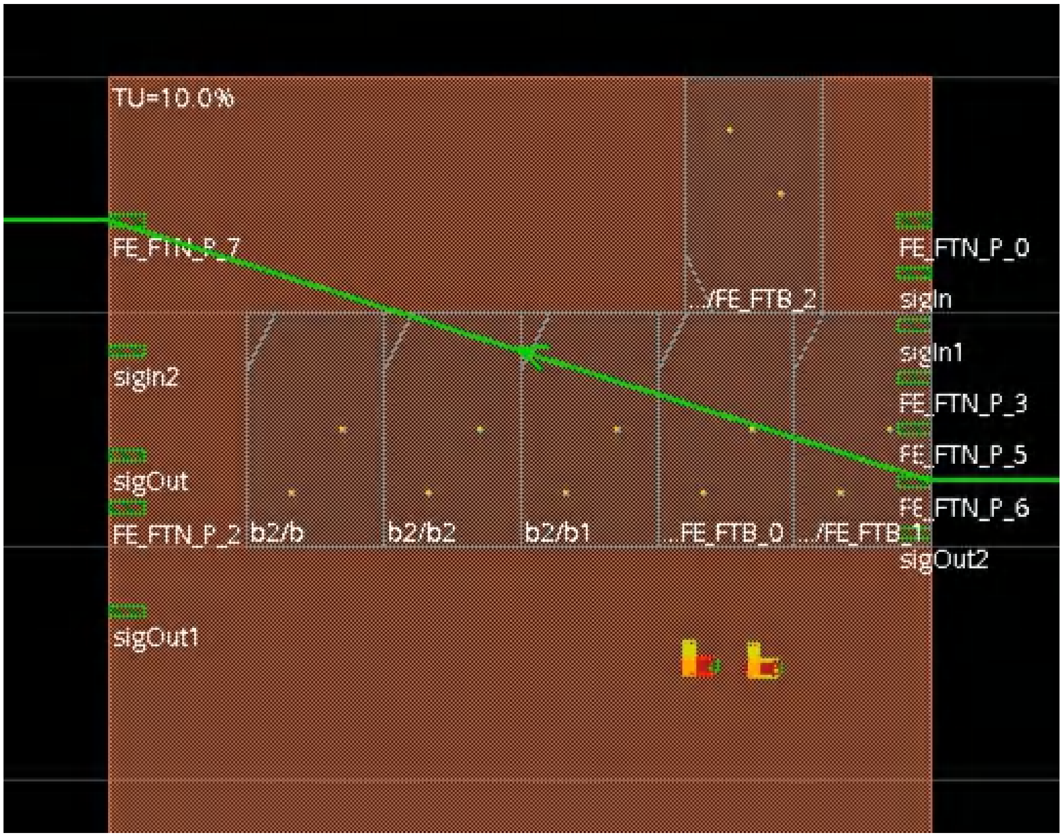
Example 2:

The following image displays the feedthrough path when the mapping file is loaded after doing the pin assignment. In this case the path traverses through actual pin locations.



Example 3:

Simply loading the mapping file after doing feedthrough insertion will display the feedthrough path like displayed with Example 1. However, if you assign partition pins and then want to see the path browsing through the partition pin location then you must load the mapping file and click the *Apply* button. The following image displays such details.



Filtering Nets Using the GUI

Use the *Filter Nets* page of the *Browse/Plan Partition Feedthroughs* form for applying filters on nets in order to choose target nets. It enables you to work on limited set of nets across different pages of the *Browse/Plan Partition Feedthroughs* form. For a description of all the fields, see the Partition Menu chapter in the *Menu Reference*.

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 Early Global Route.
   Info: At least one vertical and one horizontal routing layer must be available (that is, not blocked) on the macro(s). Otherwise, there will be no routing over the macro(s). In case the macro has all the layers blocked, manually remove the blockage over one horizontal and vertical layer.
5. Connect the built-in feedthroughs through the `connectMacroFeedthrough` command.
   Note: Before running detailed routing, take care of the unused feedthrough input pins that are left floating. For example, you might want to assign them to tie-high or tie-low. You can save the list of the unused ports with the `connectMacroFeedthrough -floatingPortList` command.

How the connectMacroFeedthrough Command Connects Feedthroughs

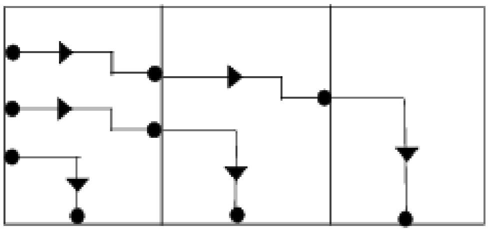
The following points illustrate the criteria for feedthrough selection and other important features of the `connectMacroFeedthrough` command:

- The `connectMacroFeedthrough` command considers all routing on all layers that cross the specified custom macro boundaries.
- The command searches for a feedthrough whose in and out pins lie *on the same sides* of the macro on which the wires enter and exit the macro.
- A feedthrough that has pins that are closer to the intersections has a higher probability of selection. Both input and output pins are considered. Layer information is ignored while evaluating the distance. To consider only pins within a specific distance from the wire crossing, use the `-maxSearchDistance` parameter.
- The command creates new nets and ports as required.

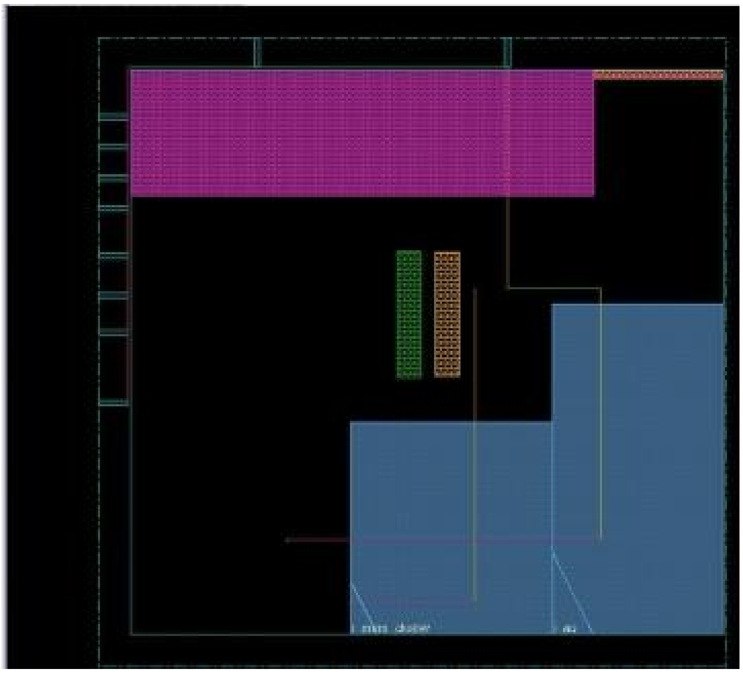
- If multiple feedthrough insertions are performed, the command keeps track of the feedthroughs already used, and does not assign such feedthroughs again.
- The new nets (the nets that connect to feedthrough output pins) have the following naming convention:
- `FE_FTM_x_` *`netName`*
  where `x` is a unique numeric identifier and *`netName`* is the name of the original net.
- You can select only specific nets for or exclude specific instances or nets. You can also specify the distance till which the command will search for a connected feedthrough. The feedthrough connectivity is described through a mapping file.

Feedthrough Connection for Abutted Macros

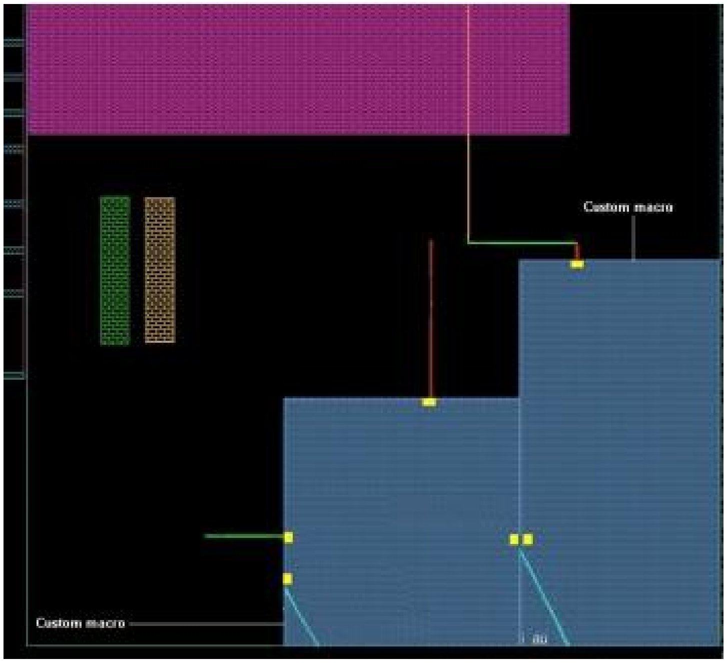
For abutted custom macros, the `connectMacroFeedthrough` command detects the paths formed by the abutted feedthrough pins. The Innovus software considers only the end points of the detected paths, and picks those feedthroughs that will give good results. The following figures show how Innovus selects the feedthroughs for insertion in the abutted custom macros.



The following figure shows pre-defined custom feedthroughs in the design.



The following figure shows how these feedthroughs are utilized by the `connectMacroFeedthrough` command. Notice the feedthrough pins, represented by yellow squares, that are added at the intersection of the macro boundary and the pre-defined nets.



Mapping File For Describing Feedthrough Connectivity

The feedthrough connectivity is defined through a mapping file that is provided as a parameter to the. If a mapping file is not specified with the `connectMacroFeedthrough` command, Innovus assumes that a file with the name `portmap` in the current directory is used by default.

The syntax of the file is as follows:

```
MACRO MacroName
  Macro definition section
END MACRO
```

The definition of the macro is provided in the *`Macro definition`* section, which can contain one or more feedthrough sections. The name of the feedthrough section is optional.

Note: The definitions for all custom macros to be used in the design should be in a single portmap file.

The syntax of the Feedthrough section is as follows. The name of the feedthrough is optional.

```
Feedthrough [ feedthroughName ]
Pin Section
END FEEDTHROUGH
```

Each Feedthrough section contains one section for the input pin and one section for the output pin.

Note: Multi-fanout feedthrough sections are not supported.

The syntax of the pin section is as follows:

```
PIN PinName
END PIN
```

Note: All the predefined macro feedthrough pins should be floating pins.

Here is an example of a mapping file:

```
MACRO RAMXXX
FEEDTHROUGH feedthrough1
PIN feedthrough1_in
END PIN
PIN feedthrough1_out
END PIN
END FEEDTHROUGH

FEEDTHROUGH feedthrough2
PIN feedthrough2_in
END PIN
PIN feedthrough2_out;
END PIN
END FEEDTHROUGH
END MACRO
```

Limitations

The `connectMacroFeedthrough` command has the following limitations:

- Multi-fanout feedthroughs are not supported.
- Routing blockage and congestion are not considered. However, because topology is derived from routing, this should not be a concern.
- Bidirectional pins (INOUT) are not supported.
- The topology is derived from the routing results. Therefore, you might need to specify certain Early Global 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.

Generating the Wire Crossing Report

You can display and write a file of wires that physically cross over partitions using the `showPtnWireX` text command or the *Partition - Show Wire Crossing* menu command. The results are saved to a *`designName`* `.wirecrossing` file that reports nets that cross each partition in a design. For any net that crosses more than one partition, you can use it as a starting point for generating a list of nets for feedthrough insertion.
Tip: Edit the *designName* .wirecrossing file to exclude high fanout nets, clock nets, and nets that are connected to two or more glue logic standard cells to avoid timing and routing problems on these nets. You can use the resulting file with the `insertPtnFeedthroughs` text command's `-selectNet` option. Note that the Innovus software determines the buffer tree topology, so not all specified nets will receive inserted feedthroughs. For example, nets that connect directly between adjacent partitions are not candidates for feedthrough insertion.
In designs with master clone partitions, the `showPtnWireX` command creates three additional output files. These files can be used by the `insertPtnFeedthrough -selectNet` *`filename`* command for path based feedthrough insertions in master and clone designs.

For example, when you specify,

```
showPtnWireX -outFile abc
```

The following files are generated:

- *abc.mc_connected_nets.txt*: For all nets connected to master/clone partitions.
- *abc.mc_crossing_nets.txt*: For all nets with wires over master/clone partitions (but not in the above file) and not intra partition nets.
- *abc.non_mc.txt*: For all remaining nets which are not in above files and are not intra partition nets.

The default outfile file, *abc.wirecrossing,* is only valid for non master/clone designs and has no significance for master and clone designs. The *abc.wireCrossing* file does not include wire crossings over clone partitions.

These files contain just the net names. Their syntax is as follows:

```
NetA
NetB  NetC NetD ...
```

To do path based feedthrough insertion for master clones nets, do the following:

- For `abc.mc_connected_nets.txt` and `abc.mc_crossing_nets.txt` files, use the following command without using the `-routeBased` parameter:

  `insertPtnFeedthrough -selectNet abc.mc_connected_nets.txt` or,

  `insertPtnFeedthrough -selectNet abc.mc_crossing_nets.txt`

- For `abc.non_mc.txt` file, use the following command with or without using the `-routeBased` parameter:

  `insertPtnFeedthrough -selectNet  abc.non_mc.txt`

Interpreting the Wire Crossing Report

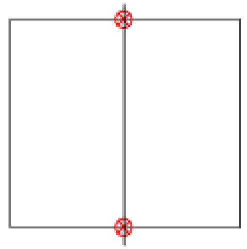
The wire crossing report section lists the nets, their wire lengths, in micrometers, and the shape of the wire in relation to the partition. For example, the following report segment is for a partition module named `ptn01`:

```
#########################################################
# Nets that cross partition module ptn01
# Box (335 335) (833 567)
# Format: Net <netName> <wireLength> <shape>
#########################################################
```

```
Net A 65 I
Net B 80 L
Net C 1050 T
Net D 132 X
...
```

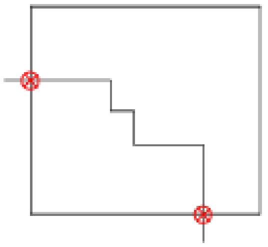
The first net in the report, `A`, has a wire length of 65 micrometers in an `` `I' `` shape, which indicates that the net crosses the partition on opposite sides, as follows:

```
Net A 65 I
```



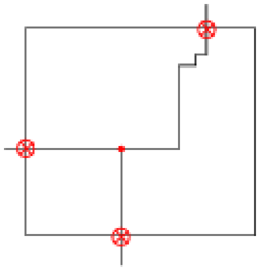
The second net in the report, `B`, has a wire length of 80 micrometers in an `` `L' `` shape, which indicates that the net crosses the partition on adjacent sides, as follows:

```
Net B 80 L
```



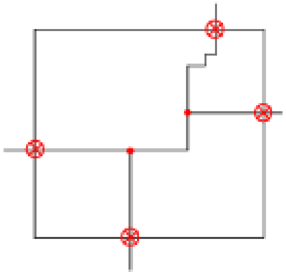
The third net in the report, `C`, has a wire length of `105` micrometers in an `` `T' `` shape, which indicates that the net crosses the partition on three sides, as follows:

```
Net C 105 T
```



The fourth net in the report, `D`, has a wire length of `132.30` micrometers in an `` `X' `` shape, which indicates that the net crosses the partition on all four sides, as follows:

```
Net D 132 X
```

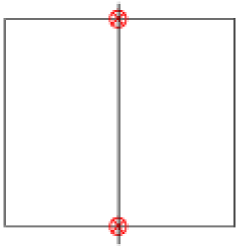


In the report, you can also include the total length of the wire crossing the block in the horizontal X direction and total length of the wire crossing the block in the vertical Y direction using the `-delta` option of the `showPtnWireX` command. For example, the following report segment is for the same partition module named `ptn01` using the `-delta` option:

```
#################################################################
# Nets that cross partition module ptn01
# Box (335 335) (833 567)
# Format: Net <netName> <wireLength> <shape> <deltaX> <deltaY>
#################################################################

Net A 65 I 0 65
Net B 80 L 38 47
...
```

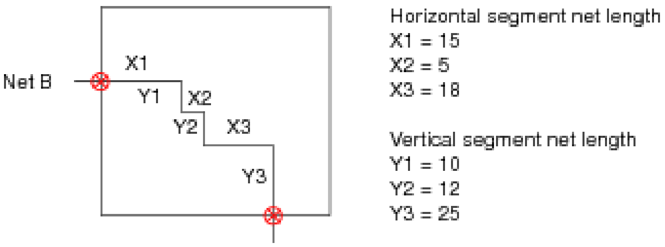
The first net in the report, `A`, has a wire length of `65` micrometers in an `` `I' `` shape, with a total of 0 length in the horizontal X direction, and 65 in the vertical Y direction:

```
Net A 65 I 0 65
```



The second net in the report, `B`, has a wire length of `80` micrometers in an `` `L' `` shape, with a total of 38 length in the horizontal X direction, and 47 in the vertical Y direction:

```
Net B 80 L 38 47
```



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

Estimating the Routing Channel Width

For committed partitions and blackboxes with assigned pins, channel width estimation uses the current pin assignment. If partition pins are not assigned, they are placed at the lower-left corner. In this case, the Innovus software issues a warning message because the estimator cannot produce a good result.

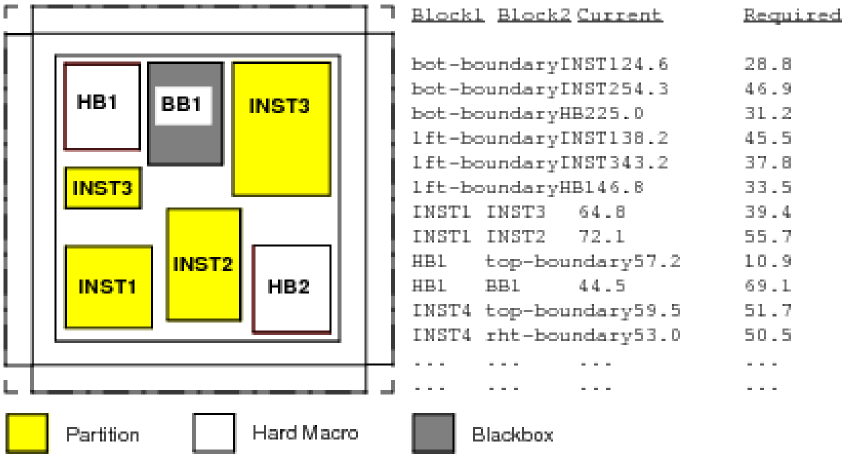
For uncommitted partitions, channel width estimation runs the Partition program, assigns pins, estimates the channel widths, and runs the Unpartition program. For blackboxes without assigned pins, it assigns pins and estimates the channel widths.

The 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 `earlyGlobalRoute`), and block halo. The channel width estimator also respects these constraints so that their top-level block floorplans are not dramatically changed. If there is conflict between a specified constraint and the minimum required channel spacing, the Innovus software honors the minimum required channel spacing.

This feature produces a report containing the following information:

- Estimated required spacing, in micrometers, between partitions, blackboxes, and hard macros.
- Estimated required spacing surrounding each partition based on its pins (the relative distance between partition blocks required for top-level routing).
- Estimated distance between blocks and core boundaries (top, bottom, left, right).

The following figure shows an example of how the channel estimation report relates to the design:



```
Block1 Block2 Current        Required

bot-boundaryINST124.6        28.8
bot-boundaryINST254.3        46.9
bot-boundaryHB225.0          31.2
lft-boundaryINST138.2        45.5
lft-boundaryINST343.2        37.8
lft-boundaryHB146.8          33.5
INST1  INST3   64.8          39.4
INST1  INST2   72.1          55.7
HB1    top-boundary57.2      10.9
HB1    BB1     44.5          69.1
INST4  top-boundary59.5      51.7
INST4  rht-boundary53.0      50.5
...    ...     ...           ...
...    ...     ...           ...
```

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 Early Global Route. When the Partition program is run, the Early Global 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 Early Global 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.

Notes:

- 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.
- Running the Partition program creates a blockage on an OVERLAP layer even though the OVERLAP layer is not defined in the technology section of the LEF file. As a result, the partition LEF file cannot be loaded into either the Innovus software or any standalone tools. If your design has rectilinear partitions or feedthroughs, the OVERLAP layer must be defined in the technology section of the LEF file.
- 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.

You can save the partition data in an OpenAccess database.

Creating a Top-Level Partition

1. Run the Partition program.
2. Run Early Global Route on the top-level partition.
3. Check for routing congestion. If there is no congestion, you are done. If there is congestion, continue to step 4.
4. Run the Unpartition program and add more routing resources to the congested area.
5. Rerun the Partition program.

Repeat steps 1 - 5 until there is no routing congestion.

Block-Level Partition

To create a block-level partition, complete the following steps:

1. Run the Partition program.
2. Check to see if each partition size is suitable. If it is, you are done. If it is not, continue to step 3:
3. Run the Unpartition program.
4. Increase the size of the block.
5. Rerun the Partition program.

Continue with the steps above until you have reached suitable partition sizes.

Pushing Down Signal Routes

During partition program, you can use the `-pushRoute` parameter of the `partition` command to push down signal routes to the respective partitions.
Info: Before running the partition `-pushRoute` command, you can check the hierarchy violations for nets on the partitions with the `checkHierRoute` command.

Here's the pushdown behavior with the `-pushRoute` parameter of the `partition` command:

- The following routes are pushed down:
  - Intra-partition nets routed completely within the routed boundary.
  - Inter-partition nets that cross the partition boundary only once *and* that pass through the partition pin location.
- Top nets that are routed completely in the top channels are retained at the top
- All other nets are deleted.

Note: All the wires of a hierarchy-violating net are discarded, in case even a single wire of that net has a hierarchy violation.

How Top-level Stripes Are Pushed Down

When you use the `partition` command, it retains the stripes that are not on a layer reserved by the partition at the top level and also copies them 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.

Stripe Position	How Stripes Are Pushed Down

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

How Bumps, Routes, and Area I/O Cells Are Affected

This section illustrates how bumps and routes are handled when the design uses hierarchical partitioning with flip chip RDL routing and 45-degree routes. This information pertains to the partition command. 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 in this section:

- Area I/O Cells are Part of the Top-level Netlist
- Area I/O Cells are Part of the Partition Netlist

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.

The following sections discuss the behavior for the following cases:

- Bumps and Routing are on Top Routing Layer
- Bumps and Routing are on Reserved Routing Layer

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" section.

Bumps and Routing are on Top Routing Layer

The following table summarizes the behavior when the bumps and the routing are on the top routing layer and you run the `partition` command.

Object Type	**Top Level**	**Partition Level**
Area I/O cell	An pin equivalent pin to the area I/O pin is created in the partition LEF file. This pin has the same size, location, and metal layer as the area I/O pin.	Area I/O cell is retained in the partition netlist
Signal bump	Signal bump stays on top and, additionally, an equivalent pin is created in the partition LEF file.	• If the bump overlaps fully or partially with the partition, and connects to the partition: An equivalent pin for the signal bump is created in the partition LEF file. This pin has the same size, location, and metal layer as the bump. • If the bump overlaps with the partition but is *not* connected to the partition: The signal bump is pushed down as a routing blockage.
Signal Routes	Signal Routes routed on the top routing layers stays at top.	• If the signal route overlaps the partition and is also connected to a area I/O cell inside the overlapping partition and a signal bump at the top, the signal route is copied and pasted to the partition. The pushed down net will be the internal net in the partition and will be named based on the partition port it is connected to inside the partition. • If the signal route overlaps the partition to which it is not connected (that is, it is not connected to any instance inside the partition but to a bump at top), these routes are copied and pasted as routing blockages inside the overlapping partition.

Bumps and Routing are on Reserved Routing Layer

The following table summarizes the behavior when the bumps and the routing are on the reserved routing layer and you run the `partition` command.

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

Limitations

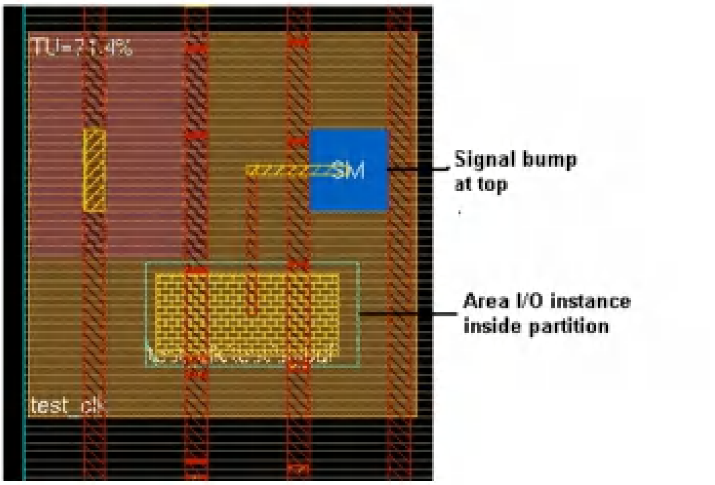
- The pushdown of the signal bumps as an equivalent pin inside the partition is not supported for the non-rectangular shapes of the bump cell.
- If the pushed down area I/O cell has pin shapes on the top routing layers, the blockages created on the top routing layers are not cut against these component 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:

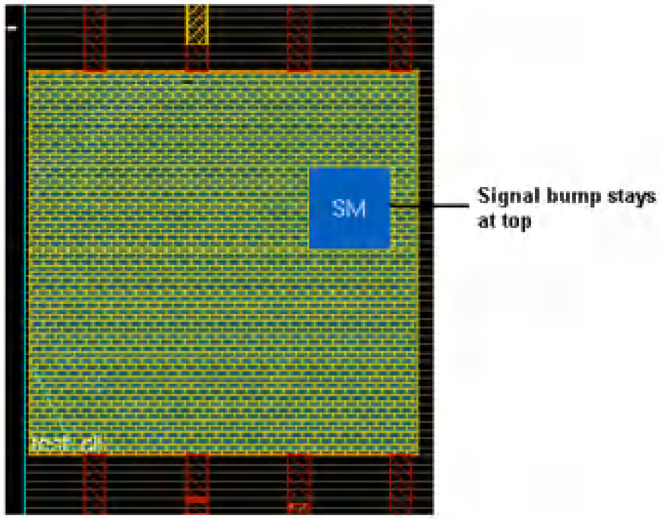
- Case 1: All Routing Layers Reserved for the Partition
- Case 2: Top Layer Not Reserved for Routing

Case 1: All Routing Layers Reserved for the Partition

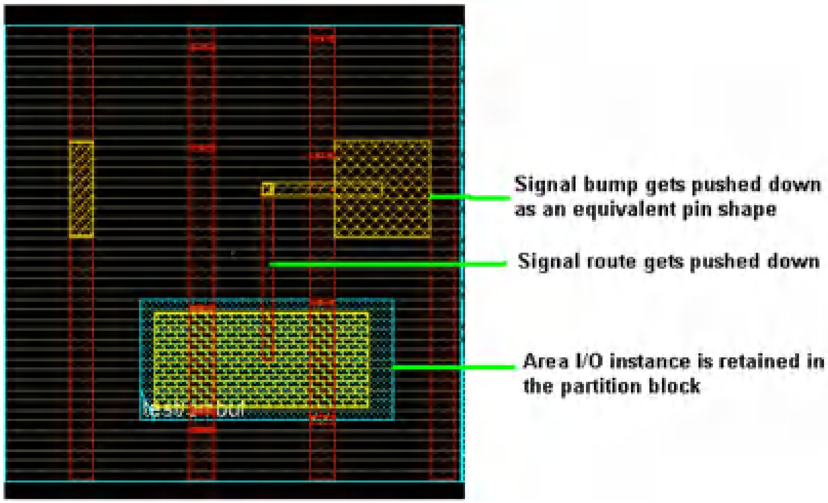
The design has six routing layers. All the layers are reserved for the partition. Signal Bump SM is connected to area I/O cell inside the partition. The following diagram shows the floorplan view before partitioning.



The following figure shows the view at top after partitioning.

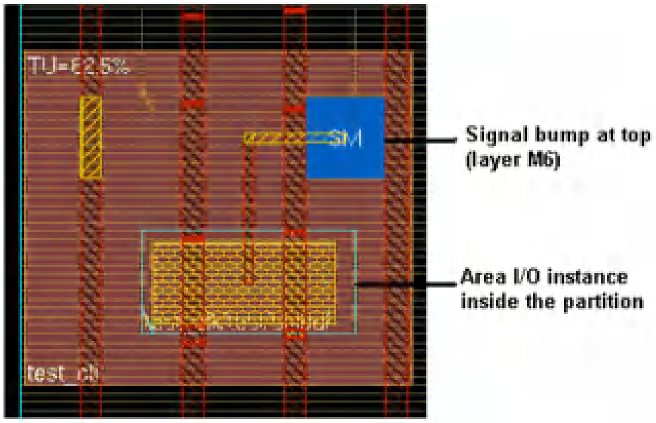


The following figure shows the view inside the partition


Signal bump gets pushed down as an equivalent pin shape
Signal route gets pushed down
Area I/O instance is retained in the partition block

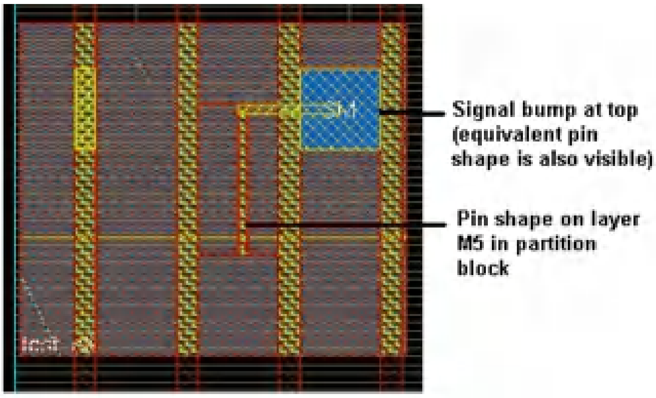
Case 2: Top Layer Not Reserved for Routing

The design has six routing layers. Layers Metal1-Metal5 are reserved for the partition. Metal6 is the top routing layer. The following diagram shows the floorplan view before partitioning.

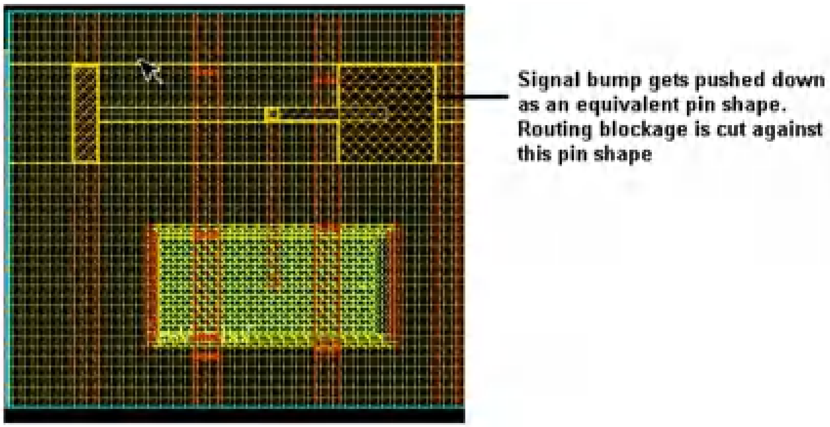

TU=62.5%
SM
test_ct
Signal bump at top (layer M6)
Area I/O instance inside the partition

The following diagram shows the view at the top after partitioning.

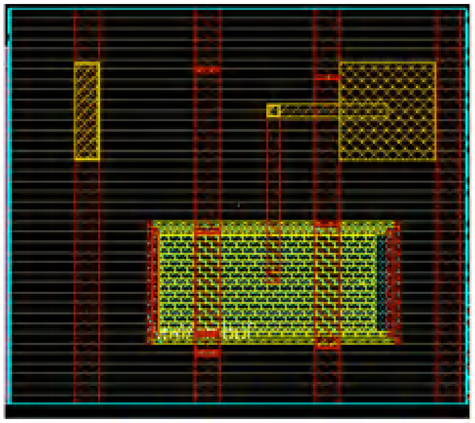
The following figure shows the view on the top after partitioning with the pins visible.


SM
Signal bump at top (equivalent pin shape is also visible)
Pin shape on layer M5 in partition block

The following figure shows the view on the top after partitioning with visible routing blockages on layer Metal6.


Signal bump gets pushed down as an equivalent pin shape. Routing blockage is cut against this pin shape

The following figure shows the view inside partition with the display of the routing blockages turned off



Saving Partitions

You can save partition results, including the top-level partition, to their own subdirectories so that each partition can be worked on concurrently. Each partition directory contains all files necessary to run the Innovus software. Files necessary to run back-end tools are in the Innovus proprietary format. Def file can additionally be saved by using the -def option while saving partitions.

To save a partition, use the `savePartition` text command.
Note: For the iHDB flow, use the `savePartition -module_model_tag` text command.

> **Warning:** Cadence recommends you NOT to use the *Save Design* form to save a partition.

Working with OpenAccess Database

> **Note:** The iHDB flow does not support OpenAccess flow yet.

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.

- 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
```

Pushing Down a Network into Block Partitions

In a hierarchical chip design methodology, some networks (i.e. std cells and nets along with shape of wires connected to such nets) are designed in a highly symmetric manner to ensure similar latency and minimal skew across all its anchor buffers. This method of network tree construction does not consider the underlying floorplan. To the extent that such a tree – with its wires and anchor buffers – is completely out of tune with the complicity requirements of a hierarchical design methodology. Such un-hierarchical nature is often designed for, but not limited to clock, reset signals, along with high speed buses and other chip critical nets.

As a result, such networks need to be properly reallocated, as a separate step, to fit the nature of the hierarchical floor plan. In a hierarchical design methodology, every logic and wire must fit the nature of underlying partitions. To ensure this, such networks need to be “pushed down” logically and physically into the partitions they overlap with. In this way, they are available during partition implementation, and play their necessary timing closure and design closure role. For example, a clock network which is suitably pushed down into overlapping partitions, can serve well in accurately closing the partitioned block design timing and SI flows.

The idea behind “Push Down Network” is to modify the design – without losing its logical equivalence nature – in such a manner that it becomes amenable to be hierarchically partitioned into various sub-designs. This enables the continuance of hierarchical design intent.

Flow to Push Down a Network into Block Partitions

This flow will push down the logic physical data as-is to the overlapping partition fences. Below are the flow highlights:

- Before doing network push down, ensure that all partitions in the design should have proper non-overlapping fences. You must also make sure that the floorplan is clean and does not have any overlaps. Additionally, ensure that the nets to be pushed down are routed and the logic connected to the net is already placed.
- Any regular wires of the net to be pushed down are irreversibly converted to special wires.
- The pushed down logic maintains its DEF position (location and orientation).
- The pushed down wires maintain their DEF position.
- The (special) wires connected to the net are cut at the partition fence boundaries. Also, if the net has been pushed down to another (lower) level of verilog hierarchy, then the (special) wires are transferred to this new net.
- Some Verilog ports and corresponding partition pins are created in the partition modules as per the routing topology of the net.
- For every partition, the new Verilog ports are assigned. Pin shapes are created in the partition for every Verilog port created.
- New top level nets may be required to connect the Verilog modules
- The pushed down network capability does not create any Verilog assign statements in the top logic. If required, the assign statements are created only inside the partition module.
- The net is pushed down to the “lowest logical level”, as per the logic connected to the net.
- In abutted designs, the partition pins are created abutted to each other at the abutted boundary.
- Master and clone designs are not supported. During network push down of a design with master and clone partitions, only the master partitions are considered for push down. Innovus discards clone partitions in such designs.
- The pushed down network capability handles wide-wires.

Flow Overview

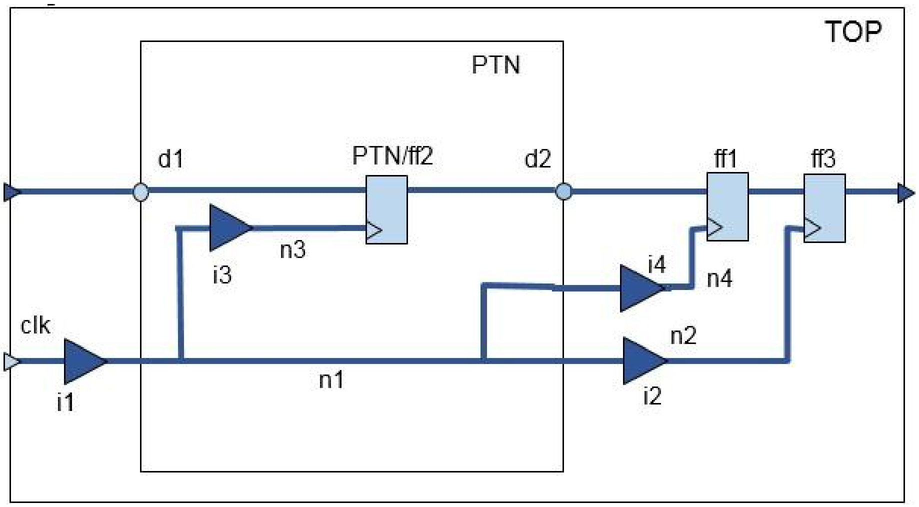
The following are the flow overview steps:

1. Load the design with partitions defined and CCT clock pre-routed and pre-placed
2. Push down the CCT clock tree as it is into the partitions it's overlapping to.
   a. `push_ptn_network $clock_nets_to_pushdown`
   b. Creation of assignment statements.
   c. Creation of partition pins
   d. The list of nets to be pushed down has to the provided
3. `deriveTimingBudget`
4. Turn partitions into blocks
   `partition` -pushDownNonPGSpecialNet
5. Save partition data.
   For example:
   `savePartition` –def –dir PTN
   For iHDB, `savePartition -def -module_model_tag PTN`

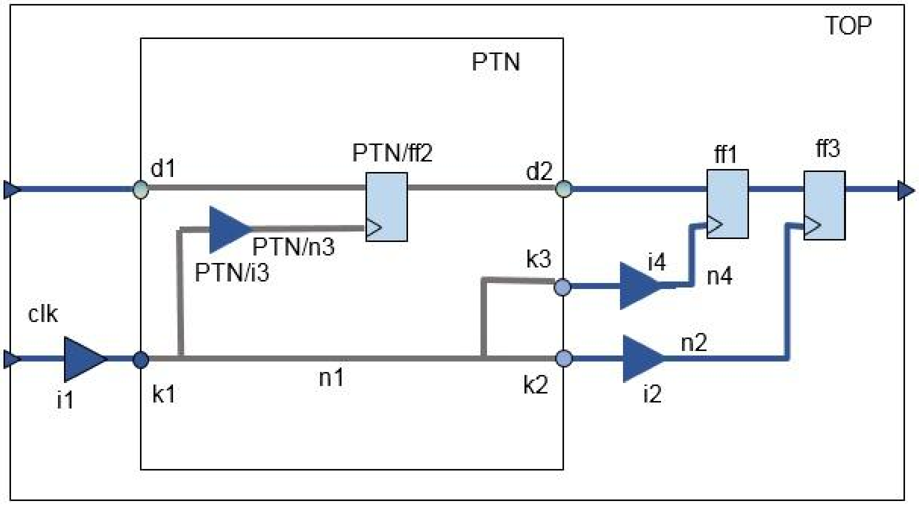
Example

In the design below:

- The clock network (std cell and wires) is designed in a flat manner, without consideration of underlying partitions.
- The clock buffer i3 is logically outside, but physically inside the partition PTN.
- The wire n1 is crossing PTN boundary multiple times, where as there is no verilog port of PTN which logically connects to n1 net.

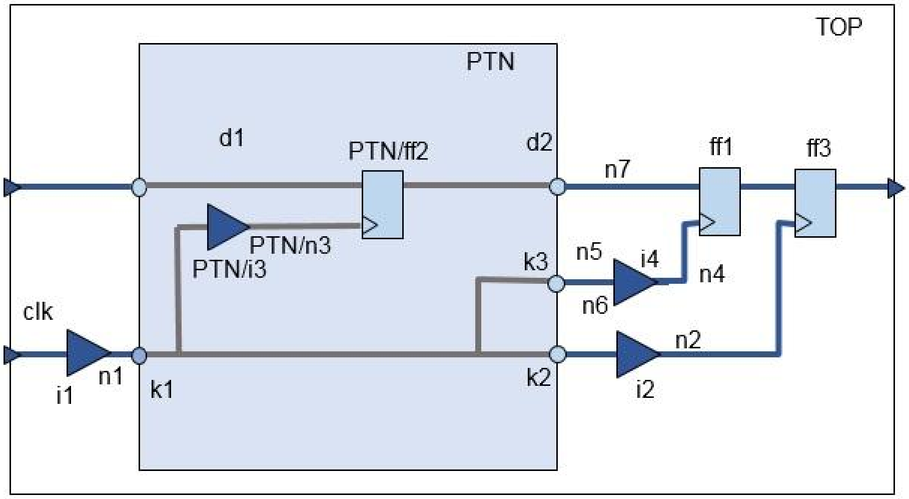


After pushing down the network:



- The clock buffer i3 is pushed down into PTN as PTN/i3. It is at the same absolute physical location as it was originally.
- To push down n1, n3 verilog ports (k1, k2, k3) are created in PTN. They are assigned to each other inside PTN module.

Post Commit Partition:



- n1 is connected to k1. Then, it is emerging as n5 and n6 nets which connect to i2 and i4. At a high level, the net n1 is basically split at each partition port. The partition ports themselves are created as per the routing topology of n1.
- PTN/n3 is completely transferred to PTN block, along with its logic and (special) wires.
- Verilog assign statements are created to mutually assign ports k1, k2, k3 inside PTN block verilog.
- After `assembleDesign`, the net n1 will re-emerge with its connections and (special) wires.

  Note: For the iHDB flow, the net n1 will re-emerge with its connections and (special) wires, after you do the following:

  ```
  set_module_model -cell * -type pnr
  ```

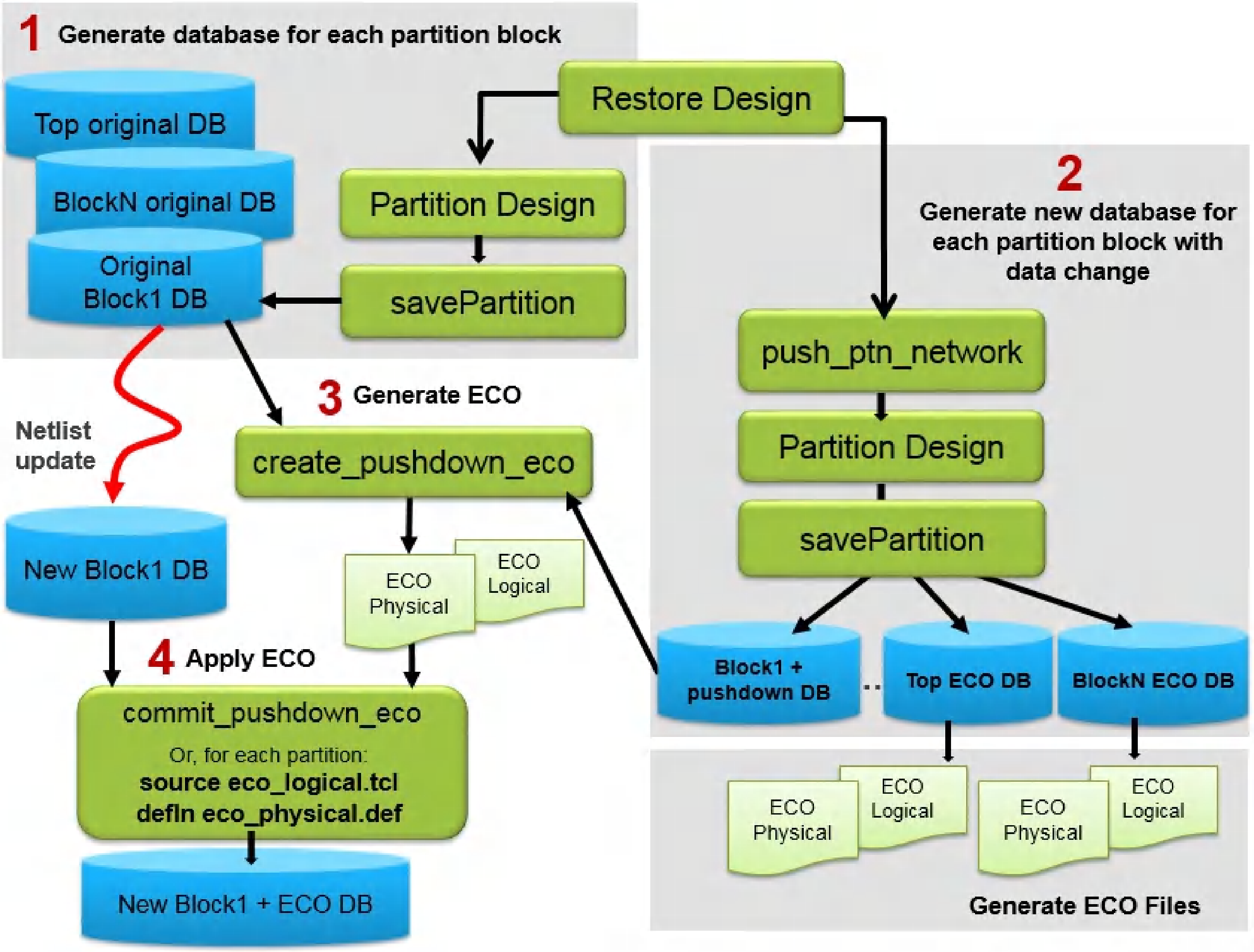
  ```
  commit_module_model
  ```

Partition Pushdown Replay Flow

In the partition pushdown replay flow ECO objects (logical netlist changes and physical changes) are generated by comparing two partition databases. These ECO objects can be applied to a new version of the partition databases. This capability caters to the specific requirements of the clock pushdown flow (using `push_ptn_network` command).

For example, in the clock pushdown flow (`push_ptn_network`), the clocks are pushed down (logically as well as physically) into overlapping partitions, and saved into partition database, let’s say pushdown_blocks database. Now, by comparing the original partition block database and pushdown_blocks database, ECO objects can be generated for each partition. Later, these partition specific ECO objects can be applied to the new versions of the partition database.

This is illustrated in the diagram below:



The following support this flow:

- `create_pushdown_eco`
  Creates logical and physical ECO files that are used to push down ECO objects by comparing two partition databases. It generates logical and physical ECO objects for each partition, in parallel, using EDP (parallel/distributed) framework. DEF ECO is generated for all logically modified nets. Additionally, if the `-ptn_net_dir` parameter specified, then a union of nets specified in `-ptn_net_dir` and the logically-modified-nets, is selected for DEF ECO generation.

- `commit_pushdown_eco`

Commits ECO for a partition block by applying ECO and generating a database for the block. It applies the previously generated ECO objects, to each partition, in parallel, using EDP (parallel/distributed) framework.

The partition pushdown replay flow involves the following steps:

- Generate the original database for each partition.
- Generate the new database that is later compared with the original database for ECO data generation.
- Generate ECO for all partitions and top using the new `create_pushdown_eco` command.
  The following is an example dbTCL script for generating the original database of each partition , generating the pushdown database, and then generating partition pushdown ECO.

  ```
  # load full-chip database;
  partition -pushDownSpecialNetAsObs $snets
  savePartition -dir $originalDir -def
  flattenPartition
  # Create push down (say, htree/push_ptn_network) database
  push_ptn_network -nets $clk_nets -output_nets_for_wire_distribution nets.txt …
  partition -pushDownSpecialNetAsObs $snets -inputNetsForWireDistribution nets.txt
  savePartition -dir $htreeBlocksDir -def
  # Create ECO by comparing original and pushdown databases
  create_pushdown_eco \
  -original_dir $originalDir \
   -pushdown_dir $htreeBlocksDir \
   -ptns [concat [dbget top.ptns.master.cell.name] [dbget top.name]] \
  -eco_object_dir $ecoObjectDir \
  -gen_script_only
  ```

  The above script generates ECO objects, for each partition, in `$ecoObjectDir`:
  ECO is generated only for the partitions specified via `-ptns` option.
- Commit ECO for a partition block on a new database of the block using the `commit_pushdown_eco` command. Alternatively, you can read the ECO objects individually for each partition.
  - Use the `commit_pushdown_eco` command for applying pushdown ECO at each partition.
    Example dbTCL script:
    On Innovus prompt:

```
commit_pushdown_eco \
  -pre_eco_dir $preEcoDir \
  -eco_object_dir $ecoObjectDir \
```

The above command applies ECO changes, specified in `$ecoObjectDir`, to all the partition databases in `$preEcoDir`, and saves post ECO database for all partitions in `$postEcoSaveSesignDir`.

- Commit the ECO objects one by one for required the partition block using the two output files – eco_logical.tcl and eco_physical.def. This option enables you to check and validate the changes before committing them.
  Example dbTCL script:

```
cd $originalDir/$ptn_name
restoreDesign $ptn_name
source ./../../$ecoObjectDir/$ptn_name/eco_logical.tcl
defIn ./../../$ecoObjectDir/$ptn_name/eco_physical.def
saveDesign ./../../$post_eco_dir/$ptn_name.enc -def
```

⚠ The partition pushdown ECO flow does not support the iHDB flow in the current release.

Focused Methodologies

In addition to generic flow methodologies, there are some specific requirements for various design styles. They are covered as under:

- Correcting Pin Illegality On Selected Pins
- Selecting Pins Using a File
- Assigning Pins of a Net
- Assigning Pins in Pre-feedthrough Netlist
- Promoting Selected Macro Pins
- Doing Pin Prioritization
  - Prioritizing Few Pins in a Selected Pin Assignment Flow
- Speeding Up Interactive Pin Assignment
- Deciding the Closest Legal Location to a Selected Position
- Pin QoR Metrics and Comparison

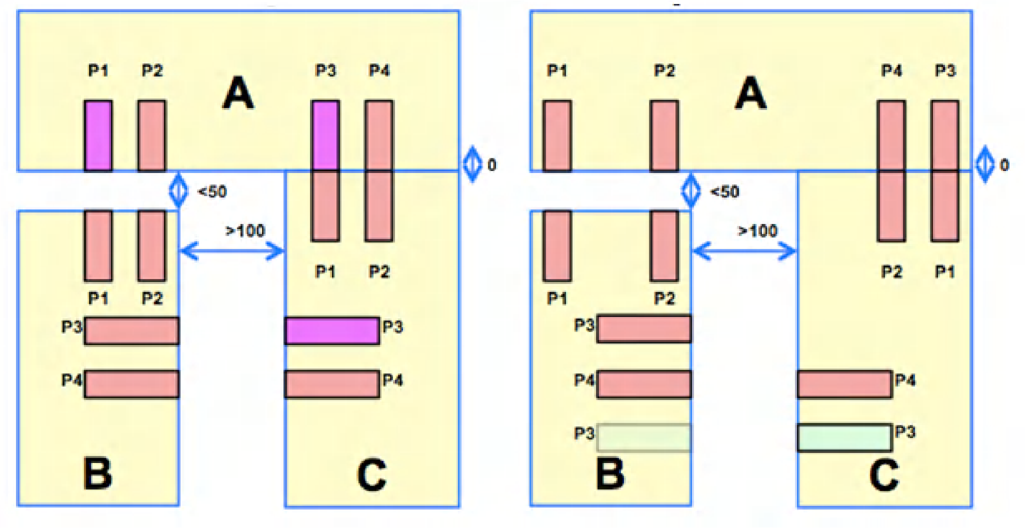
Correcting Pin Illegality On Selected Pins

You can use the `legalizePin` command to automatically correct all violations on the design. When the `checkPinAssignment` command suggests an error on a pin, this pin is part of a net that has one pin emanating from the partition containing the source and the other pin concluding at partition with sink. In order to avoid bends in the route, it is always better to move pins on both partitions. This is particularly important when a design has a narrow or no channel in between the two partitions. For abutted designs (no channels), even if you select a pin for legalization (`legalizePin -ptn` *`ptnName`* `-pin` *`pinName`*) the Innovus will move its connected pin to a location where both pins will be legal.

The behavior of moving pins in pairs (for a selected pin) is controlled by distance between two partitions. With the use of the `-auto_pairing` parameter, the `legalizePin` command automatically calculates the channel width or distance between pins in a design which will be considered for pairing.

Alternatively, for a distance (in microns) between the pins, to move corresponding connected pins of 2 pin connection nets, you can use the `setPinAssignMode -max_distance_pairing` command. The `-max_distance_pairing` option behavior extends to partitions which are less than 50 microns apart (default value) and can be easily used to set the value of the distance (lower or higher than 50) for which the pins should be moved in pairs automatically.

The following example illustrates this behavior:



In the above figure, the design has no channel between Partition A and Partition C. It has <50 microns channel between Partition A and Partition B and a channel of >100 microns between Partition B and Partition C. The pink colored pins (P1, P3 on partition A and P3 on Partition C) are illegal in the design.

Specifying the following command will only move - P1 and P3 of Partition A, P1 of Partition B, and

P1 of Partition C:

```
legalizePin -ptn A -pin {P1 P3}
```

and specifying the following command will only move P3 on Partition C:

```
legalizePin -ptn C -pin {P3}
```

In order to move pin P3 of Partition B along with P3 of Partition C, you must use the following command:

```
setPinAssignMode -max_distance_pairing 100
```

Now, P3 of Partition B will also move to the shaded green location (as shown above) along with P3 of Partition C.

Note: Even though the pins P1 of Partition B or Partition C are legal, they will still be moved to get an alignment with their corresponding connected pin.

You can avoid moving the corresponding pin for any channel width (routing bends are acceptable in channels) by using the distance value for the `-max_distance_pairing` option as 0. In such a case, whatever may be the distance between the channel, but the pins will not move in pairs and the legal pins will remain on their original position.

```
setPinAssignMode -max_distance_pairing 0
```

Selecting Pins Using a File

The `assignIoPins, assignPtnPin, checkPinAssignment, legalizePin` commands can work on selected pins also.

For example,

```
assignIoPins -pin {out1 out2 out3}
assignPtnPin -ptn A -pin {in1 in2 in3} -ptn B -pin {pina pinb pinc}
checkPinAssignment -ptn A -pin {in1 in2 in3}
legalizePin -ptn A -pin {in1 in2 in3}
```

This selection of pins can also be achieved using a list of pins, per partition, from a file.

In the pin file, the pins are specified in the following format:

```
Partition: PtnName/BlockName
Pin1
Pin2
..
PinN
Partition: PtnName/BlockName
Pin1
Pin2
..
```

```
PinN Nets:
netA
netB
..
netN
```

For example, consider the following use model:

```
assignIoPins -l_pin_file  pin.list
assignPtnPin -s_pin_file pin.list
checkPinAssignment -l_pin_file pin.list
legalizePin -l_pin_file pin.list
```

where, `pin.list` contains:

```
Partition: top
out1
out2
out3
Partition: A
in1
in2
in3
Partition: B
pina
pinb
pinc
```

Now, the `checkPinAssignment` and `legalizePin` commands will work for block: top, Partition: A and B.
All pins present in the pin.list file will be considered for selection.

Now,
For `assignIoPins` command, all Io pins will be selected, under the section Partition:top.
For `assignPtnPin` command, all pins of all partitions in the file will be considered, except the pins under the section Partition:top.
For `checkPinAssignment` and `legalizePin` commands, pins of all partitions and block (block: top, Partition: A and B) will be considered.

Note: To skip one of the partitions from selection, you can use the `-exclude_ptn` parameter. However this parameter is not applicable for the `assignIoPins` command.

For example:

- The following command ignores the partition B mentioned in the pin file pin.list.

  ```
  assignPtnPin -s_pin_file pin.list -exclude_ptn {B}
  ```

The select pins are similar to the ones selected with `assignPtnPin -ptn A {in1 in2 in3}` command.

- The following command only considers pins of partition A . pins of block Top and of partition B are ignored.

  ```
  checkPinAssignment -l_pin_file pin.list -exclude_ptn {top B}
  ```

  The select pins are similar to the ones selected with `checkPinAssignment -ptn A -pin {in1 in2 in3}` command.

- The following command considers pins of block Top and of partition B. It ignores pins of partition A mentioned in the pin file pin.list

  ```
  legalizePin -l_pin_file pin.list -exclude_ptn {A}
  ```

  The select pins are similar to the ones selected with:

  ```
  legalizePin -ptn top -pin { out1 out2 out3}
  legalizePin -ptn  B -pin {pina pinb pinc }
  ```

Assigning Pins of a Net

In the selected pin assignment flow, using the pin file for selecting pins offers an additional benefit. Instead of defining a set of pins which are connected to a net and is part of different partitions, you can select the net name itself. As a result, all the pins connected to different partitions will be derived automatically.

```
Partition: PtnName/BlockName
Pin1
Pin2 Nets:
netA
netB
..
netN
```

For example, consider a Net n1 which connects to 3 partition pins, pin x on partition A, pin y on partition B, and pin z on partition C.

The following command will do selected pin assignment of just net n1.

```
assignPtnPin -ptn a -pin {x} -ptn b -pin {y} -ptn c -pin {z}
```

Alternatively, you can use the `-pin_file` parameter to do the pin assignment for net n1 using the following command:

```
assignPtnPin -s_pin_file a.list
```

where, a.list contains:

```
Net:
n1
```

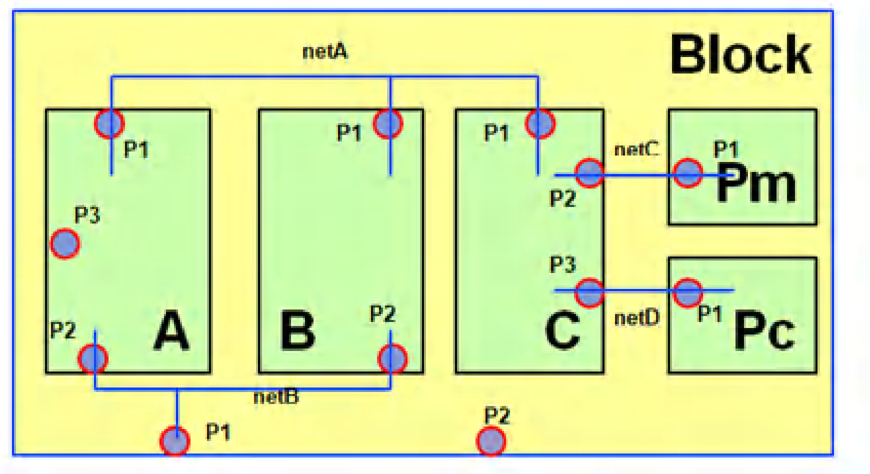
All net names specified under the Net section of the pin file will have all their pins assigned.

Note: You can use the `-exclude_ptn` *`ptnName`* parameter to ignore the pins of the specified partition.

Using pins and nets in the same pin file

The pin file supports explicit pin names for each partition but the section for net names is implicit of pins of more than one partition

Use Model: Design where the master and clone are interacting with each other



The pin file `p.list` contains:

```
Nets:
netA
netB
Ptn:Block
P2
Ptn:A
P3
```

- The following command will place block pin P1, which has been implicitly mentioned in the pin file, through netB and block pin P2, which has been explicitly mentioned in the pin file.

  `assignIoPins -l_pin_file p.list`

- The following command will place pins on nets, netA and netB, connected to all partitions (implicit mention in the file) pin P1 of PTN C, PTN B, PTN A and pin P2 of PTN B, PTN A. Pin P3 of PTN A is mentioned explicitly in the pin file.

  ```
  assignPtnPin -s_pin_file p.list
  ```

- The following command will place pins on nets, netA and netB, connected to PTN B (pin P1 and P2) and PTN C (pin P1) but will not place Pin P1 (netA) and Pin P2 (netB) on PTN A (implicit mention in file) and Pin P3 on PTN A (explicit mention in file)

  ```
  assignPtnPin -s_pin_file p.list -exclude_ptn {A}
  ```

- The following command check and legalize all pins including blocks pins P1 of PTN C, PTN B, PTN A and pin P2 of PTN B and PTN A, P3 pin on PTN A, P1 and P2 pin on block.

  ```
  checkPinAssignment -l_pin_file p.list;
  legalizePin  -l_pin_file p.list
  ```

- The following commands will check and legalize pin P1 on PTN B and pin C and pin P2 on PTN B but will ignore pin P1, P2 of Block and pins P1, P2, P3 of PTN A

  ```
  checkPinAssignment -l_pin_file p.list -exclude_ptn {A Block};
  legalizePin -l_pin_file p.list -exclude_ptn {A Block}
  ```

For master and clone connections:

If pin file `pc.list` contains:

```
Nets:
netC
```

- The following command will place pin P1 on PTN P (both master and clone) and pin P2 only on PTN C.

  ```
  assignPtnPin -s_pin_file pc.list
  ```

If pin file `pd.list` contains:

```
Net:
netD
```

- The following command will place pin P1 on PTN P (both master and clone) and pin P3 only on PTN C.

  ```
  assignPtnPin -s_pin_file pd.list
  ```

- The following command will place pin P1 on both master and clone.

  ```
  assignPtnPin -s_pin_file pd.list -exclude_ptn {C}
  ```

If pin file `pe.list` contains: `Ptn:P` *`P1`*

- The following command will place pin P1 on PTN P (both master and clone).

  `assignPtnPin -s_pin_file pe.list`

If pin file `pf.list` contains:

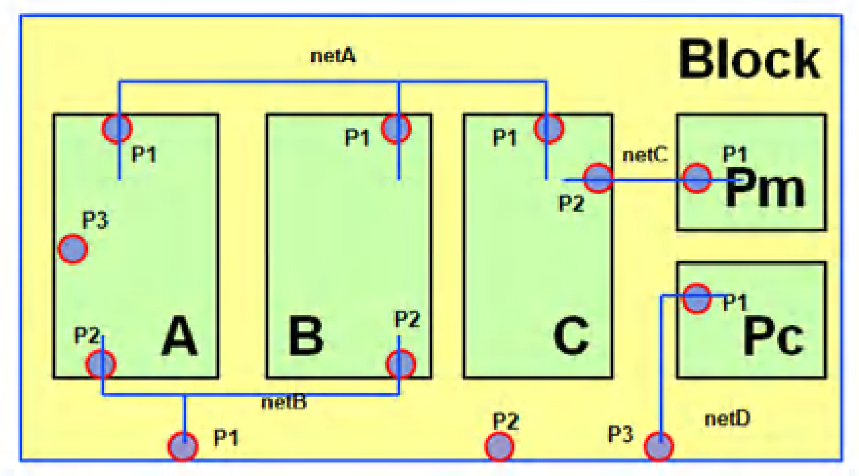
```
Ptn:C P2
```

- The following command will only place pin P2 on PTN C.

  `assignPtnPin -s_pin_file pf.list`

Note: The pin selection for commands `assignIoPins`, `checkPinAssignment`, and `legalizePin` will be the same.

Use Model: Design where the master and clone are interacting with top



If pin file `pc.list` contains:

```
Nets:
netC
```

- The following command will consider pin P1 on partition P (both master and clone) and only consider pin P2 on PTN C.

  `assignPtnPin -s_pin_file pc.list`

- The following command will consider pin P1 on partition P (both master and clone)

  `assignPtnPin -s_pin_file pc.list -exclude_ptn {C}`

- The following command will only consider pin P2 on PTN C

  `assignPtnPin -s_pin_file pc.list -exclude_ptn {P}`

If pin file `pd.list` contains:

```
Nets:
netD
```

- The following command will consider pin P1 on partition P (both master and clone)

  ```
  assignPtnPin -s_pin_file pd.list
  ```

- The following command will assign pin P3 of block.

  ```
  assignIoPins -s_pin_file pd.list
  ```

If pin file `pe.list` contains:

```
Ptn:P P1
```

- The following command will place pin P1 on PTN P (both master and clone).

  ```
  assignPtnPin -s_pin_file pe.list
  ```

If pin file `pf.list` contains:

```
Ptn:C P2
Ptn:Block
P3
```

- The following command will only place pin P2 on PTN C.

  ```
  assignPtnPin -s_pin_file pf.list
  ```

- The following command will assign pin P3 on Block.

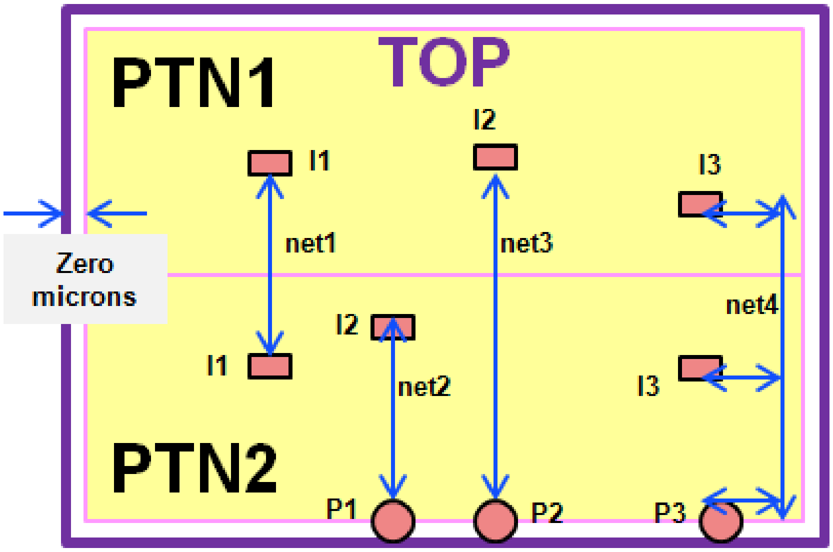
  ```
  assignIoPins -l_pin_file pd.list
  ```

Note: The pin selection for commands `assignIoPins`, `checkPinAssignment`, and `legalizePin` will be the same.

Assigning Pins in Pre-feedthrough Netlist

Abutted designs have no channels between partitions. You must use the `insertPtnFeedthrough` command on such designs before pin assignment so that ports are inserted into the partitions which lie in between the partitions that contain source and sinks. This makes the design routable by avoiding any violations in the hierarchy.

The following example illustrates this behavior:

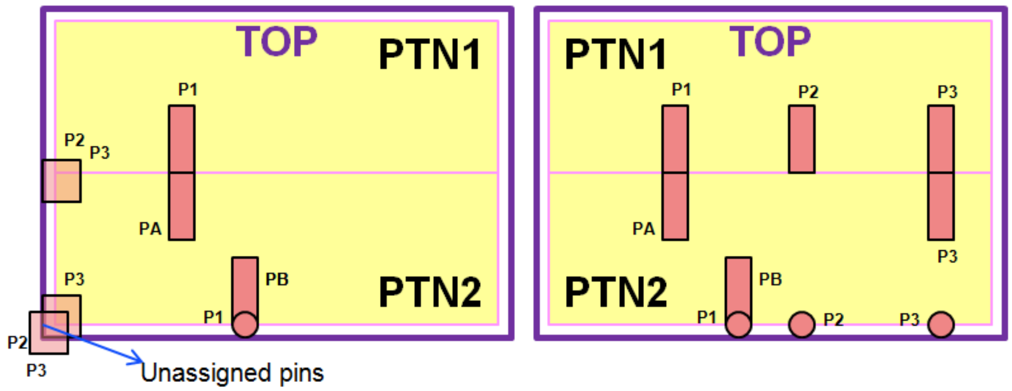


The design shown in the figure above is a fully abutted design with block TOP and partitions PTN1 and PTN2.
It has the following 4 nets:

- net1 connects inst I1 of PTN1 to inst I1 of PTN2
- net2 connects inst I2 of PTN2 to TOP (block) port P1
- net3 connects inst I2 of PTN1 to TOP port P2
- net4 connects inst I3 of PTN1 to inst3 of PTN2 and top port P3

For nets net1 and net2 a feedthrough is not required as the net routing will not cross over unrelated partitions. However, nets net3 do require a feedthrough as the route would cross over unrelated partition PTN2.
for and net4, feedthrough is required because PTN2 requires 2 ports (for entry and exit points of routes) but it just has the entry point port.

When you use the `assignIoPins`, `assignPtnPin`, `pinAlignment` commands, they will give the results shown in Figure 1. They cannot put pins for nets net3 and net 4, because it will create illegal pins. The `checkPinAssignment` command will report abutment violations on pins if you manually try to keep pins as shown in Figure 2. The `legalizePin` command will not touch them, as it does not have any legal pin location for such pins



Thus to resolve these issues and get legal pin locations, the feedthrough of these nets is required. To get pins (of nets that would require feedthrough) assigned in pre feedthrough netslist stage use the following command:

```
setPinAssignMode -strict_abutment false
```

This would:

- Enable the `assignIoPins, assignPtnPin, pinAlignment` commands to keep pins with abutment violations.
  Note: Pin P3 of PTN2 can come at 2 locations: aligned with P3 of PTN1 or can be aligned to top port P3 (shown in light color in the figure)
- Enable the `checkPinAssignment` command to avoid reporting abutment violation on these nets.
- Enable the `legalizePin` command to move such pins in order to remove other type of issues.

Note: The `checkPinAssignment` command will continue to flag genuine abutment violations. For example, if pin P1 of PTN1 and PA of PTN2 are placed at not aligned location, then the `checkPinAssignment` command will report abutment violation for them, even with the strict abutment mode is set to false. To avoid reporting such violations, you can use the `checkPinAssignment -ignore {pin_abutment` } command to ignore the abutment check all together.

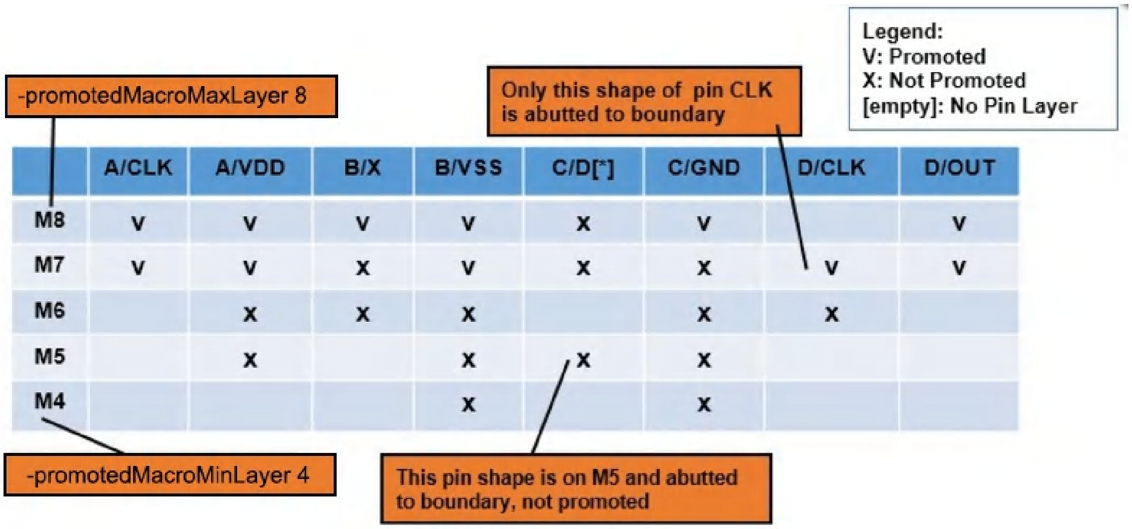
Promoting Selected Macro Pins

You can use the `setPromotedMacroPin` command to select or mark signal and/or PG pin shapes to be promoted later on. Using the `setPromotedMacroPin -layers` parameter you can specify the metal layer(s) of the macro pins. Alternatively, if `-layers` is not specified, all pin layers between `setPinAssignMode -promotedMacroMinLayer` and `setPinAssignMode -promotedMacroMaxLayer` range are selected for macro pin promotion.

Note: The `setPromotedMacroPin` command should be called after specifying the maximum and minimum metal layer names for promoting macro pins using the `setPinAssignMode -promotedMacroMaxLayer` *`maxLayer`* `and -promotedMacroMinLayer` *`minLayer`* parameters, respectively.

The following commands promote the top most layer pins of selected macros:

```
setPinAssignMode -promotedMacroMinLayer 4 -promotedMacroMaxLayer 8
setPromotedMacroPin -reset
setPromotedMacroPin -insts {A B} -pins * -layers {7 8}
setPromotedMacroPin -insts B -pins X -layers 8 -override
setPromotedMacroPin -insts C -layers 5 -abuttedToBoundaryOnly
setPromotedMacroPin -insts C -layers {8}
setPromotedMacroPin -insts C -pins D[*] -layers {} -override
setPromotedMacroPin -insts D -pins OUT
setPromotedMacroPin -insts D -pins CLK -abuttedToBoundaryOnly
assignIoPins
```



Legend:
V: Promoted
X: Not Promoted
[empty]: No Pin Layer

-promotedMacroMaxLayer 8

Only this shape of pin CLK is abutted to boundary

	A/CLK	A/VDD	B/X	B/VSS	C/D[*]	C/GND	D/CLK	D/OUT
M8	V	V	V	V	X	V		V
M7	V	V	X	V	X	X	V	V
M6		X	X	X		X	X	
M5		X		X	X	X		
M4				X		X		

-promotedMacroMinLayer 4

This pin shape is on M5 and abutted to boundary, not promoted

The `setPromotedMacroPin` command settings are honored by the `assignPtnPin, assignIoPins` and `partition` commands automatically.

- `assignPtnPin`: In top-down flow, the `assignPtnPin` command only promotes signal pins from the list of pins promoted by the `setPromotedMacroPin` command.
  For example, the following command assigns partition pins and promotes macro signal pins from layer 4 to layer 6.

  ```
  setPinAssignMode -promotedMacroMinLayer 4 -promotedMacroMaxLayer 8
  setPromotedMacroPin -insts * -layers {4 5 6}
  assignPtnPin
  ```

- `partition`: In top-down flow, the partition command only promotes P/G pins from the list of pins promoted by the `setPromotedMacroPin` command.

For example, the following commands promote metal 10 VSS pins on layer 10 of macros that have prefix MEM. Design technology has 12 metal layers

```
setPinAssignMode -promotedMacroMinLayer 10
setPromotedMacroPin -reset
setPromotedMacroPin -insts MEM* -pins VSS -layers 10
partition
```

- `assignIoPins`: In bottom-up flow, the `assignIoPins` command promotes both signal and P/G pins from the list of pins promoted by the `setPromotedMacroPin` command.

  For example in a design where signal pins are already placed, the following command promotes P/G pins on metal 12. Design technology has 12 metal routing layers.

```
setPtnPinStatus -cell [dbGet top.name] -pin * -status fixed
setPromotedMacroPin -layers 5 -pins [dbGet -u [dbGet -p top.insts.cell.subClass
block].pgTerms.name]
assignIoPins
```

The `-reportOnlyFile` parameter of the `setPromotedMacroPin` command can be used to write out all `setPromotedMacroPin` settings in memory into an output TCL file which can be sources later on. For example, the following settings will be written to the mmMPP.tcl file.

```
setPromotedMacroPin -insts A -layers {7 8}
setPromotedMacroPin -insts B -layers 7
setPromotedMacroPin -insts C -pins S -layers 6
setPromotedMacroPin -insts D -pins T
setPromotedMacroPin -reportOnlyFile myMPP.tcl
```

The output report file (*myMPP.tcl*) contains:

```
setPromotedMacroPin -reset
setPromotedMacroPin -insts A -layers {7 8}
setPromotedMacroPin -insts B -layers 7
setPromotedMacroPin -insts C -pins S -layers 6
setPromotedMacroPin -insts D -pins T
```

If the `setPromotedMacroPin` command is specified, the `assignPtnPin`, `assignIoPins`, and `partition` commands display the count of the Signal/PG pins promoted for partitions/design in a log file (as applicable).

The following is an example of the log file of the `assignPtnPin` command in the top down flow.

```
Summary Report for Partition: tdsp_core
Total number of promoted signal pins: 10
Total number of promoted signal pins through switch -abuttedToBoundaryOnly: 3
Total number of all promoted pins: 13
…
```

Doing Pin Prioritization

As there may be limited slots which ensure less congestion and a better route length resulting in an improved timing result, you may want to prioritize putting some pins first in the pin assignment flow. You can use the following methods to prioritize pins and get better pin locations for certain pins:

Method 1

Use the `specifyNetWeight` command to specify the priority weighting of a net. Pins that connect to a net that has a higher net weight are assigned before the pins that connect to a net with a lower net weight. For example, the following commands set the priority net weighting for nets net1 to first and net2 to second. All pins on net1 will have more slots to choose from:

```
specifyNetWeight net1 5
specifyNetWeight net2 4
assignPtnPin
```

Pins with net weights are prioritized after putting:

- Pins with location constraints, set using the `setPinConstraint` command
- Pin/net grouped/guided pins
- Master and clone pins

Note: If a slot has been assigned to a grouped pin in the default pin assignment flow, this slot cannot be assigned to any pin of a net by setting the net weight on it.

Method 2

Select only a few pins for pin assignment to ensure that these pins have maximum slots to choose from and decide on the best pin locations. For example, the following commands place the pins in[1:5] (`in1, in2, in3, in4`, and `in5`) first. The pins out[1:5] (`out1, out2, out3, out4,` and `out5`) will be placed later. The out[1:5] pins will not move pins in[1:5] from their positions, therby ensuring out[1:5] will occupy from the rest of the pins.

```
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}
assignPtnPin -ptn A -pin {out1 out2 out3 out4 out5}
```

Note: You can also do the pin selection using the pin_file flow to achieve the same results.

Note: If the selected pins are part of a pin group, then all the pins of the group will be placed in the flow.

For example, If pin in2 is part of pin group GRP having pins {data reset in2} then the following command will place all pins including pins in1, in2, in3, in4, in5, data, and reset.

```
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}
```

However, you can do the following to avoid placing the whole group of pins:

- Use the `assignPtnPin -ignore_group_pins` command. The following command will place only the pins in1, in3, in4, and in5 .

  ```
  assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5} -ignore_group_pins
  ```

- Use the `createPinGroup` *`PinGroup`* `-optimizeOrder` command to s pecify that pins in the *PinGroup* are reordered to optimize wire length.
  Now pin in2 is part of pin group GRP having pins {data reset in2} with `-optimizeOrder`, then the following command will place pins in1, in2, in3, in4, in5

  ```
  createPinGroup  PinGroup  -cell {A} -pin {data* reset in2} -optimizeOrder
  ```

  ```
  assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}
  ```

Note: If the selected pin is of a master or clone, then pin on both the master and clone will be placed (and not the pin of the other connected partitions). However, connectivity to other partition for both master and clone will be considered.

The following example illustrates this behavior:

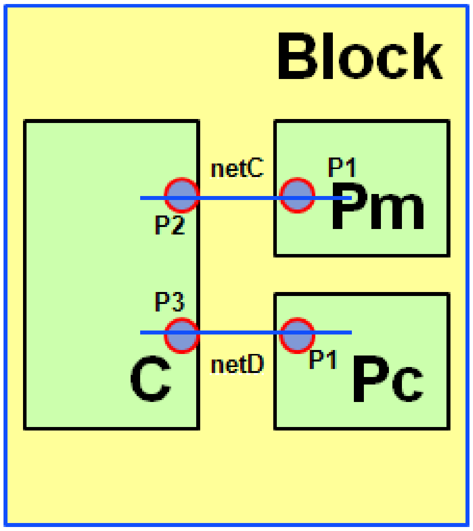


Figure 1

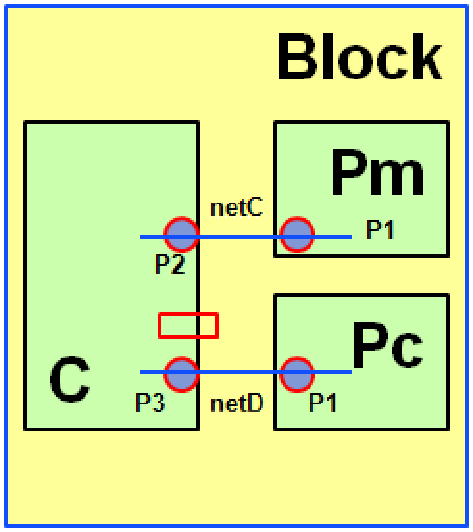


Figure 2

As shown in Figure 1, specifying the following command will place pins of both master Pm and clone Pc while considering pins P2 and P3 of partition C (part of net netC and netD).

```
assignPtnPin -ptn P -pin {P1}
```

In Figure 2, there is a pin blockage on partition C. Now, pin P1 of partition P will be placed in such a way that they consider connection of both nets connecting to (master/clone) partition pin P1. Hence, the pins will be placed at location which is optimized with position of pins P2 and P3 of partition C. In the figures above, the pins of partition C are shown for illustration only and will not be placed by command `assignPtnPin -ptn P -pin {P1}`.

Note: Both the methods of pin prioritization honor constraints on other connected pins or partitions.

For example, consider a net which has 2 pins. The first pin going to partition A through pin in1 and the second pin connected to partition B though pin out1. Pin out1 of partition B has a location constraint. In this case while doing selected pin assignment of partition A using the following command:

```
assignPtnPin -ptn A -pin {in*})
```

the position of pin in1 of partition A will be chosen considering the location constraint of pin out1 of partition B (even though pin out1 of partition B has not yet been placed).

However, if the following command was specified such that the pin in1 was not in the original selection list

```
assignPtnPin -ptn A -pin {out*})
```

then it is not guaranteed that pin in1 will be at the same location (that would be best for it, considering the location constraint on its corresponding pin (out1 of partition B)), because any of the selected {out*} pin can now occupy that location.

Prioritizing Few Pins in a Selected Pin Assignment Flow

You can combine both the methods described above to select pins and then set priority among those selected pins. For example, you can use any of the following use models to select the same set of pins - `{ in1 in2 in3 in4 in5} {out1 out2 out3 out4 out5}`. These selected pins may not have priority among them.

Model 1

```
assignPtnPin -ptn A -pin {in1 in2 in3 in4 in5}
assignPtnPin -ptn A -pin {out1 out2 out3 out4 out5}
```

Model 2

```
assignPtnPin -pin_file a.in.list
assignPtnPin -pin_file a.out.list
```

where, the pin files contain

a.in.list	a.out.list
Partition: A in1 in2 in3 in4 in5	Partition: A out1 out2 out3 out4 out5

In both these models, Innovus does not choose to favor certain pins over others. However, if you want that among the list of selected pins, priorities should be set, then you can put *netWeight* on the nets of chosen pins. For example, after selecting pins using the above commands, you may specify:

```
specifyNetWeight net5 6
assignPtnPin -pin_file a.in.list
```

Now pin in5 of net net5 will be considered first among the other in* pins out of all the pins of partition A.

Speeding Up Interactive Pin Assignment

When you use the `editPin` batch command or the Pin Editor in the interactive pin assignment flow, internal data structures are created to work on a given solution space. When you give a sequence of commands for pin assignment, the solution space (floorplan objects that impact pin assignment) is not changed. But each of the specified commands creates these data structures individually, to speed up the interactive pin assignment flow, you can choose to reuse these data structures. To enable pin-editing in batch mode, you can use the `setPinAssignMode -pinEditInBatch true` command. In the batch mode, multiple `editPin` commands reuse internal data structures for all pins without recreating them for each individual command.

For example:

```
setPinAssignMode -pinEditInBatch true
editPin -pin {P1} -layer 2 -assign X1 Y1
editPin -cell {A} -pin {P2} -layer 3 -assign X2 Y2
editPin -cell {A} -pin {P3} -layer 4 -assign X3 Y3
editPin -cell {B} -pin {P4} -layer 4 -assign X4 Y4
editPin -cell {B} -pin {data*} -layer 4 -spreadType start -start 0 0 -side r
editPin -pin {data*} -layer 4 -spreadType start -start 0 0 -side r
editPin -cell {A} -pin {data*} -layer 4 -spreadType start -start 0 0 -side r
….
….
```

```
….
setPinAssignMode -pinEditInBatch false
```

Notes:

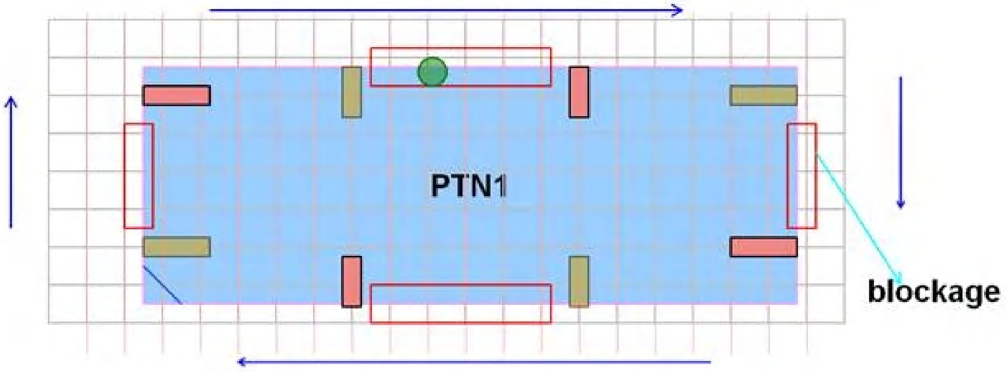
- Once the interactive pin assignment is complete, it is important to set the mode to false. This can be done using the `setPinAssignMode -pinEditInBatch false` command.
- Floorplan changes (involving but not limited to pin/routing blockages, pin guides, pre routes, macro placement, size/shape of fences/design) are not allowed and should not be carried out during the batch mode.

Deciding the Closest Legal Location to a Selected Position

To assign a pin on a selected position on a partition on a layer's top side, you use the following command:
`editPin -cell` *`cellName`* `-pin` *`pinName`* `-assign {`*`x y`*`} -layer` *`layerID`* `-side top`

For example, the following command assigns pin A of partition Ptn1 at xy position on the top side with layer 2.

```
editPin -cell PTN1 -pin A -assign X Y -layer 2 -side top
```



However, if the selected position is not a legal position, as shown by the green dot in the figure above, then by default the pin will go to the right. This is shown by the red pin in the figure. The default spread direction is clockwise for all edges.

- Top: Left to Right
- Bottom: Right to Left

Note: In the above figure, all red pins depict the chosen location (always clockwise for all edges) to keep the pin in case the provided location is not feasible. All yellow pins depict the location to keep the pin in counter clock wise direction to provided unfeasible location

In order to get the pins in the opposite direction, you can use the `editPin -spreadDirection` `{counterclockwise}` parameter. For example,

```
editPin -cell PTN1 -pin A -assign X Y -layer 2 -side top -spreadDirection
{counterclockwise}
```

You can also use the `-spreadDirection {both}` parameter to automatically choose the closest legal location in both directions. This enables the `editPin` command to decide the closest legal location to desired `x y` location in both direction and place the pin there. In the figure shown above, the pin will be placed on the left side (yellow pin), which is 1 tracks away (rather than going to right side (red pin), which is 3 tracks away from the green dot (desired x y), on the top edge.

Pin QoR Metrics and Comparison

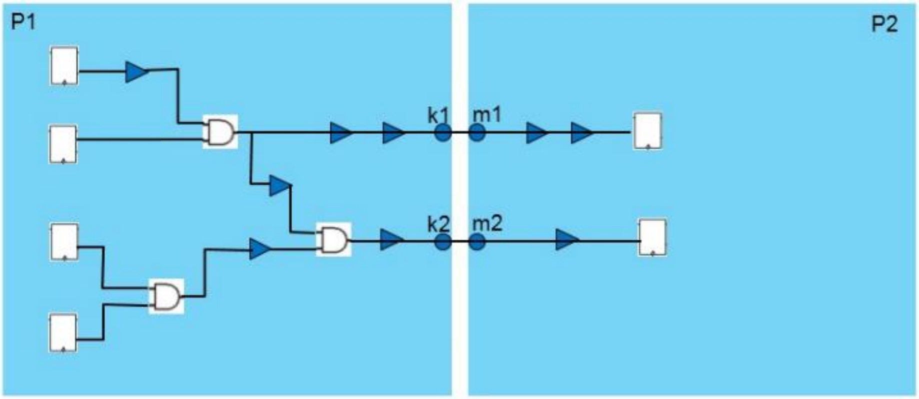
Pin QoR generation and comparison helps to determine the best pin assignment arrangement for a design. You can use the `reportPinAssignStatisticss` command to generate pin QoR data for a specified partition and use the `comparePinAssignStatistics` command to generate a pin QoR comparison file based on a reference and a target pin QoR file.

Pin QoR Metric:

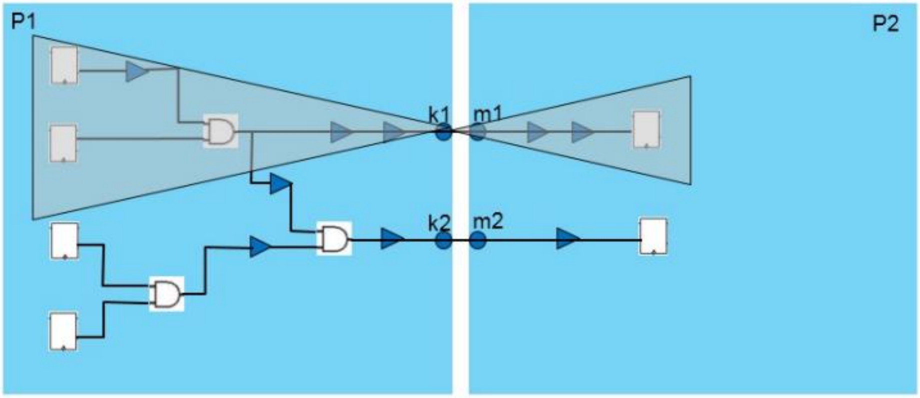
To generate a pin QoR metric for a pin arrangement, a wire-length metric is associated with every pin corresponding to the wires in the timing cone of the pin.

Pin QoR Metric Flow

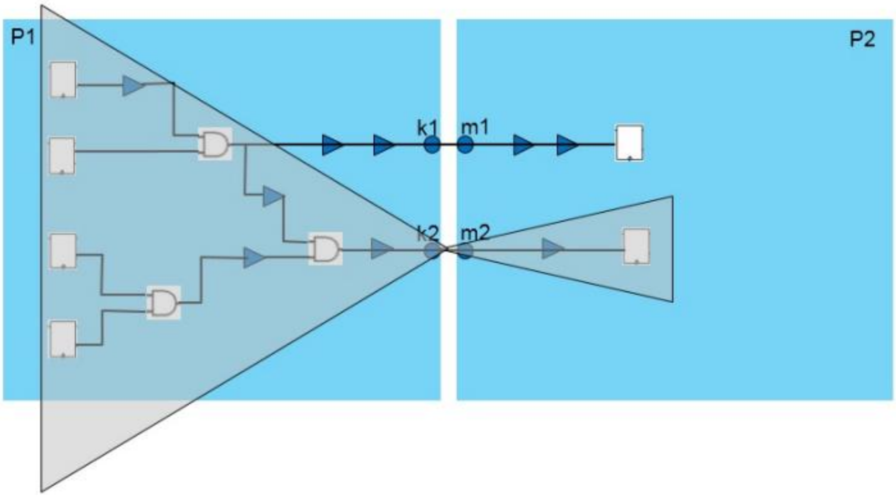
- Load a placed design.



P1/k1 Timing Cone is shown below. It is calculated as all_fanin + all_fanout of P1/k1. The same timing cone applies to P2/metal1 also.

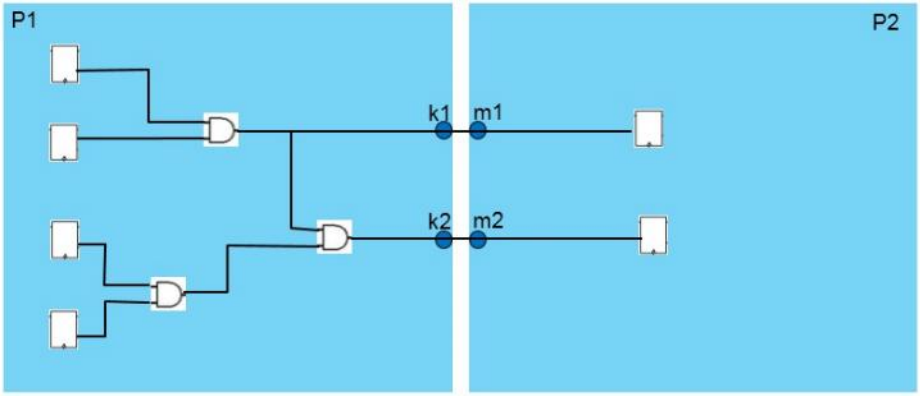


P1/k2 Timing Cone is shown below. It is calculated as all_fanin + all_fanout of P1/k2. The same timing cone applies to P2/metal2 also.



- Delete buffers and inverters. The following command can be used to delete buffer and inverters from the placed design.

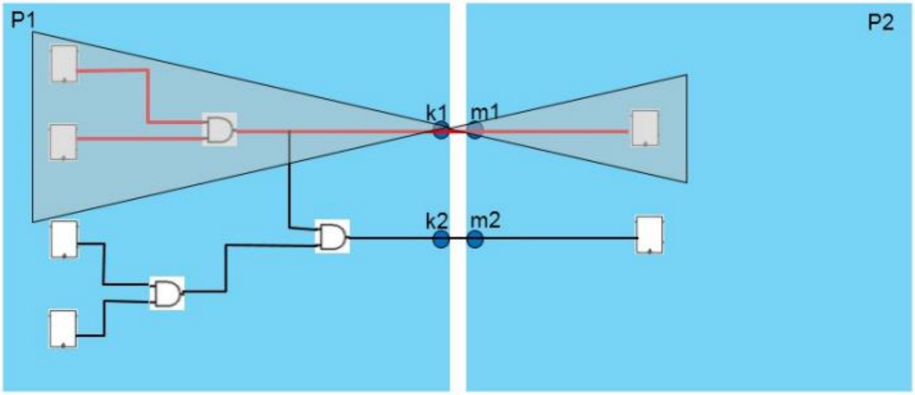
```
deleteBufferTree
```



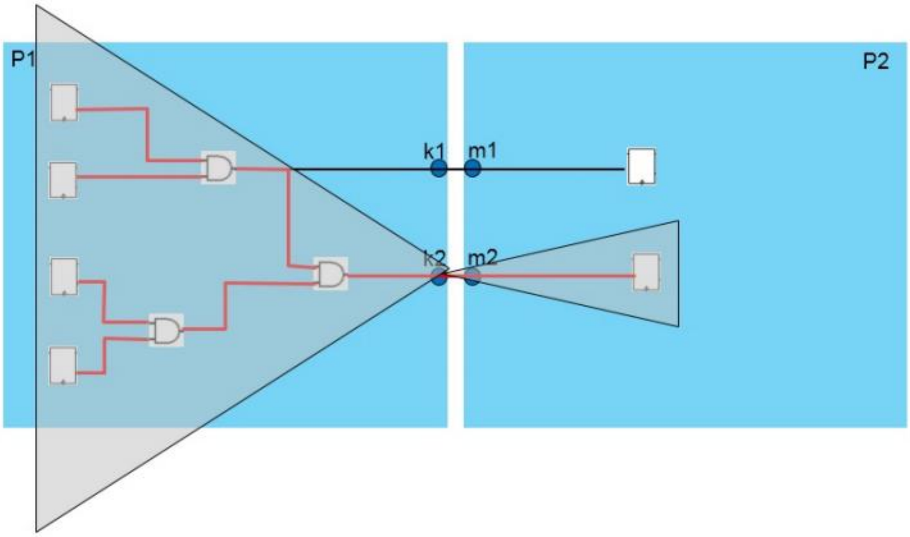
- Perform routing honoring pins and hierarchy

  ```
  setRouteMode -earlyGlobalRoutePartitionHonorFence list_of_ptn_cell_names -earlyGlobalRoutePartitionHonorPin list_of_ptn_cell_names
  earlyGlobalRoute
  ```

  The P1/k1 Timing Cone wires shown in red below:



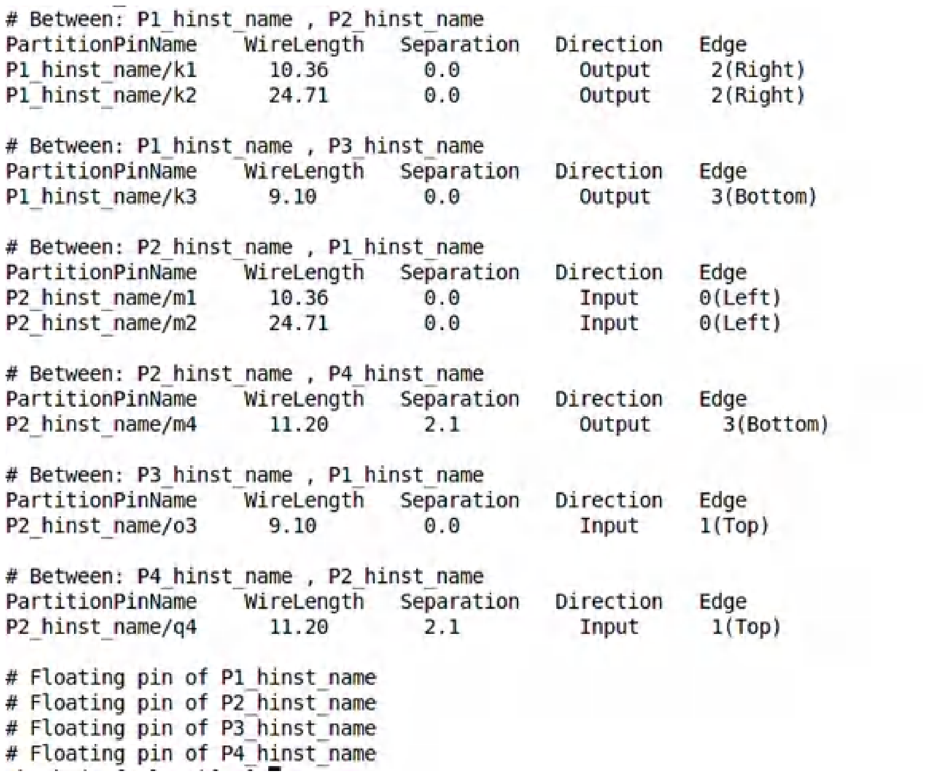
  P1/k2 Timing Cone wires shown in red below:



- Report pin QoR metric. You can use the `reportPinAssignStatistics` command to generate the pin QoR data for a specified partition(s) in an output file.

  ```
  reportPinAssignStatistics -outFile pin_qor.txt
  ```

  For Example:



```
# Between: P1_hinst_name , P2_hinst_name
PartitionPinName    WireLength   Separation   Direction   Edge
P1_hinst_name/k1      10.36        0.0          Output      2(Right)
P1_hinst_name/k2      24.71        0.0          Output      2(Right)

# Between: P1_hinst_name , P3_hinst_name
PartitionPinName    WireLength   Separation   Direction   Edge
P1_hinst_name/k3      9.10         0.0          Output      3(Bottom)

# Between: P2_hinst_name , P1_hinst_name
PartitionPinName    WireLength   Separation   Direction   Edge
P2_hinst_name/m1      10.36        0.0          Input     0(Left)
P2_hinst_name/m2      24.71        0.0          Input     0(Left)

# Between: P2_hinst_name , P4_hinst_name
PartitionPinName    WireLength   Separation   Direction   Edge
P2_hinst_name/m4      11.20        2.1          Output       3(Bottom)

# Between: P3_hinst_name , P1_hinst_name
PartitionPinName    WireLength   Separation   Direction   Edge
P2_hinst_name/o3      9.10         0.0          Input     1(Top)

# Between: P4_hinst_name , P2_hinst_name
PartitionPinName    WireLength   Separation   Direction   Edge
P2_hinst_name/q4      11.20        2.1          Input      1(Top)

# Floating pin of P1_hinst_name
# Floating pin of P2_hinst_name
# Floating pin of P3_hinst_name
# Floating pin of P4_hinst_name
```

Note:

- If a net is not routed, then, manhattan distance between source-sink pairs is be used to replace corresponding wire-length distance.
- For floating pins (from outside) of a partition,the pin is always assumed to be within the timing cone box.
- Pin QoR statistics is not generated for blackbox partitions.

Pin QoR Comparison:

For pin QoR comparison, two different pin QoR metric reports are used. The wire length metrics for every pin is compared and the %change is displayed in tabular form in a text file.

Pin QoR Comparison Flow

- Use the reports generated by the `reportPinAssignStatistics` command. Consider the following example:
  For a design, the following command generates a *`pin_qor_reference.txt`* pin QoR metric report.

  ```
  reportPinAssignStatistics –outFile pin_qor_reference.txt
  ```

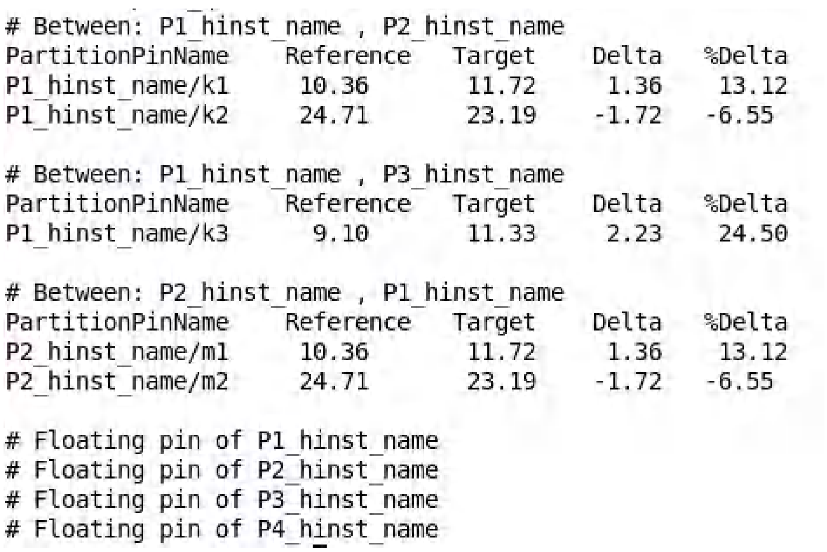
  Now, for a new version of the same design, another pin QoR metric report is generated.

Note: A new version of the same design, can have some partition ports added or deleted.

The following command generates a *`pin_qor_target.txt`* pin QoR metric report:

```
reportPinAssignStatistics -outFile pin_qor_target.txt
```

- Use the `comparePinAssignStatistics` command to generates a pin QoR comparison file based on a reference and a target pin QoR file.
  For example:
  The following command compares the two pin QoR metric reports against timing-cone-wire-length data:
  ```
  comparePinAssignStatistics -reference pin_qor_reference.txt -target
  pin_qor_target.txt -outFile qor_compare.txt
  ```
  The following is the *`qor_compare.txt`* report



```
# Between: P1_hinst_name , P2_hinst_name
PartitionPinName    Reference   Target    Delta   %Delta
P1_hinst_name/k1     10.36       11.72     1.36    13.12
P1_hinst_name/k2     24.71       23.19    -1.72   -6.55

# Between: P1_hinst_name , P3_hinst_name
PartitionPinName    Reference   Target    Delta   %Delta
P1_hinst_name/k3      9.10       11.33     2.23    24.50

# Between: P2_hinst_name , P1_hinst_name
PartitionPinName    Reference   Target    Delta   %Delta
P2_hinst_name/m1     10.36       11.72     1.36    13.12
P2_hinst_name/m2     24.71       23.19    -1.72   -6.55

# Floating pin of P1_hinst_name
# Floating pin of P2_hinst_name
# Floating pin of P3_hinst_name
# Floating pin of P4_hinst_name
```

  Note:
  - If a partition port is found in the reference report but not in the target report, and vice-versa, then it will not be reported in the compare_qor.txt.
  - In the reference report, if P1/k1 is connected to P2/k2 but in the target report, P1/k1 is connected to P2/k4. Then no comparison data will be printed for P1/k1. That is to say, the inter-partition-connectivity should remain the same, in reference and target, for it to be reported as a comparison row in compare_qor.txt

Handling Instance Groups Associated with Partitions

For a clear distribution and better handling of grouped instances associated with partitions, the inst groups of the following types can be pushed down and assembled back:

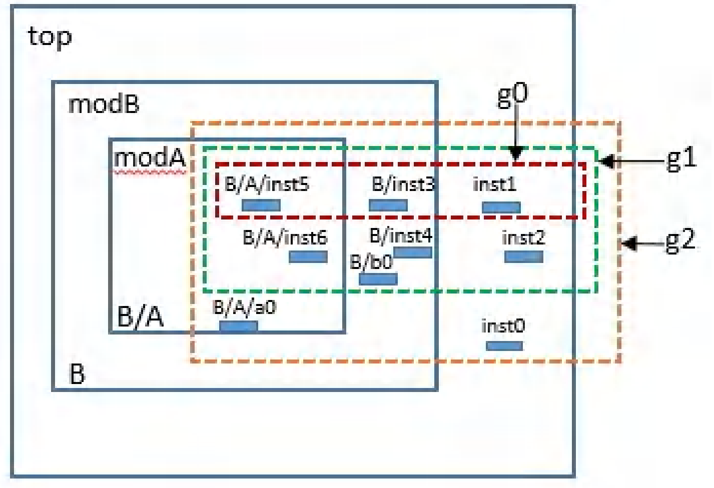
- Simple instance groups where all elements of the group are of the same partition
- Nested partitions and nested groups
- Cross partition groups
- Instance groups with master and clone partitions

For cross partition groups,

- at the time of committing the partition, a group is created in the participating partition with the same name, and only the group elements contained in it are added to the group.
- at the time of `assembleDesign`/`flattenPartition`, the groups are made unique by prefixing with the partition inst name.

Example 1: Nested partitions with nested and cross partition groups

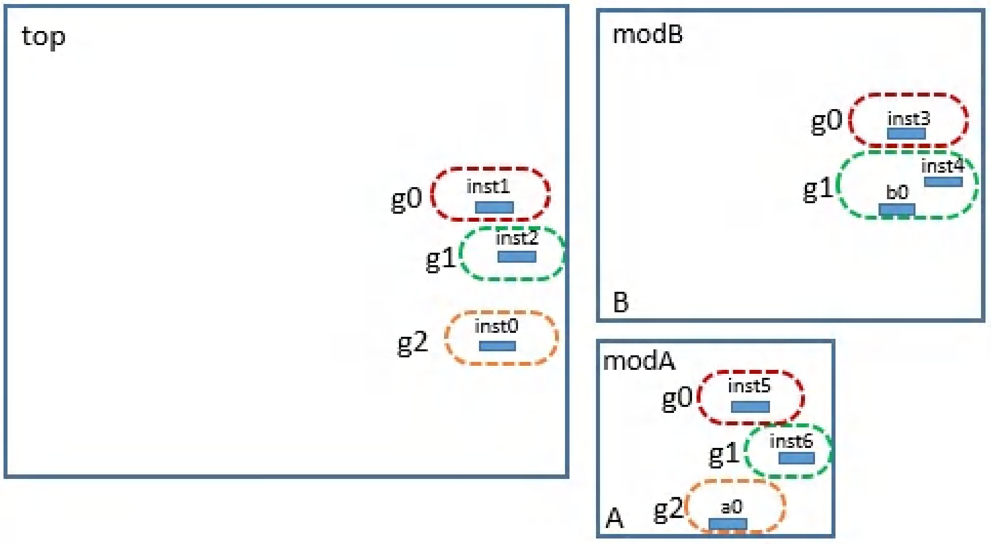
The following is an example of nested partitions with nested and cross partition groups.



The instance groups are:

```
Group:Insts
g0: {inst1 B/inst3 A/B/inst5}
g1: {inst2 B/inst4 B/b0 B/A/inst6 g0}
g2: {inst0 B/A/a0 g1 }
```

After partition, the individual partition level groups and their members are represented as:



After flattening the partition or assembling the partition back at chip, the instance groups are named uniquely where flattened inst names are prefixed to created group names.

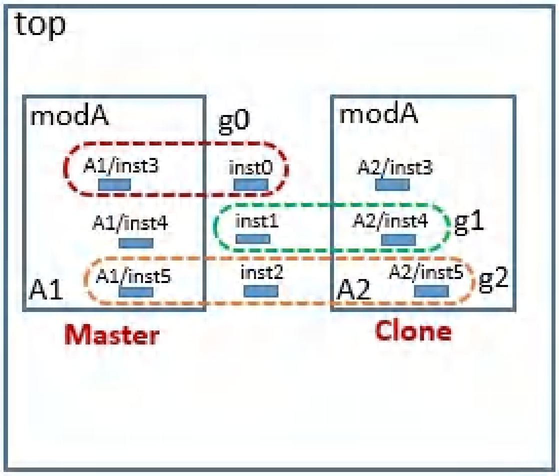
Updated instance groups:

`g0: {inst1}`	`B_g0: {B/inst3}`	`B_A_g0: {B/A/inst5}`
`g1: {inst2 g0}`	`B_g1: {B/inst4 B/b0}`	`B_A_g1: {B/A/inst6}`
`g2: {inst0 g1}`		`B_A_g2: {B/A/a0}`

Example 2: Instance groups with master and clone partitions

The following is an example of instance groups with master and clone partitions.

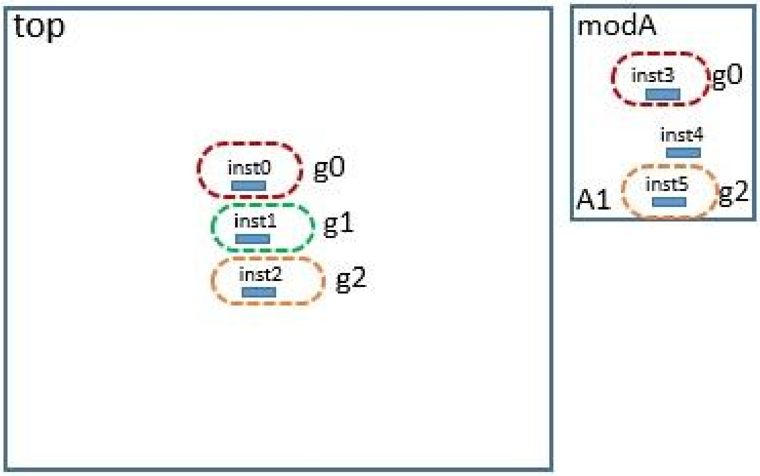
Here, A1 is master and A2 is clone.



The instance groups are:

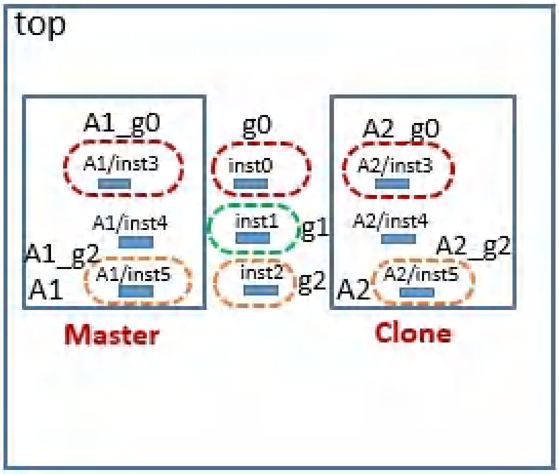
```
Group:Insts
g0: {inst0 A1/inst3}
g1: {inst1 A2/inst4}
g2: {inst2 A1/inst5 A2/inst5}
```

After partition, the individual partition level groups and their members are represented as:



After flattening the partition or assembling the partition back at chip, the instance groups are named based on the following:

- g0: When instances from the master partition and other hierarchy are added in a group, same instances from the clone are defined for the clone instance too.
- g1: When instances from the clone and other hierarchy are added in a group, the clone instances are ignored when partitioning is done.
- g2: When instances from both master and clone partitions, and other hierarchy are added in a group, the clone instances are ignored and the instances similar to the master are defined for the clone as well.

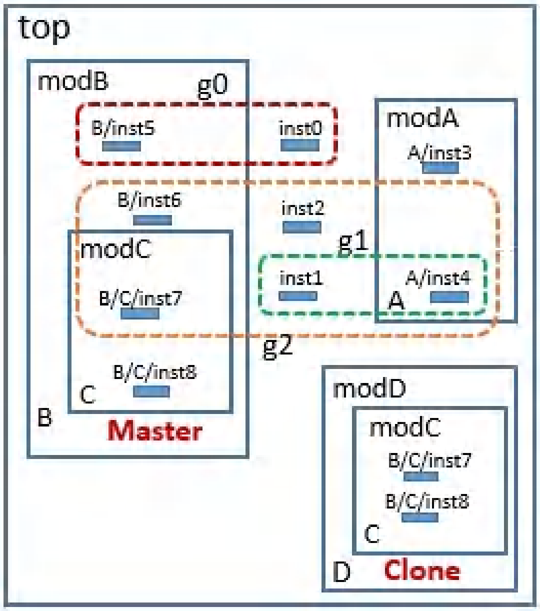


Updated instance groups:

`g0: {inst0}`	`A1_g0: {A1/inst3}`	`A2_g0: {A2/inst3}`
`g1: {inst1}`		
`g2: {inst2}`	`A1_g2: {A1/inst5}`	`A2_g2: {A2/inst5}`

Example 3: Master-Clone and nested partitions with nested and cross partition groups

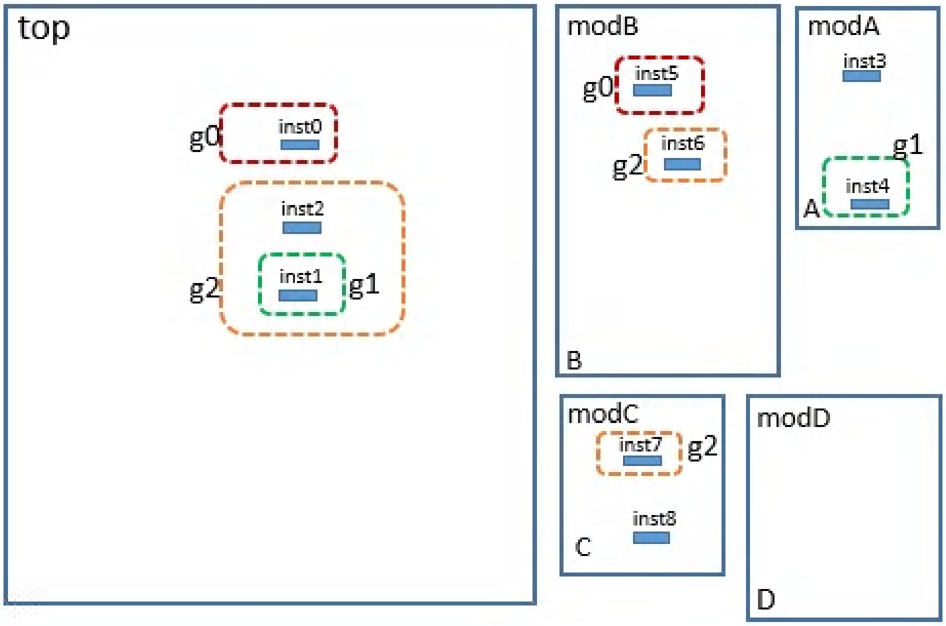
The following is an example of master-clone and nested partitions with nested and cross partition groups.



The instance groups are:

```
Group:Insts
g0: {inst0 B/inst5}
g1: {inst1 A/inst4}
g2: {inst2 B/inst6 B/C/inst7 g1}
```

After partition, the individual partition level groups and their members are represented as:



After flattening the partition or assembling the partition back at chip, the instance groups are named uniquely where flattened inst names are prefixed to created group names.

Updated instance groups:

`g0: {inst0}`	`A_g1: {A/inst4}`	`B_C_g2: {B/C/inst7}`
`g1: {inst1}`	`B_g0: {B/inst5}`	`D_C_g2: {D/C/inst7}`
`g2: {ins2 g1}`	`B_g2: {B/inst6}`	

Timing Budgeting

- Related Commands

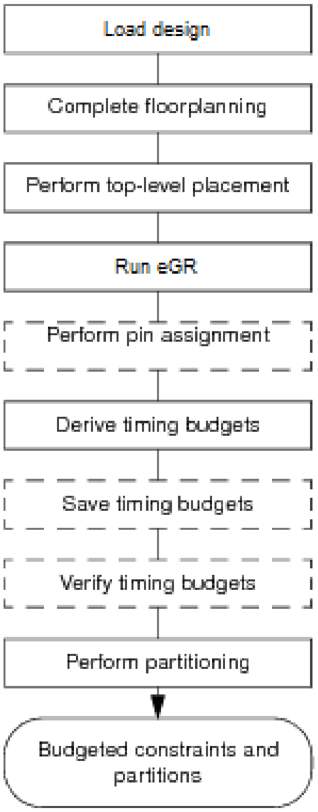
Overview

In hierarchical design flows, chip-level timing constraints must be mapped correctly to corresponding block-level constraints. The Innovus™ Implementation System (Innovus) software does this automatically to produce predictable timing convergence.

The software apportions budgets to blocks using a path-based method, which might not have a direct relationship to the size of the blocks themselves.

Timing budgeting is run per-view, and generates view files for partition implementation and top-level implementation (not for chip assembly).

The following flowchart shows how timing budgeting is performed.



Note: You can set the analysis view at any time before performing timing analysis.

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.

> ⓘ Do not connect the ports of the same partition directly through Verilog assign statements. Timing budgeting requires a physical pin inside a partition to budget timing path. This can be checked in the design by running:
>
> ```
> check_design -type budget
> ```
>
> Use `remove_assigns -buffering` before deriving timing budget to fix this issue.

Deriving Timing Budgets

You can generate timing budget constraint files for each top-level partition using either the partition graphical user interface (GUI) or the various text commands.

Budgeting Using the GUI

To generate the constraints files using the GUI, complete the following steps:

1. Read in the pin assigned data having timing constraint file during design import.
2. Derive timing budgets. Select *Derive Timing Budget* from the *Partition* menu, and specify the options you need on the Derive Timing Budget form.
3. (Optional) Save timing budgets. Select *Design - Save - Save Timing Budget*.
4. Partition the design. Select *Partition - Partition*.
5. *Save the Partition* data. Select *Design - Save - Partition* to save the top and partitions data in

a directory

This directory has sub-directories *`topCellName`* for the top-level and *`partitionName`* for the partitions.

For each partition, this procedure creates a timing constraint file named *`partitionName`*`.constr.pt`. These files are located in each partition directory. The library model files, *`partitionName`*`.lib`, are created in the top-level directory. The constraints file `top_level.top.constr` is created for the top level.

The result is budgeted constraints and partitions.

Budgeting Using Text Commands

To generate the constraints files using the text commands, complete the following steps:

1. Load pin assigned database having timing data, using source pin_assign.enc command.
2. Derive the timing budgets using the `deriveTimingBudget` command.
3. Partition the design using `partition` commands.
4. Save the partition data using `savePartition -dir` *`dir_name`* command.

Note: For the Integrated Hierarchical Database (iHDB) flow, save partitions with a specified module tag (`savePartition -module_model_tag`).

The result is budgeted constraints and partitions.

Note: The `deriveTimingBudget` command does not generate a timing graph, if it already exists. Additionally, user will have to provide a timing graph generated without using `deriveTimingBudget` and must set the value of `-virtualOptEngine` same as that of the command used to generate the timing graph.

Deriving Preliminary Budgets in Early Design Phase

The software provides a flow for deriving timing budgets in the early stages of the design to obtain a preliminary estimate of the budgets. It uses the net delay and net load that you specify using the `setBudgetingMode` command. To perform the preliminary budgeting, you use the `-preliminary` parameter of the `deriveTimingBudget` command. The software does not use virtual optimization for calculating budgets.

The software uses the top-level net delays that you specify using the `-topLevelDelayPerLen` and `-topLevelMinDelayPerNet` parameters of the `setBudgetingMode` command. It calculates the total delay value using the value of the `-topLevelDelayPerLen` parameter and the total length of the net. If the total delay value that the software calculates is less than the `-topLevelMinDelayPerNet` parameter, it uses the value of the `-topLevelMinDelayPerNet` parameter. Otherwise it uses the value of the `-topLevelDelayPerLen` parameter. It assumes the block-level delays are zero.

The software uses the lump capacitance that you specify using the `setBudgetingMode -overrideNetCap` command for both top-level and block-level nets. It uses this value to calculate the cell delay.

It assumes all the net delay is on the top-level and does not perform boundary net adjustments. Therefore, when you run the`justifyBudget` command after generating the preliminary budgets, the Adjustment by Net Delay value is zero in the budgeting report.

The software proportions the budget according to the calculated values.If you do not use the `-preliminary` parameter in the `deriveTimingBudget` command, it adds three extra buffer delays to the delay of the positively slacked path. 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

To perform preliminary budgeting, complete the following steps:

1. Load the hierarchical design, having timing data and pin assigned, in the database.

```
source init.enc
```

2. Set the delay values to be used during budgeting.

```
setBudgetingMode -topLevelDelayPerLen value
-topLevelMinDelayPerNet value
-overrideNetCap value
```

3. Derive timing budgets.

```
deriveTimingBudget -preliminary
```

4. Verify the budgets.

```
justifyBudget
```

Calculating Timing Budgets

Innovus proportions timing budget for partitions based on the path segment length, with a slight difference in calculation when the slack on a path is positive or negative.

For paths with negative slack, the proportioning formula for a setup check (max budgeting) is:
SD/TD * *AT* = *BB*(neg)
For paths with negative slack, the proportioning formula for a hold check (min budgeting) is:
SD/TD * (*AT* + *HT*)= *BB*(neg)

Note: If *AT* + *HT* is less then zero, the software does not use the proportioned value. The software uses the timing analysis values for input or output delays.

For paths with positive slack, the proportioning formula for a setup check (max budgeting) is:
SD + *SD/TD* * *PS* = *BB* (pos)
For paths with positive slack, the proportioning formula for a hold check (min budgeting) is:
SD - *SD/TD* * *PS* = *BB* (pos)

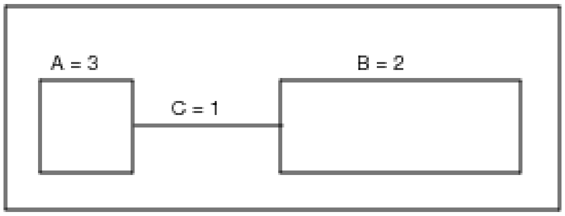
where:

- *SD* is the delay through a path segment.
- *TD* is the total delay of the path.
- *AT* is the total available time. This could be the number of clock periods for multicycle paths, or the clock period minus the fixed delays.
- *HT*is the hold time.

  Note: For max budgeting, hold time is not same as setup time. Setup time is represented as an extra delay for the path. On the other hand, hold time is equivalent to required time, that is the amount of time a signal cannot change.
- *BB* is the baseline budget
- *PS* is the positive slack

Note: For a positively slacked path, budgeting adds virtual buffer delays to the path. The software usually adds three virtual buffer delays. In case of abutted designs, budgeting adds two virtual buffer delays. In case of feedthrough paths, budgeting distributes three buffer delay through all segments of the path.

Example 26-1 Negatively Slacked Path

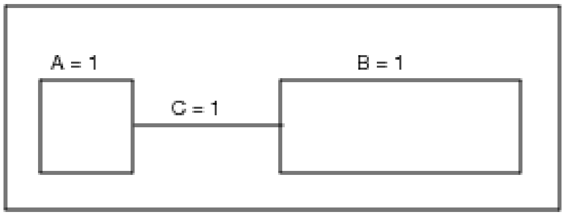


In this example, block A is connected to block B via top-level net C. The budget of the top-level net is not fixed. When placed and routed, the path segment through block A needs 3 ns, path segment through block B needs 2 ns, and net C requires 1 ns. The available time to be budgeted is 5 ns.

The software calculates the following values:

```
Budget for block A = 3/6 * 5 = 2.5 ns
Budget for block B = 2/6 * 5 = 1.67 ns
Budget for net C = 1/6 * 5 = 0.83 ns
Output delay at A = Budget for block B + Budget for net C = 2.5
Input delay at B = Budgets for blocks A + Budget for net C = 3.33
```

Example 26-2 Positively Slacked Path



In this example, the path segment through blocks A and B, and net C require 1 ns each. The total delay is 3 ns. The total available budget is 5 ns. Therefore, positive slack is 2 ns.

The software calculates the following budget values:

```
Budget for A, B, and C = 1 + 1/3 * 2 = 1.66 ns
```

Master Clone Budgeting

Master-clone partitioning/budgeting involves a trade-off between design implementation time and optimized timing for the complete design. When a master is replicated as multiple clones, a single partition is created that represents the worst of all the partitions for each pin and clock. Master-clone budgeting provides you with the flexibility to generate a single unified timing budget constraint file and timing model (.lib) for repeated partitioned modules referred to as master and clones.The data checked for includes:

- **Physical characteristics** - The following physical characteristics are compared for each interface pin across all partitions: drive resistance, internal capacitance, and external capacitance. The merged physical data includes the electrical data that contributes to the timing of every pin of the instance. When the merged data is stored, it stores the largest data for each pin. For example, drive resistance of the master for a pin = largest of the drive resistances for the pin in instances.
- **Timing characteristics** - The following timing characteristics are compared for each interface pin across all partitions: network latencies, user delay in each pin, and slack in each pin for

each clock arriving at it based on input delay and output delay. The merged timing characteristics require comparison of the slack at a pin due to each clock arriving at it across instances and saving the worst among them. For example, the clock `Clk` 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 is 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 -masterClonebudget_master_clone {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 under-constraining 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`.

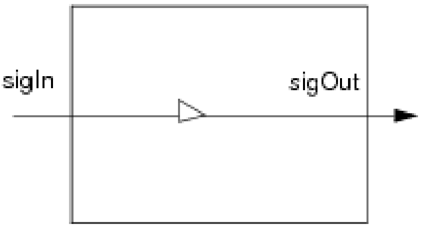
  ```
  max(SIDm, SIDc1, SIDc2, ….),
  max(SODm, SODc1, SODc2, ….)
  ```

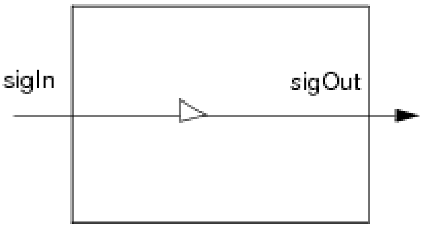
  For example, the input delay value of 2.0 is stored for the master and the value of 3.0 and 4.0 is stored for the two clones, respectively:

  ```
  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`:

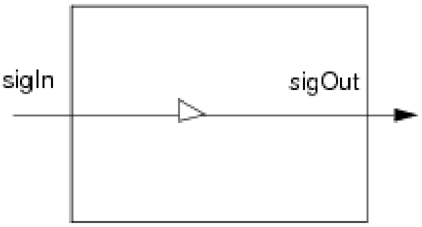
```
set_input_delay 3 sigIn
set_output_delay 1 sigOut
```

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`:

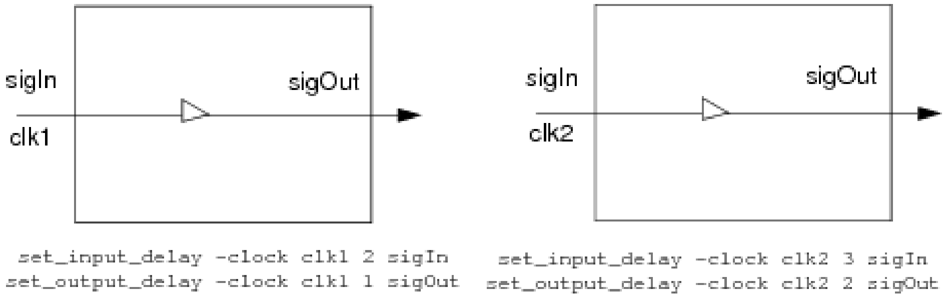
```
set_input_delay 3 sigIn
set_output_delay 2 sigOut
```

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`.



```
set_input_delay -clock clk1 2 sigIn
set_output_delay -clock clk1 1 sigOut
```

```
set_input_delay -clock clk2 3 sigIn
set_output_delay -clock clk2 2 sigOut
```

To eliminate this problem, use false paths. The following example illustrates how SDC is being generated by using `set_false_path`:

```
set_input_delay -clock clk1 2 sigIn
set_input_delay -clock clk2 3 sigIn
set_output_delay -clock clk1 1 sigOut
set_output_delay -clock clk2 2 sigOut
set_false_path -from clk1 -through sigIn -through sigOut -to clk2
set_false_path -from clk2 -through sigIn -through sigOut -to clk1
```

Constraints Adjustment

The timing budgeting process produces a `.lib` file for a partition that will be used during top-level implementation, and a SDC file for partition implementation. When top-level and block-level implementations are run in parallel, the timing model and the SDC files must match in order for chip assembly to succeed.

To ensure that the files match, timing budgeting makes adjustments for the following constraints:

- Capacitance
  For each partition input pin, the tool produces the following output:
- In the `.lib` file, a specification of the pin's capacitance.
- In the SDC constraint file, a `set_max_capacitance` constraint.
  If a `max_capacitance` constraint in the SDC file is greater than the capacitance specified in the `.lib` file, this could lead to timing violations during the reassembly. The partition optimization might change the load of a partition input pin to a value such that the buffer, chosen at the top level with respect to the small capacitance specified in the `.lib` file, would not be able to drive the load.

  The correlation adjustment done by budgeting ensures that the `[pin_capacitance` specification in the `.lib` file and the `set_max_capacitance` constraint in the partition SDC to be very nearly the same.
- Transition
  For each partition output pin, the tool produces the following output:
- In the `.lib` file, a lookup table describing the pin transitions with respect to load.
- In the SDC constraint file, a `set_max_transition` constraint.
  If the `.lib` lookup table indicates a range of transition values that are all less than the `set_max_transition` value used to constrain partition implementation, it could be possible for you to perform top-level implementation assuming that the transition will be, for example, 500 ps, while the partition implementation can pass with a transition of 1 ns on the same port. This situation could result in problems after reassembly.

  The correlation adjustment done by budgeting ensures that the `set_max_transition` constraint in the partition SDC is within the lookup table in the partition `.lib`.

- Load and max_cap

  For each partition output pin, the tool produces the following output:
- In the .lib file, a max_capacitance DRV for the pin.
- In the SDC constraint file, a set_load constraint.

  A max_capacitance constraint in the .lib greater than the set_load constraint in the SDC can lead to timing violations during reassembly. The top-level optimization might change the load of the partition output pin to an unrealistic value for the buffer implemented within the partition, and chosen with respect to the small set_load constraint.

  The correlation adjustment done by budgeting ensures that the max_capacitance in the .lib file and the set_load constraint in the partition SDC file are nearly the same.

Analyzing Timing Budgets

To analyze timing budgets, you must first identify all the boundary pins of the partitions. For each partition pin, the software generates timing constraints in the form of timing set_input_delay or set_output_delay if the pin is an input or output pin, respectively. The software divides the total available budget among all partitions involved, where their boundary pins constitute part of the path.

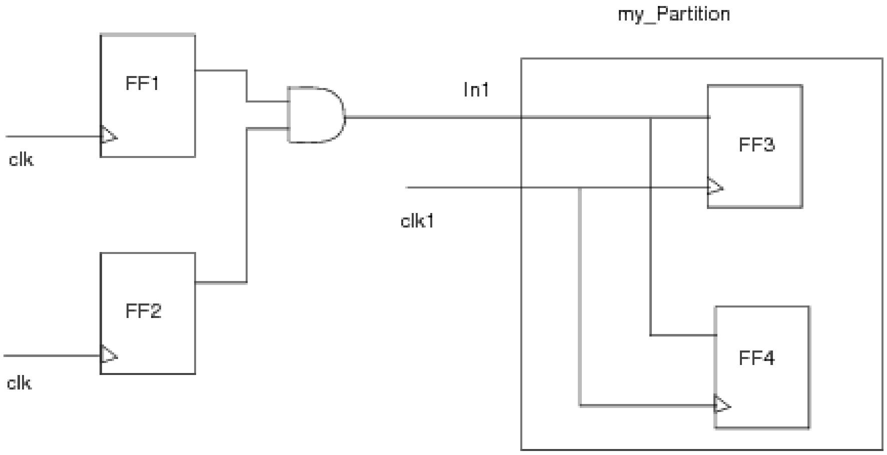
A pin might have multiple paths going through it. Multiple paths through the same port are handled by CTE budgeting. In case of multiple paths related to the same clocks and the same number of clock cycles, the tool automatically chooses the best path for deriving the budgets.

Resolving Conflicts with Path-Based Exceptions

Budgeting generates one input or output delay assertion for each group of paths controlled by the same group of path-based exceptions. For set_input_delay generation at a partition port, the path group is the union of paths originating from the same clock phase that is controlled by the same group of exceptions at the partition port. For set_output_delay generation, the path group is the union of paths with the same clock phase at the end point that is controlled by the same set of exceptions traversing backward at the partition port.

Examples

All of the following examples use the same design:



Case 1

ONE virtual clock (along with one original clock) will be created for each same group of path-based exceptions during budgeting.

- Chip-level exceptions:

  ```
  set_multicyle_path 2 -setup -from FF1 -to my_partition/FF3/D
  ```

- For single cycles path and `clk`:

  `set_input_delay -clock clk` *`number`* `In1`

  (based on single path through `In1`)

- For multicycle path and `clk`:

  `set_input_delay -clock clk_v0` *`number`* `In1`

  (based on worst path from `FF1`)

  ```
  set_multicycle_path 2 -from clk_v0 -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 clk_v0` *`number`* `In1`

  (based on path from `FF1` to my_partition)

  `set_multicycle_path 2 -setup -from clk_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 clk_v0` *`number`* `In1`

  (based on worst path from `FF2`)

  `set_multicycle_path 2 -from clk_v0 -through in1 -setup`

Case 3

Two virtual clocks are created.

- Chip-level exceptions:

  ```
  set_mulicycle_path 3 -setup -from FF1 -to my_partition/FF3/D
  set_mulicycle_path 2 -setup -from FF2 -to my_partition/FF4/D
  ```

- For multicycle 3 path and clk:

  `set_input_delay -clock clk_v0` *`number`* `In1`

  (based on worst of paths from `FF1`)

  `set_multicycle_path 3 -from clk_v0 -through in1 -setup -to FF3/D`

- For multicycle 2 path and `clk`:

  `set_input_delay -clock clk_v1` *`number`* `In1`

  (Based on worst path form `FF2`)

  `set_mulicycle_path 2 -from clk_v1 -through in1 -setup -to FF3/D`

Case 4

Two virtual clocks are created:

- Chip-level exceptions:

  ```
  set_mulicycle_path 2 -setup -from FF1 -to my_partition/FF4/D
  set_false_path -from FF1 -to my_partition/FF3/D
  set_mulicycle_path 2 -setup -from FF2
  ```

- For multicycle 2 path from `FF1`:

  `set_input_delay -clock clk1_v0` *`number`* `In1`

  (based on path from FF1 to my_partition/FF4)

  ```
  set_mulicycle_path 2 -setup -from clk1_v0 -through In1 -to FF4/D
  ```

- For false path from `F1`:

  ```
  set_false_path -from clk1_v0 -through in1 -to FF3/D
  ```

- For multicycle 2 path from `FF2`:

  `set_input_delay -clock clk2_v0` *`number`* `In1`

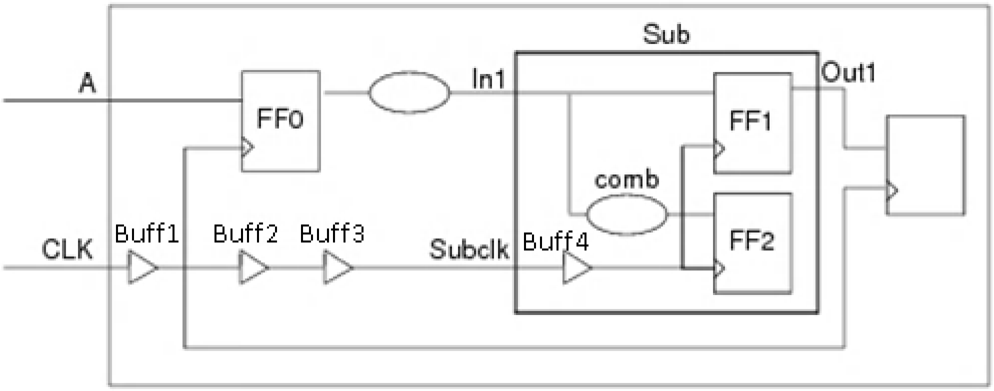
  (based on worst of paths from `FF2`)

  ```
  set_multicycle_path 2 -from clk2_v0 -through in1 -setup
  ```

Budgeting Clock Latency in Propagated Mode

Innovus includes clock latency in the constraints generated for clocks in propagated mode. The clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The clock latency is added to `set_input_delay` and subtracted from the `set_output_delay`. This feature is useful when a clock tree is present in your design.

For multiple paths, both source and propagated clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The software adds the `-source_latency_included` and `-network_latency_included` constraints in the `set_input_delay` and `set_output_delay` constraints for all inputs and outputs related to clocks in propagated mode. Consider the following figure.



The `deriveTimingBudget` command result in the following constraints for the `Sub` partition:

```
create_clock -clock subclk -waveform...
set_clock_latency -source (top_source + delay Buff1 + delay Buff2 + delay Buff3 )
subclk
set_input_delay -clock subclk (Input delay (with skew) + top_source + delay Buff1)
    -source_latency_included -network_latency_included In1
set_output_delay -clock subclk (output_delay (with skew) - top_source - delay Buff1)
    -source_latency_included -network_latency_included Out1
```

Where,

`top_source` = source latency of the clock at the top level`delay Buff*` = delay through each buffer in the clock network

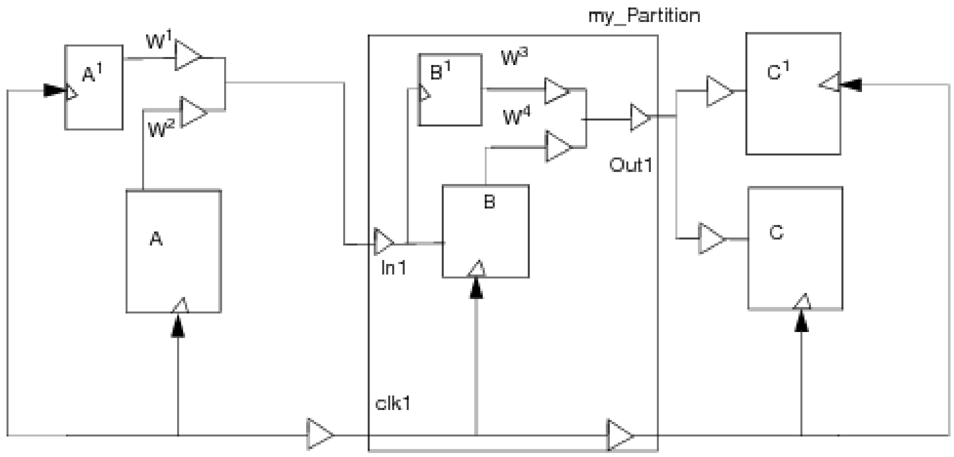
Budgeting Libraries

To enable design analysis at the top level, budgeting generates black box models of the partitions in the `.lib` format.

Resolving Conflicts with Path-based Exceptions

Budgeting generates internal pins and combinational arcs to model the budgets of each group of paths controlled by the same group of path-based exceptions. For input ports, the path group is the union of paths with the same clock phase at the end point that is controlled by the same set of exceptions traversing backwards towards the input port. For output ports, the path group is the union of paths originating from the same clock phase that is controlled by the same group of exceptions at the partition port.

Case 1: A single cycle path and a multicycle path through the partition port

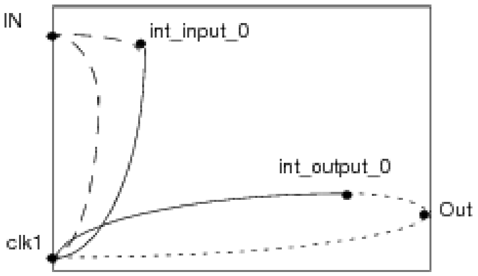


- Chip-level exceptions:

  set_multicycle_path 2 -from A^1/ck -to my_Partition/B/D

  set_multicycle_path 2 -from my_Partition/B/ck -to C^1/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`.

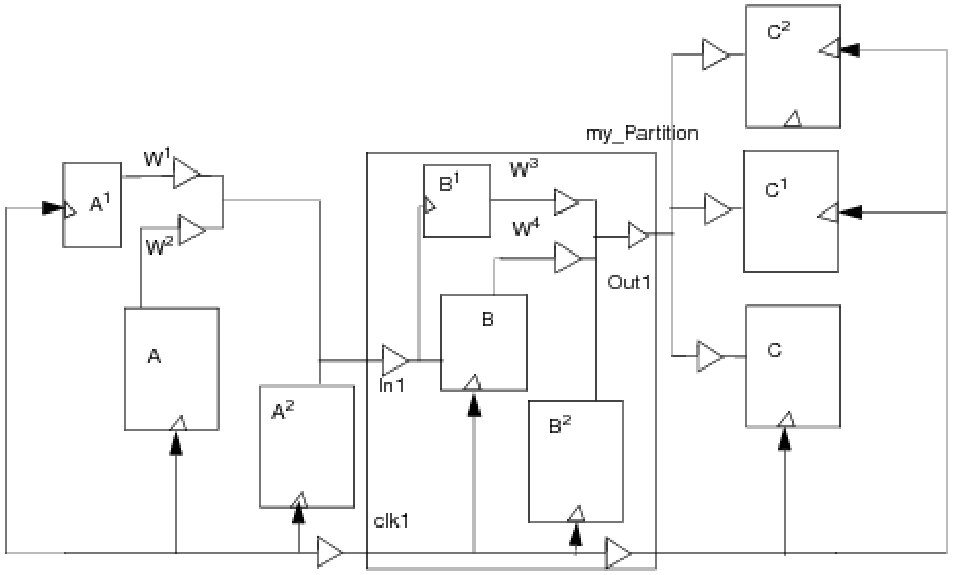


- Top-level exceptions:

  set_multicycle_path 2 -setup -from [list [get_pins {A^1/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 {C^1/D}]]

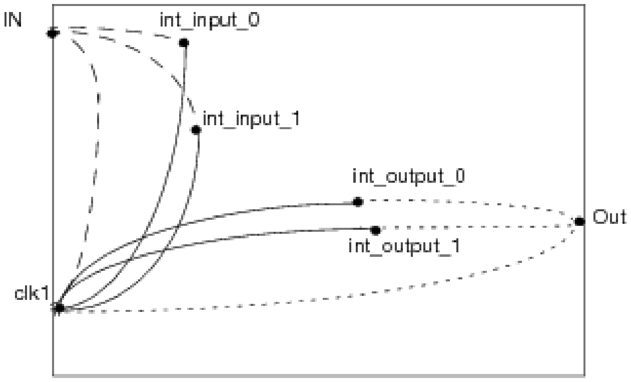
Case 2: A single cycle path and two different multicycle paths through the partition port



- 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.

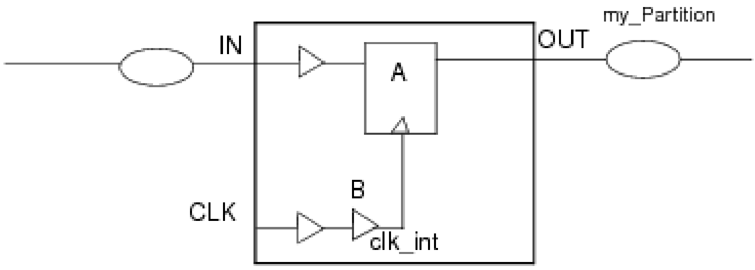


- Top-level exceptions:

```
set_multicycle_path 2 -setup -from [list [get_pins {A$^1$/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 {C$^1$/D}]]
set_multicycle_path 4 -setup -from [list [get_pins {A/CK}]] -through [list
[get_pins {my_partition/in}] ] -to [list [get_pins {my_partition/int_input_1}] ]
set_multicycle_path 4 -setup -through [list [get_pins {my_partition/int_output_0}]
] -through [list [get_pins {my_partition/out}] ] -to [list [get_pins {C/D}]]
```

Defining Clocks Inside the Partition

Budgeting generates additional internal pins to model the effect of the clocks defined inside the partition at the top level.

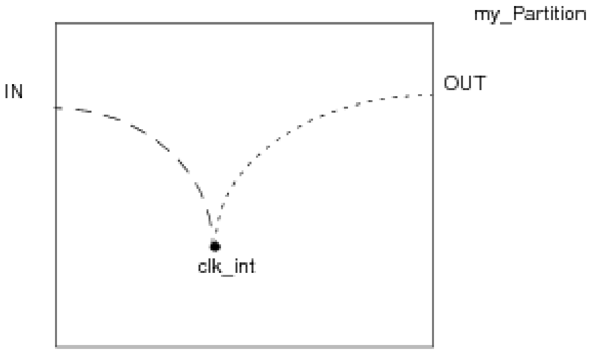


- Chip-level constraint:

```
create_clock -name clk -period 10 [get_pins {my_Partition/B/clk_int}]
```

For the above constraint, an internal pin is created in the partition. At the top level, a constraint is generated for this internal pin.

In the following `.lib` representation, the `B/clk_int` pin is created, and sequential and check arcs are defined with the internal pin.



- Top-level constraint:

```
create_clock -name clk -period 10 [get_pins {my_Partition/B/clk_int}]
```

Customizing Budget Generation

You can customize budget generation according to the design stage and timing requirements. To customize budget generation, use the following commands in Innovus:

- The `-freezeATopBLevelbudget_fix_top_level` 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 `-freezeATopBLevelbudget_fix_top_level` parameter might not make much difference in the generated budgets.
- The `setBudgetingMode -ignoreDontTouch` command is used to consider `don't_touch` blocks. The `-ignoreDontTouch` parameter does not consider `dont_touch` as fixed delay. The budgeting results change based on whether fixed delay is considered during trial IPO.
- At the top level, you can set the `set_input_delay` and `set_output_delay` constraints on the hierarchical ports (or partition ports). The software generates budgets for the hierarchical ports based on the set constraints.

Fixing Budget

You can fix budget for a particular path segment between partitions and start/end points only for the following segments:

- Chip In -> Partition In
- Start Point -> Partition In
- Partition In -> Partition Out
- Partition In -> End Point (FF D Pin) inside same partition
- Start Point (FF Q Pin) inside same partition -> Partition Out
- Partition Out -> Chip Out
- Partition Out -> End Point

To fix budget for a path segment, use the `setFixedBudget` command in Innovus.

Recommendations for Fixing Budget

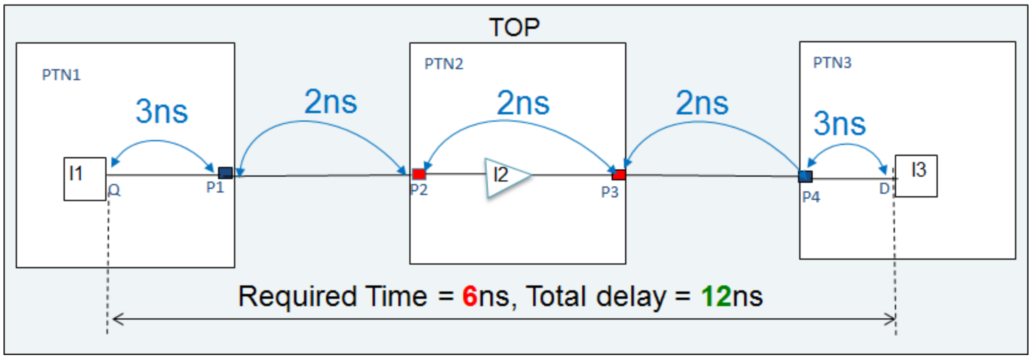
Following are the recommendations for fixing budget:

- To perfectly control the delay being distributed in RAK design, “`-virtualOptEngine {none}`” is used.
  - This avoid changing of delays inside a partition based on virtual buffers inserted by virtual optimization engine is called by `deriveTimingBudget` internally.
- Examples: `setFixedBudget -from P5/in -to P5/out -slack 0`
  - Delay of this segment in RAK design is =0.284
  - With virtual optimization more buffers can get added and delay values may change, hence changing the budgets for each of the partitions (we can’t accurately predict the number of buffers and hence the delay numbers are unknown before virtual optimization is done).
- Example: `setFixedBudget -from P5/in -to P5/out -delay .284`
  - This can fix the delay of partition ptn_fd and override the effects of optimization
  - However results may be inaccurate as paths are not optimized for delays/DRVs (partition delay value can be retrieved by running `timeDesign -preCTS`)
- For channel based design (top portion of net has some delays), top segments are also considered for slack distribution. `setFixedBudget` should be applied to any segment whose slack distribution needs to be controlled (including top segments discussed in the statement)
- Set “`-bufferDelayAdjustment 0`” to prevent `deriveTimingBudget` from automatically adding buffer delays to partition for whom we are fixing the budgets (may lead to unexpected results).

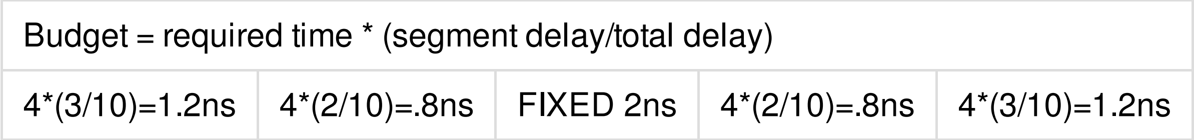
Fix Budget Example

- Generated budget constraints with required time is 6ns and total path delay is 12ns.

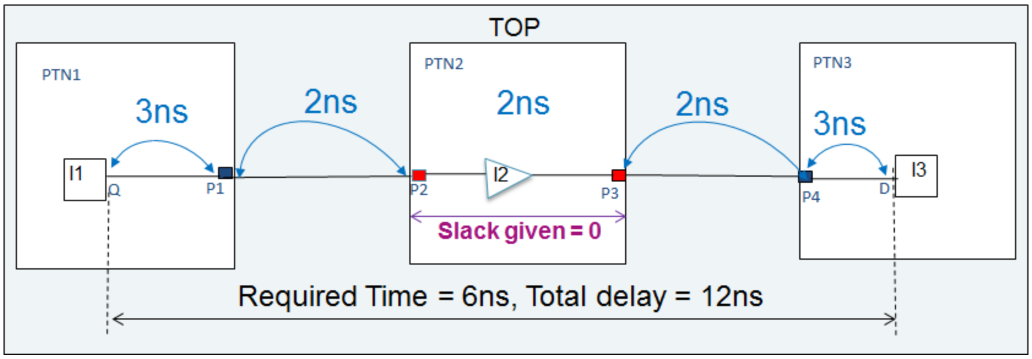
Budget = required time * (segment delay/total delay)				
6*(3/12)=1.5ns	6*(2/12)=1ns	6*(2/12)=1ns	6*(2/12)=1ns	6*(3/12)=1.5ns



- `setFixedBudget -from PTN2/P2 -to PTN2/P3 -slack 0`



Budget = required time * (segment delay/total delay)				
4*(3/10)=1.2ns	4*(2/10)=.8ns	FIXED 2ns	4*(2/10)=.8ns	4*(3/10)=1.2ns



- No slack will be distributed to segment 3, delay will remain equal to segment delay = 2ns
- For other segments
  - New required time is 6-2=4ns
  - New total delay is 12-2=10ns

Modifying Budgets

You can modify budgets after they have been saved using the `saveTimingBudget` command. To modify budgets, you must create an input budget file containing constraints that look like an SDC file. It only supports the `set_input_delay` and `set_output_delay` constraints, as shown below:

```
set_input_delay 1.5 -clock [get_clocks {clk1_setuphold}] -max -fall [get_ports
{sub_in}] -add_delay
set_input_delay 1.9 -clock [get_clocks {clk1_setuphold}] -max -rise [get_ports
{sub_in}] -add_delay
```

The clock names in these constraints are same as the 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.

A sample Tcl script is given below:

```
setBudgetingMode -virtualOptEngine none
deriveTimingBudget -justify
saveTimingBudget -dir Budget
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 `-freezeATopBLevelbudget_fix_top_level` is used, any snapped delays calculated by using `setBudgetingMode -snapFdBudgetTo` or `-snapInputBudgetTo`. 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` 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 `-topALevelbudget_top_level` 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`.
- `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` and if external segment path delay is below user defined threshold.
- `Clock Skew`

  The path clock skew.

Note: The report precision (the number of digits printed after the decimal point) is `3`.

Design Example

```
module Top(in1, clk1, clk2, out);
input in1;
input clk1;
output out;
input clk2;
wire c0, c1, c2;

bfx0 buf0(.A(in1), .Z(c0));
SUB     i_sub1(.sub_in(c0),
.sub_clk(clk1),
.sub_out(c1));

bfx0 buf1(.A(c1), .Z(c2));
```

```
SUBn     i_sub2(.sub_in(c2),
.sub_clk(clk2),
.sub_out(out));

endmodule // TOP

module SUB (sub_in, sub_clk, sub_out);
output sub_out;
input sub_in;
input sub_clk;
df1qx1 sub_FF (.D(sub_in), .CP(sub_clk), .Q(sub_out));

endmodule // SUB

module SUBn (sub_in, sub_clk, sub_out);
output sub_out;
input sub_in;
input sub_clk;
df1qx1 sub_FF (.D(sub_in), .CP(sub_clk), .Q(sub_out));

endmodule // SUBn
```

SDC Constraints for Design Example

```
current_design Top
create_clock -name clk1 -period 1 -waveform {0 0.5} [get_ports {clk1}]
set_input_delay 0.2 -clock clk1 [get_ports {in1}]
set_multicycle_path 2 -from [get_pins {i_sub1/sub_FF/CP} ] -to [get_pins
{i_sub2/sub_FF/D}]

create_clock -name clk2 -period 1 -waveform {0 0.5} [get_ports {clk2}]
set_output_delay 0.1 -clock clk2 [get_ports {out}]
```

Generated Report for Design Example

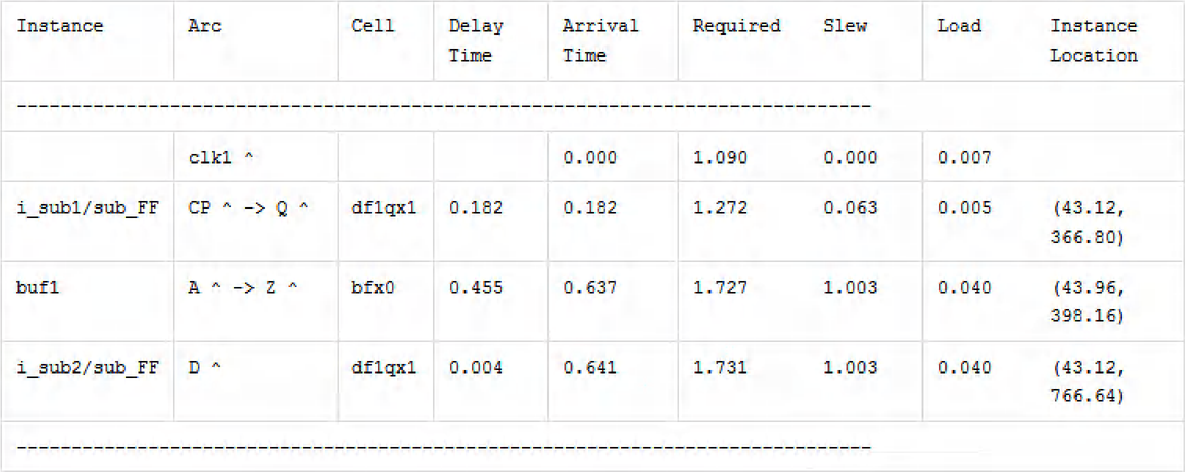
To validate the budgets with positive slack in the design example, "Design Example", type the following command:

```
justifyBudget -inst i_sub2 -pin sub_in
```

The following report was generated:

```
HInstance: i_sub2
Port: sub_in
Budgeted constraint type: set_input_delay(setup rise)
Virtual Clock: clk1_V0
Initial budget available time + clock skew = 2.000

One Buffer Delay for Adjustment(cell bfx2): 0.198
Fixed Delay for Feedthrough Paths(fixFdThru)= 0.000
External Segment Fixed Delay From Budget Snap(snapExtFixedDel) = 0.000
Total External Segment Fixed Delay(extFixDel) = 0.000
This Block's Segment Fixed Delay from budget snap(snapIntFixedDel) = 0.000
Total This Block's Segment Fixed Delay(intFixDel) = 0.000
External Segment Extra Delay From Budget Snap (snapExtDelExtra) = 0.000
This Block's Extra Delay From Budget Snap (snapIntDelExtra) = 0.000
Path Extra Delay From Budget Snap (snapExtraDel) = (0.000 + 0.000) = 0.000
Fixed Delay on the Path(pathFixDel) = (0.000 + 0.000 + 0.000 + 0.000) = 0.000
Fixed Delay Adjustment(fixDel)= 0.000
Clock Skew(clkSkew): 0.000

Adjustment for budget available time= -(pathFixDel + fixFdThru - clkSkew +
snapExtraDel)
= -(0.000 + 0.000 - 0.000 + 0.000) = -0.000
Available budget after adjustments(AvailTime)= (2.000 - 0.000) = 2.000

External Segment Delay(extSegDel): 0.638
This Block's Segment Delay(segDel): 0.273
Total delay(totDel): 0.638 + 0.273 = 0.910
Initial Slack = AvailTime - totDel
Initial Slack = 2.000 - 0.910 = 1.090
Virtual Buffering Adjustment: (3 x 0.198) = 0.594
Slack after Virtual Buffering Adjustment(slack): 1.090 - 0.594 = 0.496

External Virtual Buffering Adjustment(extVirBuf)= 0.396
Top Level Adjustment(topLev): 0.000
Virtual Clock Adjustment(virClk): 0.000
RC Adjustment(RC): 0.009
Budgeted constraint = extSegDel + slack * extSegDel / totDel + extVirBuf + topLev +
fixDel + virClk + startClkLat - RC
Budgeted constraint = 0.638 + 0.496 * 0.638 / 0.910 + 0.396 + 0.000 + 0.000 + 0.000 +
0.000 - 0.009 = 1.372

Path 1: MET Setup Check with Pin i_sub2/sub_FF/CP
Endpoint: i_sub2/sub_FF/D (^) checked with rising edge of 'clk2'
Beginpoint: i_sub1/sub_FF/Q (^) triggered by rising edge of 'clk1'
Other End Arrival Time0.000
```

```
- Setup0.269
+ Phase Shift1.000
+ Cycle Adjustment1.000
= Required Time1.731
- Arrival Time0.641
= Slack Time1.090
Clock Rise Edge0.000
= Beginpoint Arrival Time0.000
```



Instance	Arc	Cell	Delay Time	Arrival Time	Required	Slew	Load	Instance Location
--								
	clk1 ^			0.000	1.090	0.000	0.007	
i_sub1/sub_FF	CP ^ -> Q ^	dflqx1	0.182	0.182	1.272	0.063	0.005	(43.12, 366.80)
buf1	A ^ -> Z ^	bfx0	0.455	0.637	1.727	1.003	0.040	(43.96, 398.16)
i_sub2/sub_FF	D ^	dflqx1	0.004	0.641	1.731	1.003	0.040	(43.12, 766.64)
--								

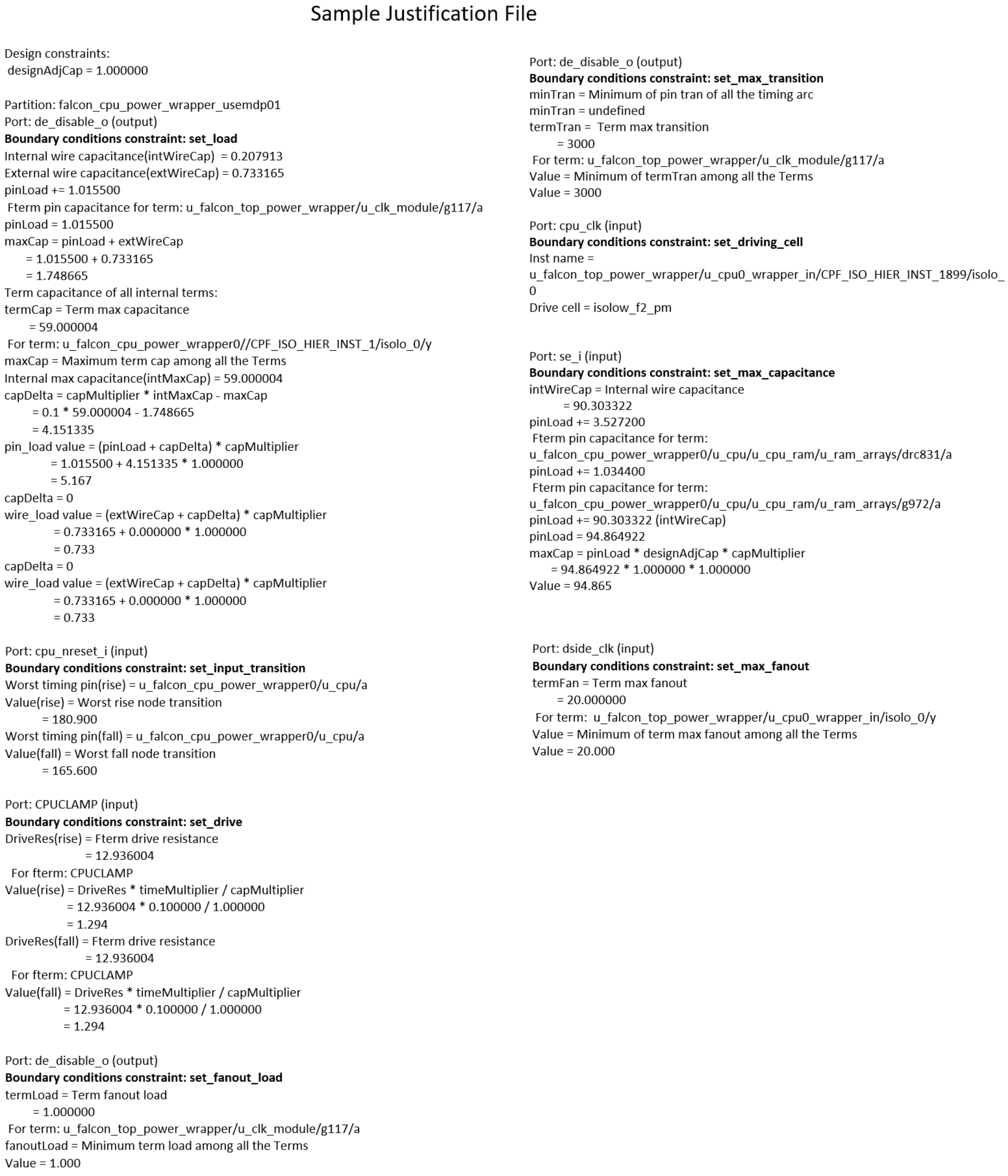
Note: The `justifyBudget` command honors the `report_timing_format` global, which allows you to customize the timing report according to the user-specified columns. You must set this global before specifying `justifyBudget` or `deriveTimingBudget -justify` to get the report in the desired format.

Dumping Justification Files for Setting Boundary Conditions Example

To dump justification files for setting boundary conditions, run the following command:

```
setBudgetingMode -justifyBoundaryCondition true
```

Following is a sample justification file set by the above command:



Sample Justification File

```
Design constraints:
 designAdjCap = 1.000000

Partition: falcon_cpu_power_wrapper_usemdp01
Port: de_disable_o (output)
Boundary conditions constraint: set_load
Internal wire capacitance(intWireCap)  = 0.207913
External wire capacitance(extWireCap) = 0.733165
pinLoad += 1.015500
 Fterm pin capacitance for term: u_falcon_top_power_wrapper/u_clk_module/g117/a
pinLoad = 1.015500
maxCap = pinLoad + extWireCap
      = 1.015500 + 0.733165
      = 1.748665
Term capacitance of all internal terms:
termCap = Term max capacitance
       = 59.000004
 For term: u_falcon_cpu_power_wrapper0//CPF_ISO_HIER_INST_1/isolo_0/y
maxCap = Maximum term cap among all the Terms
Internal max capacitance(intMaxCap) = 59.000004
capDelta = capMultiplier * intMaxCap - maxCap
        = 0.1 * 59.000004 - 1.748665
        = 4.151335
pin_load value = (pinLoad + capDelta) * capMultiplier
             = 1.015500 + 4.151335 * 1.000000
             = 5.167
capDelta = 0
wire_load value = (extWireCap + capDelta) * capMultiplier
              = 0.733165 + 0.000000 * 1.000000
              = 0.733
capDelta = 0
wire_load value = (extWireCap + capDelta) * capMultiplier
              = 0.733165 + 0.000000 * 1.000000
              = 0.733

Port: cpu_nreset_i (input)
Boundary conditions constraint: set_input_transition
Worst timing pin(rise) = u_falcon_cpu_power_wrapper0/u_cpu/a
Value(rise) = Worst rise node transition
          = 180.900
Worst timing pin(fall) = u_falcon_cpu_power_wrapper0/u_cpu/a
Value(fall) = Worst fall node transition
          = 165.600

Port: CPUCLAMP (input)
Boundary conditions constraint: set_drive
DriveRes(rise) = Fterm drive resistance
                   = 12.936004
  For fterm: CPUCLAMP
Value(rise) = DriveRes * timeMultiplier / capMultiplier
              = 12.936004 * 0.100000 / 1.000000
              = 1.294
DriveRes(fall) = Fterm drive resistance
                   = 12.936004
  For fterm: CPUCLAMP
Value(fall) = DriveRes * timeMultiplier / capMultiplier
             = 12.936004 * 0.100000 / 1.000000
             = 1.294

Port: de_disable_o (output)
Boundary conditions constraint: set_fanout_load
termLoad = Term fanout load
        = 1.000000
 For term: u_falcon_top_power_wrapper/u_clk_module/g117/a
fanoutLoad = Minimum term load among all the Terms
Value = 1.000
```

```
Port: de_disable_o (output)
Boundary conditions constraint: set_max_transition
minTran = Minimum of pin tran of all the timing arc
minTran = undefined
termTran =  Term max transition
        = 3000
 For term: u_falcon_top_power_wrapper/u_clk_module/g117/a
Value = Minimum of termTran among all the Terms
Value = 3000

Port: cpu_clk (input)
Boundary conditions constraint: set_driving_cell
Inst name =
u_falcon_top_power_wrapper/u_cpu0_wrapper_in/CPF_ISO_HIER_INST_1899/isolo_
0
Drive cell = isolow_f2_pm


Port: se_i (input)
Boundary conditions constraint: set_max_capacitance
intWireCap = Internal wire capacitance
          = 90.303322
pinLoad += 3.527200
 Fterm pin capacitance for term:
u_falcon_cpu_power_wrapper0/u_cpu/u_cpu_ram/u_ram_arrays/drc831/a
pinLoad += 1.034400
 Fterm pin capacitance for term:
u_falcon_cpu_power_wrapper0/u_cpu/u_cpu_ram/u_ram_arrays/g972/a
pinLoad += 90.303322 (intWireCap)
pinLoad = 94.864922
maxCap = pinLoad * designAdjCap * capMultiplier
      = 94.864922 * 1.000000 * 1.000000
Value = 94.865


Port: dside_clk (input)
Boundary conditions constraint: set_max_fanout
termFan = Term max fanout
       = 20.000000
 For term:  u_falcon_top_power_wrapper/u_cpu0_wrapper_in/isolo_0/y
Value = Minimum of term max fanout among all the Terms
Value = 20.000
```

Generate Summarized Report of Budget Data

You can generate a script friendly summary of what budgets have been produced for a specific port or ports. For reporting the budgeting data for buses, specify the bus name as the pin name. Use the `justifyBudgetreport_timing_budget -summary` command to generate the summarized reports.

The use model is given below:

1. Enable the collection of reporting data, before running `deriveTimingBudget`.
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.

Note: For the Integrated Hierarchical Database (iHDB) flow, save timing budget or partitions with a specified module tag (`saveTimingBudget-module_model_tag` or `savePartition -module_model_tag`).

The output 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 `justifyBudgetreport_timing_budget -summary` command is stored in the output file in the following format for bus pins:

```
Bus: Bus Name

Most Constrained Bit : bit name

SDC Data
Clock: Clock Name
SDC Source Budget(Rise,Fall): rise value, fall value
SDC Destination Budget(Rise,Fall) : rise value, fall value
Exception(s) : shows the exceptions, if any
SDC Warnings: shows the warnings if any
Justify Source Budget(Rise, Fall): rise value, fall value
Justify Destination Budgets(Rise,Fall): rise value, fall value
Justify S lack (Rise, Fall) : rise value, fall value

Least Constrained Bit: bit name

SDC Data
Clock: Clock Name
SDC Source Budget(Rise,Fall): rise value, fall value
SDC Destination Budget(Rise,Fall) : rise value, fall value
Exception(s) : shows the exceptions, if any
SDC Warnings: shows the warnings if any
Justify Source Budget(Rise, Fall): rise value, fall value
Justify Destination Budgets(Rise,Fall): rise value, fall value
Justify S lack (Rise, Fall) : rise value, fall value

Median Constrained Bit: bit name

SDC Data
Clock: Clock Name
SDC Source Budget(Rise,Fall): rise value, fall value
SDC Destination Budget(Rise,Fall) : rise value, fall value
Exception(s) : shows the exceptions, if any
SDC Warnings: shows the warnings if any
Justify Source Budget(Rise, Fall): rise value, fall value
Justify Destination Budgets(Rise,Fall): rise value, fall value
Justify Slack (Rise, Fall) : rise value, fall value
```

```
Average Source Budget (rise,fall in ns) : average budget across all bus bits(rise and
fall)

Average Destination Budget (rise,fall in ns): average budget across all bus

bits(rise and fall)
```

Reading the Justify Exception Report

You can use the `justifyException` command to provide a debugging or justification mechanism of exceptions on ports. This command allows you to debug exception constraints and justify all exceptions for the specified instances or partitions of the design.

To justify exceptions:

1. Set the `-enablejustifyException` parameter of the `setBudgetingMode` command to `true` to save the justify exception data during the `deriveTimingBudget` process.
2. Perform exception justification using one of the following two ways:

- Specify the `deriveTimingBudget` command with the `-justify` parameter.
  When the `-justify` parameter is specified, the justify file generated as `<partition|inst>_<view name>.justify` is appended with the exception information (Applied and Translated Exceptions). If the same view is used as both setup and hold, then the file names are generated as `<partition|inst>_<view name>_<setup|hold>.justify`. These files are saved in their respective partition/inst directory in the `budget_justify` directory by default, if the `setBudgetingMode -justifyBudgetDir` parameter is not specified.

> ⚠ You can run the `deriveTimingBudget` command both with and without the `-justify` parameter. When the `-justify` parameter is not specified, the data is collected only for exception justification, and will neither be written in a file nor will be displayed on screen.

- Specify the `justifyException` command to justify all exceptions on pins specified by `-pins` or `-pin`. If you do not specify an output file using the `-outfile` parameter, the exception data will be displayed on the screen.

Flow example with `deriveTimingBudget -justify`

```
restoreDesign 120.des.dat top
createFence partition 1 2 3 4
```

```
definePartition -hinst partition
setBudgetingMode -enable justifyException true -justifyBudgetDir
budget_justify_with_dtb
deriveTimingBudget -ptn {partition} -justify
```

Flow example without `-justify` but with the `justifyException` command

```
restoreDesign 120.des.dat top
createFence partition 1 2 3 4
definePartition -hinst partition
setBudgetingMode -enable justifyException true -justifyBudgetDir
budget_justify_ptn_with_command
deriveTimingBudget -ptn {partition}
```

`justifyException partition -pin out1 -view default_analysis_view_setup` # this will only print data on screen

`justifyException partition -pin out1 -view default_analysis_view_setup -outfile block_default_analysis_view_setup.jusifyExcp` #this will print data in the specified file

The report generated contains the following fields:

- `Partition`

  Specifies the partition or instance for which the exception justification is done.

- `Port`

  Specifies the partition/instance's port for which the exceptions have been justified.

- `Budgeted constraint type`

  Specifies the delay type applicable on this port. It can be either `set_input_delay` and `set_output_delay`. It also provides the information of setup/hold check for rising/falling edge of clock.

- `Virtual Clock`

  Virtual clock with which setup/hold check is done.

- `Exception`

  Specifies the exception type applied on the port. It can be one of the following:

  - `set_multicycle_path`

  - `set_max_delay`

  - `set_min_delay`

Note: `set_false_path` is not reported because only constrained paths are justified.

- `Applied exceptions`

  Specifies the path exception information picked up from the Common Timing Engine. It is the same information that is printed in the path exception section of the timing report when "`report_timing -path_exception all`" is issued on the path that has been used for budgeting of the port.
- `Translated exception`

  Specifies the full chip exceptions translated to partition/instance.

Design Example

```
module partition (
       in,
       clk,
       out,
       out1);

  input in;
  input clk;
  output out;
  output out1;

  BUFX12 B21 (.A(clk), .Y(o2));
  BUFX12 B22 (.A(o2), .Y(o4));
  BUFX12 B23 (.A(o4), .Y(o5));
  DFFHQX1 D21 (.D(in), .CK(o5), .Q(ox));
  BUFX12 BMC (.A(ox), .Y(out1));
  BUFX12 BMD (.A(ox), .Y(out1));
endmodule

module top (
       in,
       clk,
       out);

  input in;
  input clk;
  output out;

  BUFX12 B1 (.A(in), .Y(s1));
  BUFX12 B2 (.A(clk), .Y(s2));
  partition partition (.in(s1), .clk(s2), .out(s3), .out1(s4));

  BUFX12 B3 (.A(s4), .Y(out));

endmodule
```

SDC Constraints for Design Example

```
current_design top
create_clock -name clock -period 15 [get_ports {clk}]

set_input_delay 2 -clock clock  in
set_output_delay 3 -clock clock out

set_multicycle_path 2 -through partition/BMC/Y
set_multicycle_path 3 -through partition/BMD/Y
```

Generated Report for Design Example

```
Partition: partition
Port: partition/out1
Budgeted constraint type: set_output_delay(setup rise)
Exception: set_multicycle_path
Applied exceptions:
```

Through	Late	View Name
partition/BMC/Y	cycles 2	default_analysis_view_setup

```
Translated exception:
set_multicycle_path 2 -setup -through [list \
 [get_ports {out1}]] -to [list \
 [get_clocks {clock}]]

Translated exception:
set_multicycle_path 3 -setup -through [list \
 [get_ports {out1}]] -to [list \
 [get_clocks {clock}]]

Partition: partition
Port: partition/out1
Budgeted constraint type: set_output_delay(setup fall)
Exception: set_multicycle_path
```

```
Applied exceptions:
```

Through	Late	View Name
partition/BMC/Y	cycles 2	default_analysis_view_setup

```
Translated exception:
set_multicycle_path 2 -setup -through [list \
 [get_ports {out1}]] -to [list \
 [get_clocks {clock}]]

Translated exception:
set_multicycle_path 3 -setup -through [list \
 [get_ports {out1}]] -to [list \
 [get_clocks {clock}]]
```

Support for Distributed Processing in Budgeting

Innovus supports distributed time budgeting for MMMC designs to improve the overall run-time for large designs. In the distributed mode, processing of views are distributed across CPUs of a local machine or multiple machines, and the view data is collated using the `saveTimingBudget` command.

Distributed processing supports two modes:

- **Local Mode** - you can specify the number of CPUs to use on local machine. In this mode, you can distribute views across multiple CPUs of the same machine.

  `setDistributeHost -local`

  `setMultiCpuUsage -localCpu` *`integer`*

- **Remote Mode** - you can specify one or more CPUs to use on network hosts. In this mode, you can distribute views to multiple machines.

  `setDistributeHost -rsh -add {host1 host2 host3}`

  `setMultiCpuUsage -remoteHost` *`integer`*

The use model for distributed MMMC budgeting is as follows:

1. Set up the multi-host environment. Specify the `setMultiCpuUsage` command to set the maximum number of hosts and the `setDistributeHost` command to set the multiple-CPU processing configuration for distributed processing.
2. Specify the `deriveTimingBudget` command. This command internally creates distribution

scripts for all views and runs these scripts on different CPUs/machines per the multi-host environment. The logs and distribution scripts for different views are located in the `.tbMMMCDistributed` directory for later reviews.

3. Specify the `saveTimingBudget` command. This command internally collates the data for multiple views from different runs and saves it to the specified directory.

Note: All parameters of the `saveTimingBudget` command, except `-dir`, are ignored. You can set these parameters using the `setBudgetingMode` command.

Example

```
setMultiCpuUsage -remoteHost 4
setDistributeHost -lsf -queue lnx64 -resource "select[mem>10000 && tmp>400 && swp >1000
&& OSNAME==Linux && SFIARCH==OPT64]
setBudgetingMode -virtualOptEngine none
deriveTimingBudget -inst i_top_0
saveTimingBudget -dir Budget
```

Constraints Support in Budgeting

- `group_path`

  This constraint is supported in timing optimization and timing analysis. In budgeting, it is not pushed-down inside the partition and top-level SDC. This will affect timing budgets, because the constraint affects chip-level timing analysis.

- `create_clock`

  If a top-level clock `CK` is inverted, then while generating the budgets for a partition a new negative clock `CK_B_ENC` is created for the partitions connected to the negative clock.
  For example, if `CK` is defined as:

  ```
  create_clock -name CK -period 7.500 -waveform { 0.000 3.750 } \
  [list [get_ports {clk}] ]
  ```

  The negative clock is:

  ```
  create_clock -name CK_B_ENC -period 7.500 -waveform { 3.750 7.500 } \
  [list [get_ports {losdclko_rp}] ]
  ```

  Where, `losdclk0_rp` is the clock port of the partition.

- `create_generated_clock`
- `set_clock_latency`

  The `set_clock_latency` constraint is generated when you use the `setAnalysisMode -skew true` command. The clock latency is not budgeted between the partitions. The `setAnalysisMode -clockPropagation sdcControl`, along with `set_clock_propagation` constraint, do not cause the network delay through the clock tree to be budgeted for the partitions. The same clock latency is assigned to all the partitions if specified in the top-level clock constraints.
- `set_clock_uncertainty`
- `set_input_delay`
- `set_output_delay`
- `set_input_transition`
- `set_load`
- `set_drive`
- `set_driving_cell`
- `set_max_transition`
- `set_max_capacitance`
- `set_multicycle_path`
- `set_false_path`

  Innovus timing analysis requires that the `set_false_path` and `set_multicycle_path` constraints have valid startpoints and endpoints for the `-from` and `-to` options.

  Valid startpoints are:

  - Input ports
  - Input part of bidirectional ports
  - Clock pins of sequential cells
  - Pins associated with `set_input_delay`
  - Pins associated with `set_path_delay -from`

Valid endpoints are:

  - Output ports

- Output part of bidirectional ports
- Data pins of sequential cells
- Pins associated with `set_output_delay`
- Pins associated with `set_max_delay -to`
  During budgeting, the software generates valid budgets for partitions based on invalid constraints at the top. For example, if `set_multicycle_path 2 -from SUB/IN1` is set at the top level, it is ignored during timing analysis, because a hierarchical pin is not a valid startpoint for `set_multicycle_path` constraint. However budgeting generates `set_multicycle_path -from IN1` for partition which is valid when the constraints are sourced for the partition because `IN1` is a top-level port for partition and a valid start point.
- `set_case_analysis`
- `set_max_delay`
- `set_min_delay`
- `set_logic_zero`
- `set_logic_one`
  Partition ports could be left unconstrained, which means that there are some ports missing `set_input_delay` or `set_output_delay` constraints in the constraint file. Several factors can cause a partition I/O being unconstrained. For instance, `set_false_path`, `set_case_analysis`, `set_disable_timing` in an input constraint file can effectively cut paths through a port. The `set_input_delay`constraint at the top-level, without a reference clock is another possibility which can cause some partition ports being unconstrained. Missing timing arcs in cell timing model can also cut timing paths. If a constant signal (1`'b0, 1'b1`) is assigned to a net leading to a partition port in Verilog®, the constant signal can also cause that port to be left unconstrained.
- `Min` and `-hold`
  The following commands are supported:
- `set_clock_latency -min`
- `set_clock_transition -min`

- `set_clock_uncertainty -hold`
- `set_drive -min`
- `set_driving_cell -min`
- `set_input_delay -min`
- `set_output_delay -min`
- `set_false_path -hold`
- `set_load -min`
- `set_min_delay`
- `set_multicycle_path -hold`
- `set_timing_derate`
  This constraint is pushed down to a separate file with the extension `.nonsdc.constr` in the push down directory.

Warning Report

The `saveTimingBudget -ptn` command generates a warning report (`*partition_or_instance*.warn`) for the each partition and stores these reports in the partition subdirectories.

Pin Constraint Values Greater than Available Time

The `.warn` report contains a section entitled "Pin constraint values greater than available time."

The software checks whether generated `input_delay/output_delay` budgets are less than a maximum allowed time in the clock period for the delay. The maximum allowed time is defined as a delay between active edge of the starting clock and the sampling edge of the sampling clock. This time may vary based on phase shift and multicycle path directives. If `deriveTimingBudget tsConsCheck` is specified, budget checking will use more conservative value for available time:

The tool subtracts `clk2q` delays from available time when checking `output_delay` statements and setup time when checking `input_delay` statements.

The following command reports all partition ports that have slack less than <*value*>
in *partition_dir*/*partition_name*/*partition_name*/constr.warn:

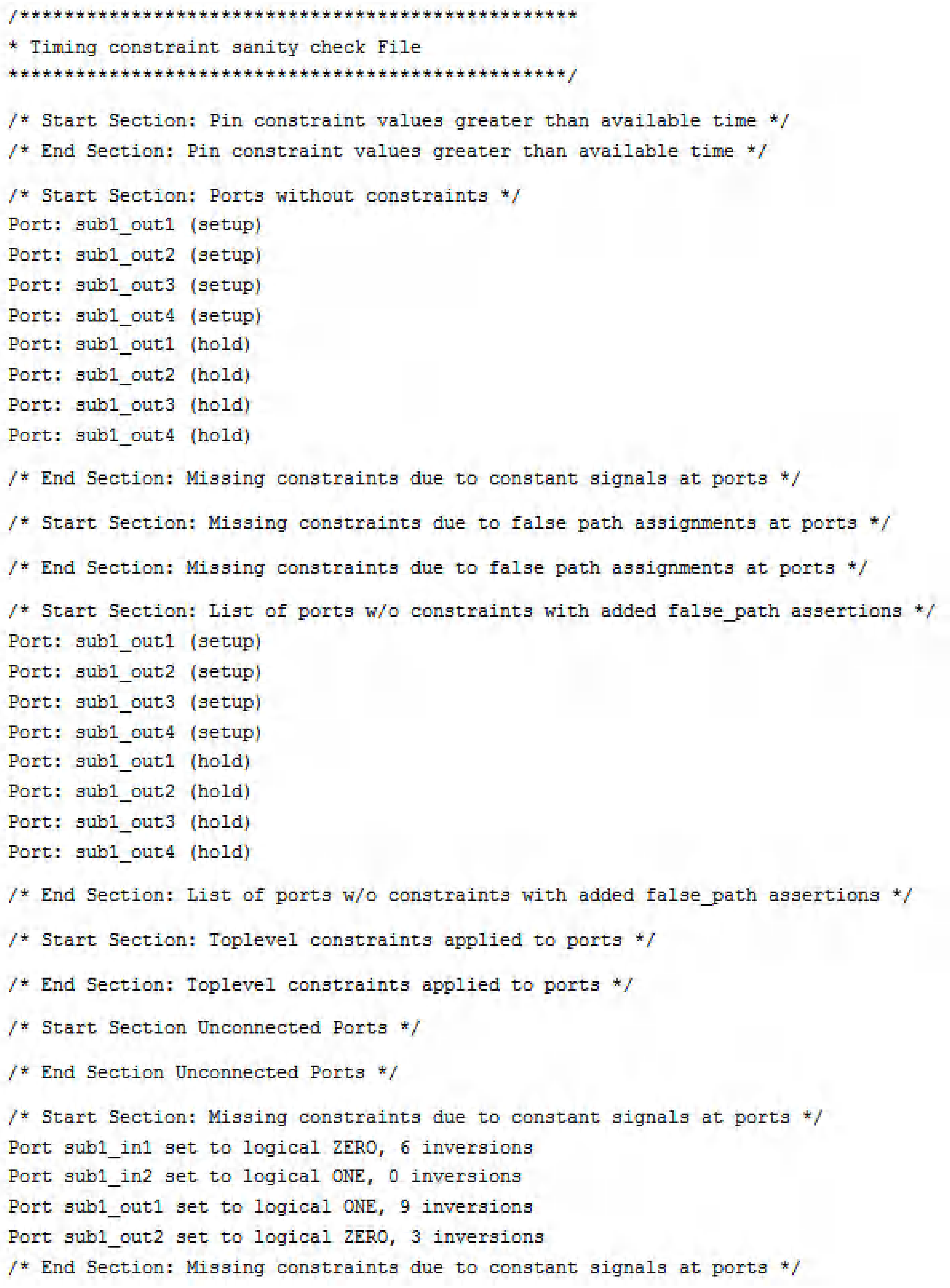
```
savePartition/saveTimingBudget -rptNegSlackOnPorts value
```

For example:

```
*.warn file:
....
/* Start Section: Instance ports with slack < 0.020 */
/* End Section: Instance ports with slack < 0.020 */
```

Warning Report Example

The warning report format is as follows:



```
/*************************************************
* Timing constraint sanity check File
*************************************************/

/* Start Section: Pin constraint values greater than available time */
/* End Section: Pin constraint values greater than available time */

/* Start Section: 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 */
```

Cycle-Based Timing Budgeting

Cycle-based timing budgeting is done by using the `-cycleRatio` parameter of the `deriveTimingBudget` command. You can set the timing budgeting constraint based on the clock cycle, which ensures that budgeting is done in minimum time and memory consumption by deriving partition block delays from the total clock cycle time. For this, you can use the `setCycleBudgetRatio` command. The `setCycleBudgetRatio` command derives the IO delays from the total cycle time of the clock, so that the budgeting can be done in least time and memory. This command supports:

- Top-level segment ratio
- Different top ratio based on critical partition to partition connection
- Different ratio between source and sink partition
- Support clock based segment control

Note: Do not use the `setFixedBudget` command during the cycle-based timing budgeting.

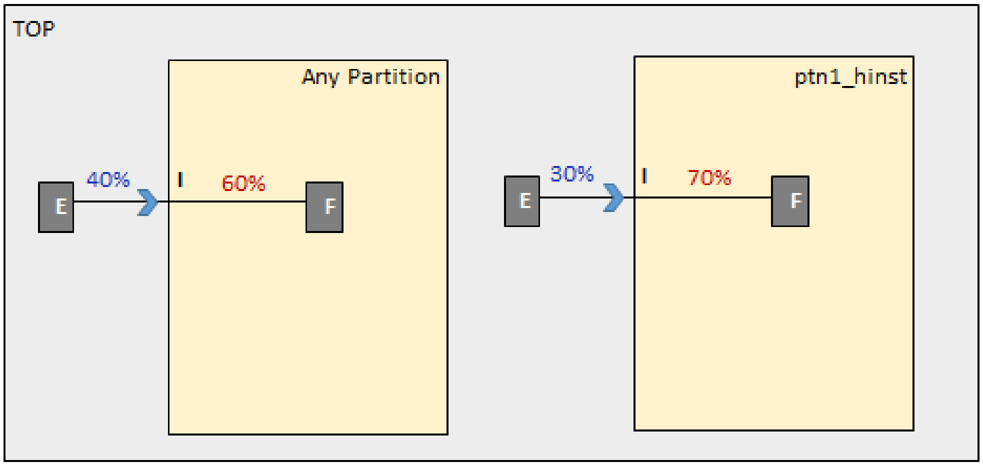
Examples

- The following commands present an example of the TOP to PTN path:

  ```
  setCycleBudgetRatio -fromTop -toPtn {* 0.6}
  ```

  ```
  setCycleBudgetRatio -fromTop -toPtn {ptn1_hinst 0.7}
  ```

  Note: * matches with any partition hinst. However, it does not matches with TOP.
  Following is the output:

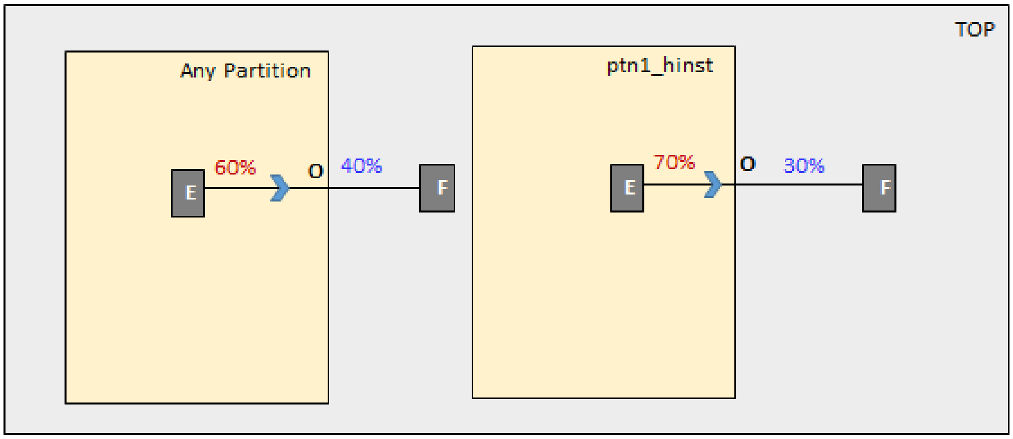


- The following commands present an example of the PTN to TOP path:

```
setCycleBudgetRatio -fromPtn {* 0.6} -toTop
setCycleBudgetRatio -fromPtn {ptn1_hinst 0.7} -toTop
```

Note: * matches with any partition hinst. However, it does not matches with TOP.

Following is the output:

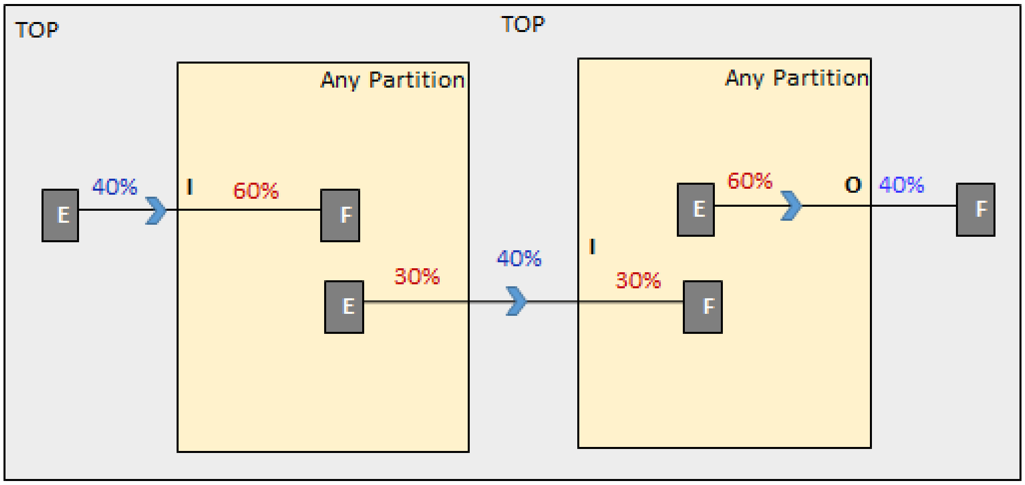


- The following commands present an example of reserving TOP percentage:

```
setCycleBudgetRatio -fromPtn {* 0.6} -toTop
setCycleBudgetRatio -fromTop -toPtn {* 0.6}
setCycleBudgetRatio -fromPtn {* 0.3} -toPtn {* 0.3}
```

Note: * matches with any partition hinst. However, it does not matches with TOP.

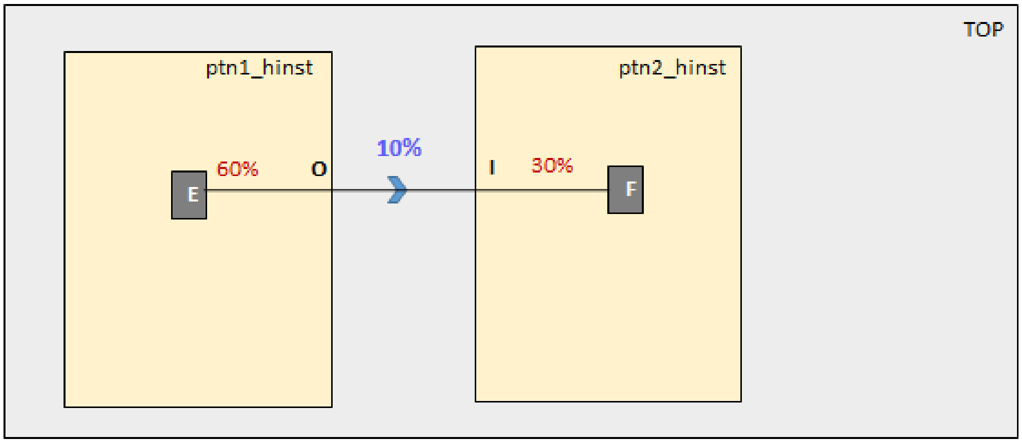
Following is the output:



- The following command presents an example of the PTN to PTN path:

```
setCycleBudgetRatio -fromPtn {ptn1_hinst 0.6} -toPtn {ptn2_hinst 0.3}
```

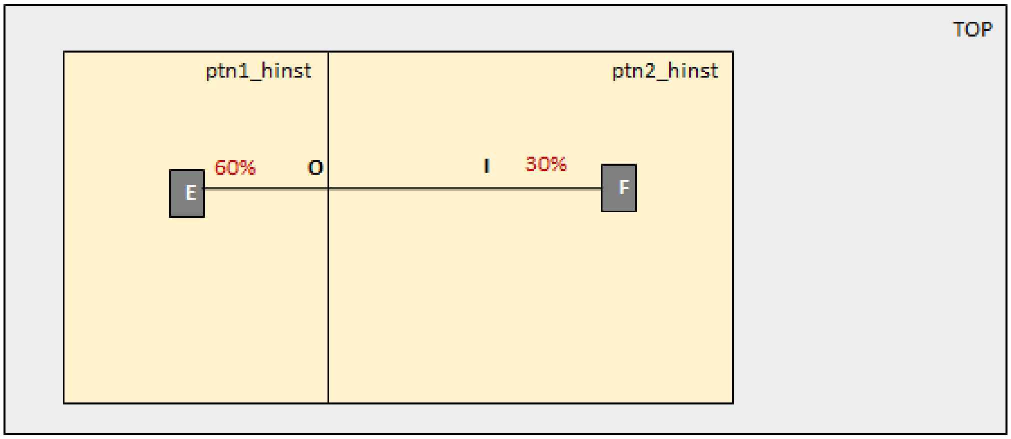
Following is the output:



- The following command presents an example of the PTN to PTN Abutted path:

  ```
  setCycleBudgetRatio -fromPtn {ptn1_hinst 0.6} -toPtn {ptn2_hinst 0.3}
  ```

  Partitions are under-constrained by 10% of clock cycle. Following is the output:

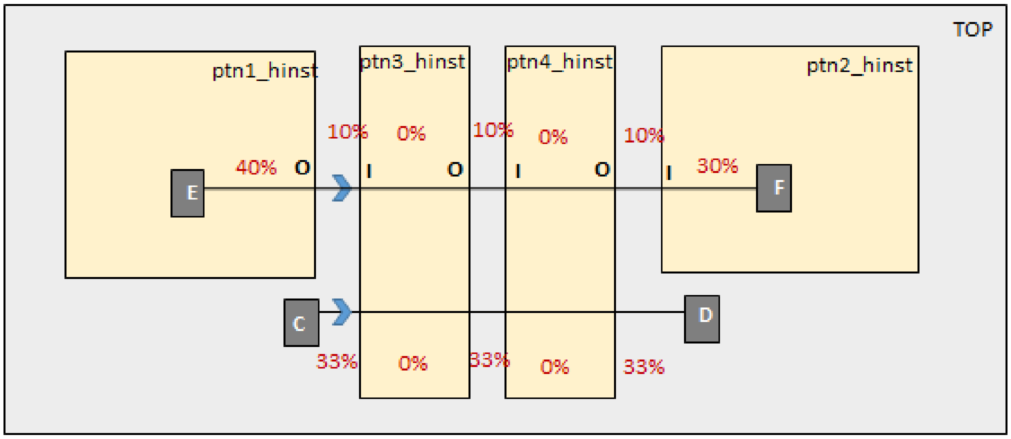


- The following command presents an example of how the top level budget is split into top and FT segments of the paths:

  ```
  setCycleBudgetRatio -fromPtn {ptn1_hinst 0.4} -toPtn {ptn2_hinst 0.3}
  ```

  Note: FT segment is over-constrained and given 0% budget.

  Following is the output:



Using setFixedBudget with setCycleBudgetRatio

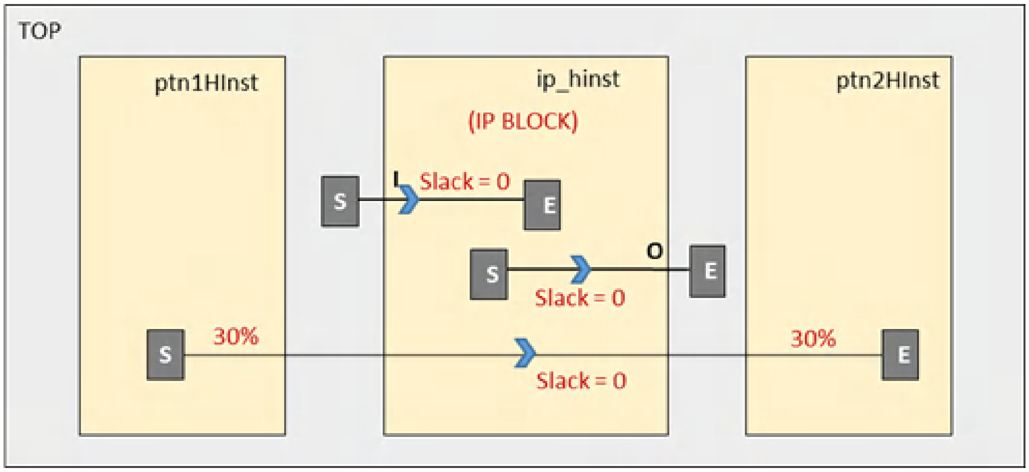
You can use the `setFixedBudget` command along with the `setCycleBudgetRatio` command to constrain paths in the cases where a block or partition budget should be fixed (the delay of block or partition should be used as its budget), and budgets should be allocated to some other partitions based on some percentage of the available time. The `setFixedBudget` command should not be used with cycle-based timing budgeting. This means that the `-cycleRatio` parameter should not be used in such a case.

Examples

- The following command presents an example of IP BLOCK:

```
setFixedBudget -through ip_hinst -slack 0

setFixedBudget -from ip_hinst/* -toFlopsOf ip_hinst -slack 0

setFixedBudget -fromFlopsOf ip_hinst -to ip_hinst/*-slack 0

setCycleBudgetRatio -fromPtn {ptn1HInst 0.3} -toPtn {ptn2HInst 0.3}

deriveTimingBudget
```

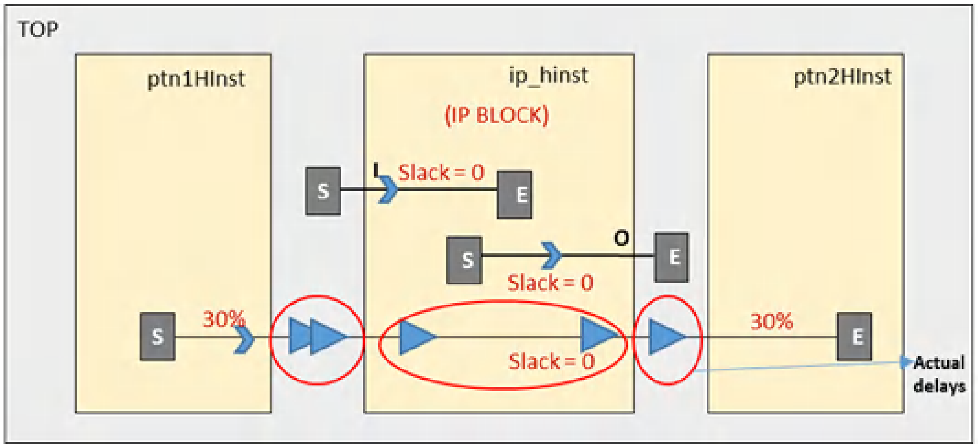
Note: Since `deriveTimingBudget` is used without `-cycleRatio`, all the percentages are with respect to the total available clock cycle time.
Following is the output:



- Following is an example of a channel based with fixed delay in channels using the `-through` parameter:

```
setFixedBudget -through ip_hinst -slack 0
setFixedBudget -through / -slack 0
setFixedBudget -from ip_hinst/* -toFlopsOf ip_hinst -slack 0
setFixedBudget -fromFlopsOf ip_hinst -to ip_hinst/*-slack 0
setCycleBudgetRatio -fromPtn {ptn1HInst 0.3} -toPtn {ptn2HInst 0.3}
deriveTimingBudget
```

Following is the output:

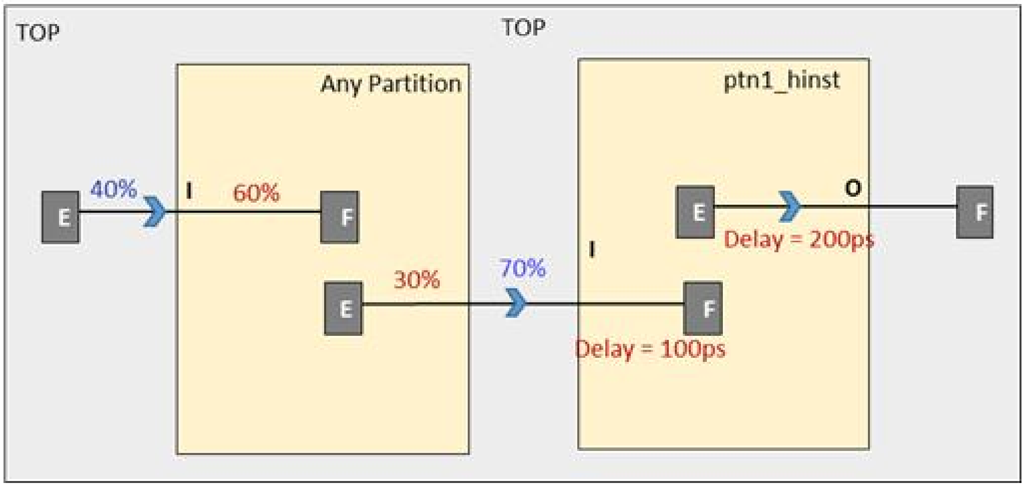


- The following command presents an example of fixed actual delay segments:

```
setCycleBudgetRatio -fromTop -toPtn {* 0.6}
setFixedBudget -from ptn1_hinst/* -toFlopsOf ptn1_hinst -delay 100ps
setFixedBudget -fromFlopsOf ptn1_hinst -to ptn1_hinst/* -delay 200ps
```

```
setCycleBudgetRatio -fromPtn {* 0.3} -toPtn {ptn1_hinst 0}
deriveTimingBudget
```

Following is the output:

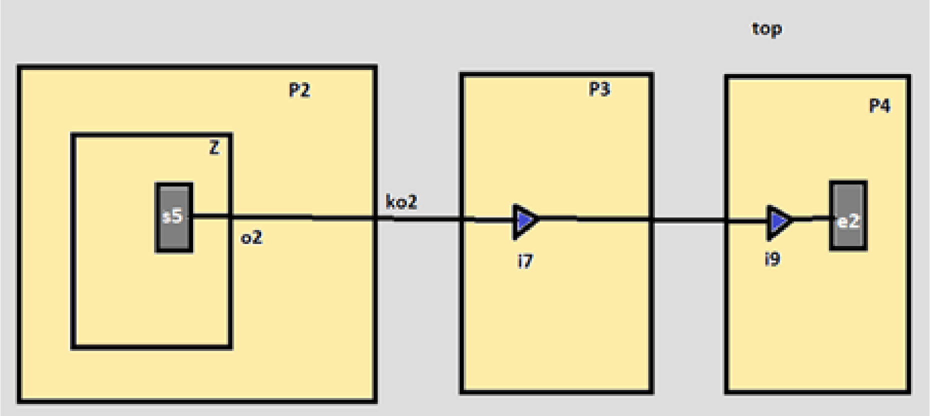


Using Cycle-Based Timing Budgeting with Nested Partitions

In addition to the individual partitions, cycle-based budgeting can also be used for nested partitions. You can use the `setCycleBudgetRatio` command to also specify the ratio of the parent and child partitions for the desired timing budgeting for parent-level segment.

Example

A design has four partitions: P2, P3, P4 and P2/Z (nested inside P2). The path used for budgeting starts from a flip-flop, s5, in the P2/Z partition to another flip-flop, e2, in the P4 partition, passing through P3, as presented in the following figure:



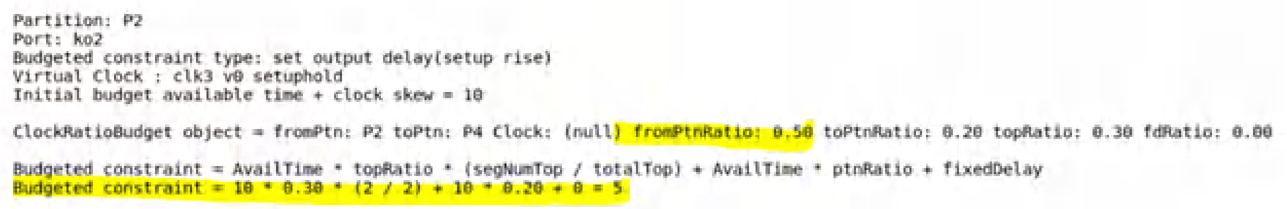
The following commands present cycle-based timing budgeting for the path presented in the above figure:

```
setCycleBudgetRatio -fromPtn {P2 0.5} -toPtn {* 0.2}
```

```
setCycleBudgetRatio -fromPtn {P2/Z 0.3} -toPtn {* 0.2}
```

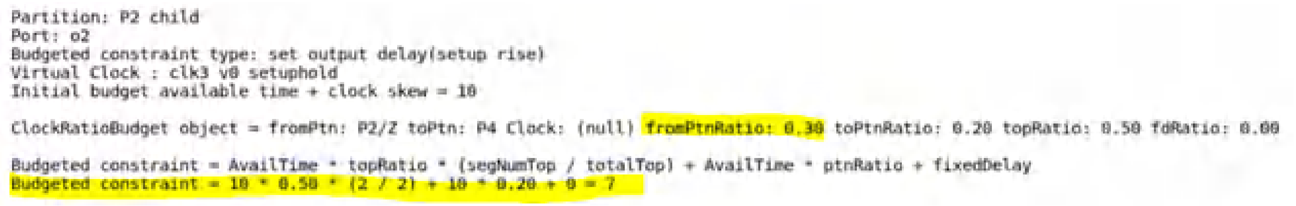
```
deriveTimingBudget -cycleRatio -justify
```

Following is the P2 justify budget report generated for the ko2 port:



```
Partition: P2
Port: ko2
Budgeted constraint type: set output delay(setup rise)
Virtual Clock : clk3 v0 setuphold
Initial budget available time + clock skew = 10

ClockRatioBudget object = fromPtn: P2 toPtn: P4 Clock: (null) fromPtnRatio: 0.50 toPtnRatio: 0.20 topRatio: 0.30 fdRatio: 0.00

Budgeted constraint = AvailTime * topRatio * (segNumTop / totalTop) + AvailTime * ptnRatio + fixedDelay
Budgeted constraint = 10 * 0.30 * (2 / 2) + 10 * 0.20 + 0 = 5
```

Following is the Z justify budget report generated for the o2 port:

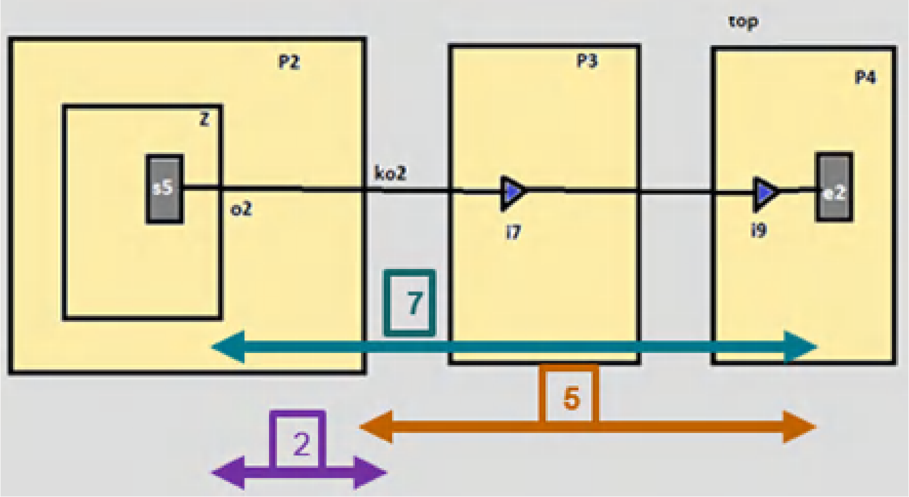


```
Partition: P2 child
Port: o2
Budgeted constraint type: set output delay(setup rise)
Virtual Clock : clk3 v0 setuphold
Initial budget available time + clock skew = 10

ClockRatioBudget object = fromPtn: P2/Z toPtn: P4 Clock: (null) fromPtnRatio: 0.30 toPtnRatio: 0.20 topRatio: 0.50 fdRatio: 0.00

Budgeted constraint = AvailTime * topRatio * (segNumTop / totalTop) + AvailTime * ptnRatio + fixedDelay
Budgeted constraint = 10 * 0.50 * (2 / 2) + 10 * 0.20 + 0 = 7
```

In the above reports:

- `set_output_delay` at Z level is 7.
- `set_output_delay` at P2 level is 5.
- So, the effective delay for partition P2 is 2.

The following figure presents allocation of budgets in this example:



⚠ Define the budgets for the parent and child partitions carefully, so that the budget allocated at the parent level is appropriate, and not negative. The software does not check for the correctness of parent and child partition ratio definition.

Stage-Based Timing Budgeting

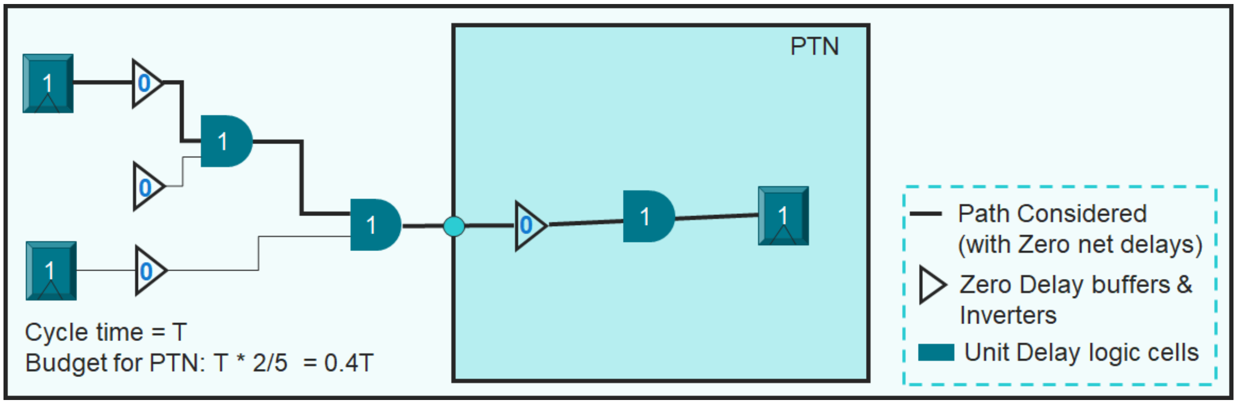
Stage-based timing budgeting is done using a timing graph, which has annotated unit delays for every logic cell or flip-flop (excluding buffers and inverters). This method of timing budgeting can be used in very early stages of the design, before applying optimization for quick budgets. It is more accurate than the cycle-based timing budgeting. In this method, timing budgeting takes the longest logic path and proportionately divides the clock cycle time as budgets to the blocks. Stage-based timing budgeting can be done using the `-stageBased` parameter of the `deriveTimingBudget` command, which is explained below.

deriveTimingBudget -stageBased

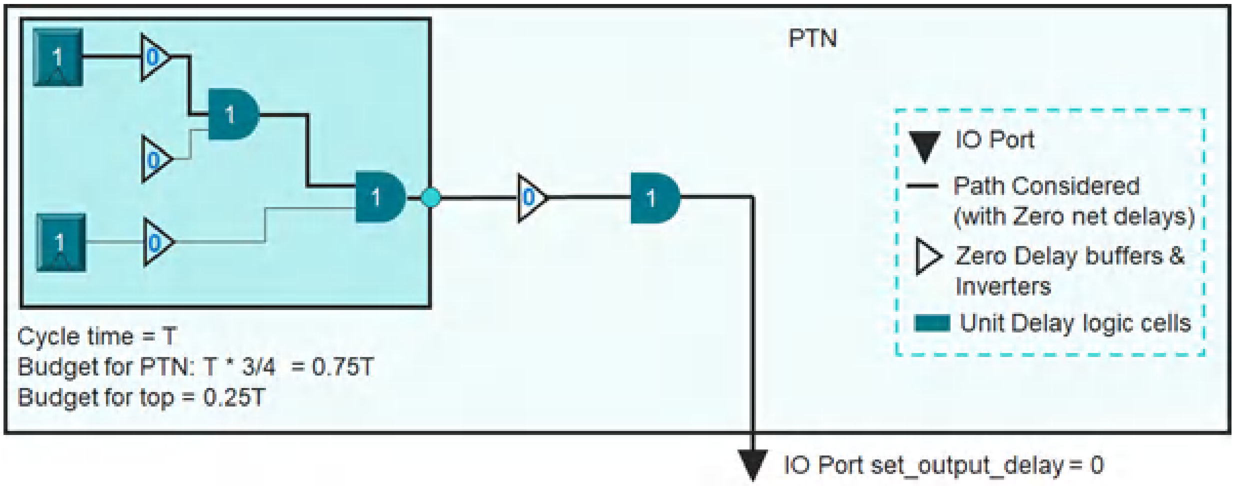
This parameter creates a quick stage-based timing graph to be used for timing budgeting. Timing budgeting with the stage-based timing graphs is used in very early stages of the design, before applying optimization for quick budgets. This method is more accurate than the cycle-based timing budgeting.

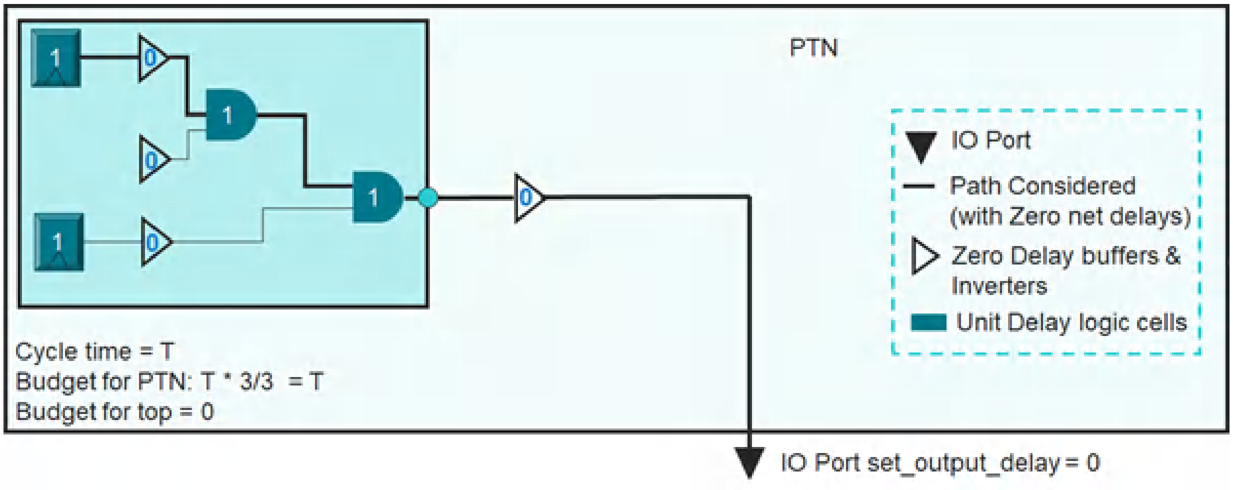
When this parameter is used, any existing timing graph in the design is overwritten by the generated special quick stage-based timing graph. The following examples present the scenarios in which this parameter can be used:

reg2reg path

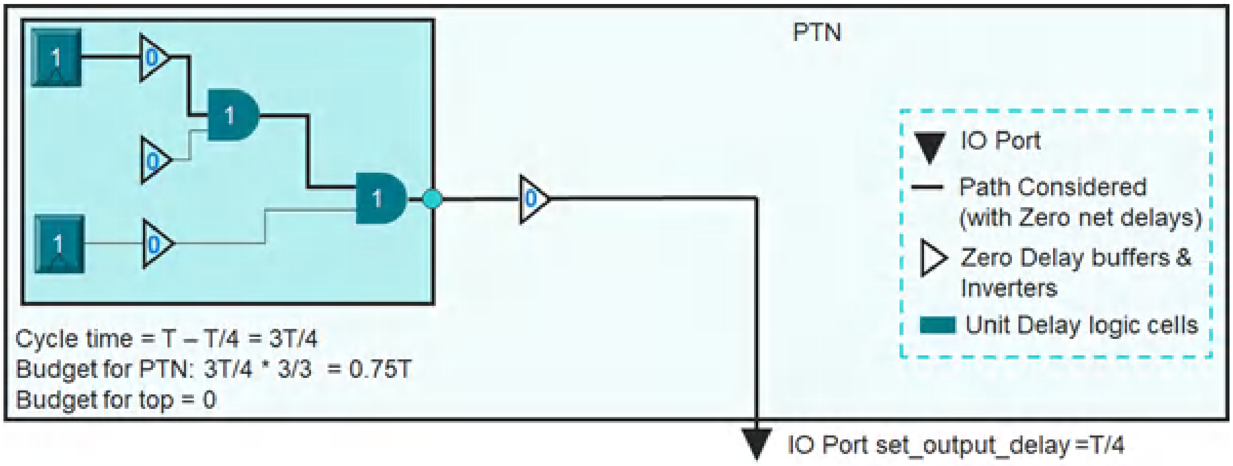


reg2out path with IO Port set_output_delay = 0



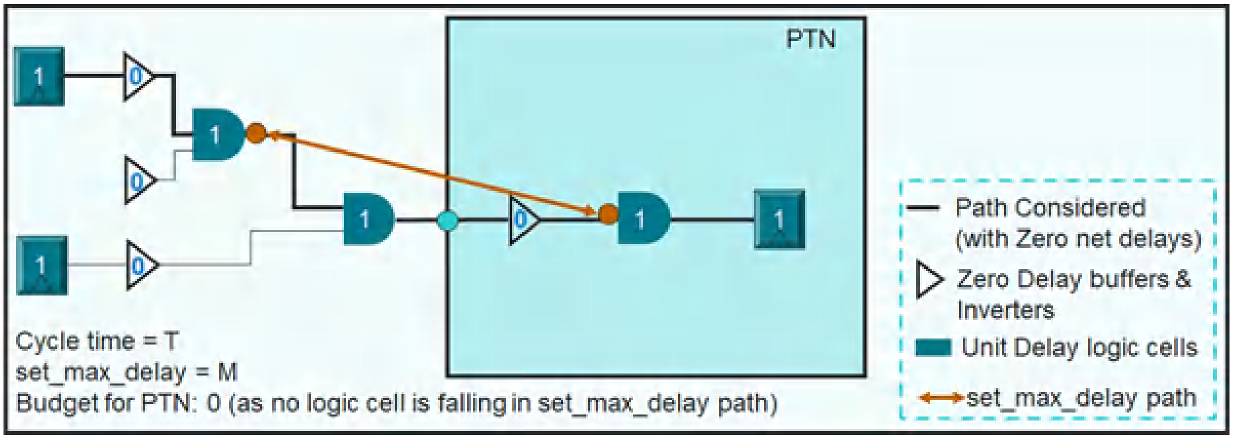


reg2out path with IO Port set_output_delay = T/4


PTN
1
0
1
0
1
1
0
0
IO Port
Path Considered (with Zero net delays)
Zero Delay buffers & Inverters
Unit Delay logic cells
IO Port set_output_delay =T/4

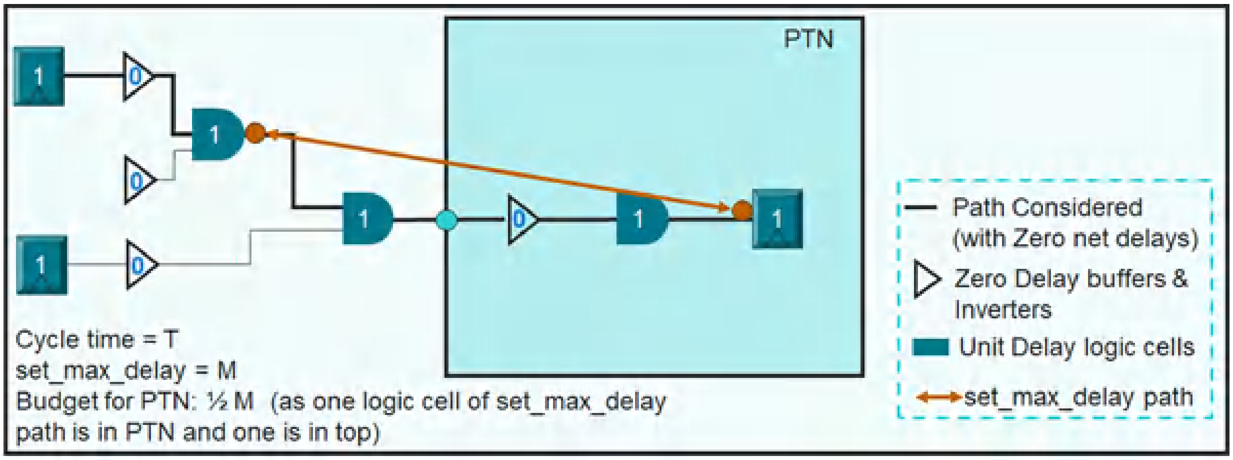
Cycle time = T – T/4 = 3T/4
Budget for PTN: 3T/4 * 3/3 = 0.75T
Budget for top = 0

reg2reg path with set_max_delay - Case 1


PTN
1
0
1
0
1
0
1
0
1
1
Path Considered (with Zero net delays)
Zero Delay buffers & Inverters
Unit Delay logic cells
set_max_delay path

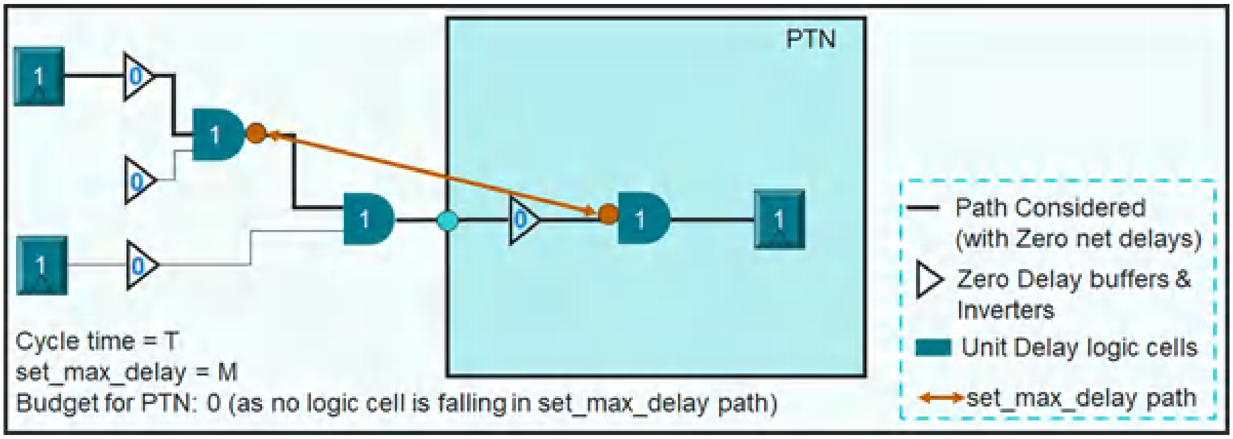
Cycle time = T
set_max_delay = M
Budget for PTN: 0 (as no logic cell is falling in set_max_delay path)

reg2reg path with set_max_delay - Case 2

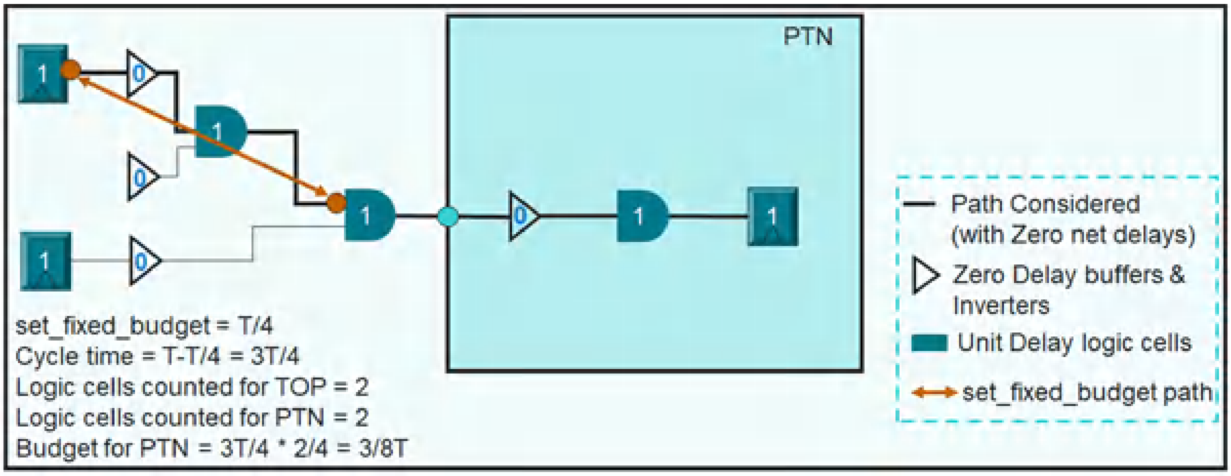

PTN
1
0
1
0
1
0
1
0
1
1
Path Considered (with Zero net delays)
Zero Delay buffers & Inverters
Unit Delay logic cells
set_max_delay path

Cycle time = T
set_max_delay = M
Budget for PTN: ½ M (as one logic cell of set_max_delay path is in PTN and one is in top)

reg2reg path with set_fixed_budget

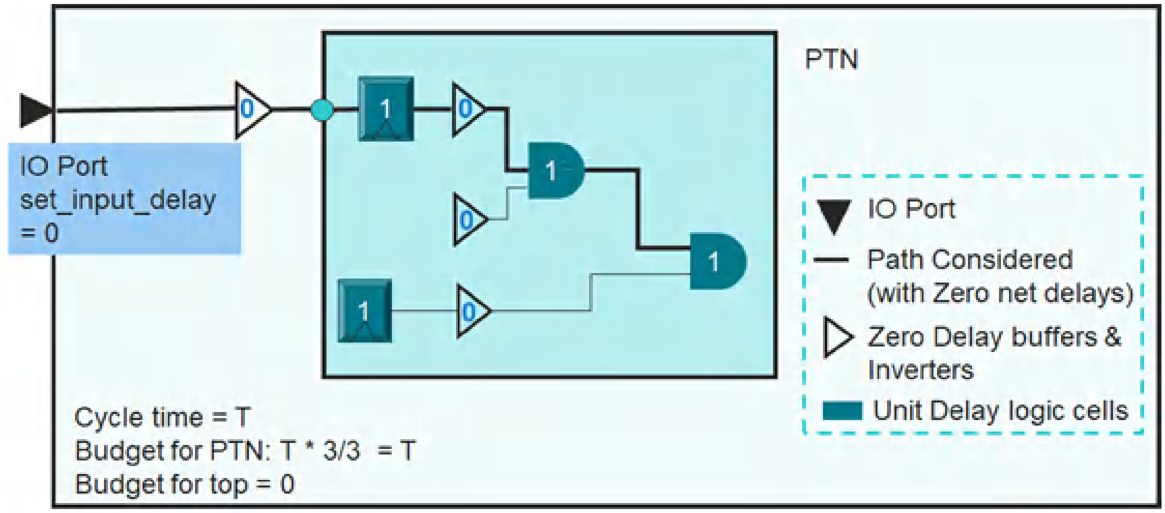

PTN
1
0
1
0
1
1
0
0
1
1
1
Path Considered (with Zero net delays)
Zero Delay buffers & Inverters
Unit Delay logic cells
set_max_delay path

Cycle time = T
set_max_delay = M
Budget for PTN: 0 (as no logic cell is falling in set_max_delay path)

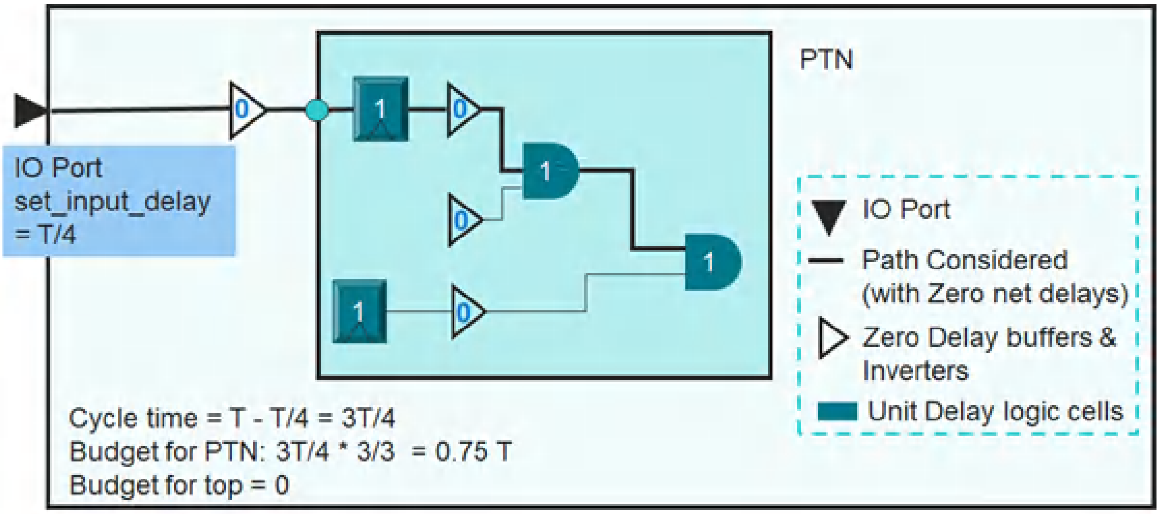

PTN
1
0
1
0
1
0
1
0
1
1
Path Considered (with Zero net delays)
Zero Delay buffers & Inverters
Unit Delay logic cells
set_fixed_budget path

set_fixed_budget = T/4
Cycle time = T-T/4 = 3T/4
Logic cells counted for TOP = 2
Logic cells counted for PTN = 2
Budget for PTN = 3T/4 * 2/4 = 3/8T

in2reg path with IO Port set_input_delay = 0


PTN
0
1
0
1
0
1
1
0
IO Port
set_input_delay
= 0
IO Port
Path Considered (with Zero net delays)
Zero Delay buffers & Inverters
Unit Delay logic cells

Cycle time = T
Budget for PTN: T * 3/3 = T
Budget for top = 0



Use the `setBudgetingMode` parameters explained in the sub-sections below for learning about more features of stage-based timing budgeting.

setBudgetingMode -stageBasedWeight

Use this attribute to specify default weights of the sequential, combinational, and buffer/inverter cells for the stage-based timing budgeting. Following are the default weights of these cells:

- Sequential: 1
- Combinational: 1
- Buffer/inverter: 1

setBudgetingMode -stageBasedPartitionMultiplier

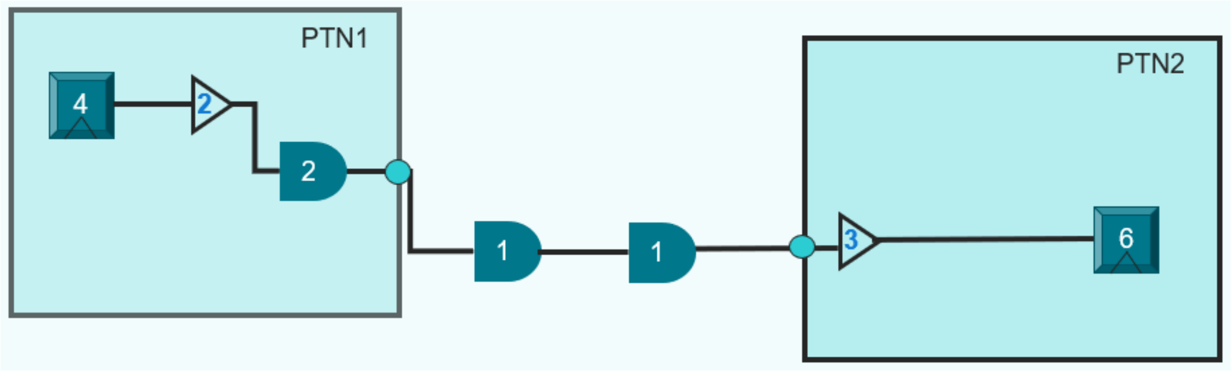
Use this parameter for specifying the weight multiplication factor for the partition hinsts. Those hInsts, which are not explicitly mentioned, will have the default weights. If the weights of hinsts have been specified using the `setBudgetingMode -stageBasedWeight` parameter, those are used with their default weights. Else, the default weights (1 for sequential cell, 1 for combinational cell, and 0 for buffer/inverter) are used. In case of nested partitions, if the child partition multiplier is not specified explicitly, then it will have its parent's multiplier. If parent and child partitions have separate multipliers, then they will be honored.

See the examples below for understanding the usage of this parameter:

Example 1:

```
setBudgetingMode -stageBasedWeight {2 1 1}
setBudgetingMode -stageBasedPartitionMultiplier {{PTN1 2} {PTN2 3}}
```

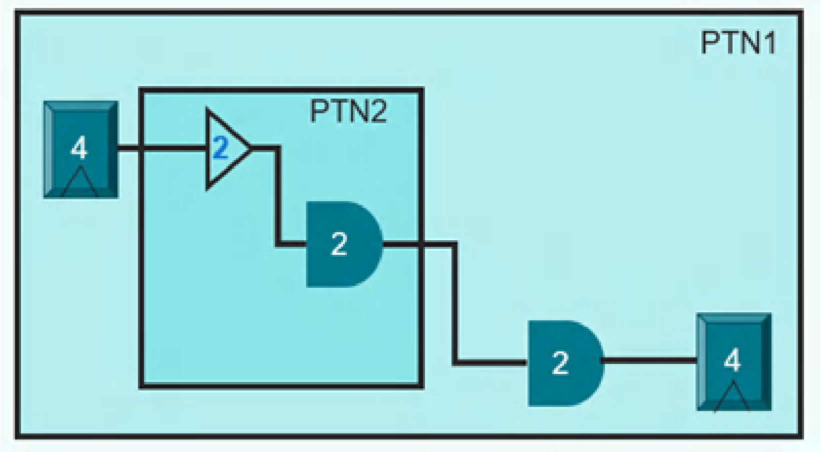
```
Cycle time = T
Budget for PTN1= 8/19 T
Budget for PTN2= 9/19 T
Budget for top = 2/19 T
```



Example 2:

This example presents usage of the `-stageBasedPartitionMultiplier` parameter in the nested design where the weight multiplier for both PTN1 and PTN2 is 2:

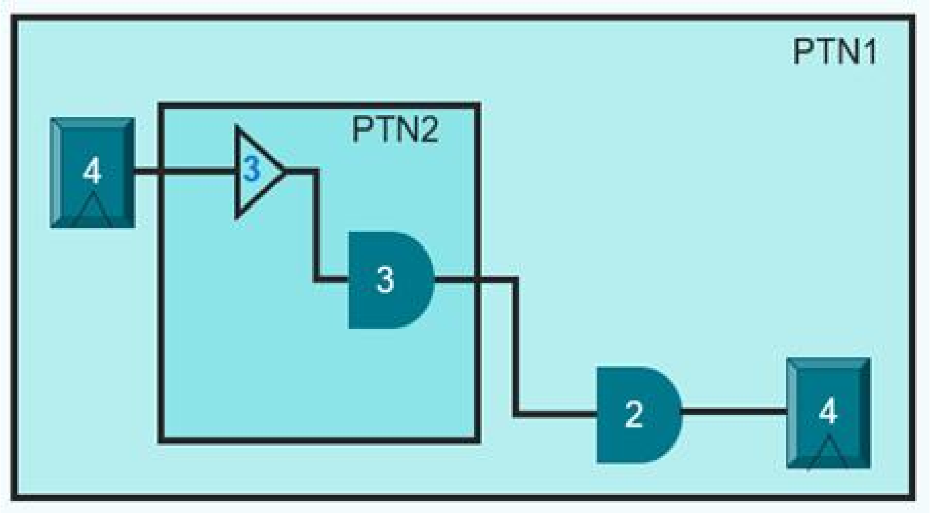
```
setBudgetingMode -stageBasedWeight {2 1 1}
setBudgetingMode -stageBasedPartitionMultiplier {{PTN1 2}}
```



Example 3:

This example presents usage of the `-stageBasedPartitionMultiplier` parameter in the nested design where the weight multiplier for PTN1 is 2 and PTN2 is 3:

```
setBudgetingMode -stageBasedWeight {2 1 1}
setBudgetingMode -stageBasedPartitionMultiplier {{PTN1 2} {PTN1/PTN2 3}}
```



setBudgetingMode -stageBasedFanoutDrivingFactor

Use this parameter for specifying the fanout count of the pin, according to which the buffer stages are added in the timing graph. This parameter provides the basis for the software to calculate the number of fanouts and buffer stages to be virtually added for each stage according to the following table:

Fanout Range	Number of Buffer Stage Added (n)
1 to d	0
(d+1) to d2	1
(d2+1) to d3	2
(d3+1) to d4	3
..	..

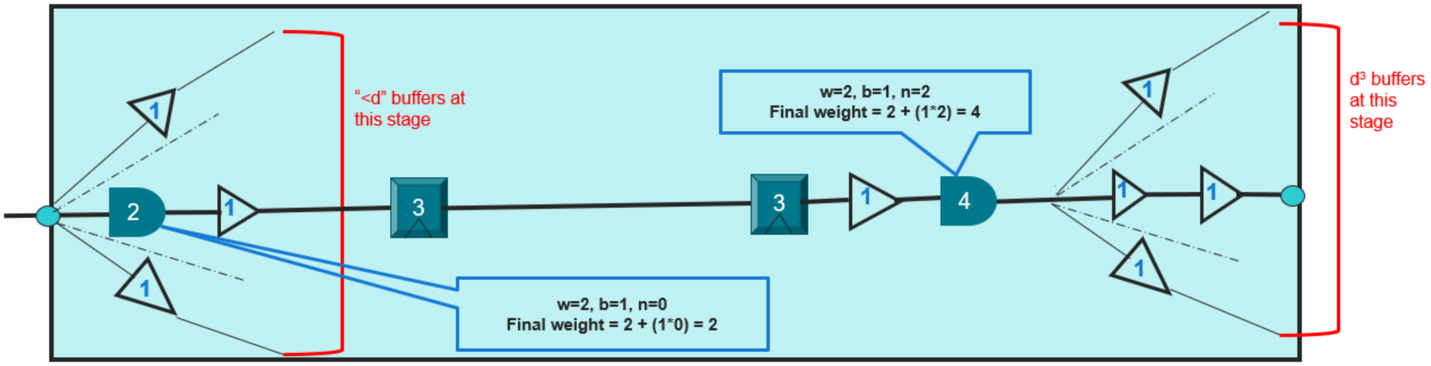
If the weight of the original cell is w, weight of the buffer is b, and the number of buffer stages is n, then:

Final weight of cell = w + (b*n)

See the examples below for understanding the functioning of the `-stageBasedFanoutDrivingFactor` parameter.

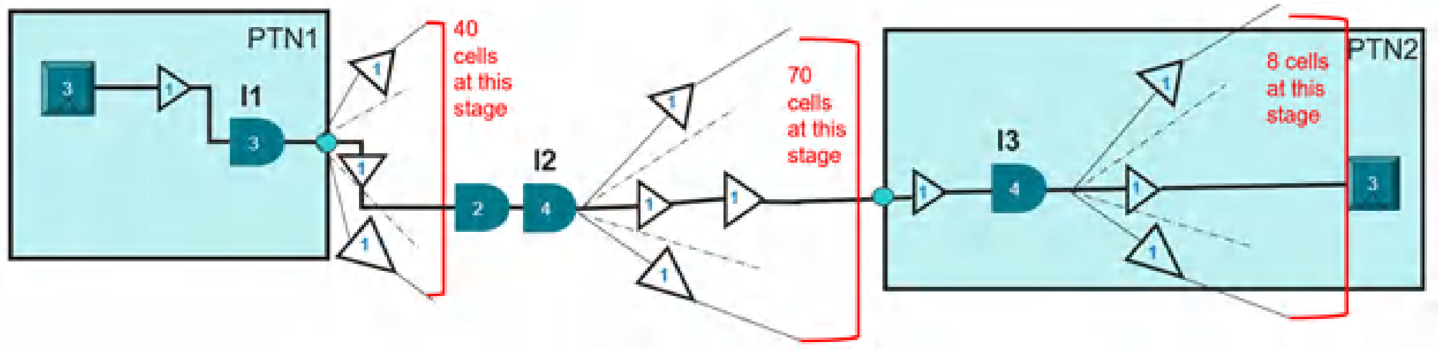
Example 1:

```
setBudgetingMode –stageBasedWeight {3 2 1}
```


"<d" buffers at this stage
w=2, b=1, n=2
Final weight = 2 + (1*2) = 4
d^3 buffers at this stage
w=2, b=1, n=0
Final weight = 2 + (1*0) = 2
2
1
3
3
1
4
1
1
1
1
1

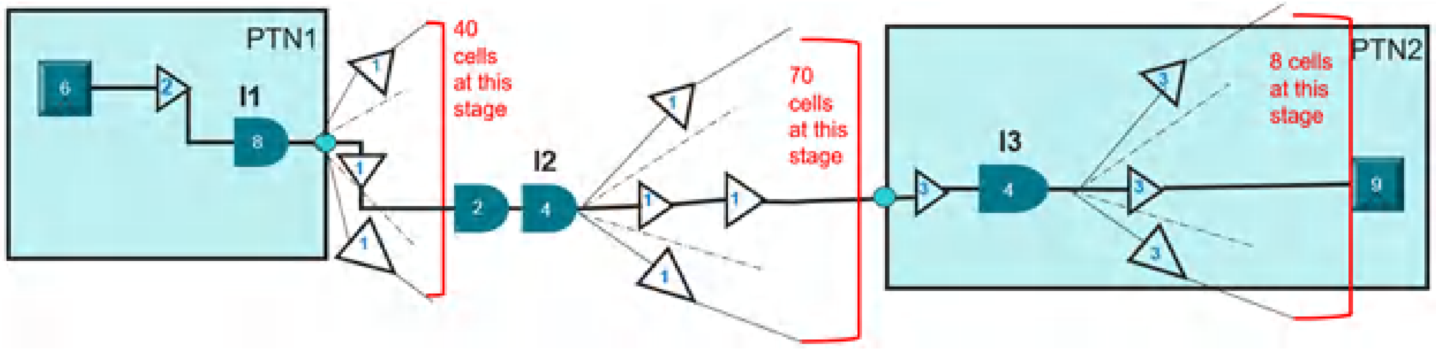
Example 2:

```
setBudgetingMode -stageBasedWeight {3 2 1}
setBudgetingMode -stageBasedFanoutDrivingFactor 8
## Default weight for combinational cell =1
## I1 has fanout of 40, so its weight = 2+ (1*1) = 3
## I2 has fanout of 70, so its weight = 2+ (1*2) = 4
## I3 has fanout of 8, so its weight = 2+ (1*0) = 2
```



Example 3:

```
setBudgetingMode -stageBasedWeight {3 2 1}  ; ## Default weight for sequential cell =3
, combinational cell =2, buffer/inverter=1 , {S C B} = {3 2 1}
setBudgetingMode -stageBasedPartitionMultiplier {{PTN1 2} {PTN2 3}}  ## PTN1 {S C B} =
{6 4 2},  PTN2 {S C B} = {9 6 3}
setBudgetingMode -stageBasedFanoutDrivingFactor 8
## I1 has fanout of 40, so its weight = 4 + (2*2) = 8 ; ## I2 has fanout of 70, so its
weight = 2+(1*2) = 4 ;
## I3 has fanout of 8, so its weight = 6 + (3*0) = 6
Cycle time = T
Budget for PTN1= 16/44 T
Budget for PTN2= 19/44 T
Budget for top = 9/44 T
```



Validating Budgets

Overview

When you generate the timing budgets for partitions using the `deriveTimingBudget` command, it is good to validate the results using the `checkPartitionSdc` command. The validation ensures that the timing data of the partitioned block is same as that of the chip. This saves the ECO iteration for the chips and partitioned blocks, and thereby reduces the overall design cycle time.

Note: This validation is carried out independently by comparing the constraints data and timing data.

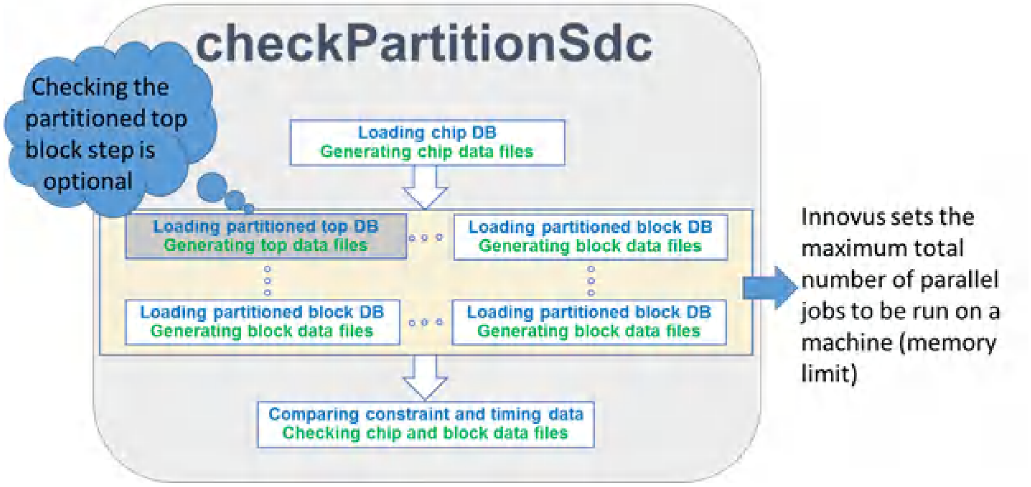
The constraints data can be generated from the third-party tools also. To run the `checkPartitionSdc` command, the constraints data and timing data for the chips and blocks must be available in the Innovus format.

On running this command, some simple constraints are directly validated though the SDC comparison. For some other checks related to pins, the chip and block values are compared. However, for some complex exception constraints, an independent validation is done by checking and comparing the phases reaching at the boundary register's pin of the chips and the blocks.

The comparison checks the files of the missing, extra, and mismatching constraints in the blocks and the chips. These can help in fixing the partitioned block constraints to get accurate budgets.

Flow

The flow of the budgets validation procedure is presented below:


checkPartitionSdc
Checking the partitioned top block step is optional
Loading chip DB
Generating chip data files
Loading partitioned top DB
Generating top data files
Loading partitioned block DB
Generating block data files
Loading partitioned block DB
Generating block data files
Loading partitioned block DB
Generating block data files
Comparing constraint and timing data
Checking chip and block data files
Innovus sets the maximum total number of parallel jobs to be run on a machine (memory limit)

When you run the command:

1. The chip database is loaded to generate the chip data files.
2. The following tasks are done in parallel :
    a. Load the partitioned top database to generate the top data files. This step is optional.
    b. Load the partitioned block database to generate the block data files. This step can be run multiple times.
3. Compare the constraints and timing data to check the chip and block data files.

Collecting Verification Data

While verifying timing budgeting, collect the following data from the chips and blocks:

- Clock Definitions
- Clock Boundary Pins
- Clock Uncertainties
- `set_case_analysis` on Boundary Pins
- `set_case_analysis` on Internal Pins
- Disable Timing on Internal Pins
- `set_dont_touch` on Instances
- Phase on End Points Connected to the Block Input Ports

Use Model

Perform the following steps:

1. Prepare a TCL file, `chip.tcl`, to load the full chip design. The file name needs to have at least the `source <design>.enc` statement.
2. Launch Innvous and run the `checkPartitionSdc` command.
3. Analyze the following results in `cps/*`:
   1. The chip data files are saved by the name ‘`chip.*`’.
   2. The block/top data files are saved by the name ‘`block.<block_name>.*`’.
   3. The compared results of the chip and block data are saved in ‘`compare_chip_vs_<block>`*. For example, the phase comparison report for block `S_T` is `TBV_REPORT/compare.chip_vs_S_T.endpt.phase.rpt`.

Following is the syntax of the `checkPartitionSdc` command:

```
checkPartitionSdc [-help]
{[-chip string -partition_dir string] | [-chip string] [-accumulated] [-
overwrite_tag] | [-compare_only]}
[-cells string]
[-master_only]
[-no_top]
[-out_dir string]
```

Note: For the Integrated Hierarchical Database (iHDB) flow, check partitions with a specified module tag (`checkPartitionSdc -module_model_tag`).

The following table presents the description of the command parameters:

`-help`	Outputs a brief description that includes the type and default information for each `checkPartitionSdc` parameter. For a detailed description of the command and all its parameters, use the man command: `man checkPartitionSdc`
`-accumulated`	Runs comparison for the accumulated budgeting mode.
`-cells` *`string`*	Specifies the list of cell names for which the SDC comparison is to be done. *Default:* `all cells`

`-chip` *`string`*	Specifies the TCL file, which loads the chip database.
`-compare_only`	Compares the dumped data with the previous timing budgeting validation run.
`-master_only`	Dumps and compares constraints for checking only the master partitions for the master-clone designs. In case this parameter is not used, the tool dumps and compares constraints taking clones also in consideration.
`-module_model_tag` *`string`*	Specifies the module model tag for the cell on which the SDC comparison is to be done. **Note:** Use this parameter for the Integrated Hierarchical Database (iHDB) flow.
`-no_top`	Disables checking of the TOP cell.
`-overwrite_tag`	Regenerates design for the accumulated Budgeting mode.
`-out_dir` *`string`*	Specifies the directory, which contains the output in the form of validation comparison reports and log files. *Default*: `./cps`
`-partition_dir` *`string`*	Specifies the directory, which contains the partition database.

Examples

- The following command runs the validation using the chip database chip.tcl and the partition database in ./PTNS:

```
checkPartitionSdc -chip ./chip.tcl -partition_dir ./PTNS
```

- The following command runs the validation using the chip database chip.tcl and the partition database in ./PTNS for partitions P1 and P2:

```
checkPartitionSdc -chip ./chip.tcl -partition_dir ./PTNS -partitions P1 P2
```

- The following command runs the validation using the chip database chip.tcl, the partition database in ./PTNS, and the dumping reports and log in ./CHK:

```
checkPartitionSdc -chip ./chip.tcl -partition_dir ./PTNS -out_dir ./CHK
```

- The following command runs the validation using the chip database chip.tcl, and partition database in ./PTNS, and only checks the partitions data (no portioned top checking with chip):

```
checkPartitionSdc -chip ./chip.tcl -partition_dir ./PTNS -no_top
```

List of Errors Detected

Clock Category

CLK-001: Clocks missing in Block, exists in Chip

CLK-002: Clocks missing in Chip, exists in Block

CLK-003: Clocks missing in Block, exists in Chip (WARN - exists on feed-thru path)

CLK-004: Clocks period mismatch

CLK-005: Clocks waveform mismatch

CLK-006: Clocks propagation mismatch

CLK-007: Clocks generation mismatch (WARN - This can occur on boundary pins)

CLK-008: Clocks uncertainty pair missing in Block, exists in Chip

CLK-009: Clocks uncertainty pair missing in Chip, exists in Block

CLK-010: Clocks uncertainty pair value mismatch

CLK-011: Clocks group missing in Block, exists in Chip

CLK-012: Clocks group missing in Chip, exists in Block

CLK-013: Clocks in clock group missing in Block, exists in Chip

CLK-014: Clocks in clock group missing in Chip, exists in Block

Case Analysis Category

CASE-001: set_case_analysis missing in Block, exists in Chip on boundary pin

CASE-002: set_case_analysis missing in Chip, exists in Block on boundary pin

CASE-003: set_case_analysis value mistmatch on boundary pin

CASE-004: set_case_analysis missing in Block, exists in Chip on internal pin

CASE-005: set_case_analysis missing in Chip, exists in Block on internal pin

CASE-006: set_case_analysis value mistmatch on internal pin

Inactive Arcs Category

DISABLE-001: set_disable_timing missing in Block, exists in Chip

DISABLE-002: set_disable_timing missing in Chip, exists in Block

Don't Touch Category

DTOUCH-001: set_dont_touch missing in Block, exists in Chip

DTOUCH-002: set_dont_touch missing in Chip, exists in Block

Data Checks Category

DCHECK-001: set_data_check missing in Block, exists in Chip

DCHECK-002: set_data_check missing in Chip, exists in Block

DCHECK-003: set_data_check (across blocks) missing in Block, exists in Chip

DCHECK-004: set_data_check (across blocks) reference pin mismatch (WARN - probably translated)

DCHECK-005: set_data_check constraint value mismatch

IO Category

IO-001: Phase at endpoint missing in Block, exists in Chip

IO-002: Phase at endpoint missing in Chip, exists in Block

IO-003: Required-time mismatch at endpoint

Related Commands

`checkPartitionSdc`

Using ART in Hierarchical Designs

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

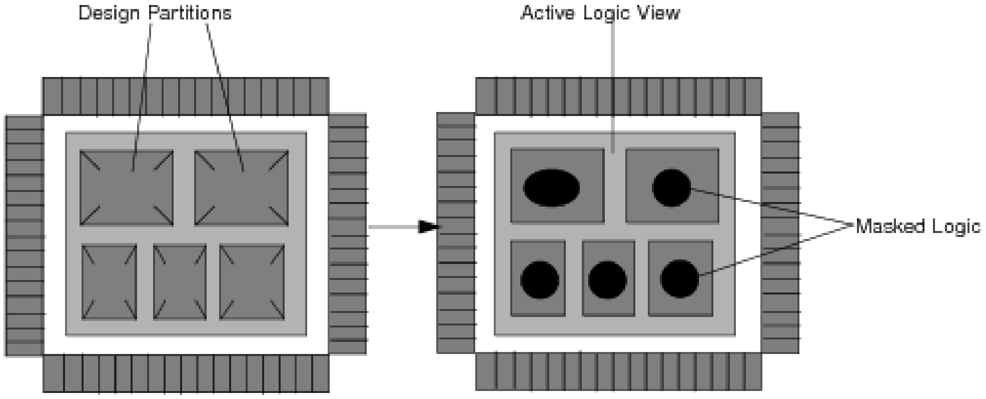
The tool creates an active logic view based on the partition boundaries, set of critical timing paths, block module boundaries, and physical area. There are two types of active logic views:

- 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.

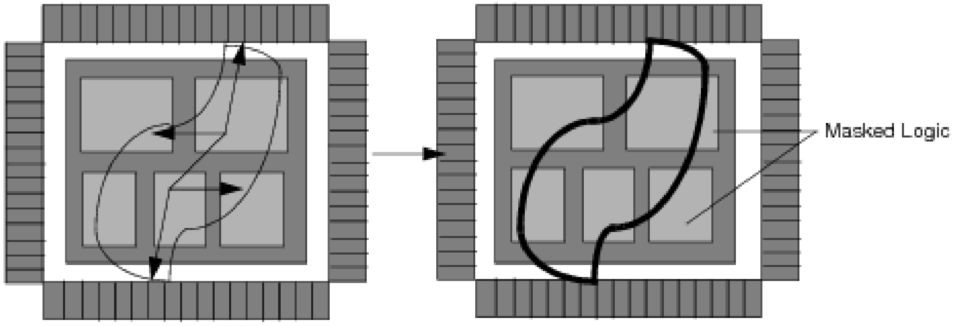
The following figure shows the flat top active logic view:



Flat Top (Partition-Based Active Logic View)

Critical

The critical active logic view activates all paths in a design that have a negative slack. All other logics in the design is masked.



Critical Path-Based Active Logic View

Creating an Active Logic View

To create an active logic view, load the entire chip as a design in the Innovus software, specify the partition, and then run the `createActiveLogicView` command with an appropriate option.

> An active logic view cannot be saved as a database or a file. Run the `createActiveLogicView` command to create an active logic view.

Note: The Innovus software considers MMMC settings while creating an active logic view.

Example of Active Logic View Creation

To create an active logic view, run the following commands:

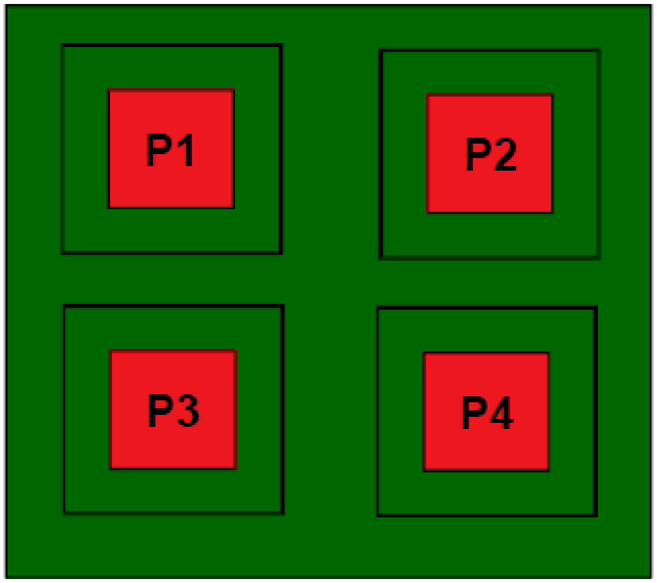
```
assembleDesign -topDir top.enc.dat -blockDir block1.enc.dat

createActiveLogicView -type flatTop
```

Note: For the Integrated Hierarchical Database (iHDB) flow, run the following commands:

```
set_module_model -default_dir ./DATA2
restore_module_model top -tag init
```

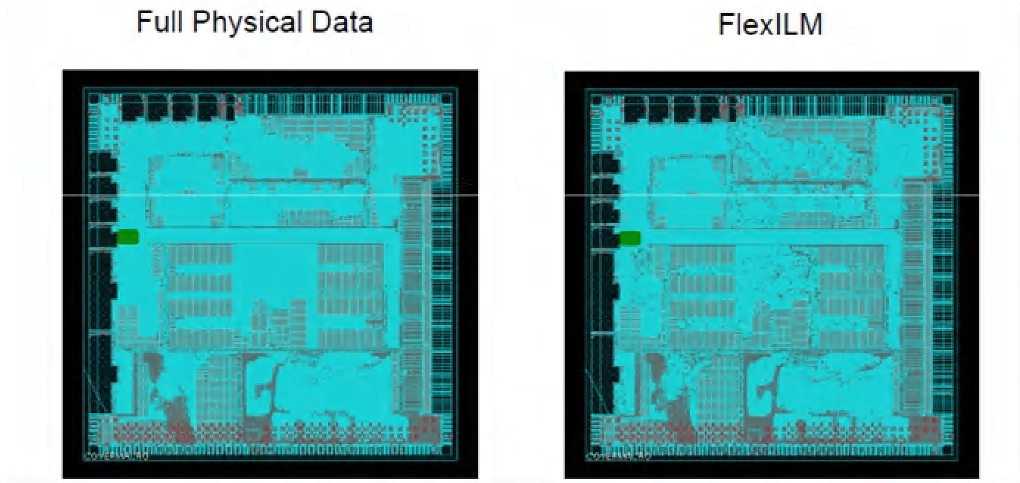
```
set_module_model -cell block1 -type pnr -tag init
commit_module_model
```

The flexILM PreCTS Closure Flow

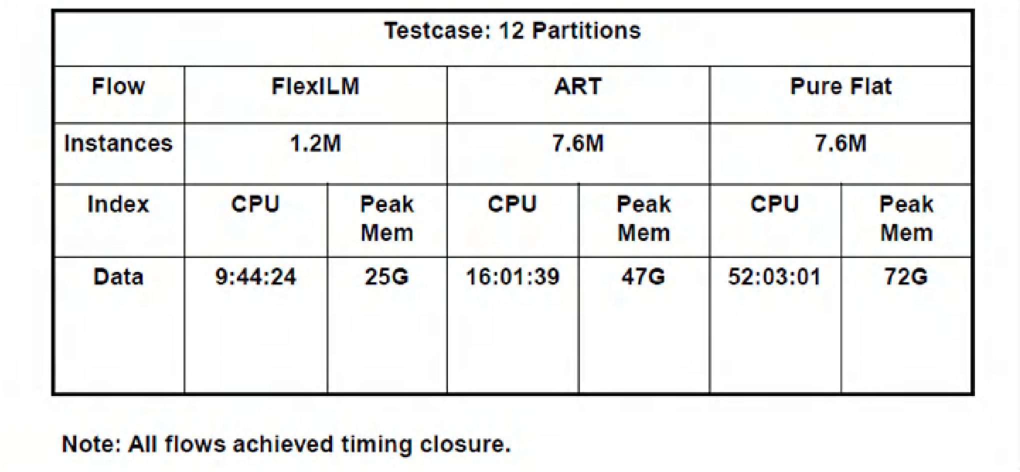


FlexILM creates reduced Netlist by trimming logic on internal path and keeping logic on interface path, and saves Netlist and DEF. In terms of Netlist and Placement ECO, FlexILM commits change on interface paths to original block data via the ECO process. It also contributes to memory reduction in timing graph and physical data.

Preliminary Results on Large Design: Floorplan



Preliminary Results on Large Design: Memory and TAT



Testcase: 12 Partitions						
Flow	FlexILM		ART		Pure Flat	
Instances	1.2M		7.6M		7.6M	
Index	CPU	Peak Mem	CPU	Peak Mem	CPU	Peak Mem
Data	9:44:24	25G	16:01:39	47G	52:03:01	72G

Note: All flows achieved timing closure.

For detail about flexILM flow, you can refer to the Top-level Timing Closure Methodologies chapter in the *Innovus User Guide*.

⚠ **Note:** For information on how to create a FlexILM model using the Integrated Hierarchical Database (iHDB) flow, see Top-level Timing Closure Methodologies for iHDB Flow.

Top-level Timing Closure Methodologies

Using Interface Logic Models (ILM)

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 nets coupling affecting the signal integrity (SI) on I/O timing paths.

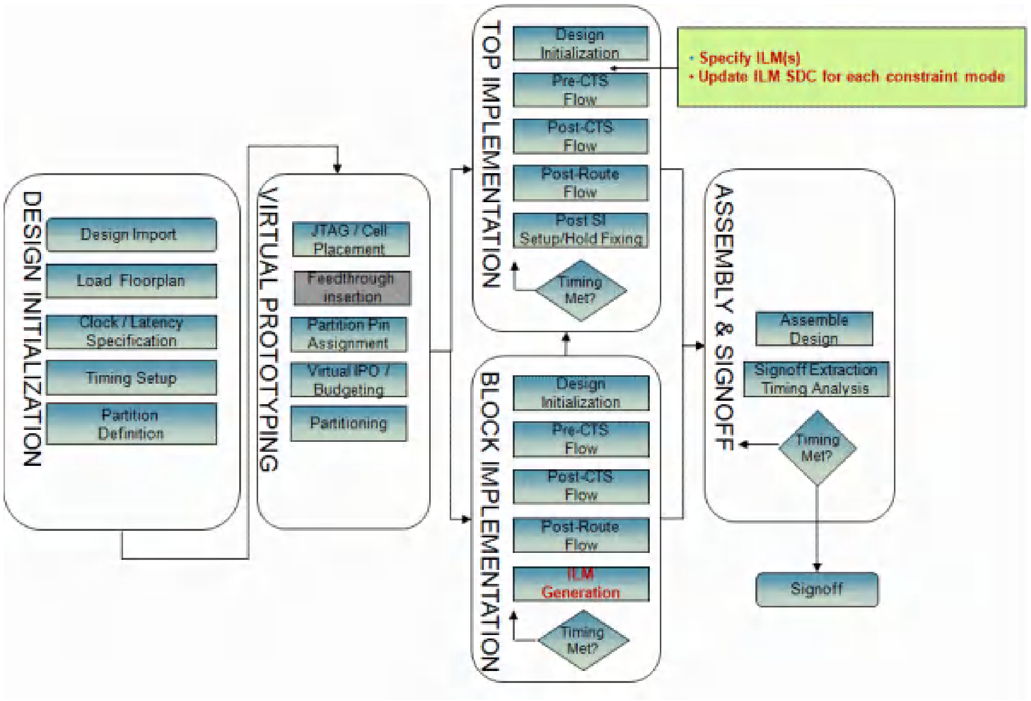
Instead of using a blackbox at the top level, you create an ILM at the block level and use it as you would use a blackbox.

The advantages of using ILMs are as follows:

- More accurate analysis than a blackbox flow
    - More SI aware than combined `.lib` or `.cdb` approach
    - Can model clock generator inside block
    - More accurate timing and SI reduces the number of design iterations to close timing and SI.
- No need to characterize blocks
    - Works on an actual design data
- Can be used in the initial prototyping stage for very big designs, when loading full design data is not feasible.
    - Allows you to modify only top-level data
    - Fully preserves implemented partitions
- Uses the original constraint file for top-level analysis
    - No abstraction for timing exceptions

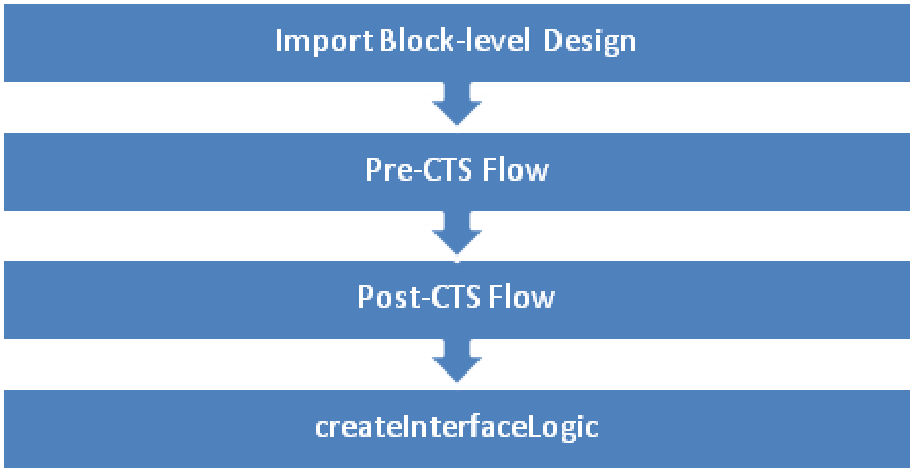
General ILM Flow

ILM is used for top-level timing closure. Models will be generated during the block implementation stage and will be used at the top implementation stage. Following is the general ILM foundation flow:



ILM Generation at Innovus Partition Block-level Design

Use the `createInterfaceLogic` command to generate ILM at the Innovus partition block-level design. Two types of models can be generated, ILM model and SI model. The ILM model will be used for timing analysis and clock tree synthesis, and the SI model can be used at postRoute stage for SI analysis. The SI model can only be generated if the block-level design is already SI-aware optimized.



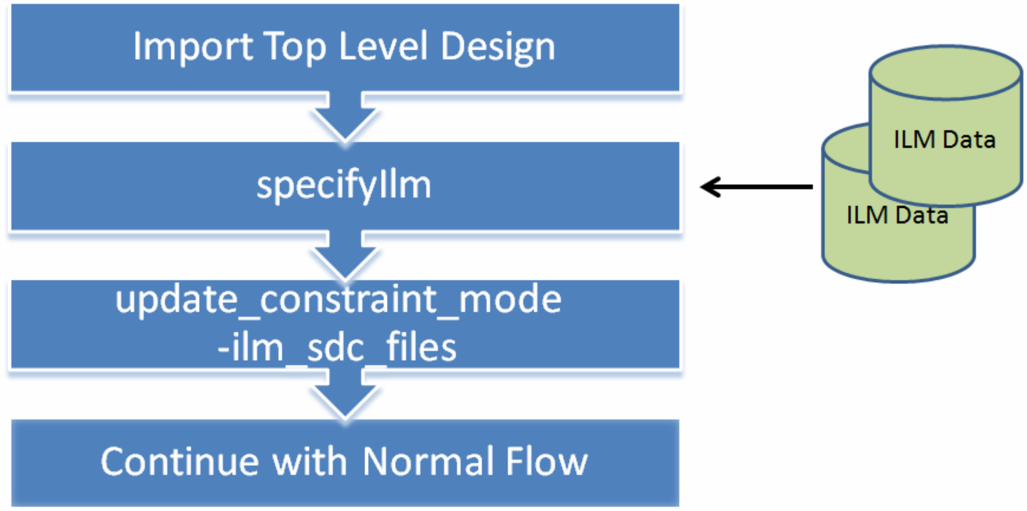
Here is an example flow script for ILM generation:

```
source DBS/init.enc

deriveTimingBudget

clearActiveLogicView

partition

savePartition -dir PTN -def

set ptnNameList {tdsp_core ram_128x16_test ram_256x16_test}

# Create partition blocks

# Cd to each partition block directory, and create ILMs

foreach ptnName $ptnNameList {

    cd  PTN/${ptnName}

    restoreDesign . ${ptnName}

    place_opt_design

    ccopt_design

    routeDesign

    saveDesign ${ptnName}_data.enc

    setAnalysisMode  -analysisType onChipVariation

    optDesign -postRoute

  createInterfaceLogic  -dir ilm

    cd ../..

}
```

Note: You can also generate an ILM model using the third-party data with the `import_ilm_data` command. However, it is recommended to use Innovus-generated models for better timing correlation.

ILM Integration at Top-level Design

Once the models have been generated, they can be specified at the top-level design for timing closure:



1. Specify a block as an ILM using the `specifyIlm` command.

   ```
   set ilmNameList {tdsp_core ram_128x16_test ram_256x16_test}

   foreach ilmName $ilmNameList {

       specifyIlm -cell ${ilmName} -dir ../../PTN/${ilmName}/ilm
   }
   ```

2. Specify the ILM SDC constraint file for each constraint mode using the full chip SDC file (not the top-level SDC file).

   ```
   update_constraint_mode -name setup_func \

       -ilm_sdc_files $dataDir/mmmc/dtmf_recvr_core_func.sdc

   update_constraint_mode -name setup_test \

       -ilm_sdc_files $dataDir/mmmc/dtmf_chip_testmode.sdc
   ```

3. Continue with the normal flow.

⚠

- ILM SDC file(s) are used in the flattened ILM view.
- Timing results are not available if the ILM SDC file is not defined.
- Top-level SDC file is used when ILMs are not specified.

Here is an example flow script for top-level timing closure using ILM:

```
restoreDesign . dtmf_recvr_core
```

```
# Specify ILMs at top-level design

set ilmNameList {tdsp_core ram_128x16_test ram_256x16_test}

foreach ilmName $ilmNameList {

    specifyIlm -cell  ${ilmName} -dir ../../PTN/${ilmName}/ilm

}

# NEED to specify SDC constraints for ILMs using FULL-chip SDC file

update_constraint_mode -name setup_func \

    -ilm_sdc_files $dataDir/mmmc/dtmf_recvr_core_func.sdc

update_constraint_mode -name setup_test \

    -ilm_sdc_files $dataDir/mmmc/dtmf_chip_testmode.sdc

flattenIlm

place_opt_design

checkPlace

earlyGlobalRoute

timeDesign -preCTS

saveDesign DBS/place_route.enc

optDesign -preCTS

timeDesign -preCTS

saveDesign DBS/optDesign.enc
```

Creating ILMs

In the hierarchical design flow, you create a detailed block-level implementation of a block, then specify the `createInterfaceLogic` command to create an ILM for the block. This command creates the specified directory containing ILM files.

You can also create ILMs for blocks that are in an intermediate stage of design, then use the data at the top level of the design for preliminary timing optimization.

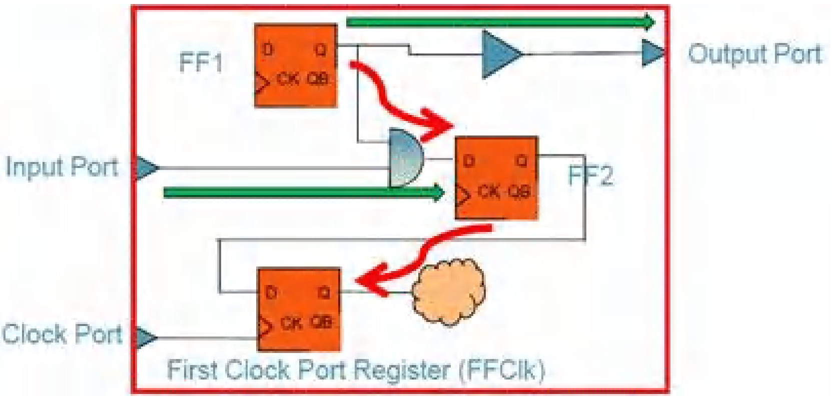
> ⓘ An ILM created for an incomplete block is not as accurate as an ILM created for a complete block. Always use ILMs for complete blocks to complete the top-level design.

The software generates ILM data for CTS, signal integrity, and other design stages (preCTS, postCTS, postRoute)

- ILM data for preCTS, CTS, postCTS, and postRoute
  The model contains the netlist of the circuitry leading from the I/O ports to interface sequential instances (that is, registers or latches), and from interface sequential instances to I/O ports. The clock tree leading to the interface registers is preserved.

  In case of CTS, the timing and CTS models have been merged to reduce the disk usage of an ILM model. The CTS data is limited to the worst clock sinks and instances/nets leading to those sinks.

  In general, internal register-to-register paths will not be kept in the ILM model. However, in special cases some internal paths will be kept as shown in the example given below. In the example, FF1 and FF2 will be kept as they are the interface flops that connect to the output port and/or input port. Also, FFClk is kept as for each clock port we need to keep at least one register for timing budget. As a result, we will have two internal register-to-register paths (shown with red arrows).



- ILM data for SI
  The model includes all the above, plus aggressor drivers or nets which affect I/O paths. It also includes the timing window files in the ILM model directory.

Note: When `createInterfaceLogic` is called, all views are generated for multi-corner, multi-mode (MMMC) analysis.

Example ILM Creation

The following method creates a model that can be used in the top-level implementation flow by both `timeDesign` and `optDesign` for setup effort, including postRoute SI optimization. This model is also used during `ccopt_design`.

```
createInterfaceLogic -dir block_A.ILM
```

Sample Summary Report

The following is a sample summary report generated at the end of the `createInterfaceLogic` command:

```
--------------------------------------------------------------
               createInterfaceLogic Summary
--------------------------------------------------------------

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 the total 7621 instances for this block have been eliminated, only 468 instances are written to the Verilog netlist, out of which 111 instances are registers.

This summary report applies to a block using MMMC. Therefore, views with worst reduction ratio are displayed for each model.

ILM Generated Data

ILM Generation provides the following output data:

File Extension	Description
.def	One def file
.v	One netlist file

.place	One Innovus place file
.sdc	One per analysis view
.spef	One per corner
.xtwf	One per analysis view. This file is only available with SI model

Preserving Selected Instances in ILMs

You can force the selected instances and nets to be included in the ILM model by using the `createInterfaceLogic -keepSelected` parameter.

1. Select instances or nets using the `selectInst` or `selectNet` commands.
2. Specify `createInterfaceLogic -keepSelected`.

Creating ILMs for Shared Modules

You can use the same sub-block module in different ILM blocks, enabling reuse of versatile modules. The `createInterfaceLogic` command considers constant propagation, so that only the enabled parts of a module are considered when creating ILMs for the reused modules. Because the Innovus database cannot handle the same module name in different circuits, the software automatically modifies the module names with the following rule:

topModuleName+timestamp+$+moduleName

As an example, one ILM block (`ModuleA`) uses an ALU module (ALU) as an unsigned ALU, and a second block (`ModuleB`) uses the ALU as a signed ALU. You can change the input signal to use the ALU differently, setting one ALU as sign enabled and the other to off. When you run the createInterfaceLogic command, the software considers only the enabled parts of the ALU when creating ILMs for `ModuleA` and `ModuleB`. The software also ensures that the name of the ALU module in `ModuleA` and the name of the ALU module in `ModuleB` are different.

Creating ILMs Without Using Innovus Database

If you do not have an Innovus database for an implemented block but have a Verilog netlist, constraints, and SPEF for that block, then use the `import_ilm_data` command to store data in the ILM format.

Following is the usage of the `import_ilm_data` command:

```
import_ilm_data -cell blk1 -dir A -model_type timing -verilog V1.v \
                -spef 1.spef -rc_corner corner1 -sdc sdcfile1 -timing_view view1

import_ilm_data -cell blk2 -dir A -model_type si -verilog V2.v \
                -spef 2.spef -rc_corner corner2 -sdc sdcfile2 -timing_view view2

import_ilm_data -cell blk1 -model_type timing -cell_view designLib \
                -verilog V1.v

import_ilm_data -cell blk1 -dir A -model_type timing -verilog V1.v \
                -spef 1.spef -rc_corner corner1

import_ilm_data -cell blk1 -dir A -model_type timing \
                -sdc sdcfile1 -timing_view view1
```

- `-cell`, `-model_type`, `{-cell_view | -dir}` are required options. When you import an ILM model from a third party, you must specify the cell that you want to transfer to Innovus, the model that will be put inside the Innovus ILM model, and the directory where the ILM data will be stored.
- `-timing_view` requires `-sdc`, and `-rc_corner` requires `-spef`. If these options are not provided together, the tool will give an error message.

Note: The `import_ilm_data` command does not merge timing and CTS data into one model. ILMs created using this command will have separate models for timing, CTS, and SI.

Specifying ILM Directories at the Top Level

Use `specifyIlm` for ILM data of a block at the top partition level rather than using the default .lib model. You can run `specifyIlm` multiple times in the same session. Each time you run this command, the software overwrites the previous setting for the same block. If master/clones exist in the design, the cell name will have the name of the master partition.

Note: You can use this command (and `unspecifyIlm`) only if the ILMs are unflattened (`unflattenIlm`). You cannot change ILM settings in flattened ILM view.

Use `unspecifyIlm` to revert to the .lib model of the block.

Example Top-level Implementation Flow with ILMs

1. Before you start the Innovus tool, prepare the top-level Verilog file, if needed.

   If you use the Innovus hierarchical flow in a previous Innovus session, then the `savePartition` command automatically creates the top-level data. Else, you need the following in the top-level directory:

   - A Verilog netlist that includes dummy modules for the blocks (ILM or Liberty) in the design.
   - A view definition file since ILMs are supported only in the MMMC mode. If you have a non-MMMC design, create or load a view definition file that contains the following:

     ```
     set_analysis_view -setup {mode1_slowCorner} -hold {mode1_fastCorner}
     ```

2. Start an Innovus session from the top-level module directory within the directory where the partitions are saved.
3. Load the design, including the top-level netlist, ILM directory name, `ilm_blocks.lib` (optional if using ILM), `stdcells.lib`, and `.lef` for the block and chip-level constraints.

   ```
    specifyIlm -cell block_A -dir ../block_A/block_A.ILM
    specifyIlm -cell block_B -dir ../block_B/block_B.ILM
   ```

4. Load the floorplan.

   ```
   loadFPlan top_floorplan
   ```

5. Flatten the ILM.

   ```
   flattenIlm
   ```

6. Place the design.

   ```
   place_
   ```

7. Build the clock tree.

   ```
   cc
   ```

8. Run postCTS timing optimization.

   ```
   optDesign -postCTS
   optDesign -postCTS -hold    ;#optional
   ```

9. Route the design.

   ```
   routeDesign
   ```

10. Run SIAware inside pre-route optimization and post-route optimization with hold.

    ```
    setDelayCalMode -SIAware true
    optDesign -postRoute
    optDesign -postRoute -hold
    ```

If you want to create an ILM of the resulting block for use in the next level up in the hierarchy, run the following steps with the above-mentioned flow:

1. Enable SI aware delay calculation.

   ```
   setDelayCalMode -SIAware true
   ```

2. Perform timing analysis.

   ```
   timeDesign -postRoute
   ```

3. Create ILM.

   ```
   createInterfaceLogic -dir block_parent
   ```

ILMs Supported in MMMC Analysis

Cadence strongly recommends that you use ILMs in the MMMC mode. If you have a non-MMMC design, create and load a view definition file that contains the following:

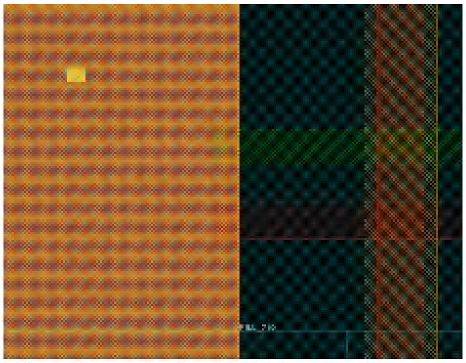
```
set_analysis_view -setup {mode1_slowCorner} -hold {mode1_fastCorner}
```

The MMMC analysis for designs including ILMs is identical to MMMC analysis for black box designs except for the following considerations:

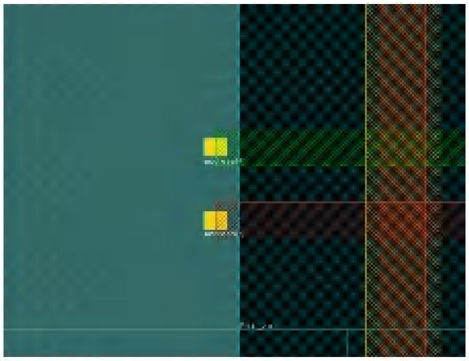
1. Views, modes, and corners at the top and partition levels must have same names.
2. When you use `create_constraint_mode` or `update_constraint_mode` to specify constraints for MMMC, you must specify the ILM constraints using the `-ilm_sdc_files` parameter (that is, timing in the presence of ILMs gets constraints from the `-ilm_sdc_files` parameter, not the `-sdc_files` parameter). The .sdc files specified with the `-ilm_sdc_files` parameter should be the constraint file of the full-chip flat netlist where it allows referencing nets or pins internal to the ILM model.

If you want to see the LEF pins of the ILM in GUI, the design must be in the unflattened mode.

The following figure shows the flattened and unflattened ILM. The LEF pins of the ILM are visible after unflattening the ILM.



Flattened ILM



Unflattened ILM

ILMs Supported in SI

ILM supports the SI aware 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 postRoute optimization scripts should run successfully in the presence of ILMs (without any additional changes).

The following command can be used to get timing reports containing the SI push-out delays on nets using the `setIlmType -model` command:

```
# Reflattens to SI model, then does not unflatten (All other design
# commands unflatten upon exit, regardless of the flattened/unflattended
# state before invocation)
  setDelayCalMode -SIAware true
  timeDesign -postroute
```

```
# Adds incremental delay column (for SI push-out delays) in timing output:

  set_global report_timing_format {instance arc cell delay arrival required}

# 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 `setExtractRCMode` settings determine whether the SI model is generated (the extraction mode will not be changed internally to generate the SI model).

ILM Model

ILM model contains both timing and CTS information instead of having a separate model for each of them to reduce the disk usage. In this model, the timing and CTS models have been merged by only the:

- interface paths from the timing model.
- best/worst latency registers that are to be kept as a part of the CTS model. All other registers are excluded.
- worst inter-clock paths for the timing model.

Interactive Use of ILMs

When `setIlmMode -keepFlatten` is set to true, the flow will work under the flattenIlm mode. As a result:

- The number of flatten/unflattenIlmunflatten_ilm calls inside super commands, such as `place_opt_design`, `timeDesign`, `optDesign`, `ccopt_design`, is reduced.
- Super commands except `assembleDesign` will return in the same ILM mode as it was.
- Once the design is flattened, it will be kept in the flattened state until you run unflattenIlm. This will help reduce the number of flatten/unflattenIlm operations during the timing closure where you can run `timeDesign`, interactive debugging, setting additional SDC constraints, manual ECO in the flattenIlm mode.
  Note: For multi-stage scripts, you need to set this option only once.

The list of commands for which Innovus will automatically unflatten ILMs/flattenILMs when `setIlmMode -keepFlatten` is set to `true` is given below:

Handling Interactive Constraints

You cannot specify the interactive constraints when ILMs are not flattened (unflattenIlm). In the flattened mode (flattenIlm), you can specify both interactive and modeless constraints and these constraints are used during various cycles of unflattenIlm to flattenIlm. During `saveDesign`, these constraints are honored.

To specify additional constraints while running unflattenIlm, set the following:

```
set_global timing_defer_mmmc_object_updates true
```

```
set_interactive_constraint_modes [all_constraint_modes -active|or
your_own_list_of_modes]
```

```
foreach mode {list_of_modes_to_be_updated} {update_constraint_mode -name $mode
-ilm_sdc_files \
[concat get_constraint_mode -name $mode -ilm_sdc_files] additional.sdc]
}
```

Include all mode-less constraints such as `timing_derates` and `group_path` in a separate file, and run:

```
source <mode-less_constraint.sdc>
set_global timing_defer_mmmc_object_updates false
set_analysis_view -update_timing
```

Top-level Timing Closure Methodologies for iHDB Flow

Using Interface Logic Models (ILM)

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 nets 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.

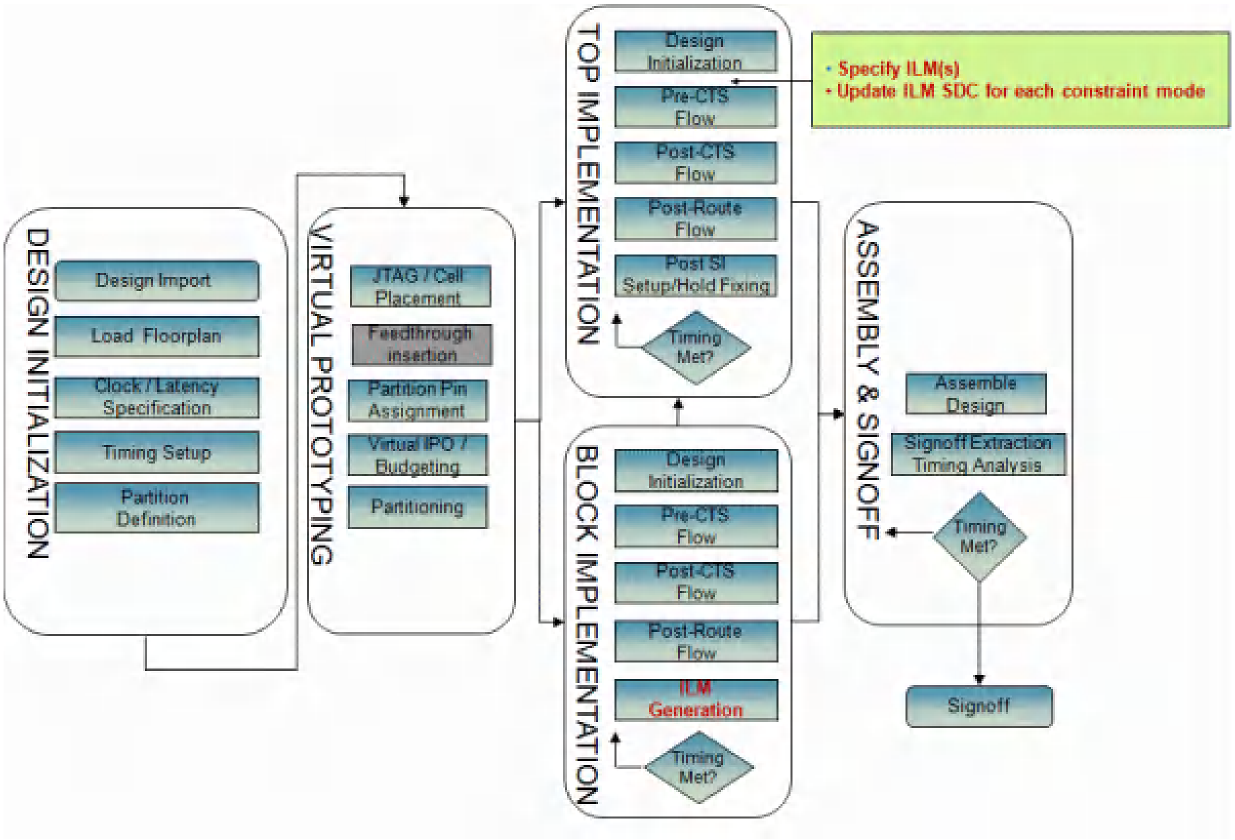
The advantages of using ILMs are as follows:

- More accurate analysis than a blackbox flow
    - More SI aware than combined `.lib` or `.cdb` approach
    - Can model clock generator inside block
    - More accurate timing and SI reduces the number of design iterations to close timing and SI.
- No need to characterize blocks
    - Works on an actual design data
- Can be used in the initial prototyping stage for very big designs, when loading full design data is not feasible.

  - Allows you to modify only top-level data
  - Fully preserves implemented partitions
- Uses the original constraint file for top-level analysis
  - No abstraction for timing exceptions

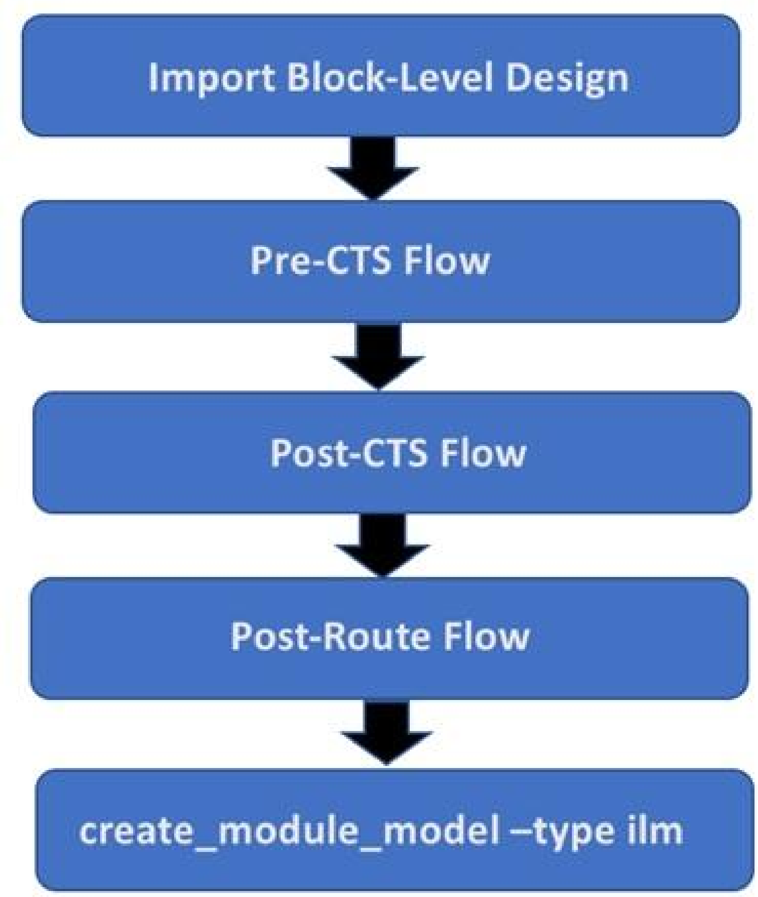
General ILM Flow

ILM is used for top-level timing closure. Models will be generated during the block implementation stage and will be used at the top implementation stage. Following is the general ILM foundation flow:



ILM Generation at Innovus Partition Block-level Design

Use the `create_module_model -type ilm` command to generate ILM at the Innovus partition block level design.
Two types of models can be generated, ILM model and SI model. The ILM model will be used for timing analysis and clock tree synthesis, and the SI model can be used at the postRoute stage for SI analysis. The SI model can only be generated if the block-level design is already SI-aware optimized.



Note: User can also create ILM models after pre-cts, and post-cts flow.

Here is an example flow script for ILM generation:

```
restore_module_model $block -tag init

place_opt_design

create_module_model -tag my_preCts
create_module_model -tag my_preCts -type ilm

ccopt_design

create_module_model -tag my_postCts

create_module_model -tag my_postCts -type ilm

routeDesign

setDelayCalMode -SIAware false

optDesign -postRoute

verify_drc

verifyConnectivity

create_module_model -tag postRoute_v1

create_module_model -tag postRoute_v1 -type etm

create_module_model -tag postRoute_v1 -type ilm
```

ILM Integration at Top-level Design

Once the models have been generated, they can be specified at the top-level design for timing closure:

1. Specify a block as an ILM using the `set_module_model` command.

   ```
   set_module_model -cell tdsp_core   -tag my_preCts -type ilm
   set_module_model -cell ptn_wrapper -tag pre_cts_v2 -type ilm
   ```

2. Commit the specified ILMs using the `commit_module_model` command to load them into the design. The full chip SDC constraint files can be specified with the `commit_module_model` command using the `-mmmc_file` option.

   ```
   commit_module_model -mmmc_file design/dtmf_recvr_core.enc.dat/viewDefinition.tcl
   ```

   Note: If the MMMC file does not have the `-ilm_sdc_files` information, full chip constraint files must be loaded using the `create_constraint_mode` or `update_constraint_mode` command after the ILM models are specified and before running the `commit_module_model` command as follows to avoid the redundant ILM SDC reading:

   ```
   set_global timing_defer_mmmc_object_updates true
   foreach mode [all_constraint_modes] {update_constraint_mode -name $mode -
   ilm_sdc_files <full_chip_sdc_file>}
   commit_module_model
   ```

```
set_global timing_defer_mmmc_object_updates false
```

3. Continue with the normal flow.

> ⚠
>
> - ILM SDC file(s) are used in the flattened ILM view.
> - Timing results are not available if the ILM SDC file is not defined.

Here is an example flow script for top-level timing closure using ILM:

```
set_module_model -default_dir /myproject/DATA
restore_module_model dtmf_recvr_core -tag init
set_module_model -cell tdsp_core -tag my_preCts -type ilm
set_module_model -cell ptn_wrapper -tag preCts_v2 -type ilm
commit_module_model -mmmc_file <full_chip_view_definition_tcl_file>

setDesignMode -process 65
setAnalysisMode -analysisType onChipVariation -cppr both

place_opt_design
create_module_model -tag top_pod

set_module_model -cell tdsp_core -tag my_postCts -type ilm
set_module_model -cell ptn_wrapper -tag postCts_v2 -type ilm
delete_ccopt_clock_tree_spec
commit_module_model

ccopt_design
create_module_model -tag top_postCts

set_module_model -cell tdsp_core -tag postRoute_v1 -type ilm
set_module_model -cell ptn_wrapper -tag postRoute_v2 -type ilm
commit_module_model

routeDesign
optDesign -postRoute

setExtractRCMode -engine postRoute
extractRC
timeDesign -postRoute

create_module_model -tag final
```

Creating ILMs

In the hierarchical design flow, you create a detailed block-level implementation of a block, then specify the `create_module_model -type ilm` command to create an ILM for the block. This command creates the ILM model under the central data repository directory.

You can also create ILMs for blocks that are in an intermediate stage of design, then use the data at the top level of the design for preliminary timing optimization.

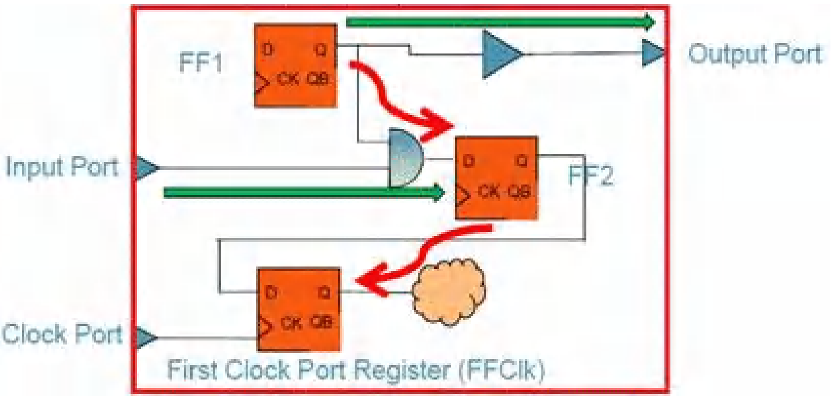
> ⓘ An ILM created for an incomplete block is not as accurate as an ILM created for a complete block. Always use ILMs for complete blocks to complete the top-level design.

The software generates ILM data for CTS, signal integrity, and other design stages (preCTS, postCTS, postRoute)

- ILM data for preCTS, CTS, postCTS, and postRoute
  The model contains the netlist of the circuitry leading from the I/O ports to interface sequential instances (that is, registers or latches), and from interface sequential instances to I/O ports. The clock tree leading to the interface registers is preserved.

  In case of CTS, the timing and CTS models have been merged to reduce the disk usage of an ILM model. The CTS data is limited to the worst clock sinks and instances/nets leading to those sinks.

  In general, internal register-to-register paths will not be kept in the ILM model. However, in special cases some internal paths will be kept as shown in the example given below. In the example, FF1 and FF2 will be kept as they are the interface flops that connect to the output port and/or input port. Also, FFClk is kept as for each clock port we need to keep at least one register for timing budget. As a result, we will have two internal register-to-register paths (shown with red arrows).



- ILM data for SI
  The model includes all the above, plus aggressor drivers or nets which affect I/O paths. It also includes the timing window files in the ILM model directory.

Note: When the `create_module_model -type ilm` command is called, all views are generated for multi-corner, multi-mode (MMMC) analysis.

Example ILM Creation

The following method creates a model that can be used in the top-level implementation flow by both `timeDesign` and `optDesign` for setup effort, including postRoutepost_route SI optimization. This model is also used during `ccopt_design`.

```
set_module_model -default_dir /myproject/DATA

restore_module_model $block -tag init
place_opt_design
…
create_module_model -tag my_postroute -type ilm
```

Sample Summary Report

The following is a sample summary report generated at the end of the `create_module_model -type ilm` command:

```
--------------------------------------------------------------
               create_module_model -type ilm Summary
--------------------------------------------------------------

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 the total 7621 instances for this block have been eliminated, only 468 instances are written to the Verilog netlist, out of which 111 instances are registers.

This summary report applies to a block using MMMC. Therefore, views with worst reduction ratio are displayed for each model.

ILM Generated Data

ILM Generation provides the following output data:

File Extension	Description

.def	One def file
.v	One netlist file
.place	One Innovus place file
.sdc	One per analysis view
.spef	One per corner
.xtwf	One per analysis view. This file is only available with SI model

Preserving Selected Instances in ILMs

You can force the selected instances and nets to be included in the ILM model:

1. Select instances or nets using the `selectInst` or `selectNet` commands.
2. Specify the following commands:

```
set_module_model -default_option {-keepSelected} -type ilm
create_module_model -tag my_cts -type ilm
```

Creating ILMs for Shared Modules

You can use the same sub-block module in different ILM blocks, enabling reuse of versatile modules. The `create_module_model -type ilm` command considers constant propagation, so that only the enabled parts of a module are considered when creating ILMs for the reused modules. Because the Innovus database cannot handle the same module name in different circuits, the software automatically modifies the module names with the following rule:

```
topModuleName+timestamp+$+moduleName
```

As an example, one ILM block (`ModuleA`) uses an ALU module (ALU) as an unsigned ALU, and a second block (`ModuleB`) uses the ALU as a signed ALU. You can change the input signal to use the ALU differently, setting one ALU as sign enabled and the other to off. When you run the `create_module_model` 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.

Note: With Integrated Hierarchical Database (iHDB), ILM models cannot be imported from third-party tools.

Specifying ILM Directories at the Top Level

Use the `set_module_model -type ilm` command for ILM data of a block at the top partition level rather than using the default .lib model. You can run the `set_module_model -type ilm` command multiple times in the same session. Each time you run this command, the software overwrites the previous setting for the same block. If master/clones exist in the design, the cell name will have the name of the master partition.

Once ILMs are specified, the `commit_module_model` command should be invoked to load specified ILM models. To revert specified ILM models back to .lib model of the block, use the following commands;

```
set_module_model -type lef -add_ons etm
```

Example: Top-level Implementation Flow with ILMs

1. Before you start the Innovus tool, prepare the top-level Verilog file, if needed. If you use the Innovus hierarchical flow in a previous Innovus session, then the `savePartition -module_model_tag` command automatically creates the top-level data.
2. Start an Innovus session.
3. Load the design, including the top-level netlist, and ILM specification.

   ```
   restore_module_model dtmf_recvr_core -tag init
   set_module_model -cell tdsp_core -tag my_preCts -type ilm
   set_module_model -cell ptn_wrapper -tag my_preCts -type ilm

   set_global timing_defer_mmmc_object_updates true
   foreach mode [all_constraint_modes] {
   update_constraint_mode -name $mode -ilm_sdc_files <full_chip_sdc_file>
   }
   commit_module_model
   set_global timing_defer_mmmc_object_updates false
   ```

4. Place and optimize the design.

   ```
   place_opt_design
   ```

5. Optionally save a design

```
create_module_model -tag top_preCts
```

6. Build the clock tree.

```
set_module_model -cell tdsp_core -tag my_postCts -type ilm
set_module_model -cell ptn_wrapper -tag my_postCts -type ilm
delete_ccopt_clock_tree_spec
commit_module_model

ccopt_design
create_module_model -tag top_postCts
```

7. Route the design.

```
set_module_model -cell tdsp_core -tag postRoute_v1 -type ilm
set_module_model -cell ptn_wrapper -tag postRoute_v1 -type ilm
commit_module_model
routeDesign
```

8. Run SIAware inside pre-route optimization and post-route optimization with hold.

```
setDelayCalMode -SIAware true
optDesign -post_route
optDesign -post_route -hold
setExtractRCMode -engine postRoute
extractRC
timeDesign -postRoute
create_module_model -tag top_postRoute
```

Nested ILM Support

Stylus Hierarchical Data Base provides an easy way for supporting design with multiple levels of hierarchy where an ILM can be nested inside another ILM.

1. First implement all block-level designs and generate their ILMs using the `create_module_model -type ilm` command.
2. At top-level design, specify all ILMs including nested ones using the set_module_model command.
3. Run the `commit_module_model` command to load all ILMs including nested ones to the top-level design where Innovus will automatically stitches all the ILM SPEFs together.
4. Run the normal top-level timing closure flow.

   Example: `dtmf_recvr_core design` has `tdsp_core` and `ptn_wrapper` as first-level of physical hierarchy. Inside ptn_wrapper design, it has ram_128x16_test cell as second-level of physical hierarchy.

- **Note:** The ILM model should be a donut model. If a block-level design has specified ILM model, when generating an ILM model for this design, the `create_module_model -type ilm` command will automatically treat the ILM as a LEF.

  For example, consider a block-level design, ptn_wrapper, that has ram128X16_test as an ILM. Specifying the `create_module_model -type ilm` command will generate an ILM model for ptn_wrapper with ram128x16_test as a LEF.

- At top-level design, do the following:

```
set_module_model -default_dir /myproject/DATA
restore_module_model dtmf_recvr_core -tag init
setOptMode -opt_skew false
#setDesignMode -earlyClockFlow true
set_module_model -cell tdsp_core -tag my_preCts -type ilm
set_module_model -cell ptn_wrapper -tag my_preCts -type ilm
set_module_model -cell ram_128x16_test -tag my_preCts -type ilm
commit_module_model -mmmc_file design/dtmf_recvr_core.enc.dat/viewDefinition.tcl
reportIlmStatus
place_opt_design
…
```

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_view -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` or `update_constraint_mode` to specify constraints for MMMC, you must specify the ILM constraints using the `-ilm_sdc_files` parameter (that is, timing in the presence of ILMs gets constraints from the `-ilm_sdc_files` parameter, not the `-sdc_files` parameter). The .sdc files specified with the `-ilm_sdc_files` parameter should be the constraint file of the full-chip flat netlist where it allows referencing nets or pins internal to the ILM model.

ILMs Supported in SI

ILM supports the SI aware for `optDesign` and `timeDesign`. These commands automatically switch to SI ILM models during postRoute stage. Therefore, your present postRoute optimization scripts should run successfully in the presence of ILMs (without any additional changes).

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 `setExtractRCMode` settings determine whether the SI model is generated (the extraction mode will not be changed internally to generate the SI model).

ILM Model

ILM model contains both timing and CTS information instead of having a separate model for each of them to reduce the disk usage. In this model, the timing and CTS models have been merged by only the:

- interface paths from the timing model.
- best/worst latency registers that are to be kept as a part of the CTS model. All other registers are excluded.
- worst inter-clock paths for the timing model.

Handling Interactive Constraints

All timing related commands works in flattened or global view only. With dual-view ILMs user should stay in the flattened view throughout the top-level timing closure flow. To improve loading run time of SDCs, suggest to do the following to delay all SDC loading after all constraints are specified:

```
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_mode -name $mode
-ilm_sdc_files \
[concat get_constraint_mode $mode -ilm_sdc_files] additional.sdc]
}
```

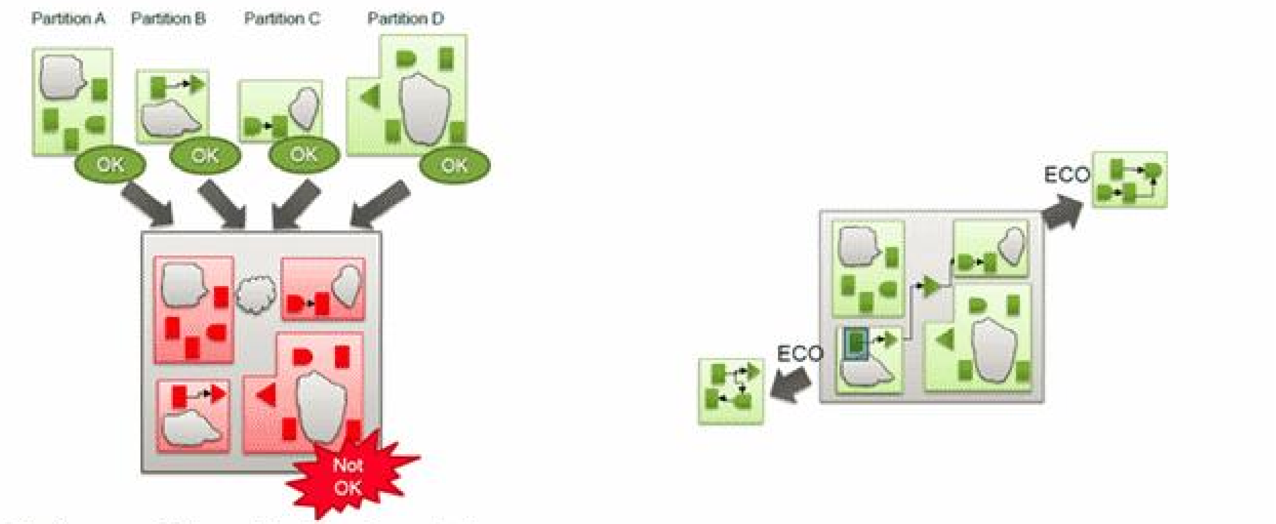
Include all mode-less constraints such as `timing_derates` and `group_path` in a separate file, and run:

```
source <mode-less_constraint.sdc>
set_global timing_defer_mmmc_object_updates false
set_analysis_view -update_timing
```

Using Flexible Interface Logic Models (FlexILM)

Overview

In the top-level timing closure, timing budgeting is not accurate and inter-partition critical paths are hard to close. After partition, blocks are implemented in a number of days and are modeled as ILM blocks at the top-level design for the top-level timing closure. ILM blocks are read-only, and as a result, top-level timing issues are not fixed easily, especially in channel-less designs where there is no room for buffer insertion. Designers may need to do at least two or three passes of hierarchical flow to close timing. To address this challenge, a single-pass hierarchical solution with Flexible Interface Logic Models (FlexILM) can be used. FlexILM is a reduced netlist where logics on interface paths are kept and logics on internal paths are removed. At the top-level design, interface paths of FlexILMs can be optimized, and netlist and placement changes can be ECO back to partition blocks automatically. FlexILM also reduces the memory in timing graph and physical data where removed instances are replaced by placement blockages to avoid violations with the newly added optimized logics. Additionally, routing of removed nets will be replaced by RC grids to improve RC extracted correlation.



Challenge of Hierarchical Implementation

FlexILM Single-pass Hierarchical Implementation

The advantages of using FlexILM are as follows:

- Enables concurrent top and block optimization for interface paths
    - Fixes inter-partition timing critical paths early in the flow where interface logics can be

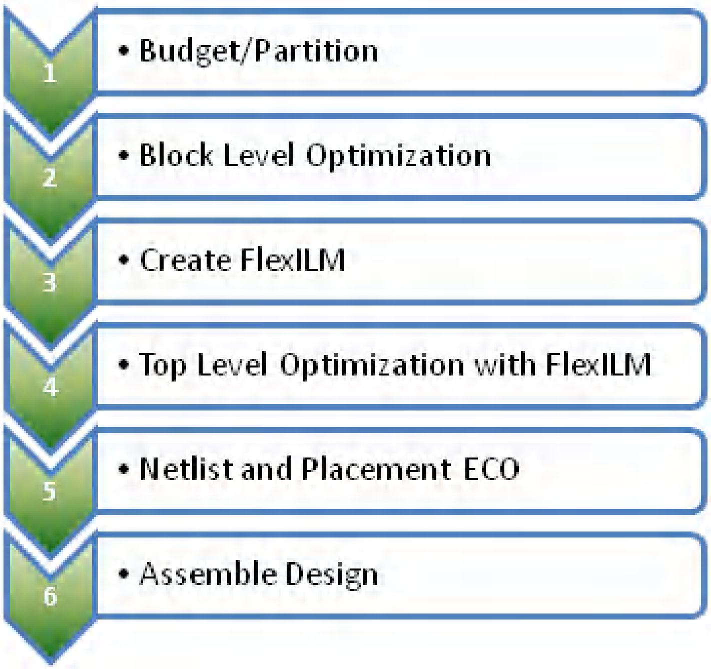
touched

- Does not need accurate timing budget
- Is the only solution for a channel-less design

Currently, FlexILM does not provide support for master/clone, route, and CCopt with useful skew, and there is minor delay or RC correlation due to a different routing pattern.

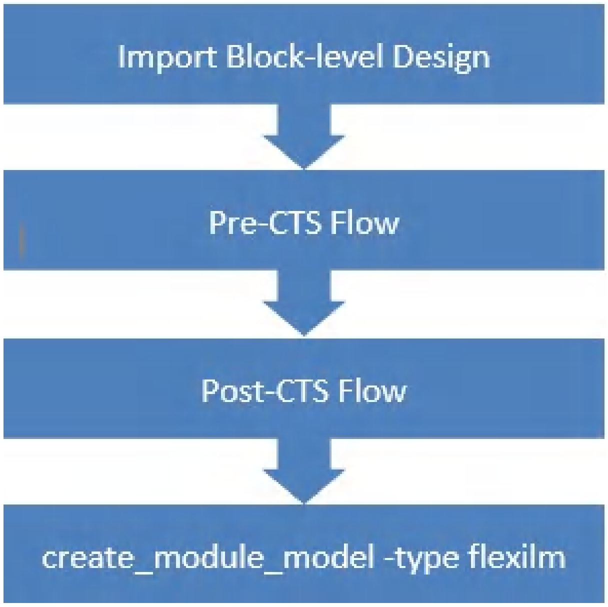
General FlexILM Flow

FlexILM is used for the top-level timing closure. Models should be generated during the block implementation stage and will be used at top-level design for timing closure. Following is the general FlexILM flow:


1
• Budget/Partition
2
• Block Level Optimization
3
• Create FlexILM
4
• Top Level Optimization with FlexILM
5
• Netlist and Placement ECO
6
• Assemble Design

FlexILM Model Creation

FlexILM can be generated at the Innovus partition block-level design using the `create_module_model -type flexilm` command. FlexILM supports the preCTS and/or postCTS optimization stage. It converts reduced instances to placement blockages and reduced net wires to RC Grid.



Following is an example flow script for generating FlexILM:

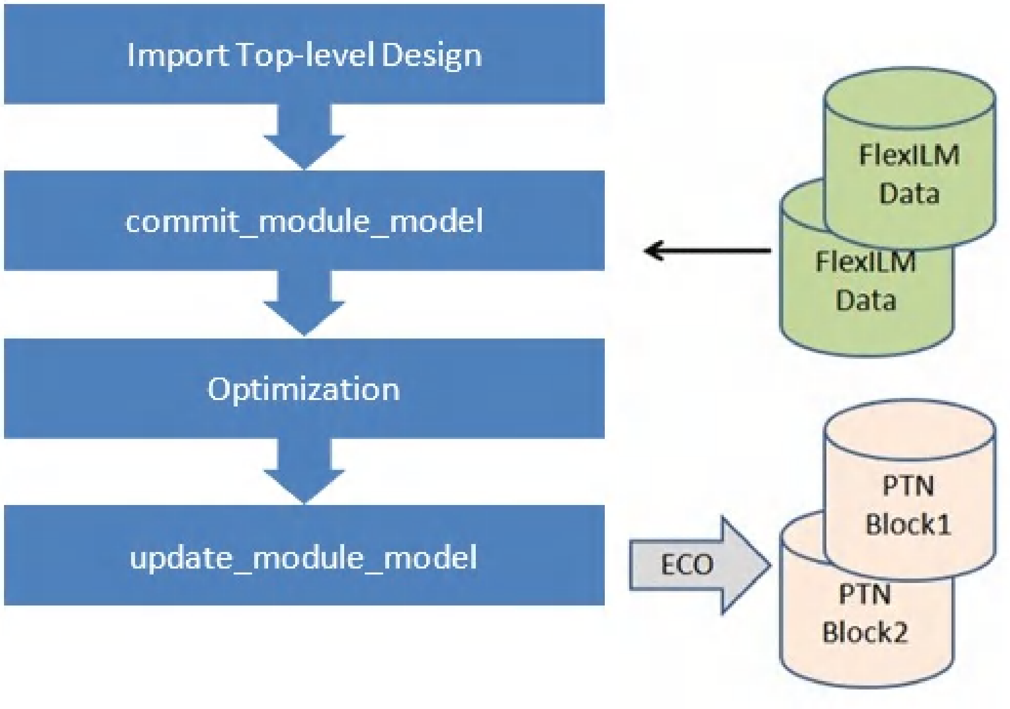
```
set_module_model -default_dir /myproject/DATA
set_module_model -default_options {-optStage preCTS} -type flexilm
restore_module_model $block -tag init

place_opt_design

create_module_model -tag my_preCts
create_module_model -tag my_preCts -type flexilm
```

Note: The `-optStage` parameter is mentioned to specify the stage at which the flexILM model will be used, preCTS or postCTS.

Note: When the `create_module_model -type flexilm` command is called, all the views are generated for multi-corner, multi-mode (MMMC) analysis.

Top-Level Optimization

After creating FlexILM model from each partition, these models can be specified and committed at the top level using the `commit_module_model` command. Once the design is optimized, changes can be ECO back to block-level design using the `update_module_model` command.



Following are the main steps of the top-level optimization flow:

1. Set the central data repository

   ```
   set_module_model -default_dir /myproject/DATA
   ```

2. Restore the design for the top level.

   ```
   restore_module_model dtmf_recvr_core -tag init
   ```

3. Commit FlexILM model for all the blocks.

   ```
   set_module_model -cell * -tag my_prects -type flexilm
   commit_module_model -mmmc_file $dataDir/dtmf_recvr_core.enc.dat/viewDefinition.tcl
   report_module_model
   ```

   ⚠

   - The full chip MMMC view file should be specified with the `-mmmc_file` option when running commit_module_model.
   - You can use the following command to get back to the full block after you have committed the FlexILM model at the top level.

     ```
     set_module_model -cell * -type pnr -tag init
     commit_module_model
     ```

4. After committing FlexLM model, you optimize the design to improve timing by following the

steps given below:

a. Ensure that you keep logic ports during optimization.

```
set flexIlmList {tdsp_core_inst ram_128x16_test_inst}
foreach hi $flexIlmList { dbSet [dbGetHInstByName $hi].dontTouchHports true }
redirect {} > keepPorts.hinsts
redirect {foreach hi $flexIlmList {puts “$hi”}} >> keepPort.hinsts
```

```
setScanReorderMode -keepPort keepPort.hinsts
```

```
setScanReorderMode -enable_for_partition true
setRouteMode -earlyGlobalRoutePartitionHonorFence {.}
```

b. Run the `place_opt_design` command to optimize the design.

```
place_opt_design
```

5. Run the `update_module_model` command to update the partition blocks with ECO changes.

```
set blocks “tdsp_core ram_128x16_test”
foreach m $blocks {
update_module_model -cell $m -plugin_tcl {
setMultiCpuUsage -localCpu 8
ecoPlace
create_module_model -tag post_opt
create_module_model -tag post_opt -type flexilm -include_early_global_route
}
```

6. After net list changes had been ECO back to block-level designs, block designs with ECO changes can be brought back to the top-level design for chip assembly to check timing. Following is the sample script:

```
foreach m $blocks { set_module_model -cell $m -tag post_opt -type flexilm}
commit_module_model
dbDeleteTrialRoute
setRouteMode -earlyGlobalRoutePartitionHonorPin true
earlyGlobalRoute
timeDesign -preCTS
```

FlexILM Model Data

FlexILM generation provides the following output data:

File Extension	Description

.v	Netlist file
.sdc	One per constraint mode
.def	One DEF placement file
.rcg	One RC Grid file
.blkg	One placement blockage file

Using ILM ECO Methodology

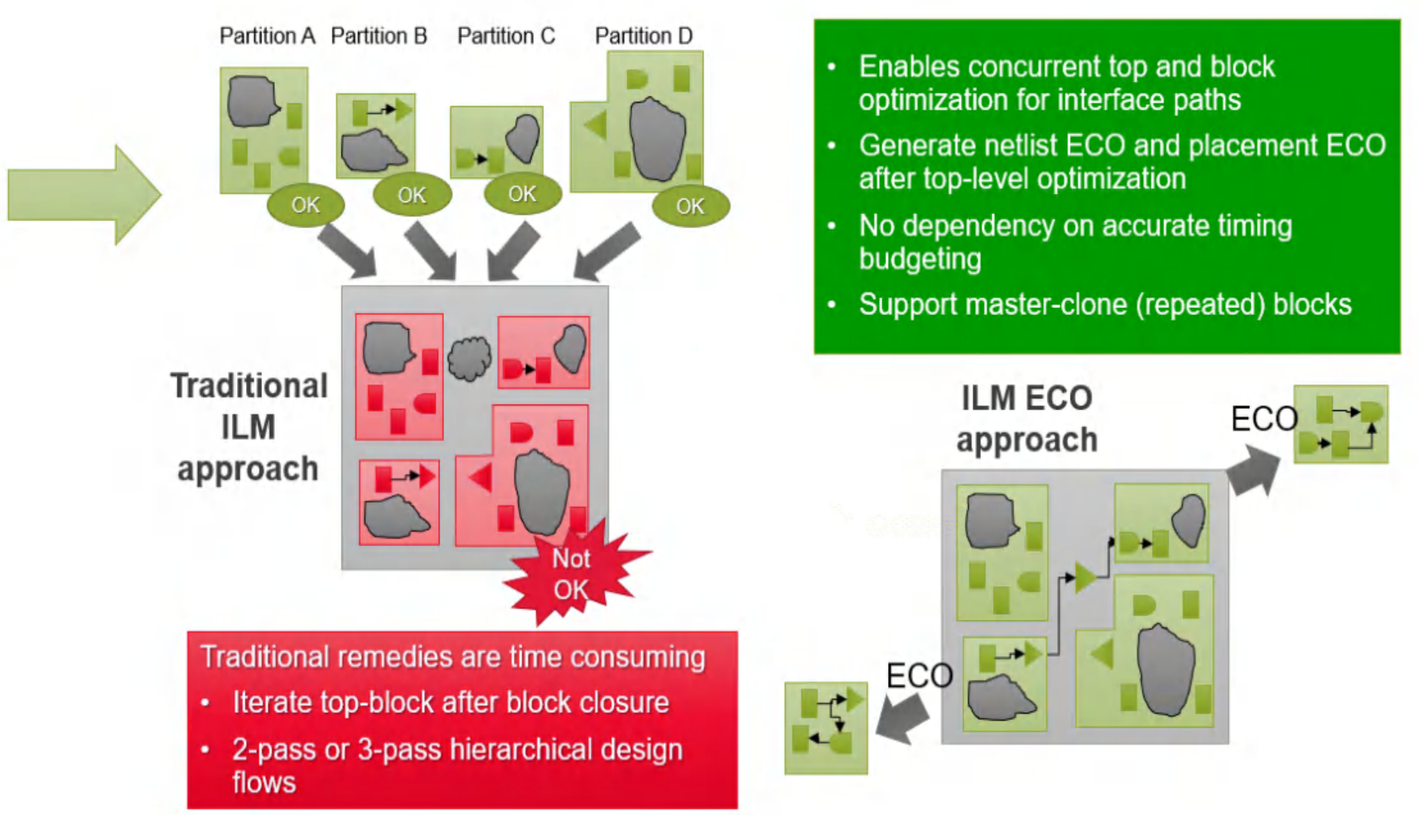
Overview

ILM model is read-only where optimizer cannot change the ILM block netlist. As a result, you may need to iterate few passes between the top- and block-level designs to close timing. ILM methodology has been enhanced to ILM ECO methodology. In ILM ECO methodology, Innovus optimizer can optimize ILM boundary interface logics and can ECO back the changes into partition blocks. The ILM ECO methodology supports:

- Post-route optimization style where it can add buffers and/or resize cell(s) to improve WNS/TNS
- Master/clones where all the ECO changes are applied to both master/clone hinsts

For ILM ECO operation:

- Logics between registers to registers are converted to dummy logic cells instead of removing them so refinePlace can check edge constraints when legalizing new added buffers
- ILM SPEF has node locations so the extractor can update SPEF for new added net(s)
- ILM placement/routing blockages are removed while actual blockages inside block design are preserved



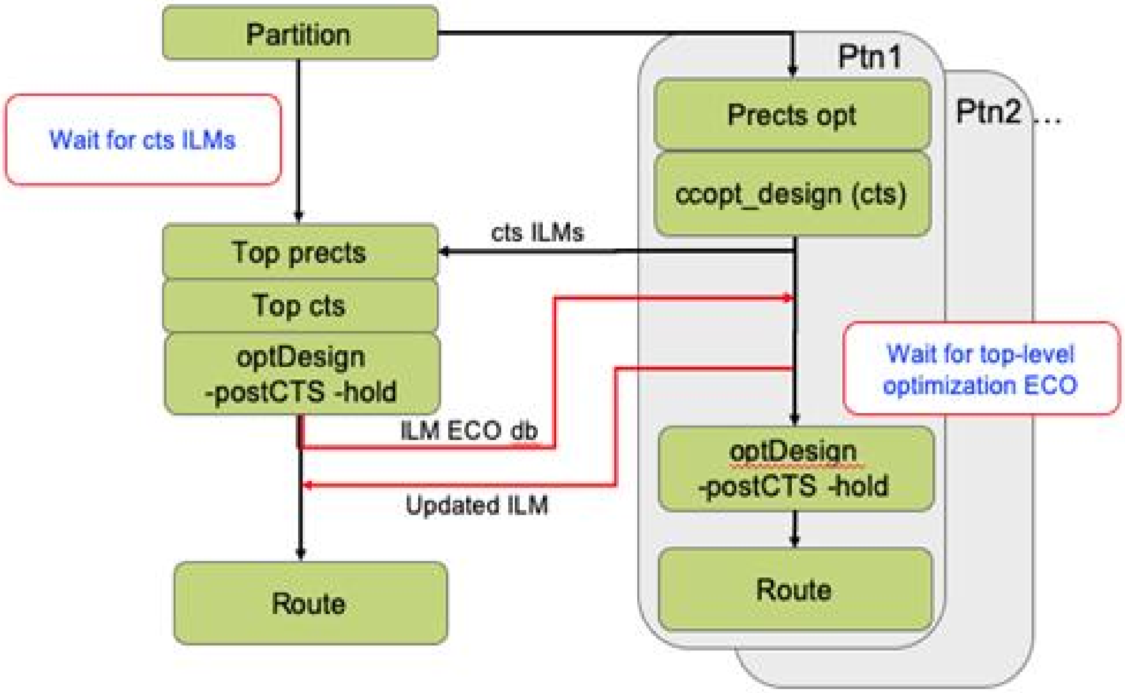
- Enables concurrent top and block optimization for interface paths
- Generate netlist ECO and placement ECO after top-level optimization
- No dependency on accurate timing budgeting
- Support master-clone (repeated) blocks

Traditional remedies are time consuming

- Iterate top-block after block closure
- 2-pass or 3-pass hierarchical design flows

ILM ECO Integrated in the Flow

When ILM ECO is integrated in the flow, the boundary timing issues can be fixed earlier in the flow. However, it is hard to execute top and block timing closure in parallel because of the ECO netlist consistency requirement.



ILM Model Generation for ILM ECO Flow

To generate an ILM model for ILM ECO flow, `-allowIlmEco` option should be specified:

```
set_module_mode -default_options {-allowIlmEco} -type ilm
```

```
create_module_model -type ilm -tag postcts
```

Note: The ILM model netlist generated with `-allowIlmEco` tends to be larger than the netlist of default ILM model as the optimizer requires to keep complete connection of kept netlist. Therefore, it is recommended to use `-allowIlmEco` only for the models that you plan to open-up for optimization to improve runtime/memory.

ILM ECO At Top-Level Design

After specifying ILMs at the top-level design, you need to specify the ILMs that can be opened for optimization by setting cell DB allowIlmEco flag/bit.

Usage model:

```
dbSet [dbGetCellByName <cellName>] .allowilmEco true
```

Optimizer checks this flag/bit of an ILM hinst. If it is true, optimizer optimizes the ILM boundary interface logics to improve timing.

Note: This bit setting has higher precedence than any other DB ILM dont touch settings.

Saving the ILM ECO Information

The `write_ilm_eco_db` command saves the netlist and placement information changes at the top-level design. The `write_ilm_eco_db` command generates a binary ECO file that can be read/loaded back to a block-level design.

- Generate ECO TCL script with `read_ilm_eco_db -module_model_tag` for iHDB.
- Also output a corresponding ECO report file for viewing purpose.

Sample Scripts

Here is a sample script for the block-level ILM ECO model generation:

```
    set_module_model -default_dir DATA2

    # Restore block-level design generated by savePartition -module_model_tag command

restore_module_model <block1Name> -tag init

    #place_opt_design

    #ccopt_design

    # Generate an ILM to be used at top-level design so top-level implementation can

    #  be ran in parallel

    set_module_mode -default_options {-allowIlmEco} -type ilm

create_module_model -type ilm -tag postCTS
```

Here is a sample script for the top-level design implementation with ILM ECO:

```
    set_module_model -default_dir DATA2

    restore_module_model <topDesign>

    set_module_model -cell <block1Name> -tag postcts -type ilm
    set_module_model -cell <block2Name> -tag postcts -type ilm

    commit_module_model

    # Specify which ILMs can be optimized

    dbSet [dbGetHInstByName <block1HinstName>].cell.allowIlmEco true

    dbSet [dbGetHInstByName <block2HinstName>].cell.allowIlmEco true

    place_opt_design

    ccopt_design

optDesign -postCts -hold


create_module_model -tag postcts_hold

    # Recommend to write out ECO file(s) after postCts stage

    write_ilm_eco_db -module_model_tag postcts
```

Here is a sample script for ILM ECO implementation and verification:

For block-level ILM ECO implementation:

```
    set_module_model -default_dir DATA2

    # Restore block-level design generated by savePartition -module_model_tag command

    restore_module_model <block1Name> -tag postcts -add_ons {eco}

    #place_opt_design

    #ccopt_design

    # Generate an ILM to be used at top-level design so top-level implementation can

    #  be ran in parallel

    set_module_mode -default_options {-allowIlmEco} -type ilm

    create_module_model -type ilm -tag postcts_eco
```

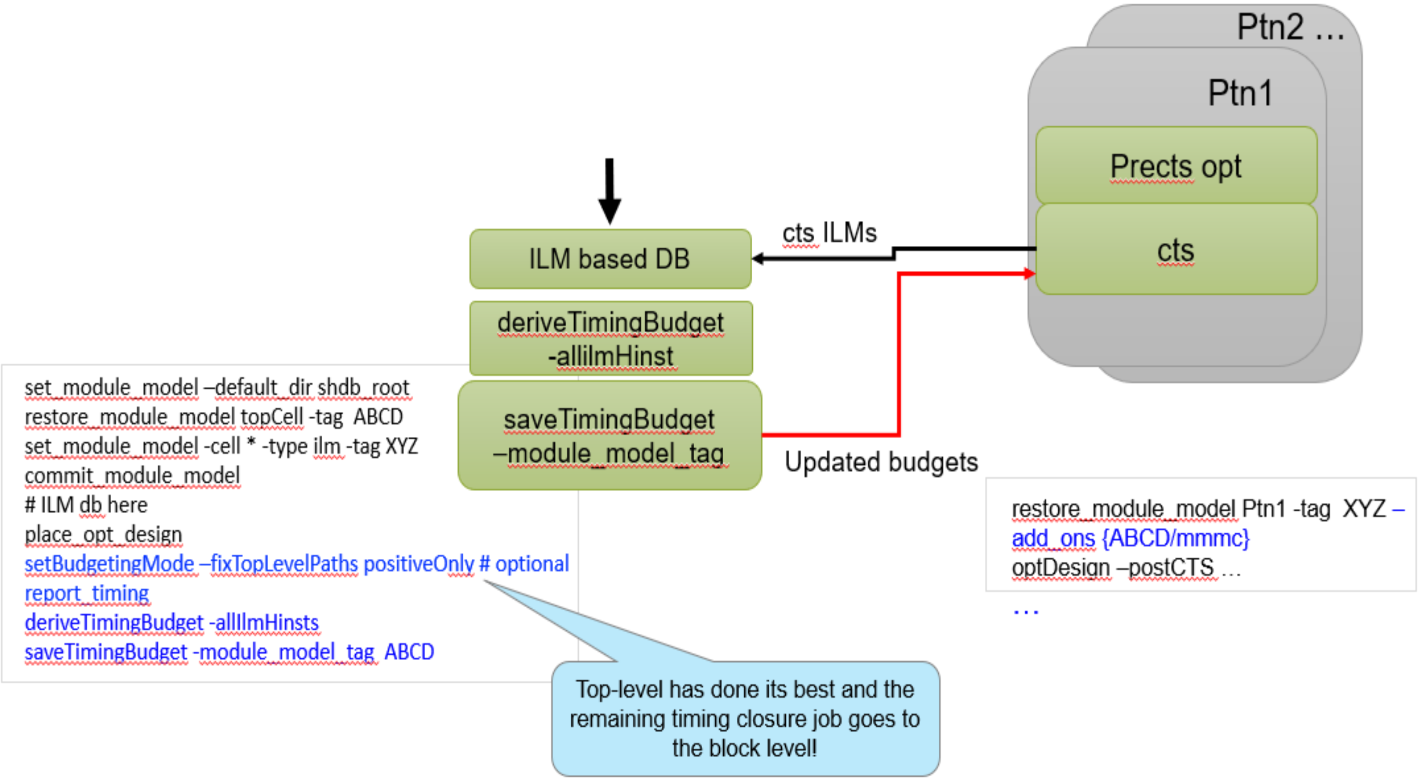
For Merged Top-Level Timing Verification

```
    set_module_model -default_dir DATA2

    restore_module_model <topDesign> -tag postcts_hold

    set_module_model -cell <block1Name> -tag postcts_eco -type ilm
```

```
set_module_model -cell <block2Name> -tag postcts_eco -type ilm

commit_module_model

...

timeDesign -postCTS

create_module_model -tag postcts_hold_eco
```

ILM-based Timing Re-Budgeting

Optionally, timing budgeting can be re-generated for all ILMs based on the current optimized design.



```
set_module_model –default_dir shdb_root
restore_module_model topCell -tag  ABCD
set_module_model -cell * -type ilm -tag XYZ
commit_module_model
# ILM db here
place_opt_design
setBudgetingMode –fixTopLevelPaths positiveOnly # optional
report_timing
deriveTimingBudget -allIlmHinsts
saveTimingBudget -module_model_tag  ABCD
```

```
restore_module_model Ptn1 -tag  XYZ –
add_ons {ABCD/mmmc}
optDesign –postCTS ...
...
```

Top-level has done its best and the remaining timing closure job goes to the block level!

Sample Scripts

Rebudgeting and ECO

Here is a sample script top-level design implementation with ILM ECO:

```
set_module_model -default_dir DATA2

restore_module_model <topDesign>

set_module_model -cell <block1Name> -tag postcts -type ilm

set_module_model -cell <block2Name> -tag postcts -type ilm

commit_module_model -mmmc_file <fullChipViewDefinition>


# Specify which ILMs can be optimized

dbSet [dbGetHInstByName <block1HinstName>].cell.allowIlmEco true

dbSet [dbGetHInstByName <block2HinstName>].cell.allowIlmEco true

place_opt_design

ccopt_design

optDesign -postCts -hold

create_module_model -tag postcts_hold

# Optional to re-budget timing for the block design

setBudgetingMode -fixTopLevelPaths positiveOnly

report_timing

deriveTimingBudget -allIlmHinsts

saveTimingBudget -module_model_tag postcts

# Recommend to write out ECO file(s) after  postCts stage

write_ilm_eco_db -module_model_tag postcts
```

Here is a sample script for block-level ILM ECO implementation with updated budgets

```
set_module_model -default_dir DATA2

restore_module_model <block1Name> -add_ons {mmmc eco} -tag postcts

# If  block1 post CTS hold step is running parallel with top-level design then
block1 postcts hold should be loaded instead

# restore_module_model <block1Name> -tag postcts_hold \

# -add_ons {postcts/mmmc postcts/eco}
```

...

Extracting Timing Models

Overview

Timing model extraction is the process of abstracting the interface timing of hierarchical blocks in a timing library. Typically, model extraction has the following advantages:

- Reduces the memory requirements by generation of extraction timing models (ETMs) for respective blocks, which may be huge in size.
- Reduces the static timing analysis (STA) run time.
- Hides the proprietary implementation details of the block from a third-party.

The `do_extract_model` command is used to build a timing model of a digital block to be used with a timing analyzer. The automatic derivation and extraction of the "actual" timing context for a lower-level module instance is required for achieving timing convergence with large design databases.

The software has the following advantages for timing extraction:

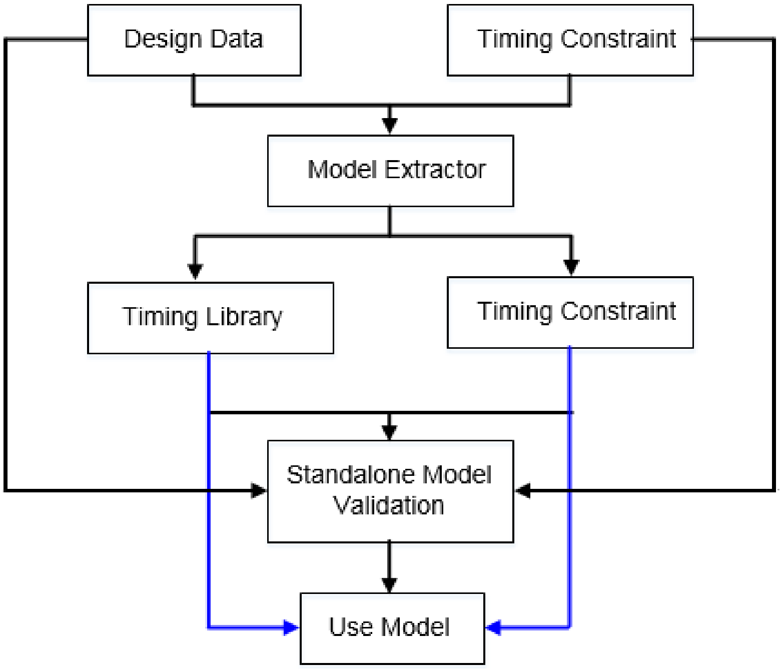
- Provides a fast timing engine that allows for quick derivation of a module's timing context.
- Extracts data correctly across multiple clock domains.
- Allows merging of various models in various modes for their respective blocks.

You can start with model extraction with the following data loaded in the software:

- Design netlist.
- Timing libraries.
- Block context (i.e. constraints like clocks, path exceptions, operating conditions etc.)
- RC data of the nets.

The following diagram illustrates this flow:

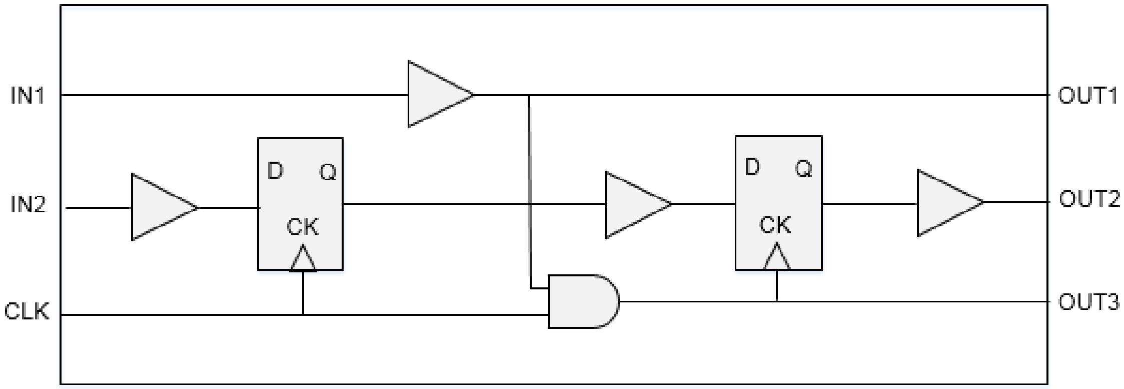
Model Extraction Flow



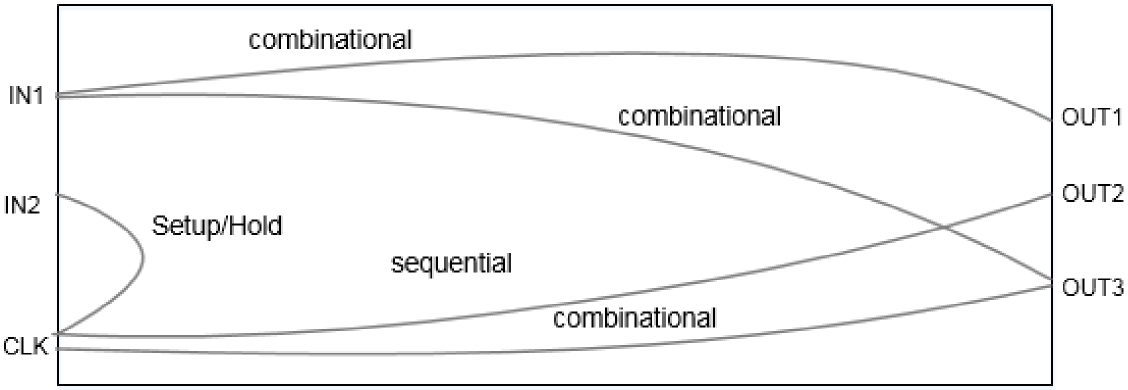
ETM Generation

ETM represents the timing for interface paths. Input to flop/gate, input to output and flop/gate to output paths of a design are preserved as timing arcs in the ETM. If there are multiple clocks capturing the data from an input port then an arc with respect to each input port is extracted.

The flop-to-flop type of paths are not written in the ETM, as these do not affect the interface paths timing. The following figure shows the equivalent ETM for a given netlist:



The following figure shows the Extracted Model:



The timing model extracted is context independent as it gives the correct value of arcs/delays, even with varying values of input transitions, output loads, input delays, or output delays. In such cases, it is not required to re-extract the model if some of the context gets changed at a later stage of development.

However, the model depends upon the operating conditions, wire load models, annotated delays/loads, and RC data present on internal nets defined in the original design. So if these elements change at the development stage of design then we need to re-extract the model for correlation with the changed scenario.

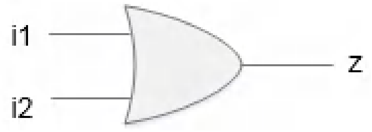
ETM Generation for MMMC Designs

In MMMC configuration, for generating ETM, only one view should be active. If you need to generate ETM for a specified view, however, the view is not active in setup as well as hold mode, then the software will show an error. It is a prerequisite that the view should be active for both setup and hold analysis. You can use the `set_analysis_view -setup {$viewName} -hold {$viewName}` command before running the `do_extract_model` command to achieve this.

After model extraction for MMMC designs, the dumped assertion files will be added with ${viewName} suffix, and the dumped derate related assertion files will be added with ${delayCornerName} prefix. You need to choose the corresponding file while using the extracted models.

Slew Propagation Modes in Model Extraction

Model extraction supports two modes of slew propagation - worst slew propagation and path-based slew propagation. These are described below.



Worst Slew Propagation

In worst slew propagation mode, the worst slew of all the incoming arcs at a converging point is taken to propagate further. For example, if at input pin i1 , the slew index was {1 3 5} and at input pin i2 the slew index was {1 2 6}, then the resulting slew at the output will be corresponding to the slew index {3 5 6}, which is worst of all the slew indices.

That is, slew (i1) = slew (i2) = max {(actual slew (i1->z), actual slew (i2->z)}.

Path-Based Slew Propagation

In path-based slew propagation mode, the actual slew for the path elements is propagated for the extracted arcs. For example, if at input pin i1 the slew index was {1 3 5} and at input pin i2 the slew index was {1 2 6}, then the resulting slew index for path through i1 will be {1 3 5} and through i2 will be {1 2 6}.

The slew propagation mode can be controlled by using the `timing_extract_model_slew_propagation_mode` global variable. The default value of this global variable is `worst_slew`.

Basic Elements of Timing Model Extraction

The following are the various basic elements which are relevant to model extraction:

- Nets – internal and boundary nets
- Timing Paths - check and delay paths – in2reg, in2out, and reg2out paths
- Minimum pulse width and periods checks
- Path Exceptions – false, multicycle, min delay, max delay, disable timing
- Constants
- Unconstrained Paths
- Clock Gating Checks

- Design Rules
- Clocks - created clocks and generated clocks

These are explained in the following sections.

Nets

Nets can mainly be classified into two types – boundary nets and internal nets. The nets which are directly connected to the input or output ports of the block are termed as the boundary nets. The nets which drive and are driven by some internal instance pin of the block are termed as the internal nets. Model extraction treats both the nets differently as boundary nets are always related and connected to the context.

Boundary Nets

The boundaries are connected to the context. So the RC data for them may change if the context of the block changes at a later stage. To be better context independent the model should not consider SPEF or DSPEF data for their delay calculation and a separate SPEF/DSPEF file for the boundary nets can be stitched to the top level data while instantiating the extracted model.

The software by default considers the RC data while extracting the model. The software provides a command line option which can be used to ignore the RC data for boundary nets by using the following setting:

```
setDelayCalMode -ignoreNetLoad true
```

Note: Currently, the software does not output the SPEF file for boundary nets which can be used at the top level. To solve this problem, you need to create the top level SPEF file for boundary nets of this block or use the default behavior to consider the boundary nets RC data for model extraction.

Internal Nets

As the internal nets connect the pins for the internal instance of the block, they are in no way dependent on the context environment. So the RC data defined for the internal nets is used as it is for delay calculation and is added in the extracted paths. If no RC is defined for these nets, then the wire load models are used to calculate their delays.

However, if a load or resistance is annotated using the `set_load`/`set_resistance` command, then it will override the annotated SPEF or the applied wire load model.

If the nets are annotated with the delays using the `set_annotated_delay` command or SDF annotation, then the annotated delay is used instead of the calculated delays. In case of incremental delays the addition of calculated and incremental delays is used.

Timing Paths

Timing paths are broadly divided in four types:

- In2reg
- reg2out
- in2out
- reg2reg: The reg2reg paths are not relevant to the model extraction process.

These are described below.

In2Reg Paths

A In2reg path is a setup/hold check that starts from an input port and is captured at a flop or gating element by a clock. So the in2reg type of paths are captured in equivalent setup or hold checks. The setup/hold values to be written in ETM are calculated using the data path delay, the setup/hold value of the library cell and the clock path delay. The equations can be written as follows.

Setup arc value = data path delay (input to flop) + setup value of flop – clock path delay (clock source to clk pin)
Hold arc value = data path delay (input to flop) - hold value of flop – clock path delay (clock source to clk pin)

The value for these arcs is the function of the transition on the input port and the transition at the clock source. If a flop is captured by multiple clocks then separate setup/hold arcs are extracted with respect to each clock source.

Reg2out Paths

The reg2out paths are paths starting from a register and ending up on an output port. These paths are a combination of trigger arcs (of starting register) and the combinational delay from sink of trigger arc to the output port. So for such paths an equivalent trigger/sequential arc is modeled in the extracted model. The delay for the arc will be equal to:

Sequential arc delay = delay (clock source to CK of register) + delay (register CK pin to out port).

The delay for these arcs is a function of the slew at the clock source and the capacitance at the output port. As the extracted model can be used for max as well as min analysis, two arcs are preserved in the model to represent the longest and the shortest path. Different types (rising_edge/falling_edge) of arcs are extracted for the different valid clock edges.

In2Out Paths

The in2out paths are the path starting from an input port and ending up on an output port. These paths are pure combinational paths. So for such paths an equivalent combinational arc is modeled in the extracted model. The delay for the arc will be equal to:

Combinational arc delay = delay (delay of all elements in the path)

The delay for these arcs is a function of the slew at the input port and the capacitance at the output port. As the extracted model can be used for max as well as min analysis, two arcs are preserved in the model to represent the longest and the shortest combinational path. In case if no path exists for a particular transition (rise/fall), half unate (combinational_rise /combinational_fall) arcs will be extracted. The timing sense for the arc will depend on the function of worst (early/late) paths.

Minimum Pulse Width and Period Checks

The minimum pulse width and period constraints defined at the CK pin of the registers are transferred to the clock source pins during model extraction.

There may be several different type of registers in the fanout of a clock source. So while transferring the minimum pulse width to the clock source you can use the worst values present on the fanout registers. The pulse width is calculated as:

pulse width = MAX(Arrival time of Launch clock Path - Arrival time of Capture Clock Path + pulse width at clock pins from library)

The minimum period for any clock pin is calculated in the same way as the minimum pulse width. The minimum period is calculated as:

Min Period = MAX(Arrival time of Launch clock Path - Arrival time of Capture Clock Path + Min Period at clock pins from library)

These checks can be modeled in the following three ways:

- Library attribute
- Scalar tables
- Arc-based modeling

These are explained below.

Library Attribute

By default, these checks are written as library attributes in ETM.

```
pin (CLKIN2 ) {
   clock : true ;
   min_period : 2.0554;
   min_period : 2.6248;
   min_pulse_width_low : 0.2773;
   min_pulse_width_high : 0.7673;

}
```

If `min_period` checks corresponding to both rise and fall transitions are present in a design, then they wiil be modeled as duplicate entries in ETM when written as library attributes. To avoid duplicate entries we can model these checks as scalar tables or arcs, where rise and fall transition of min_period checks can be modeled as shown in following sections. These arcs will be honored by the timer depending on the context, hence will be more accurate.

Scalar Tables

If the `timing_extract_model_write_clock_checks_as_scalar_tables` global variable is set to `true`, then these checks will be written as scalar tables as shown below:

```
pin (CLKIN2 ) {
   clock : true ;
   timing() {
      timing_type : minimum_period ;
      rise_constraint (scalar){
         values( " 2.0554");
         }
      fall_constraint (scalar){
         values( " 2.6248");
         }
      related_pin :" CLKIN2 ";
  }
   timing() {
      timing_type : min_pulse_width ;
      rise_constraint (scalar){
         values( " 0.7673");
         }
      fall_constraint (scalar){
         values( " 0.2773");
         }
      related_pin :" CLKIN2 ";
  }
}
```

Arc-Based Modeling

These checks can also be written as arcs by setting the `timing_extract_model_write_clock_checks_as_arc` global variable to `true`. In this case the resulting arc is the function of rise/fall slew of CLK port, hence delay and slew propagation of the clock network for rise and fall transitions is considered for writing these arcs. The arc-based modeling of clock-style checks is more accurate and is recommended for use.

```
pin (CLKIN2 ) {
    clock : true ;
    timing() {
        timing_type : minimum_period ;
        rise_constraint (lut_timing_2 ){
           values(" 2.000, 2.000, 2.000, 2.000, 2.000, 2.000, 2.000");
           }
        fall_constraint (lut_timing_2 ){
           values(" 2.000, 2.000, 2.000, 2.000, 2.000, 2.000, 2.000");
           }
        related_pin :" CLKIN2 ";
    }
    timing() {
        timing_type : min_pulse_width ;
        rise_constraint (lut_timing_2 ){
            values("0.767, 0.767, 0.767, 0.767, 0.742, 0.087, 0.100");
            }
        fall_constraint (lut_timing_2 ){
            values("0.201, 0.201, 0.201, 0.201, 0.238, 0.247, 0.2636");
            }
        related_pin :" CLKIN2 ";
    }

}
```

Path Exceptions

Timing extraction flow honors path exceptions defined in the given constraints' file. Typically, the following path exceptions are used:

- set_false_path
- set_multicycle_path
- set_min_delay
- set_max_delay

For more information on handling of these exceptions, refer to section “Constraint Generation during Model Extraction”.

Constants

The case analysis and the constants are propagated while extracting the model. In case of `set_case_analysis` command, we can control how constants propagated at o/p or bi-di ports are modeled in ETM through the following global variable.

```
set timing_extract_model_case_analysis_in_library true (or false)
```

When set to `false`, this global variable specifies the propagated or applied constants, using the `set_case_analysis` command to output ports in the generated constraints file (generated using `-assertion` parameter in the `do_extract_model` command).

When set to `true`, this global variable specifies that propagated constants to output ports are written as pin functions in the extracted model.

Unconstrained Paths

The unconstrained end points exist when proper launch/capture clock phase is not propagated at the desired endpoint due to any exceptions. The unconstrained input ports due to false path exceptions are ignored and are not modeled during ETM extraction. The unconstrained combinational IO paths due to false path exceptions are ignored and are not modeled during ETM extraction. The unconstrained trigger arcs are calculated during ETM extraction.

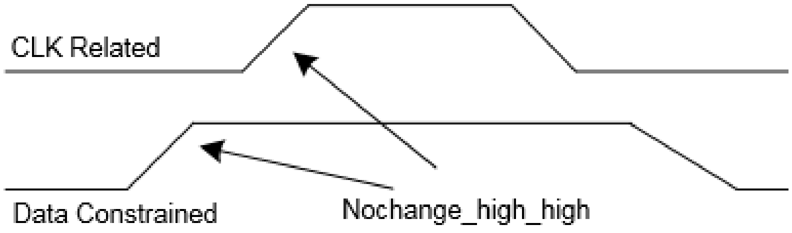
Clock Gating Checks

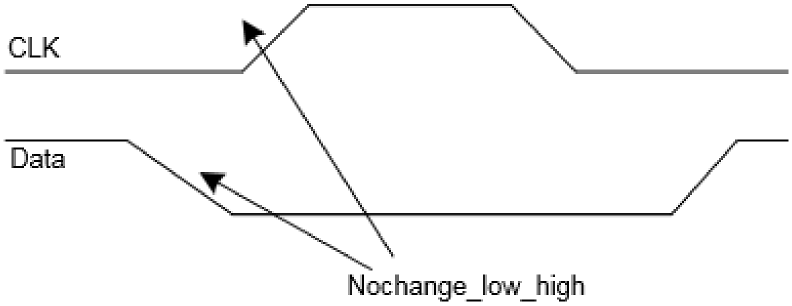
If integrated clock gating (ICG) cells are used for gating the clocks, then these arcs are preserved as setup/hold arcs. If combinational cells are used for gating, then these arcs will be preserved as no-change arcs between a clock pin and its enabling signal pin. If you wish to extract these checks as regular setup/hold checks, you can set the `timing_extract_model_gating_as_nochange_arc` global variable to `false`.

Depending on the type of clock gating situation, no change checks are inferred as follows:

The following waveform shows how no_change checks are modeled:

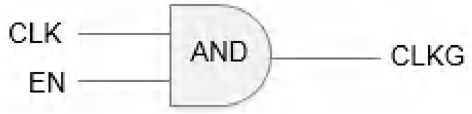
no_change Checks Modeling





Simple Clock Gating with AND or Logic

The following diagram shows clock gating AND/OR logic:



A simple AND gate check will be as follows:

```
timing () {
timing_type: nochange_high_high;
}
timing () {
timing_type: nochange_low_high;
}
```

Note: After extracting the clock gating check arc the signal downstream the gate output is not propagated to the clock pins of the registers.

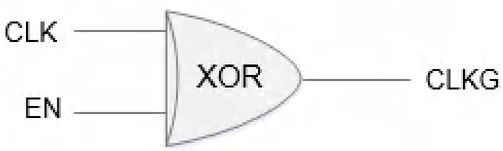
Clock Gating with Blackbox or Unknown Logic

In case of clock gating with unknown logic, as shown below, no_change checks will be checked with respect to both the rising edge and the falling edge of the clock signal.

CLK
EN
Black Box
CLKG

No Clock Gating Logic

There are some gates, such as multiplexers or XOR gates, where a clock signal cannot control the clock gate, as shown below.



In such cases, no checks are inferred so you need to explicitly set the gating check if you want these interface paths to be extracted in the ETM. To know where static timing analyzer will not infer the gating in such situations, you can use the following command:

```
check_timing -check_only clock_gating_controlling_edge_unknown -verbose
```

Annotate Delays, Load, and Slews

Model extraction uses the back annotated delay slews and the load information during the extraction process and reflect the effect in the extracted model. Here is a small description how these are used.

Annotated Delays
The annotated delays using SDF or the `set_annotated_delay` command override the calculated delay (from library or RC data) value for the arc; however, the output slew for the arcs is used as calculated. In case of incremental delays the delta delay is added to the calculated arc delay.

Annotated Slews
Annotated slews are ignored during timing model extraction. These constraints are dumped in assertions file generated with the `do_extract_model -assertions` parameter.

Annotated Load
Annotated load will override the pin capacitance defined in the library, and will be used for the C-eff and hence the delay calculation.

Note: The annotated delays are generated with a particular context and remain true for that context (transition times) only. So annotating the delays/slews makes the model context dependent. So to create a context independent model it is recommended not to annotate the SDF.

Design Rules

Model extraction uses the design rules defined in the library. The worst (smaller for max design rules and vice-versa) among all the fanout pins (topologically first level) is used for the input ports and the worst of all the fanin pins (topologically first level) is used for the output ports.

If design rule limits are defined at the pin and library level, the design rule from the pin is chosen, rather than the worst of the pin and library level, because the pin level information overrides the cell level, which in turn overrides the library level information. If design rule violations (DRV) are coming from the constraints, then you can choose them based on the following setting:

```
set timing_extract_model_consider_design_level_drv {true/false}
```

When set to `false`, the user-asserted design level DRVs are not considered while modeling DRVs in the extracted timing model.

When set to `true`, the user-asserted design level DRVs are considered while modeling DRVs into the extracted timing model.

Clocks

Created Clocks

If the `create_clock` assertion is applied on the pin of a design (not port), then the pin is modeled as an internal pin with the same name as that of the clock asserted on it. If a clock is created on a design port, then this port will be preserved as ETM I/O pin, with the same name as that of the port itself.

```
create_clock [get_ports {CLKIN}] -name clk -period 3 -waveform {0 1.5}
create_clock [get_pins buf5/Y] -name clk3 -period 3 -waveform {0 1.5}
```

```
    pin (CLKIN ) {
       clock : true ;
       direction : input ;
       capacitance : 0.0042;
   }
   pin (clk3 ) {
       clock : true ;
       direction : internal ;
       capacitance : 0.0110;
   }
```

Generated Clocks

The generated clocks that defined in a hierarchical block, using the `create_generated_clock` command, are supposed to be a part of the block itself and do not come from the top level. So ETM preserves generated clocks inside the model as internal pins using the Liberty `generated_clock` construct. The name of any such internal pin is the same as that of the generated clock.

Some of the examples are described below.

Example 1: Single generated clock on a single pin

```
create_generated_clock -name gclk -source [get_ports clk] -divide_by 2 [get_pins
buf1/A]
```

During the extraction process the pin “buf1/A” will be preserved as an internal pin in the library with name `gclk`. The model will show as follows:

```
generated_clock (gclk) {
master_pin : clk;
divided_by : 2;
clock_pin : “gclk “;
}
pin (gclk ) {
clock: true ;
direction: internal ;
.
.
}
```

Example 2: Multiple clocks on the same pin

There are some design scenarios when multiple generated clocks are defined on a single pin. This asserts multiple clock definitions on the pins. In this case, during model extraction the pin is duplicated with the name of the clock. For example, in a design if there are two clock definitions, such as:

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK -
divide_by 2 [get_pins buf1/A]
create_generated_clock -name gclk2 -source [get_ports clk] -add -master_clock CLK -
divide_by 2 [get_pins buf1/A]
```

During the extraction process the pin buf1/A will be preserved as an internal pin with names gclk1 and gclk2. The model will be as follows:

```
generated_clock (gclk1) {
master_pin : clk;
divided_by : 2;
clock_pin : “gclk1 “;
}
generated_clock (gclk2) {
master_pin : clk;
divided_by : 2;
clock_pin : “gclk2 “;
}
pin (gclk1) {
clock: true;
direction: internal;
…..
}
pin (gclk2) {
clock: true;
direction: internal;
…..
}
```

Example 3: Generated clocks on multiple pins

When a generated clock is defined on multiple pins, the software asserts a single clock definition on multiple pins. To handle this scenario, all the pins asserted with this clock are preserved as internal pins in the model with the name `[clock_name_<count>]` and the `generated_clock` construct is written in the model with all the pin names in the `clock_pin` attributes with a space between them. For example, consider a netlist having clock definitions on multiple pins.

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK -
divide_by 2 [get_pins {buf1/A buf2/A}]
```

During the extraction process, the pin buf1/A will be preserved as an internal pin in the library with the name gclk1_1; and the pin buf2/A will be preserved as an internal pin with name gclk1. The model will be as follows:

```
generated_clock (gclk1) {
master_pin : clk;
divided_by : 2;
clock_pin : "gclk1_1 gclk1 ";
}
pin (gclk1_1) {
clock: true;
direction: internal;
…
}
pin (gclk1) {
clock: true;
direction: internal;
…
}
```

In this case when ETM is read, two generated clocks gclk1_1 and gclk1 will be created on two internal pins - gclk1_1 and gclk1, respectively. Any constraint coming from the top level will match only with clock gclk1, and not gclk1_1. To avoid such a situation, set `timing_library_genclk_use_group_name` global variable to `true`. When this global variable is enabled, the original clock (gclk1) is generated at two internal pins - gclk1_1 and gclk1.

When `timing_library_genclk_use_group_name true` is set, then the `report_clocks` command output shows the following output:

```
-------------------------------------------------------------------------
                        Clock Descriptions
-------------------------------------------------------------------------
Clock Name  Source                     Period  Lead    Trail   Gen  Prop
-------------------------------------------------------------------------
clk         clkin                      3.600   0.000   1.800   n    y
gclk1       BLOCK/gclk1_1 BLOCK/gclk1  7.200   0.000   3.600   y    y
-------------------------------------------------------------------------
On turning off this global, report_clocks will return:
-------------------------------------------------------------------------
                        Clock Descriptions
----------------------------------------------------------------
Clock Name   Source         Period   Lead    Trail   Gen Prop
----------------------------------------------------------------
clk          clkin          3.600    0.000   1.800   n   y
gclk1        BLOCK/gclk1    7.200    0.000   3.600   y   y
gclk1_1      BLOCK/gclk1_1  7.200    0.000   3.600   y   y
-----------------------------------------------------------
```

Example 4: generated clock in case of multi ETM instantiation at top

When the `timing_prefix_module_name_with_library_genclk` global variable is set to `true`, the software appends the instance name to the clock pin name when creating a generated clock. When set to `false`, the software uses only the clock pin name when creating a generated clock. For example, assume that the cell for instance a/b in the timing library contains the following generated clock group:

```
generated_clock ( genclk1 ) {
      divided_by    :  2 ;
      clock_pin     :  " genclk1 ";
      master_pin    :  CLKIN ;
}
```

By default (`true`), the software creates a generated clock with the name - genclk1. If you set `timing_prefix_module_name_with_library_genclk` to `false`, the software creates a generated clock with the name - etm/genclk1.

If ETM is instantiated at the top level as BLOCK1 and BLOCK2, then by default the `report_clocks` command will show the following output:

```
-----------------------------------------------------------
                      Clock Descriptions
-----------------------------------------------------------------------------
```

```
Clock Name     Source          Period  Lead   Trail   Generated   Propagated
-----------------------------------------------------------------------------
BLOCK1/genclk1 BLOCK1/genclk1  7.200   0.000  3.600   y           y
BLOCK2/genclk1 BLOCK2/genclk1  7.200   0.000  3.600   y           y
-----------------------------------------------------------------------------
```

If you do not want to differentiate between genclk1 generated in various ETM instantiations, you can set `timing_prefix_module_name_with_library_genclk` to `false`. In such a case, the `report_clocks` command will show the following output:

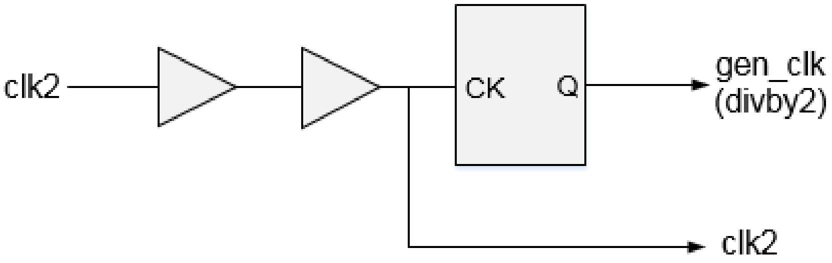
```
------------------------------------------------------------------------
                     Clock Descriptions
------------------------------------------------------------------------
Clock Name   Source          Period   Lead   Trail  Generated Propagated
------------------------------------------------------------------------
genclk1      BLOCK2/genclk1  7.200    0.000  3.600  y         y
------------------------------------------------------------------------
```

Note: You will need to ensure that the constraints are applied with respect to the generated clock naming. If required, you can remove the library generated clocks and then apply the specified generated clocks so that minimal changes are required in the constraints.

Clocks with Sequential and Combinational Arcs

Consider a case where both the sequential and combinational arcs exist between a clock (CLK) and an output (OUT) pin. If any of the early/late sequential/combination arcs is missing (due to a path exception) between the CLK and OUT pins, the CLK pin is duplicated as two pins with _SEQ_pin and _COMB_pin suffixes. All the combinational arcs starting from this clock root to the duplicated pin are bound with the “_COMB_pin” suffix and all the sequential arcs are bound to be duplicated pin with the “SEQ_pin” suffix.

Sequential and combinational arcs between a clock



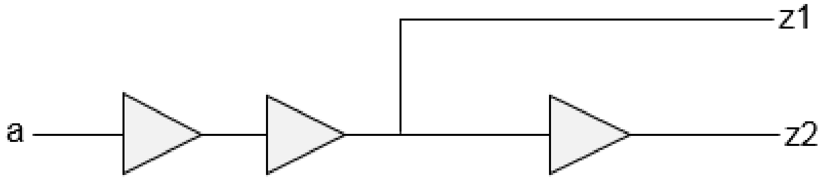
The following will be created in ETM:

```
pin (CLKIN_SEQ_pin ) {
    clock : true ;
    direction : internal ;
    capacitance : 0.0110;
    }
pin (CLKIN_COMB_pin ) {
    clock : true ;
    direction : internal ;
    capacitance : 0.0110;
    }
pin (DATAOUT ) {
    direction : output ;
    timing() {
        timing_type : combinational ;
        fall_transition (lut_timing_3 ){ … }
        related_pin :" CLKIN_COMB_pin ";
    }
    timing() {
        timing_type : rising_edge ;
        fall_transition (lut_timing_3 ){ … }
        related_pin :" CLKIN_SEQ_pin ";
    }
```

Secondary Load Dependent Networks

If there is a timing path from one output pin to another pin, the model extractor considers both the primary output loading and the secondary output loading when output-to-output delays are computed. In the following figure there is an output-to-output path from z1 to z2:

Output-to-output path from z1 to z2



The extracted timing model for this block consists of two delay arcs - one from a -> z1 and another from a -> z2. But the delay of arc a -> z2 also depends on the secondary loading, that is, at port z1.

This gives rise to a delay arc from `a` to z2 which is dependent on two loads and one input slew. So this arc will be dumped as a three-dimensional delay table because the delay depends on the primary output loading at z2, as well as a secondary output loading at z1.

The table template will be as follows:

```
lu_table_template (lut_timing_1) {
variable_1: input_net_transition;
index_1 ("0...0 1.0 2.0 3.0 4.0 5.0");
variable_2: total_output_net_capacitance;
index_2 ("0...0 1.0 2.0 3.0 4.0 5.0");
variable_3: related_out_total_output_net_capacitance;
index_3 ("0...0 1.0 2.0 3.0 4.0 5.0");
}
```

By default, 3D arc modeling is disabled - the software considers 3D arcs as 2D arcs in timing modeling.

Note: You should always buffer the port to avoid 3D dependencies. If there is more than a couple of load dependencies, then there will be accuracy loss since model extraction cannot accurately model beyond 3D arcs accurately.

Characterization Point Selection

Selecting characterization during point extracted models depends on the following selection criteria:

If no input_slews/clock_slews/output_load is specified as an argument in the `do_extract_model` command, then the software determines the characterization points from the design's interface elements - that are supplied with the external slew/load. This can be explained as follows.

All the check arcs (e.g., clk->in) will be characterized for the slew points as follows:

- Reference slew points will be taken from the slew index of the first element just after the "clk" port.
- Signal slew points will be taken from the slew index of the first element just after the "in" port.

All the sequential (e.g., clk->out) arcs will be characterized as follows:

- Input slew will be taken from the slew index of the first element just after the *clk* port.

All the combinational arcs (e.g., in->out) will be characterized as follows:

- Input slew will be taken from the slew index of the first element just after the *in* port.
- Output load will be taken from the load index of the last element just before the *out* port.

Consider the following topology in a netlist:

in --------> buf1-- -----> ff1/d -------->buf2 --------> out

clk ------> clk_buf ---- ->ff1/clk-------->buf2 --------> out

in ------> buf3 ------> buf4 --------> buf5 --------> out

A snapshot of the original timing library for the interface elements (i.e., BUF and CLK_BUF) is as follows:

```
Cell (BUF)
{
timing ()
index_1 ("0.0500, 1.4000, 4.5000");
index_2 ("1.0500, 6.5000, 10.0000");
}
Cell (CLK_BUF)
{
timing ()
index_1 ("1.0500, 2.4000, 3.5000");
index_2 ("0.0500, 4.5000, 5.0000");
}
```

If the `do_extract_model` command is used, then the extracted model will be characterized as follows:

```
Check Arc clk->in
index_1 ("0 0.0500, 1.4000, 4.5000");
```

Here the signal slew is taken from buf1 slew-index in original libraries.

```
index_1 ("0 1.0500, 2.4000, 3.5000");
```

Here the reference slew is taken from clk_buf slew-index in the original libraries.

```
Seq. Arc clk->out
index_1 ("0 1.0500, 2.4000, 3.5000");
```

The input slew is taken from clk_buf slew-index in the original libraries.

```
index_2 ("0 1.0500, 6.5000, 10.0000");
```

Output load is taken from buf2 slew-index in the original libraries.

Combinational arc in->out

```
index_1 ("0 1.0500, 1.4000, 4.5000");
```

The input slew is taken from buf3 slew-index in the original libraries.

```
index_2 ("0 1.0500, 6.5000, 10.0000");
```

Output load is taken from buf5 slew-index in the original libraries.

Note: A value of 0 is always added to the characterization points.

Constraint Generation during Model Extraction

The extracted models need to be validated before they are transferred to other designers to be used for a top-level analysis or optimization flow. For validation purposes, we need a subset of the original timing constraints and further a subset of these constraints is needed for fitting the model in a top level netlist.

The model extraction process will output a timing library and two constraint files containing the subset of the original constraint. One of these constraint files will be used for a standalone validation of the model compared to the original netlist. By default model.asrt and model.asrt.latchInferredMCP will be written in the CWD. If the `do_extract_model -assertions test.asrt` command is specified, then three assertion files are generated - test.asrt, test.asrt.latchInferredMCP, and top_model.asrt. The other constraint file (top_model.asrt) will be used when the extracted model will be stitched to a top level netlist and test.asrt will be used for standalone model validation.

Two separate constraint files are required for:

- Standalone validation will need all the context parameters of the design.
- STA or optimization flow when stitched to a top level environment, the context parameter will be automatically coming from the top level. So some of the constraints (in the above file) need to be filtered.

This is what is achieved with an automated procedure executed with every model extraction. The following are the handling of these constraints in a model extraction flow:

False Path Exceptions

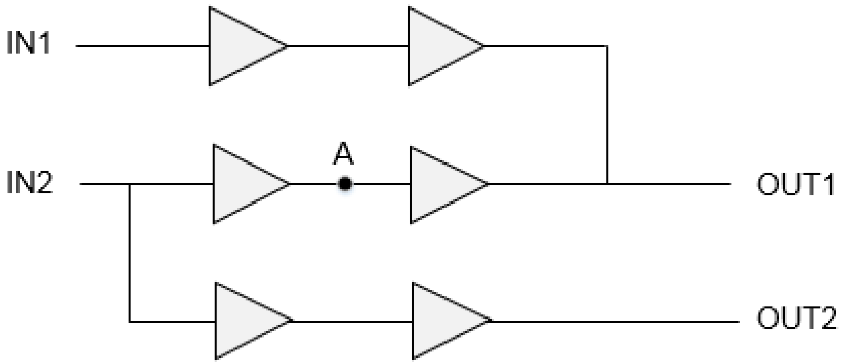
- If the `set_false_path` constraint is applied through some internal pins/ports, then those paths/arcs are removed during extraction.
- If the `set_false_path` constraint is applied between clocks or a clock and a port, then these paths are removed and these exceptions are saved in a top_model.asrt as well as test.asrt file.

Multi-Cycle Path Exceptions

If multi-cycle paths through pins/ports are applied, then during extraction the worst paths through them mapped to the IO ports are calculated and the exceptions through these IOs are dumped in the top_model.asrt file.

Consider the following design scenario with multi-cycle path exception applied at A.

Multi-cycle path exception



If there is no path from IN1/OUT1 and IN2/OUT2, the multi-cycle path will be pushed to IN2/OUT1. Since there are other paths going through these ports, multi-cycle paths cannot be pushed to IN2/OUT1 and cycle adjustment is done in delay values of the arcs. This will make the model context dependent.

When you set `timing_extract_model_disable_cycle_adjustment` global variable to `true`, the software will disable the cycle adjustment to make ETM context independent. You will have to manually apply the multi-cycle path at the top level while using this ETM.

Note: You should examine the constraint file (”top_model.asrt”), as these constraints are just indicative of what is needed to be modeled at the top level. In certain situations, it may not be possible to extract all possible exceptions for top level due to limitations of liberty to model them correctly for a given path in case of conflicts.

set_disable_timing

If the `set_disable_timing` constraint is applied on any arc, then this arc gets disabled and is not used for the extraction purpose. Thus, such arcs are handled internally by the software and are not written to any of the constraint files. Also, the path broken by them will not be extracted during ETM generation.

set_case_analysis

Constants applied/propagated by the `set_case_analysis` command can be written as a function attribute of a pin in ETM by setting the `timing_extract_model_case_analysis_in_library` global variable to `true`. This can be written in the ETM-validation constraint file generated by setting this global variable to `false`.

create_clock

The pins/ports having `create_clock` definitions on them are preserved in the timing model. If a clock is created on some internal pin then the pin name needs to be modified to reflect the instance name of the extracted model. This is achieved using the same variable approach.

```
[get_pins “$ETM_CORE/pin1 $ETM_CORE/pin2 $ETM_CORE/pin3”]
```

When the extracted model is stitched to some top level netlist, then the clock definitions are supposed to come from the top level netlist itself. So that such clock definitions are not needed to be dumped in the top level constraint file, but need to exist in the assertion file for standalone validation.

create_generated_clock

The pins having the `create_generated_clock` defined on them are preserved in the extracted model as internal pins. As generated clocks are very much intended for the block itself and are not supposed to be coming from the top level, liberty provides syntax to define the generated clocks in the timing model. You can use the following use model:

```
generated_clock (gclk) {
clock_pin : gclk ;
master_pin : clk ;
multiplied by : 2 ;
}
```

While loading a model, the software names the generated clocks after their target pin names, so while dumping the generated clocks in the library the target pin name is modified to the clock name itself. This facilitates the same name of generated clock names in the original netlist and the extracted model. This name changing cuts down the need to modify other constraints (clock uncertainty/path exceptions) related to this clock. In this case it is not required to dump these generated clocks as a separate constraint.

set_input_delay/set_output_delay

The constraints like `set_input_delay` and `set_output_delay` lie in this category. Though these constraints can be applied on ports as well as some internal pins. Such constraints applied on some internal pins are of no use for the extracted models, as in an extracted model only interface timing is considered.

The constraints applied on the ports need not any change as the port will be preserved with the same name. So these constraints are dumped out in the assertion file for validation and do not have any impact on the generated ETM. These need not to be dumped out for the top level constraint file as in a top level the data must be coming to the port from some other top level register or port. So the input delay value is supposed to be the delay of path from top level register/port to this port.

set_max_transition/set_max_capacitance

If these constraints exist in the SDC file for a block and the `timing_extract_model_consider_design_level_drv` global variable is set to `false`, then these max_cap/max_trans constraints (from SDC) will not have any impact on the generated model.

If the `timing_extract_model_consider_design_level_drv` global variable is set to `true`, then these constrains are considered while generating the timing model. A conservative value of max_transition defined at the library cell associated with a port, or defined `set_max_transition` setting in the SDC is chosen for all the ports. Similarly, in case of max_capacitance a conservative value defined at a library cell associated with a port, or the `set_max_capacitance` setting defined in the SDC is chosen for all output ports.

Note: The `set_max_transition or set_min_transition` constraint defined in the SDC is always dumped in the constraints file that is read during validation.

set_load/set_resistance/set_annotated_transition
The `set_load`, `set_resistance`, or `set_annotated_transition` constraints can be applied on pins as well as ports. Such constraints applied on internal pins are handled in the extraction flow during delay calculation. So these are included in the model itself. But these constraints applied on the port are treated as the out context environment and need to be dumped out in the constraints file to be read for model validation.

set_annotated_delay/set_annotated_check
The `set_annotated_delay`, or `set_annotated_check` constraint is applied on the internal arcs as incremental or absolute values. The annotated delays/checks are handled in the extraction flow during delay calculation, so these are included in the model. This constraint is not written out in any of the constraint files.

set_input_transition/set_driving_cell
The `set_input_transition`, or `set_driving_cell` constraints are applied only on ports. At the top level , the input transition at ETM pins depends upon the transition of the signal coming from the top-level circuitry. Similarly, the driving cell is needed only at the block level to replicate the actual driver of ETM at the top level. Hence, these are written only in the validation constraint file.

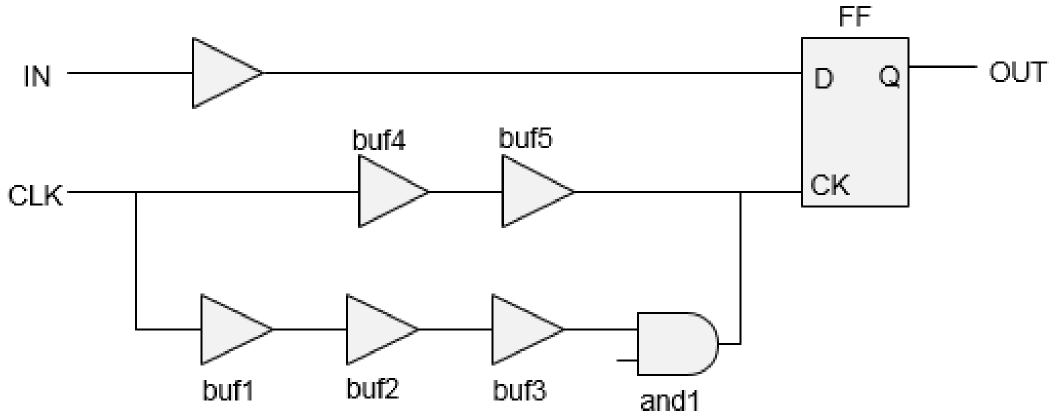
clock_uncertainty/latency
By default, the clock uncertainty or clock latency constraints are written in the validation constraint file. But if `timing_extract_model_include_latency_and_uncertainty` is set to `true`, then these values will be considered during ETM characterization. And the delay tables in ETM are written with these values included.

Parallel Arcs in ETM

If a mode is extracted in the BC-WC or OCV mode, then timing is included through the early and late path analysis (as is done with the `report_timing` command).

Consider the following design:

Parallel arcs in ETM



In the above design there are two paths CLK->IN (check path) and CLK->OUT (sequential path).

Here, the check (setup) path will be captured by the early path of clock (CLK->buf4->buf5->FF/CK). But the sequential path will be using the late clock path (CLK->buf1->buf2->buf3->and1->FF/CK). The extraction of early/late attributes is differentiated through the min_delay_arc attribute. So, in the extracted model for combinational/trigger/latency arc, both the rise/fall maximum and minimum arcs exist.

Latency Arcs Modelling

The latency arcs exist between a generated clock and the respective master clock. The actual paths are considered during extraction. For example, consider a generated clock with `-divide_by 2`. The edge relationship of master and generated clock will be “R ->R” and “R -> F”. The paths “F->F” and “R->F” arc will not be dumped, even if such paths are present in the design for data propagation.

In case of an ideal master clock, the arcs between master and generated clock source, is controlled through the following setting:

```
set timing_extract_model_ideal_clock_latency_arc {false/true}
```

When set to `true`, the zero delay arc from the master clock to generated clock is considered in the extracted model. When set to `false`, this arc is not modeled.

Latch-Based Model Extraction

Extraction model handles transparent latches during model extraction based on the arrival times defined on input ports and the specified clock periods - whether a path from an input port to a latch causes borrowing or not. The following points should be noted:

- If the path causes borrowing then path traversal should continue through the latch until either a non-borrowing latch or edge-triggered flip-flop is encountered. If the path does not cause borrowing, then path tracing should be stopped and a setup arc should be extracted relative to the closing edge of the first (interface) latch.

- The borrowing behavior of latches does not impact the extracted hold arc. So hold arcs should always be extracted relative to the close edge of the first interface register (latch or flip-flop) that is encountered while tracing paths from input ports.
- Borrowing can result in paths being traced through latches from an input port to an output port. In this case, a delay (comb) arc should be extracted from the input port to output port. In scenarios where path becomes more than 1 cycle, inferred multi-cycle paths are dumped in a separate file named as “model.asrt. latchInferredMCP”.
- When tracing paths from interface registers to output ports, transparent latches will be handled by tracing through borrowing latches backwards and generating a sequential delay arc from a clock port to an output port.

Note: As all such latch traversal arcs are true for setup analysis, there will be a `set_false_path -hold` statement in the assertion file for any such arcs. Please also note that “model.asrt. latchInferredMCP” file is indicative of what assertions you may need to use at the top level. They should be audited before using multi-cylce paths, as they cannot be inferred in all the cases.

Model Extraction in AOCV Mode

Innovus supports modeling of AOCV derated delays and stage weights in ETM. These stage weights and AOCV derates are used during the top level analysis with ETM by correctly computing the stage counts and derates of the paths related to ETM. The AOCV analysis mode can be enabled by using the following setting:

```
setAnalysisMode -aocv true
```

Stage Weight Modeling in ETM

The stage weight modeling feature can be enabled by using the following parameter:

```
do_extract_model -include_aocv_weights
```

When specified, this enables the software to write stage weights on individual arcs.

The weights written in the ETM are not meant to derate the model. The model will be containing the derated delays. The stage weight extracted on arcs will be used to calculate the stage count of chip level paths going across the ETM.

The extraction of stage weight will be done by tracing the minimum stage count path on arc-by-arc basis. A path-based cell stage count will be printed on all sequential and combinational arcs. The extraction strategy for computing stage weight will be as follows:

The stage weights in the extracted model are dumped as user-defined attributes. This will require defining three attributes at the arc level.

```
define ("aocv_weight","timing","float");
define ("clock_aocv_weight","timing","float");
The first attribute will be used for combinational and trigger arc and will be dumped
at arc level
as below.
timing () {
timing_type : combinational;
timing_sense : positive_unate;
aocv_weight : 4;
.
.
}
```

The check arcs will be dumped with two separate stage weights for data and clock:

```
timing () {
timing_type : setup_rising;
aocv_weight : 4;
clock_aocv_weight : 3;
.
.
}
```

AOCV Derating Mode

The AOCV derating mode can be specified to path-based, graph-based, or none by using the following setting:

```
set timing_extract_model_aocv_mode option
```

You can specify one of the following options:

- `graph_based`: Delays in ETM are derated using the graph-based stage counts.
- `path_based`: Delay in ETM are derated using total path stage count of worst path between those pin pairs.
- `none`: Models derated delays that contain the effect of user-derate but does not contain effect

of AOCV derates.

The default is `none`.

Before setting this global variable to `graph_based or path_based`, it is mandatory that the AOCV libraries are read in and AOCV analysis is enabled (by using `setAnalysisMode -aocv true`).

Merging Model with stage_weight Attribute

The merged timing model contains minimum stage_weight defined between various input library files, hence merged timing model is pessimistic.

Points to be Considered for Block Level AOCV Run

The following points need to be taken care of for the block level AOCV run:

1. If analysis of a block is performed in the standalone mode, then you need to specify the stage weight seen at the port(s). The applied stage weight affects the internal weight count of the block for the IO paths. If these weights are not applied, the pessimistic analysis will be done for the interfacing paths of the block (when block is top). Hence the ETMs will also have pessimistic numbers for such paths.
2. The same ETM may not be used for multiple instantiation at the top level, due to the following reasons:
   - Different stage counts at the ports (as seen from the top) is possible.
   - Different critical paths between the ports for different instances is possible.

   You can use ETM with most conservative stage count for all the instances, but this will lead to pessimistic analysis.

   3. The ETMs are written taking AOCV derates into account. At the top level, you need to have two types of AOCV derate tables:
   - Cell level AOCV tables
   - Design level AOCV tables for net

If design level cell AOCV derates are used at the top level, then the block delays (as seen from the top) will have the AOCV effect twice - once during model extraction at the block level and second at the top level from design level cell AOCV tables. In case design level AOCV tables are being passed with ETM at the top level, then you should provide a cell level table for ETM with derating as 1 for all the possible stage counts and/or distance.

PG Pin Modeling During Extraction

Power-ground pins are defined as current source or sink pins, respectively. Information of PG pins can be fed to the software by CPF/UPF files.

Information regarding power/ground pins can be written in the ETM. To enable writing PG pin information in a timing model, you can use the `do_extract_model -pg` option. The various low power attributes are written at the library level, cell level, and pin level in ETM.

Pre-requisites of PG-based flow are:

- Input CPF/UPF and libraries in the flow for power/ground aware ETM generation.
- Signal Integrity (SI) analysis uses some commands and may fall back to CDB (whenever the relevant power rail/voltage information is missing in the CPF/dotlib) for getting supply information. The CPF or input dotlib will have sufficient information to correctly model ETM.

PG Modeling in ETM

Modeling of voltage maps

Voltage maps definitions are typically modeled inside the Liberty .lib format at library scope level. It specifies the voltage value associated with the power rail name (voltage names). At cell level pg_pin groups are associated with voltage values using the voltage names.

```
voltage_map (VDD, 0.8);
voltage_map (VSS, 0.0);
```

Modeling of power/ground pins

At the cell level, the pg_pin groups will be modeled in the ETM in the following way:

```
pg_pin (VDD1) {
voltage_name : VDD ;
pg_type : primary_power;
}
pg_pin (VSS1) {
voltage_name : VSS ;
pg_type : primary_ground ;
}
```

The supported pg_type in model extractions: primary_power, primary_ground, internal_power, internal_ground. If a power rail is the output of a power switch cell, it will be marked as internal_power/ground.

Associating power and ground pins to signal pin

This information will be written for all the logic input/output/in-out pins that are written in the ETM.

For internal pins that are preserved inside the ETM dotlib (e.g., a pin on which the generated clock was asserted), no such information will be retained.

```
pin (A) {
…
related_power_pin: VDD1;
related_ground_pin: VSS1;
}
```

For ports, that are written as pins in ETM, the associated pin (driver or receiver of a net) will be checked and printed in the same way as related_power_pin and related_ground_pin information.

ETM Merging Requirements for Power/Ground Aware ETMs

- Voltage maps from two libraries will be merged successfully as long as the voltage rail names are distinct.
- pg_pins will be merged successfully as long as power rail names and the pg_type match.
- related_power_pin and related_ground_pin will be merged as long as they are available in all the libraries and refer to the same power rail name.

Extracted Timing Models with Noise (SI) Effect

To generate a timing model with SI effects contained within a block, you can use the following flow for a block under consideration:

- Load the database and the relevant information for performing STA/SI analysis.
- Perform SI analysis, or if you have an incremental SDF from the previous SI run just load the incremental SDF.
- Run model extraction. During this phase, the incremental delays will be added to the computed base delays for characterization of the dotlib table. When the final dotlib is written, it contains the effect of the SI analysis in terms of characterized delays.

The block ETM can then be instantiated in the top level flow and the required file can be loaded for performing top level analysis with ETM.

Merging Timing Models

The software supports merging library models through the `merge_model_timing` command. You can generate ETM in various modes and then merge them in a single library (in which all the modes are defined inside a merge_group) that can be used for timing optimization during design flow. These mode_group/modes are defined in merged ETM as mode_definition/mode_value.

To merge two ETMs (corresponding to funct and scan views) use the following:

```
merge_model_timing -library_file funct_etm.lib scan_etm.lib -modes funct scan -
mode_group funct_scan -outfile merged_funct_scan.lib
```

These modes are defined in merged ETM as:

```
mode_definition (funct_scan){
mode_value (funct){
}
mode_value (scan){
}
```

In a merged ETM, arcs corresponding to different modes are defined as the following:

```
timing() {
mode( funct_scan," funct " );
min_delay_arc : "true" ;
related_pin :" SI_ClkIn ";
}

timing() {
mode( funct_scan ," scan " );
related_pin :" EJ_TCK ";
}
```

To read arcs corresponding to funct (scan) mode, use `set_mode` funct (scan). If the set_mode command is not issued, the arcs corresponding to both the modes will be reported by `report_timing`. Single mode can be specified with the set_mode command. An error is issued if more than one mode is passed as an argument to this command. The mode merging enables you to do the analysis in various modes on a single shell.

Some considerations while merging models are given below:

- All the input timing models should have the same cell name.
- If the timing arcs are not comparable i.e., the number of index values differ, then they are written as separate timing arcs with separate mode definitions.
- Boundary pins of single mode libraries should be equivalent in number and name while

merging. An error message will be flagged if that is not the case.

- If you have generated clocks with the same name, as well as same definition in ETM libraries, then just this clock is preserved in the merged library, without issuing an error message.
- If you have generated a clock with the same name, but different divided_by/multiply_by/master_pin/clock_pin definition, then an error message is displayed stating that “genclk is defined with different specification in two modes, merging is not possible”.
- If there is a mismatch in timing arcs among the input libraries, then the merged library will contain the union of all the timing arcs with the respective mode. For example, if the first library has only setup arcs and the second library has only hold arcs then the merged library will have both setup and hold arcs with respective modes appended.
- If there are mismatches in internal pins among input libraries, that is, if a particular internal pin is missing in one or more libraries then that internal pin and its timing arcs will be part of a merged library with respective mode.
- For both maximum and minimum DRV, you can use the most conservative value. The minimum value from maximum DRVs will be used and the maximum value from minimum DRVs will be used in the merging.
- When same modes are specified in the modes list for two libraries e.g., `merge_model_timing -library_files "setup.lib hold.lib" - modes "M M" -mode_group “MG”`, then the arcs from the two libraries will be assigned the same mode i.e., “M” while merging.

Limitations of ETM

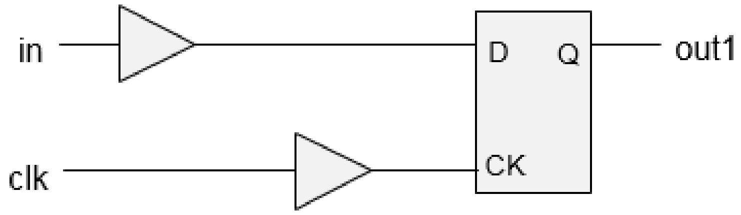
The limitations of ETM are as follows:

1. In path-slew propagation mode, the slew/delay dependency on side inputs will be lost.

2. Consider the following two cases:

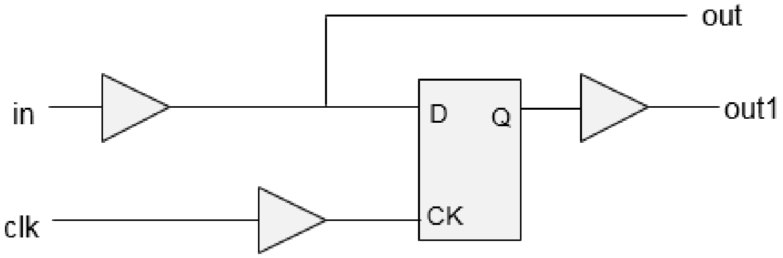
(a) clk-in check ac will depend on the slew at the D and CK pins of the register. Hence these check arcs are modeled as 2D-check arcs, as expected.

Check arc



(b) In the second case, the slew at D pin will depend upon the loading at the out port. Hence check arc value will essentially depend upon the slews at the D and CK pins and the out load as well. But currently model extraction does not support 3D modeling of check arcs.

Load dependent check arc



The following timing report shows setup_rising present in the ETM:

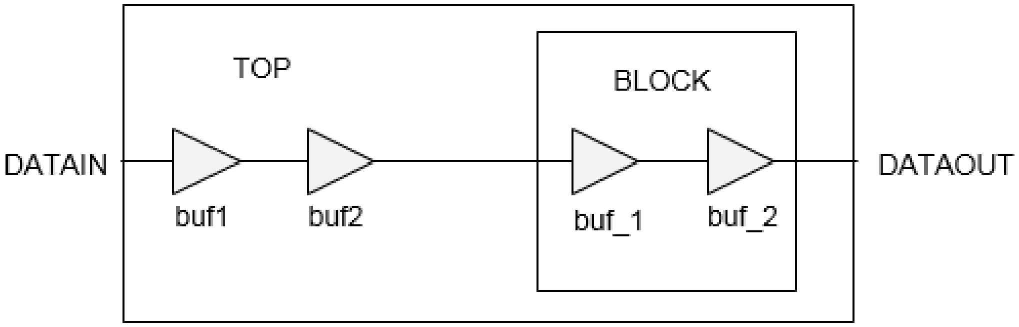
```
Path 1: VIOLATED Hold Check with Pin cppr_block/CLKIN
Endpoint: cppr_block/DATAIN (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of 'clk2'
Path Groups: {clk}
Other End Arrival Time          0.000
+ Hold                          5.000
+ Phase Shift                   0.000
= Required Time                 5.000
Arrival Time                    0.500
Slack Time                     -4.500
Clock Rise Edge                        0.000
+ Input Delay                          0.500
= Beginpoint Arrival Time              0.500
Timing Path:
--------------------------------------------------------------------
Pin                 Arc            Cell            Edge      Arrival
                                                              Time
--------------------------------------------------------------------
DATAIN ->           DATAIN ^                        ^          0.500
cppr_block/DATAIN                 cppr_block        ^          0.500
--------------------------------------------------------------------
```

The following timing report shows no setup_rising in the ETM:

```
Path 1: VIOLATED Hold Check with Pin cppr_block/CLKIN
Endpoint: cppr_block/DATAIN (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of 'clk2'
Path Groups: {clk}
Other End Arrival Time          0.000
+ Hold                          5.000
+ Phase Shift                   1.800
= Required Time                 6.800
Arrival Time                    0.500
Slack Time                     -6.300
Clock Rise Edge                         0.000
+ Input Delay                           0.500
= Beginpoint Arrival Time               0.500
Timing Path:
-------------------------------------------------------
Pin                Arc      Cell         Edge   Arrival
Time
-------------------------------------------------------
DATAIN ->          DATAIN ^                 ^     0.500
cppr_block/DATAIN            cppr_block     ^     0.500
-------------------------------------------------------
```

4. When ETM is read at the top, the information about the stage count inside the block is lost. Hence for paths coming in/out of the block, incorrect stage count is calculated while applying AOCV derate (on the top level instances which fall in this path).

ETM at the top level



When BLOCK netlist is read at the top level, the stage count for buf1 and buf2 is 4, but when an ETM of BLOCK is read, their stage count becomes 2.

The following example shows a timing report with BLOCK netlist:

```
Path 1: MET Late External Delay Assertion
```

```
Endpoint: DATAOUT2 (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of '@'
Path Groups: {clk}
Other End Arrival Time          0.000
- External Delay                0.010
+ Phase Shift                   3.600
= Required Time                 3.590
- Arrival Time                  0.354
= Slack Time                    3.236
Clock Rise Edge                          0.000
+ Input Delay                            0.000
= Beginpoint Arrival Time                0.000
-------------------------------------------------
Aocv       Aocv       Cell       Pin
Stage      Derate
Count
------------------------------------------------
                                DATAIN
4.000      1.169     BUF        buf2/Y
4.000      1.169     BUF        buf_2/Y
4.000      1.169     BUF        BLOCK/buf7/Y
4.000      1.169     BUF        BLOCK/buf8/Y
5.000      1.167                DATAOUT2
------------------------------------------------
```

The following example shows a timing report with BLOCK ETM:

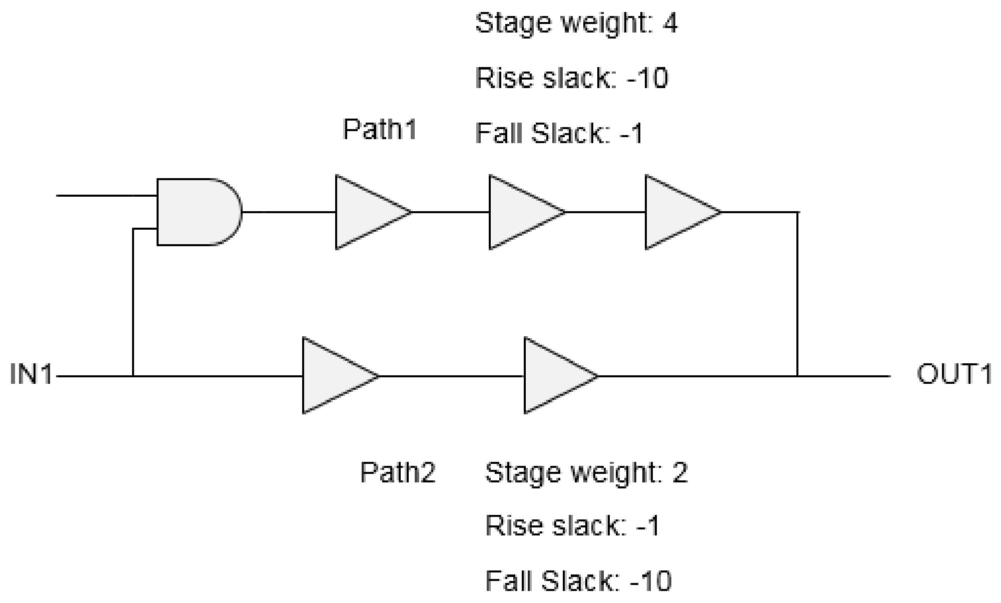
```
Path 1: MET Late External Delay Assertion
Endpoint: DATAOUT2 (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of '@'
Path Groups: {clk}
Other End Arrival Time         0.000
- External Delay               0.010
+ Phase Shift                  3.600
= Required Time                3.590
- Arrival Time                 0.325
= Slack Time                   3.265
Clock Rise Edge                         0.000
+ Input Delay                           0.000
= Beginpoint Arrival Time               0.000
--------------------------------------------------------------------
Aocv      Aocv        Cell         Pin
Stage     Derate
Count
```

```
--------------------------------------------------------------------
                                   DATAIN
2.000    1.173       BUF           buf2/Y
2.000    1.173       BUF           buf_2/Y
3.000    1.171       cppr_block    BLOCK/DATAOUT2
4.000    1.169                     DATAOUT2
--------------------------------------------------------------------
```

While modeling any arc, the rise and fall constraints are written separately. While modeling stage weights, the worst of rise/fall stage weight is written in ETM.

Consider the following example showing two paths with given stage weight and rise/fall slack values. The rise and fall arcs will be modeled corresponding to Path1 and Path2, respectively, between IN1/OUT1. The single value of stage weight (4 in this case) will be written in ETM. Ideally, the fall constraint should have the stage weight of 2 in this case, but due to the limitation, the stage weight becomes 4 adding to the pessimism in analysis.

Modeling stage weights



```
pin (DATAOUT ) {
direction : output ;
timing() {
    timing_type : combinational ;
    timing_sense : positive_unate ;
    cell_rise (lut_timing_2 ){
       values(\
       " -10, -20", \
       " -30, -40" \
       );
```

```
     }
cell_fall (lut_timing_2 ){
    values(\
    " -10, -20", \
    " -30, -40" \
    );
   }
aocv_weight : 4.0000;
related_pin :" CLKIN ";
}
```

6. The modeling of ETM with waveform propagation is not fully supported in Innovus. It is recommended to turn off waveform propagation before running the `do_extract_model` command. It can be done by using the following setting:

```
setDelayCalMode -equivalent_waveform_model none
```

Validation of Generated ETM

The extracted models should be validated before stitching to the top level for their accuracy and coverage. This block-level validation can be achieved by comparing the interface timing of the extracted timing models against the original gate-level netlist, by using the following commands.

- `write_model_timing`
- `compare_model_timing`

A brief description of the ETM commands used in a validation flow is given below

Commands Used in Validation Flow

do_extract_model
The ETM is generated by using the following command:

```
do_extract_model top.lib -verilog_shell_file test.v -verilog_shell_module test-
assertions test.asrt
```

This command writes the following files:

- verilog_shell_file (test.v): Verilog having an ETM instantiated in it. In case the block constraints file has the `set_driving_cell` command, then the cell used there will be instantiated in test.v.

- verilog_shell_module(test): Module name of test.v.
- top.lib: The resulting ETM.
- test.asrt: The resulting exception file to be used in a validation flow.

The `do_extract_model -check` parameter checks the extracted timing model for any possible Liberty-related issue. When specified, the command displays detailed error/warning messages and their summaries.

write_model_timing

The `write_model_timing` command writes a report on the interface timing of a specified netlist. The use model of the command is as follows:

`write_model_timing` –type slack ref.rpt

The report ref.rpt contains the following timing view properties to capture the timing characteristics of the design being extracted as a timing model:

- Slack or Arc Value: Reports the worst-case slack or arc value for each path from input port to clock, from clock to output port, and from input port to output port.
- Transition Time: Reports the actual transition time at each port for the four delay types: min_fall, min_rise, max_fall, and max_rise.
- Capacitance: Reports the maximum total (lumped) capacitance at each port, and if available, the effective capacitance.
- Design Rules: Reports all the design rules that apply to each port, including the maximum capacitance, minimum capacitance, maximum transition time, maximum fan-out (for input ports), and fan-out load (for output ports).

compare_model_timing

The `compare_model_timing` command compares two reports generated by the `write_model_timing` command. If the timing parameter values in the two files are the same or within the specified tolerance, then the result is pass (otherwise fail). The use model is as follows:

```
compare_model_timing -ref ref.rpt -compare comp.rpt -outFile final.rpt
percent_tolerance 3 -absolute_tolerance [expr 0.30/$timeUnit]
```

The `compare_model_timing` report shows the comparison results for individual paths, ports, and timing parameters. The resulting comparison report has the same sections as the interface timing report: slack, or arc value, transition time, capacitance, design rules. There are two tolerance settings as follows:

- Absolute tolerance: Indicates the absolute acceptable difference between compared parameters in library time units.

- Percentage tolerance: Shows the percentage of acceptable difference between compared parameters.

A path is flagged as a failure by the `compare_model_timing` command if it violates both the absolute and percentage tolerance values.

Validation Flow - MMMC Designs

Case 1: In MMMC configuration, only one view should be active before the `do_extract_model` command is issued. To set a view, you can use the `set_analysis_view -setup {$viewName} -hold {$viewName}` command.

The model is extracted using the `do_extract_model` command along with the following set of commands:

```
set viewList [all_analysis_view]
foreach viewName $viewList {
set_analysis_view -setup {$viewName} -hold {$viewName}
spefIn -rc_corner $rcCorner.spef
file mkdir $viewName
do_extract_model -view $viewName $viewName/test.lib -assertions \ $viewName/test.asrt -
verilog_shell_file $viewName/test.v \
-verilog_shell_module test
write_model_timing -view $viewName -type slack -verbose $viewName/ref.rpt
}
exit
```

This will create a separate directory for each view in $viewList, and files from each view will be written in the corresponding directory. The generated ETM and the `write_model_timing` report from the complete netlist will be saved in the corresponding ($view) folder.

Case 2: For validation of ETM generated from multiple views, the following set of commands can be used to read the ETMs from the corresponding directory:

```
set_analysis_view -setup {view1} -hold {view1}
do_extract_model my_view1.lib -lib_name extracted_model -cell_name extracted_cell -
tolerance 0.0 -verilog_shell_file top.v -verilog_shell_module test_top -view view1
write_model_timing -type slack netlist_view1.rpt -view view1
set_analysis_view -setup {view2} -hold {view2}
do_extract_model my_view2.lib. -lib_name extracted_model -cell_name extracted_cell -
tolerance 0.0 -verilog_shell_file top.v -verilog_shell_module test_top -view view2
write_model_timing -type slack netlist_view2.rpt -view view2
```

```
exit
```

The last step is the validation report generation for each analysis view. The `write_model_timing` command with the `-view` option will generate a report file that will contain the interface timing information for this design for the specific view. First you should save the timing for the original netlist and then this will be used to compare with the report file generated with the model instantiation.

Auto-Validation of ETM

Innovus provides an automated validation flow for ETM, in which all the validation steps mentioned in the previous steps can be run automatically. This can be done by using the `do_extract_model -validate` parameter.

Note: When the `-validate` parameter is used, the existing validation directory will be removed and a new directory will be created. It is recommended to specify different directories for the output ETM file and auto-validation reports.

This flow will write out the required `write_model_timing` and `compare_model_timing` reports in `$val_dir`. At the end of the auto-validation, the shell with block netlist is retained. The comparison of both setup and hold checks is performed by the auto-validation.

The following summary is generated at the end of the auto-validation:

```
-----------------------------------------------------------------------
|Total_fail | Slack_Fail | Cap_Fail | Trans_Fail | DRV_Fail | checkType
-----------------------------------------------------------------------
|    5      |     2      |    1     |     0      |    2     |   setup
|    2      |     1      |    0     |     0      |    1     |   hold
-----------------------------------------------------------------------
```

ETM Extremity Validation

The ETM models the timing arcs in the design for a range of discrete input-slews/output-loads. The slew/load asserted using SDC at the block-level during ETM characterization represents the anticipated context of the ETM instance at the top-level. However, the current validation of the ETM is done only for the input-slews/output- loads that have been asserted on the design during the ETM extraction. Therefore, the current validation of ETM is incomplete in the sense that the validation is not being done for a complete range of slew/load points for which ETM has been characterized.

Additionally, in GBA mode the problem of ETM validation is computationally more difficult. In GBA mode, the assertion of a slew on a particular input port may have an impact on the timing of a path starting from some other input port. Therefore, different combinations of input slews at the input ports will result in different timing behavior of the block in GBA mode. Since there can be exponentially large number of combination of slews at the input ports, validation of ETM in GBA mode is computationally more difficult.

The “Extremity-Validation” of ETM (also referred as EV_ETM) validates ETM at the minimum/maximum value of the slew at the input ports and minimum/maximum value of the output load at the output ports, thereby validating the ETM to a greater extent of the possible scenarios at the top. The minimum/maximum slew at the input port is defined as the minimum/maximum slew for which that input port has been characterized in the ETM library. Similarly, the minimum/maximum load at the output port is defined as the minimum/maximum load for which that output port has been characterized in the ETM library.

The minimum/maximum value of the slew/load will yield four corners of the ETM context, namely:

1. Corner 1 (Fast): the minimum slew at the input ports and the minimum load at the output ports

2. Corner 2: the minimum slew at the input ports and the maximum load at the output ports

3. Corner 3: the maximum slew at the input ports and the minimum load at the output ports

4. Corner 4 (Slow): the maximum slew at the input ports and the maximum load at the output ports

The validation flow in exhaustive mode, is same as that in non-exhaustive mode, that is:

1. `do_extract_model`
2. `write_model_timing` at the block level
3. Read ETM at the top level
4. `write_model_timing` at the top level
5. `compare_model_timing`

This flow is controlled by the `timing_extract_model_exhaustive_validation_mode` global variable.

Additional files generated in EV-ETM (but not generated in normal validation) will be stored in the current working directory by default. However, location to place these files can be controlled by the `timing_extract_model_exhaustive_validation_dir` global variable. These files are written as hidden files.

Also note that the `timing_extract_model_exhaustive_validation_dir` global variable should point to the same directory at the block and top level. When this global variable is not used at the block level, then the EV-ETM files (in step 1 and 2 above) will be dumped in the working directory at the block level. If the working directory at the top level is different than that at the block level, then at the top level `timing_extract_model_exhaustive_validation_dir` global variable should point to the block level working directory.

The `compare_model_timing` command will compare the `write_model_timing` report output written at the block level with the corresponding top-level report. The output of `compare_model_timing` command, corresponding to four corners is written to a file named, <output_file_name>_ev, where <output_file_name> is the name of the output-file. The results of four different corners are written in four different columns as shown below:

```
#Slack(SS)/Status   Slack(SF)/Status    Slack(FS)/Status    Slack(FF)/Status
 Transition   Arc-Type   From To

########################################################################################
#######################
7.511[7.511]/PASS   7.511[7.511]/PASS   7.466[7.466]/PASS   7.466[7.466]/PASS
rise/rise   setup CLK1   CLOCK1
9.666[9.665]/PASS   9.666[9.665]/PASS   9.336[9.321]/PASS   9.336[9.321]/PASS
fall/rise   setup CLK3   CLOCK2
```

In this mode, there is no change in the use models of `do_extract_model`, `write_model_timing`, and `compare_model_timing` commands. The behavioral difference (between default validation and EV_ETM) is the creation of additional four files corresponding to the four corners.

If the slew propagation is set to worst, then the `do_extract_model` and `write_model_timing` commands will issue a warning

Limitation/Implications of EV-ETM

The EV-ETM flow will have the following limitations/implications:

- The runtime of validation will appreciably increase.
- The EV-ETM will be supported only in the path_based mode and not in worst_case mode.
- The EV-ETM, though validates the ETM at the corners, cannot be considered as fully rigorous. For example, the proposed methodology does not validate the timing obtained by interpolation of the tables characterized in ETM.

7

Prototyping Flow Capabilities

Innovus enables quick full-chip virtual prototyping to accurately capture downstream physical/electrical impacts at the beginning of the design cycle. Its unique prototyping flow capabilities make hierarchical implementation easier and faster for giga-scale, high-speed designs.

Using Early Global Route for Congestion and Timing Analysis

The Early Global Route or the `earlyGlobalRoute` Router performs quick global routing for estimating routing-related congestion and parasitic (resistance and capacitance) values. It is the default pre-route routing engine for full flow that aims to provide a good turnaround time (TAT) and a good better correlation with NanoRoute. You can use Early Global Route results to estimate and view routing congestion, and to estimate parasitic values for optimization and timing analysis. When used during prototyping, Early Global Route creates actual wires, so that you can get a good representation of RC and coupling for timing optimization at an early stage in the flow. Early Global Route also produces a congestion map that you can view to get an early feedback on whether the design is routable.

Note: The Early Global Router does not guarantee DRC-clean routing results. In congestion estimation, Early Global Route is used to calculate the track utilization by routing wires. Using the Early Global Router is correct if there is any overlap between Early Global Route wires and pre-routed wires as you should focus on the congestion value.

Note: Do not perform signal integrity analysis on a design that has been routed using Early Global Route, because the routes are only used to estimate parasitic values for timing analysis. Route designs with NanoRoute, if you want to perform signal integrity analysis.

You can use Early Global Route during virtual prototyping, hierarchical floorplanning, block implementation, and top-level implementation.

Related Information

- Prerequisite for Running Early Global Route
- Routing a Flat Design
- Routing a Partitioned Design
- Using Early Global Router on MSV Designs
- Analyzing Route Data
- Congestion Distribution Report

Prerequisite for Running Early Global Route

For running Early Global Route directly, the design must be successfully placed and it must be loaded into the current Innovus session.

Note: Early Global Route is the default engine in flat design flow.

Additional Information

- **Correlation with NanoRoute**: Early Global Route solves the correlation issue between pre and post NanoRoute. The difference between the Early Global Route congestion map and NanoRoute DRC violation marker indicates a correlation issue that needs to be fixed.
- **Wire Overlap**: Early Global Route is only used for congestion repair and timing estimation. The running results may not be DRC clean. Cadence recommends you to use NanoRoute for the final tape out so that you can find overlapping with wires or routing blockage boundary by Early Global Route. However, Early Global Route can consider the available routing tracks and wires spacing during congestion calculation, resulting in a congestion value that is correct.

Related Information

- Using Early Global Route for Congestion and Timing Analysis
- Routing a Flat Design
- Routing a Partitioned Design
- Using Early Global Router on MSV Designs
- Analyzing Route Data
- Congestion Distribution Report

Routing a Flat Design

In a flat design, Early Global Route can route through guides, regions, and fences, as long as there are no routing blockages or hard blocks.

- Use Model
- Tuning the Early Global Routing Congestion Value
- Using Bus Guides
- Routing Secondary PG Pins
- Routing on Reverse Direction

Use Model

Run Early Global 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 the congested areas that might cause routing problems later in the design session.

1. Set the preferred routing layer using the `setDesignMode` command.
    a. Define the bottom routing layer using the `-bottomRoutingLayer` *`value`* parameter.
    b. Define the top routing layer using the `-topRoutingLayer` *`value`* parameter.
2. The clock routing wires can have a larger width and spacing value than the other signal wires. This requires Early Global Route to preserve more routing resources for clock wires. Use one of the following methods to specify the routing resources for clock wires:
    a. Load the FE-CTS spec files using the `specifyClockTree -file` *`string`* command and then enable the `-earlyGlobalHonorClockSpecNDR` parameter of the `setRouteMode` command (`setRouteMode -earlyGlobalHonorClockSpecNDR true`) to force Early Global Route to honor the routing constraints defined in clock tree spec file.

    Note: The FE-CTS flow is a limited-access feature in this release. This feature is enabled by a variable specified using the `setLimitedAccessFeature` command. To use this feature, contact your Cadence representative to explain your usage requirements.

    b. Define the number of tracks occupied by Early Global Route, using the `setRouteMode -earlyGlobalNumTracksPerClockWire` *`value`* command.
    Note: The CCOpt flow contains options to convert routing constraints into db. To use this feature, contact your Cadence representative for the flow support.
    Note: Early Global Route also honors the following net attributes defined by the `setAttribute` command.

Attribute Name	**Description**
`non_default_rule`	Defines the specified non-default rule for the nets.
`preferred_extra_space`	Specifies the preferred extra spacing for the nets.
`top_preferred_routing_layer`	Specifies the preferred highest layer for routing specified signal nets.
`bottom_preferred_routing_layer`	Specifies the preferred lowest layer for routing specified signal nets.
`skip_routing`	Specifies that the specified net should not be routed.
`shield_net`	Preserves the routing resource for shielding nets.

3. Call Early Global Route using the `earlyGlobalRoute` command.

Tuning the Early Global Routing Congestion Value

Early Global Routing enables you to tune the congestion results using the `-earlyGlobalSupplyScaleFactorH` and `-earlyGlobalSupplyScaleFactorV` parameters of the `setRouteMode` command. This provides a better correlation between Early Global Route and NanoRoute.

```
setRouteMode -earlyGlobalSupplyScaleFactorH value
setRouteMode -earlyGlobalSupplyScaleFactorV value
```

Where, the default *`value`* is 1

Max : `1.250000`
Min : `0.750000`

The *`value`* defines the capacity of available tracks in each gcell. You can get a pessimistic congestion map with a smaller value and an optimistic result with a larger value.

You can increase the value of these parameters over 1 if Early Global Route is showing too much congestion but the number of DRCs are not many. Alternatively, you can decrease the value if the congestion is small but the number of DRCs are many.

Note: Even though Early Global Route is tuned by technology continuously, some cases may require tuning manually. Even though these parameters can be used for tuning the congestion results, Cadence recommends you to contact your Cadence representative if you see a correlation issue.

Correlation issues that may require tuning can be identified by comparing the congestion(hotspot) to the NanoRoute DRC. These issues can degrade all sorts of QOR. After tuning the Early Global Routing congestion value, there may be improvement in all sorts of metrics, however, there may be tradeoffs. For example, if you make the congestion pessimistic you can improve DRC and runtime but you might increase density and maybe end up with worse timing.

Note: You can use the `reportCongestion` command after routing the design to view the average congestion and the local hotspot score report.

Using Bus Guides

Early Global Route honors the routing pattern constraint defined by bus guides. When you define the routing layer and pattern for specified nets, the `earlyGlobalRoute` command automatically checks the bus guide information, then routes these nets based on their defined bus guide constraints. However, the bus guide is a soft constraint for Early Global Route, which means the nets can be routed outside a bus guide.

The following limitations exist for the Early Global 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.

Routing Secondary PG Pins

Early Global Route supports routing of secondary PG pins. This makes the congestion results accurate. The following steps describe the routing flow for secondary PG pins:

- Use the `setPGPinUseSignalRoute` command to set the internal bit for the cell/pin pair to be routed by the signal router.
- Use the `setRouteMode -earlyGlobalRouteSecondPG true` command to enable the secondary PG pin feature.
- *Optional*: Define the non-default rules using the `setAttribute` command.
- Optional: Define the maximum fanout for Early Global Route using the `setRouteMode -earlyGlobalSecondPGMaxFanout` command.
- *Optional*: Define the connected stripe layer using the `setRouteMode -earlyGlobalRouteStripeLayerRange` command. This option enables you to control the layer of power stripe to connect.
- Call the `earlyGlobalRoute` command.

Note: To enable the secondary PG pin feature, you can use the `setRouteMode -earlyGlobalRouteSecondPG true` command.

Routing on Reverse Direction

Early Global Route partially supports routing along the reverse direction. You can use the `-earlyGlobalReverseDirection` parameter of the `setRouteMode` command to reverse the routing direction in the given region on the specified layer-range.

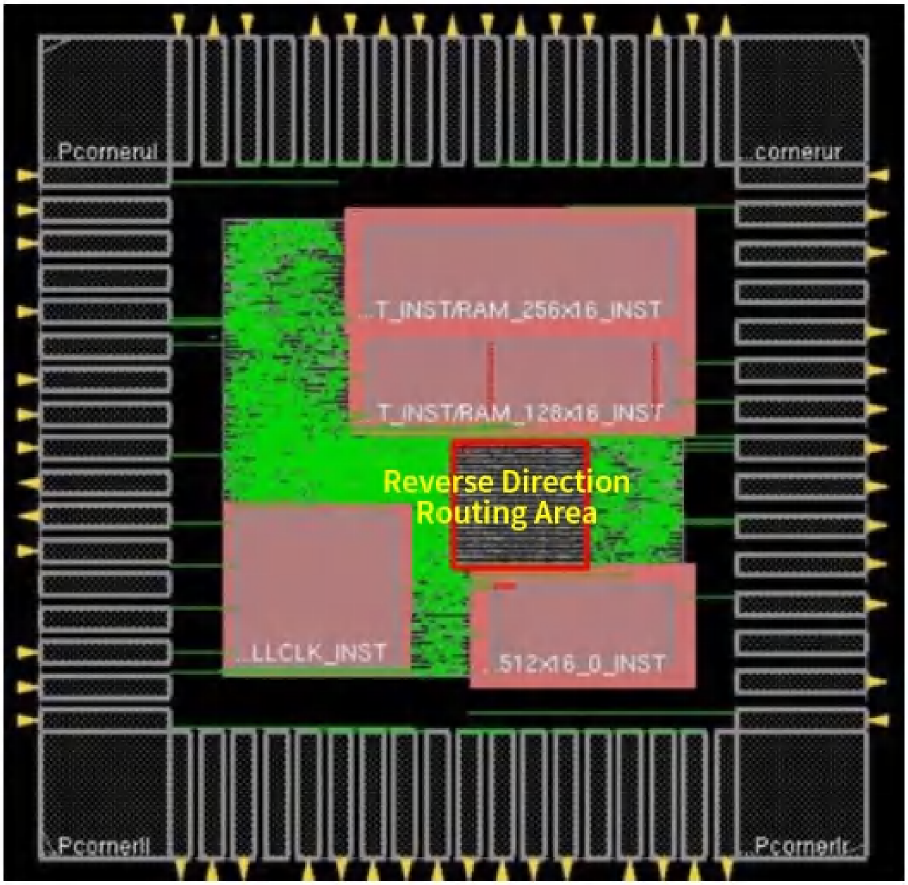
Use the following command to specify an area in which Early Global Route routes wires in the non-preferred direction:

```
setRouteMode -earlyGlobalReverseDirection "(x1 y1 x2 y2) Metal2:Metal2 (x3 y3 x4 y4)
Metal3:Metal3 ..."
```

Note: You can define the reverse direction routing area by *"(x1 y1 x2 y2)"* and routing layer by *"Metal2:Metal2"*. In release 15.2, the Early Global Router only supports a single layer with this option. Consequently, *"Metal2:Metal3"* is not supported.
The `-earlyGlobalReverseDirection` parameter only has impact on the congestion value calculation. By routing along the reverse direction, you can reduce the routing capacity on the specified layer and add extract resources on the above and below layers.
Innovus does not display the routing wires in the reverse direction routing region.

For example, the following command reverses the routing direction in the specified region on layer Metal3.

```
setRouteMode -earlyGlobalReverseDirection "(757.996 542.697 995.323 772.5685) Metal3:Metal3"
```



The congestion value without reverse direction routing:

```
Overflow after earlyGlobalRoute 0.02% H + 0.00% V
```

The congestion value with reverse direction routing:

```
earlyGlobalRoute overflow: 0.93% H + 0.10% V
```

Related Information

- Using Early Global Route for Congestion and Timing Analysis
- Prerequisite for Running Early Global Route

Routing a Partitioned Design

In a flat design, Early Global 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. Early Global Route provides options to route the partitioned designs with or without honoring fence and pin constraints.

Note:

- Early Global Route supports specified routing constraint setting for each partitions. You can define the complete list of partitions that route in a HonorFence and HonorPin style.
- Early Global Route fully supports hierarchical design flow and is turned on automatically.

Use model

As per requirement, use the following attributes of the `setRouteMode` command:

- `setRouteMode -earlyGlobalRoutePartitionHonorFence` *`list_of_ptn_cell_names`*
  Defines the partition cells in which Early Global Route honors the fence constraints. You should specify the partition cell names, which are Verilog module names that can contain alphabetic characters, numeric characters, an underscore, and/or a dollar sign.
- `setRouteMode -earlyGlobalRoutePartitionHonorPin` *`list_of_ptn_cell_names`*
  Defines the partition cells in which Early Global Route honors partition fences with single-entry constraints and pre-assigned pins (pins marked `FIXED`) and assigned pins (pins marked `PLACED`). Placed pins includes the pins with placed and fixed status.
- `setRouteMode -earlyGlobalRoutePartitionPinGuide {true|false}`
  Defines whether Early Global Route should honor fixed pin and user-specified pin guide. When enabled, Early Global Route uses the pin guide statements in the floorplan file to guide the routing through partition pin points.

Notes:

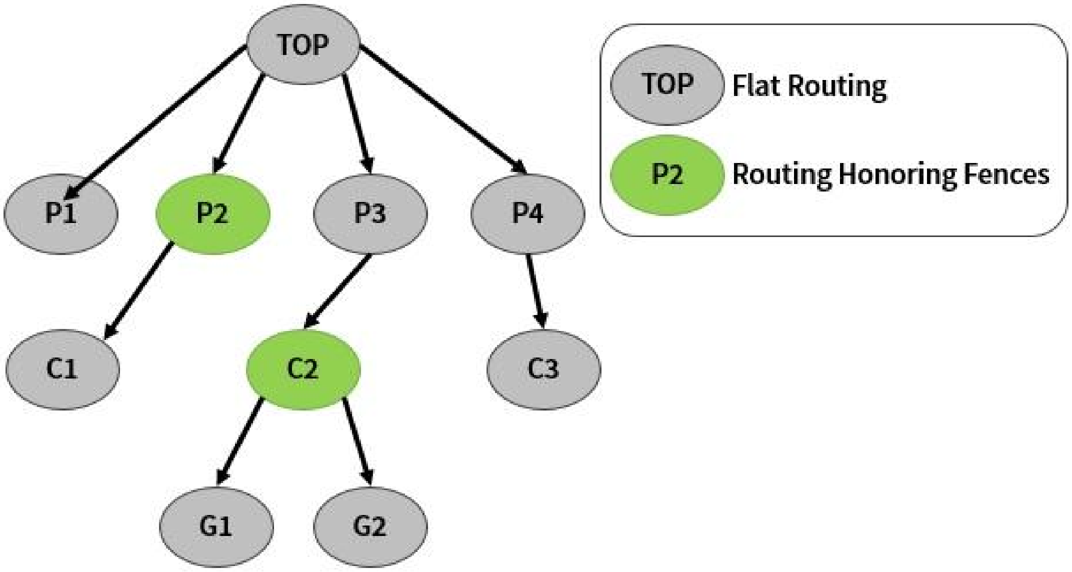
- Early Global Route does not honor the fences or fixed pin constraints by default. You must define the `-earlyGlobalRoutePartitionHonorFence` attribute to honor the constraints.
- For flat partitioned designs, you can specify `-earlyGlobalRoutePartitionHonorFence` to simulate channel-based routing. For channel-less designs, you must perform feedthrough insertion before using this attribute.
- The HonorPin setting has priority over HonorFence setting. As a result, if the HonorPin option is specified before or after the HonorFence option for a partition, the partition will be routed as HonorPin. You must reset the HonorPin setting if the same partition has to be routed as HonorFence later.
- The default value for `setRouteMode -earlyGlobalRoutePartitionPinGuide` is true.
- If the `-earlyGlobalRoutePartitionHonorPin` attribute has been defined, Early Global Route automatically honors the fence constraints even if the `-earlyGlobalRoutePartitionHonorFence` attribute has not been specified.

Nested Partition Support by Early Global Route

Early Global Route supports nested partitions. The partitions specified with the `-earlyGlobalRoutePartitionHonorFence` and the `-earlyGlobalRoutePartitionHonorPin` parameters are routed in the style that honors fence and pin constraints. All other partitions are still routed flat.

Example:
In the following diagram, P1, P2, P3, P4, C1, C2, C3, G1, and G2 are partitions, however, only P2 and C2 are defined to honor fence constraints. Consequently, `earlyGlobalRoute` routes partition P2 and C2 using the HonorFence routing style while other partitions are routed flat.

```
setRouteMode -earlyGlobalRoutePartitionHonorFence {P2 C2}
earlyGlobalRoute
```



Early Global Route Behavior in Partitioned Designs

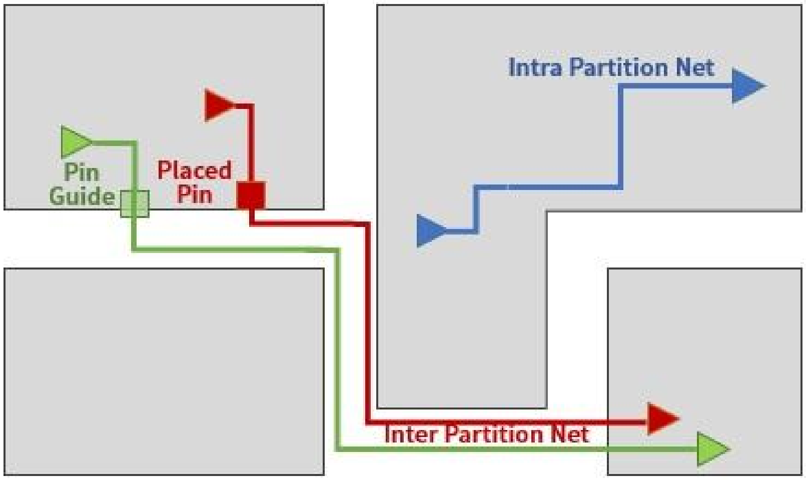
Routing Behavior When Pin Guides Are Honored:

The following table explains the Early Global Route behavior in partitioned designs when pin guides are honored:

	Default Behavior	With -earlyGlobalRoutePartitionHonorFence	With -earlyGlobalRoutePartitionHonorPin

Honor Partition Fences with Single Entry	**N**	**Y**	**Y**
Allow Crossing Over Other Partitions	**Y**	**N**	**N**
Honor Pin Guides	**N**	**Y**	**Y**
Honor Fixed Pins	**N**	**Y**	**Y**
Honor Placed Pins	**N**	**N**	**Y**

```
setRouteMode -earlyGlobalRoutePartitionPinGuide true
earlyGlobalRoute
```

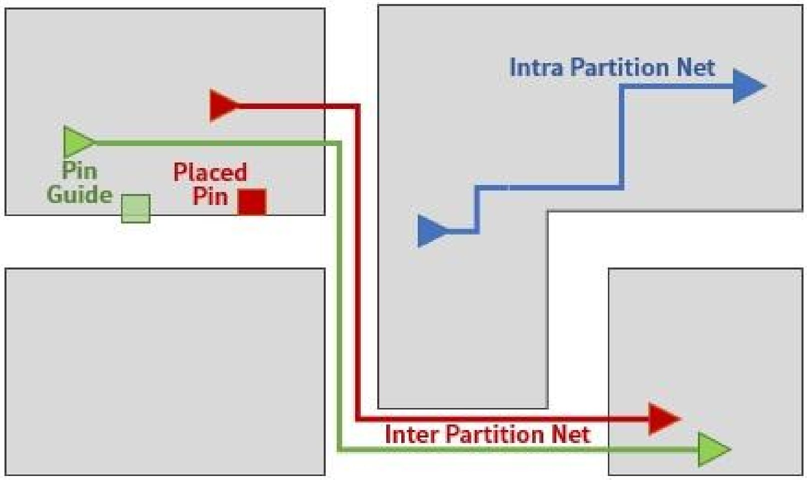


Routing Behavior When Pin Guides Are NOT Honored

The following table explains the Early Global Route behavior in partitioned designs when pin guides are not honored:

	Default Behavior	**With -earlyGlobalRoutePartitionHonorFence**	**With -earlyGlobalRoutePartitionHonorPin**
Honor Partition Fences with Single Entry	**N**	**Y**	**Y**
Allow Crossing Over Other Partitions	**Y**	**N**	**N**
Honor Pin Guides	**N**	**N**	**N**
Honor Fixed Pins	**N**	**N**	**Y**
Honor Placed Pins	**N**	**N**	**Y**

```
setRouteMode -earlyGlobalRoutePartitionHonorFence . -earlyGlobalRoutePartitionPinGuide false
earlyGlobalRoute
```



Related Information

- Using Early Global Route for Congestion and Timing Analysis
- Prerequisite for Running Early Global Route

Using Early Global Router on MSV Designs

In multiple supply voltage (MSV) designs, the nets have preferred and non-preferred power domains that have constraints on the routing pattern. Early Global Route honors this setting and routes wires through the preferred power domains. However, since it is a soft routing constraint, Early Global Route can route through non-preferred power domains with a high cost.

To enable the MSV routing feature in Early Global Route, use the following command:

```
setRouteMode -earlyGlobalHonorMsvRouteConstraint true
```

This option honors MSV domain constraints. It is used to decide whether Early Global Route should honor power domain settings. By default, Early Global Route ignores the power domain settings and routes as a flat design. When enabled, wires are routed inside preferred power domains as much as possible.

The following illustrations explain the behavior of Early Global Route results for MSV designs.

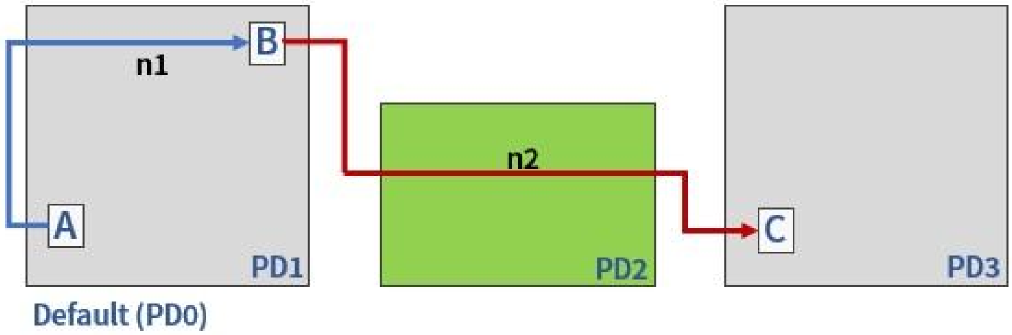
Net Name	Color	Net Type
n1	Blue	intra power domain net (PD1)

n2	Red	inter power domain net (PD1 and PD3)

Example

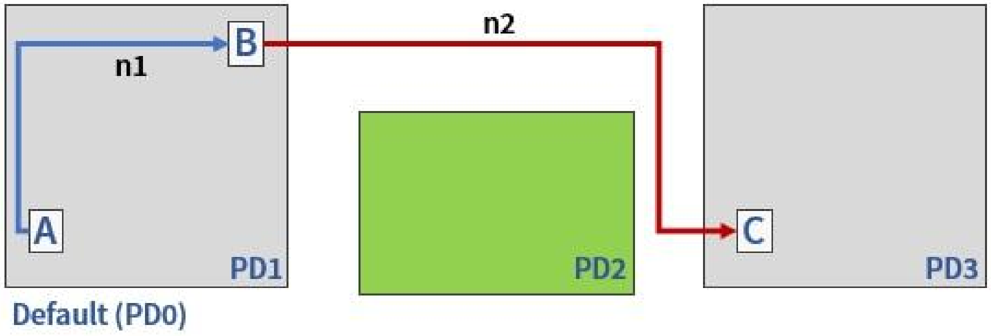
- The following command routes the design as a flat design and does not honor the MSV constraints.

```
setRouteMode -earlyGlobalHonorMsvRouteConstraint false
earlyGlobalRoute
```



- The following command routes the design by honoring the MSV constraints and the different power domain settings.

```
setRouteMode -earlyGlobalHonorMsvRouteConstraint true
earlyGlobalRoute
```



Related Information

- Using Early Global Route for Congestion and Timing Analysis
- Prerequisite for Running Early Global Route

Analyzing Route Data

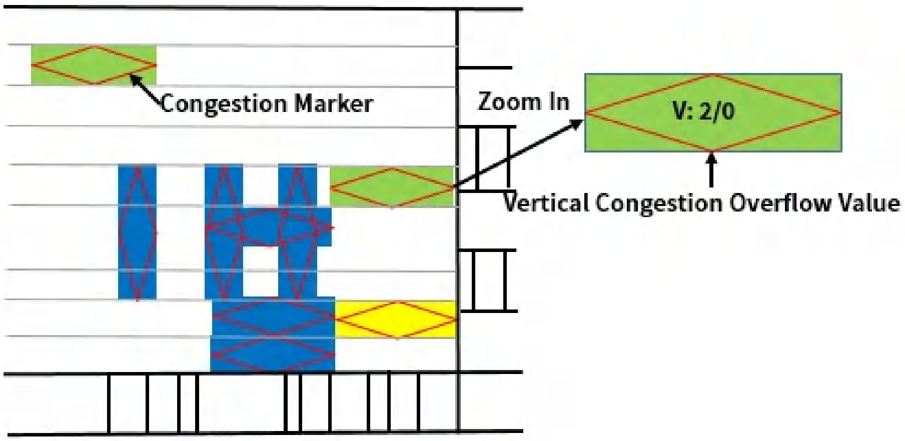
Early Global Route is the route engine before NanoRoute, which is used for congestion evaluation and timing analysis. After running Early Global Route, you can analyze the GUI and log results to check if your design is routable.

- 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.
- In the log file, inspect the Early Global Route congestion value and wire length. Use the congestion value to check whether the design is routable and the routing layer is reasonable.

Note: The Early Global Router has honors partial routing blockages that have a density attribute that defines how many percent of routing tracks are available for each layers. Early Global Route updates the supply calculation based on the available tracks per blockage ratio in the region where the blockage is.

Congestion Markers in the Display

Early Global Route can calculate the required and available tracks in each gcell. You can visually check the Early Global Route congestion statistics in the design display area of the main Innovus window to identify the tight clusters of congestion markers. Check the design display area to make sure there are no markers grouped closely together. These usually occur around blocks or between large blocks. The indicators are diamond shaped and red by default. Zoom into the area to display the vertical and horizontal congestion overflow values, as shown in the following figure.



Congestion markers contain a vertical or horizontal overflow value to identify the number of tracks required for that grid, and the actual number of tracks available. For example, in the above illustration, the vertical overflow is 2/0, which indicates that two additional tracks are required, and 0 tracks are available. Congestion marker values are based on an integer number of adjacent gcells that are grouped together to form a "super gcell." Horizontal congestion super gcells are tall, narrow boxes that typically have a height of four gcells and a width approximately equal to the height of a vertical congestion super gcell. Vertical congestion super gcells are short, wide boxes that typically have a height of one gcell and a width approximately equal to the height of a horizontal congestion super gcell.

Vertical and horizontal overflow values are calculated separately for better accuracy. The overflow value is the amount by which the track demand exceeds the track supply. The required track value is calculated by totaling 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.

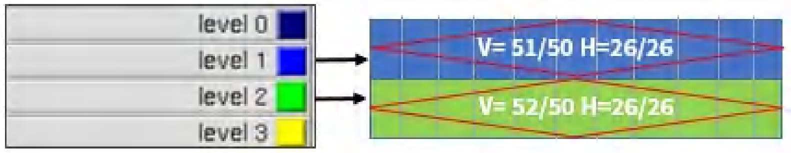
To change the size of super gcells, define the following variable:

```
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 Early Global Route requires 51 vertical tracks, the congestion marker color box is blue (by default), indicating a one-track overflow. If Early Global 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)

Related Information

- Using Early Global Route for Congestion and Timing Analysis
- "Multicolor Layers" section in the *The Main Window* chapter of the *Menu Reference*.

Congestion Distribution Report

Congestion Value Calculation

Early Global Route prints the congestion value and wire length in the innovus.log and innovus.logv files.

Note: The congestion calculation method used by Early Global Route uses the same formula as Global Route. After Early Global Route completes, you can find the following value in the log file:

```
[NR-eGR] Overflow after Early Global Route 0.03% H + 0.00% V
```

Note: The total overflow value for Early Global Route is calculated by adding the demand and capacity from each layer for each GCell.

For example, the overflow value for Early Global Route is calculated using the the following formula:

```
eGR_total_overflow = 0
For each GCell {
       Total_gcell_demand = 0
       Total_gcell_capacity = 0

       For each layer{
                    Total_gcell_demand += demand;
                    Total_gcell_ capacity += capacity
       }
       overflow = Total_gcell_demand - Total_gcell_capacity
       If ( overflow > 0 ) {
                    eGR_total_overflow += sqrt( overflow )
      }
}
```

The final calculation for the overflow value is done by dividing the total overflow value for Early Global Route by the total number of unblocked gcells:

```
Overflow eGR = eGR_total_overflow  / total number of unblocked gcells
```

Note: This calculation is done by both directions so you need to add (H) and/or (V) to present this for both directions.

Early Global Route Congestion Formula

In Early Global Route congestion formula, Innovus first adds up the total overflow value for each GCell on all the layers on vertical or horizontal direction and then derives the total overflow from its square root

Innovus adds up all the overflow numbers together and divides it by the total available cell numbers. Similar to the Global Route compatible congestion formula, Innovus also calculates the horizontal and vertical congestion value separately:

```
Overflow (H) = sqrt (number of overflow gcells for all layers (H) / total number of unblocked
GCells for a single layer)
```

The Early Global Route congestion formula uses the square root of overflow number but not the overflow gcell to calculate the overflow value. For example, if there are 1000 gcells and only 1 gcell with 9 overflow along the vertical direction, the congestion value in that direction is 0.3%. The gcell height and width equals to one stand cell row height. This formula also excludes all the fully blocked gcells from this calculation, if there are 200 fully blocked gcells, the congestion value should be 0.375%

Note: For the double pattern technology (DPT), Early Global Route cannot color wires automatically so it preserves 15% capacity on all DPT layers.

Wire Length Report

Early Global Route also calculates the routing length and via number on each layer to get the wire length distribution and compare it with the final NanoRoute result. Early Global Route honors the top and bottom layer setting by the `setDesignMode` and `setAttribute` commands.

```
[NR-eGR]             Length (um) Vias
[NR-eGR] ----------------------------------
[NR-eGR] Metal1 (1H) 0      25155
[NR-eGR] Metal2 (2V) 134127 35802
[NR-eGR] Metal3 (3H) 189844 5296
[NR-eGR] Metal4 (4V) 95825  1646
[NR-eGR] Metal5 (5H) 77137  416
[NR-eGR] Metal6 (6V) 29560  0
[NR-eGR] ----------------------------------
[NR-eGR] Total       526492 68315
[NR-eGR] ----------------------------------------------------------------------
```

Note: The default value of min routing layer is Metal2. As shown in the above results, Early Global Route does not use Metal1 for routing by default. Early Global Route can use Metal1 layer with the `-bottomRoutingLayer` setting, but cannot fully use Metal1 as the other layers.

Related Information

- Using Early Global Route for Congestion and Timing Analysis

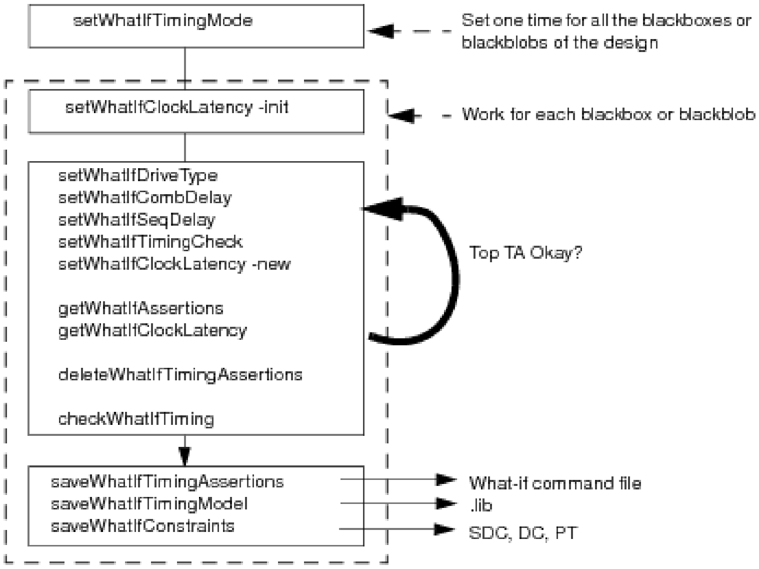
What-If Timing Analysis

Performing What-If Timing Analysis

You use blackboxes in large designs containing hierarchical flows when gate-level details are not available at the beginning of the design cycle. You can easily modify the timing model of a blackbox at the top level because it is not a hard macro. Using the Innovus™ Implementation System, you can make quick modifications to the timing model of a blackbox, and run timing analysis to check the impact of the modifications. This feature is known as what-if timing budgeting. The Innovus software provides what-if timing commands to support what-if timing budgeting. For more information on what-if timing commands, see the chapter What-if Timing Commands," in the *Innovus Text Command Reference*.

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

The following diagram shows the what-if budgeting flow.



Prerequisite

Prior to using what-if timing commands, you must load the what-if timing models into the database because the what-if timing commands simulate the modifications of the timing arcs.

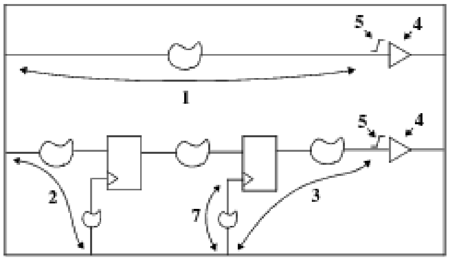
If you do not have timing models in the early design phase, you can use the `setWhatIfClockPort` command to create clock ports. You can then use the clock port to create timing arcs.

Timing Models Supported for What-If Timing Analysis

The Innovus software supports two timing models for what-if timing analysis: intrinsic and normalized. You can select only one mode at a time.

The following figure shows the intrinsic timing model.

Figure 10-1 Intrinsic Timing Model



The data types associated with the numbers in the figure above and the corresponding commands that you use to specify that data are as follows:

#	Data Type	Command
1	Type of Driver	`setWhatIfDriveType`
2	Driver input slew	
3	Clock insertion delay to internal registers	`setWhatIfClockLatency`

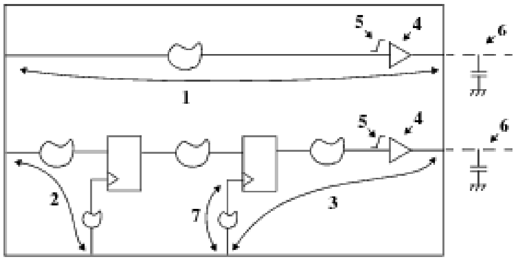
An intrinsic timing model uses the following formula for timing arcs ending on output ports:

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

If you do not use slew specifications in an intrinsic timing model, the timing arc is a 2-D timing table containing input slew and output capacitance dependencies. With slew specifications, the timing arc is only load dependent.

The following figure shows the normalized timing model.

Figure 10-2 Normalized Timing Model



The data types associated with the numbers in the figure above, and the corresponding commands that you use to specify that data is as follows:

Data Type	Command
Driver type	`setWhatIfDriveType`
Driver input slew	
Total driver output net capacitance	
Clock insertion delay to internal registers	`setWhatIfClockLatency`

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

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

Where,

`constant delay` = Timing arc delay including driver delay

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

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

In a normalized timing model mode driver input slew is always required. In this mode, timing arcs are only load dependant. If you do not specify the driver total output net capacitance, the software takes real net capacitance into account.

Using the What-If Timing Commands

You can perform the following tasks with the what-if timing commands:

- Selecting Timing Model
  Use the following command to select the timing mode:
  - `setWhatIfTimingMode`
- Defining generated clocks on internal pins:
  Use the following command to create an internal pin and to define a generated clock on the pin.
  - `createWhatIfInternalGeneratedClock`
- Set the following values on the what-if ports, if required:
  - Capacitance
  - Maximum capacitance
  - Maximum transition
  - Maximum fanout

Use the following command to set these values on the what-if ports:

- `setWhatIfPortParameters`
  By default, the parameters specified with the `setWhatIfPortParameters` command are applied to all ports in the what-if timing analysis model. If you want to apply the values for a particular port, specify the port name with the `setWhatIfPortParameters -port` parameter.

- Selecting the precedence between the values set by setWhatIfDriveType command and the values set by the setWhatIfPortParameters command
  On output ports, parameters such as capacitance value, maximum capacitance values, maximum transition value, or the maximum fanout value can come from the driver (setWhatIfDriveType command) or they can be set through the `setWhatIfPortParameters` command.

  Use the following command to define which of these values will take precedence in case of a conflict.

  - `setWhatIfTimingMode`

- **Modifying Timing Arcs**
  While what-if commands are the same for both intrinsic and normalized timing models, the delay value specified in the commands for the combinatorial and the sequential timing arcs has different meaning. The driver output net capacitance is a characteristic of the normalized timing model only. Whenever you create or modify a timing arc, the timing graph is updated automatically. The Innovus software recomputes the entire timing arc whenever any of the parameter such as clock insertion delay, timing arc delay or driver type is modified.

  Note: The timing sense of the driver is taken into account in the combinatorial what-if timing arc description--while applying the drive type, the timing sense of the combinatorial arc is replaced by the timing sense of the driver's timing arc. For sequential arcs, the timing sense is always set to `non_unate`.

  Use the following commands to modify timing arcs:

  - `setWhatIfDriveType`
  - `setWhatIfClockPort`
  - `setWhatIfClockLatency`

- **Getting Timing Arcs Assertions**
  Use the following command to get what-if timing arc assertions:

  - `getWhatIfTimingAssertions`

- **Saving Timing Arcs Assertions**
  Use the following command to save what-if timing arc assertions:

  - `saveWhatIfTimingAssertions`

- **Deleting Timing Arcs Assertions**

Use the following command to delete the what-if timing arc assertions:

  - `deleteWhatIfTimingAssertions`

- **Checking Timing Assertions**
  Use the following command to check the what-if timing assertions:

  - `checkWhatIfTiming`

- **Generating what-if timing Models**
  After modifying the what-if timing model (in memory) using the what-if command, you can generate an updated timing model (.lib).

  Use the following command to generate an updated .lib file:

  - `saveWhatIfTimingModel`

- **Generating What-If SDC constraints**
  The Innovus software generates the what-if timing constraints considering the top-level environment of the blackbox or blackblob. It provides a higher convergence for a top-down flow. The software generates drive, load and transition as IN context. The software generates the input and output delays as OUT context taking into account the last modifications done when you use the what-if commands.

  Use the following command to save the What-If constraints:

  - `saveWhatIfConstraints`

Fast Slack Timing Analysis

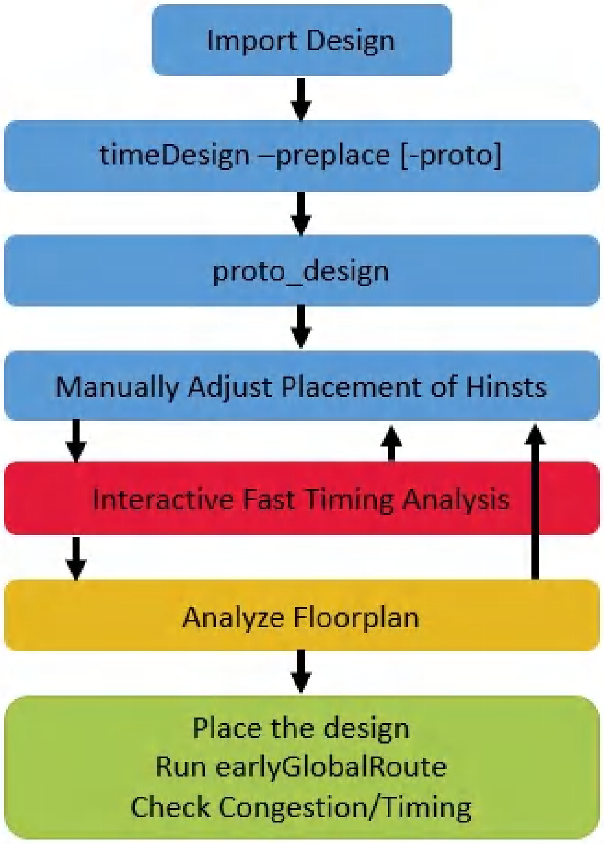
The Fast Slack Analysis feature can be used to analyze and display slack information of FlexModels and/or specified modules. Fast Timing Analysis calculates the initial timing between FlexModels and/or specific modules based on manhattan distance using psPM timing value for net delay.

Related Information

- Performing Fast Slack Timing Analysis
- Initializing Fast Slack Timing Analysis

Performing Fast Slack Timing Analysis

The Fast Slack Timing Analysis provides a fast and reliable way to interactively check timing between FlexModels and/or selected modules using GUI. The timing report is based on the reliable psPM model and the slack is calculated between two models. Once the timing between models is initialized, slack of timing path between models is updated if locations of these models are changed.

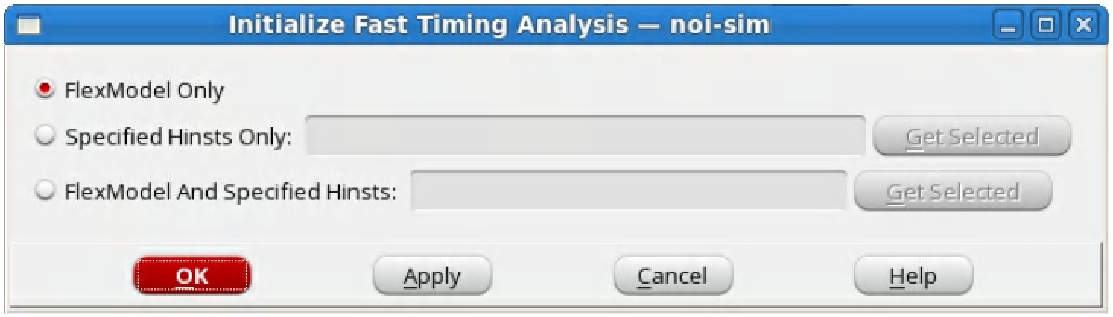


Note: It can be used for the chip planning stage of a prototyping or a during hierarchical implementation flow.

Initializing Fast Slack Timing Analysis

You can initialize the fast timing analysis using the *Initialize Fast Timing Analysis* GUI form that can be displayed by choosing *Floorplan - Generate Floorplan - Initialize Fast Timing Analysis* in the Innovus UI.

Note: The psPM model should be available before invoking fast slack timing analysis.



Initialize Fast Timing Analysis — noi-sim

● FlexModel Only
○ Specified Hinsts Only:
○ FlexModel And Specified Hinsts:

When the *Initialize Fast Timing Analysis* form is displayed:

1. Choose one of three usage models based on your design.
   - *FlexModel Only*: Initializes the timing analysis for FlexModels only.
   - *Specified Hinsts Only*: Initializes the timing analysis for user specified hinsts by GUI selection.
   - *FlexModel and Specified Hinsts*: Initializes the timing analysis for FlexModels and user specified hinsts by GUI selection.
2. Click *OK*

Once initialization is done, slack is automatically updated if the location of a model is changed and the *Slack Preference* form is displayed

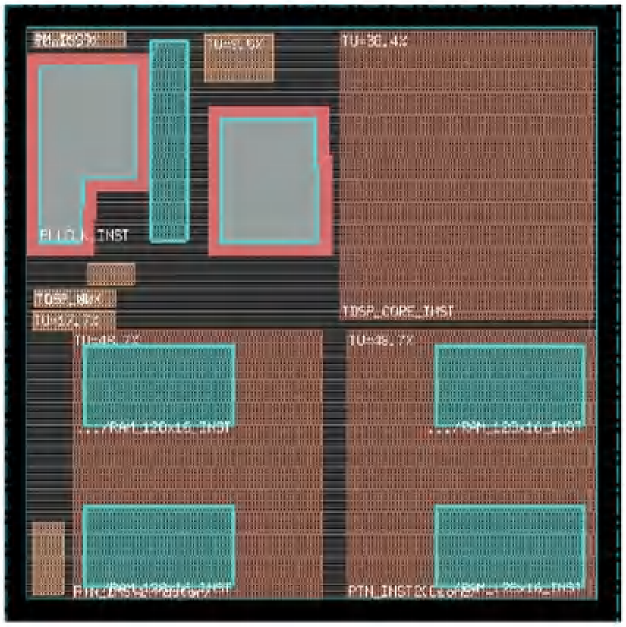
For example:

Examples of Fast Slack Timing Analysis

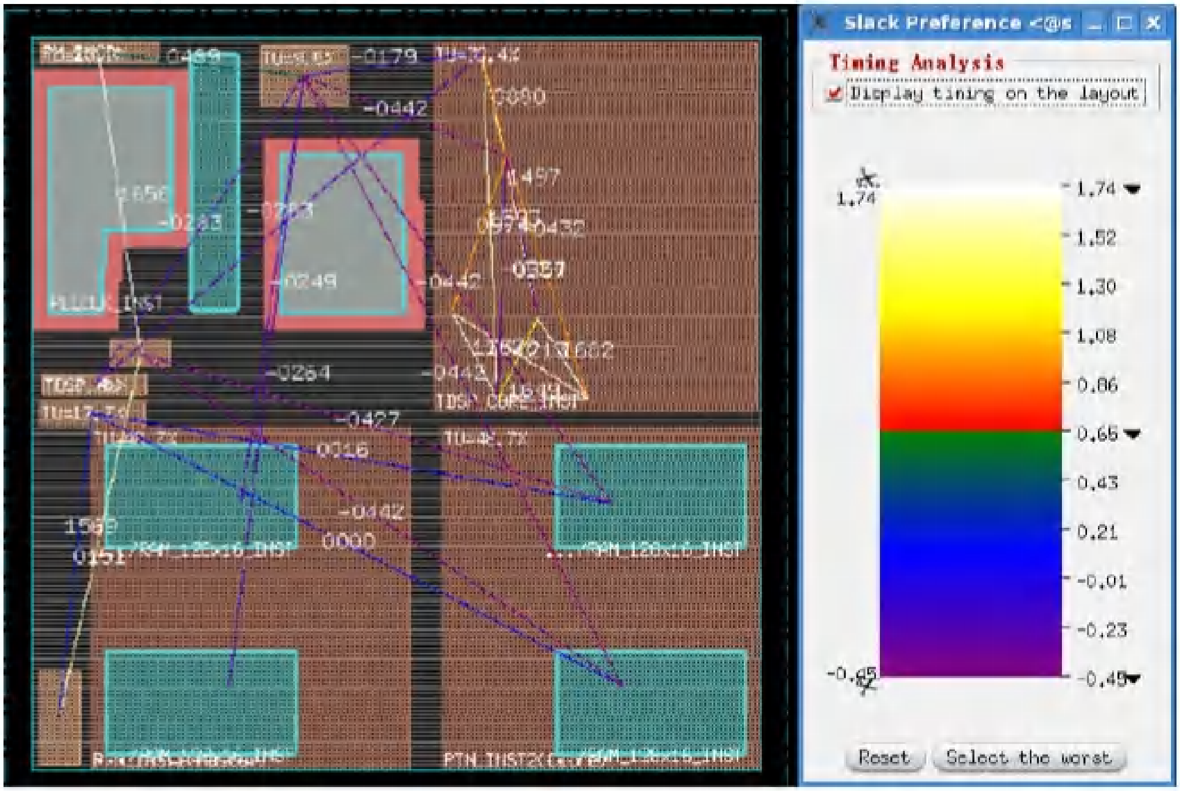
Example 1

When you choose *Floorplan - Generate Floorplan - Initialize Fast Timing Analysis - FlexModel Only*

The Initial Floorplan View



The Floorplan View with Stack Value

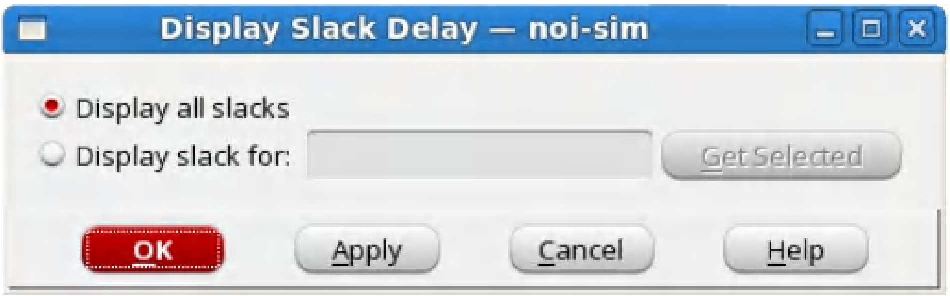


Timing Analysis

☑ Display timing on the layout

Example 2

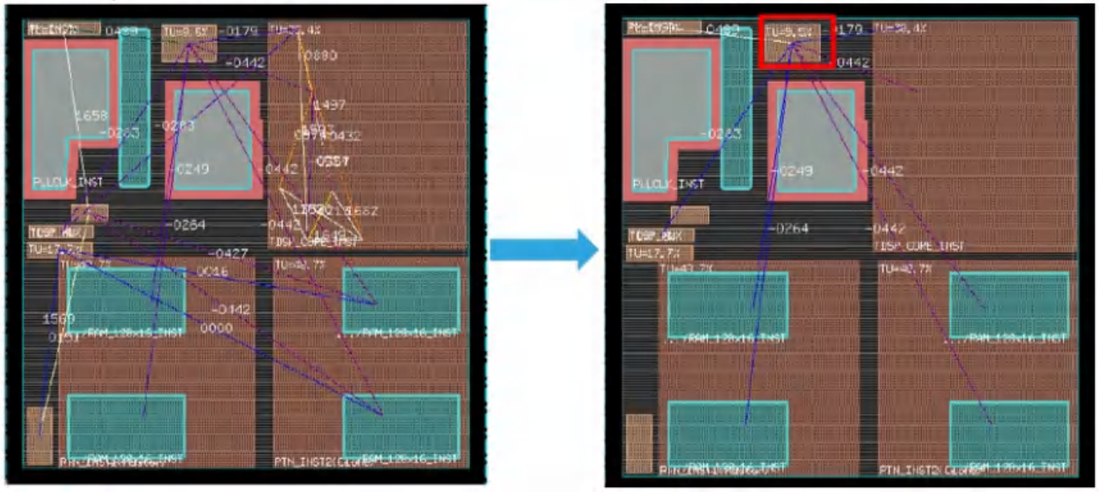
When you choose *Floorplan - Generate Floorplan - Fast Slack Analysis/Display - Display slack for*



Display Slack Delay — noi-sim

● Display all slacks

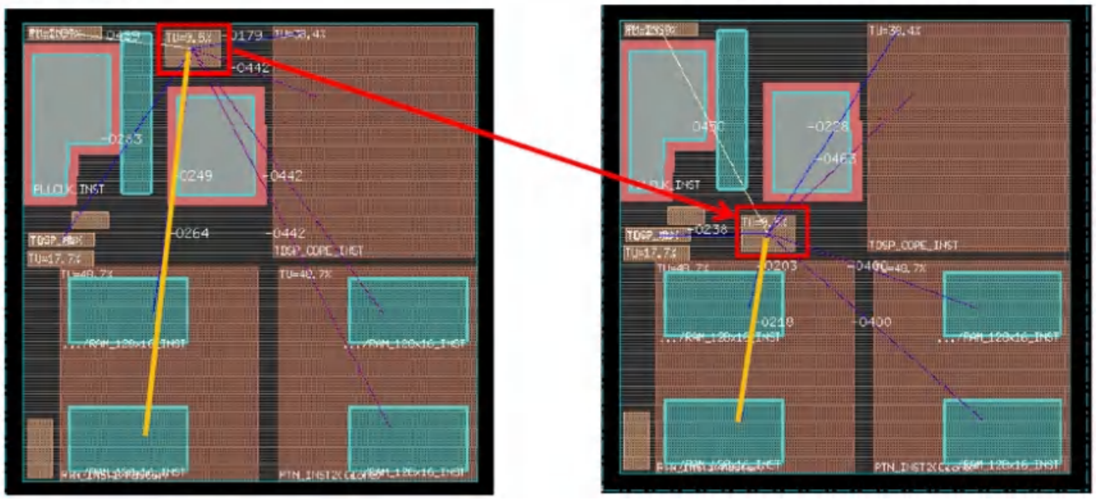
○ Display slack for:



The Floorplan View with Stack Value

Only Display Slack for the Selected Model

When the model is moved, slack value is automatically updated



Before Move

After Moving, the Slack is Updated

Prototyping Methodologies

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. 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 placement

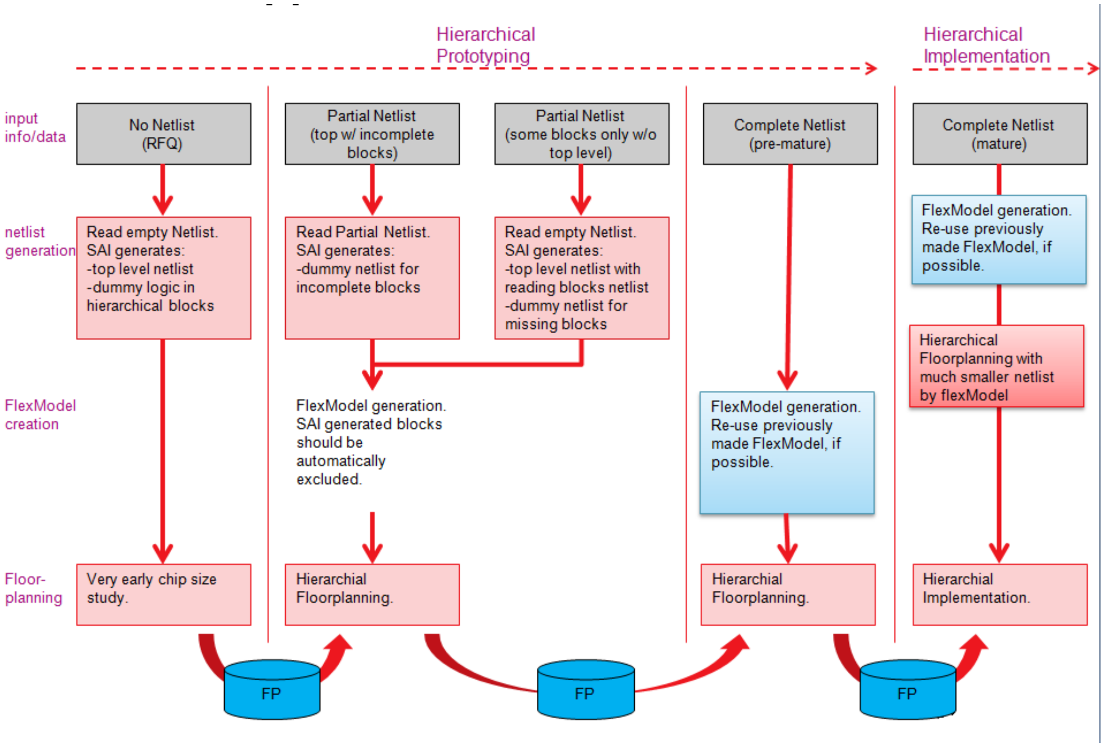
  By creating abstracts of the design to ten times fewer instances than the full netlist, you get ten times faster turnaround.
- GUI: Simplified GUI is provided to obtain abstract timing information relevant to the global context.
- Productivity: It provides a unified correct by construction flow for partitioning, 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 `earlyGlobalRoute`

The prototyping methodologies have been designed to handle growing design sizes. It allows you to do productive chip planning and concurrently handle multiple design objectives. You can use the following methodologies:

- SoC Architecture Information (SAI)
- FlexModels

Possible Application of SAI/FlexModel Flows


Hierarchical Prototyping
Hierarchical Implementation
input info/data
netlist generation
FlexModel creation
Floor-planning
No Netlist (RFQ)
Partial Netlist (top w/ incomplete blocks)
Partial Netlist (some blocks only w/o top level)
Complete Netlist (pre-mature)
Complete Netlist (mature)
Read empty Netlist. SAI generates: -top level netlist -dummy logic in hierarchical blocks
Read Partial Netlist. SAI generates: -dummy netlist for incomplete blocks
Read empty Netlist. SAI generates: -top level netlist with reading blocks netlist -dummy netlist for missing blocks
FlexModel generation. Re-use previously made FlexModel, if possible.
Hierarchical Floorplanning with much smaller netlist by flexModel
FlexModel generation. SAI generated blocks should be automatically excluded.
FlexModel generation. Re-use previously made FlexModel, if possible.
Very early chip size study.
Hierarchial Floorplanning.
Hierarchial Floorplanning.
Hierarchial Implementation.
FP
FP
FP

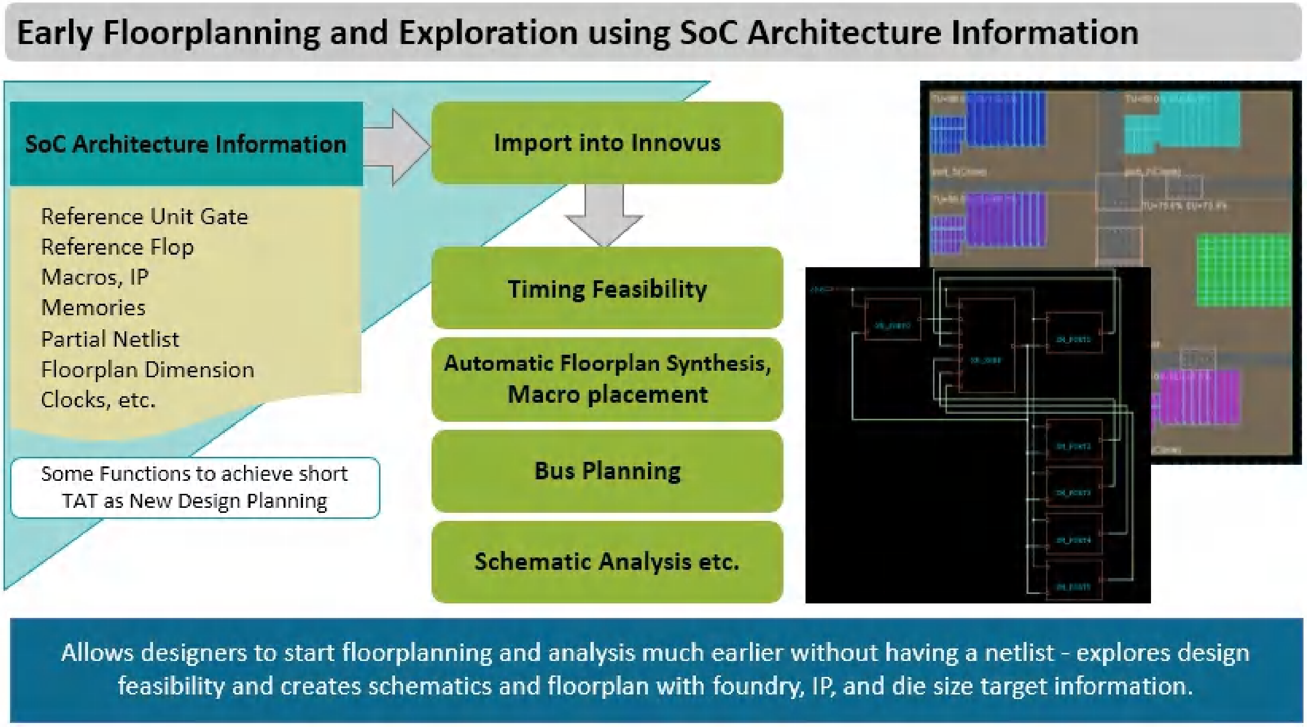
Related Information

- Using SAI Methodology for Prototyping Without Netlist
- Using SAI 2.0 Methodology for Early Prototyping and Planning
- Using FlexModel for Prototyping

Using SAI Methodology for Prototyping Without Netlist

The SoC Architecture Information (SAI) methodology allows floorplanning and analysis much earlier in the design process when a netlist is not available. SAI methodology allows exploring design feasibility and creates schematics and floor plan with foundry, IP, and die size target information. Additionally, you can turn block diagram information into a simple format, and create a netlist to enable Innovus floor planning.

The SAI architecture is shown below:



Early Floorplanning and Exploration using SoC Architecture Information

SoC Architecture Information

Reference Unit Gate
Reference Flop
Macros, IP
Memories
Partial Netlist
Floorplan Dimension
Clocks, etc.

Some Functions to achieve short TAT as New Design Planning

Import into Innovus

Timing Feasibility

Automatic Floorplan Synthesis, Macro placement

Bus Planning

Schematic Analysis etc.

Allows designers to start floorplanning and analysis much earlier without having a netlist - explores design feasibility and creates schematics and floorplan with foundry, IP, and die size target information.

The SAI file contains the following details:

Chip Architecture and IPs information is captured: Includes reference unit “gate” from the foundry specification, reference flop, macros or special IPs, memory with different sizes and ports, bus connection, soft modules (partitions), existing partial netlist, clocks, and floorplan dimension. Additionally, the software can also convert the block diagram language into a simple format.

Enabling SAI Mode

The SAI mode enables you to turn a high-level block diagram into a real netlist and timing constraint (.sdc) ready for floorplanning and timing analysis much earlier without having a netlist or just with a partial netlist.

You can use the following Innovus commands to enable or disable the SAI mode:

- `read_sai`: Enables the SAI mode that allows the use of SAI commands interactively at the command line.
- `end_sai`: Disables the SAI mode.

Enabling SAI in Batch Mode

To enable SAI in batch mode, you can use the following commands:

```
setUserDataValue init_lef_file …
setUserDataValue init_mmmc_file …
setUserDataValue init_verilog ... #partial netlist
setUserDataValue init_top_cell <my_top_cell_name>
```

```
read_sai demo.sai
```

In the above example, `read_sai` enables you to interactively use SAI commands on the prompt or in turn, use it within a script to execute SAI steps. In the batch mode, you can directly use `read_sai demo.sai`.

Note: The content of `demo.sai` (used to build design content through SAI commands) is shown in the section Sample SAI File below.

Enabling SAI in Interactive Mode

For an interactive session, you can use the `read_sai` command to invoke SAI capabilities and follow up with individual SAI commands on the prompt to build the database.

```
read_sai
```

Type in any SAI related commands you want to use. For example `create_module`, `connect`, `add_macro`, ...

You can also use the `-help` option to check the SAI command options. Once completed, you can end the SAI session with the `end_sai` command.

Notes:

- `end_sai` terminates the SAI input in both interactive or SAI script based commands.
- You can use the `timeDesign -prePlace` parameter to check if timing is ready. Then you can open GUI to floorplan modules.

Creating Automatic Netlist with SoC Architecture Information (SAI)

Based on the SAI, you can create a netlist to enable Innovus floorplanning. This provides the following features:

- Parse a partial netlist and the SAI file and to create a netlist.
- Add dummy cells to mimic the size of the define modules (RFQ).
- Add dummy flop to mimic the boundary connection from module to module.
- Add dummy memory or the real memory in order to assist sub-chip floorplan.

- Add pipe line stage registers per the specification in the connection file.
- Create SDC timing constraints file for the defined boundary connection and read into Innovus.
- Define the die size and read into Innovus.
- Create all the net groups and pipeline net groups.
- Handle master-clone, where connection model is single-driver but multiple receiver.

Partial Netlist Support

The SAI commands can be used to generate top level netlist or add new modules or modify the existing ones. Consider the following examples:

- Top level netlist is ready but consists of some incomplete modules

  Incomplete modules might have only port definition or have some logic inside. The SAI command `create_module` can be used to fill those modules with pseudo logic incrementally, as shown below:

  ```
  create_module xcmUnit1 -cell XM_PORT$i -gate_count 5000
  -memory_bit_count 8192 -util 0.4 -aspect_ratio_range {0.5 2.0}
  -flop_count 450 -flop_ref_clock clk
  ```

  You can create a net connection by using the following command:

  ```
  connect xcmUnit1/out -to xsw/in -clock clk -bus_width 256 -pipeline_stages 2
  ```

  Module ports can be created on demand when the `connect` command is used.

- There is no top level netlist but some modules for partition blocks are ready. The SAI command `create_module` will generate top level netlist and add existing modules.

Sample SAI File

```
set_sai_version 1.0
set_ref_flop DFFSRX4 ;# define a Flop
set_ref_gate CLKBUFX8 ;# define the basic “unit gate” cell
set_ref_memory mem1 -bit_size 1024 -area_per_bit 1.1 -aspect_ratio 0.75
for {set i 0} {$i<6} {incr i} {
create_module iport$i -cell XM_PORT$i -gate_count 5000 -memory_bit_count 8192 \
-util 0.4 -aspect_ratio_range {0.5 2.0} -flop_count 450 \
-flop_ref_clock clk
}
create_module xsw -cell XM_XBAR -gate_count 8000 -memory_bit_count 4096 -util 0.5\
-aspect_ratio_range {0.5 2.0}
add_macro -cell XM_PORT0 {ram_128x16A 10 rom_512x16A 20}
add_macro -cell XM_PORT1 {ram_128x16A 10 rom_512x16A 10}
add_macro -cell XM_PORT2 {ram_128x16A 10 rom_512x16A 20}
add_macro -cell XM_PORT3 {ram_128x16A 10 rom_512x16A 10}
add_macro -cell XM_PORT4 {ram_128x16A 10}
add_macro -cell XM_PORT5 {ram_128x16A 10 rom_512x16A 10}
add_macro -cell XM_XBAR -memory mem1
for {set i 0} {$i<6} {incr i} {
connect iport$i/out -to xsw/in$i -clock clk -bus_width 256 -pipeline_stages 2
connect xsw/out -to iport$i/in -clock clk -bus_width 256
}
add_clock clk -period 6.0 -waveform {0.0 3.0} -buffer CLKBUFX8
set_floorplan -aspect_ratio 1.0 -util 0.2 -side_spacing 80.0
```

Related Information

- Prototyping Methodologies
- Using SAI 2.0 Methodology for Early Prototyping and Planning
- Supported SAI Commands

Using SAI 2.0 Methodology for Early Prototyping and Planning

The Soc Architecture Information (SAI) methodology is a powerful and self-contained design planning capability. It provides an ideal design/floorplan hand-off mechanism between front-end and back-end teams. Designers can turn a high-level block diagram into a real netlist and timing constraint (.sdc) ready for floorplanning and timing analysis much earlier without having a netlist or just with a partial netlist.

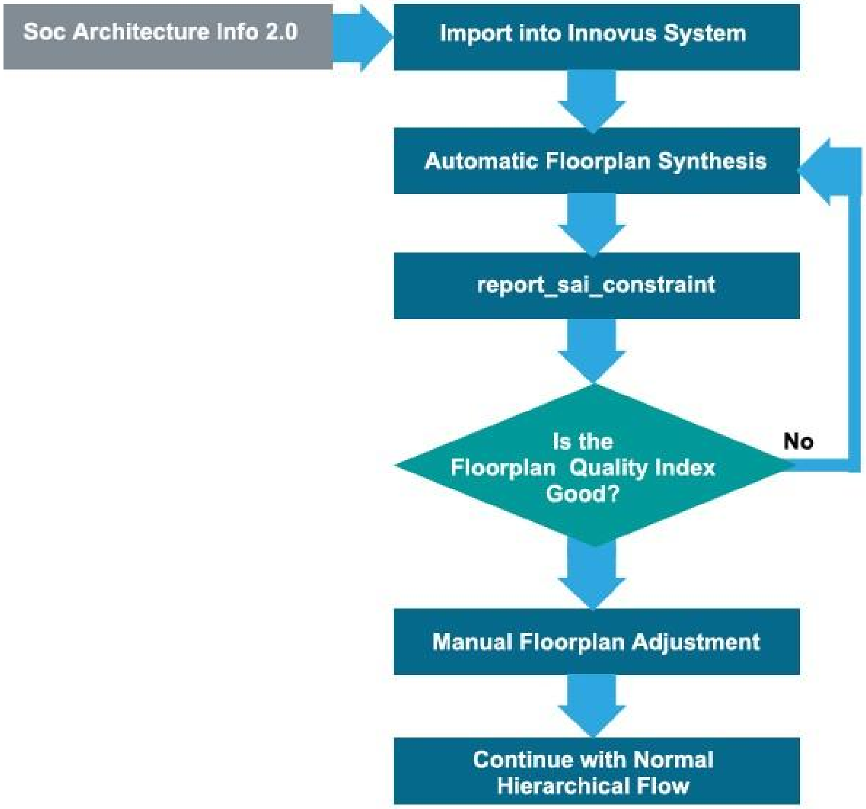
With SAI version 2.0, designers are able to specify floorplan constraints to guide module and/or macro placement, report floorplan quality index, and any constraint violations. The SAI 2.0 methodology provides you the ability to:

- Specify floorplan constraints using SAI version 2.0 that will be honored during module and/or macro placement.
- Specify cost functions to bias floorplan synthesis results.
- Identify the floorplan constraint violations on GUI, and report floorplan quality index based on the specified constraints and its scoring.

Recommended SAI 2.0 Flow

The recommended SAI 2.0 flow for early prototyping and planning involves the following steps:

1. Creating a SAI file with floorplanning constraints.
2. Reading a SAI file into Innovus to generate a SAI design.
3. Invoking Automatic Floorplan Synthesis to place modules/macros.
4. Checking the floorplan constraints and reporting the Floorplan Quality Index.
5. Once the Floorplan Quality Index is zero or close to it, continue with the normal hierarchical flow.


Soc Architecture Info 2.0
Import into Innovus System
Automatic Floorplan Synthesis
report_sai_constraint
Is the Floorplan Quality Index Good?
No
Manual Floorplan Adjustment
Continue with Normal Hierarchical Flow

Creating a SAI file with Floorplanning Constraints

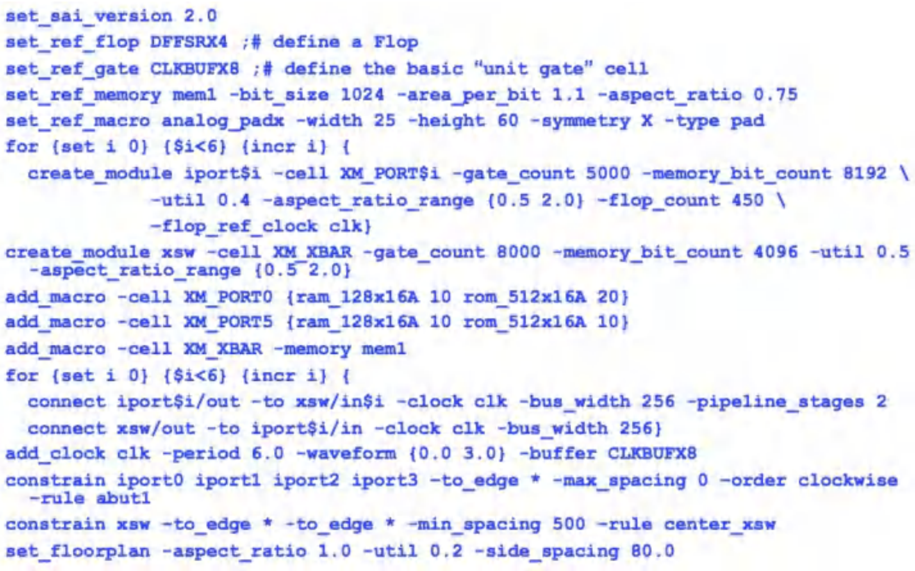
A SAI file should be created with information such as reference unit gate and flop, macros, memory with different sizes and ports, partition modules, bus connections between modules, clocks, and floorplan dimensions. In addition, SAI 2.0 can support floorplanning constraints with the `constrain` SAI command. This command allows you to set the following constraints/rules:

- Place specified macros along the boundary.
- Place modules with minimum spacing to core box.
- Place modules away from each other with minimum spacing.
- Modules/macros abutment

Furthermore, the new `-weight` option is added to the existing connect SAI command that can be used to specify the net weight for a critical net. The `proto_design` command will honor the specified net weight constraint of a net/bus such that its connected modules will be placed close together to minimize the wire length.

Sample SAI 2.0 File

The following is a sample of the SAI 2.0 file:



```
set_sai_version 2.0
set_ref_flop DFFSRX4 ;# define a Flop
set_ref_gate CLKBUFX8 ;# define the basic "unit gate" cell
set_ref_memory mem1 -bit_size 1024 -area_per_bit 1.1 -aspect_ratio 0.75
set_ref_macro analog_padx -width 25 -height 60 -symmetry X -type pad
for {set i 0} {$i<6} {incr i} {
  create_module iport$i -cell XM_PORT$i -gate_count 5000 -memory_bit_count 8192 \
             -util 0.4 -aspect_ratio_range {0.5 2.0} -flop_count 450 \
             -flop_ref_clock clk}
create_module xsw -cell XM_XBAR -gate_count 8000 -memory_bit_count 4096 -util 0.5
  -aspect_ratio_range {0.5 2.0}
add_macro -cell XM_PORT0 {ram_128x16A 10 rom_512x16A 20}
add_macro -cell XM_PORT5 {ram_128x16A 10 rom_512x16A 10}
add_macro -cell XM_XBAR -memory mem1
for {set i 0} {$i<6} {incr i} {
  connect iport$i/out -to xsw/in$i -clock clk -bus_width 256 -pipeline_stages 2
  connect xsw/out -to iport$i/in -clock clk -bus_width 256}
add_clock clk -period 6.0 -waveform {0.0 3.0} -buffer CLKBUFX8
constrain iport0 iport1 iport2 iport3 -to_edge * -max_spacing 0 -order clockwise
  -rule abut1
constrain xsw -to_edge * -to_edge * -min_spacing 500 -rule center_xsw
set_floorplan -aspect_ratio 1.0 -util 0.2 -side_spacing 80.0
```

Reading a SAI file into Innovus to Generate a SAI Design

When the SAI 2.0 file is available, the next step is to read it into Innovus using the existing `read_sai` command. After reading in the SAI file, the design is ready for floorplanning.

The `read_sai` command enables the SoC Architecture Information (SAI) mode that allows the use of SAI commands interactively at the command line.

- `-rule_only`: Use this option when you already have a SAI design and only want to read the floorplan constraints/rules.
- `-reduce_by_flexfiller`: Use this option to reduce the size of the generated SAI netlist using the FlexFiller technology that is same as the FlexModel methodology. This helps to address the capability and run-time limitations of a growing design size (10 or 100 of million instances).

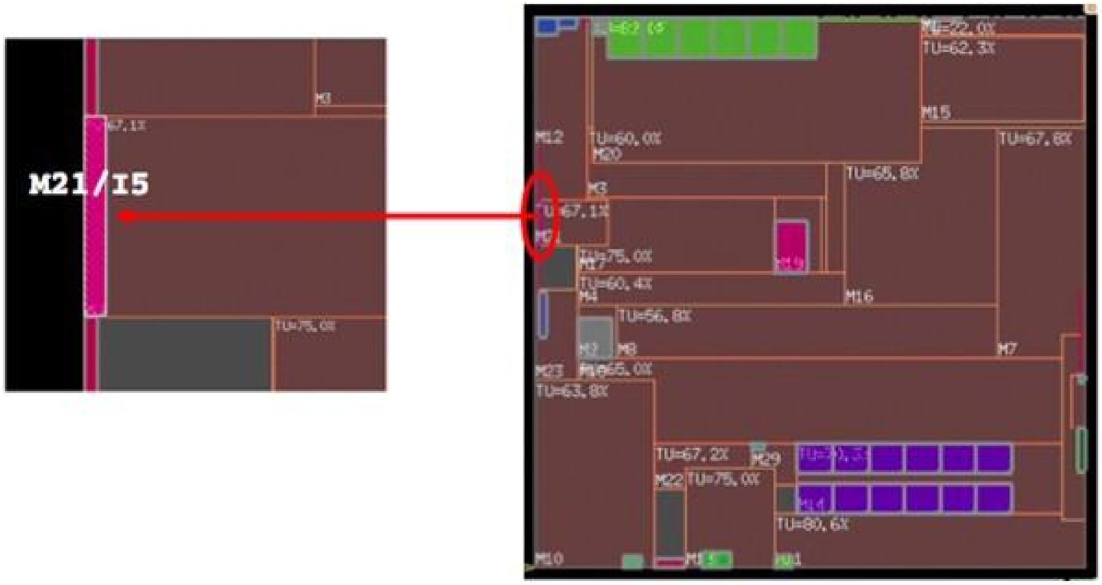
For example, the following command reads in the *example2.0.sai* file and reduces the SAI netlist while maintaining the module areas:

```
read_sai example2.0.sai -reduce_by_flexfiller
```

Generating an Initial Floorplan

The next step in the flow is to generate an initial floorplan. Given a SAI netlist and a design physical boundary, the proto_design command can generate an initial floorplan that can be used as a start point for making the final floorplan.

The following diagram displays a floorplan that was generated by `proto_design` that met SAI constraints.



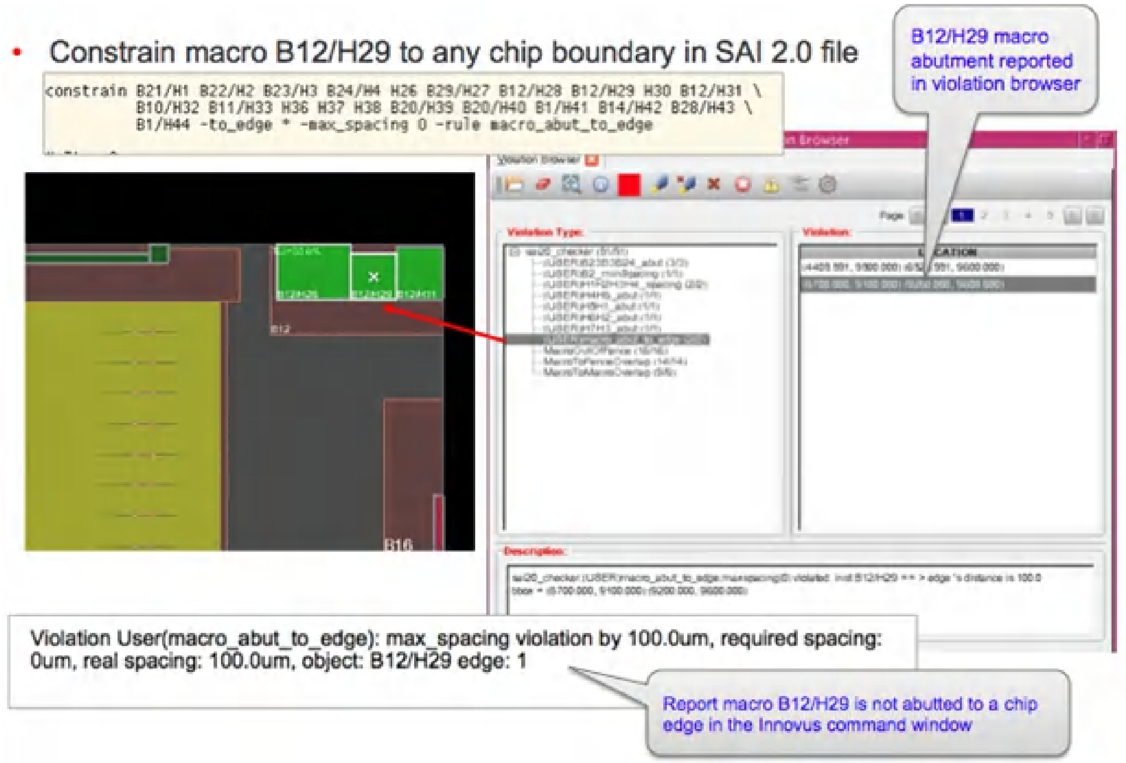
```
constrain M21/I5 -to_edge 0 -max_spacing 0 -rule I5I7_abut
```

Checking and Reporting SAI Constraints

You can use the `report_sai_constraint` command to validate a floorplan result and to check whether the specified SAI constraints are met. This enables front-end designers to easily evaluate a floorplan provided by their physical designers against the specified floorplan rules without requiring them to have much knowledge about using the Innovus System.

Example of SAI Violation Markers

In the following example, the SAI file has the rule “macro_abut_to_edge” where macro B12/H29 should be abutted to any design boundary. As a result, it is placed close to the top edge boundary with 100 micron spacing. This violates the “macro_abut_to_edge” rule; hence the `report_sai_constraint` command displays a violation message and creates the violation marker.



- Constrain macro B12/H29 to any chip boundary in SAI 2.0 file

```
constrain B21/H1 B22/H2 B23/H3 B24/H4 H26 B29/H27 B12/H28 B12/H29 H30 B12/H31 \
          B10/H32 B11/H33 H36 H37 H38 B20/H39 B20/H40 B1/H41 B14/H42 B28/H43 \
          B1/H44 -to_edge * -max_spacing 0 -rule macro_abut_to_edge
```

B12/H29 macro abutment reported in violation browser

```
Violation User(macro_abut_to_edge): max_spacing violation by 100.0um, required spacing:
0um, real spacing: 100.0um, object: B12/H29 edge: 1
```

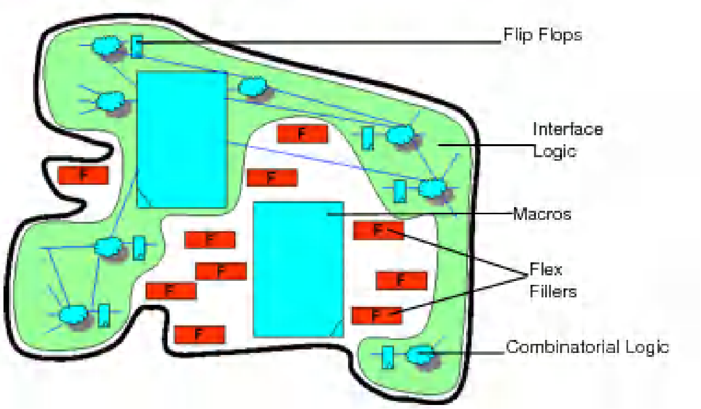
Report macro B12/H29 is not abutted to a chip edge in the Innovus command window

Related Information

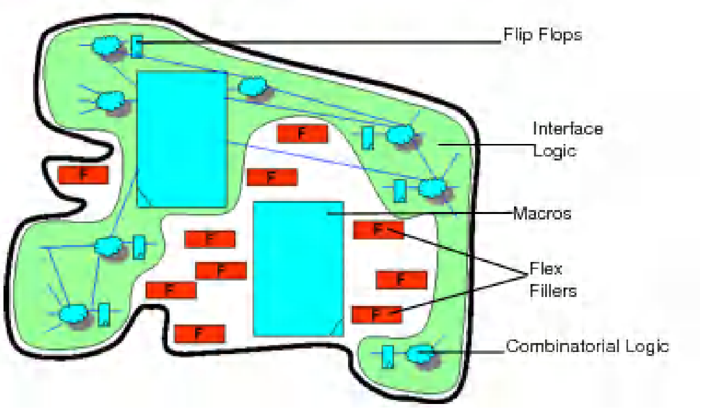
- Prototyping Methodologies
- Using SAI Methodology for Prototyping Without Netlist
- Supported SAI Commands

Using FlexModel for Prototyping

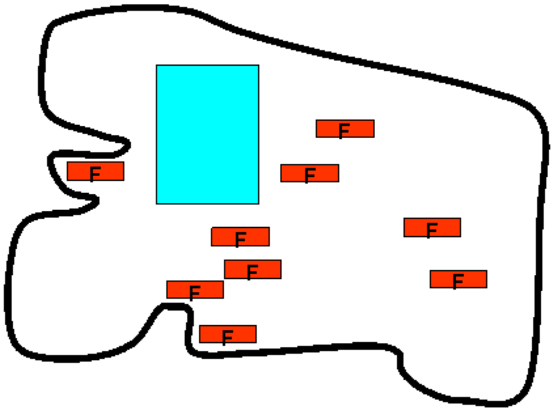
A FlexModel is a Verilog netlist that can contain macros, interface standard cells, and flex fillers. The flex fillers reserve space for the removal of internal register-to-register logic. The flex fillers have no timing model associated with them and they connect such that the model holds together during placement. So the placer will place them together in one group.



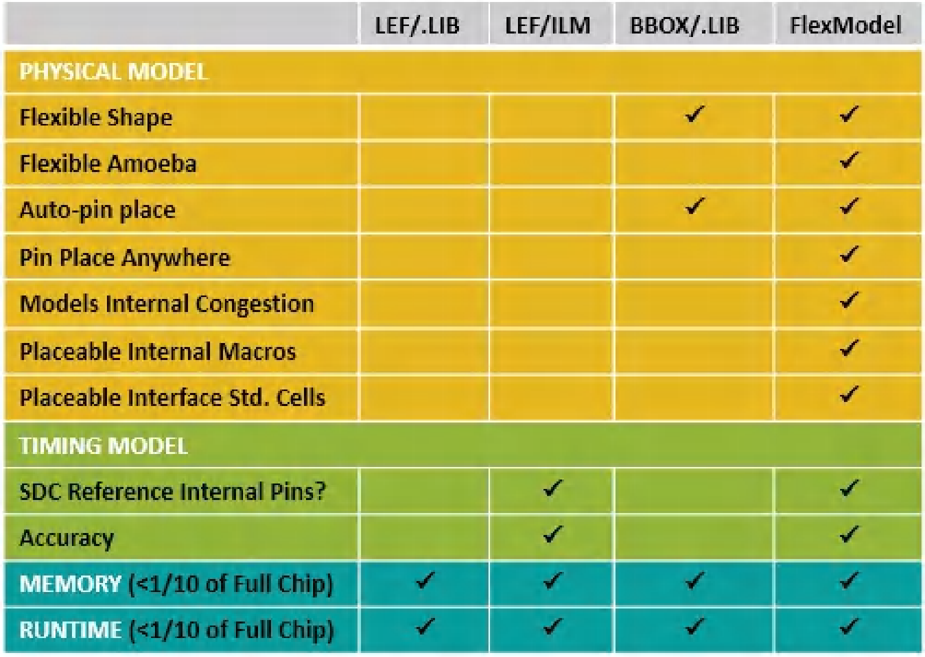
A FlexModel netlist is usually one tenth the number of instances of its full netlist. It is used during early design planning to reduce the run time and memory while accurately modeling the timing and area. A FlexModel can be created even in early stages of the design where the netlist is in its early stages. If your netlist is a full/partial netlist which has combinatorial logic, then the FlexModel will have the interface logic, the macros as well as the flex fillers. This will help in accurate timing and area estimation.



If your netlist is a skeleton netlist (that is it contains only flops at the model's interface ports), then your FlexModel's interface logic will only consist of flops (no gates). There will no combinatorial logic. This type of netlist will help find gross timing problems. In case of an empty netlist, your FlexModel will have only the macros and flex fillers. Using an empty netlist, you can estimate the design area but you cannot estimate timing.



Following is the comparison table between different supported Innovus abstractions:



	LEF/.LIB	LEF/ILM	BBOX/.LIB	FlexModel
PHYSICAL MODEL				
Flexible Shape			✓	✓
Flexible Amoeba				✓
Auto-pin place			✓	✓
Pin Place Anywhere				✓
Models Internal Congestion				✓
Placeable Internal Macros				✓
Placeable Interface Std. Cells				✓
TIMING MODEL				
SDC Reference Internal Pins?		✓		✓
Accuracy		✓		✓
MEMORY (<1/10 of Full Chip)	✓	✓	✓	✓
RUNTIME (<1/10 of Full Chip)	✓	✓	✓	✓

- Advantages of Using FlexModel Methodology
- Stages of Prototyping with FlexModels
    - Generating Models
    - Debugging Constraints and Prototype Design
    - Analyzing and Adjusting the Floorplan
    - Defining Partitions
    - Finishing and Saving Partition
- Creating Hierarchical FlexModels
    - Hierarchical FlexModel Generation Flow

Advantages of Using FlexModel Methodology

Following are the advantages of using the prototyping foundation flow:

- FlexModels reduce the instance count to 1/10th that of full netlist.
    - Allow designs upto 100 million instances
    - Significantly improve the run time and memory

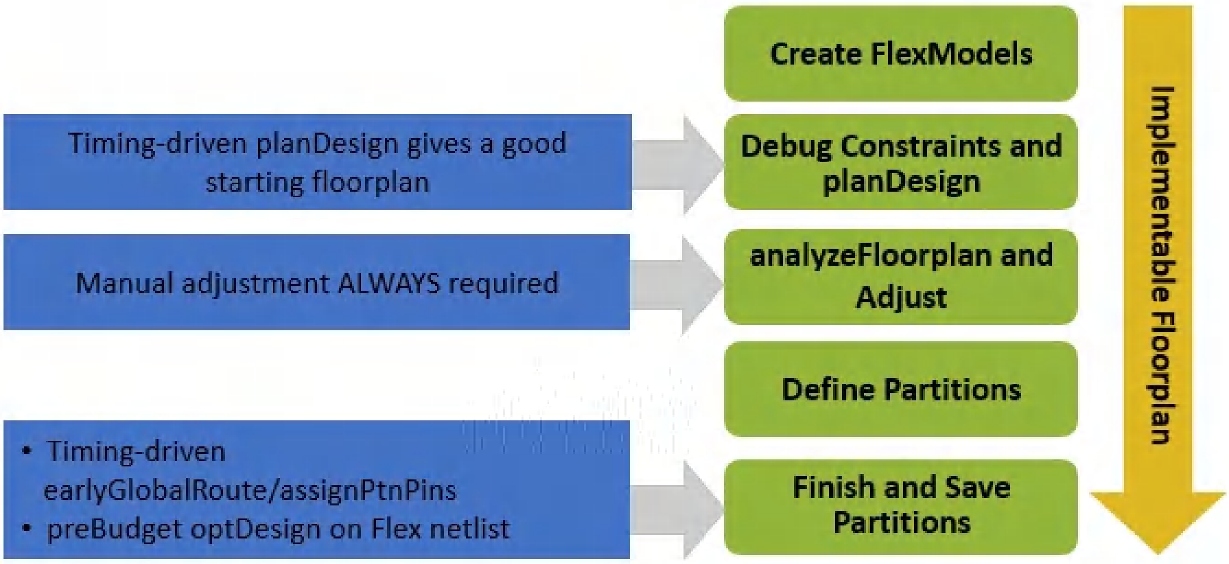
- Using FlexModels, you can perform the floorplanning of top level and partitions simultaneously. Many FlexModels per partition provide visibility into a partition's macros for congestion analysis and internal timing details.
- This flow allows accurate chip-level timing.
    - A FlexModel is created with all of its I/O paths optimized to be as fast as possible
        - Short paths are optimized
        - Eliminates re-spin of models with different budgets
        - Models are created only once and then used for many turns of the floorplan
        - No top-level optimization is needed. It is done only once during the model creation.
- This flow minimizes the partition/channel resizing.

Stages of Prototyping with FlexModels

The prototyping foundation flow runs in the following stages:

- Generating Models
- Debugging Constraints and Prototype Design
- Analyzing and Adjusting the Floorplan
- Defining Partitions
- Finishing and Saving Partition

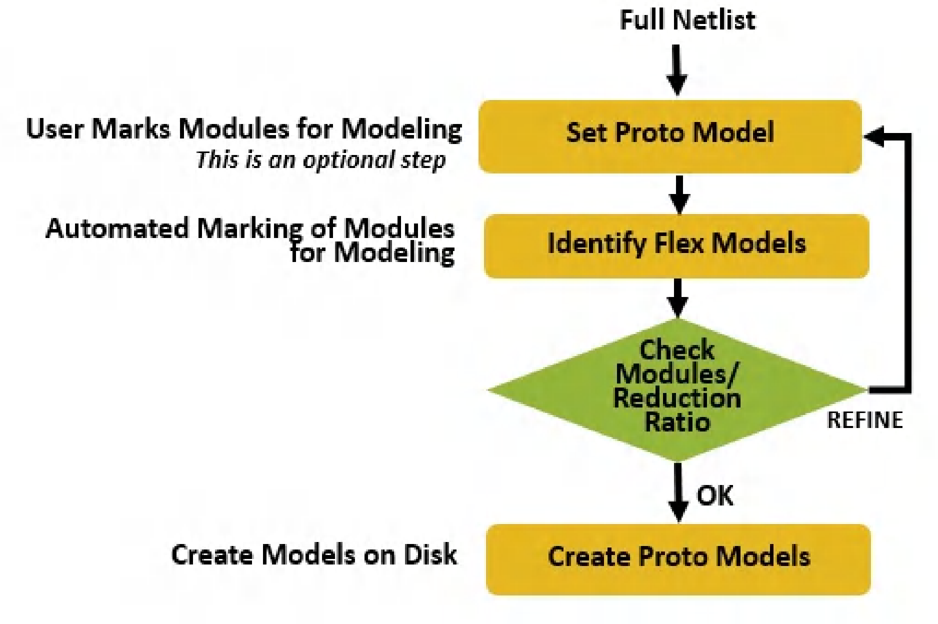
The following diagram shows the various stages of prototyping foundation flow.


Create FlexModels
Timing-driven planDesign gives a good starting floorplan
Debug Constraints and planDesign
Manual adjustment ALWAYS required
analyzeFloorplan and Adjust
Define Partitions
• Timing-driven earlyGlobalRoute/assignPtnPins
• preBudget optDesign on Flex netlist
Finish and Save Partitions
Implementable Floorplan

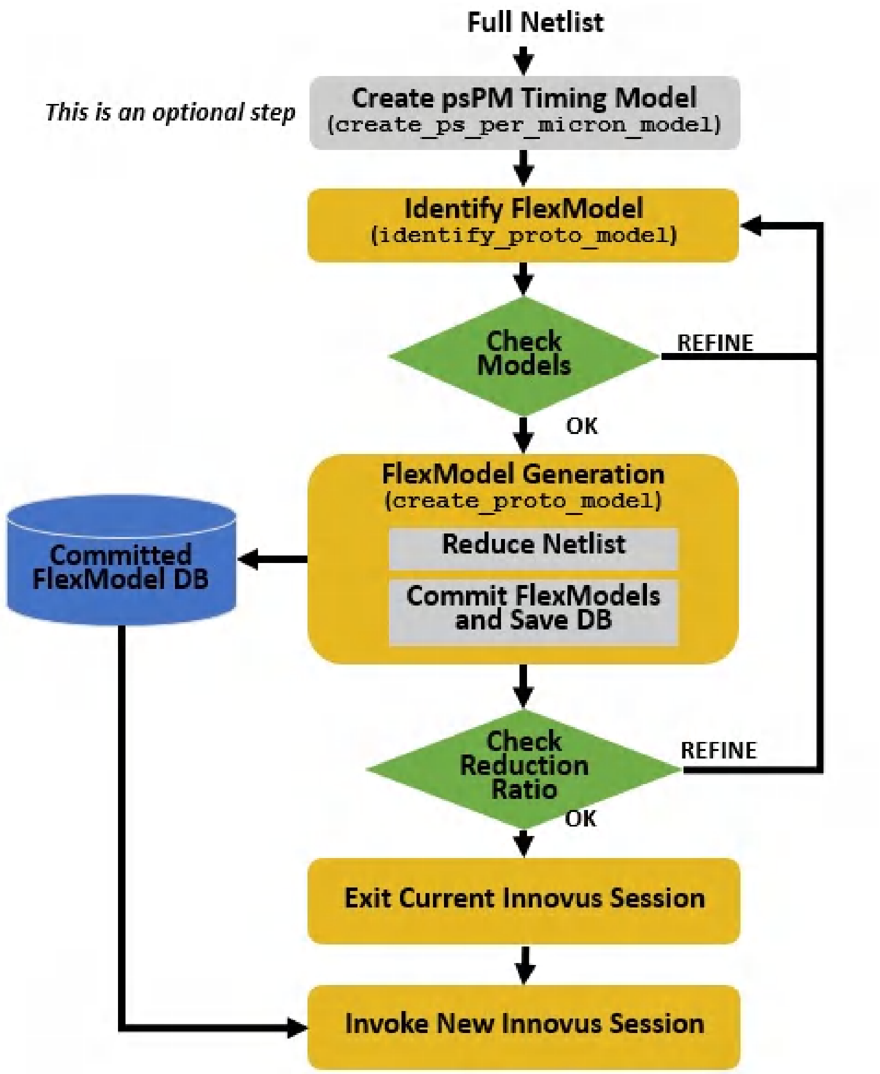
Generating Models

This is the first step in the prototyping foundation flow. In this step, you identify, create, and replace existing modules in the design with physical and timing models on disk for each identified model.

The Model Generation Flow



Note: Innovus will automatically replace original full netlist with new FlexModel information and save a committed FlexModel design on disk. Once FlexModel generation is done, you should exit the current Innovus session and restore the saved FlexModel design with a new Innovus session.



Examples

- The following example shows the use of `set_proto_model` command using which you can mark the modules for modeling.

```
set_proto_model -model prog_bus_mach -type flex_module

set_proto_model -model port_bus_mach -type flex_module

set_proto_model -model mult_32  -type flex_module

set_proto_model -model mult_32 -create_total_area 9770

get_proto_model -all


Output:

module          type        source    create_total_area

prog_bus_mach   flex_module  user
```

```
mult_32           flex_module   user      9770

port_bus_mach     flex_module   user


get_proto_model -all -tcl
{{name prog_bus_mach}    {type flex_module}  {source user}}
{{name port_bus_mach}    {type flex_module}  {source user}}
{{name mult_32}          {type flex_module}  {source user}  {create_total_area 9770}}
get_proto_model -type_match {flex_module} -name -tcl
prog_bus_mach port_bus_mach mult_32
```

- The following example shows the use of `identify_proto_model` command that is used for automated marking of modules for modeling.

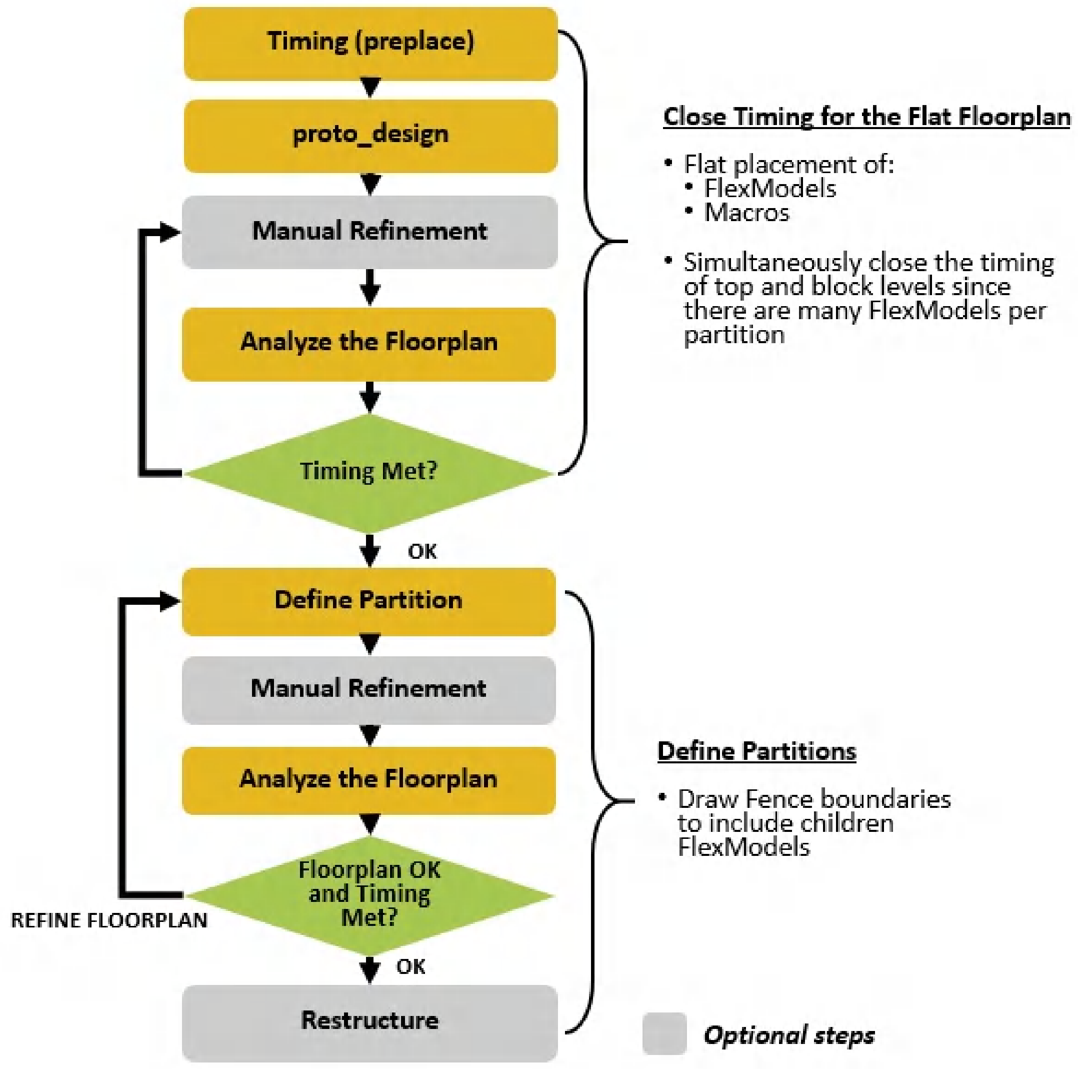
```
set_proto_mode -identify_min_insts 205
set_proto_mode -identify_max_insts 4190
identify_proto_model
```

- The following example shows the use of `create_proto_model` command that is used to create models on the disk. The committed FlexModel design is saved into DBS/model_gen.enc.dat directory

```
setMultiCpuUsage -remoteHost 2 -cpuPerRemoteHost 1
setDistributeHost -local
create_proto_model -out_dir DBS/model_gen.enc
report_proto_model -created
```

Debugging Constraints and Prototype Design

This is the second stage of prototyping foundation flow. In this stage, first you need to close timing on the flat floorplan by placing the FlexModels. Since there are many FlexModels per partition, so you will also simultaneously close the block-level placement of the FlexModels. Second, you need to define the partitions and draw fences around the already placed FlexModels.



Closing Timing for the Flat Floorplan

You can close the timing for a flat floorplan by:

- Running `timeDesign`
- Running `proto_design`

Running timeDesign

To begin closing the flat floorplan, you first need to perform `timeDesign` pre-placement. This is done to check the initial timing and constraints.

```
timeDesign -proto -preplace
```

The `timeDesign -proto` parameter allows you to call the prototype timing if FlexModels are present.

By default prototype `timeDesign` assumes:

- No detours
- No pin or wire overload
- Best routing layers used (that is the lowest RC delay)
- Runs very fast and is useful for debugging constraints

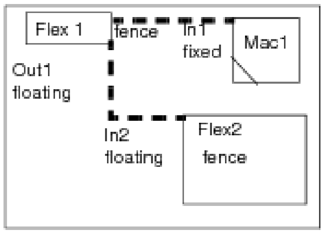
The prototyping foundation flow accepts `set_proto_mode -timing_ps_per_micron`. This option represents the amount of total delay (buffer and wire) that a technology can drive a signal at a certain distance. You also have an option to automatically create a pico-second per micron models (psPM.model) that is based on the net length and layer for timing estimation using `create_ps_per_micron_model` command

Note: This psPM.model correlates well with `optDesign`.

The `timeDesign -proto` parameter always uses the best intrinsic gate delays.

Examples

- Consider a design in which FlexModel, `Flex1`, has terminal pins `Out1` and `In1`. If you run `proto_design` without running standard cell placement, then the standard cell driving `Out1` within `Flex1` is unplaced. If the macro `In1` terminal location is known, then the distance is calculated as the Minimum Manhattan Distance between the `Flex1` fence and the `In1` terminal.

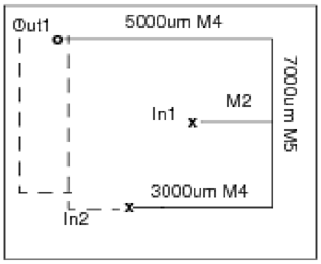


If a standard cell containing `In2` within `Flex2` standard cell is unplaced, then the distance is calculated as a the minimum distance between the two FlexModels. The delay is zero for any case where one of the terminals is of a unplaced standard cell which does not have a parent guide/region/fence (that is, neither the standard cell nor its parents have any placement information).

- If `place_design` is run, and if
  - the value of `get_proto_mode -timing_net_delay_model is best_delay_no_detour (default value)`, and
  - `set_proto_mode -timing_ps_per_micron` is set to `{Metal1:Metal5 0.25 Metal6:Metal7 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:

`Metal6delay` + `Metal7delay`

`0.14*2000 + 0.14*7000 = 1260ps`



If the value of `-timing_net_delay_model` is `use_actual_wire`, and if no routing exists, the delay calculation is same as with `best_layer_no_detour` value. Else, actual routing distances and layers are used for calculating the delay:

+ `Metal5delay`

`0.25*8000 + 0.25*7000 = 3750ps`

Running proto_design

In the next step, you need to run timing-driven `proto_design`.

To do this, set the following:

```
set_proto_design_mode -flexmodel_constraint_type fence
set_proto_design_mode -timing_aware true
```

The `-flexmodel_constraint_type fence` parameter:

- forces `proto_deisgn` to keep FlexModels from overlapping.
- forces `proto_design` to place a FlexModel's hard macros within the FlexModel fence.
- forces `place_design` to place standard cells within the FlexModel.

Analyzing and Adjusting the Floorplan

After `proto_design` some manual refinement is usually necessary. This may require the following:

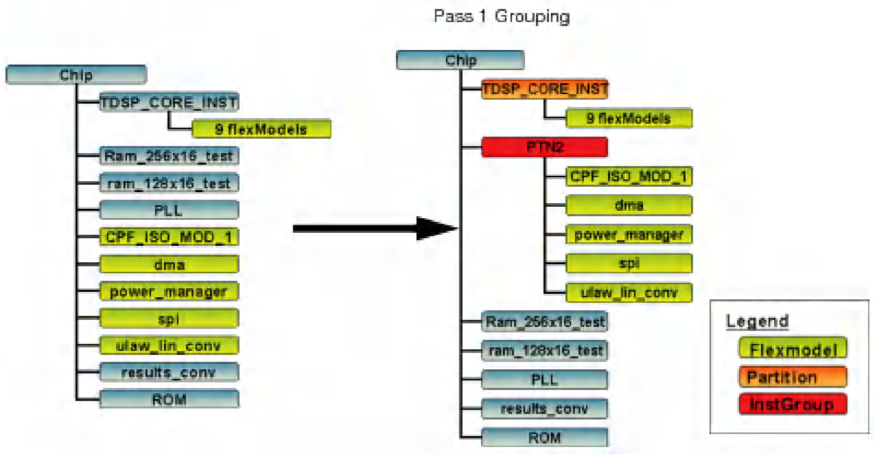
- Checking the floorplan using the `checkFPlan` command.
- Analyzing the congestion and timing for the floorplan using the following commands:
    - `place_design`
    - `earlyGlobalRoute`
    - `timeDesign -proto`
    - `load_timing_debug_report -proto`

The prototyping timing flow is fast as the calls to `earlyGlobalRoute` and `optDesign` are not required. Buffering is also not needed since long wire delay is estimated using the `set_proto_mode -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 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.

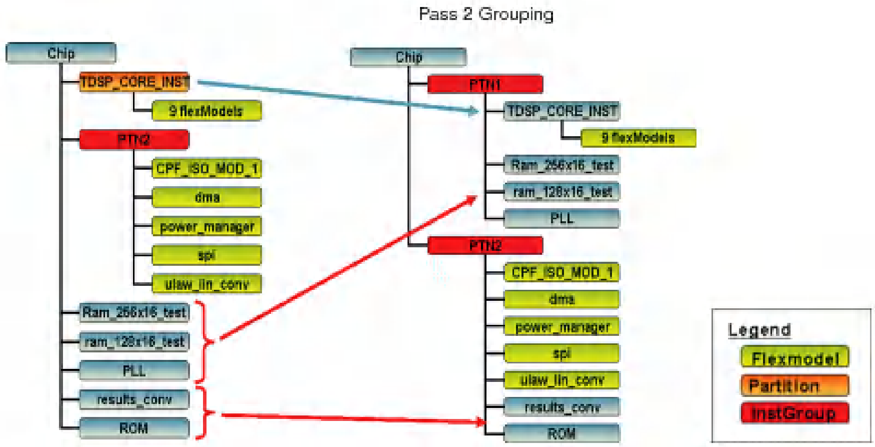
Defining Partitions

After running `proto_design` and getting a flat FlexModel placement that closes timing, you need to define partitions. In the first pass for defining a partition, it is recommended to use the modules from the original design hierarchy. For example, in the design hierarchy shown below, the `TDSP_CORE_INST` partition has nine FlexModels. There are other FlexModels also. Since there are many FlexModels at the top level of the design example below, to create a partition to hold them, an instance group can be created, which a restructuring step will change the netlist by changing that instance group into a module. So you define an instance group named `PTN2` and define all the FlexModels within that instance groups.



Once the instance group PTN2 is created, the next step is to draw its fence using the `generate_fence` command. To get a report of the FlexModels and their hierarchical names to decide if an instance group is right for your design, use the `report_proto_model -identified` command.

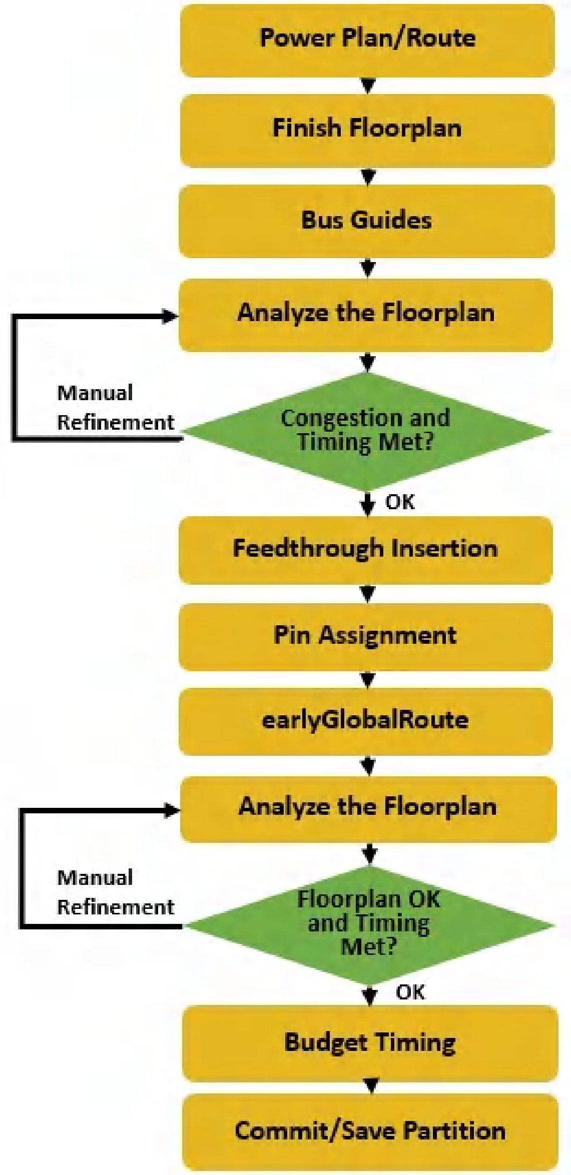
In the second pass, you define the partitions and perform manual refinement. Here you delete the `TDSP_CORE_INST` partition and create a new instance groups `PTN1`.



Once your floorplan looks good, the next step is to create a logical hierarchy for created instance groups PTN1, and PTN2 using `createLogicHierarchy` command. Once new logical modules are created, you can define them as partitions.

Finishing and Saving Partition

This is the last stage of running the prototyping foundation flow. In this stage, you analyze the floorplan to make sure that design is routable and has met timing requirements. To begin, you first create the power structure of a design and finish the floorplan so that you can perform detailed placement. You can also create the bus guides to guide the routing of critical nets. After doing this, you should run the , `earlyGlobalRoute`, `timeDesign -proto`, and `load_timing_debug_report -proto` command 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 `earlyGlobalRoute` 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

While finishing the floorplan, you can create the power structures using the `sroute` command. To prepare the floorplan for detailed placement and feedthrough insertion, you can add the block halo for your macros, add standard cell row blockages around partition fences to reserve the area for feedthrough insertion, and add partial placement blockages between macros and boundaries to handle congestion.

Analyzing the Floorplan

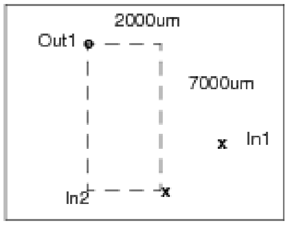
Use the following commands for analyzing the floorplan and detecting timing and congestion problems.

1. `place_design:` Places standard cells based on the global settings for placement, RC extraction, timing analysis, and early global routing. It also relieves the congestion and reorders the scan cells.
2. `earlyGlobalRoute:` Performs a quick global routing for estimating routing-related congestion and parasitic (resistance and capacitance) values.
3. `timeDesign -proto:` Runs early global route, extraction, and timing analysis, and generates detailed timing reports for FlexModel designs.
4. `load_timing_debug_report -proto:` Loads the violation report in Innovus for debugging timing results. It creates flex model categories and displays the top path of the top 8 (by default) categories.

Calculating Delay Using Timing-Driven Early Global Route

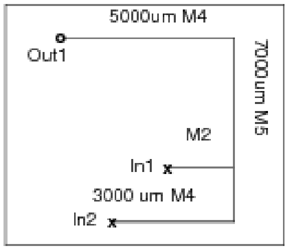
In a typical design described in the following example, you might see that the initial Early Global Route results have shown the worst negative slack of `-1ns` versus `0ns` for the timing-driven `earlyGlobalRoute` due to the following reasons:

- Before `earlyGlobalRoute`, `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)=1260ps as shown below

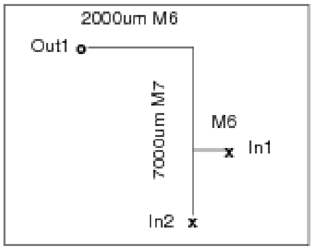


  - assuming best and non-detouring layers

- After running the initial iteration of Early Global Route, 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, Metal4 delay (0.25*8000) + Metal5 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: Metal6 delay (0.14*2000) + Metal7 delay (0.14*7000) = 1260 ps

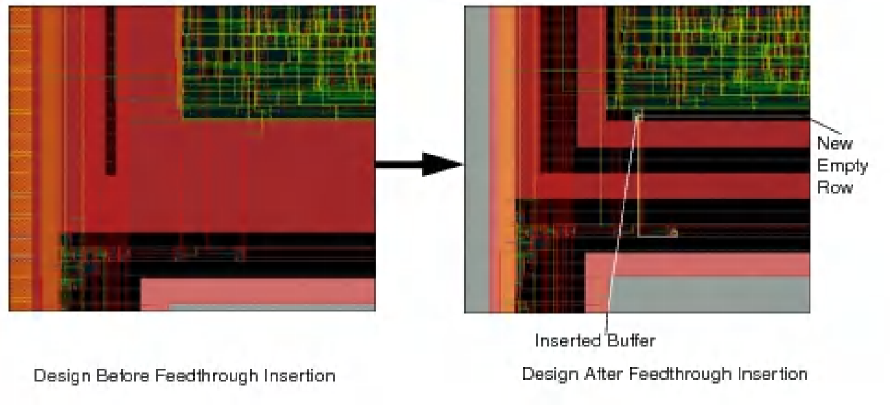


Feedthrough Insertion

After solving the congestion problems, you perform feedthrough insertion. In this step, you:

- Delete two rows of placement blockages around partition fence (Two inner row and two outer row blockages) to provide space for inserted buffers.
- Run the `insertPtnFeedthrough` command.
- Run the `earlyGlobalRoute` command to re-route old/new feedthrough nets and any existing top and/or inter partition nets that are changed by the `refinePlaceplace_detail` command during feedthrough

insertion.


New
Empty
Row
Inserted Buffer
Design Before Feedthrough Insertion
Design After Feedthrough Insertion

```
createPlaceBlockage -allPartition -innerRingBySide {2 2 2 2}  -outerRingBySide {2 2 2 2}
insertPtnFeedthrough -routeBased -netMapping ft_mapping_file -doubleBuffer
#re-route new nets created by inserted buffers:
earlyGlobalRoute
deselectAll
```

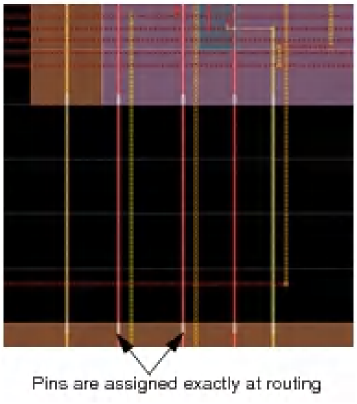
Pin Assignment

After running `earlyGlobalRoute`, 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,

```
assignPtnPin -enforceRoute
checkPinAssignment
legalizePin -keepLayer -verbose
reportUnalignedNets -ptnToPtn {unaligned} -topToPtn -rptFile ptnToPtn_unaligned.pins
reportUnalignedNets -ptnToPtn {layerMismatch} -topToPtn -rptFile ptnToPtn_layerMismatch.pins
```



Running timeDesign

In this step, do the following:

- Run final `setRouteMode` with `-earlyGlobalRoutePartitionHonorFence` by honoring partition pins (`-earlyGlobalRoutePartitionHonorPin`) to check congestion.
    - Honor partition pin constraints while routing top and inter-partition nets.
- Run `timeDesign` to get the final timing report.

For example,

```
setRouteMode -earlyGlobalRoutePartitionHonorPin list_of_ptn_cell_names
earlyGlobalRoute
saveDesign assignPin.enc
timeDesign -proto -expandedViews
load_timing_debug_report -proto
```

Budget Timing

To perform timing budgeting, you have a choice to do psPM timing budgeting or based on the `optVirtual` command. It is recommend to run psPM budgeting for a quick turn-around time.

- `Run timeDesign -proto`
- Derive timing budgets for partitions in a design.

For example,

```
timeDesign -proto; load_timing_debug_report -proto
setBudgetingMode -writeLatencyPerClock true
setBudgetingMode -enableJustifyException true
setBudgetingMode -noHoldView true
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 the `restoreDesign` command to load FlexModel netlists from the directory specified with the `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.

Replace FlexModel Netlist Back to Full Netlist

Once the top and partition block level design data bases are created, you should run:

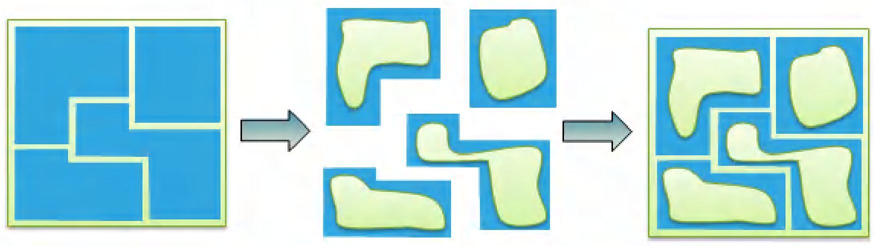
```
replace_proto_model -ptn_dir value
```

where,

`value` is the same output directory name that was specified with the `-dir` option of the `savePartition` command to replace FlexModel netlists for top and block-level designs back to full netlist.

Creating Hierarchical FlexModels

For generating FlexModels you need to load a full netlist which utilizes a lot of memory. To improve memory usage, Innovus provides the ability to create FlexModels for each partition block netlist, if the information is already available. FlexModels for each partition can be loaded back to the top level design. The following figure shows how FlexModels can be created at each partition block and loaded into the top level design to reduce memory usage:

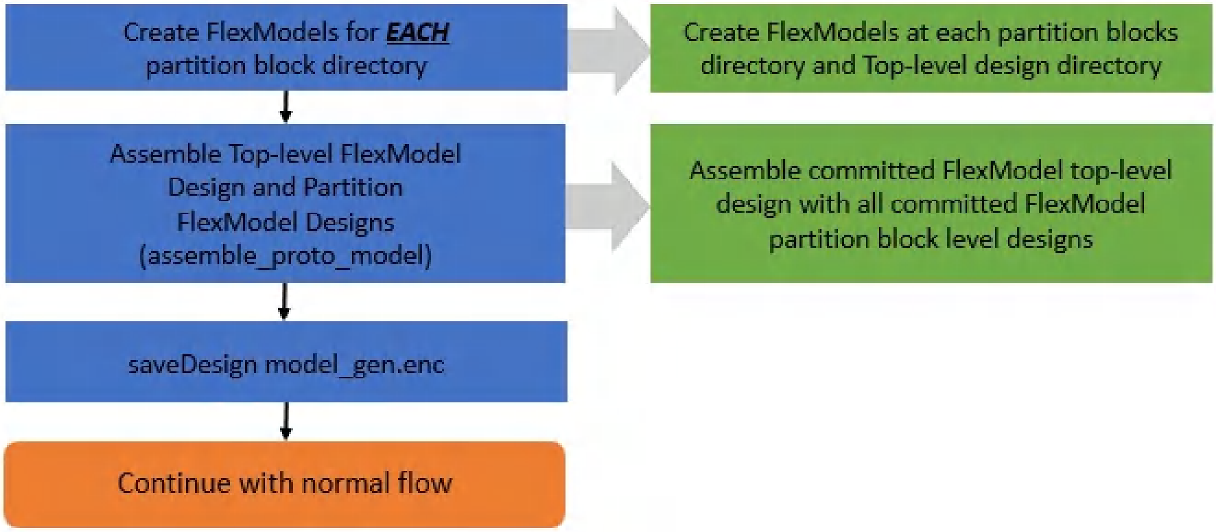


Hierarchical FlexModel Generation Flow

You can load a full chip FlexModel netlist of a design using the following flow:

1. Create FlexModels for partition block using an existing model generation flow at the partition block.
2. Create FlexModels for top-level netlist, if required.
3. Run the `assemble_proto_model` command at the current run directory where full chip FlexModel netlist is to be loaded.

The following diagram shows the Hierarchical FlexModel generation flow:


Create FlexModels for EACH partition block directory
Create FlexModels at each partition blocks directory and Top-level design directory
Assemble Top-level FlexModel Design and Partition FlexModel Designs (assemble_proto_model)
Assemble committed FlexModel top-level design with all committed FlexModel partition block level designs
saveDesign model_gen.enc
Continue with normal flow

Sample Script for Creating Models at Partition Blocks

To improve run time, you can use multiple CPUs to run block model generation in parallel:

```
set ptnName "tdsp_core"
set minInst 100

cd PTN_dir/$ptn
restoreDesign . $ptn
create_ps_per_micron_model
set_proto_mode -identify_min_inst ${minInst}
identify_proto_model
create_proto_model -out_dir fm.enc

cd ../..

exit
```

Sample Script for Assembling All Models

```
assemble_proto_model -topdir {PTN_dir/DTMF_CHIP/fm.enc.dat PTN_dir/DTMF_CHIP/proto_model} \
-blockdir { PTN_dir/tdsp_core/fm.enc.dat PTN_dir/tdsp_core/proto_model} \
-blockdir { PTN_dir/arb } \
-mmmc_file viewDefinition.tcl

saveDesign DBS/model_gen.enc
report_proto_model -created

timeDesign -preplace -proto
# Continue with normal flow
…
```

Related Information

- Prototyping Methodologies
- Prototyping Commands chapter of the *Text Command Reference*.

8

Analysis Capabilities

RC Extraction

Overview

You can perform two types of extraction in Innovus™ Implementation System (Innovus) software:

- PreRoute Extraction
- PostRoute Extraction
  Generates more accurate parasitics for cross-coupling and signal integrity analysis, timing and SI optimization flow, or obtain sign-off quality detailed parasitic extraction. The postRoute extraction engine has four variants that allow selection based on the performance versus accuracy needs. Following is a list of extraction engines in increasing order of accuracy:
  - Native Detailed
  - TQuantus
  - Integrated Quantus (IQuantus)
  - Standalone Quantus

The following table summarizes the types of extraction used during the design process.

Extraction Type		When	Quantus License Required
PreRoute		Used during optimization both before and after clock tree synthesis.	✗
PostRoute	Native Detailed	Used during postRoute and SI optimization flow in older technologies.	✗
	TQuantus	Used during postRoute optimization flow in newer technologies.	✗
	IQuantus	Used after ECO and for near signoff. **Note:** For IQuantus, Quantus XL license is required	✓

	Standalone Quantus	Used during chip assembly and timing sign-off processes.	✓

Related Information

- Pre-Requisites for RC Extraction
- Performing Extraction in Innovus
- Types of RC Extraction
- PreRoute RC Extraction
- PostRoute RC Extraction
- Setting the Scale Factors
- Generating a Capacitance Table
- Reading a Capacitance Table
- Reading a Quantus Techfile
- PreRoute Extraction Flow without Capacitance Table Data
- Correlating Native Extraction With Sign-Off Extraction
- Specifying the Scale Factors
- Distributed Processing in Extraction
- Using Advanced Virtual Metal Fill in Extraction

Pre-Requisites for RC Extraction

RC extraction refers to the extraction of resistance (R) and capacitance (C) values in a design to generate the RC database. The inputs for perfroming extraction are detailed below.

The following files are required for extraction:

- Capacitance table - Used by preRoute extraction engine and native detailed extraction engines for above 32nm technology. For more information, see Generating a Capacitance Table.
- Quantus technology file(s) - Used by preRoute extraction engine for 32nm and below technology, TQuantus/IQuantus, and Standalone Quantus engines.

Before running extraction, provide the RC scale factor values in the Edit RC Corner form for the MMMC design environment. Scale factor values provide better correlation between the Innovus estimated parasitics and the signoff extraction results by multiplying the extracted resistance and capacitance. For example, a capacitance scale factor of 1.1 increases the extracted values by ten percent.

Use of scale factors is recommended for preRoute and Native Detailed extraction engines, and is optional for TQuantus. The IQuantus scale factor also allows for optional fine tuning for optimal correlation with third party signoff extractor. In addition, all scale factors allow you to use simple scaling for non-typical corners as an alternative for the recommended use of corner-specific capacitance tables (captables) and Quantus technology files.

Note: Scaling for engine with `effortLevel` set to `signoff` is not supported.

Results

- Binary RC Database (RCDB) is created. It contains parasitics for each process corner.
- An ASCII SPEF file can be generated from the parasitics database for the specified process corner, if required.

Specifying Temporary File Locations

You can specify a temporary file location for TQuantus and IQuantus 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.

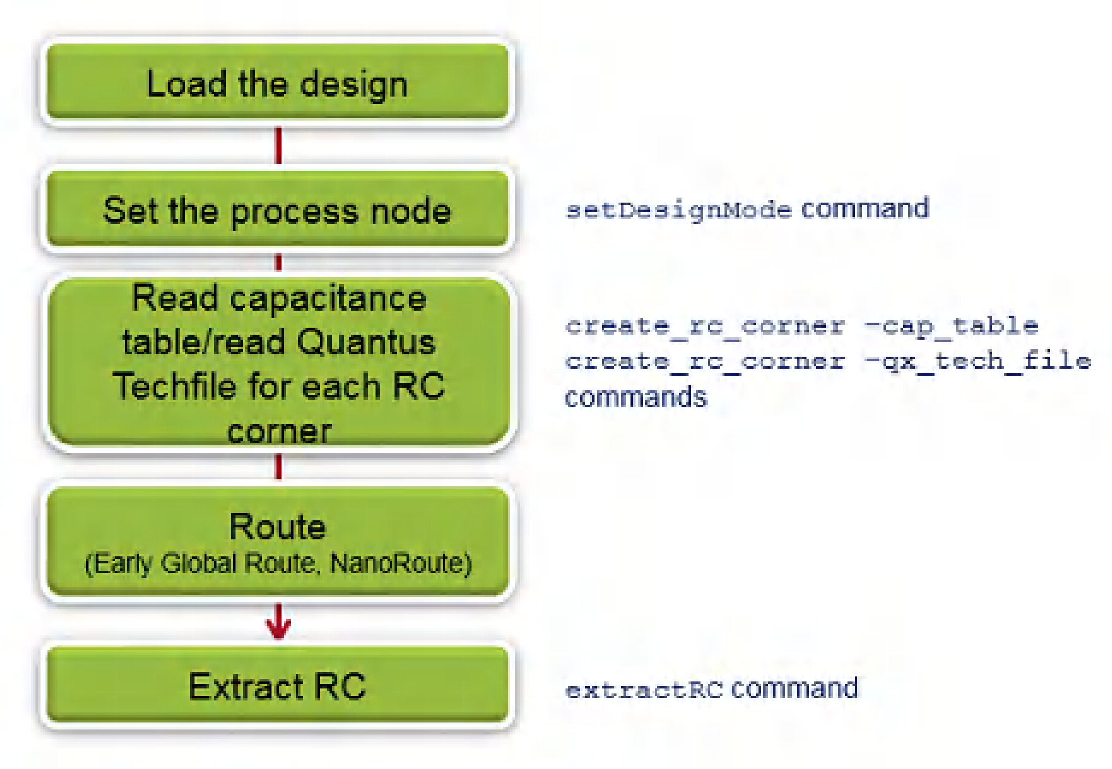
Note: For IQuantus /TQuantus extraction in the distributed processing mode using different machines, do not store the cache or temporary data to the `/tmp` directory. The temporary data must be visible on all machines used for distributed processing. Either use the current directory, or specify a directory using the `TMPDIR` or `FE_TMPDIR` environment variable.

Related Information

- RC Extraction

Performing Extraction in Innovus

The extraction flow in Innovus is shown below.


Load the design
Set the process node
Read capacitance table/read Quantus Techfile for each RC corner
Route
(Early Global Route, NanoRoute)
Extract RC

`setDesignMode` command

`create_rc_corner -cap_table`
`create_rc_corner -qx_tech_file`
commands

`extractRC` command

To extract resistance and capacitance data in innovus and generate the RC database, perform the following steps.

1. Load the design.
2. Specify the process technology value, to automatically set the technology node dependent parameters, by using the following syntax: `setDesignMode -process` *`processnode`*
   Note: For maximum accuracy and optimal automatic threshold setting, use the `-process` *`processnode`* parameter of the `setDesignMode` command prior to running `extractRC`.
3. Read in the captable file(s) for extracting interconnect capacitance values with preRoute and postRoute `-effortLevel` engine choices for above 32nm technology. For more information, see Correlating Native Extraction With Sign-Off Extraction.
   When using preRoute extraction (for 32nm and below), TQuantus, IQuantus, or Standalone Quantus extraction, read in the Quantus Techfile that contains the interconnect models used

by these engine choices. For more information, see Reading a Quantus Techfile.
Note: If the design is half node and the layout scale value is not specified in the captable, you can specify it using the `setShrinkFactor` command.

4. Use the `setExtractRCMode` command to set up extraction parameters and to specify the extraction engine to be used for subsequent extraction.
5. Run `extractRC` command to perform extraction. You can also use the Specify RC Extraction Mode and Extract RC GUI forms to perform extraction. An RCDB is created. This database contains extracted parasitics.
6. Optionally, use the `rcOut` command to retrieve SPEF files (corresponding to each active RC corner) with the parasitics results. Timing and SI analysis commands use the RC database directly.
7. Use the `spefIn` command to load the existing SPEF files with the parasitic data.
8. For hierarchical designs, use the `read_parasitics` command for reading both the top-level and block-level parasitic data in either the RCDB format or the SPEF format, or a mix of both. This command generates a flat-level RCDB for the complete design after performing hierarchical stitching of the RC data.

Related Information

- RC Extraction

Types of RC Extraction

RC extraction refers to the extraction of parasitics in a design to generate the RC database. This RC parasitic database can be output in specified formats. The parasitic database that is generated is required for timing analysis and signal integrity (SI) anaysis.

Depending upon when extraction is performed in the flow, the extraction is classified into two types.

- PreRoute RC Extraction
  Provides quick parasitic extraction for design prototyping.
- PostRoute RC 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.

PreRoute RC Extraction

RC extraction is classified into two types, preRoute and postRoute, pepending upon when extraction is performed in the flow.

PreRoute Extraction provides a quick parasitic extraction for design prototyping. It is used during optimization, both before and after clock tree synthesis (CTS). It is only used for static timing analysis (STA).

This type of extraction does not require a Quantus license.

PostRoute RC Extraction

PostRoute Extraction is a detailed parasitic extraction. It generates more accurate parasitics than preRoute extraction for cross-coupling and signal integrity (SI) analysis, and Timing and SI optimization flows. It is also used to obtain sign-off quality detailed parasitic extraction. The postRoute extraction engine has four variants that allow selection based on the performance versus accuracy needs.

- Native Detailed - Used during postRoute and SI optimization flow in older technologies. It does not require a Quantus license.
- TQuantus - Used during postRoute optimization flow in newer technologies. It does not require a Quantus license.
- Integrated Quantus (IQuantus) - Used after ECO and for near signoff. It requires a Quantus XL license.
- Standalone Quantus - Used during chip assembly and timing sign-off processes. It requires a Quantus license.

Native Detailed

In the native detailed mode:

- RC values that are generated can be used for both standard timing analysis (including cross-coupling) and signal integrity (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 captable is provided during design import.

To invoke native detailed extraction, use the following command:

```
setExtractRCMode -engine postRoute -effortLevel low
```

TQuantus Extraction

The TQuantus extraction engine is an advanced extraction engine that is enabled by default for postRoute effort-level medium extraction.

```
setExtractRCMode -engine postRoute -effortLevel medium
```

The TQuantus engine is tightly integrated with NanoRoute and drives track assignment-based timing and SI optimization/postRoute optimization, and timing-driven routing. The use model is detailed below.

The TQuantus flow is enabled by default. In this mode, the software instructs `optDesign`/`timeDesign -postRoute` to use the TQuantus model file for optimization. You can specify a file name for the TQuantus model file by using the `-tQuantusModelFile` parameter of the `setExtractRCMode` command. If this file is not specified, it is generated automatically by the software.

The TQuantus extraction engine supports the following `setExtractRCMode` parameters:

- `-capFilterMode relAndCoup`

  Note:
  It does not support the `relOnly` and `relOrCoup` arguments of the `-capFilterMode` parameter as they are for old nodes only.
- -relative_c_th
- -total_c_th
- -coupling_c_th
- -extraCmdFile
- -coupled

The TQuantus extraction engine does not support the following parameters of the `setExtractRCMode` command:

- `-hardBlockObs`
- `-lefTechFileMap`

TQuantus versus IQuantus

TQuantus and IQuantus are more accurate as compared to native detailed extraction. These are based on the same technology as the Standalone Cadence signoff extraction tool, Quantus. The TQuantus extraction engine is recommended for the implementation phase because it is optimized for performance with a small tradeoff for accuracy. The IQuantus extraction engine is recommended for the ECO flow, as it has near-signoff accuracy.

In IQuantus , there are two modes for RC extraction:

- **Full-chip extraction**
  Forces full extraction on the complete design.
- **Incremental extraction**
  Enables incremental extraction on the design. The software recognizes the changes that have taken place since the last extraction. If the changes are less than the pre-defined threshold, the software incrementally extracts the changed regions of the design and then stitches the new data with the previously extracted parasitic data.

By default, the incremental mode is enabled. In TQuantus there is no incremental extraction. This is not needed due to the high speed of this extraction engine.

Note: The performance and accuracy of TQuantus falls between that of native detailed extraction and IQuantus. IQuantus provides superior accuracy compared to TQuantus. Both IQuantus and TQuantus support distributed processing. TQuantus does not require a Quantus license while IQuantus requires a Quantus XL license.

IQuantus Extraction

To invoke IQuantus, use the following command:

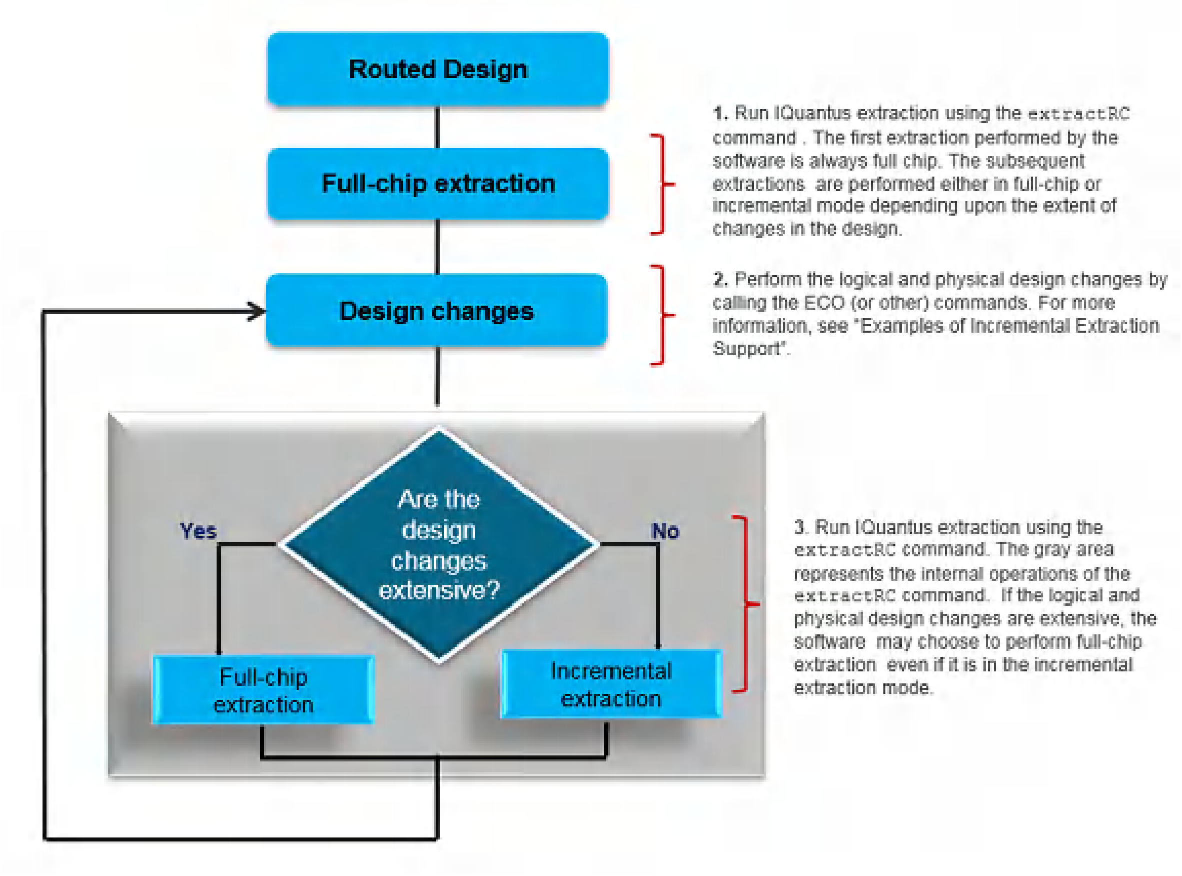
```
setExtractRCMode -engine postRoute -effortLevel  high
```

To invoke TQuantus, use the following command:

```
setExtractRCMode -engine postRoute -effortLevel medium
```

The following figure shows the IQuantus extraction flow.

IQuantus Extraction Flow



1. Run IQuantus extraction using the `extractRC` command . The first extraction performed by the software is always full chip. The subsequent extractions are performed either in full-chip or incremental mode depending upon the extent of changes in the design.

2. Perform the logical and physical design changes by calling the ECO (or other) commands. For more information, see "Examples of Incremental Extraction Support".

3. Run IQuantus extraction using the `extractRC` command. The gray area represents the internal operations of the `extractRC` command. If the logical and physical design changes are extensive, the software may choose to perform full-chip extraction even if it is in the incremental extraction mode.

Examples of Incremental Extraction Support

Following are some of the examples for incremental extraction flow in Innovus:

- **Example of Interactive ECO Commands**:

  ```
  setExtractRCMode  -engine postRoute -effortLevel high
  extractRC
  ecoAddRepeater  -net netName - Name
  ...
  ecoPlace
  ecoRoute
  extractRC
  ```

- **Example of Wire Edit Commands**:

  ```
  setExtractRCMode -engine postRoute -effortLevel high
  extractRC
  editSelect  -area areaValue -net netName
  ```

```
editMove  y distance
...
extractRC
```

- **Example of Timing Optimization Command**: In the postRoute optimization cycle, extraction is called multiple times. Depending upon the type of changes, the extraction called may be either full-chip or incremental.

```
setExtractRCMode -engine postRoute -effortLevel high -incremental true
optDesign -postRoute
```

Note: Incremental extraction is not supported for all designs or for all types of changes in the design. For example, incremental extraction will not take place if the design contains 45-degree wire(s). In such cases, the software automatically goes info the full-chip extraction mode after printing an appropriate message citing the reason for selecting full-chip extraction.

Standalone Quantus for Signoff Extraction

Running Standalone Quantus from within Innovus

Standalone Quantus extraction is accessible through Innovus for generating detailed signoff quality parasitics. You can perform signoff extraction after the detailed routing phase.

Note: Standalone Quantus extraction requires a separate license and requires installation of the EXT software package.

Inputs for Quantus Sign-Off Extraction

When you perform signoff extraction through the Innovus interface, either a routed DEF is created or an OA database is saved automatically. The Quantus technology file is required before you can start signoff extraction:

- **Quantus technology file**: Contains the process-dependent model files and manufacturing effects used by the extractor to calculate resistance and capacitance.
- **Quantus command file**: Providing a Quantus command file is optional. It contains commands and variables that define the extraction environment (technology filename, library name, and so on), specifies which net(s) to extract and how to extract them, controls the resistance and capacitance extraction, and specifies the extraction outputs. This is applicable only for partial and custom command files.

Use Model

You can run Quantus extraction by using the `extractRC` command after setting the Quantus extraction mode using the `-effortLevel_effort_level signoff` option of the `setExtractRCMode` command.

By default, Standalone Quantus extraction is a Design Exchange Format (DEF)-based flow. However, you can specify the OpenAccess (OA)-based flow for invoking Quantus extraction. For this, set the `setExtractRCMode -useQRCOAInterface_use_qrc_oa_interface` option to `true`.

Note: For TSV designs, you are required to provide the name of the layer map file for IQuantus and Standalone Quantus.

Running Standalone Quantus outside Innovus

You can invoke standalone Quantus from outside Innovus. For this, you need to generate the Quantus command file or the Common Command language (CCL) file using the `write_extraction_spec` command. Before running this command, you need to load the design, set up the MMMC environment, and specify the CCL layer map file, if required. You can specify a directory path where technology library file, LEF list file (for DEF-based flow), and DEF or OA design will be stored. However, if no directory path is specified, then a directory will be created in the current directory. You can also specify whether you want to save the design data or not by using the `-no_design_data` parameter of this command.

Note: For OA-based flow, ensure that the `setExtractRCMode -useQRCOAInterface` parameter is set to `true`.

Related Information

- Pre-Requisites for RC Extraction
- Performing Extraction in Innovus

Setting the Scale Factors

To better correlate native extraction results, both in preRoute and postRoute stages with sign-off extraction, Innovus allows you to set scale factors for total capacitance, cross-coupling capacitance, and resistance. As the accuracy of the different engines varies, engine-dependent scale factors can be entered when using different extraction engines in the flow.

To generate scale factors, you can also use the `generateRCFactor` command.

Related Information

- Specifying the Scale Factors
- Correlating Native Extraction With Sign-Off Extraction

Generating a Capacitance Table

When the Quantus technology files are unavailable, a capacitance table is needed for extraction by the preRoute and postRoute effort level `low` extraction engines for above 32nm technology nodes. The table contains three parts:

- **Header**: Contains process information and manufacturing effects. The information in the header is used for resistance extraction and to correct the extracted values for the specified manufacturing effects. For preRoute extraction, the header section also provides default values for scale factors.
- **Basic Captable Part**: Contains the coefficients used by the preRoute capacitance extraction engine. This part contains the area, fringe, and lateral coupling capacitance coefficients organized per conductor layer for wires with different width and spacing. The basic captable 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 captable part because the coefficients are generated on more complex profiles, which account for geometries on multiple layers. The extended captable 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.

For higher than 32nm technology nodes, either Quantus technology files or capacitance tables need to be specified. If neither of these is provided, Innovus generates a basic capacitance table using the default process parameters and heuristic equations for calculation. However, this severely compromises accuracy.

For 32nm and below technology nodes, Quantus technology files must always be provided.

- Inputs for Generating a Capacitance Table
- Capacitance Table Generation Flow
- Capacitance Table Examples
- Generating Capacitance Table with Specified Scale Factors

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 captable part only.

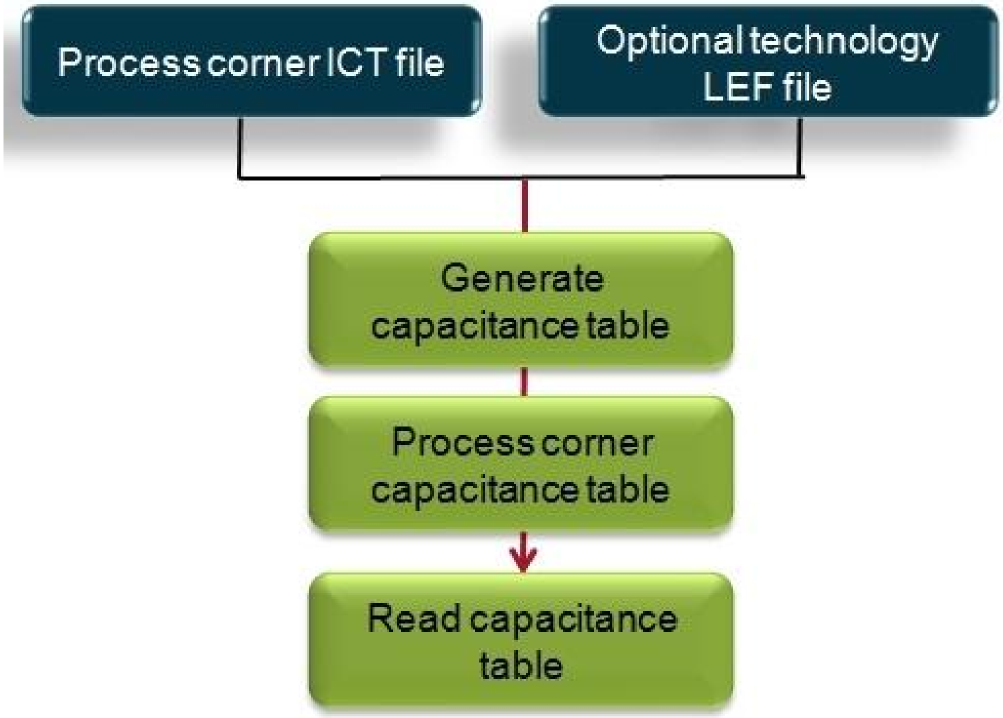
Fabrication process information in the ICT file can consist of the following:

- 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 Creating the ICT File.

Capacitance Table Generation Flow

The following figure shows the flow for generating a process corner-specific capacitance table:


Process corner ICT file
Optional technology LEF file
Generate capacitance table
Process corner capacitance table
Read 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:

- Generate an ICT file for each process corner. For an example of an ICT file, see the chapter titled, Creating the ICT File.
- Generate a capacitance table for each ICT file. Use the `generateCapTbl` command within Innovus or the `generateCapTbl` standalone executable.

Note: Generating a capacitance table is CPU-intensive and can take several hours to run for newer technologies. The `generateCapTbl` standalone executable which can be found in the `bin` directory of your Innovus hierarchy runs independent of Innovus. It has the same syntax as the `generateCapTbl` command.

Capacitance Table Examples

Example 1: Capacitance Table

```
PROCESS_VARIATION ...

LAYER M1

MinWidth 0.09000

MinSpace 0.09000

# Height 0.54000

Thickness 0.20260

TopWidth 0.12100

BottomWidth 0.09300

WidthDev 0.00000

ThermalC1 2.65000e-03

ThermalC2 -2.64100e-07

WireEdgeEnlargement

WeeWidths 0.107 0.127 0.152 0.197 0.287 0.377 0.467 0.557 0.647 0.917 1.017 2.017 3.017
4.517 7.517 12.017

WeeSpacings 0.073 0.093 0.118 0.163 0.253 0.343 0.433 0.523 0.613 0.883 0.983 1.483
1.983 2.483 2.983 4.983

WeeAdjustments -0.001 -0.003 -0.003 -0.009 -0.009 -0.01 -0.01 -0.011 -0.016 -0.018 -
0.018 -0.019 -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 0.0216 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
0.0216 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
0.0215 0.0214 0.0213 0.0213


WireThicknessRatio

WtrMinThicknessRatio 0.914688

WtrMaxThicknessRatio 1.03875

WtrTileWidth 100 100

WtrStepperWindowWidth 50 50

WtrMaxSpacing 5

WtrWidthRanges 0.09 0.18 12

WtrDensityPolynomialOrder 4

WtrWidthPolynomialOrder 4

WtrPolynomialCoefficients

{

0 7180.07 -3744.08 625.29 -33.3498

0 -8418.26 4361.73 -720.647 37.5707

0 3011.8 -1560.96 257.063 -13.1704

0 -369.306 193.11 -31.9649 1.58144

0 -2.73935 -0.912962 0.476445 0.0054143

}
```

```
{
-0.00355249 0.0134349 -0.377017 2.18449 -1.14899
0.0086884 0.0669357 0.00360917 -3.62062 1.91715
-0.0108639 -0.126014 0.84772 1.42329 -0.92238
0.00784181 0.0469798 -0.580639 0.11264 0.0642999
-0.00204508 -0.00330229 0.125923 -0.179971 0.100731
}
END
LAYER M2
...
...
...
VIA VIA1
TopLayer M2
BottomLayer M1
ThermalC1 7.81500e-04
ThermalC2 -2.57400e-06
Resistance 3.00000
END
...
END_PROCESS_VARIATION
BASIC_CAP_TABLE ...
M1
width(um) space(um) Ctot(Ff/um) Cc(Ff/um) Carea(Ff/um) Cfrg(Ff/um)
0.090 0.072 0.3549 0.1313 0.0502 0.0123
0.090 0.090 0.3115 0.1248 0.0502 0.0147
0.090 0.270 0.1803 0.0237 0.0502 0.0418
0.090 0.450 0.1728 0.0074 0.0502 0.0541
```

```
0.090 0.630 0.1721 0.0023 0.0502 0.0587
0.090 0.810 0.1720 0.0007 0.0502 0.0602
0.090 0.990 0.1720 0.0002 0.0502 0.0607
0.090 1.170 0.1720 0.0001 0.0502 0.0608
0.270 0.072 0.3797 0.1114 0.1267 0.0151
...
9.000 0.990 4.2967 0.0002 4.2226 0.0369
9.000 1.170 4.2967 0.0001 4.2226 0.0370
M2
width(um) space(um) Ctot(Ff/um) Cc(Ff/um) Carea(Ff/um) Cfrg(Ff/um)
0.100 0.080 0.3330 0.1275 0.0458 0.0161
0.100 0.100 0.2659 0.0919 0.0458 0.0182
'''''
END_BASIC_CAP_TABLE
EXTENDED_CAP_TABLE ...
# SolverExe: coyote
# Solver Type: coyote
1.02 8 8 t 1
0.5385 0.2046 3.9 0 0
0.017 0 3 2 1
3 0
....
END_EXTENDED_CAP_TABLE
```

Example 2: Rho (Resistivity Table) Included in the Capacitance Table

```
Rho
RhoWidths 0.14 0.28 10
RhoSpacings 0.14 0.28 1
```

```
RhoValues 0.0351 0.02 0.0176

0.0451 0.03 0.01

0.0702 0.04 0.02
```

Example 3: Wire Edge Enlargement - Resistance Included in the Capacitance Table

```
WireEdgeEnlargementR

WeeWidths 0.12 0.16 0.24

WeeSpacings 0.108 0.17 0.24

WeeAdjustments 0 0 0.002

0.011 0.007 0.002

0.022 0.012 0.002
```

Example 4: Wire Edge Enlargement - Capacitance Included in the Capacitance Table

```
WireEdgeEnlargementC

WeeWidths 0.12 0.16 0.24

WeeSpacings 0.108 0.17 0.24

WeeAdjustments 0.01 0.01 0.02

0.01 0.07 0.02

0.02 0.02 0.02
```

Generating Capacitance Table with Specified Scale Factors

You can specify scale factors to convert a corner-specific capacitance table into another capacitance table for a different process corner. Use the `generateCapTbl` command to input a capacitance table and to specify the scale factors. Use the following parameters of the `generateCapTbl` command:

- `-incaptable` *`fileName`*

Specifies the name of an existing capacitance table in ASCII format.

- `-cap` *`totalCapFactor`*

  Specifies the capacitance scale factor.

- `-xcap` *`crossCouplingFactor`*

  Specifies the cross-coupling capacitance scale factor.

- `-res` *`resistanceFactor`*

  Specifies the resistance scale factor.

Related Information

- Performing Extraction in Innovus
- Pre-Requisites for RC Extraction

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 captable file to be used for a specific corner. Information related to the specified corner can be modified using the `update_rc_corner` command. This information is saved in the `viewDefinition.tcl` file, which is read in the subsequent Innovus sessions during design import.

For more information, see Configuring the Setup for Multi-Mode Multi-Corner Analysis in Importing and Exporting Designs.

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.

Related Information

- Generating a Capacitance Table
- Performing Extraction in Innovus

Reading a Quantus Techfile

Quantus Techfiles are required by preRoute (32nm and below), TQuantus, IQuantus, and signoff Quantus extraction engines. Use the `-qx_tech_file` parameter of the `create_rc_corner` command to read in the Quantus Techfile for each process corner.

```
create_rc_corner -name rcCornerName -qx_tech_file fileName
```

This information is saved in the `viewDefinition.tcl` file, which is read in the subsequent Innovus sessions during design import.

Related Information

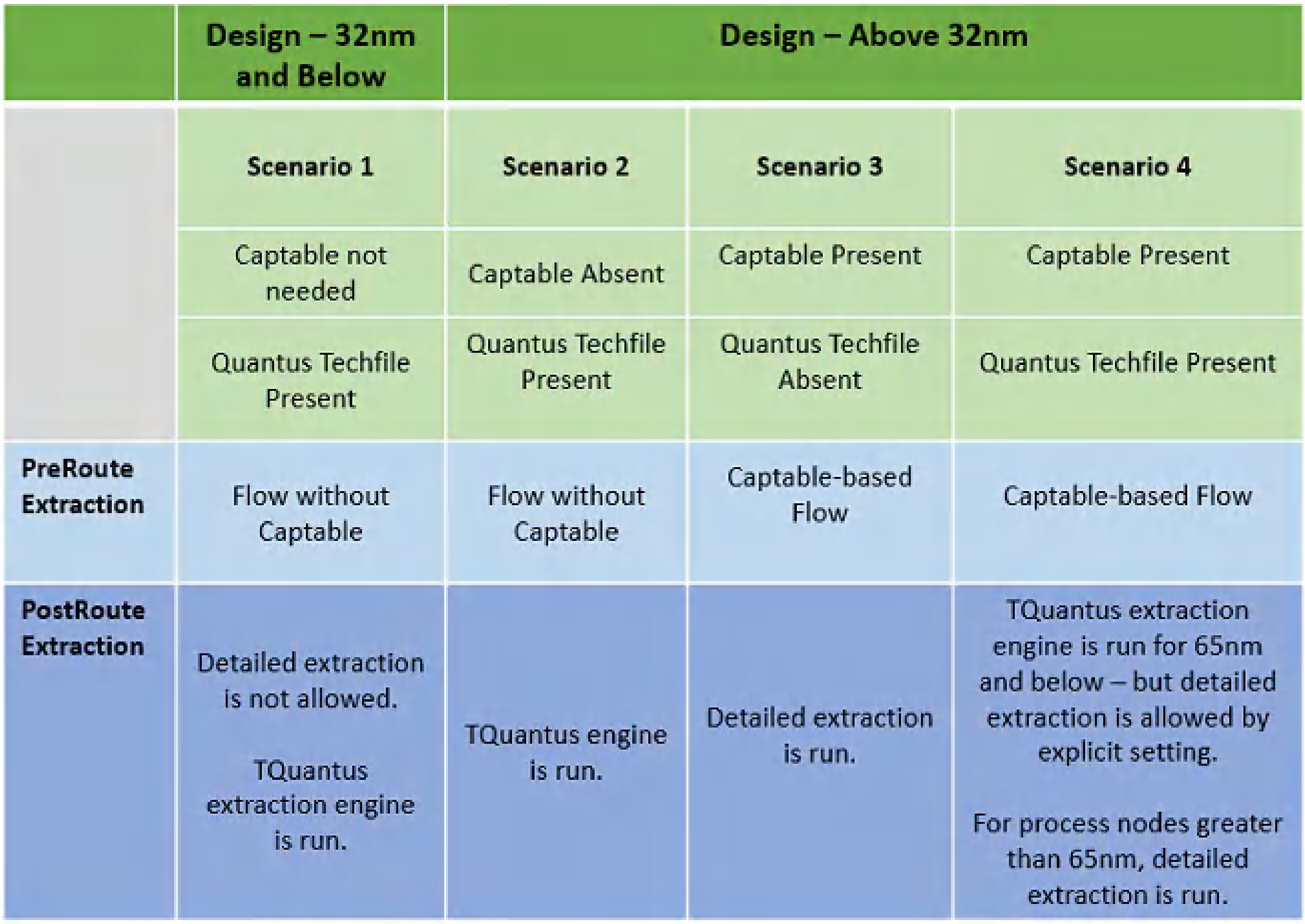
- See Configuring the Setup for Multi-Mode Multi-Corner Analysis in Importing and Exporting Designs.
- Pre-Requisites for RC Extraction

PreRoute Extraction Flow without Capacitance Table Data

Quantus techfiles are essential for performing RC Extraction on designs at 32nm or below nodes. Captable-based extraction is not supported for these designs, and so captables need not be specified for them. PreRoute extraction uses Quantus techfiles (instead of captable files) and postRoute extraction invokes `setExtractRCMode -engine postRoute -effortLevel medium` (which is TQuantus).

Use Model

Extraction Flow without Capacitance Table Data



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

For Designs at 32nm or Below Nodes

Note: The design mode is set in the software using the `setDesignMode -process` command.

- Captable files are not needed and Quantus techfiles are specified - preRoute extraction uses Quantus techfiles even when captable files are provided. PostRoute extraction uses TQuantus extraction engine and detailed extraction is not allowed. This is shown as **Scenario 1** in the above table.

For Designs above 32nm

- Captable files are not specified and Quantus techfiles are specified - preRoute extraction uses Quantus techfiles. PostRoute extraction uses TQuantus extraction engine. Detailed extraction is not allowed. This is shown as **Scenario 2** in the above table.
- Captable files are specified and the Quantus techfiles are not specified - preRoute extraction uses the captable files. PostRoute extraction is called with `-effortLevel low` (detailed extraction) that uses captable files. This is shown as **Scenario 3** in the above table.
- Captable files and Quantus Techfiles are specified - preRoute extraction uses the captable files. PostRoute extraction uses TQuantus extraction engine for designs at 65 nm and below nodes. Detailed extraction is run for designs above 65 nm. This is shown as **Scenario 4** in the above table.

Note: When neither captables nor Quantus techfiles are specified, the software gives an error.

Related Information

- PreRoute RC Extraction
- RC Extraction

Correlating Native Extraction With Sign-Off Extraction

The software accommodates an extraction flow that uses process-dependent scale factors to generate extraction values that are close to the signoff extraction values. With these scale factors, the results generated by the native extraction correlate to the results of signoff extraction. The run time for the native extraction flow is much less than that for signoff extraction.

To generate RC scale factors, use the `generateRCFactor` command. Alternatively, complete the following steps to generate them to correlate native extraction results with sign-off extraction:

From the routed DEF, generate a SPEF file using the sign-off extraction engine. For more information on generating a SPEF file, see Standalone Quantus for Signoff Extraction.

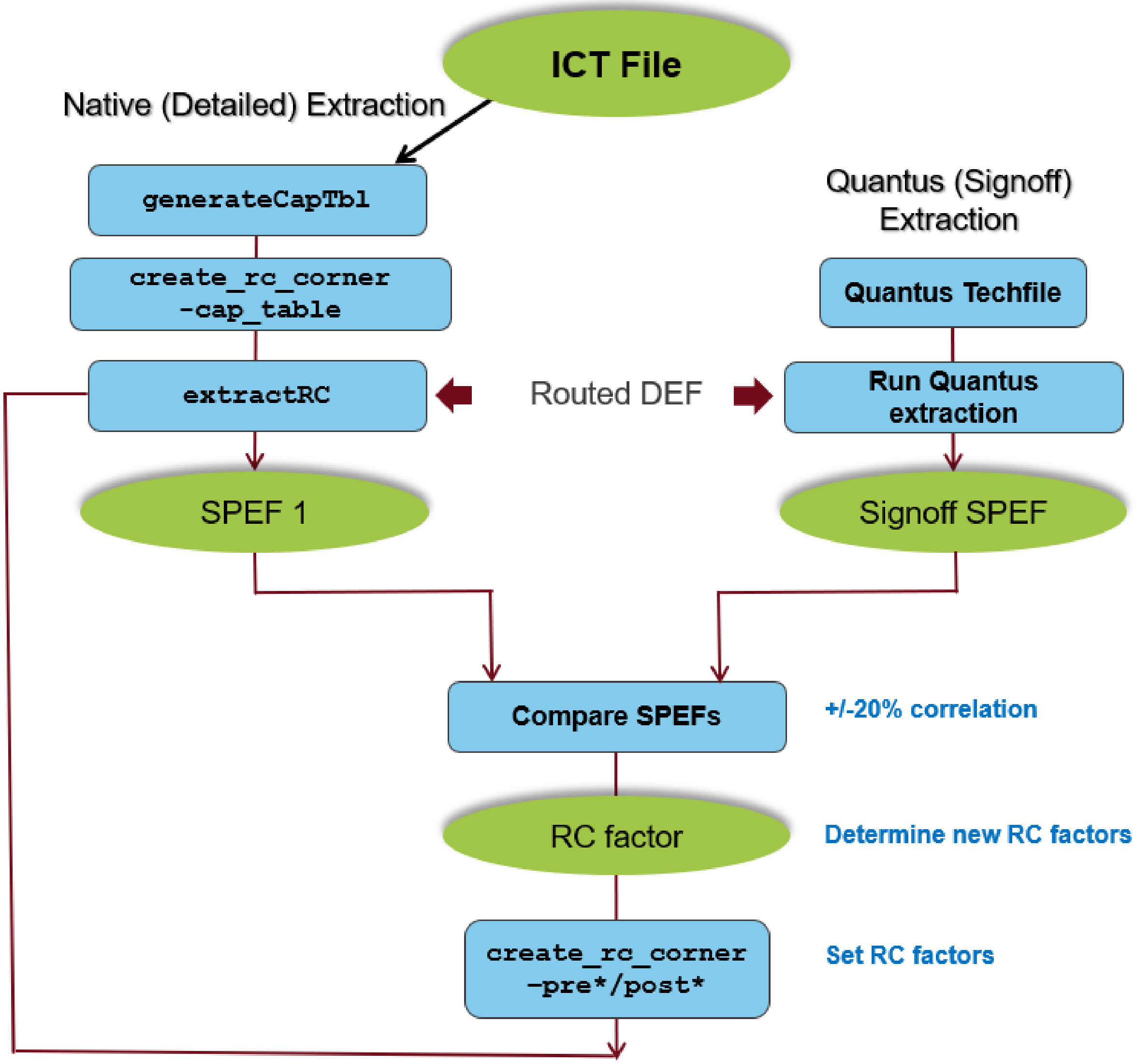
1. Specify the process technology value to automatically set the technology node dependent parameters, by using the following syntax:
   `setDesignMode -process` *processnode*
2. Read in the capacitance table file(s) for extracting interconnect capacitance values with

preRoute and postRoute `-effortLevel low` engine choices.

When using preRoute (32nm and below), TQuantus or IQuantus, read in the Quantus Techfile that contains the interconnect models used by these engine choices. For more information, see "Reading a Quantus Techfile".

3. Generate native extraction SPEF file(s) using the `extractRC` and `rcOut` commands.
4. 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 early global routes and final routes. For postRoute mode, extraction should be run on the same routed design that was used for the signoff extraction SPEF file generation.
5. Compare the SPEF file from native extraction with the SPEF file from signoff extraction using the Ostrich parasitics correlation utility. Use the correlation utility to generate RC factors (scale factors) for total capacitance, cross-coupling capacitance, and resistance. For more information, see Correlating SPEF Files Using the Ostrich Utility.
6. Specify these scale factors using the `-pre*` and `-post*` parameters of the `create_rc_corner` command before future runs of native extraction. For more information, see the section titled, Defining the Scale Factor.
7. Rerun the `extractRC` command to generate a new SPEF file. This file contains capacitance and resistance values that correlate to the values in the Quantus signoff SPEF file.

The following figure shows the flow for generating RC scale factors.



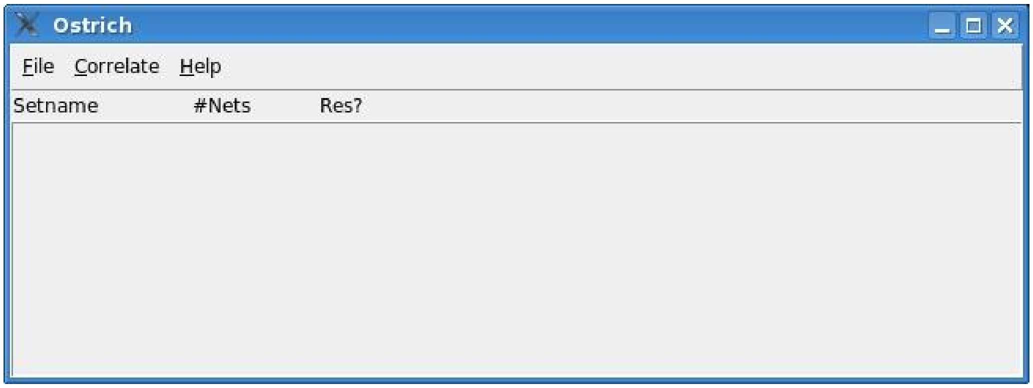
Correlating SPEF Files Using the Ostrich Utility

Use Ostrich to correlate the SPEF files generated using native (`-engine postRoute -effortLevel low`) extraction and signoff extraction. Ostrich is a standalone utility in Innovus. Ostrich generates the scale factors after correlating the SPEF files. You can then set the scale factors for the next extraction cycle.

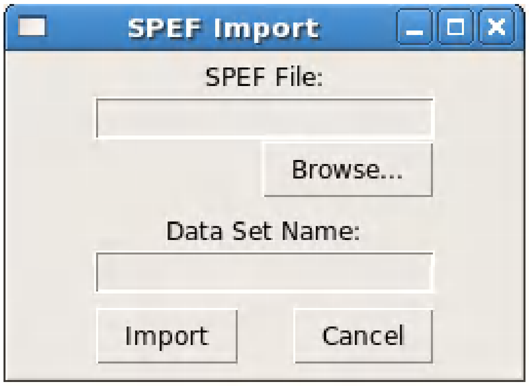
To correlate the SPEF files, complete the following steps:

1. Type `ostrich` at the Innovus prompt. This opens the main Ostrich window.

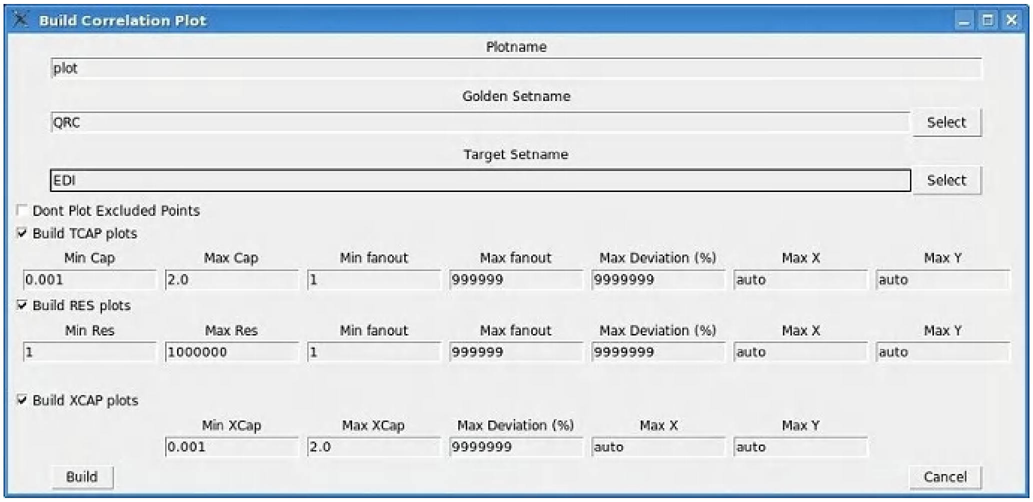
   Note: As an alternative for the GUI, you can also use the command line mode by adding `-nowin` as an argument when starting the tool.



2. In the Ostrich window, click on *File - Import - SPEF*. This opens the SPEF Import form.



3. In the SPEF Import form, specify the name of the sign-off SPEF file and a name in the *Data Set Name* field. Next, click on the *Import* button to add the SPEF values in the Ostrich window.
4. Similarly, import the native extraction SPEF file using the SPEF Import form.
5. Click on *Correlate - Build Plot* option in the Ostrich Window. This opens the Build Correlation Plot window.



Build Correlation Plot

Plotname: plot

Golden Setname: QRC Select

Target Setname: EDI Select

□ Dont Plot Excluded Points

☑ Build TCAP plots

Min Cap	Max Cap	Min fanout	Max fanout	Max Deviation (%)	Max X	Max Y
0.001	2.0	1	999999	9999999	auto	auto

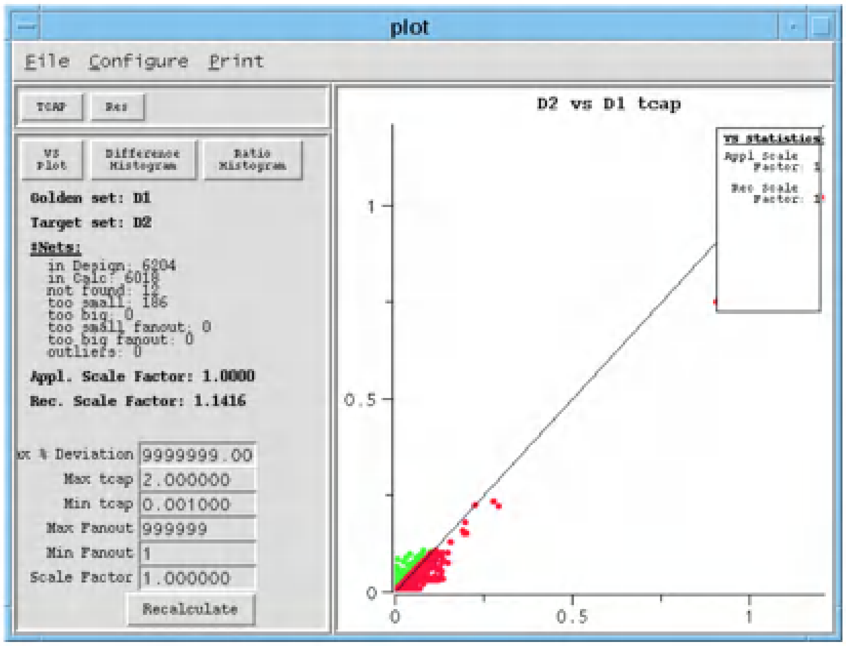
☑ Build RES plots

Min Res	Max Res	Min fanout	Max fanout	Max Deviation (%)	Max X	Max Y
1	1000000	1	999999	9999999	auto	auto

☑ Build XCAP plots

Min XCap	Max XCap	Max Deviation (%)	Max X	Max Y
0.001	2.0	9999999	auto	auto

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 *Build*.
8. In the Ostrich main window, click *Correlate,* and then *Draw Plot.* This opens the plot window.



```
Golden set: D1
Target set: D2
#Nets:
   in Design: 6204
   in Calc: 6018
   not found: 12
   too small: 186
   too big: 0
   too small fanout: 0
   too big fanout: 0
   outliers: 0
Appl. Scale Factor: 1.0000
Rec. Scale Factor: 1.1416

x % Deviation 9999999.00
     Max tcap 2.000000
     Min tcap 0.001000
   Max Fanout 999999
   Min Fanout 1
 Scale Factor 1.000000
```

The plot window displays the suggested scale factor.

Related Information

- PostRoute RC Extraction
- Reading a Capacitance Table
- Reading a Quantus Techfile

Specifying the Scale Factors

You can specify the scale factors in the following manner:

- Change the technology file. You can change the ScaleFactor in the technology file. This scaling is used for each technology.
- Use the `-pre*` and `-post*` parameters of the `create_rc_corner` command.

  You can set scale factors for the resistors and capacitors that are extracted in either preRoute or postRoute extraction mode. You can set different postRoute scale factors for the postRoute engine variants by specifying them as duplets and triplets. For example, `{value1 value2 value3}`

Where:

Single value: If you specify one value, the scale factor applies to effort level low. Scale factor value of 1 is used for medium and high by default.

Duplet: If you specify two values, the first value is used for effort level low and the second value is used for medium. Scale factor value of 1 is used for high by default.

Triplet: If you specify three values, the first value is used for effort level low, the second value is used for medium, and the third value is used for high.

Example

```
create_rc_corner -postRoute_xcap {1.1 1.05} -postRoute_cap 1.2 -postRoute_res 1.1
-preRoute_cap 1.3 -preRoute_res 1.4 -preRoute_clkcap 1.11
```

Note: When saving the design with the `saveDesign` command, the scale factors that are specified with the `create_rc_corner` command are saved in the `viewDefinition.tcl` file, which can be later restored.

Note: The default value for all scale factors, except clock net scale factors, is 1.0. The default value for all clock net scale factors is a symbolic value 0. This indicates that the value of the specific clock net scale factor follows the matching signal net scale factor.

Related Information

- Setting the Scale Factors

Distributed Processing in Extraction

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] [-args arguments] | -custom [-custom_script script]
```

Generating a Capacitance Table in Multi-CPU Mode

You can use the Multiple-CPU Processing Commands to generate a capacitance table in parallel mode when you use the `generateCapTbl` command within Innovus. This functionality is not available for standalone capacitance table generation.

TCL Script to Run the generateCapTbl Command in the Distributed Mode

To run the `generateCapTbl` command in the parallel mode on different hosts, specify the following commands:

```
setDistributeHost -rsh -add { host1 host2 host3}
setMultiCpuUsage -remoteHost
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 IQuantus, TQuantus, and Standalone Quantus Extraction in Multi-CPU Mode

IQuantus, TQuantus, and Standalone Quantus Extraction engines support distributed processing. You can use the Multiple-CPU Processing Commands to invoke IQuantus, TQuantus, and Standalone Quantus Extraction in the multi-CPU mode.

TCL Script for IQuantus, TQuantus, and Standalone Quantus Extraction Invoked in the Distributed Mode

To run IQuantus, TQuantus, and Standalone Quantus 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 IQuantus, TQuantus, and Standalone Quantus Extraction Invoked in the Local Mode

To run IQuantus , TQuantus, and Standalone Quantus 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
```

Related Information

- Performing Extraction in Innovus
- Types of RC Extraction

Using Advanced Virtual Metal Fill in Extraction

Quantus, IQuantus, and TQuantus support Advanced Virtual Metal Fill (VMF). Advanced VMF gives better metal-fill effect estimation as compared to the original default VMF. To use Advanced VMF with IQuantus and TQuantus, an extra command file containing the Quantus CCL commands is required. This file can be specified by using the `setExtractRCMode -extraCmdFile qrc.cmd` command. Include the set-up detailed below in the `qrc.cmd` file.

Setting-up the Advanced VMF Rules

To use Advanced VMF, specify the external VMF rule files. These files will be used regardless of whether VMF rules are included in the `qrcTechFile` or not.

To use the external VMF rule files, use the combination of the following three commands inside the `qrc.cmd` file:

```
metal_fill -type virtual \

    -enable_advanced_virtual_fill true \

   -vmf_metal_scheme_file <metal_scheme_file_name> \

   -vmf_rule_file <param_file_name>
```

When specified, the above options overwrite the VMF specification in the ICT file. This feature lets you use the specified external VMF rule files at runtime. In this case, if the `qrcTechFile` does not contain the VMF rules, Quantus, IQuantus, and TQuantus will use the external VMF rule files. If the `qrcTechfile` contains the VMF rules, the rules in the external VMF rule file will supersede the VMF rules specified in the `qrcTechFile`.

- `-enable_advanced_virtual_fill true`

  Enables Advanced VMF feature. The default value is "`false`".

- `-vmf_metal_scheme_file metal_scheme_file_name`

  Specifies the metal scheme file. This file defines the mapping between layer names and layer types. The format of the file is provided below:

  `Layer`: metal layer name in ICT file

  `Type`: layer type (1, x, y, z, ...)

  `Dir`: routing direction

An example of a metal scheme file is provided below:

Layer	type	Directory
M2	M1	H
M2	Mx	V
M3	Mx	H
M4	Mx	V

- `-vmf_rule_file` *param_file_name*

  Specifies the param file. This file defines the VMF rules, such as size, x/y fill-fill spacing, and net-fill spacing, per metal type defined in the scheme file. The format of the file is provided below:

  `Type`: layer type defined in scheme file

  `L`: VMF length

  `W`: VMF width

  `nf_sp`: minimum net-fill spacing

  `ff_sl`: fill-fill spacing along with length

  `ff_sw`: fill-fill spacing along with width

An example of a param file is provided below.

Type	L	W	nf_sp	ff_sl	ff_sw
M1	1.00	0.12	0.60	0.12	0.12
Mx	1.00	0.12	0.60	0.12	0.12
My	1.20.	0.50	0.30	0.30	0.30
Mz	0.80	0.80	0.60	0.40	0.40

Note: For Quantus, XL+AA license is required when `-enable_advanced_virtual_fill` CCL option is set to `true`. For IQuantus, when Innovus Advanced VMF is turned on, XL+AA license is required.

Related Information

- Pre-Requisites for RC Extraction

Base Delay Analysis

Overview

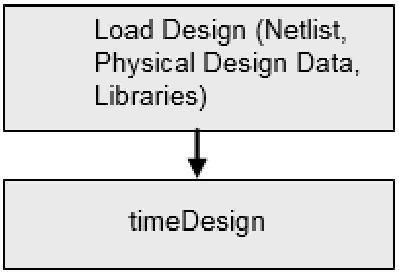
Innovus enables you to perform fast and precise signal integrity-aware delay calculations for cell-based designs. The software combines signal integrity (SI) analysis with timing analysis to check for functional failures due to SI glitch and performs accurate timing calculations (that account for both the SI and IR-drop effects). Innovus utilizes the multi-threaded circuit simulation methods to deliver accuracy, capacity, and performance needed for nano meter designs.

This chapter describes the delay analysis flow and reporting.

Base Delay Analysis Flow

The base delay analysis flow is given below:

Base Delay Analysis Flow



Sample Base Delay Calculation Script

You can use the following script to calculate base delay:

1. Load the design data:

   ```
   restoreDesign
   ```

2. Use `timeDesign` to run Early Global Route, extraction, and timing analysis:

   ```
   timeDesign -postCTS
   ```

3. Debug timing further, if required:

   ```
   report_timing
   ```

4. Report details of base delay calculation for an arc:

   ```
   reportDelayCalculation -from inst1/A -to inst1/Y
   ```

Base Delay Analysis Inputs

- Netlist
- SDC (timing information)
- Routed Innovus database or DEF file (placement and routing information)
- LEF file (physical library)
- XILM data (for hierarchical designs)
- Liberty library (.lib)
- Innovus extended capacitance table file
- Quantus QRC standalone extraction technology file and library (optional)

Base Delay Reporting

The following commands can be used to generate reports on base delay:

- `timeDesign`
- `report_timing`
- `reportDelayCalculation`

Using the timeDesign Command

You can use the `timeDesign` command to run Early Global Route, extraction, and timing analysis. The `timeDesign` command generates detailed timing and DRV reports. For example,

```
Innovus > timeDesign -postCTS
------------------------------------------------------------
                  timeDesign Summary
------------------------------------------------------------

Setup views included:
mission+worst-rcMax test+worst-rcMax mission+worst-rcTyp

+------------------+-------+---------+---------+
| Setup mode       | all   | reg2reg | default |
+-----------------+-------+---------+---------+
| WNS (ns):       | 2.320 | 4.140   | 2.320   |
| TNS (ns):       | 0.000 | 0.000   | 0.000   |
| Violating Paths:| 0     | 0       | 0       |
| All Paths:      | 377   | 368     | 9       |
+-----------------+-------+---------+---------+

+-------------+----------------------------+----------------+
|             | Real                       | Total          |
| DRVs        +----------------+-----------+----------------|
|             | Nr nets(terms) | Worst Vio | Nr nets(terms) |
+-------------+----------------+-----------+----------------+
| max_cap     |  0 (0)         | 0.000     | 0 (0)          |
| max_tran    | 19 (19)        | -0.056    | 27 (27)        |
| max_fanout  | 101 (101)      | -58       | 111 (111)      |
| max_length  | 0 (0)          | 0         | 0 (0)          |
+-------------+----------------+-----------+----------------+

Density: 40.649%
         (100.000% with Fillers)
Routing Overflow: 0.06% H and 0.06% V
------------------------------------------------------------
Reported timing to dir ./timingReports
```

Using the report_timing Command

You can use the `report_timing` command to report timing paths. It is recommended to use the `report_timing -net` parameter to produce a comprehensive report. For example,

```
Innovus > set_global report_timing_format {hpin cell slew delay arrival}
Innovus > report_timing -net

Path 1: VIOLATED Setup Check with Pin seg3/u9/CK
Endpoint: seg3/u9/D (v) checked with leading edge of 'CLK_W_3'

Beginpoint: seg3/u3/Q (v) triggered by leading edge of 'CLK_W_3'
Path Groups: {CLK_W_3}
Other End Arrival Time 1.104
- Setup 0.152
+ Phase Shift 2.000
- Uncertainty 0.050
```

```
= Required Time 2.902
- Arrival Time 3.151
= Slack Time -0.249

Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 1.135
= Beginpoint Arrival Time 1.135
-------------------------------------------------
Pin             Cell    Slew     Delay    Arrival
                                          Time
-------------------------------------------------
seg3/u3/CK      -        0.091    -       1.135
seg3/u3/Q ->    DFF      0.318    0.303   1.438
seg3/u4/A       BUF      0.318    0.008   1.446
seg3/u4/Y       BUF      0.003    0.158   1.604
seg3/u5/A       INV      0.005    0.003   1.607
seg3/u5/Y       INV      0.499    0.140   1.748
seg3/u6/A       INV      0.549    0.152   1.900
seg3/u6/Y       INV      0.793    0.528   2.427
seg3/u7/A       BUF      0.794    0.003   2.430
seg3/u7/Y       BUF      0.596    0.404   2.835
seg3/u7_a/A     BUF      0.596    0.060   2.895
seg3/u7_a/Y     BUF      0.003    0.250   3.145
seg3/u8/A       BUF      0.003    0.001   3.146
seg3/u8/Y       BUF      0.042   -0.023   3.123
seg3/u9/D       DFF      0.072    0.029   3.151
------------------------------------------------
```

Using the reportDelayCalculation Command

You can use the reportDelayCalculation command to report the delay calculation information for a cell or net timing arc. For example,

```
innovus > reportDelayCalculation -from seg3/u5/Y -to seg3/u6/A

From pin    : seg3/u5/Y
Cell        : INV
Library     : cell_w
To pin      : seg3/u6/A
```

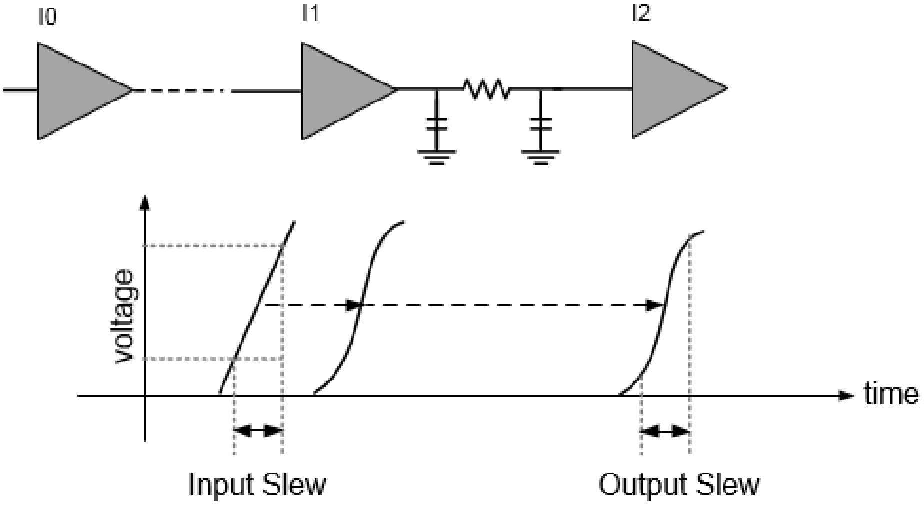
```
Cell : INV
Library : cell_w
Delay type: net

-------------------------------------------------
RC Summary for net seg3/n5
-------------------------------------------------
Number of capacitance   : 17
Net capacitance         : 0.293534 pF
Total rise capacitance  : 0.320849 pF
Total fall capacitance  : 0.320780 pF
Number of resistance    : 17
Total resistance        : 567.671387 Ohm
----------------------------------------------------
                          Rise           Fall
----------------------------------------------------
Net delay                 : 0.151900 ns  0.119000 ns
From pin transition time  : 0.499000 ns  0.211500 ns
To pin transition time    : 0.549100 ns  0.248000 ns
----------------------------------------------------
```

Limitations of Traditional Delay Calculators

Traditional delay calculators use delay as a function of the input slew and output load. With traditional delay calculators, a single linear slew value is used as the input to analyze a stage. This methodology cannot produce the desired accuracy that new technologies demand.

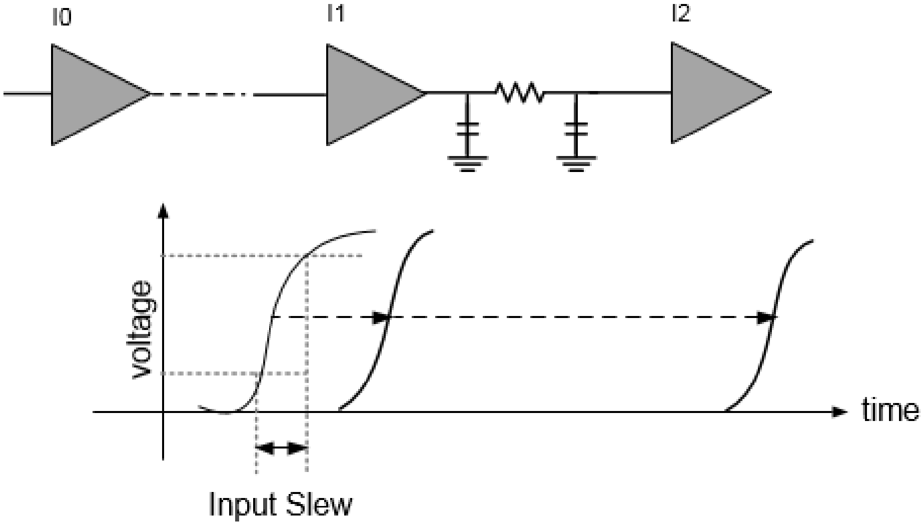
This is illustrated in the figure below.

Traditional Delay Calculation using Delay as function of slew and load



The advanced technologies (28nm and below) require waveform-based delay calculators to accurately calculate the delays based on waveforms. The waveform-based delay calculators use real waveforms as input to analyze a stage, as shown in the figure below.

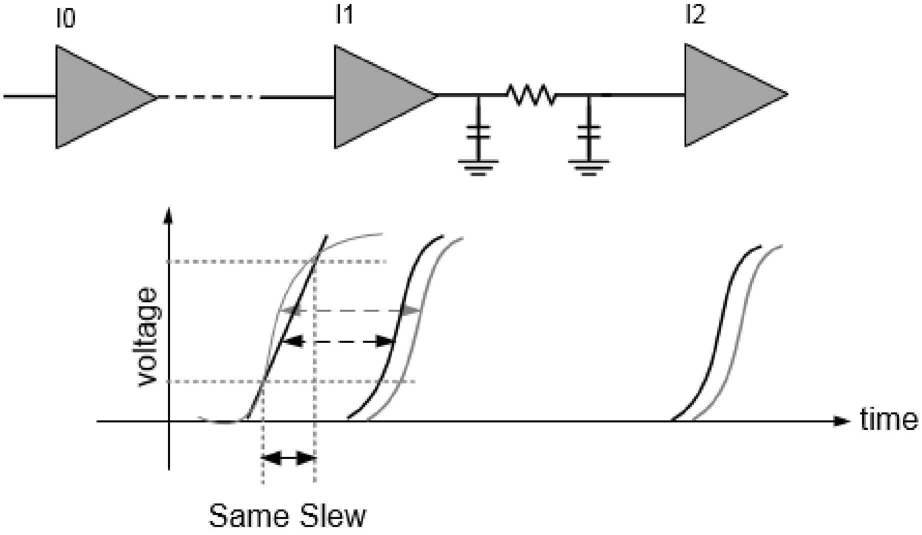
Ideal Waveform based Delay Calculation



There are several shortcomings when you use traditional delay calculators. Some of these are:

- Traditional delay calculators use single slew to calculate stage delays. This may not produce accurate results.
- Different waveforms with the same slew value produce the same delays.

Shortcomings of Traditional Delay Calculation



In the above figure, the grey waveforms indicate the input waveforms in an ideal case. The black lines indicate the linear input slew values and slew waveforms, respectively.

Here, the linear input slew value and the input waveforms use the same slew, however, the delay numbers are different. The long curve of the input waveform (grey) can produce larger delays as compared to the one produced with the linear input slew (black). Also, the measurement point shift can contribute to delay inaccuracy.

Base Delay Analysis with Equivalent Waveforms

To overcome the shortcomings of traditional delay calculators, Innovus provides two different approaches for performing delay calculations. These are described below:

- Equivalent Waveform Model
- Waveform Propagation

Equivalent Waveform Model (EWM)

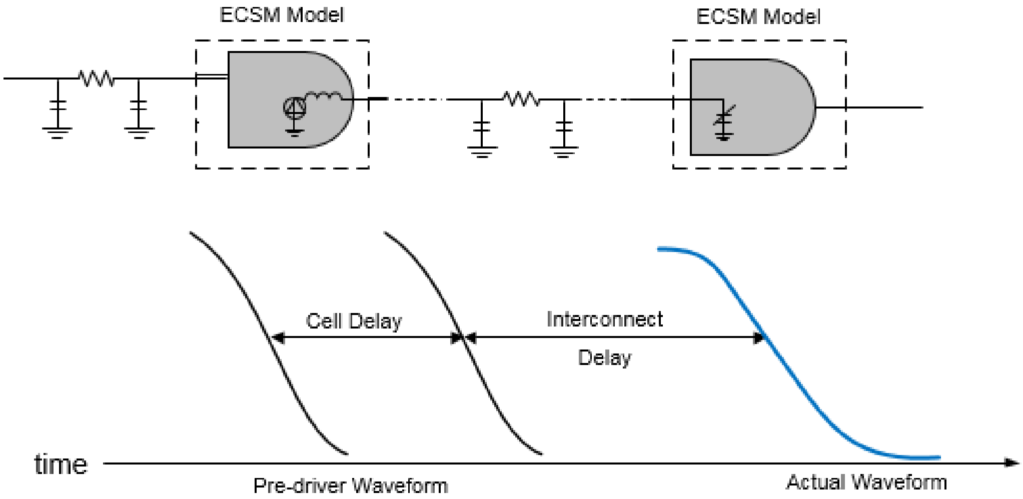
To achieve accuracy, the waveform shapes are required to be included during delay calculations. The equivalent waveform model (EWM) computes equivalent receiver output based on the input waveform shapes and adjusts the interconnect delay accordingly. The adjustment in delay compensates for any inaccuracies that delay calculation might cause in the next stage due to lack of waveform shape information. The EWM approach provides a technique for producing higher accuracy results when compared to Spice.

When a stage is analyzed during delay calculation, a pre-defined waveform from the library (based on single input slew value) is used as a stimulus. When the EWM mode is not enabled, the input slew is measured from the actual waveform computed during the previous stage analysis. This may have entirely different characteristics compared to the pre-defined waveform used in the current stage. As a result, the delay impact due to waveform shape differences may be affected.

The following figure illustrates delay calculation without EWM.

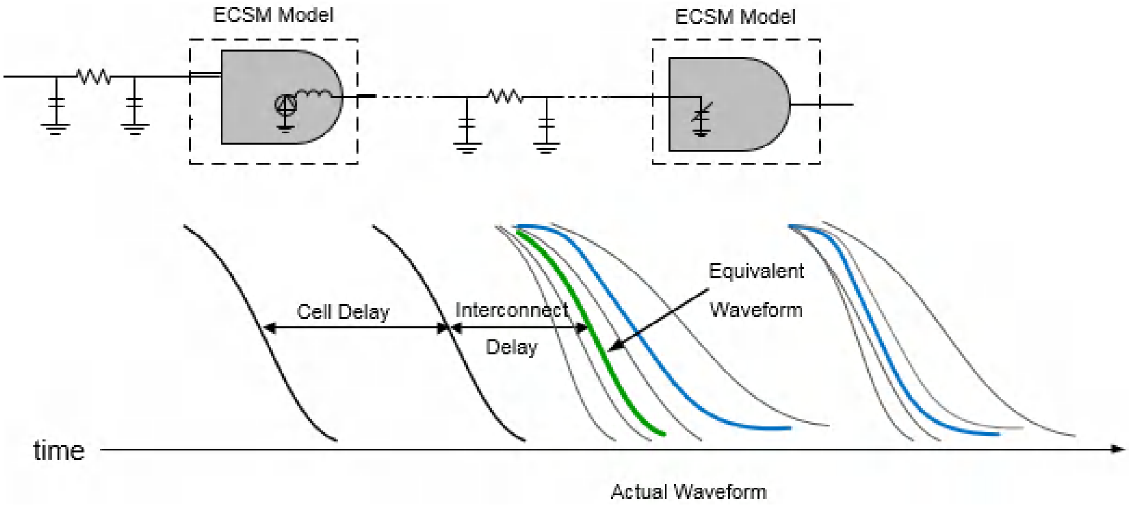
The following figure illustrates delay calculation without EWM.

Delay Calculation without EWM



The following figure illustrates delay calculation with EWM.

Delay Calculation with EWM



When EWM is enabled, the software computes the delay impact of waveform shapes on receiver cells, and computes the delay impact - thus providing an overall improvement in path delay accuracy. When EWM is enabled in SI analysis, the software provides delay adjustments based on the receiver noise response to a noisy transition. This helps to reduce the SI pessimism that will be reported if the total delay is measured on noisy waveforms at the receiver input.

Use Model

Equivalent waveform model can be enabled by using the following setting:

```
setDelayCalMode -equivalent_waveform_model no_propagation
```

Waveform Propagation

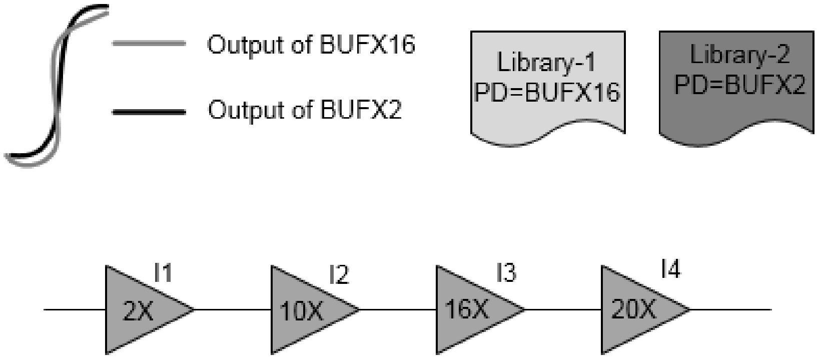
The waveform shapes have a significant impact on delay calculations. Traditionally, delay calculation uses a single pre-driver waveform for specific slew value at the cell input to compute the response on the output of a cell.

Consider two libraries characterized with pre-driver cells, BUFX16 and BUFX2. An accurate delay on all the instances driven by BUFX16 will be reported, when the library uses BUFX16 as a pre-driver cell. In this case, the instance I4 produces more accurate results as the input waveform matches with the pre-driver waveform. The accuracy of other cells is impacted due to a difference in results for the pre-driver cell versus the actual driving cell. To facilitate actual waveform propagation through a path, the waveform propagation feature stores the actual waveform instead of a single slew value at the input of each cell.

As shown in the figure below, both the BUFX16 slew (in grey) and BUFX2 slew (in black) have the same value but their waveform shapes are different. The same slew having different waveform shapes can produce different delays. The delay accuracy is a function of input waveform shapes, even if the slew values are same. If the input waveform shape is different from the input waveform used for cell characterization, the delay accuracy will be significantly affected.

The following figure illustrates waveform impact on delay.

Waveform impact on delay



Requirements for Waveform Propagation

The waveform propagation using ECSM models will be more accurate with additional receiver pincap points. Cadence recommends at least eight points. The last three points can be the lower slew threshold, delay measurement point, and upper slew threshold. The rest of the five points must be selected to represent the tail waveform accurately. It is also recommended to have actual pre-driver waveform in the ECSM libraries.

Use Model

Waveform propagation can be enabled by using the following setting:

```
setDelayCalMode -equivalent_waveform_model propagation
```

EWM-Only vs Waveform Propagation

EWM-Only

- Real waveform tail impact on the next stage is predicted and added to the current wire delay.
- The receiver cell is assumed to be the driver lumped load.

Waveform Propagation

- Real waveforms are stored and used as input for the next stage. The input waveform tail impact is used at the appropriate point.
- Unlike EWM-Only, the waveform propagation computes accurate impact of the tail as it uses distributed parasitics of wires.

EWM and Waveform Propagation - Impact on the Flow

Using Equivalent Waveform Model or Waveform Propagation methodology can impact the flow in the following ways:

- Waveform propagation is only supported in Innovus for post-route timing analysis. For pre-route STA, even if waveform propagation has been enabled, delay calculation is with EWM-only (that is, no propagation).
- Enabling equivalent waveform modeling increases runtime by 10% with no significant increase in memory.

- Enabling waveform propagation increases runtime by 15%. In graph-based analysis mode, there is an 8% increase in the memory.

Normalized Driver Waveform in Library

The normalized driver waveform (pre-driver waveform) should be added to a library while characterizing so that delay can be computed accurately (as there can be different waveforms with the same slew). There are two types of pre-drivers – active and analytical. Cadence recommends that you use the active pre-driver. The active pre-driver waveform has a longer tail than the analytical pre-driver, and thus represents the real design scenario. In the absence of NDWs (normalized driver waveform) in a library, Innovus auto-generates analytic pre-driver waveforms.

An example of normalized driver waveform in a library is given below:

```
normalized_driver_waveform (waveform_template_name) {
    driver_waveform_name : "PreDriver20.25:rise";
    index_1 ("0.00233, 0.01301, 0.05092, 0.1233, 0.2361, 0.3943, 0.6026");
    index_2 ("0, 0.083, 0.166, 0.25, 0.333, 0.417, 0.5, 0.583, 0.666, 0.75, 0.833,
0.917,1");
      values ( \
        "0, 0.00037, 0.0005041, 0.0006424, 0.0008009, 0.0009783, 0.001179, 0.00140,
0.00166, 0.001995 0.002375, 0.002833, 0.003389", \
        "0, 0.00220 0.0029643 0.003777, 0.004709, 0.005752, 0.006935, 0.00828374,
0.00988784, 0.0117336, 0.013969, 0.0166759, 0.0199283", \
        "0, 0.00862494, 0.0116002, 0.01473, 0.0184297, 0.0225117, 0.0271421, 0.0324164,
0.0386937, 0.0459165, 0.05466, 0.06571, 0.0779846", \
        "0, 0.0208867, 0.0280917, 0.0357977, 0.0446304, 0.0545157, 0.065729, 0.0785015,
0.0937029, 0.111194, 0.132378, 0.15803, 0.188852", \
        "0, 0.0399897, 0.0537844, 0.0685384, 0.0854496, 0.104376, 0.125845, 0.150299,
0.179404, 0.212893, 0.253452, 0.302566, 0.361577", \
```

Timing Library Requirement for Accurate Analysis for 20nm and Below

The ECSM library characterization for 20nm static timing analysis (STA) signoff meets the challenges of accurate timing analysis in 20nm. To produce more accurate results for the 20nm process nodes, you can use the following:

- 8-piece pin capacitances in ECSM timing libraries

- 2% - 98% ECSM waveform range

ECSM Libraries with 8-Piece Pin Capacitances

The 8-piece pin capacitance in the ECSM timing libraries are required to accurately model back miller current. Traditionally, the receiver pin capacitance in an ECSM library characterization is measured at the slew thresholds - that may be 30% to 70% of the VDD. As a result, the use of such thresholds in the ECSM libraries may result in some missing important data at the tail of the waveform. The 3-piece capacitance tables are extended to 8-piece tables for 20nm nodes to better capture the waveform distortions due to back miller current at the waveform tail. The selection of 8-piece pin capacitance is made such that the required 20nm waveform distortion information can be captured correctly. Since the waveform distortion happens mostly at the tail of waveforms, the pin-cap thresholds are selected so that there are more points on the tail.

Timing Analysis

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 allows you determine how fast a design can run without incurring timing violations. You can use the results of timing analysis to fine tune and debug the speed-limiting, critical paths in a design.

You can perform timing analysis using Cadence® and third-party constraint formats and timing libraries (dotlib).

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 black boxes. Once you have defined the initial timing model of the black boxes, you can modify arc definitions and verify the consequences in timing analysis.

Minimum and Maximum Timing Analysis

To read in libraries with multiple operating conditions for minimum and maximum analysis, you can:

- Create a MMMC configuration file. You can use the Innovus System GUI.
- Specify the operating conditions using the `create_delay_corner` command.
- Specify the `setAnalysisMode` command.

Timing Analysis Ideal and Propagated Modes

setAnalysisMode		Clock Propagation	Clock Latency
`-skew false`		Forced Ideal	No Effect
`-skew true`	`-clockPropagation    forcedIdeal`	Forced Ideal	SDCs in Effect
`-skew true`	`-clockPropagation    sdcControl`	*SDCs in Effect	**SDCs in Effect

* Both `-clockPropagation sdcControl` and `set_propagated_clock` are 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

The software supports MMMC-based designs during initializiation. Innovus follows the `init_design` based flow. This flow requires a valid MMMC specification to provide the necessary timing, SI, constraint, and extraction related data for the system. The default MMMC objects are treated as real user MMMC objects and are saved/restored from the MMMC 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 File Menu in the *Innovus Menu Reference*.
- Specify the operating/timing conditions to use for timing analysis.

Use the operating conditions to specify process, voltage, and temperature (PVT) values. Operating conditions are defined in the timing library and read into the Innovus session when you import the design. You can use a single set of operating conditions for setup and hold analysis, or you can specify minimum and maximum conditions.

- Check and report timing libraries by generating the timing library report.
- Check and report cell footprints by generating the cell footprint report.
- Define RC corners for extraction. In the MMMC configuration file you can define three different types of RC corners - typical, best, and worst. Several RC corners can be defined.
- Specify the analysis mode you want to use for timing analysis. There are three types of analysis modes: single, best-case worst-case (BC-WC), and on-chip variation. For more information, see Timing Analysis Modes.

For more information, see "Importing and Exporting Designs" chapter of the *Innovus User Guide*.

Calculating Clock Latency

The Innovus software calculates clock latency based on the following two settings:

- Analysis mode set using the `setAnalysisMode` command.
- The `set_propagated_clock` and `set_clock_latency` constraints values.

Depending on these settings, the clock latency can be equal to either 0.0 or the value of the `set_clock_latency` constraint, or the delay computed by propagation along the clock path.

The Innovus 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, `0`ns clock latency is reported (ideal mode).

  Note: The `-clockPropagation forcedIdeal` option forces ideal clock mode, even if the `set_propagated_clock` command is specified in the constraints file.
- `setAnalysisMode -skew false -clockPropagation sdcControl`

  Or,

  `setAnalysisMode -skew false -clockPropagation forcedIdeal`
- No latency is reported (ideal mode).

Note: When you use the `-skew false` parameter, clock latencies are ignored.

Path Exception Priorities

The following are the path exception priorities if a path in the design matches more than one path exception:

- set_false_path
- set_min_delay
- set_max_delay
- set_multicycle_path

If there is more than one exception of a given type, for example, the set_multicycle_path command, the path exception that is more specific has higher priority. A path exception is more specific if it specifies a longer path than the other. For example, the -from, -to options will have priority over the -from option.

If the path has the same number of reference points, then:

- -from option has priority over the -to option
- -to option has priority over the -through option

Note: To check for ignored path exceptions, use the report_path_exceptions -ignored command.

The following list shows the priorities (from highest to lowest) for path exceptions applied to the same path.

Path Exception Priorities

1. set_false_path (Highest)
2. set_max_delay -from pin_list
3. set_max_delay -to pin_list
4. set_max_delay -through pin_list
5. set_max_delay -from clkwave_name
6. set_max_delay -to clkwave_name
7. set_max_delay (The most constraining adjustment has higher priority over less constraining adjustments.)
8. set_multicycle_path -from pin_list
9. set_multicycle_path -to pin_list

Timing Analysis Modes

The Innovus software provides different timing analysis modes and accordingly performs calculations for setup and hold checks for each mode. The timing analysis mode types are:

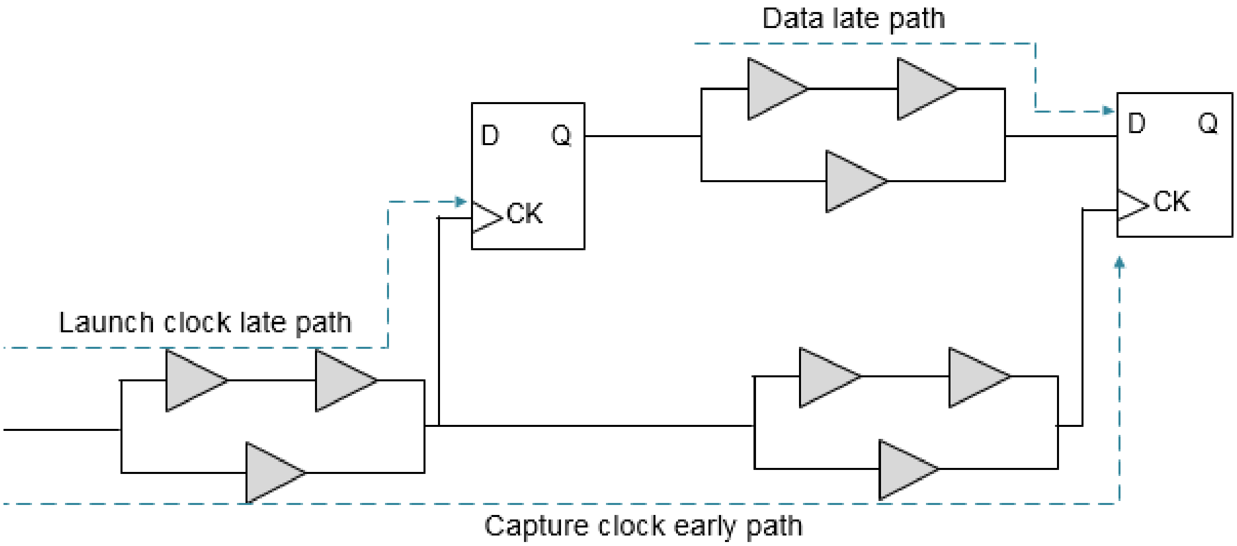
- Single Timing Analysis Mode
- Best-Case Worst-Case (BC-WC) Timing Analysis Mode
- On-Chip Variation (OCV) Timing Analysis Mode

For a better understanding of these modes, it is important to understand the impact of early and late paths in a design, as explained in the following section.

Definition of Early and Late Paths

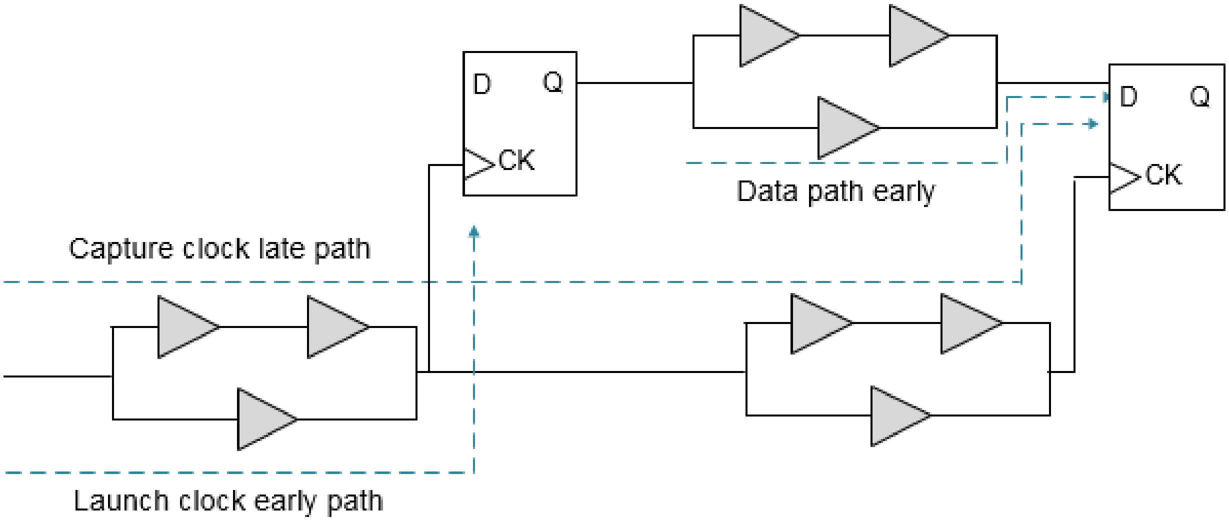
Early and late paths are referred to as the shortest and longest paths, respectively, and are used for delay calculations. The early or minimum path delay is the minimum delay through cells and nets in the path. The late or maximum path delay is the maximum delay through cells and nets in the path. The following figure shows a setup check with late launch clock and early capture clock:

Illustration of Setup Check- Setup check with early and late paths



The following figure shows hold check with early launch clock (shown in dotted line), and late capture clock (shown in solid line).

Illustration of Hold Check - Hold check with early and late paths



Single Timing Analysis Mode

In this mode, the Innovus software uses a single set of delays (using one library group) based on one set of process, temperature, and voltage conditions.

To set the timing analysis mode as single, you can use the following command:
`setAnalysisMode -analysisType single`

In single analysis mode, only maximum delay values are used for delay calculation. The `-max` options of commands in constraints are used for both minimum and maximum analysis in single analysis mode. For example, `set_annotated_delay -max 10` command will be used for both minimum and maximum analysis.

Even with single library group, cell delay can have variation in maximum delay based on sdf_cond, timing derates, and other inputs. In single analysis mode, early and late path delays are the minimum and maximum, respectively, of this range of maximum delay.

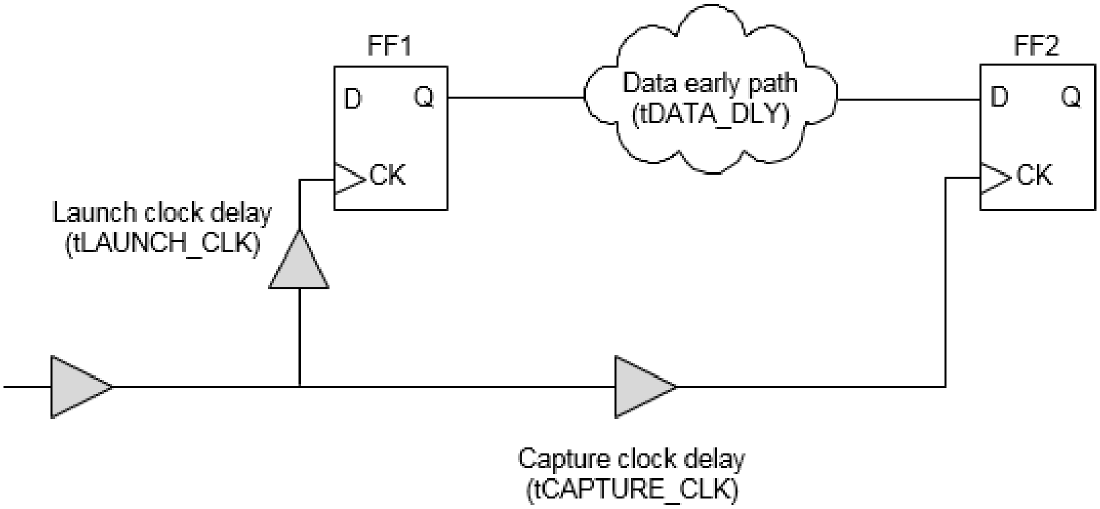
Single Timing Analysis Mode - Setup and Hold Checks

These are explained in the sections below.

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.

Setup Check in Single Timing Analysis Mode

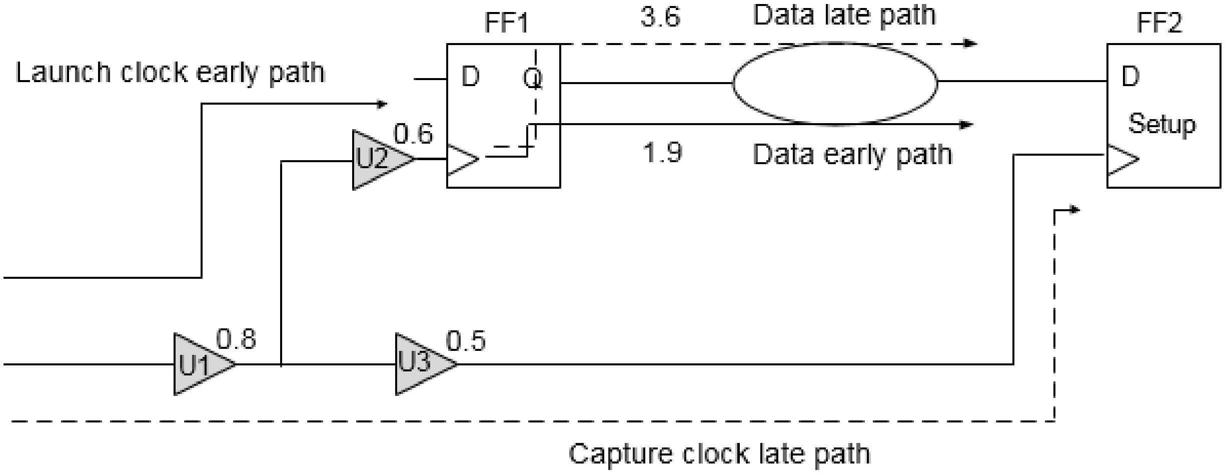


For zero slack value in a setup check, the following condition should be met:

launch clock late path ($_{\text{tLAUNCH\_CLK}}$) + *data clock late path* ($_{\text{tDATA\_DLY}}$) <= *capture clock early path* ($_{\text{tCAPTUE\_CLK}}$) + *clock period* - *setup*

The following figure shows the setup check on the path from FF1 to FF2.

Setup Check in Single Timing Analysis Mode



The software uses a library to calculate the maximum delay. For setup check, the software considers two paths between the two registers, FF1 and FF2 - the late path delay is used to calculate slack during setup check.

The following values are assumed in this example:

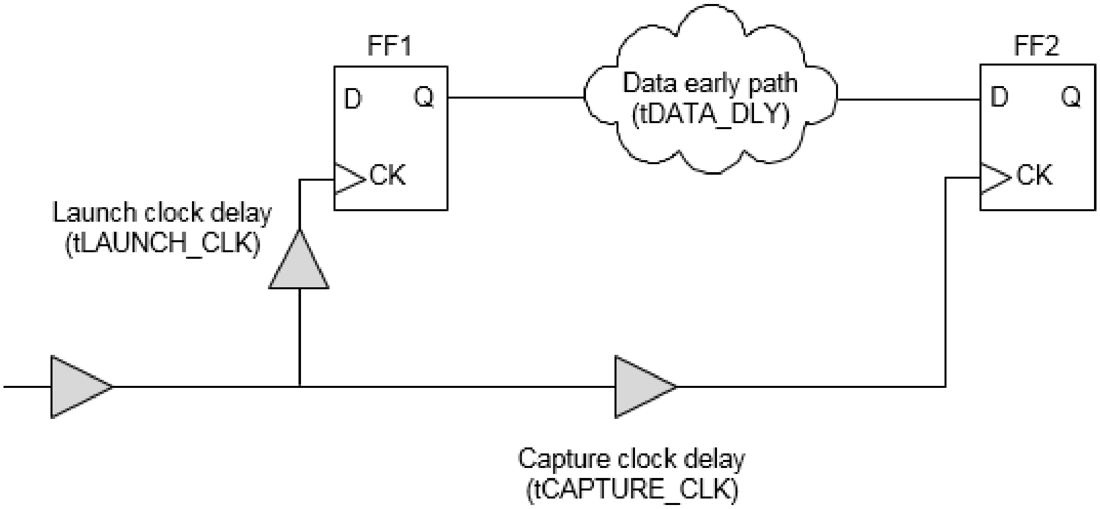
Clock period = 4 ; Clock source latency = none ; Clock mode = Propagated

Data late path delay	3.6
Data early path delay	1.9
Launch clock late path delay	0.8 + 0.6 =1.4
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
Setup Slack	5.1 - 5 = 0.1

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.

Hold Check in Single Timing Analysis Mode

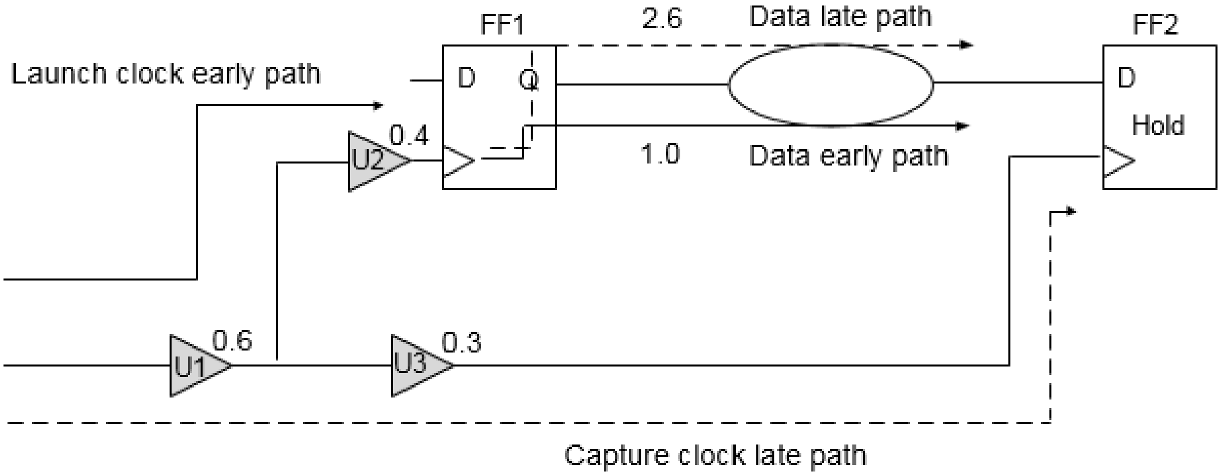


For zero slack value in a hold check, the following condition should be met:

launch clock early path ($_{\text{tLAUNCH\_CLK}}$) + *data early path* ($_{\text{tDATA\_DLY}}$) => *capture clock late path* ($_{\text{tCAPTUE\_CLK}}$) + *hold*

The following example shows the hold check on a path from FF1 to FF2.

Hold Check in Single Timing Analysis Mode



The following values are assumed in this example:

Clock period = 4 ; Clock source latency = none ; Clock mode = Propagated

Data late path delay	2.6
Data early path delay	1.0
Launch clock early path delay	0.6 + 0.4 = 1.0
Capture clock late path delay	0.6 + 0.3 = 0.9
Hold	0.2
Data arrival time	1 + 1 = 2
Data Required Time	0.2 + 0.9 = 1.1
Hold Slack	2 – 1.1 = 0.9

Performing Timing Analysis in Single Analysis Mode

To perform Timing Analysis in Single Analysis Mode, follow the steps given below:

1. Create a single corner MMMC configuration file and set the `init_mmmc_file` variable to point to it.
2. To create library sets, corners, and modes, use the following set of commands:

```
create_library_set -name my_max_library_set
```

```
-timing [list /icd/libs/syn/stdcell/slow/slow.lib]

create_constraint_mode -name my_constraint_mode
-sdc_files [list ./constraints/design.sdc]

create_rc_corner -name my_wc_corner_worst
-qx_tech_file /icd/libs/tech/6mlv-wc.tch

create_delay_corner -name my_delay_corner_max -library_set my_max_library_set
-rc_corner my_wc_corner_worst
-opcond slow

create_analysis_view -name my_wc_analysis_view -constraint_mode my_constraint_mode
-delay_corner my_delay_corner_max

set_analysis_view -setup my_wc_analysis_view
-hold my_wc_analysis_view
```

3. Load the design using the following commands:

```
source init.globals
init_design
```

4. Set the analysis mode to single, setup and propagated clock mode:

```
setAnalysisMode -analysisType single -checkType setup
-skew true -clockPropagation sdccontrol
```

5. Generate the timing reports for setup:

```
report_timing
```

6. Set the analysis mode to hold and propagated clock mode:

```
setAnalysisMode -checkType hold
-skew true -clockPropagation sdcControl
```

7. Generate the timing reports for hold:

```
report_timing
```

Best-Case Worst-Case (BC-WC) Timing Analysis Mode

In best-case worst-case (BC-WC) analysis mode, the software uses delays from the maximum library group for all the paths during setup checks and minimum library group for all the paths during hold checks. In this mode, the Innovus software considers two operating conditions and checks both operating conditions in one timing analysis run. To set the timing analysis mode as BC-WC, you can use the following command:
setAnalysisMode `-analysisType bcwc`

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 this 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 minimum and maximum operating conditions.

Note: The software considers the network latency, specified using the `set_clock_latency -max` (or `-min`) command, for ideal clocks only. In propagated mode, actual clock propagated latency values will be used.

Setup Check in Best-Case Worst-Case Mode

For setup check, the software calculates delay values from the maximum library group for data arrival time, and network delay of both launch and capture clocks (in propagated mode).

The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

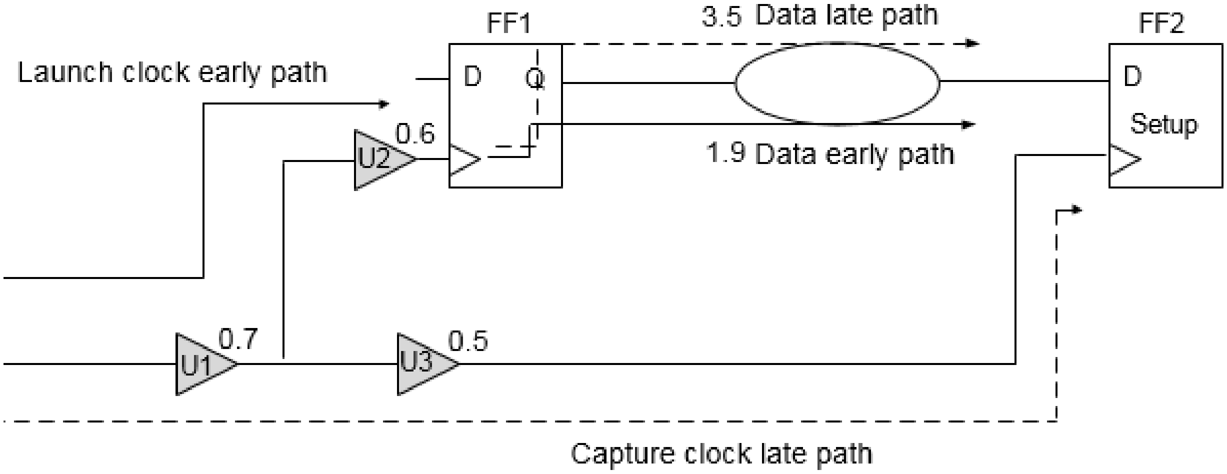
Clock Path **(Operating Condition)**	**Constraint Used**
Launch clock late path (max)	`set_clock_latency -source -late -max` *`value`*
Capture clock early path (max)	`set_clock_latency -source -early -max` *`value`*

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	**Constraint Used**
Launch clock late path (max)	`set_clock_latency -max` *`value`*
Capture clock early path (max)	`set_clock_latency -max` *`value`*

The following example shows the setup check on a path from FF1 to FF2.

Setup Check in Best-Case Worst-Case Analysis Mode



The software uses the max library to scale all delays at worst-case conditions.

The following values are assumed in this example:
Clock period = 4 ; Clock source latency = none ; Clock mode = Propagated

Data late path delay	3.5
Data early path delay	1.9
Launch clock early path delay	0.7 + 0.6 = 1.3
Capture clock late 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
Setup Slack	5 - 4.8 = 0.2

HOLD Check in Best-Case Worst-Case 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 source latency in both ideal and propagated modes for hold checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	`set_clock_latency -source -early -min` *value*
Capture clock late path (min)	`set_clock_latency -source -late -min` *value*

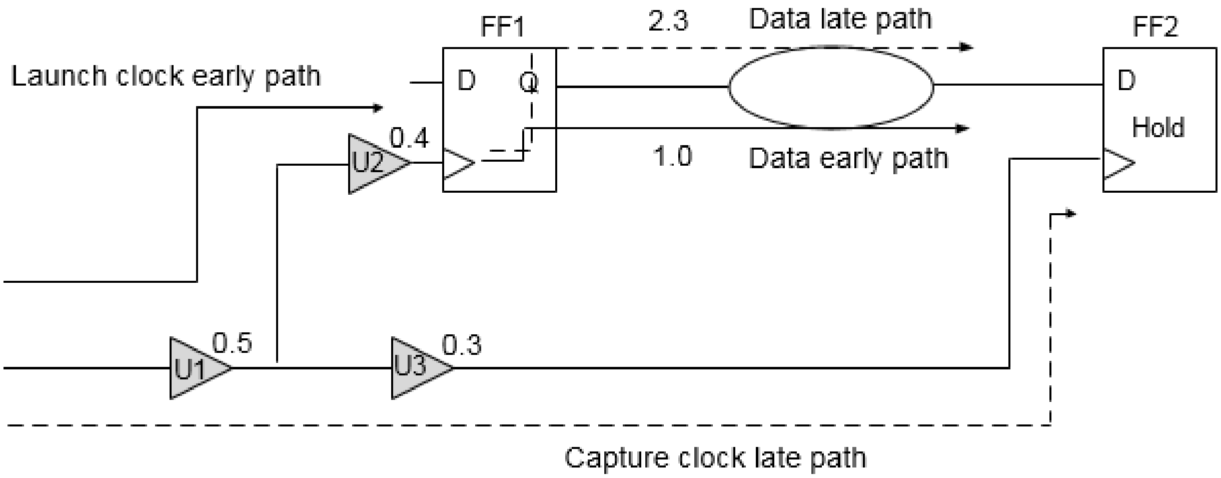
The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	`set_clock_latency -min` *value*
Capture clock late path (min)	`set_clock_latency -min` *value*

Note: You can also use one library containing two operating conditions in this mode.

The following shows the setup check on the path from FF1 to FF2.

Hold Check in Best-Case Worst-Case Timing Analysis Mode



The software uses the min library to scale all delays at best-case conditions.

The following values are assumed in this example:

Clock period = 4 ; Clock source latency = none ; Clock mode = Propagated

Data late path delay	2.3
Data early path delay	1.0
Launch clock early path delay	0.5 + 0.4 = 0.9

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
Hold Slack	1.9 - 0.9 = 1

Performing Timing Analysis in Best-Case Worst-Case Analysis Mode

To perform timing analysis in best-case worst-case (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.
2. To create library sets, corners, and modes, use the following set of commands:

   ```
   create_library_set -name my_max_library_set
   -timing [list /icd/libs/syn/stdcell/slow/slow.lib]
   create_library_set -name my_min_library_set
   -timing [list /icd/libs/syn/stdcell/fast/fast.lib]

   create_constraint_mode -name my_constraint_mode
   -sdc_files [list ./constraints/design.sdc]

   create_rc_corner -name my_wc_corner_worst
   -qx_tech_file /icd/libs/tech/6mlv-wc.tch
   create_rc_corner -name my_bc_corner_worst
   -qx_tech_file /icd/libs/tech/6mlv-wc.tch

   create_delay_corner -name my_delay_corner_max
    -library_set my_max_library_set
   -rc_corner my_wc_corner_worst
        -opcond slow
   create_delay_corner -name my_delay_corner_min
   ```

```
-library_set my_min_library_set
    -rc_corner my_bc_corner_worst
    -opcond fast

create_analysis_view -name my_wc_analysis_view
-constraint_mode my_constraint_mode
    -delay_corner my_delay_corner_max
create_analysis_view -name my_bc_analysis_view
-constraint_mode my_constraint_mode
     -delay_corner my_delay_corner_min

set_analysis_view -setup my_wc_analysis_view -hold my_bc_analysis_view
```

3. Load the design using the following commands:

   ```
   source init.globals
   init_design
   ```

4. Set the analysis mode to BC-WC, setup and propagated clock mode:

   ```
   setAnalysisMode -analysisType bcwc -checkType setup -skew true -clockPropagation
   sdcControl
   ```

5. Generate the timing reports for setup:

   ```
   report_timing
   ```

6. Set the analysis mode to hold and propagated clock mode.

   ```
   setAnalysisMode -checkType hold
   ```

7. Generate the timing reports for hold.

   ```
   report_timing
   ```

On-Chip Variation (OCV) Timing Analysis Mode

In on-chip variation (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. These delays are used together in the analysis of each check.

The 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.

OCV Timing Analysis Mode - Setup and Hold Checks

These are explained in the sections below.

Setup Check in OCV Mode

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 use one library instead of both maximum and minimum libraries, and apply timing derates for performing min/max analysis, respectively.

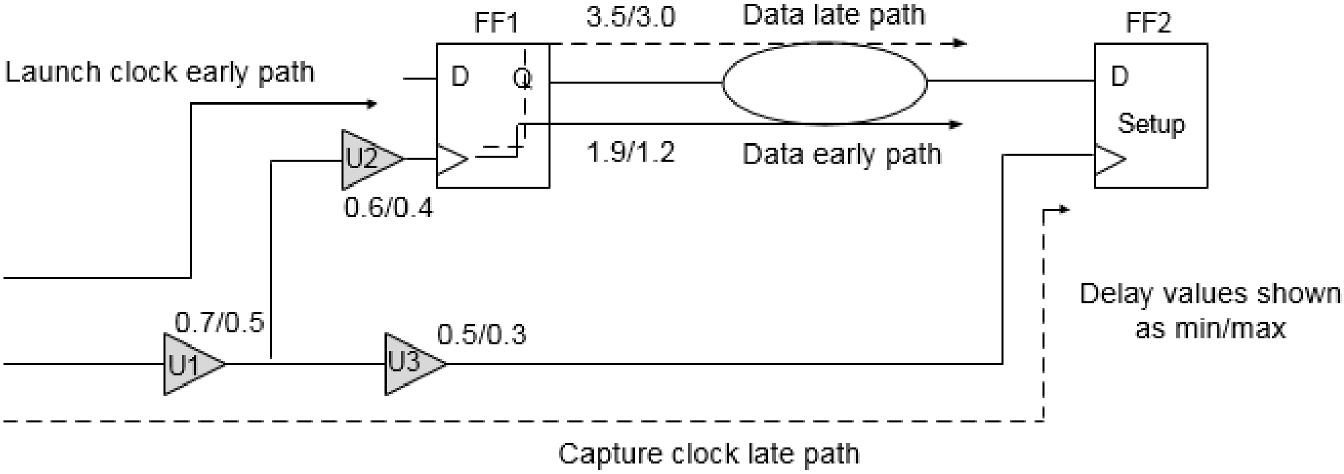
The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path (max)	`set_clock_latency -source -late -max` *value* Or, `set_clock_latency -source -late` *value*
Capture clock early path (min)	`set_clock_latency -source -early -min` *value* Or, `set_clock_latency -source -early` *value*

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path	`set_clock_latency -max` *value*
Capture clock early path	`set_clock_latency -min` *value*

Setup Check in OCV Mode



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 period = 4 ; Clock source latency = none ; Clock mode = Propagated

Hold Check in OCV Mode

For 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:

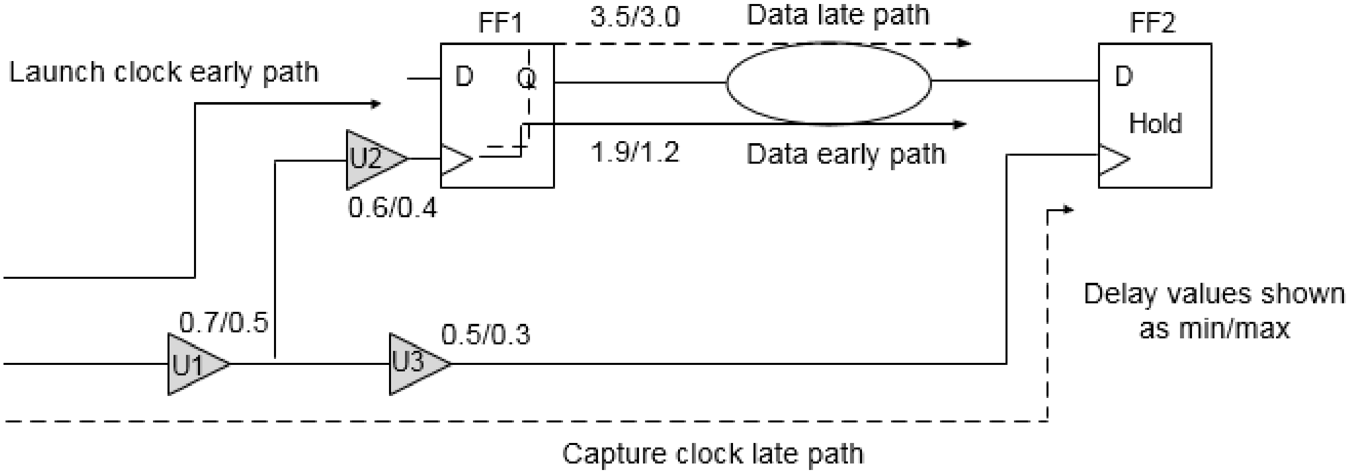
Clock Path (Operating Condition)	**Constraint Used**
Launch clock early path (min)	`set_clock_latency -source -early -min` *value* Or, `set_clock_latency -source -early` *value*
Capture clock late path (max)	`set_clock_latency -source -late -max` *value* Or, `set_clock_latency -source -late` *value*

The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock early path	`set_clock_latency -min` *value*
Capture clock late path	`set_clock_latency -max` *value*

The following example shows the hold check on the path from FF1 to FF2.

Hold Check in OCV Mode



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 period = 4 ; Clock source latency = none ; Clock mode = Propagated

Data early path delay (max)	
Data early path delay(min)	1.2
Launch clock early path delay (min)	0.5 + 0.4 = 0.9
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
Hold Slack	2.1 - 1.3 = 0.8

Performing Timing Analysis in OCV Mode

To perform Timing Analysis in OCV Mode, follow the steps given below:

1. Create a MMMC configuration file for OCV analysis.
   Note: 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.
2. 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
   ```
3. Load the design using the following commands:
   ```
   source init.globals
   init_design
   ```
4. Set the analysis mode to OCV and propagated clock mode.
   ```
   setAnalysisMode -analysisType onChipVariation -skew true  -clockProagation
   sdcControl
   ```
5. Generate the timing reports for setup.
   ```
   report_timing -late
   ```
6. Generate the timing reports for hold.
   ```
   report_timing -early
   ```

Using set_timing_derate with OCV Analysis Mode

When the `set_timing_derate` command is used, the following paths in OCV mode are affected:

Violations	Data	Launch Clock	Capture Clock
SETUP	`-late -data`	`-late -clock`	`-early -clock`
HOLD	`-early -data`	`-early -clock`	`-late -clock`

Clock Path Pessimism Removal

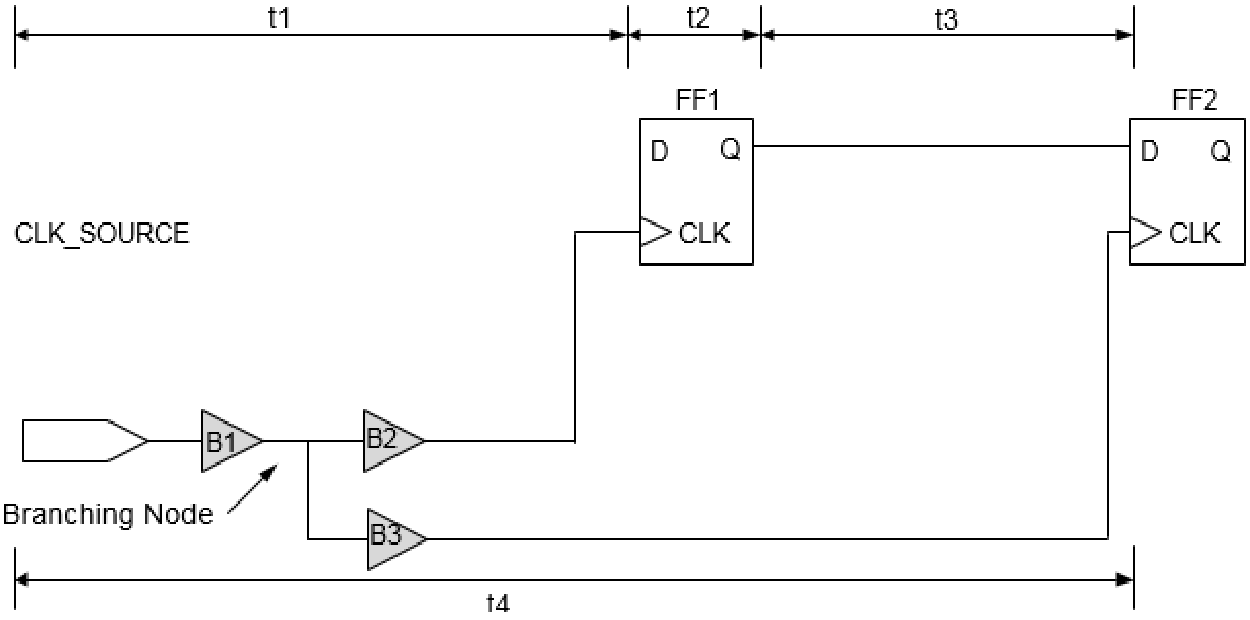
Clock Path Pessimism Removal (CPPR) is the process of identifying and removing pessimism introduced in slack reports for clock paths when launch and capture clock paths have a segment in common.

You can introduce early or late delay variations using the `setAnalysisMode -analysisType onChipVariation` command or by using the `set_timing_derate` for early and late clock paths. In CPPR calculations, the difference between late and early delays (for the common clock segment between launch and capture clock path) is calculated first and then this number is adjusted in the slack calculations to remove the pessimism, which existed because of considering common clock path to be both late and early at the same time. To remove this pessimism in propagated clock mode, you can use the following command:

```
setAnalysisMode -cppr true
```

Consider the following figure for setup check between flops FF1 and FF2.

Example Signal Path



Here,

$t_{1max} + t_{2max} + t_{3max} <= t_{4min} + t_{cp} - t_{su}$

where t_{cp} is the clock period and t_{su} is the setup requirement at D pin of flop FF2.

The above 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). Using the CPPR to remove this pessimism, the setup check equation is as follows:

$t_{1max} + t_{2max} + t_{3max} <= t_{4min} + t_{cp} - tsu + t_{cppr}$

where t_{cppr} is the difference in the maximum and minimum delay from the clock source to the branching node.

Similarly, hold check equation using CPPR is as follows:

$t_{1min} + t_{2min} + t_{3min} + t_{cppr} <= t_{4max} + t_H$

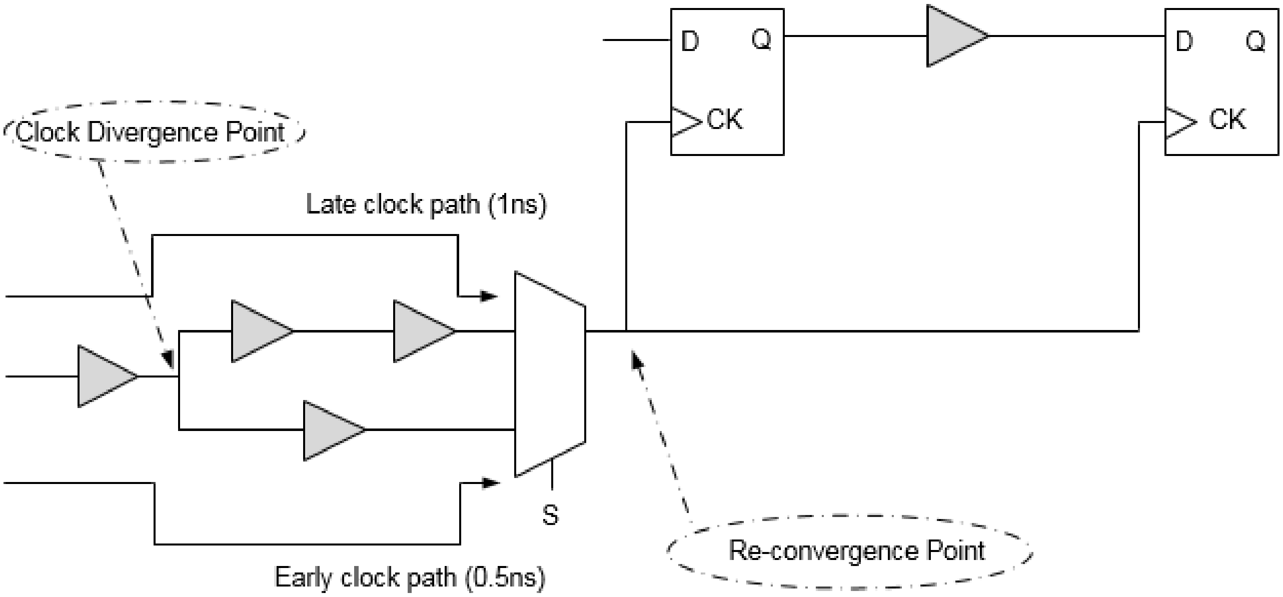
where t_H is the hold requirement at D pin of flop FF2.

Clock Reconvergence and CPPR

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.

Illustration of Clock Reconvergence



In this case, if the `set_case_analysis` command has not been set at point S of the multiplexer, the timing analysis software will assume different delay values for early and late paths. For example, if common early clock path from the clock source to the common clock point has a delay of 0.5ns, and the same common late clock path has delay of 1ns, then 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 methodologies. You can use CPPR to remove pessimism introduced due to reconvergence.

Innovus uses a default threshold of 20ps during pessimism removal. This means that 20ps of uncertainty remains in the analysis. To reset the threshold value, you can set `timing_cppr_threshold_ps`. When specified, all the paths might be reported without having their pessimism removed.

CPPR Flow

To remove pessimism, use the `setAnalysisMode` `-cppr` parameter. Follow the steps given below to support the CPPR feature:

1. Load the design.
2. Set the analysis mode to setup, propagated clock and CPPR:
   ```
   setAnalysisMode -checkType setup -skew true
   -clockPropagation sdcControl -cppr both
   ```
3. Set the derating values:
   ```
   set_timing_derate -late 1 -early 0.9 -clock
   ```
4. You use the `report_timing` command to remove delay pessimism from paths that have a portion of the clock network in common. To generate timing report, use the following command:
   ```
   report_timing
   ```

Supported CPPR Global Variables

In Innovus software, multiple global variables control CPPR behavior. These can be changed based on the use case requirements. Changing these global variables will cause changes in common clock path pessimism removal in terms of accuracy, common point selection, SI behavior, and so on.

Some of these global variables include:

- `timing_cppr_remove_clock_to_data_crp`
- `timing_cppr_self_loop_mode`
- `timing_cppr_skip_clock_reconvergence`
- `timing_cppr_threshold_ps`
- `timing_cppr_transition_sense`
- `timing_enable_pessimistic_cppr_for_reconvergent_clock_paths`
- `timing_enable_si_cppr`
- `timing_cppr_opposite_edge_mean_scale_factor`
- `timing_cppr_opposite_edge_sigma_scale_factor`

Innovus uses a default threshold of 20ps during pessimism removal. This means that 20ps of uncertainty remains in the analysis. To set 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 may be reported without having pessimism removed by the given value of the global variable. During SI analysis, the CPPR behavior for setup and hold checks can be controlled by setting `timing_enable_si_cppr` global variable.

Timing Analysis Results Before and After CPPR

The following example shows a full clock path timing report generated before CPPR analysis. Timing slack is 7.388 without any CPPR adjustments:

```
Path 1: MET Setup Check with Pin ff2/CK
Endpoint: ff2/D (v) checked with leading edge of 'clk'
Beginpoint: ff1/Q (v) triggered by leading edge of 'clk'
Path Groups: {clk}
Other End Arrival Time     0.972
- Setup                    0.335
+ Phase Shift             10.000
= Required Time           10.637
- Arrival Time             3.249
= Slack Time               7.388
Clock Rise Edge                0.000
= Beginpoint Arrival Time      0.000
Timing Path:
-------------------------------------------------------------------
Instance    Cell        Arc            Delay      Arrival   Required
                                                  Time      Time
-------------------------------------------------------------------
                        clk ^          0.000      7.388
c1          CLKBUFX1    A ^ -> Y ^     0.312      0.312      7.700
c2          CLKBUFX1    A ^ -> Y ^     0.343      0.655      8.042
c3          CLKBUFX1    A ^ -> Y ^     0.175      0.829      8.217
c4          CLKBUFX1    A ^ -> Y ^     0.126      0.956      8.343
c5          CLKBUFX1    A ^ -> Y ^     0.152      1.108      8.496
mux1        MX2X1       B ^ -> Y ^     0.211      1.319      8.706
cp11        CLKBUFX1    A ^ -> Y ^     0.146      1.465      8.852
cp12        CLKBUFX1    A ^ -> Y ^     0.131      1.596      8.983
cp13        CLKBUFX1    A ^ -> Y ^     0.157      1.752      9.140
```

```
ff1          DFFHQX1      CK ^ -> Q v     0.422       2.175       9.562
u2           BUFX1        A v -> Y v      0.309       2.483       9.871
u4           BUFX1        A v -> Y v      0.279       2.762       10.150
u5           BUFX1        A v -> Y v      0.258       3.020       10.408
u6           BUFX1        A v -> Y v      0.228       3.248       10.636
ff2          DFFHQX1      D v             0.001       3.249       10.637

-------------------------------------------------------------------
Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Other End Path:
--------------------------------------------------------------------
Instance     Cell          Arc           Delay      Arrival      Required
                                                    Time         Time
--------------------------------------------------------------------
                          clk ^                     0.000        -7.388
c1           CLKBUFX1     A ^ -> Y ^     0.124      0.124        -7.264
c2           CLKBUFX1     A ^ -> Y ^     0.160      0.284        -7.103
cp21         CLKBUFX1     A ^ -> Y ^     0.150      0.434        -6.954
cp22         CLKBUFX1     A ^ -> Y ^     0.133      0.566        -6.821
cp23         CLKBUFX1     A ^ -> Y ^     0.120      0.687        -6.701
cp24         CLKBUFX1     A ^ -> Y ^     0.112      0.799        -6.589
cp25         CLKBUFX1     A ^ -> Y ^     0.093      0.891        -6.496
cp26         CLKBUFX1     A ^ -> Y ^     0.081      0.972        -6.416
ff2          DFFHQX1      CK ^           0.000      0.972        -6.416
---------------------------------------------------------------------
```

When CPPR adjustment is made, timing slack improvement is seen as pessimism is removed by “CPPR Adjustment”. In the below example “c2” is common clock point after which clock diverges:

```
Path 1: MET Setup Check with Pin ff2/CK
Endpoint: ff2/D (v) checked with leading edge of 'clk'
Beginpoint: ff1/Q (v) triggered by leading edge of 'clk'
Path Groups: {clk}
Other End Arrival Time        0.972
- Setup                       0.335
+ Phase Shift                10.000
+ CPPR Adjustment             0.370
= Required Time              11.008
- Arrival Time                3.249
= Slack Time                  7.758
Clock Rise Edge                    0.000
= Beginpoint Arrival Time          0.000
Timing Path:
```

```
---------------------------------------------------------------------
Instance    Cell         Arc          Delay       Arrival     Required
                                                  Time        Time
---------------------------------------------------------------------
                         clk ^                    0.000       7.758
c1          CLKBUFX1     A ^ -> Y ^    0.312      0.312       8.070
c2          CLKBUFX1     A ^ -> Y ^    0.343      0.655       8.413
c3          CLKBUFX1     A ^ -> Y ^    0.175      0.829       8.588
c4          CLKBUFX1     A ^ -> Y ^    0.126      0.956       8.714
c5          CLKBUFX1     A ^ -> Y ^    0.152      1.108       8.866
mux1        MX2X1        B ^ -> Y ^    0.211      1.319       9.077
cp11        CLKBUFX1     A ^ -> Y ^    0.146      1.465       9.223
cp12        CLKBUFX1     A ^ -> Y ^    0.131      1.596       9.354
cp13        CLKBUFX1     A ^ -> Y ^    0.157      1.752       9.511
ff1         DFFHQX1      CK ^ -> Q v   0.422      2.175       9.933
u2          BUFX1        A v -> Y v    0.309      2.483       10.242
u4          BUFX1        A v -> Y v    0.279      2.762       10.520
u5          BUFX1        A v -> Y v    0.258      3.020       10.778
u6          BUFX1        A v -> Y v    0.228      3.248       11.007
ff2         DFFHQX1      D v           0.001      3.249       11.008
--------------------------------------------------------------------

Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Other End Path:
----------------------------------------------------------------------
Instance    Cell         Arc           Delay      Arrival     Required
                                                  Time        Time
---------------------------------------------------------------------
                         clk ^                    0.000       -7.758
c1          CLKBUFX1     A ^ -> Y ^    0.124      0.124       -7.634
c2          CLKBUFX1     A ^ -> Y ^    0.160      0.284       -7.474
cp21        CLKBUFX1     A ^ -> Y ^    0.150      0.434       -7.325
cp22        CLKBUFX1     A ^ -> Y ^    0.133      0.566       -7.192
cp23        CLKBUFX1     A ^ -> Y ^    0.120      0.687       -7.072
cp24        CLKBUFX1     A ^ -> Y ^    0.112      0.799       -6.960
cp25        CLKBUFX1     A ^ -> Y ^    0.093      0.891       -6.867
cp26        CLKBUFX1     A ^ -> Y ^    0.081      0.972       -6.786
ff2         DFFHQX1      CK ^          0.000      0.972       -6.786
--------------------------------------------------------------------
```

Analyzing Timing Problems

In addition to the detailed timing violation report, the following report commands are helpful in analyzing timing problems:

- `check_timing`

  Performs a variety of consistency and completeness checks on the timing constraints specified for a design. Use the check_timing command after setting all constraints, but before any timing analysis commands, such as `report_timing`, to verify that the timing environment is complete and self-consistent.

- `get_property`

  Retrieves timing information for the specified property on the given pin, net, timing arc, or clock.

- `report_analysis_coverage`

  Provides information about the timing checks in the design.

- `report_annotated_check`

  Reports coverage of annotated timing checks.

- `report_annotated_delay`

  Reports SDF design annotations coverage.

- `report_annotated_parasitics`

  Reports the back-annotated parasitics of the design.

- `report_case_analysis`

  Reports ports and pins with `set_case_analysis` constraint.

- `report_cell_instance_timing`

  Reports instance pin and delay arc timing information.

- `report_clock_timing`

  Generates a clock skew report for the current design.

- `report_clocks`

  Reports clock waveform, clock arrival point and clock uncertainty information.

- `report_inactive_arcs`

  Reports all disabled timing arcs and checks.

- `report_path_exceptions`

Reports design path exceptions such as `set_false_path`, `set_multicycle_path`, `set_max_delay`, and `set_min_delay`.

- `report_timing`

  Generates a timing report that provides information about the various paths in the design. The report typically contains data on the delay through the entire path. The start node and the end node of each path is identified.

- `report_constraint`

  Reports constraint information of current design.

- `report_fanin`

  Allows a cone traversal that is not tied to the timing graph, that is, not blocked by `set_disable_timing` and case analysis.

- `report_fanout`

  Allows a cone traversal that is not tied to the timing graph, that is, not blocked by `set_disable_timing` and case analysis.

- `timeDesign`

  Performs routing, extraction, timing analysis and generates detailed timing reports.

Resolving Buffer-Related Problems

You may encounter some of the following buffer-related problems when running timing analysis:

- The logical cell or buffer equivalence, based on cell functionality not used during timing optimization, can cause timing optimization to ignore timing violations.
- If an incorrect buffer footprint name was entered for the set of buffers to run timing optimization, use the `reportFootPrint` command to list the current footprint information.
- If the `in_place_swap_mode:match_footprint` statement is in the timing library, then timing optimization matches up all the cells with same `cell_footprint` name, and logical cell or buffer equivalence will not be used.
- If the `in_place_swap_mode:match_footprint` statement does not exist, then timing optimization derives logical cell equivalence based on matching `function:`*`boolean_eq`*.
- If you want the logical cell equivalence based on matching, comment out the `in_place_swap_mode:match_footprint` statement in the timing library.

Debugging Timing Results

Overview

Innovus provides the Global Timing Debug feature for debugging the timing results. The various Timing Debug forms provide easy visual access to the timing reports and debugging tools.

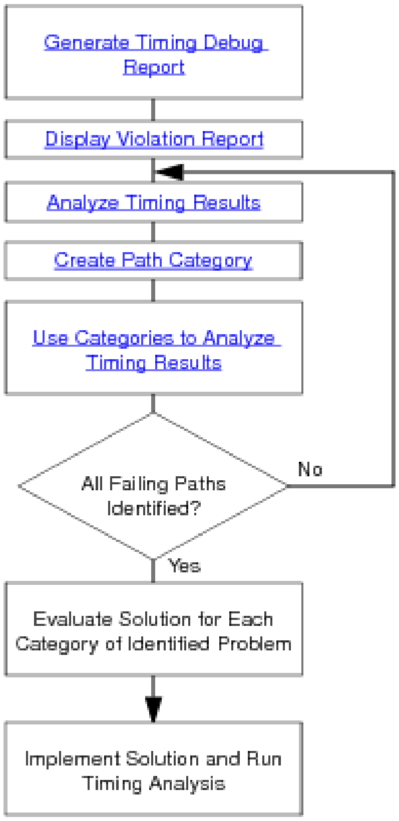
You can group all paths that are failing for the same reason and apply solutions for faster timing closure. You can cross-probe between the timing paths in the timing report and display area in the Layout window.

Note: If you have a previously saved timing debug report, you can use the timing debug feature even when the design is not loaded in the Innovus session.

Timing Debug Flow

You can generate a detailed violation report to list the details of all violating paths. You can then use the timing debug capability to visually identify problems with critical paths in this report. After identifying the problems, you group all paths with the same problem under a single category. You can define several categories to capture all problems related to the violating paths before fixing the problems and running timing analysis again.

Following is the flow for debugging timing results.


Generate Timing Debug Report
Display Violation Report
Analyze Timing Results
Create Path Category
Use Categories to Analyze Timing Results
All Failing Paths Identified?
No
Yes
Evaluate Solution for Each Category of Identified Problem
Implement Solution and Run Timing Analysis

Note: The `gtd.pref.tcl` file is loaded automatically at launch.

Generating Timing Debug Report

Innovus uses a machine readable timing report to display timing debug information. The report is generated in the ASCII format and contains details of all violating paths. By default, the report has `.mtarpt` extension.

To generate a violation report, use one of the following options, use the `report_timing` command.

You can also generate text-format report from a machine readable report.

To generate the text report, use the following:

- Use the `write_text_timing_report` command.
- Use the *Write Textual Timing Report* form

Displaying Violation Report

To analyze the timing results, you need to load the machine readable timing report in Innovus.

To display the violation report, use one of the following options:

- Specify the file name in the *Display/Generate Timing Report* form.

  Note: To select an existing file, deselect the *Generate* option before clicking on the directory icon to the right of the *Timing Report File* field.
  By default, the global timing debug engine uses the following command to generate a machine-readable timing analysis report for the GUI display:

  ```
  report_timing -machine_readable -max_points 10000 -max_slack 0.75
  ```

- Use the `load_timing_debug_report` command.

Note: Use the *Append to Current Repor*t option in the *Display/Generate Timing Report* form to load multiple reports in a single session.

Analyzing Timing Results

Innovus provides Timing Debug feature to visually analyze timing problems.

You can analyze the following data in the *Timing Debug* form:

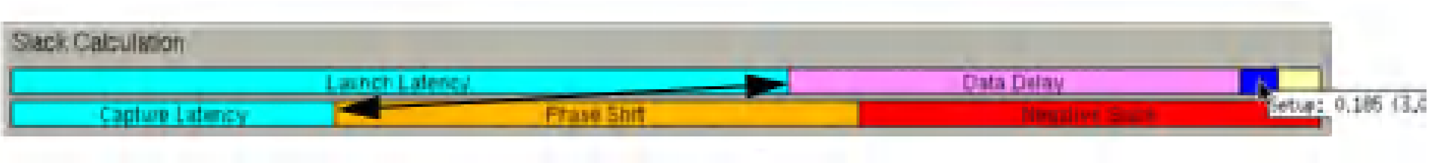
- Visual display of passing and failing paths as a histogram. Failing paths are represented in red and passing paths are represented in green color. The goal of timing debug process is to identify paths that fall in red category.
- Details of the critical paths in the Path list. You identify a critical path in this list for further analyses using the Timing Path Analyzer form.
- Visual display of paths reported in different timing reports. When you load multiple debug reports in a single timing debug session, the paths are displayed in different colors corresponding to the report file they are coming from. You can move the cursor over a path to display the name of the report file.

You analyze the following data in the *Timing Path Analyzer* form:

- Slack calculation bars for arrival and required times. You can identify clock skew issues, latency balancing or large clock uncertainty issues using these bars.

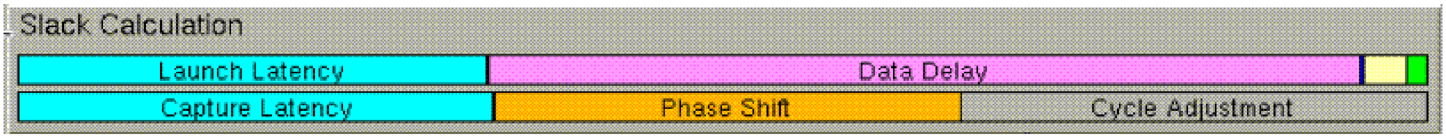
The following examples illustrate the problems that you can identify using the slack calculation bars in the *Timing Path Analyzer* form.

Example 9-1



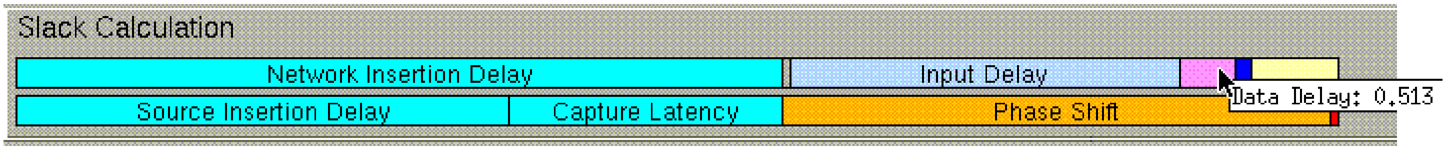
Launch and capture latency components are not aligned. Therefore there can be large clock-latency mismatch in this path.

Example 9-2



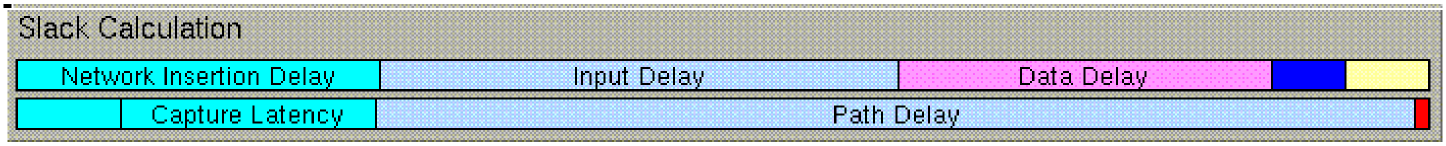
The cycle adjustment bar in the required time indicates presence of multicycle path.

Example 9-3



Large input delay in an I/O path is represented by the blue bar in the arrival time.

Example 9-4



Path Delay bar in the required time indicates a `set_max_delay` constraint.

- Path details including launch and capture. This information is provided as tabs in the Timing Path Analyzer form. You can click on a single path to display it in the design display area. You can select each element consecutively to trace the entire path in the design display area. This form has a Status column that indicates the status of the path as follows:

Flag	Description
a	Assign net

b	Blackbox instance
c	Clock net
cr	Cover cell
f	Preplaced instance
i	Ignore net
s	Skip route net
t	"don't touch" net
t	instance marked as "don't touch"
T	instance not marked as "don't touch" when the module it belongs to is marked as "don't touch"
u	Unplaced cell
x	External net

- SDC related to the path. The Path SDC tab displays the SDC constraints related to the selected path. The list contains the name of the SDC file, the line number that indicates the position of the constraint in the SDC file, and the constraint definition.

The commands that can be displayed in the Path SDC tab are:

- `create_clock`
- `create_generated_clock`
- `group_path`
- `set_multicycle_path`
- `set_false_path`
- `set_clock_transition`
- `set_max_delay`
- `set_min_delay`
- `set_max_fanout`
- `set_fanout_load`

- `set_min_capacitance`
- `set_max_capacitance`
- `set_min_transition`
- `set_max_transition`
- `set_input_transition`
- `set_drive`
- `set_driving_cell`
- `set_logic_one`
- `set_logic_zero`
- `set_dont_use`
- `set_dont_touch`
- `set_case_analysis`
- `set_input_delay`
- `set_output_delay`
- `set_annotated_check`
- `set_clock_uncertainty`
- `set_clock_latency`
- `set_propagated_clock`
- `set_load`
- `set_disable_clock_gating_check`
- `set_clock_gating_check`
- `set_max_time_borrow`
- `set_clock_groups`

When you create a constraint on the command line, the Path SDC tab interactively displays the result of the additional constraint.

Note: MMMC views are not displayed interactively.

Note: You can create a path category directly from SDC constraints in the Path SDC form. When you right-click a constraint and view the *Create Path Category* form, to see the line number (from the SDC file) and the name of the constraint.

In Innovus, you can also ceate a path category based on SDC constraint using the `-sdc` parameter of the `create_path_category` command:

```
create_path_category -name category_name -sdc {file_name line_number}
```

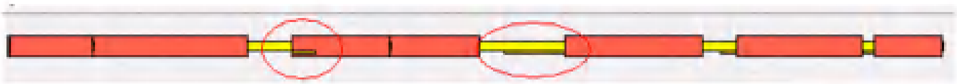
where `file_name` is the name of the constraint file and line_number is the line number of the SDC constraint.

- Schematic display of the path. The Schematics tab displays the gate-level schematic view of the critical path. For more information, see Viewing Schematics.
- Timing interpretation for the path. This feature provides rule-based path analysis to help you discover sources of potential timing problems in a path. By default, the software performs the following checks on the following rules:
  - Path structure
    - Transparent Latch in Path
    - Clock Gating
    - Hard Macros
    - HVT Cells
    - Buffering List
    - Net Fanout
    - Level Shifters
    - Isolation Cells
    - Net too long
    - Couple capacitance ratio
  - Timing and constraints
    - Large Skew
    - Divider in Clock Path
    - Total SI Delay
    - SI Delay

        - External Delay
    - Floorplan
        - Fixed Cells
        - Distance from start to end
        - Distance of repeater chain
        - Detour
        - Multiple power domains
    - DRVs
        - Max transitions
        - Max capacitance
        - Max fanout

You can customize the type of timing information reported. The *Edit Timing Interpretation* GUI Innovus lets you add, modify, or delete rules you want the tool to check and report.

- Timing bar to analyze delays associated with instances and nets in a path. Use this information to identify issues related to large instance or net delays, repeater chains, paths with large number of buffers, and large macro delays. The small bars superimposed on net delays or within element delays show incremental (longer or shorter) delays due to noise effects:

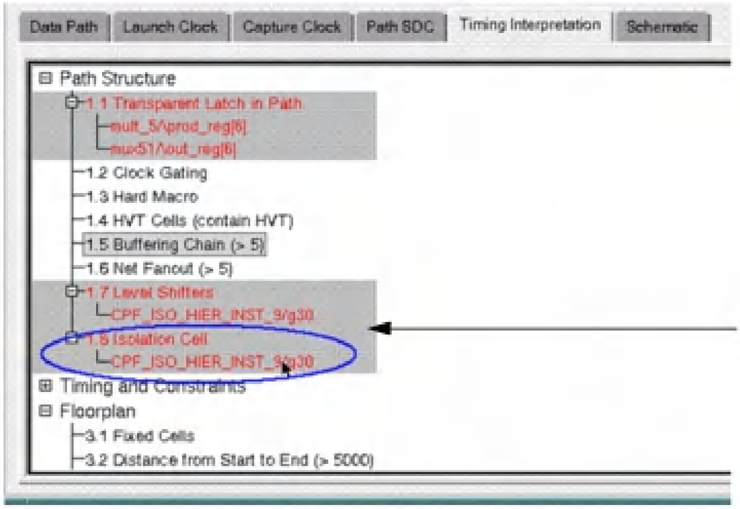


- Hierarchical representation of the path in the Hierarchy View field. This representation of the path-delay shows the traversal of a path through the design hierarchy drawn on the time axis. A longer arrow means that there are more instances on its path. Use this information to see the module where the path is spending more time or to identify inter-partition timing problems.

Viewing Power Domain Information

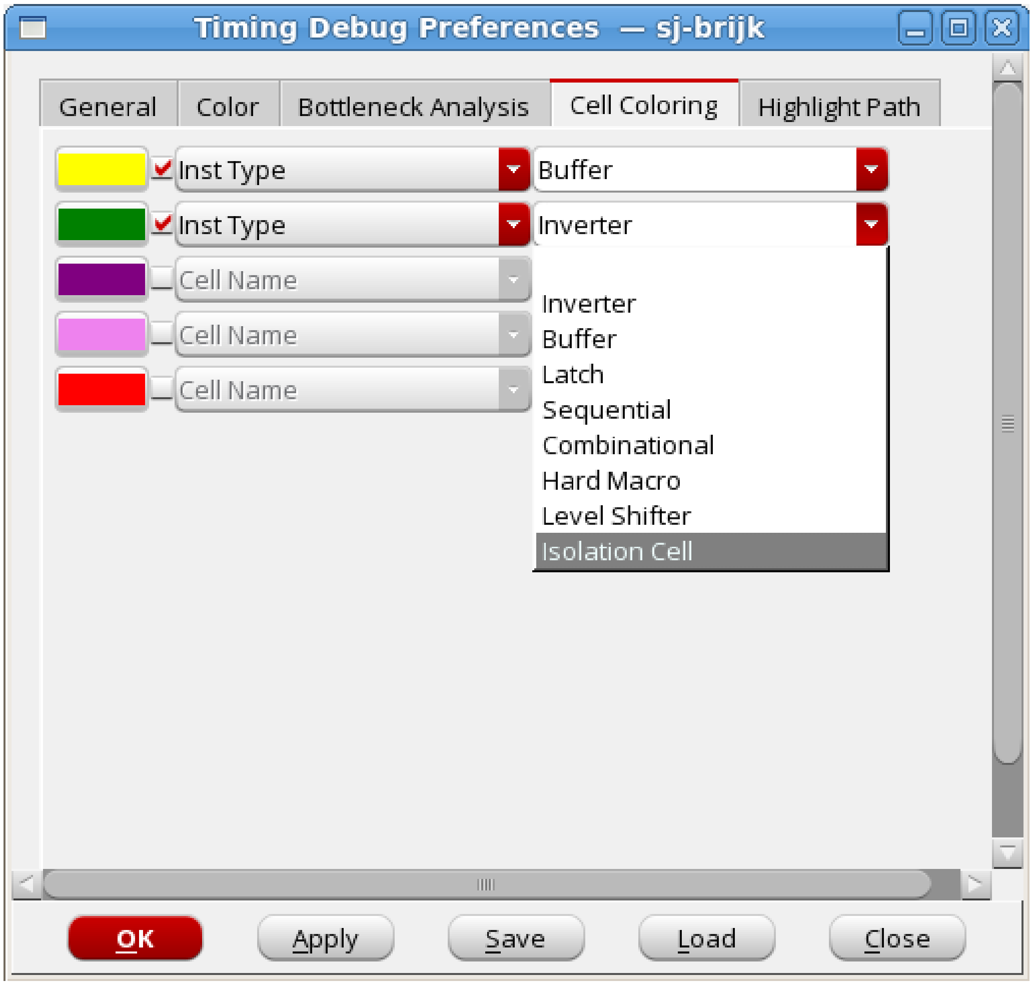
While debugging critical paths on MSV designs, it is useful to be able to identify power domains and low power cells. The Timing Debug feature displays this information in the following ways:

- The level shifters and isolation cells are listed in the Timing Interpretation tab of the Timing Path Analyzer.



Level shifters and
Isolation Cells are
displayed in the
Timing
Implementation tab

- 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.



🟨	☑	Inst Type	Buffer
🟩	☑	Inst Type	Inverter
🟪	□	Cell Name	
▨	□	Cell Name	
🟥	□	Cell Name	

Inverter
Buffer
Latch
Sequential
Combinational
Hard Macro
Level Shifter
Isolation Cell

Creating Path Categories

After analyzing the paths in the timing report, you identify problems in various paths. Then you create a group of paths such that all paths in that group have the same timing problem and can be fixed at the same time. In timing debug such a group of paths is called a category. In Innovus, you can either define your own category or use predefined categories to group your paths. The categories that you define are then displayed in the *Path Category* field in the *Analysis* tab. The form also displays the paths associated with each category.

Creating Predefined Categories

There are following predefined categories:

- Basic Path Group
  Creates standard path categories according to basic path groups.

  The basic path groups are:

  - Register to register
  - Input to register
  - Register to output
  - Input to Output

  Registers can be macros, latches, or sequential-celltypes. To create categories by basic path groups, choose the *Analysis* tab - *Analysis* drop-down menu - *Path Group Analysis* option.

- Clock Paths
  Creates categories according to launch clock - capture clock combinations.

  Following categories are created:

  - Paths with clock fully contained in a single domain, clk1.
  - Paths with clocks starting one clock domain and ending at another, clk1->clk2.

  To create categories for clock paths, choose the *Analysis* tab - *Analysis* drop-down menu - *Clock Analysis* option.

- Hierarchical Floorplan
  Creates categories according to the hierarchical characteristics of a path.

- View

  Creates categories according to the view for which the path was generated.

  To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *View Analysis* option.

- False Paths

  Creates a category with paths defined as False paths.

  To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *Critical False Path* option.

- Bottleneck

  Creates categories based on instances that occur often in critical paths.

  To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *Bottleneck Analysis* option.

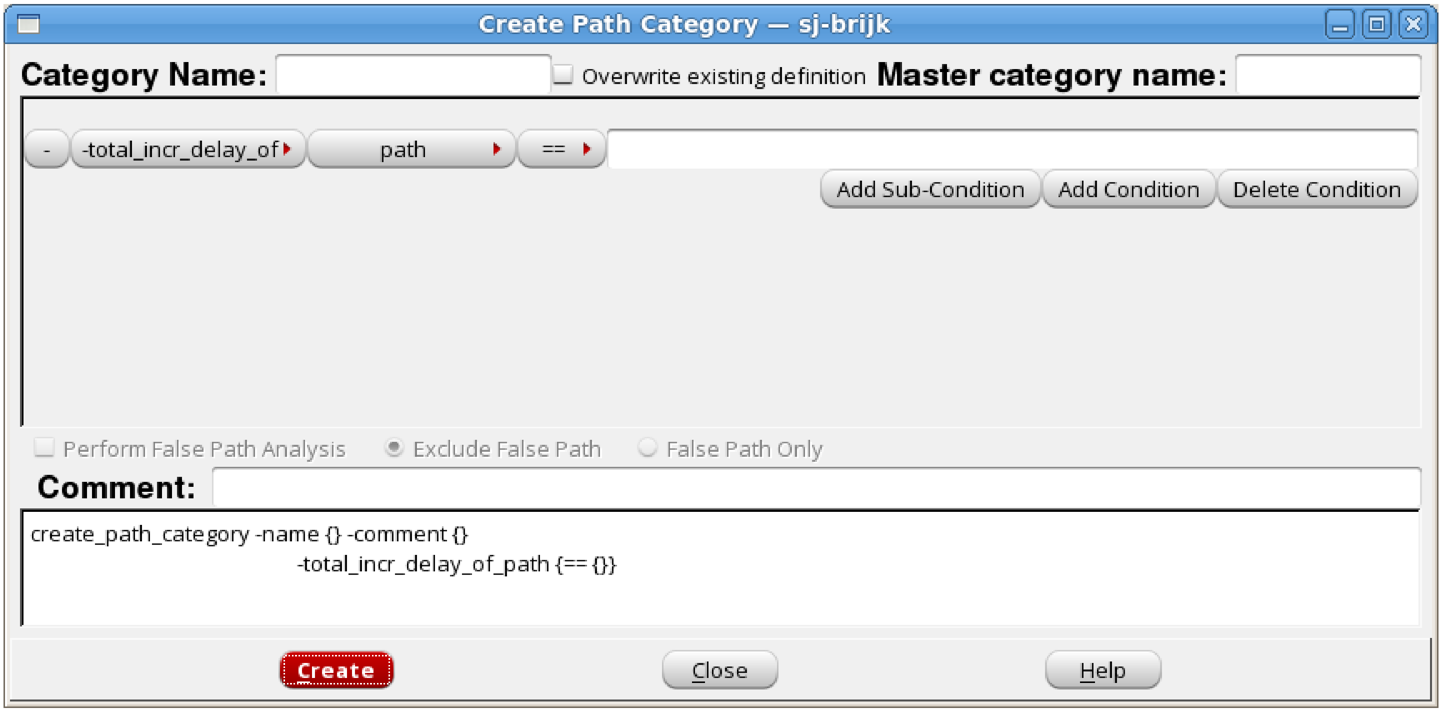
- DRV Analysis

  Generates or loads a DRV report containing capacitance, transition, or fanout violations. Paths that are affected by the selected DRV types are grouped in a category.

  To create or load this report, choose the Analysis tab - Analysis drop-down menu - DRV Analysis option.

Creating New Categories

To define a new category, use the *Create Path Category* form. The Create Path Category form contains drop-down menus with conditions that you use to define a path category. The conditions are characteristics that a path must have to be added to the named category. You can define multiple conditions that a path must meet to be added to the category.



```
create_path_category -name {} -comment {}
                          -total_incr_delay_of_path {== {}}
```

The category that you create is added in the *Path Category* field in the *Analysis* tab. All paths that meet the conditions set for this category are grouped under the category name. Paths are separated automatically according to MMMC views into different categories, for example:

```
CLOCK1<View_test_mode>
CLOCK2<View_mission_mode>
```

- Double-click on the category name in the *Path Category* field in the *Analysis* tab to display the list of paths in the *Path List* field.

Note: You can add a comment in the *Comment* field to record any notes that you would like to include with the category. The comment appears in the category report file.

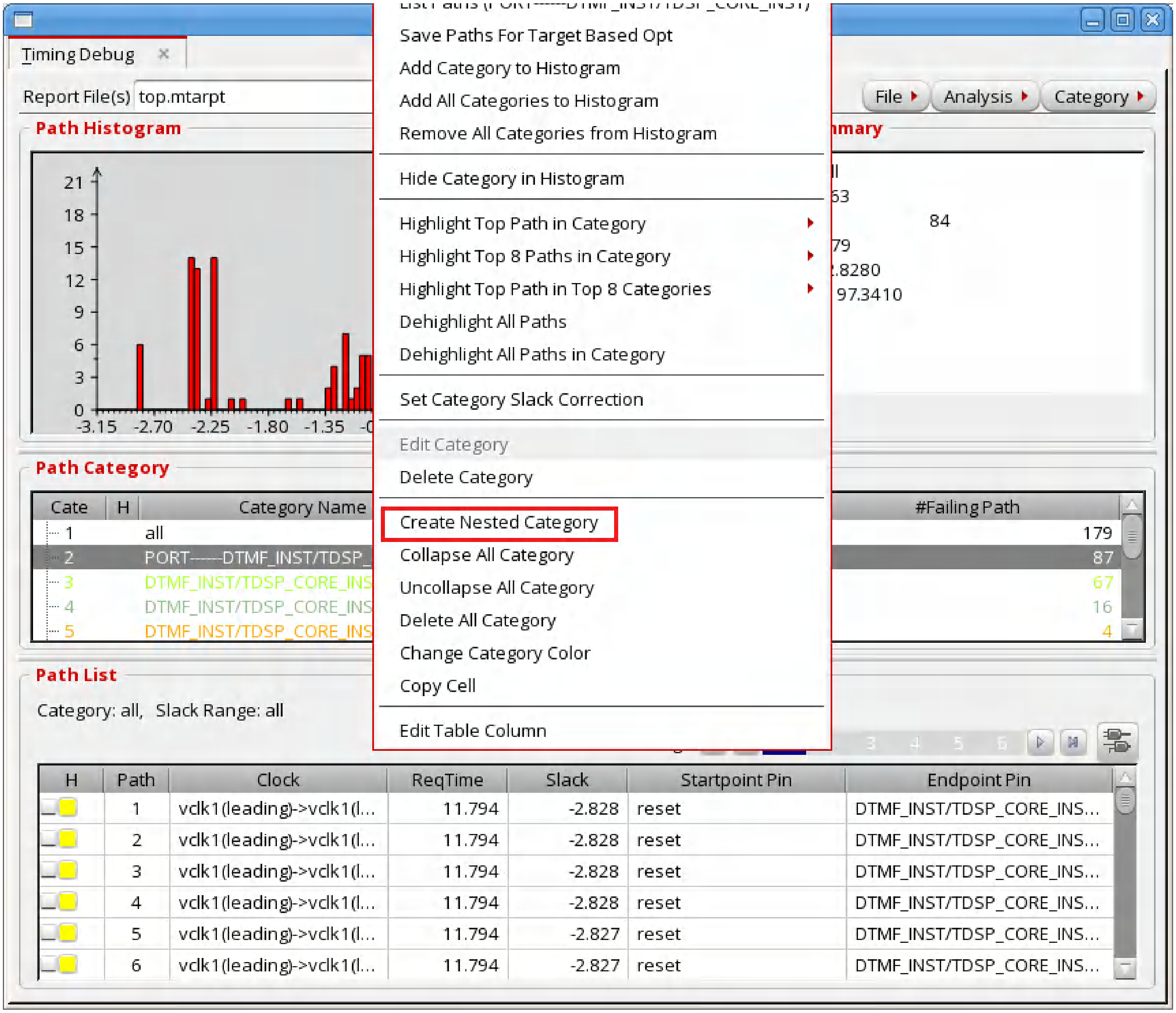
Creating Sub-Categories

You can create sub-categories based on existing categories. While analyzing a sub-category, global timing debug will traverse the paths in the master category instead of all the paths in the current report.

- Creating Sub-Categories through the GUI
- Creating Sub-Categories through Command Line

Creating Sub-Categories through the GUI

- In the GUI, you can create a sub category as follows:
- Right-click a path category, and select *Nested Category*.



Timing Debug
File
Analysis
Category
Path Histogram
21
18
15
12
9
6
3
0
-3.15
-2.70
-2.25
-1.80
-1.35
mary
Path Category
Path List
List Paths (PORT-----DTMF_INST/TDSP_CORE_INST)
Save Paths For Target Based Opt
Add Category to Histogram
Add All Categories to Histogram
Remove All Categories from Histogram
Hide Category in Histogram
Highlight Top Path in Category
Highlight Top 8 Paths in Category
Highlight Top Path in Top 8 Categories
Dehighlight All Paths
Dehighlight All Paths in Category
Set Category Slack Correction
Edit Category
Delete Category
Create Nested Category
Collapse All Category
Uncollapse All Category
Delete All Category
Change Category Color
Copy Cell
Edit Table Column

Report File(s) top.mtarpt

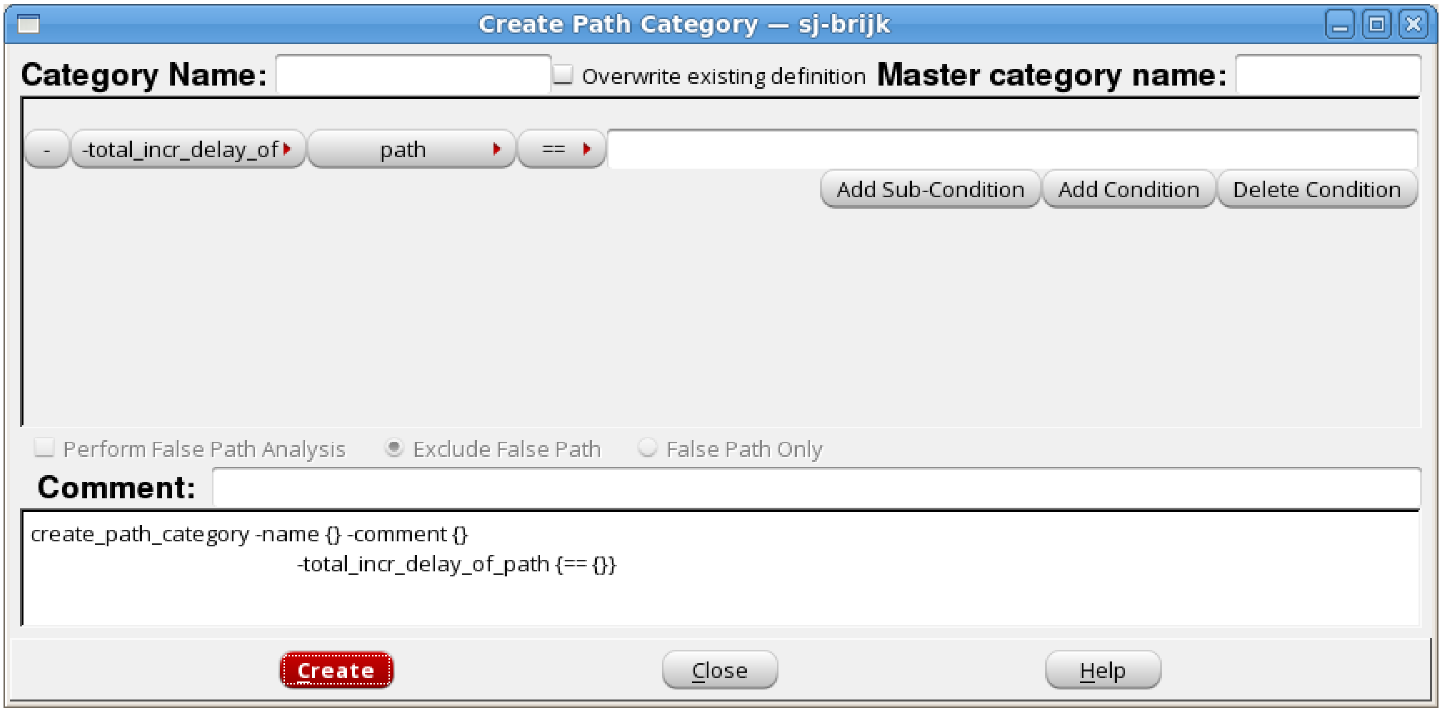
ll
63
84
79
.8280
97.3410

Cate	H	Category Name	#Failing Path
1		all	179
2		PORT-----DTMF_INST/TDSP_	87
3		DTMF_INST/TDSP_CORE_INS	67
4		DTMF_INST/TDSP_CORE_INS	16
5		DTMF_INST/TDSP_CORE_INS	4

Category: all, Slack Range: all

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
□ 🟨	1	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	2	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	3	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	4	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	5	vclk1(leading)->vclk1(l...	11.794	-2.827	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	6	vclk1(leading)->vclk1(l...	11.794	-2.827	reset	DTMF_INST/TDSP_CORE_INS...

The *Create Path Category* form is displayed.



```
create_path_category -name {} -comment {}
                     -total_incr_delay_of_path {== {}}
```

- In the *Master category name* field, the name of the category you selected previously is displayed. Create one or more subcategories as explained in Creating Path Categories.

Note: You can create nested sub-categories, that is, you can further create sub-categories for a sub-category.

You can also use the Category - Create menu command to bring up the Create Path Category form. In the *Master category name* field, type the name of the category for which you want to create the sub-category, and then create one or more subcategories as explained in Creating Path Categories.

Creating Sub-Categories through Command Line

Use the -master parameter of the `create_path_category` command to create sub-categories. The category created will be a sub-category of the category name specified with the -master parameter.

The following other commands also support sub-categories; to run these commands only on the sub-categories of a particular master category, specify the master category name with the -master parameter.

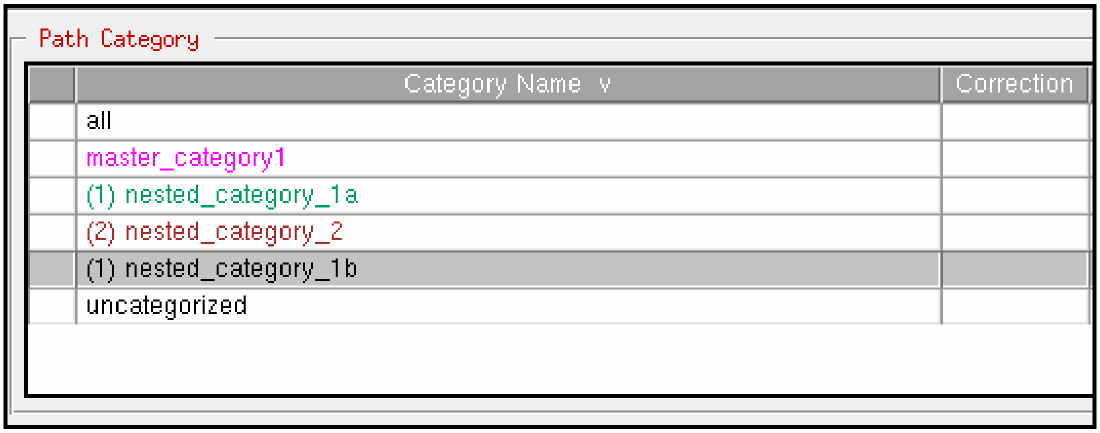
- `analyze_paths_by_basic_path_group`
- `analyze_paths_by_bottleneck`
- `analyze_paths_by_clock_domain`

- `analyze_paths_by_critical_false_path`
- `analyze_paths_by_drv`
- `analyze_paths_by_hierarchy`
- `analyze_paths_by_view`

Note: If the parent category of a sub-category is deleted, the sub-category cannot be edited or changed anymore. However, the sub-category is still displayed in case you want to refer to it.

Viewing Sub-Categories

The subcategories for a master category are displayed in a hierarchically numbered list below the master category. As an illustration, consider the example shown here:



Path Category

	Category Name v	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

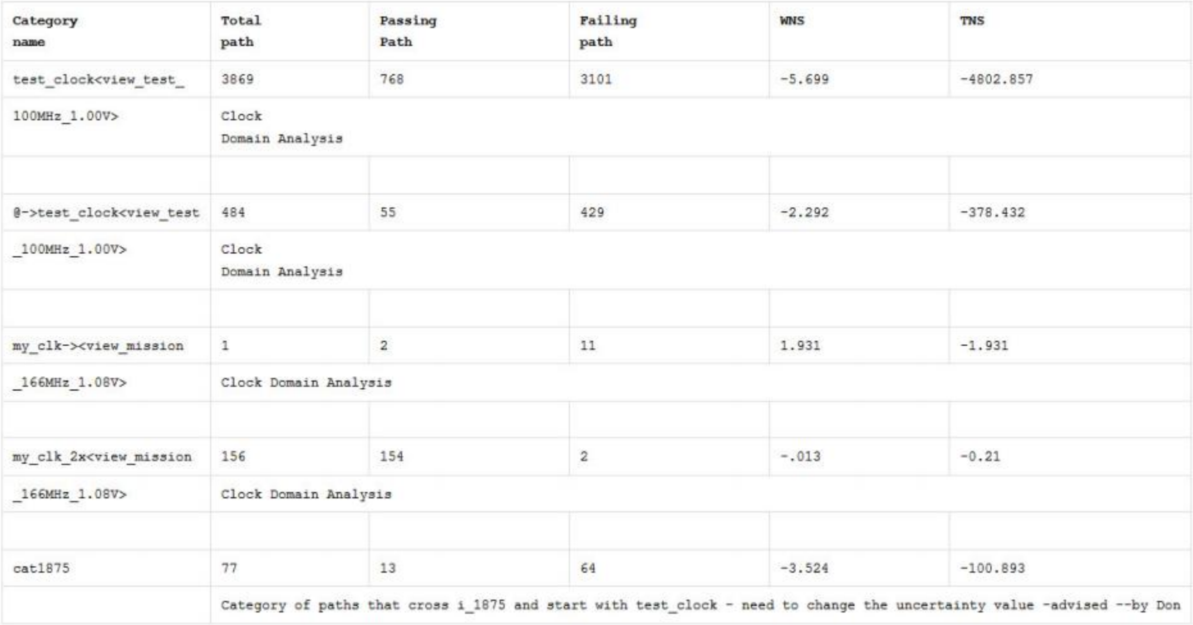
To generate a report containing information about path categories, use the following options:

- The `write_category_summary` command
- The Write Category Report File GUI

The text file contains the following information:

- Category name
- Total number of paths
- Number of passing paths
- Number of failing paths
- Worst negative slack
- Total negative slack
- TNS

Sample report:



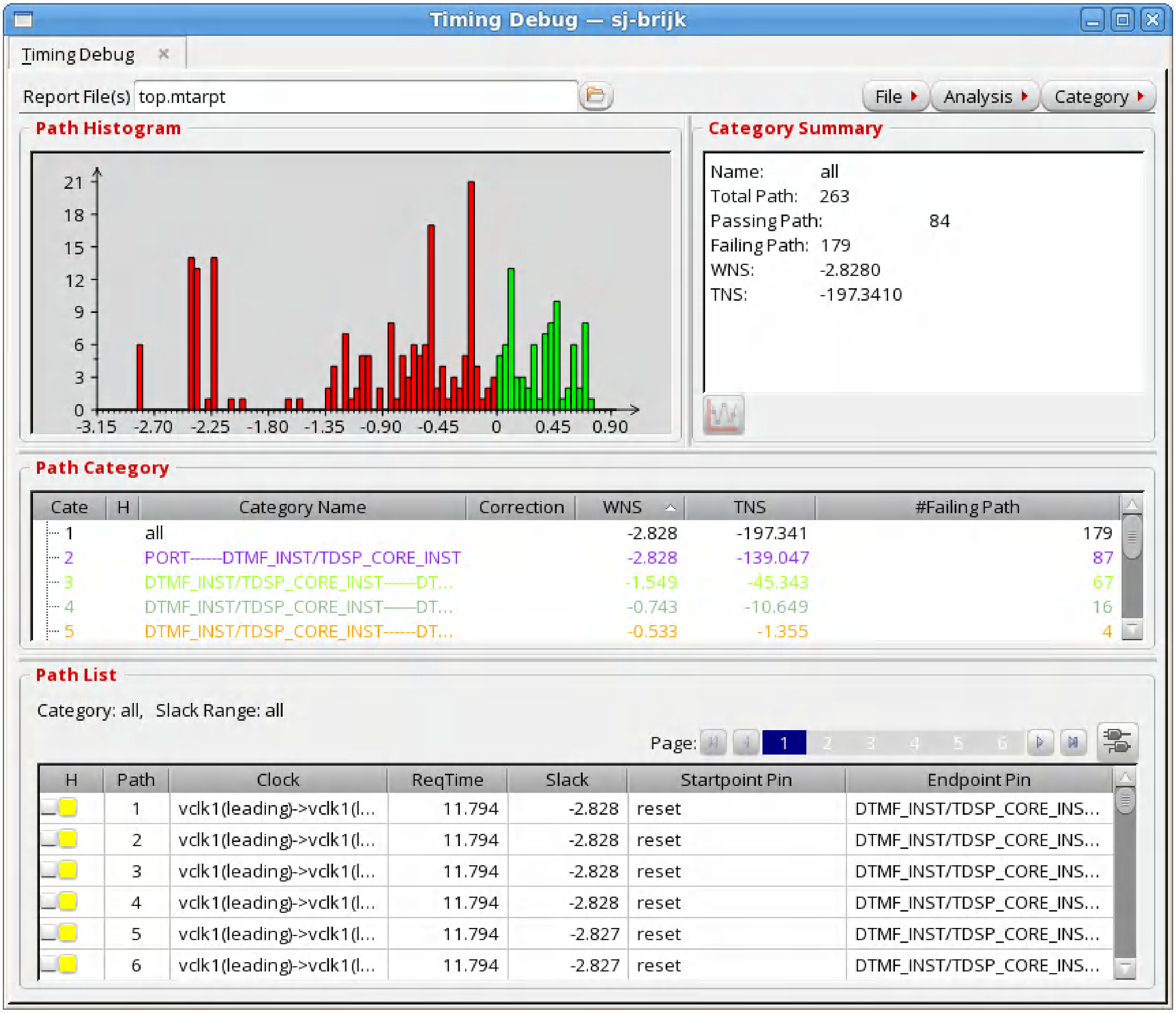
Category name	Total path	Passing Path	Failing path	WNS	TNS
test_clock<view_test_	3869	768	3101	-5.699	-4802.857
100MHz_1.00V>	Clock Domain Analysis				
@->test_clock<view_test	484	55	429	-2.292	-378.432
_100MHz_1.00V>	Clock Domain Analysis				
my_clk-><view_mission	1	2	11	1.931	-1.931
_166MHz_1.08V>	Clock Domain Analysis				
my_clk_2x<view_mission	156	154	2	-.013	-0.21
_166MHz_1.08V>	Clock Domain Analysis				
cat1875	77	13	64	-3.524	-100.893
	Category of paths that cross i_1875 and start with test_clock - need to change the uncertainty value -advised --by Don				

Using Categories to Analyze Timing Results

You can use the categories that you create to group the timing paths, and display them as histogram in the *Analysis* tab. The Analysis tab displays the category details in the Path Category field. You can perform the following tasks in the *Analysis* tab to analyze the timing results:

- Double-click on any category to display the details of the paths grouped in that category in the *Path List* field.
- Right-click on the category name and select *Add to Histogram* option. The paths related to the selected category are highlighted in a different color in the histogram. This gives you a visual representation of the number of paths that meet the conditions in that category and can possibly have the same timing problem.

For example, in the following figure the new category was added to the histogram.



Report File(s) top.mtarpt

Name: all
Total Path: 263
Passing Path: 84
Failing Path: 179
WNS: -2.8280
TNS: -197.3410

Cate	H	Category Name	Correction	WNS	TNS	#Failing Path
1		all		-2.828	-197.341	179
2		PORT------DTMF_INST/TDSP_CORE_INST		-2.828	-139.047	87
3		DTMF_INST/TDSP_CORE_INST------DT...		-1.549	-45.343	67
4		DTMF_INST/TDSP_CORE_INST------DT...		-0.743	-10.649	16
5		DTMF_INST/TDSP_CORE_INST------DT...		-0.533	-1.355	4

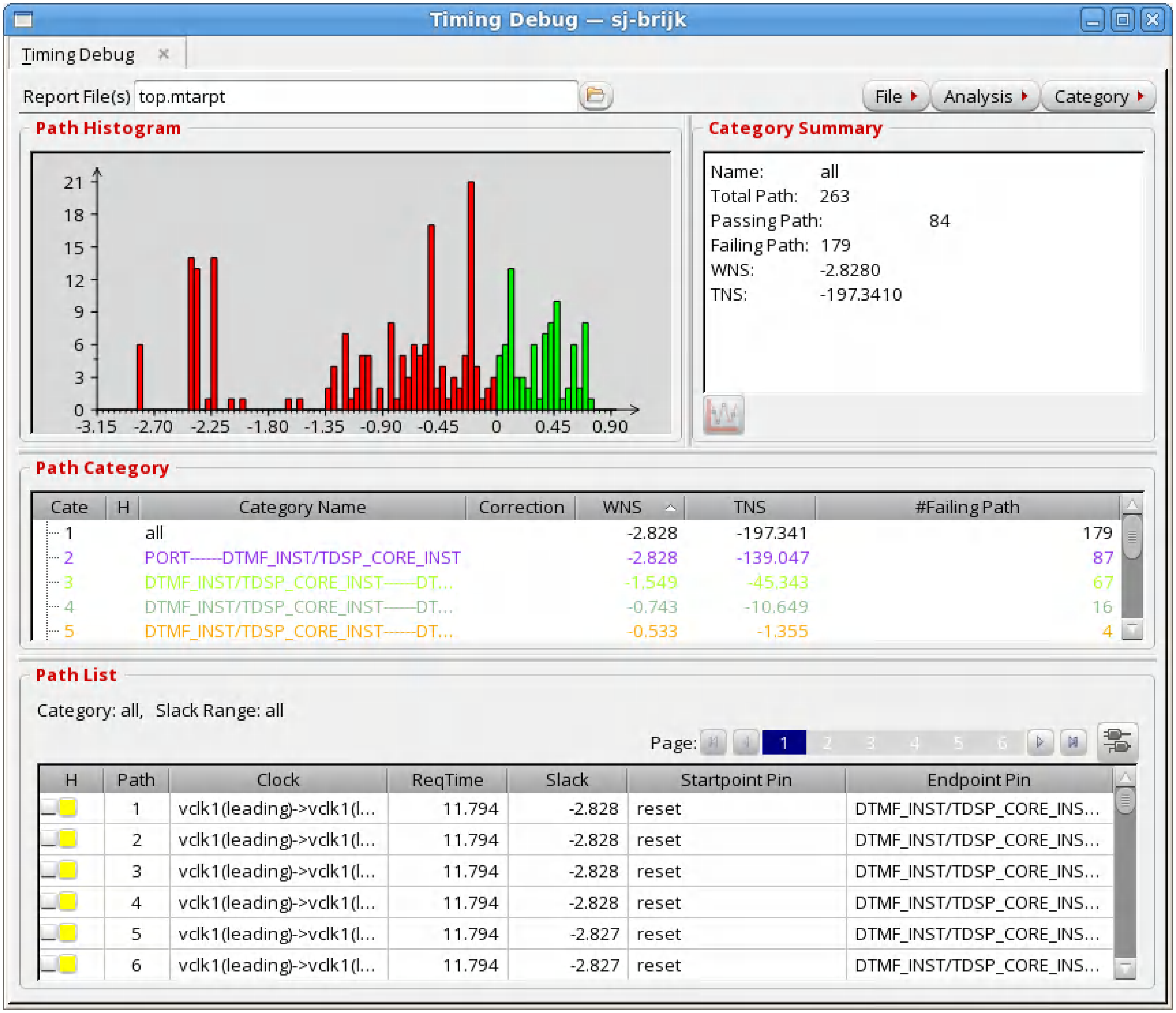
Category: all, Slack Range: all

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
□ 🟨	1	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	2	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	3	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	4	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	5	vclk1(leading)->vclk1(l...	11.794	-2.827	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	6	vclk1(leading)->vclk1(l...	11.794	-2.827	reset	DTMF_INST/TDSP_CORE_INS...

Analyzing the *Analysis* tab gives you information for fixing problems related to larger sets of timing paths. After identifying the problems, you can make the required changes such as modify floorplan, script or SDC files and run timing analysis again for further analysis.

Analyzing MMMC Categories

Paths are separated automatically according to MMMC views into different categories, for example, the following figure shows multiple categories based on MMMC views.



Report File(s) top.mtarpt

Name: all
Total Path: 263
Passing Path: 84
Failing Path: 179
WNS: -2.8280
TNS: -197.3410

Cate	H	Category Name	Correction	WNS	TNS	#Failing Path
1		all		-2.828	-197.341	179
2		PORT------DTMF_INST/TDSP_CORE_INST		-2.828	-139.047	87
3		DTMF_INST/TDSP_CORE_INST------DT...		-1.549	-45.343	67
4		DTMF_INST/TDSP_CORE_INST------DT...		-0.743	-10.649	16
5		DTMF_INST/TDSP_CORE_INST------DT...		-0.533	-1.355	4

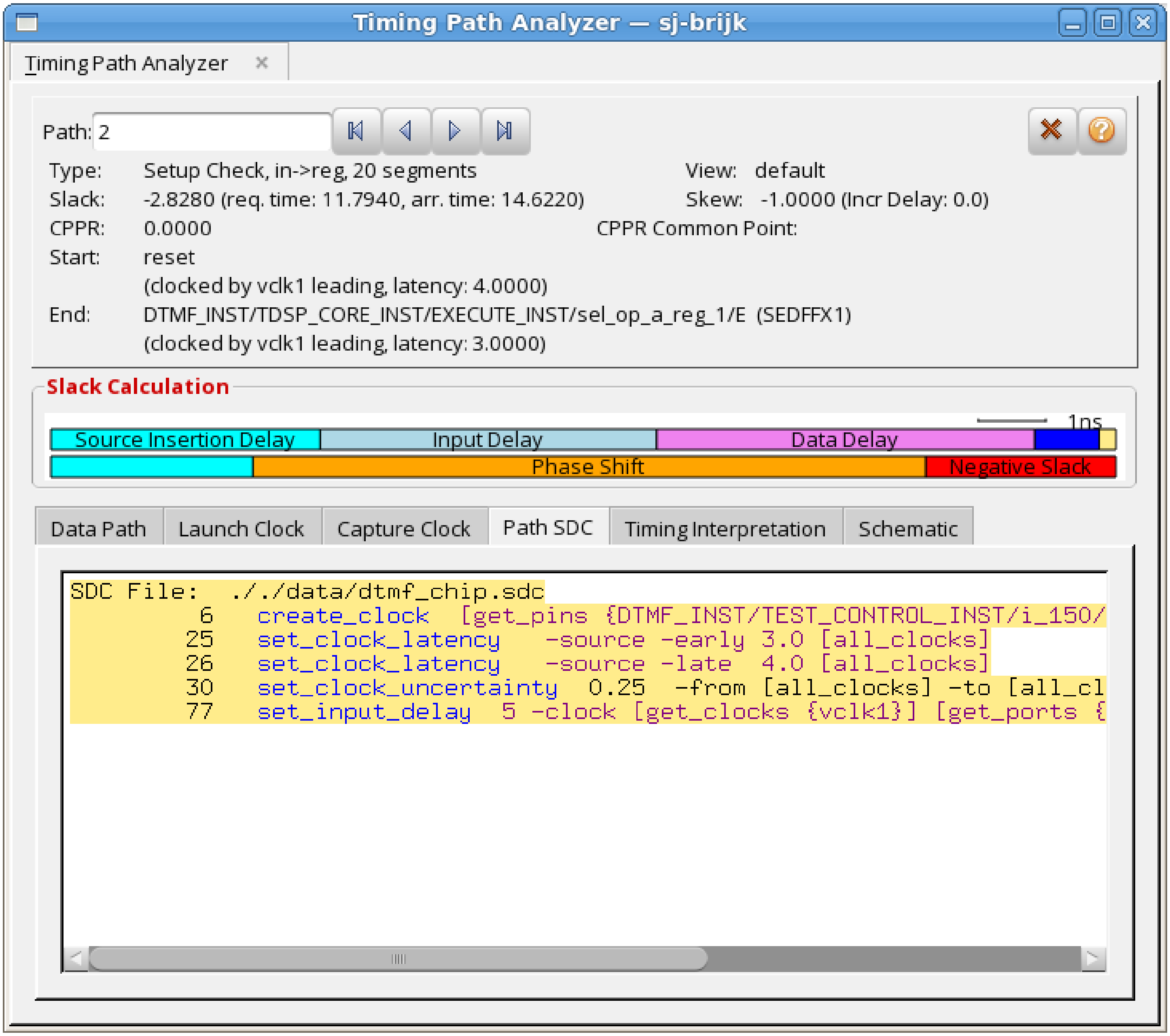
Category: all, Slack Range: all

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
□ 🟨	1	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	2	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	3	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	4	vclk1(leading)->vclk1(l...	11.794	-2.828	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	5	vclk1(leading)->vclk1(l...	11.794	-2.827	reset	DTMF_INST/TDSP_CORE_INS...
□ 🟨	6	vclk1(leading)->vclk1(l...	11.794	-2.827	reset	DTMF_INST/TDSP_CORE_INS...

- Right-click on one of the categories and choose *List Paths*.
- Right-click on one the paths and choose Show Timing Path Analyzer.

The Timing Path Analyzer is displayed.

- Click on the Path SDC tab to display the SDCs:



Path: 2

Type: Setup Check, in->reg, 20 segments
View: default
Slack: -2.8280 (req. time: 11.7940, arr. time: 14.6220)
Skew: -1.0000 (Incr Delay: 0.0)
CPPR: 0.0000
CPPR Common Point:
Start: reset
(clocked by vclk1 leading, latency: 4.0000)
End: DTMF_INST/TDSP_CORE_INST/EXECUTE_INST/sel_op_a_reg_1/E (SEDFFX1)
(clocked by vclk1 leading, latency: 3.0000)

```
SDC File:   ././data/dtmf_chip.sdc
         6   create_clock  [get_pins {DTMF_INST/TEST_CONTROL_INST/i_150/
        25   set_clock_latency   -source -early 3.0 [all_clocks]
        26   set_clock_latency   -source -late  4.0 [all_clocks]
        30   set_clock_uncertainty  0.25  -from [all_clocks] -to [all_cl
        77   set_input_delay  5 -clock [get_clocks {vclk1}] [get_ports {
```

Note that the SDCs relative to mode `mission_140MHz` that produced the path are highlighted.

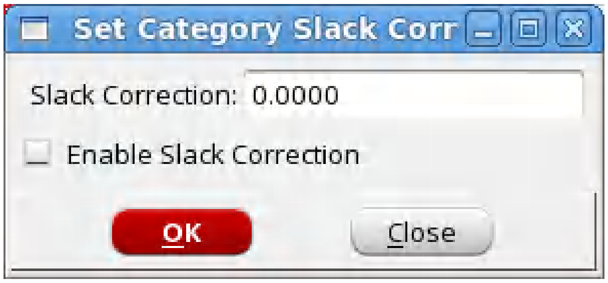
Manual Slack Correction of Categories

Use the Set Category Slack Correction form to specify the estimated slack correction for the selected category of paths. A slack correction that you apply to a category modifies all the paths in that category. If a path belongs to several categories, all the correction from the categories are added. The worst negative slack and total negative slack values of a category can be affected by the correction applied to another category.

Once you enable the slack correction, the histogram is updated to reflect the slack correction. An asterisk (*) is added next to the slack value of paths that belong to this category in the *Path List* field in the *Analysis* tab. Paths are reordered based on new specified slack. This allows you to filter out the paths that can be fixed and work on the remaining paths.

To access the Set Category Slack Correction form complete the following steps:

- Click on the Analysis Tab in the main Innovus window.
- Right click on the category name in the *Path Category* field.
- Choose the *Set Category Slack Correction* option.



Slack Correction: 0.0000

□ Enable Slack Correction

To disable the set slack correction value:

- Right click on the category name in the *Path Category* field.
- Choose the *Deactive Category Slack correction* option.

Editing Table Columns

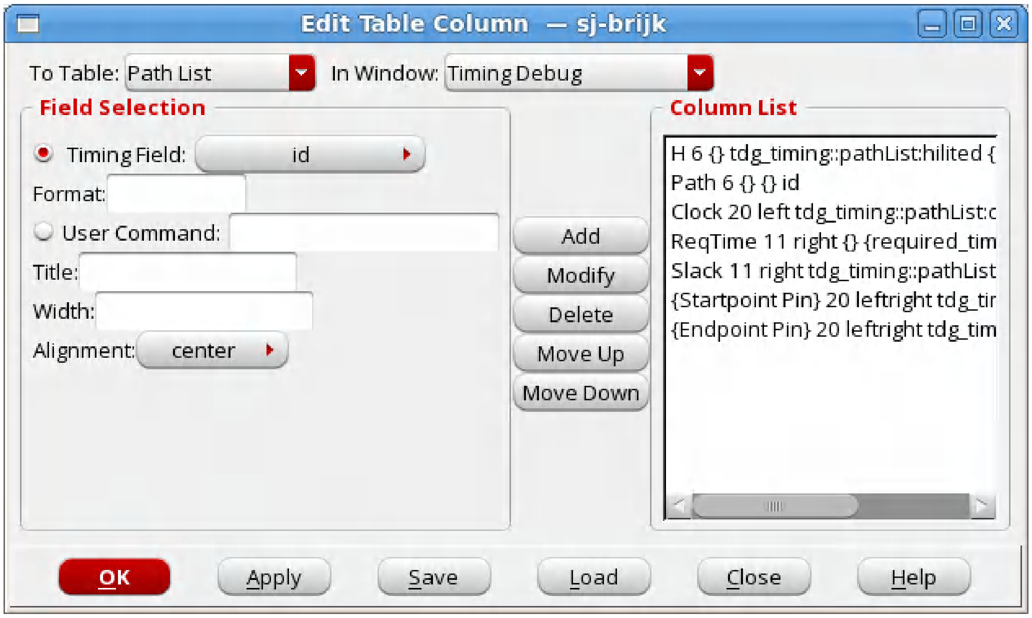
You can customize the dimensions and contents of table columns to suit your needs.

- To begin customizing a table, click on the *Analysis* tab, then right-click on a path.

A drop-down menu is displayed.

- Select *Edit Table Column*.

The Edit Table Column form is displayed.


Edit Table Column — sj-brijk

To Table: Path List
In Window: Timing Debug

Field Selection

● Timing Field: id
Format:
○ User Command:
Title:
Width:
Alignment: center

Column List

```
H 6 {} tdg_timing::pathList:hilited {
Path 6 {} {} id
Clock 20 left tdg_timing::pathList:c
ReqTime 11 right {} {required_tim
Slack 11 right tdg_timing::pathList
{Startpoint Pin} 20 leftright tdg_tir
{Endpoint Pin} 20 leftright tdg_tim
```

- Choose the timing window that contains the table to want to customize.
- Choose the table whose columns you want to customize. The selections change according to the timing window you choose.
- Choose a column item or specify a command.

For commands, specify the procedure you want to use to determine the information you want to include in the column. Source the file containing the procedure before you specify the procedure here.

For example:

```
# Combine fedge (from edge) and tedge (to edge) #information into a single field
proc my_get_edge {id var} {
    upvar #0 $var p
    if {$p(type) == "inst"} {
        return "$p(fedge) -> $p(tedge)"
    } elseif {$p(type) == "port" } {
        return $p(fedge)
    } else {
        return ""
    }
}
```

- Build the column list.
  - *Add* adds a column to the column list.
  - *Modify* let you modify characteristics. Click on a column in the column list. Edit the

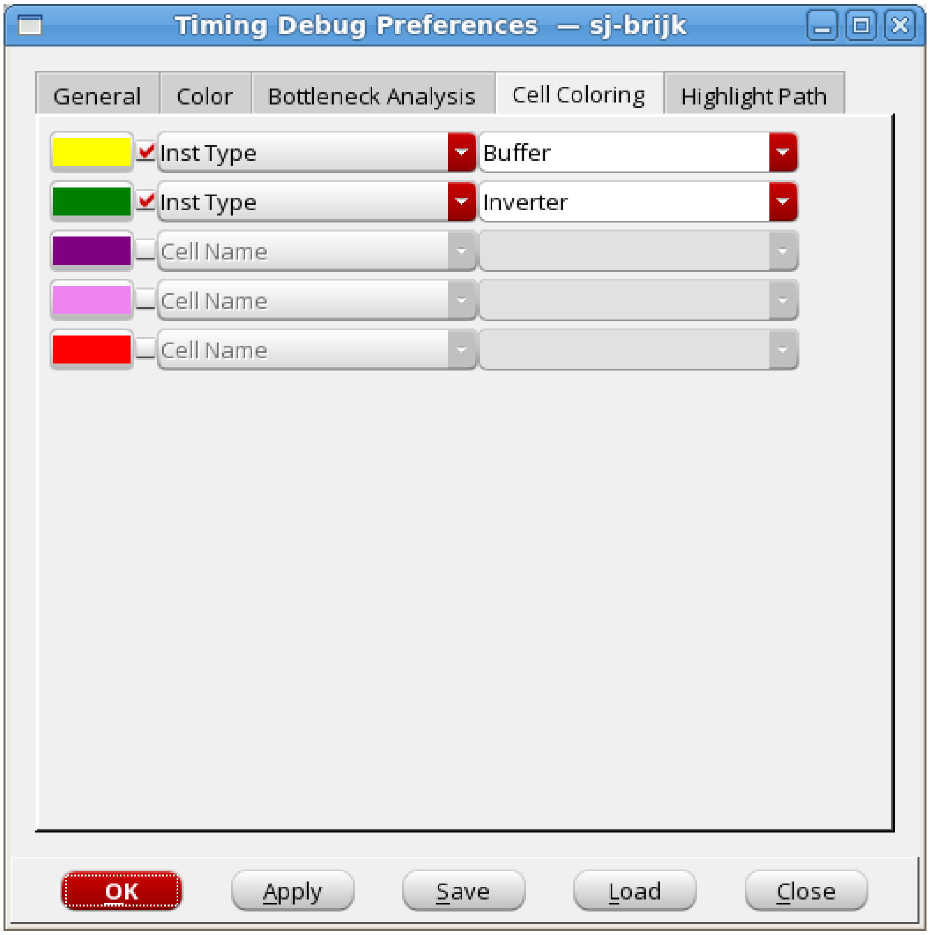
information, then click *Modify*.

- *Delete* removes a column from the column list.
- *Move Up* moves a column up in the column list. This effectively moves a column to the left in the table.
- *Move Down* moves a column down in the column list. This effectively moves a column to the right table.

- (Optional) Click *Load*. The opens the GTD (Global Timing Debug) Preferences form. Specify a file name.

Cell Coloring

Use the *Cell Coloring* page of the Timing Debug Preferences form to choose colors for specific cells in the delay bar.



🟨	☑	Inst Type	Buffer
🟩	☑	Inst Type	Inverter
🟪	□	Cell Name	
🟪	□	Cell Name	
🟥	□	Cell Name	

When you assign colors, this same colors will be restored when you start a new session.

In the *Cell Name Selection Elements* field for each color, you can choose whether you are providing one of the following:

- Cell name
- Instance
- Procedure that you have defined

The procedure is invoked with the full instance name as the argument. You must source the file containing the procedure before you use this feature.

For example:

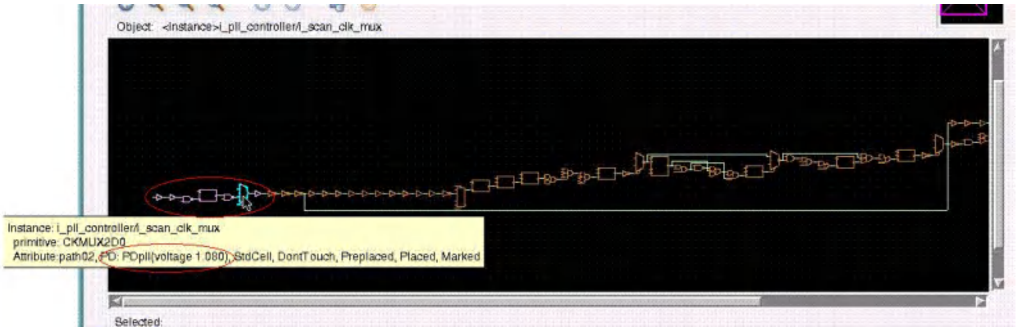
```
# Colors when the instance name contains "core/block1"
proc belongs_to_block1 {inst_name} {
     if [regexp {core/block1} $inst_name] {
        return 1
     } else {
        return 0
     }

}
```

Viewing Schematics

The Critical Path Schematic Viewer displays the gate-level schematic view of the critical path. To display the Schematic Viewer, click on the schematics icon in the *Path List* field of the *Analysis* tab. You can display additional paths in the Schematic Viewer by using the middle mouse button to drag the path from path list to Schematic Viewer.

You can also display the Schematics by selecting the Schematics tab in the *Timing Path Analyzer* form. The form is displayed when you double-click on a critical path in the Path List in the *Analysis* tab.

On displaying the Schematic Viewer, you can see the power instance colored and the power domain information displayed in a popup message box as well as in terminal.



Object: <instance>i_pll_controller/i_scan_clk_mux

Instance: i_pll_controller/i_scan_clk_mux
primitive: CKMUX2D0
Attribute:path02, PD: PDpll(voltage 1.080), StdCell, DontTouch, Preplaced, Placed, Marked

You can perform the following tasks in the Module Schematic Viewer:

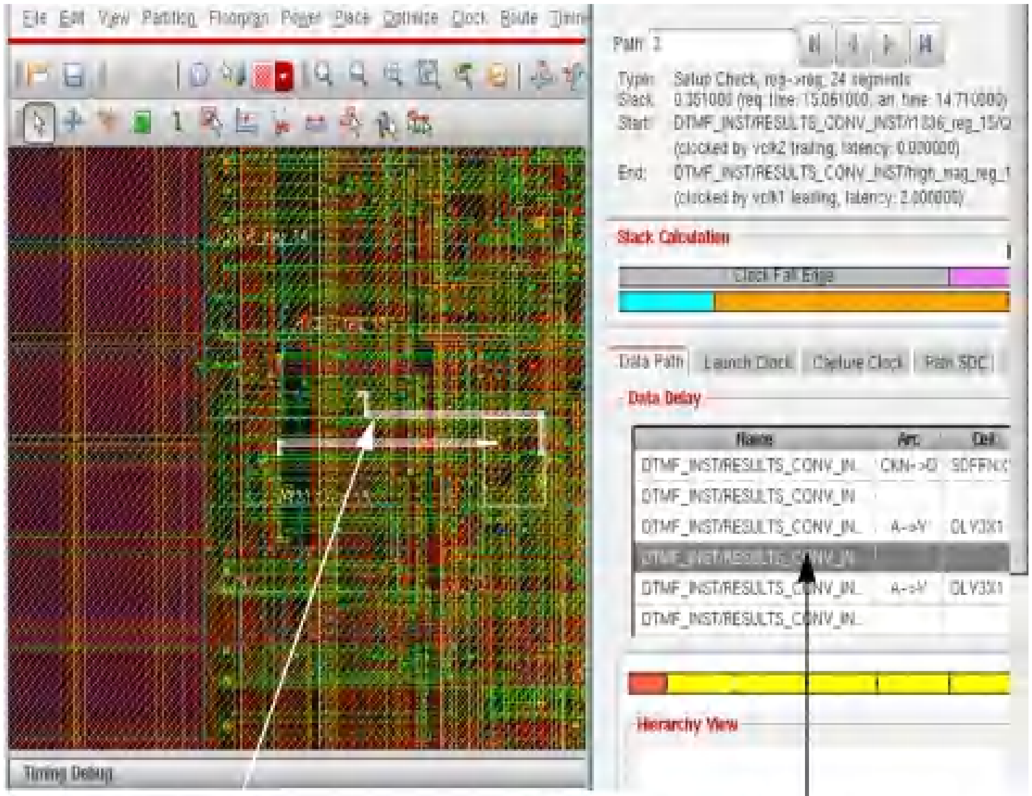
- View the Gate-level design elements.
- Select an element in the schematic.
    - Click on an object in the schematic to select and highlight it. When you move the cursor to an object, the object type and name of the object appear in the information box.
- Scroll over an object to display the object type and name of the object in the *Object* field.
- Cross-probe between the Schematics window and *Path List* field.
    - Select a path and left-click on the Schematics button above the Path List Table. (This is the button at the far-right side, just above the table).
    - To show multiple paths, select another path, and drag and drop it to the Schematics window.
- Use the menu options provided in the Schematic Viewer. To access the menu options, you can either click on the menu bar or right-click on an object in the schematic. You can use the menu options to perform the following tasks:
    - Manipulate schematic views of fan-in and fan-out cones.
    - Trace connectivity between drivers, objects, and loads.
    - Move between different levels of instance views.

Running Timing Debug with Interface Logic Models

You can use the timing debug feature with designs containing Interface Logic Models (ILMs).

- The Timing Path Analyzer - Path SDC form displays ILM SDCs rather than the original SDCs.
- The software highlights the entire ILM module instead of the instances and nets inside the ILM. The instances and the nets inside the ILMs are greyed out in the Timing Path Analyzer -

Path SDC form.



Path 3

Type: Setup Check, reg->reg, 24 segments
Slack: 0.351000 (req time: 15.061000, arr time: 14.710000)
Start: DTMF_INST/RESULTS_CONV_INST/r1036_reg_15/Q
(clocked by vclk2 trailing, latency: 0.000000)
End: DTMF_INST/RESULTS_CONV_INST/high_mag_reg_1
(clocked by vclk1 leading, latency: 2.000000)

Name	Arc	Cell
DTMF_INST/RESULTS_CONV_IN...	CKN->Q	SDFFNX
DTMF_INST/RESULTS_CONV_IN		
DTMF_INST/RESULTS_CONV_IN...	A->Y	DLY3X1
DTMF_INST/RESULTS_CONV_IN		
DTMF_INST/RESULTS_CONV_IN...	A->Y	DLY3X1
DTMF_INST/RESULTS_CONV_IN...		

The entire ILM module is highlighted

Instances and nets inside the ILM module are greyed out in the Timing Path Analyzer – Path SDC form.

Power and Rail Analysis

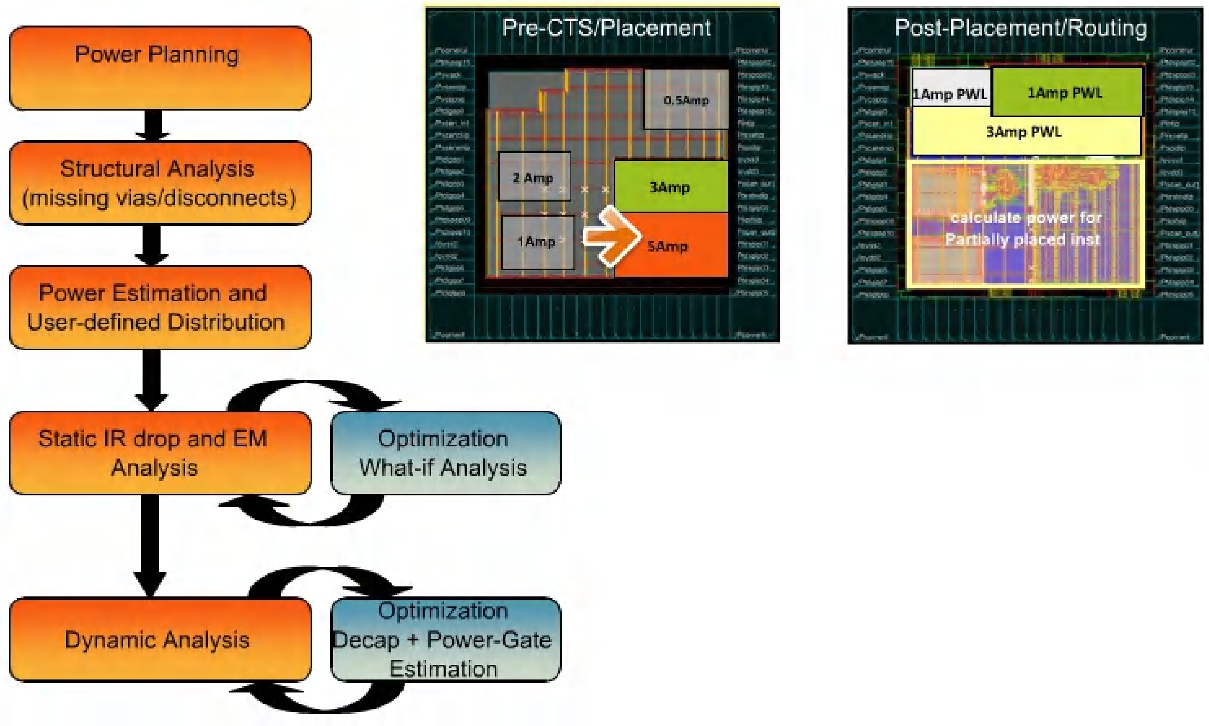
Overview

Voltus Power Integrity Solution sign-off power and rail analysis engines are fully integrated in Innovus Implementation System (Innovus). The TCL and GUI use-models are identical in stand-alone Voltus and its integration in Innovus, allowing a smooth transition from early power planning to sign-off power analysis. The power and rail analysis in Innovus is available under the *Power* menu (Voltus is under *Power & Rail* menu). These menus are arranged differently between the two products (See Innovus and Voltus Menu Differences).
The ERA feature provides rail analysis at the early stage of design with the same use model as Signoff Rail Analysis. Static ERA can also be run with Innovus base license. ERA is not intended for sign-off; if you want to include custom power-grid views or do dynamic analysis, then a Voltus license is required. ERA is intended for early stage analysis and can run on a power-grid floorplan before placement and routing.

Early Rail Analysis

The Early Rail Analysis (ERA) feature inside Innovus works using the `set_rail_analysis_mode` and `analyze_rail` commands. This flow utilizes the same power-grid extraction engine used in Voltus to have seamless transition between early and signoff power-grid analysis. The flow also supports advanced features such as what-if wires, what-if vias, and flexible create current region, along with the ERA flow parameters of the `set_rail_analysis_mode` command. Refer to *Voltus User Guide* for flow details related to the advanced features.

ERA has the ability to analyze power-grid integrity early in the floorplanning stage, after placement, as well as postrouting. It helps fix power-grid problems early in the flow, rather than waiting for when the layout is mostly done and the problems are much more difficult to correct. ERA will take whatever blocks, macros, standard cells, and routing that is available to help improve the accuracy of early rail analysis. The following diagram illustrates the ERA flow:


Power Planning
Structural Analysis (missing vias/disconnects)
Power Estimation and User-defined Distribution
Static IR drop and EM Analysis
Optimization What-if Analysis
Dynamic Analysis
Optimization Decap + Power-Gate Estimation
Pre-CTS/Placement
0.5Amp
2 Amp
3Amp
1Amp
5Amp
Post-Placement/Routing
1Amp PWL
1Amp PWL
3Amp PWL
calculate power for Partially placed inst

The following steps describe the ERA flow:

- **Grid Completion**: The ERA engine checks if the followpin routing has been done or not in the design; if not, it creates virtual followpin automatically and drops required virtual vias. Missing virtual vias between stripes are also dropped by default at this stage.
- **Power Estimation and User-Defined Distribution**: In the ERA static flow, you can specify total power and the ERA power engine will distribute that internally to all placed and unplaced instances in the design. For unplaced instances, current regions are created. You can also selectively assign a specific power value to placed macros, cell or instances using an ASCII file. User-specified power can be enforced for unplaced instances using an explicit current regions file.
- **Static IRdrop and EM Analysis**: Run static IRdrop and electromigration analysis.
- **Dynamic Analysis**: Run dynamic IRdrop analysis. For this, you need to specify explicit dynamic current region file or the dynamic instance current file.

ERA uses Voltus extractor and rail analysis engines. They can be used during power-grid prototyping to analyze power, IRdrop and power-grid integrity. This section includes:

- Early Rail Analysis Key Features
- Setting up and Running Early Rail Analysis

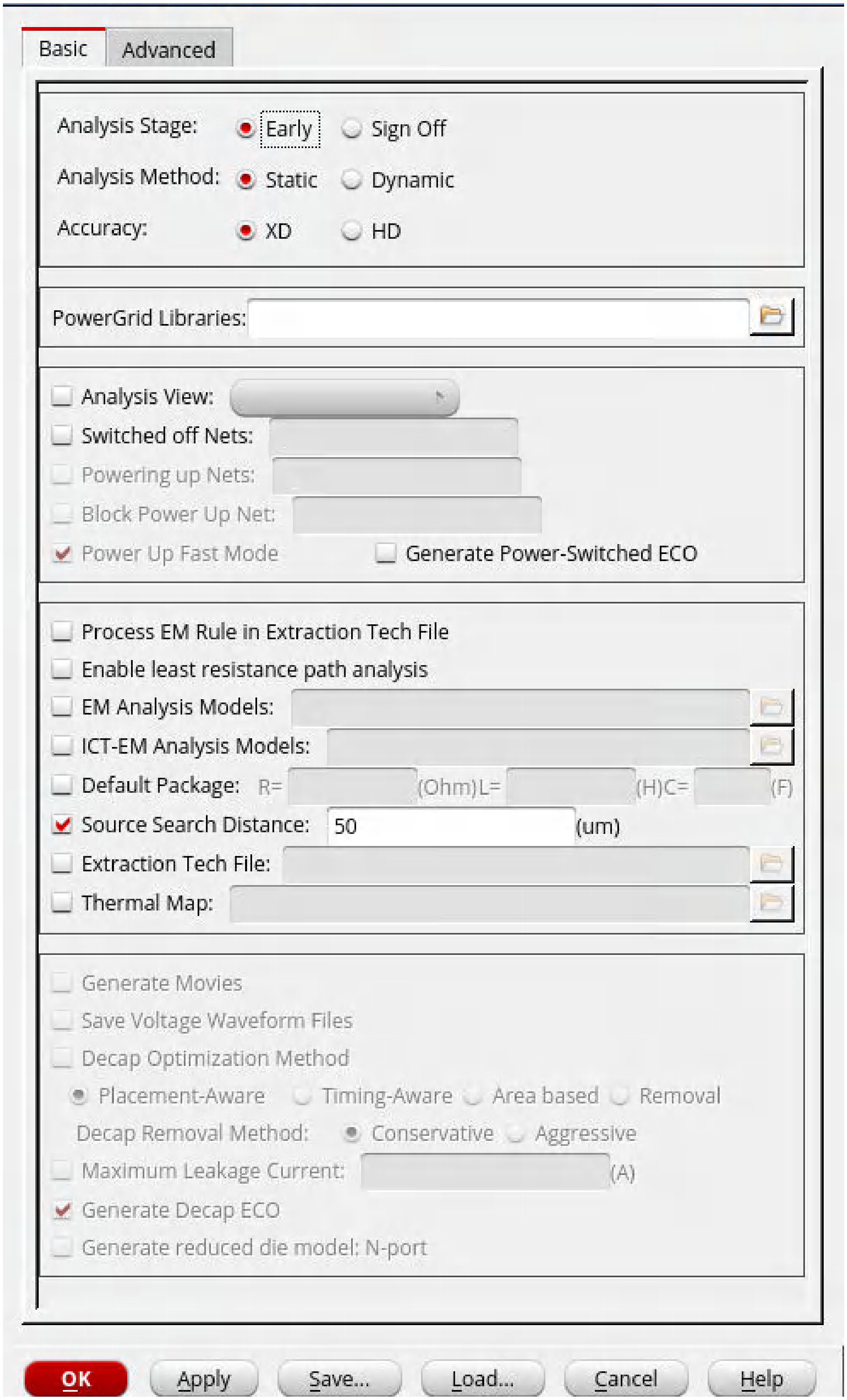
Early Rail Analysis Key Features

- Static and Dynamic Rail analysis during the floorplanning stage using grid based interactive current specification use-model. Static and Dynamic Rail Analysis requires the VTS-L/VTS-XL license. Static Power Analysis can be run using the Innovus license.
- Interactive current specification use-model to enable static and dynamic rail analysis at the floor-planning stage
- Power and rail analysis during the placement stage using ERA driven virtual follow pin routing and virtual via for grid completion.
- Automatic current region generation accounting for unplaced instance in the design, and automatic distribution of power among placed instances in the design.
- Support of user-specified explicit power value at macro, cell and instance level in an ASCII file format, hence, enabling flexible power distribution.
- PGV library is optional. If not provided, it is generated on the fly using the specified technology file.
- Static Power and Rail Analysis without PGV can also be run using the Innovus license.
- Power and rail analysis on a placed and routed database.
- Early power-switch analysis to refine power-switch placement.
- Support for multi-CPU in static/dynamic power and rail analysis, and static and dynamic power and rail analysis. For information on how to set up multi-CPU analysis, see the "Distributed Processing" chapter in *Voltus User Guide*.
- Support for the unplaced flow during static and dynamic analysis.
- What-if shape analysis to guide power-grid optimization.
- Support for native power-up analysis.

Setting up and Running Early Rail Analysis

To setup and run early rail analysis, perform the following steps.

1. Select *Power - Rail Analysis - Setup Rail Analysis Mode* menu. The following form appears:



Analysis Stage: ● Early ○ Sign Off

Analysis Method: ● Static ○ Dynamic

Accuracy: ● XD ○ HD

PowerGrid Libraries:

□ Analysis View:

□ Switched off Nets:

□ Powering up Nets:

□ Block Power Up Net:

☑ Power Up Fast Mode □ Generate Power-Switched ECO

□ Process EM Rule in Extraction Tech File

□ Enable least resistance path analysis

□ EM Analysis Models:

□ ICT-EM Analysis Models:

□ Default Package: R= (Ohm)L= (H)C= (F)

☑ Source Search Distance: 50 (um)

□ Extraction Tech File:

□ Thermal Map:

□ Generate Movies

□ Save Voltage Waveform Files

□ Decap Optimization Method

● Placement-Aware ○ Timing-Aware ○ Area based ○ Removal

Decap Removal Method: ● Conservative ○ Aggressive

□ Maximum Leakage Current: (A)

☑ Generate Decap ECO

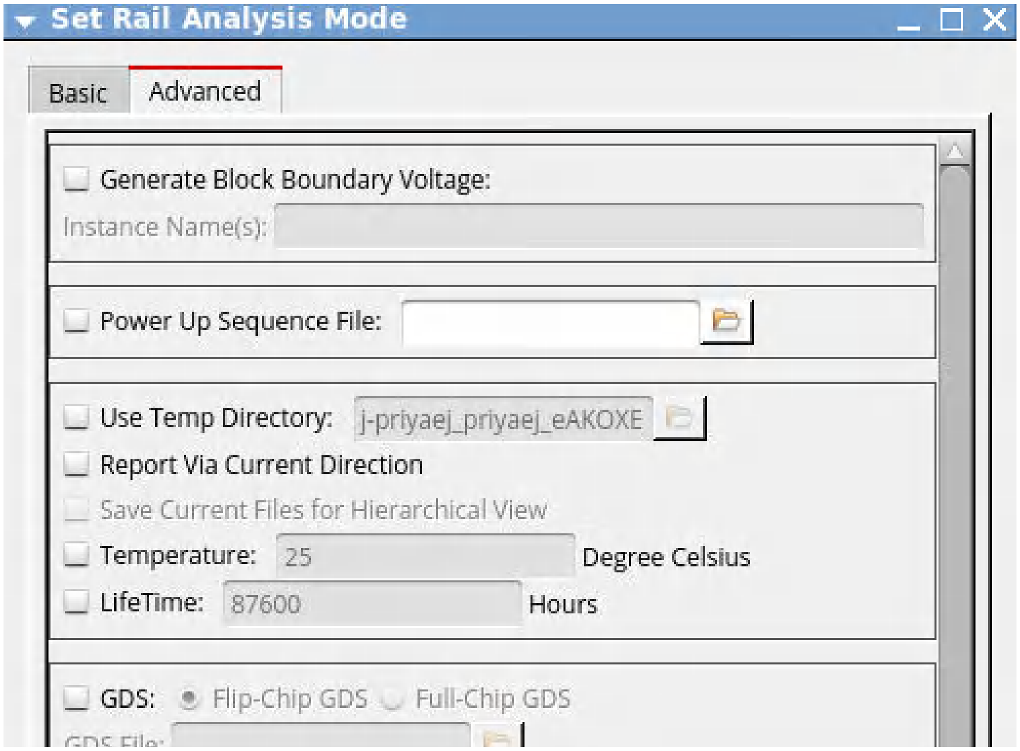
□ Generate reduced die model: N-port

Note: The content of the two tabs of this form change depending on the selection of the analysis method.

2. Set Analysis Stage as *Early .*
3. Set Analysis Method as *Static* or *Dynamic.*
   By default, *XD* is selected as the Accuracy mode. HD is disabled for early rail analysis.
4. Specify *Power-Grid Libraries* or *Extraction Tech File*.
   See the "Power-Grid Library Generation" chapter of the *Voltus User Guide* for details on generating power-grid libraries.
5. If CPF is loaded, select *Analysis View*.
6. If it is a power gated design, optionally specify *Switched off Nets.*
7. For optional Electromigration analysis, specify either *EM Analysis Models* or *Process EM rules in Extraction Tech File*.

   Note: If you do not specify an Electromigration model file, the software will select the Electromigration models from the Quantus technology file. You can use the command `set_rail_analysis_mode -em_models` *`file`* to override these settings.

8. Select the *Advanced* tab of the *Set Rail Analysis Mode* form. The following form appears:



Set Rail Analysis Mode

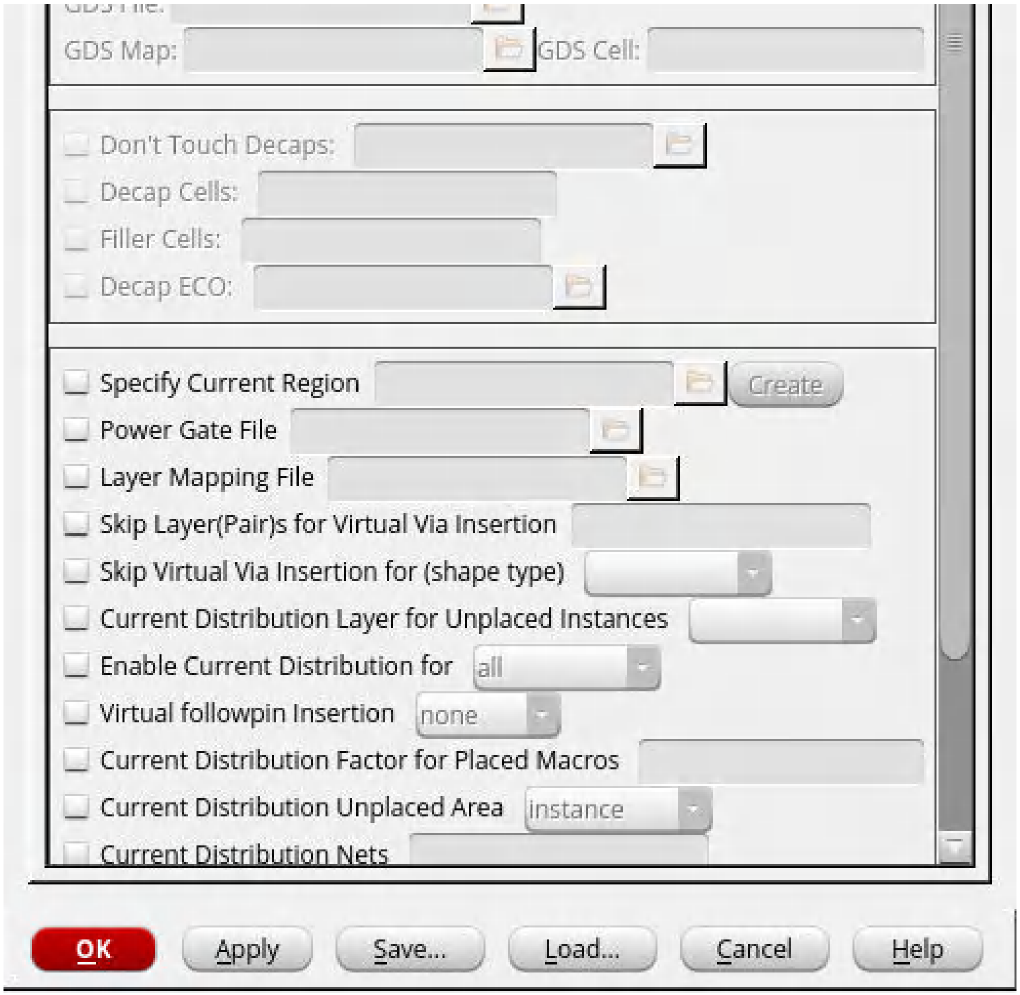
Basic | Advanced

□ Generate Block Boundary Voltage:
Instance Name(s):

□ Power Up Sequence File:

□ Use Temp Directory: j-priyaej_priyaej_eAKOXE
□ Report Via Current Direction
□ Save Current Files for Hierarchical View
□ Temperature: 25 Degree Celsius
□ LifeTime: 87600 Hours

□ GDS: ● Flip-Chip GDS ○ Full-Chip GDS
GDS File:



GDS Map:
GDS Cell:

□ Don't Touch Decaps:
□ Decap Cells:
□ Filler Cells:
□ Decap ECO:

□ Specify Current Region
□ Power Gate File
□ Layer Mapping File
□ Skip Layer(Pair)s for Virtual Via Insertion
□ Skip Virtual Via Insertion for (shape type)
□ Current Distribution Layer for Unplaced Instances
□ Enable Current Distribution for: all
□ Virtual followpin Insertion: none
□ Current Distribution Factor for Placed Macros
□ Current Distribution Unplaced Area: instance
□ Current Distribution Nets

9. You can specify one or more of the following advanced options:

   - *Generate Boundary Voltage File* if using hierarchical view for the block
   - *GDS for Flip-chip RDL* or *Full-chip GDS* if needed.
     Note: ERA would work only when virtual connections between GDS and DEF are not required.
   - *Specify Current Region* to specify a file that includes a list of regions and the amount of current to be distributed within them for the power-grid. When you select this checkbox, the *Create* button gets enabled that lets you create current regions for Static and Dynamic analysis. See "*Creating Regions for Static and Dynamic Analysis*".
   - *Power Gate File* to specify a power-gate file to analyze nets which are power-gated. The power-gate file syntax is as follows:

     ```
     CELL cellname SUPPLY unswitched_net_name SWITCHED switched_net_name
     ```

     ```
     RON r_value IDSAT idsat_value ILEAK ileak_value
     ```

`CELL` *`cellname`* - the name of the cell.

`SUPPY` *`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 milliamps.

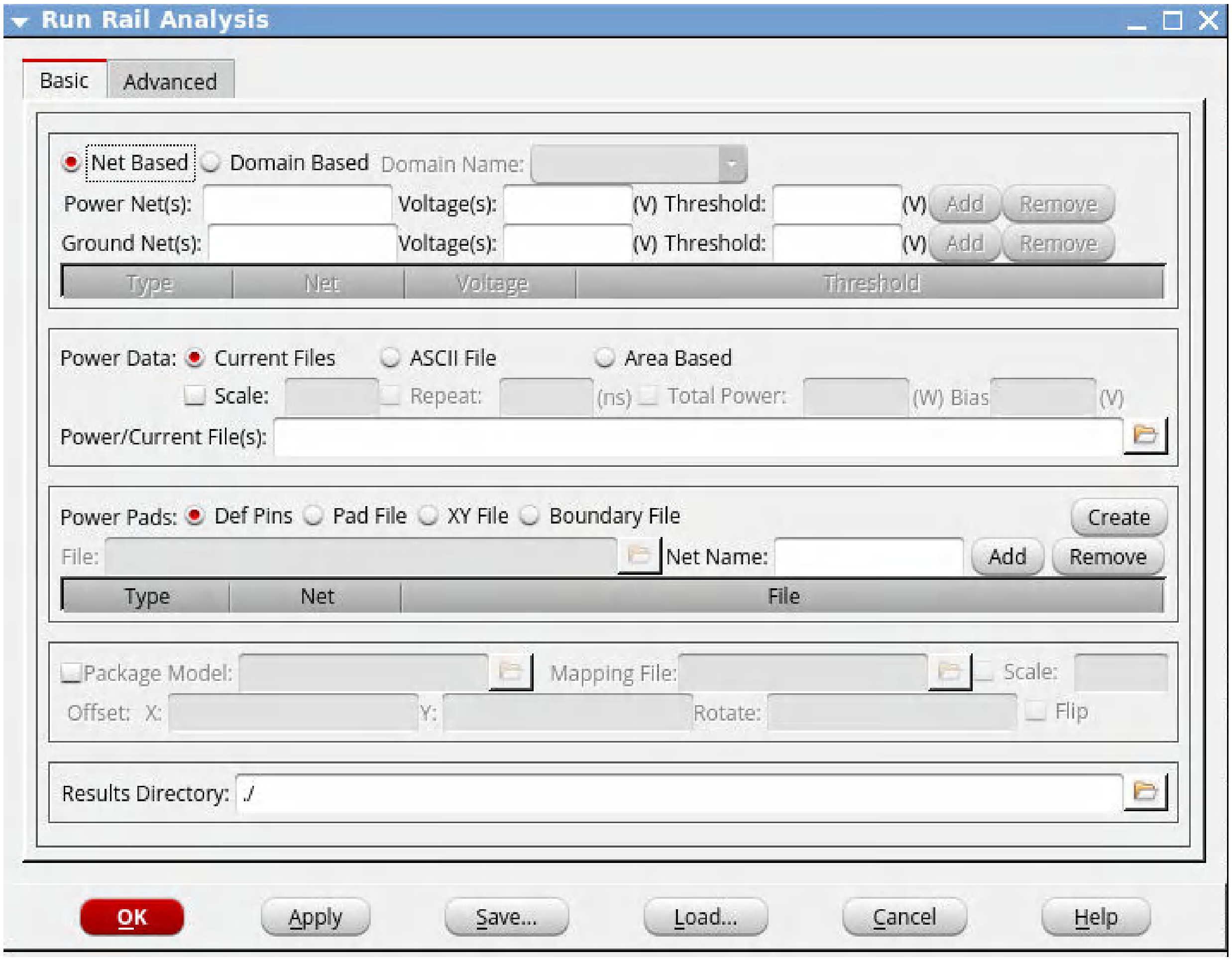
`ILEAK` *`ileak_value`* - the value of the leakage current in milliamps.

If this file is specified, it is expected that the power-switches are fully connected to the appropriate alwaysOn and switched power nets. ERA will extract the power-grid and perform steady state IRdrop analysis. For information about power gate analysis, see "Power Gate (Switch) Analysis" in the *Voltus User Guide*.

- *Layer Mapping File* to specify a layer map file to generate a techonly view. If a layer map file is not provided, it would be automatically inferred by the tool.
- *Skip Layer (Pair)s for Virtual Via Insertion* to skip via insertion between stripes and non-stripes, on the specified LEF layer pairs.
- *Skip Virtual Via Insertion for Shape Type* to skip a given via type. By default, ERA generates all virtual via layer types.
  whatif - vias are virtual vias that have connectivity to user-defined what if shapes.
  def - vias are virtual vias between two metal shapes defined in DEF.
  all - will skip all virtual via generation.
- *Current Distribution Layer for Unplaced Instances* to specify the layer name for distributing unplaced current in the early rail analysis mode.
- *Enable Current Distribution for* to control the behavior of era current distribution. `set_power_data -format area` based power, or `-era_current_region_file` need to be specified for ERA current distribution to work. Placed instanced without uti or ascii based power can also be considered for era current distribution.
  Unplaced: Enable current distribution only for unplaced instances. If `set_power_data -format area` based power is specified, `-era_current_distribution_layer` will be required.
  Placed: Enable current distribution for placed instances without any power specified.
  All: Both unplaced and placed instances without power specified; will have ERA current.
  None: Disable ERA current distribution.

- *Virtual Followpin Insertion* to generate virtual followpins. The `extended` followpins will create followpins that extend from one stripe to another. The `standard` followpins may extend to previous stripe but does not reach the next stripe.
- *Current Distribution Factor for Placed Macros* to control the current distribution factors for the placed instances, hence power allocated for area-based power calculation. For example, if you specify 0.5, the software will assume all placed instances to be 50% of its actual size and distribute current accordingly.
- *Enable Manufacturing Effects* to honor DFM effects.

10. Click *OK* to save and apply the early rail analysis setup information.
11. Select *Power - Rail Analysis - Run Rail Analysis* to run early rail analysis.
    The following form appears:



● Net Based

Power Data: ● Current Files

Power Pads: ● Def Pins

Results Directory: ./

12. Select *Net-Based* or *Domain-Based* Analysis. *Domain Name* is populated automatically if using CPF.

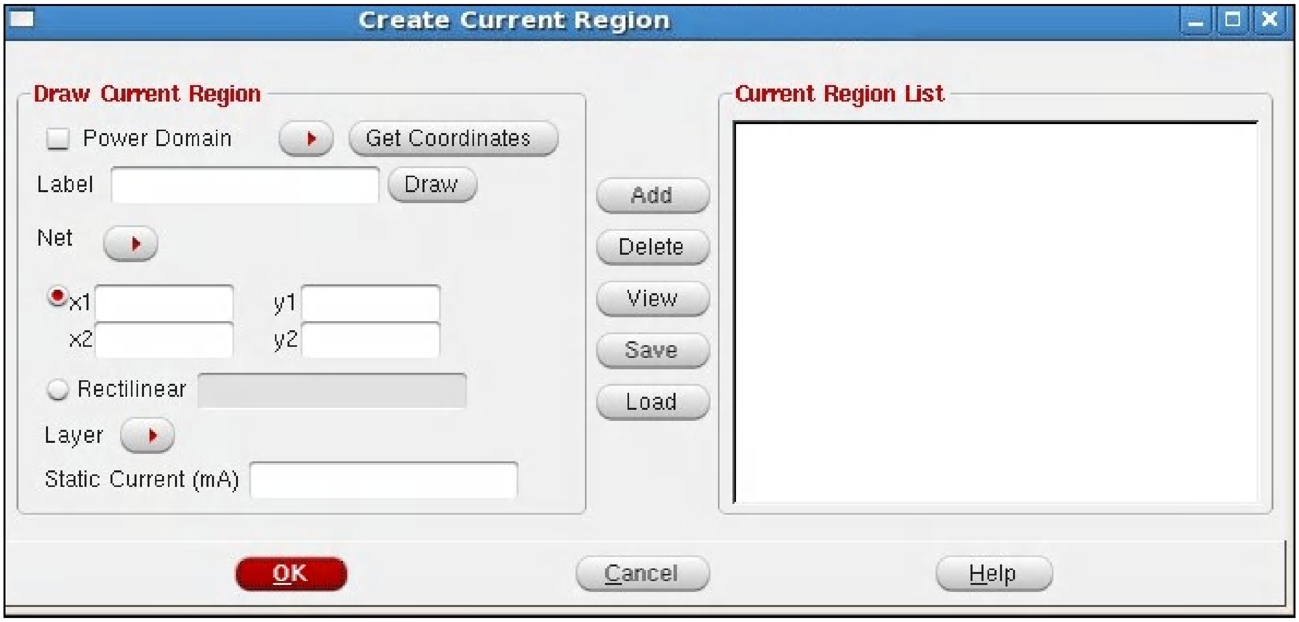
Power and Ground net names are populated automatically if using CPF, otherwise these will need to be entered.

13. Specify power data type and information.
14. Specify power pads. This can be DEF pins, I/O pad cell file, X/Y location file, or if doing a hierarchical analysis, a boundary voltage file.
15. Specify package information if available. This includes a spice subckt and a mapping file to power pads.
16. Specify results directory.
17. Click *OK* to run early rail analysis.
    Upon successful completion of the analysis, the *Power & Rail Results* form appears. In the *Basic* tab, the *State Directory* field is automatically filled with the most recent analysis run. The automatic run naming convention: is `VSS_25C_avg_2` (VSS rail analysis, at 25 degrees Celsius, average or static power, run number 2). Running VSS analysis again will increment 2 to 3.

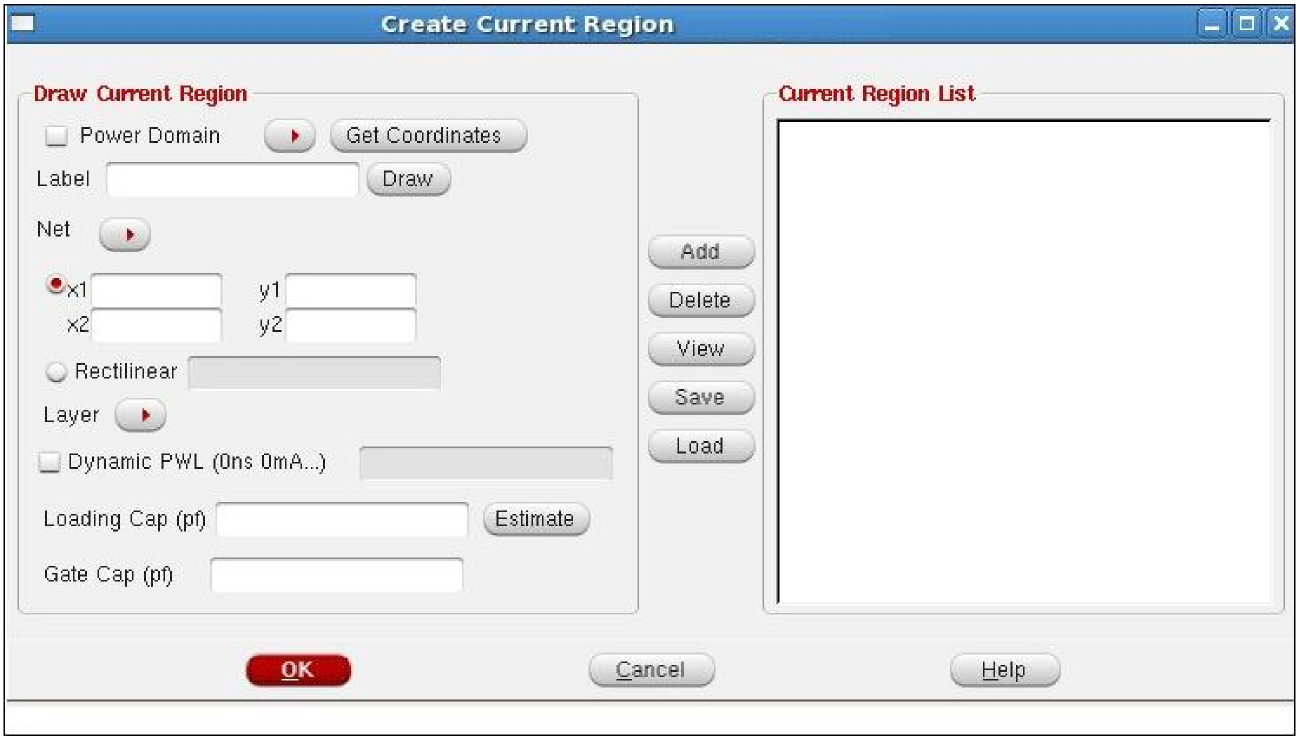
Creating Regions for Static and Dynamic Analysis

You can create regions for Static and Dynamic analysis.

- When you set Analysis Method as *Static*, the following form appears:



- When you set Analysis Method as *Dynamic*, the following form appears:



You can do the following:

1. *Draw Current Regions* to create a *Current Region List*.
   Use this when you have an area that has not been placed, but you would like to have its power consumption influence the overall grid. You can specify the coordinates of the region,

the layer, and the current.

Power Domain lets you specify a power domain based current region. You can use the *Power Domain* field to select a power domain name and click *Get Coordinates* to automatically get the coordinates of the power domain. When the power domain is selected, the label name of the current region will be the power domain name and the boundary box coordinates will be the power domain boundary, and you cannot modify these fields.

Label specifies a name for the region. If not specified, Voltus will provide a name (region1, region2...). The ***Draw*** button lets you draw a window where you want the current to be applied. If you click *Draw*, and then select a box in the main window, the coordinate of this box will be automatically populated in x1 y1 x2 y2. Use the left mouse button to draw the box.
x1, *y1*, *x2*, and *y2* specifies a rectangular region that the current will be distributed within.

Rectilinear specifies the rectilinear current region that the current will be distributed within. You can specify a rectilinear box to add a current region. The rectilinear box enables you to specify multiple x,y points to add current regions in the areas that are not rectangular in shape. To draw the rectilinear box, use the left mouse button and select multiple points. Use the `Esc' key to the last point of the rectilinear box to finish and capture the box co-ordinates.
Note: In the static mode, ERA splits the rectilinear region into several rectangular regions and distributes current to the rectangular regions based on the area.
Current in rectangular region = Area of the rectangle / Area of the rectilinear

Layer specifies the metal layer that the current sink will be placed on.

Static Current specifies the current to be attached in the window.

For dynamic current regions, the PWL waveform is specified in time (ns) and current (mA) pairs. In addition to the dynamic current, you can specify loading capacitance and cell intrinsic capacitance which impacts dynamic IRdrop. If this information is not available for the region, you can click the ***Estimate*** button to populate these values automatically. These capacitance values are derived by calculating loading capacitance of the design using wire-load models and using percentage ratio of loading capacitance to estimate cell intrinsic capacitance in the region.
Note: The effect of loading capacitance depends upon on-resistance through which it is connected to the global power-grid. Generally, this on-resistance value

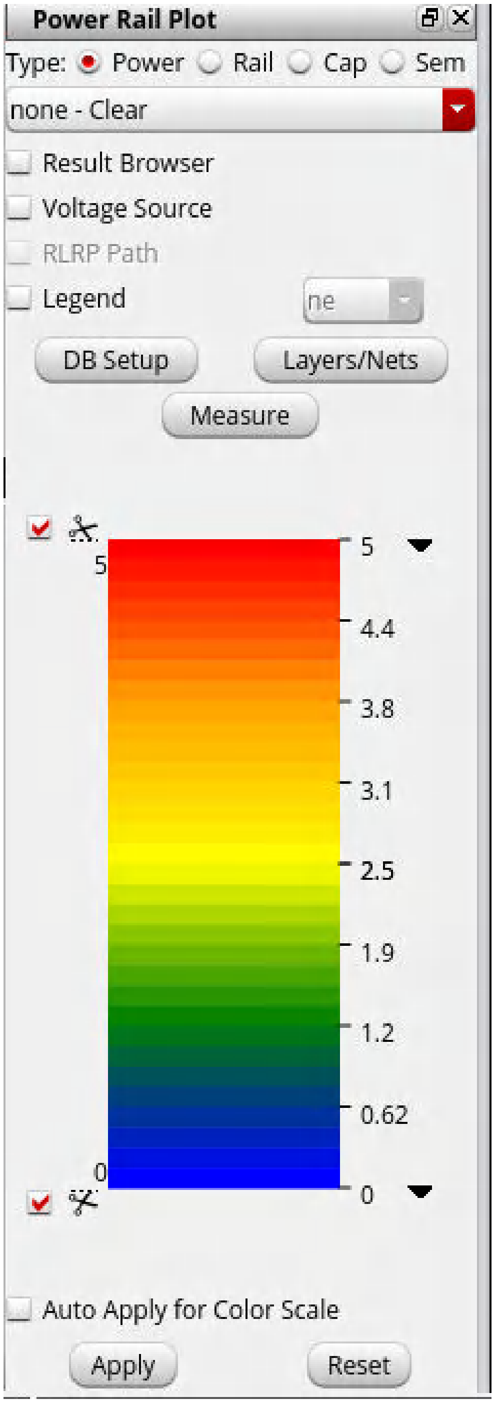
is high and limits the effectiveness of loading capacitance. Therefore, specification of loading capacitance is optional and when specified, you must also specify the on-resistance value.

2. Click the *Add* button to add the region to the *Current Region List* section. 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.
   You can create multiple current regions and add it to the *Current Region List* section.
3. Click *Save* to save the current regions to a file (For example, `vss.curRegion`).
4. Click *OK* to close the *Create Current Region* window.

Viewing Early Rail Analysis Results

Selecting *Power - Report - Power & Rail Result* menu item will bring up the following form:

Power & Rail Results



Power Rail Plot

Type: ● Power ○ Rail ○ Cap ○ Sem

none - Clear

□ Result Browser

□ Voltage Source

□ RLRP Path

□ Legend ne

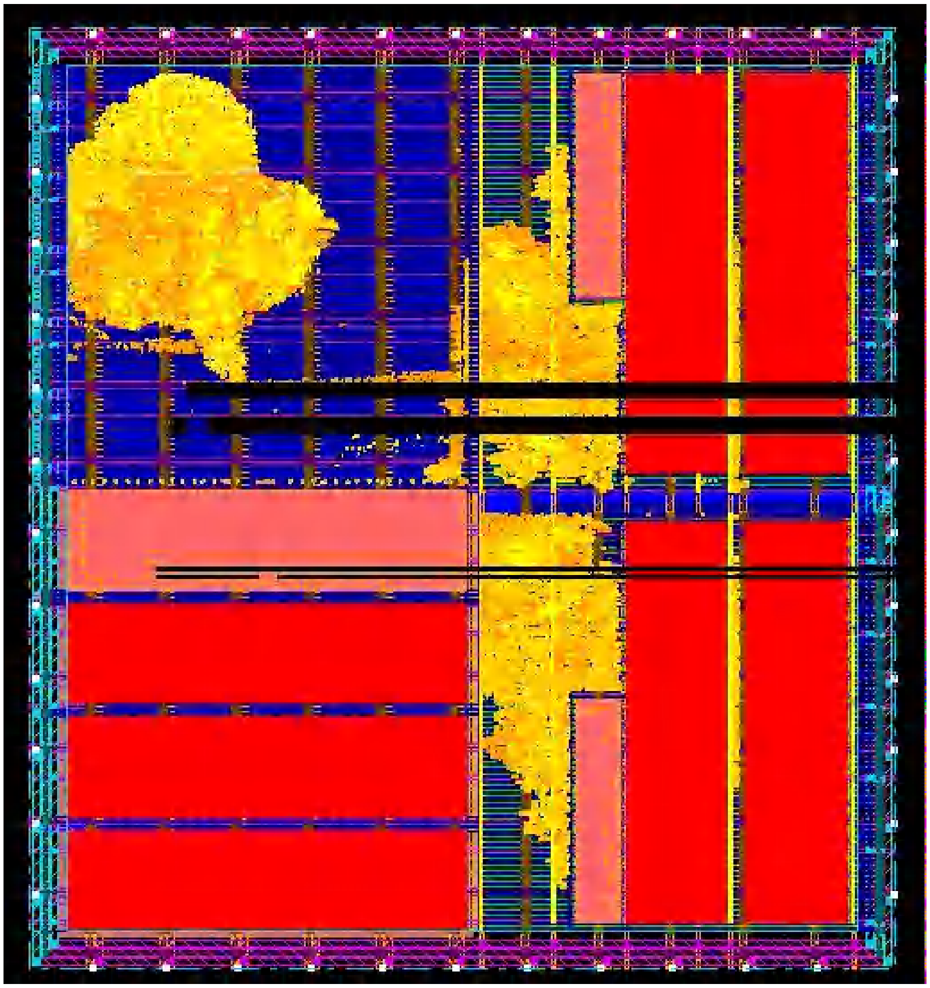
□ Auto Apply for Color Scale

You can use this form to browse to other analysis runs and load them as well to view early analysis results and compare runs. You can specify the type of plot (Rail Analysis, Power Analysis, or Capacitance) and then select the specific plot type. An instance power (ip), load capacitance (load), and irDrop (ir) plot are shown in Instance Power Plot , Load Capacitance Plot, and irDrop Plot, respectively.

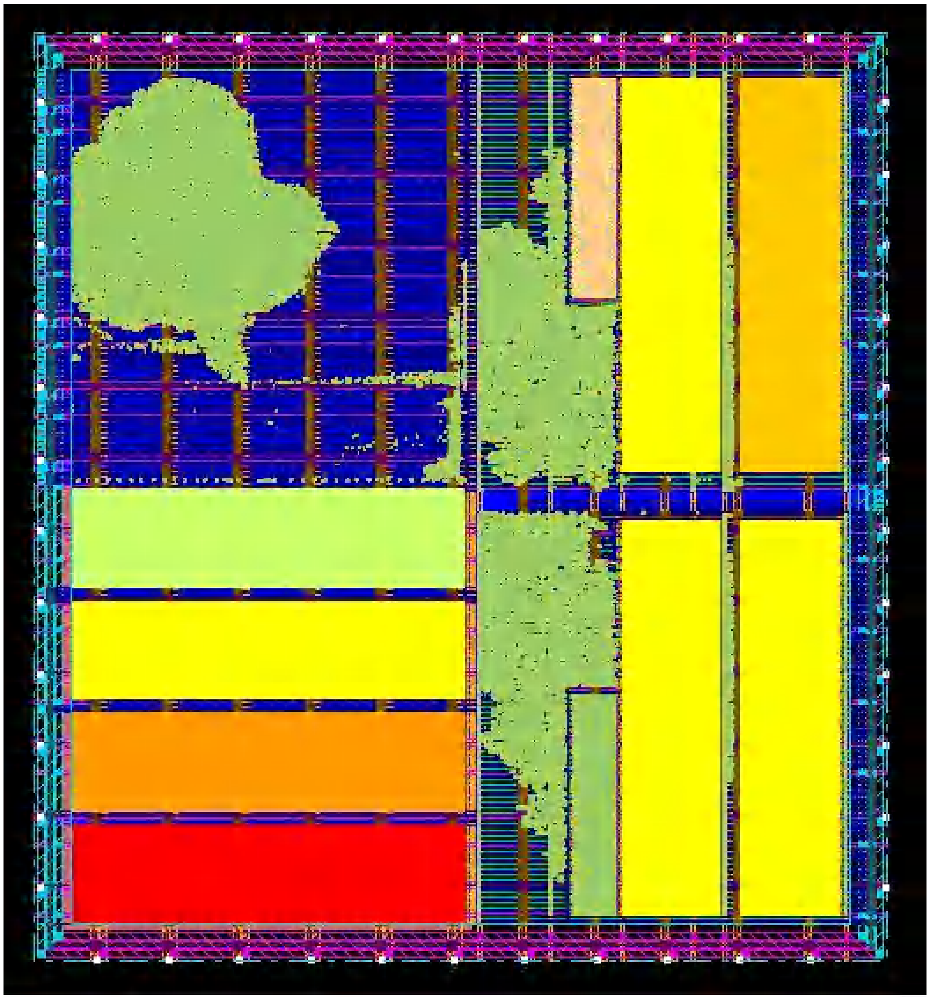
For Early Rail Analysis, the viewing of Power & Rail Results is the same as that used for Sign-off

Analysis. For additional information on viewing the plots, see "Static Power Analysis Plots" and "Static Rail Analysis plotting steps" in the *Voltus User Guide*.

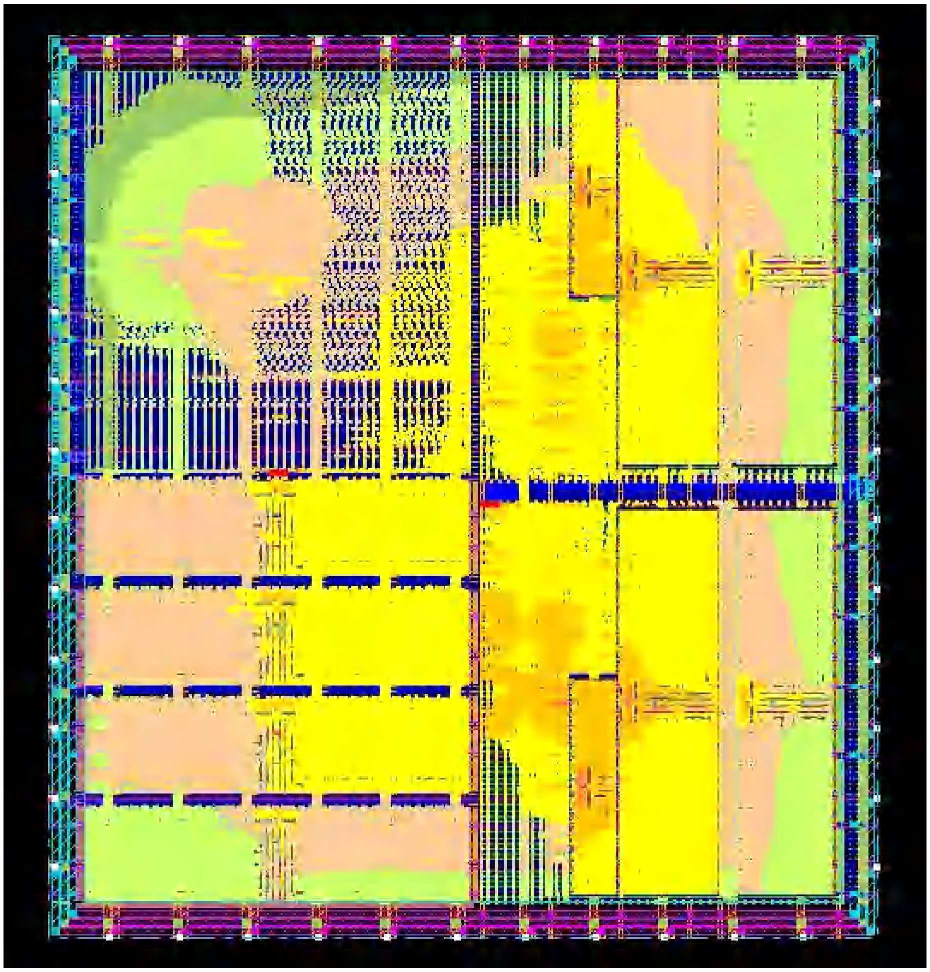
Instance Power Plot



Load Capacitance Plot



irDrop Plot



Signoff-Rail Analysis

For details on running Signoff Power and Rail Analysis within Innovus, see *Voltus User Guide* chapters 5-12.

TCL Command

An example TCL command for Floorplan stage design grid analysis with current regions is as follows:

```
read_lib -lef  design/full.lef
read_verilog design/test.v.gz
set_top_module test
```

```
read_def design/full.def
set_rail_analysis_mode -method era_static -accuracy xd  -extraction_tech_file
design/tech.tch -era_current_region_file design/current_region
set_pg_nets -net VDD -voltage 1.1 -threshold 1.067
set_power_pads -net VDD -format xy -file pads/vdd.pp
analyze_rail -type net -results_directory early_vdd VDD
```

An example TCL command for Power Gate Design with area based power distribution is as follows:

```
read_design -physical_data design.dat CHIP
set_pg_nets -net VDD -voltage 1.1 -threshold 0.9
set_power_pads -net VDD -format xy -file design/vdd.pad
set_power_data -bias_voltage 1.2 -power 1.2 -format area
set_rail_analysis_mode -method era_static -accuracy xd -extraction_tech_file
design/tech.tech -era_power_gate_file design/power_gate_file
analyze_rail -type net VDD
```

An example TCL command for MSMV dynamic analysis is as follows:

```
read_verilog design/test.v.gz
set_top_module test
read_def design/full.def
set_pg_nets -net VSS -voltage 0 -threshold 0.18
set_pg_nets -net VDDm -voltage 0.84 -threshold 0.756
set_pg_nets -net VDD -voltage 0.84 -threshold 0.756
set_power_pads -net VDD -format xy -file design/vdd.pp
set_power_pads -net VDDm -format xy -file design/vddm.pp
set_power_pads -net VSS -format xy -file design/vss.pp
set_power_data -format current { instance_current_files/dynamic_VSS.ptiavg
instance_current_files/dynamic_VDD.ptiavg instance_current_files/dynamic_VDDm.ptiavg
instance_current_files/dynamic_VDDlu.ptiavg
instance_current_files/dynamic_VDDau.ptiavg}
set_rail_analysis_mode -method era_dynamic -accuracy xd -power_grid_library {
stdcells_accurate/accurate_stdcells.cl  lpcells_accurate/accurate_stdcells.cl
memories_accurate/MEM.cl  } -off_rails VDDau
set_rail_analysis_domain -name PD -pwrnets {VDD VDDm} -gndnets VSS
analyze_rail -type domain PD
```

Innovus and Voltus Menu Differences

Form	Innovus Menu	Voltus Menu
Set Power Analysis Mode	*Power - Power Analysis*	*Power & Rail*
Run Power Analysis	*Power - Power Analysis*	*Power & Rail*
Set PG Library Mode	*Power - Rail Analysis*	*Power & Rail*
Generate PG Library	*Power - Rail Analysis*	*Power & Rail*
Setup Rail Analysis Mode	*Power - Rail Analysis*	*Power & Rail*
Analyze ESD	*Power - Rail Analysis*	*Power & Rail*
Optimize ESD	*Power - Rail Analysis*	*Power & Rail*
Set Power Network Optimization Mode	*Power - Rail Analysis*	*Power & Rail*
Run Rail Analysis	*Power - Rail Analysis*	*Power & Rail*
Run Resistance Analysis	*Power - Rail Analysis*	*Power & Rail*
PowerGrid Library Report	*Power - Report*	*Power & Rail - Textual Reports*
Power Report	*Power - Report*	*Power & Rail - Textual Report*
Power Histograms	*Power - Report*	*Power & Rail Analysis - Histograms*
Power & Rail Results	*Power - Report*	*Power & Rail*
Dynamic Movies	*Power - Report*	*Power & Rail - Dynamic Results*

Dynamic Waveforms	*Power - Report*	*Power & Rail - Dynamic Results*

Power Analysis and Reports

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.

   $P = 0.5^*C_LV^2\ F^*A$

   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:

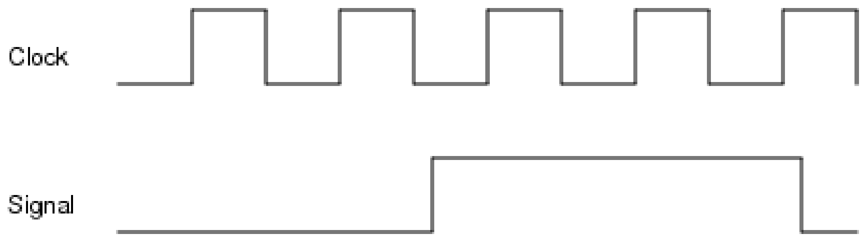
   - Pin Power
   - Arc Power
     Internal power is calculated by using the internal power tables provided in the .lib, which capture the characterized internal power over a range of input slew rates and external loading. The tables reflect the combination of both the internal switching and internal feedthough 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,
Activity = (Number of (0->1 or 1->0) transitions / 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):

$$y = f(x_0, x_1, \ldots, x_n)$$
$$P(y) = P(x_i) * P(y|_{x_i=1}) + (1 - P(x_i)) * P(y|_{x_i=0})$$

The procedure for the weight calculation is similar. However one complication of the energy weighting is that the coverage of the `when' clauses might not be complete. For example, one set of state dependent tables includes only two `when' clauses:

```
when : "A & !B";
```

```
when : "!A & B";
```

This set of clauses does not account for cases where A and B are either both high or both low. In normal operation, neither of these conditions may appear, so the incomplete coverage may not matter. However, the transition density data is static and lacks signal correlation; the conditions not included in the `when' clauses should be accounted for, or else the internal power will be underestimated. Scaling can resolve this. But it turns out that the most common situation for incomplete clauses is in memories, where assuming the energy to be 0 is the more correct thing to do. As a result, the power engine assumes energy of 0 for missing clauses.

Implementation for state-based internal power on the inputs is straightforward. The implementation for the internal power contribution from a single input port is:

$$inputEnergy = \frac{1}{2}(port \rightarrow riseEnergy() + port \rightarrow fallEnergy())$$

For multiple input ports, each port has a set of energy tables, instead of just one pair, and each pair includes a `when' clause, from which a probability can be calculated:

$$inputEnergy = \frac{\sum_i prob(port \rightarrow when(i)) * (port \rightarrow when(i) \rightarrow riseEnergy() + port \rightarrow when(i) \rightarrow fallEnergy())}{2\sum_i prob(port \rightarrow when(i))}$$

As an example, consider a port with the following data, and with P(A) = 0.25 and P(B) = 0.50:

Index	When	Rise Energy	Fall Energy
0	!A&B	3.0nJ	4.1nJ
1	A&!B	3.2nJ	5.0nJ
2	!A&!B	7.0nJ	9.0nJ

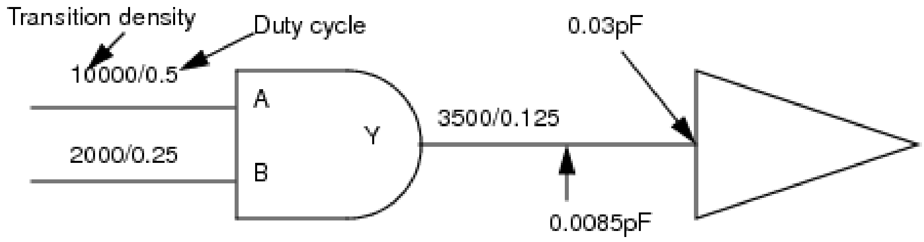
The calculation of energy for a single transition on this input would be:

$$\begin{aligned} inputEnergy &= \frac{prob(!A \& B) * (3.0nJ + 4.1nJ) + prob(A\&!B) * (3.2nJ + 5.0nJ) + prob(!A\&!B) * (7.0nJ + 9.0nJ)}{2(prob(!A \& B) + prob(A\&!B) + prob(!A\&!B)} \\ &= \frac{0.375(7.1nJ) + 0.125 * (8.2nJ) + 0.375(16nJ)}{2(0.375 + 0.125 + 0.375)} = \frac{9.6875}{1.75} = 5.5357nJ \end{aligned}$$

State Dependent Arc-based Internal Power (output pins)

Output internal energy is a weighted sum of values extracted from internal power tables that are associated with timing arcs. These tables are indexed by input transition time and output load capacitance. The values for each arc are weighted by the transition density of the corresponding inputs. The resulting energy value is multiplied by the transition density of the output pin to calculate the power.

When state dependent arc-based (output) internal power tables are present, if two or more tables apply to the same arc, they are weighted by the `when' clauses in the same manner as described in the previous section for "state dependent internal power (input pins)". As an example, we will calculate the output internal power for an AND gate with inputs A and B and output Y. The goal is to calculate the internal power contributed by the output Y. This example does not include any state-dependency.



The first step is to determine the output load capacitance on Y. The output load is the sum of the parasitic net capacitance on the net connected to the Y pin of the instance, plus the pin capacitances of all of the input pins that the net drives. In the example shown, this value is 0.0085pF + 0.03pF = 0.0385pF. 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:

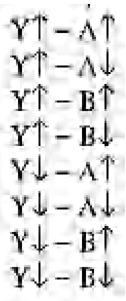
$$tt' = \frac{rise - fall}{100} * tt = \frac{90-10}{100} * 0.5ns = 0.4ns$$

If there are no slew threshold options set in the `.lib` file, or they are commented out, the default is 20-80.

The resulting time should be converted, if necessary, to the time units provided in the `.lib`:

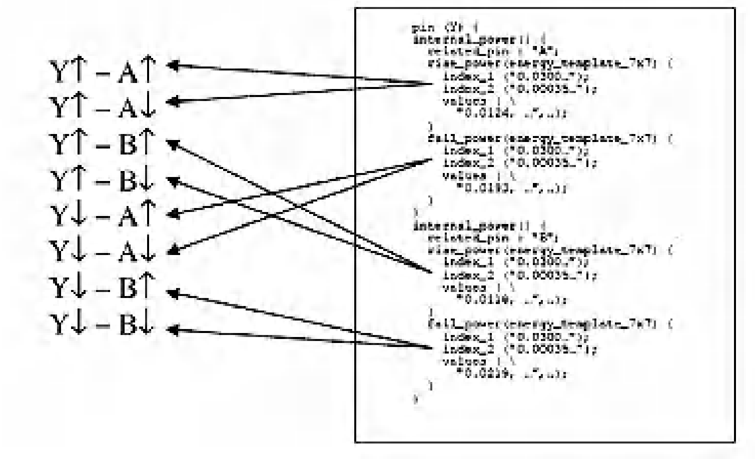
```
time_unit : "1ns";
```

The next step is to associate the timing arcs with the tables in the `.lib` file. An AND gate has eight arcs, each corresponding to a change in output logic level in response to a change in an input logic level:



Y↑ – A↑
Y↑ – A↓
Y↑ – B↑
Y↑ – B↓
Y↓ – A↑
Y↓ – A↓
Y↓ – B↑
Y↓ – B↓

Normally, these eight arcs are represented by four tables in the .lib file. This is the correspondence.



```
pin (Y) {
internal_power() {
  related_pin : "A";
  rise_power(energy_template_7x7) {
    index_1 ("0.0300…");
    index_2 ("0.00035…");
    values ( \
      "0.0124, …",…);
  }
  fall_power(energy_template_7x7) {
    index_1 ("0.0300…");
    index_2 ("0.00035…");
    values ( \
      "0.0183, …",…);
  }
}
internal_power() {
  related_pin : "B";
  rise_power(energy_template_7x7) {
    index_1 ("0.0300…");
    index_2 ("0.00035…");
    values ( \
      "0.0130, …",…);
  }
  fall_power(energy_template_7x7) {
    index_1 ("0.0300…");
    index_2 ("0.00035…");
    values ( \
      "0.0219, …",…);
  }
}
```

The energy for a transition has two components, rise (EY,rise) and fall (EY,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:

$$\begin{aligned}E_{Y,rise} &= \frac{10000s^{-1} * {}^{1}\!/_{2}(0.0134pJ + 0.0134pJ) + 2000s^{-1} * {}^{1}\!/_{2}(0.0135pJ + 0.0135pJ)}{10000s^{-1} + 2000s^{-1}} \\ &= \frac{161pJs^{-1}}{12000s^{-1}} = 0.0134167pJ \\ E_{Y,fall} &= \frac{10000s^{-1} * {}^{1}\!/_{2}(0.0244pJ + 0.0244pJ) + 2000s^{-1} * {}^{1}\!/_{2}(0.0267pJ + 0.0267pJ)}{10000s^{-1} + 2000s^{-1}} \\ &= \frac{297.4pJs^{-1}}{12000s^{-1}} = 0.0247833pJ\end{aligned}$$

The power is:

$$\begin{aligned}P &= {}^{1}\!/_{2}(E_{Y,rise} + E_{Y,fall})D(Y) \\ &= 0.5(0.0134167pJ + 0.0247833pJ)3500s^{-1} \\ &= 66.85pW\end{aligned}$$

State Dependent Leakage Power

As the leakage component of power dissipation increases, accurate estimation of it becomes more and more critical. Originally, the leakage component of a cell's power dissipation was modeled in Liberty libraries as a single number. However, nowadays it is more common to associate different leakage power values with different input combination ("state-dependence").

The method to compute leakage power is as follows. Extract the state-dependent leakage data from a cell's Liberty `.lib` description and compute a weighted sum of the leakage values based on the instance's input probabilities.

The state-dependence is expressed as a set of logical functions describing various input conditions. This set of functions may or may not be complete (e.g. covering all possible input combinations). In addition, a generic leakage power value may also be provided. The power engine covers all of the possible combinations of available data.

State-dependent leakage data appears in a Liberty library in the following form:

```
cell_leakage_power : 14.335 ;

leakage_power() {

when : "!A1 !A2" ;

value : 9.120 ;

}

leakage_power() {
```

```
when : "!A1 A2" ;

value : 16.467 ;

}

leakage_power() {

when : "A1 !A2" ;

value : 12.364 ;

}

leakage_power() {

when : "A1 A2" ;

value : 19.390 ;

}
```

Note that the set of `when' clauses are complete; all possible combinations of the inputs A1 and A2 are accounted for. Also note that there is a generic leakage power value that is not associated with any condition.

We assume the following combinations of input data for our approach:

1. Complete clause set with or without generic leakage value
2. Incomplete clause set with generic leakage value
3. Incomplete clause set without generic leakage value
4. Over-complete clause set with generic leakage value (error condition)
5. Over-complete clause set without generic leakage value (error condition)

If the clause set is complete, we expect the sum of the probabilities of all of the clauses to add up to 1.0. Verification of a complete clause set requires logical analysis of the statements. As extensive library verification is not within the functional requirements of the power engine, we base our assessment of the clause set's completeness by the sum of the probabilities. If this sum is equal to 1.0, the set is complete. If the sum is less than 1.0, we will assume the set is incomplete. If the sum is greater than 1.0 (which would indicate over-coverage), we assume that there is an error in the library data. In order to decide which of the above five conditions we have, we need two pieces of information: whether or not we have a generic leakage value and the sum of the clause probabilities.

We find the probability sum as follows:

$$probTotal = \sum_{i} prob(cell \rightarrow when(i))$$

Then we use the following equations and procedures to detect and calculate the five input conditions:

1. Complete clause set with or without generic leakage value
   This condition is assumed if probTotal is equal to 1.0. For this case we calculate the weighted sum of all available clauses:
   $$leakSum = \sum_i cell \rightarrow when(i) \rightarrow value() * prob(cell \rightarrow when(i))$$
   $$leakagePower = leakSum$$
2. Incomplete clause set with generic leakage value
   For this case, we use the generic leakage value to "fill in" the missing clauses. We do this by weighting the generic leakage value with 1.0 minus probTotal.
   leakagePower = *leakSum* + (*cell_when_generic_value* () * (1.0 -*probTotal*))
3. Incomplete clause set without generic leakage value
   For this case we simply scale up the leakSum value to accommodate the missing clauses. We accomplish this by dividing it by probTotal.
   $$leakagePower = \frac{leakSum}{probTotal}$$
4. Over-complete clause set with generic leakage value (error condition)
   This case occurs when probTotal is greater than one, and there is a generic leakage value. For this case, we simply revert to the generic value (a warning is also printed to alert the user that there is a possible problem with the library).
   $$leakagePower = cell \rightarrow when\_generic\_value()$$
5. Over-complete clause set without generic leakage value (error condition)
   Without a generic leakage value, we must use the incorrect leakage data as best as we can. We do this by calculating leakSum and scaling it down. It uses the same equation as condition 3. (A warning is also be printed).

Switching Power

Switching power is calculated using the basic equation:

`SwitchingPower` = CAV^2F

Where:

C = Loading net capacitance (SPEF/DSPF or a default load value)
V = Voltage
A = Nodal activity
F = Operating frequency

The product of "A*F" is the transition density (D) calculated during the activity propagation inside the power engine. Since the calculated transition density includes both rising and falling transitions, the equation is modified for a given power rail as:

`SwitchingPower` = $1/2CV^2D$

Where a net is driven by multiple outputs, a good example is a clock mesh driven by parallel clock drivers, the capacitance is split or divided amongst the output drivers.

Vector-based Average Power Calculation

The vector-driven approach uses the VCD or TCF output of a logic simulator to obtain the number of transitions for each net. PM requires gate-level VCD or TCF with good functional coverage for accurate power calculation results.

You can use VCD or TCF information to calculate accurate power consumption figures, if the following conditions apply:

- Gate-level simulation is possible at the full-chip level.
- Gate-level simulation provides sufficient functional coverage for the design.
- The vectors include those that cause the highest power consumption.

The power engine calculates the number of transitions from 0<->1, 0/1<->X and 0/1<->Z. The 0<->1 transition is counted as 1, 0/1 <-> X transitions are counted as 0.5 by default and 0/1<->Z transitions are counted as 0.25 by default. You can use the `set_power_analysis_mode -x_transition_factor` and `-z_transition_factor` options for changing the default value of 0/1<->X or 0/1 <->Z transitions. The power engine also calculates the duty cycle of each net for state dependent internal or leakage power calculation as described previously. The power engine also takes clock definition from VCD or TCF and gives higher priority to clock definition from VCD or TCF if there is discrepancy between clock frequency in SDC or TWF and VCD or TCF.

Propagation-based average power calculation

The power engine calculates the switching probability, as well as static state probability, of each net in the design. The propagation based approach is vector-independent and provides coverage for all nets in a design.

However, the accuracy depends on good starting values, that is, information about the switching probabilities at the primary inputs in a design. Simple examples are clock and reset or enable inputs. Obtaining an accurate prediction without information about the switching probabilities of these special inputs is difficult, and in most cases an inaccurate prediction causes an over estimation of the power consumption.

Activity Propagation in the power engine

Activity propagation inside the power engine can be divided into following categories.

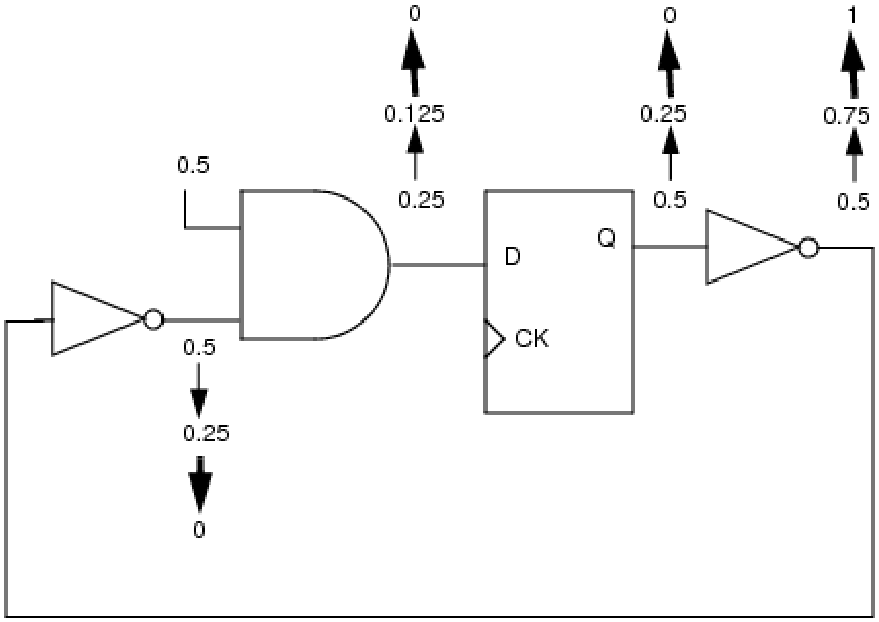
Activity propagation through combinational cells

Activity propagation through combinational cells is easier. The power engine gets function of combinational cell from `.lib` and uses the function to propagate activity through combinational cells. The power engine also propagates duty cycle through combinational cells. The only tricky part is when there are combinational loops inside design. In this case the power engine seeds activity at input to break combinational loop. The seeded activity is based upon internal heuristic of power engine and takes into account activity of other neighboring pins.

Activity propagation through sequential Cells

Activity Propagation through sequential cell is based upon activity of input pin, set or reset pin, and scan enable pin. However most of sequential cells are in sequential loops like state machines which make activity propagation through sequential cell based on heuristics by seeding activity at input of sequential cell. Therefore it is recommended to provide activity at outputs of sequential cells. With the power engine you can use either the `set_default_switching_activity` command to specify the average activity on sequential cells or use RTL, VCD, or TCF for seeding activity at sequential cells.

As seen in the example below, the activity at the output of the sequential cell in the loop can not be resolved using propagation. In this case, iterating the loop to determine the activity at the output Q of the sequential cell will result in diminishing activity towards 0. It proves that using heuristic method to compute activity in sequential loop is an intractable problem for propagation based power calculation. Therefore, in order to get good average power numbers for the design under test, user should always specify average activity at the output of sequential cell using above mentioned command. In addition, user can override this default activity using VCD or TCF.



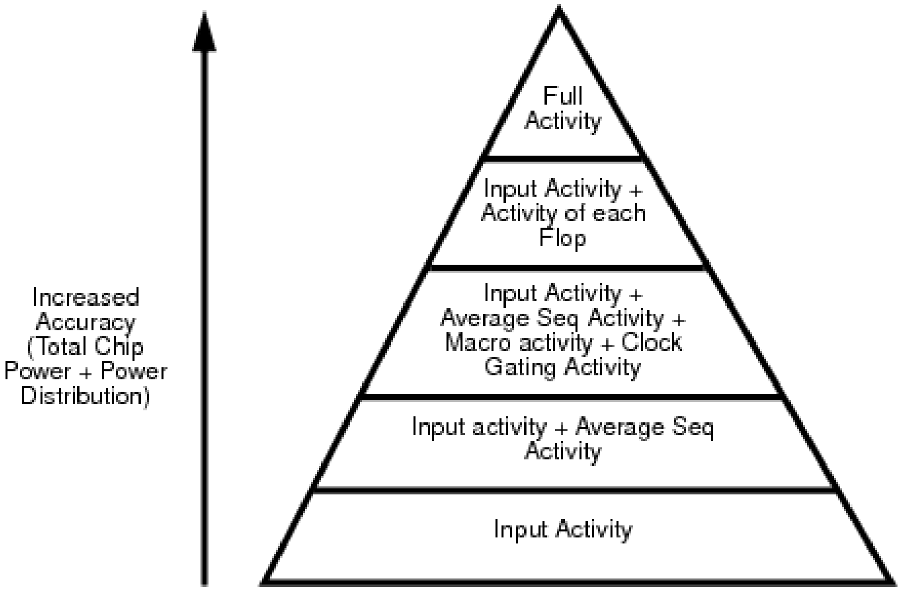
Activity propagation through macros

The major component of power in macros is internal power. Internal power of macros is highly sensitive to activity on read and writes signals. Small change in activity of read or write signal can cause large change in internal power numbers. Therefore it is recommended that users specify activity at the read and write signals of macros.

Activity propagation through clock network and clock gates

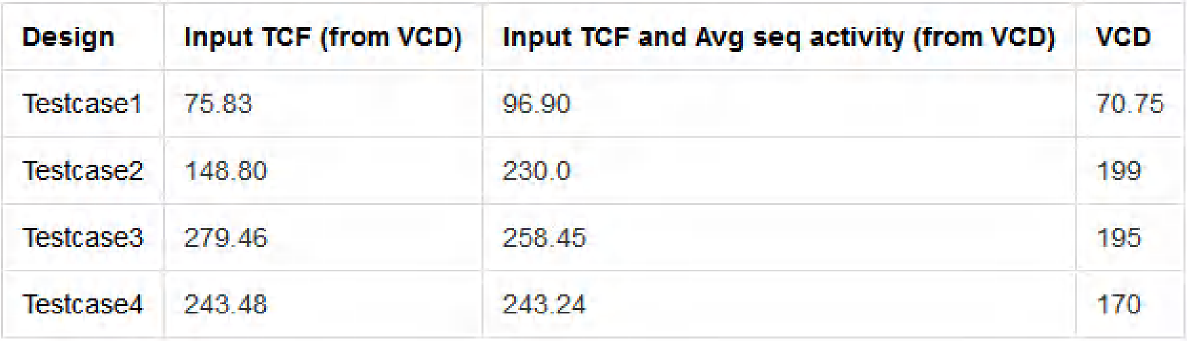
For accurate propagation through clock network it is important that user specifies TWF file which has clock frequency of generated clocks as well. The activity propagation of clock through clock gating cells depends upon activity of clock enable signal. Since clock enable is a signal net, it will generally have low activity unless specified, which will cause lot of optimism in power calculation. Therefore it is recommended that user should specify activity at enable of clock gating cells for proper propagation through clock network. You can use `set_default_switching_activity` command in Innovus for specifying average activity at enable signal of all clock gating cells.

Recommended methodology for activity propagation



The above diagram explains the recommended methodology for activity propagation. The accuracy of results increase as you specify more inputs to the power engine. At the very least user should specify the average sequential activity, which is MUST for accurate activity propagation.

Here are some of examples which depict how well the above methodology works:



Design	Input TCF (from VCD)	Input TCF and Avg seq activity (from VCD)	VCD
Testcase1	75.83	96.90	70.75
Testcase2	148.80	230.0	199
Testcase3	279.46	258.45	195
Testcase4	243.48	243.24	170

The table clearly shows that if you specify just activity on inputs then the result vary from -25% from +45% whereas if you specify average sequential activity of design then results are pessimistic by 25-30%, which is expected because combinational activity propagation is meant to be pessimistic.

Here are another two examples which show how well results correlate after specifying activity at Macros and clock gating cells.

Testcase 5:

- Expected power = ~375mW

- Power after specifying default input activity = 225mW
- Power after specifying default input activity + macro activity = 254mW
- Power after specifying default input activity + macro activity + clock gating activity = 370mW

Testcase 6:

- Expected power = 10W
- Power after specifying default input activity = 5.5W
- Power after specifying clock gating activity + Macro activity = 10.8W

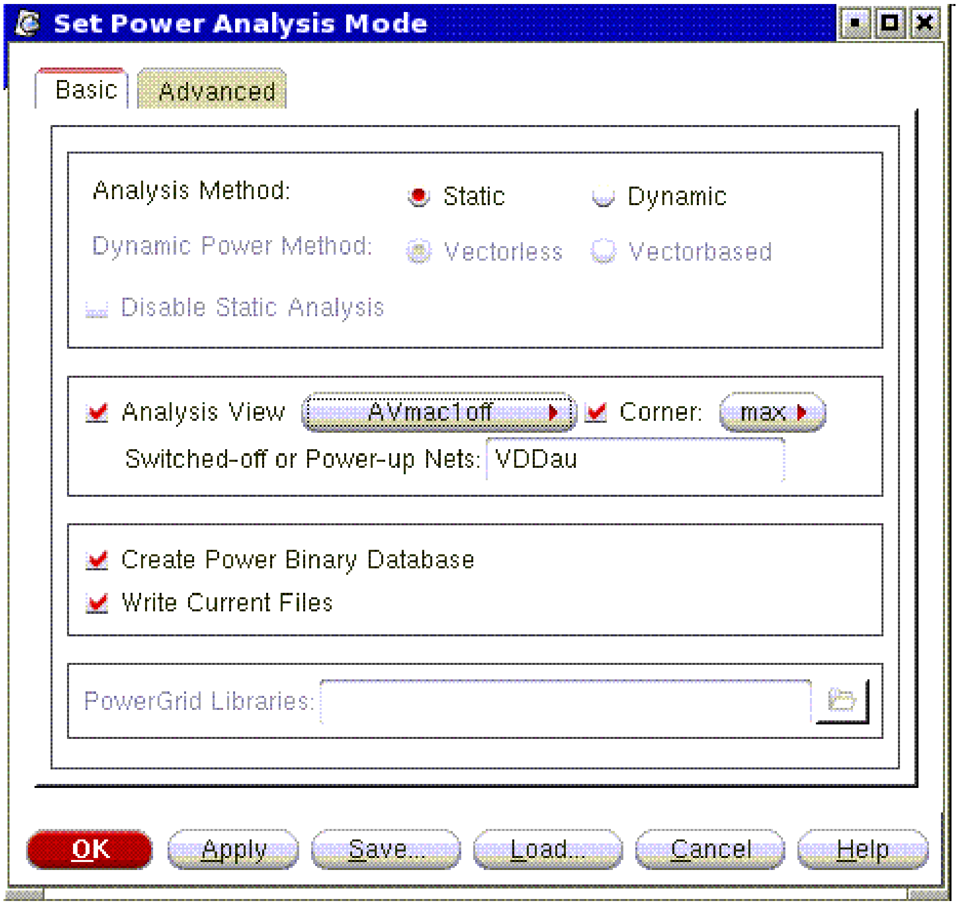
Static Power Analysis Flow

To perform static analysis one must setup the analysis mode and then run power analysis.

Set Power Analysis Mode

To setup the static power analysis mode do the following steps:

1. Select *Power - Power Analysis - Setup* form.
   The following form appears:



Set Power Analysis Mode

Basic | Advanced

Analysis Method: ● Static ○ Dynamic

Dynamic Power Method: ○ Vectorless ○ Vectorbased

□ Disable Static Analysis

☑ Analysis View AVmac1off ☑ Corner: max

Switched-off or Power-up Nets: VDDau

☑ Create Power Binary Database

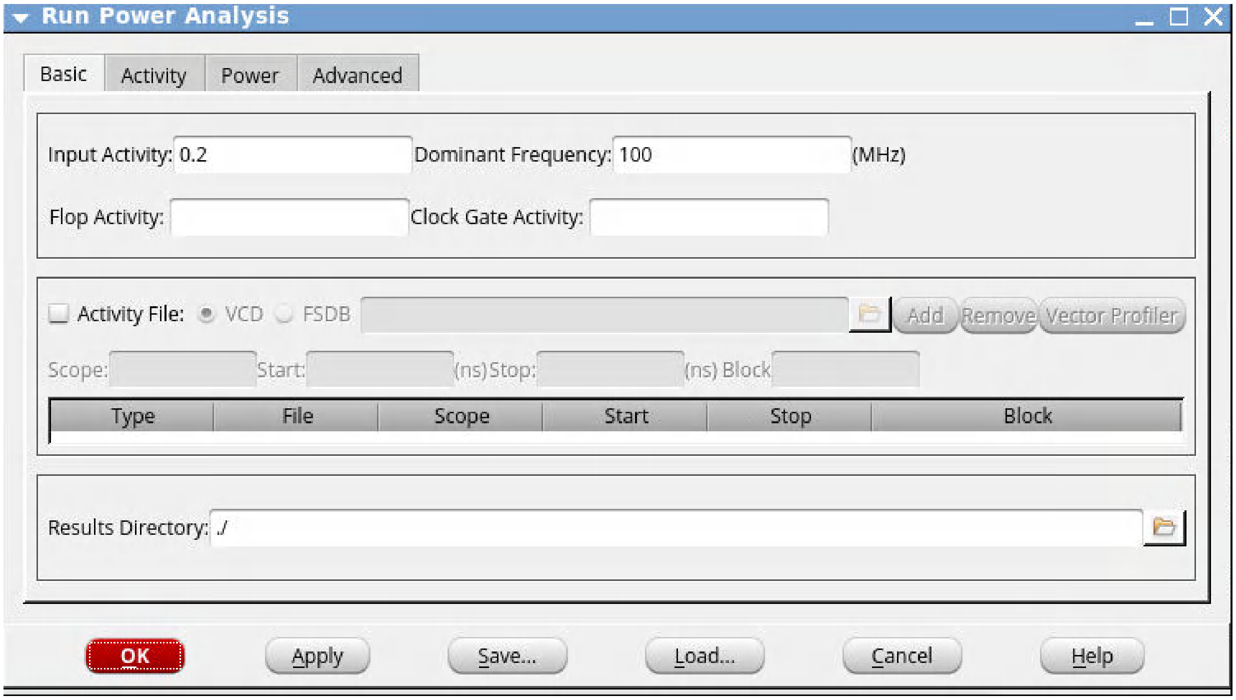
☑ Write Current Files

PowerGrid Libraries:

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*

Run Power Analysis

1. Select *Power - Power Analysis - Run*.
   The following form appears:

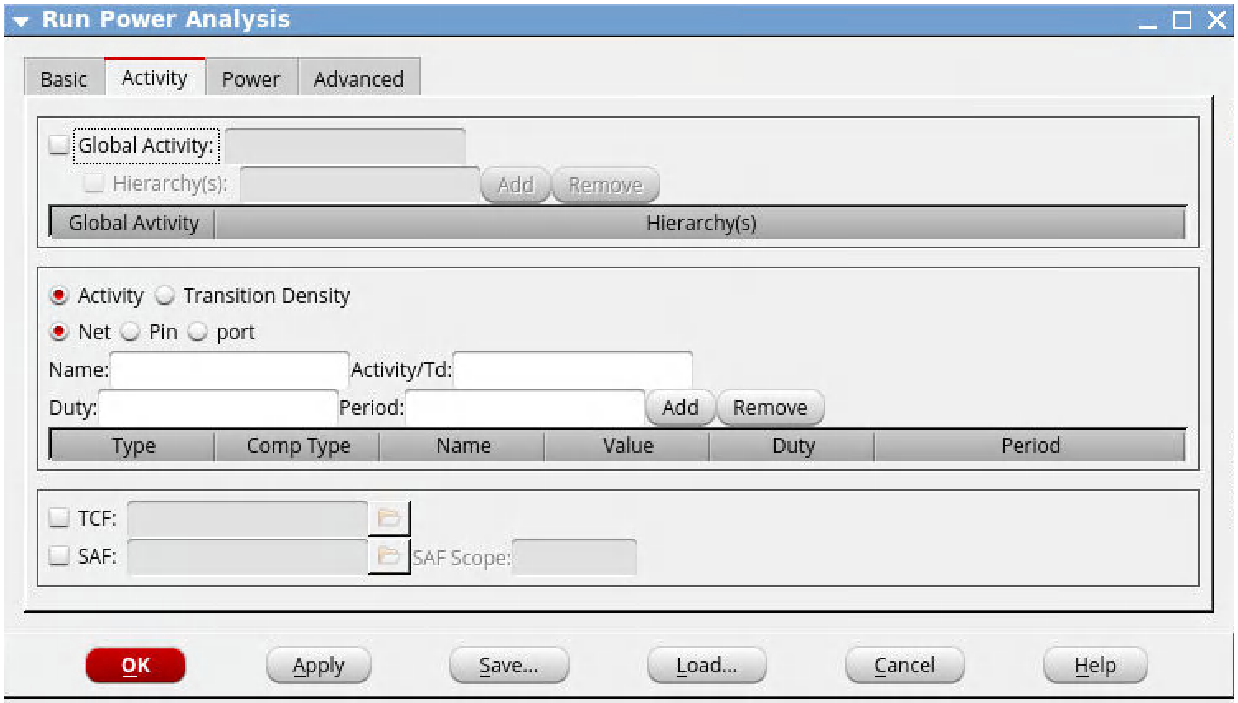


Input Activity: 0.2

Dominant Frequency: 100 (MHz)

Results Directory: ./

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.
   The following form appears:



□ Global Activity:
□ Hierarchy(s):

Global Avtivity	Hierarchy(s)

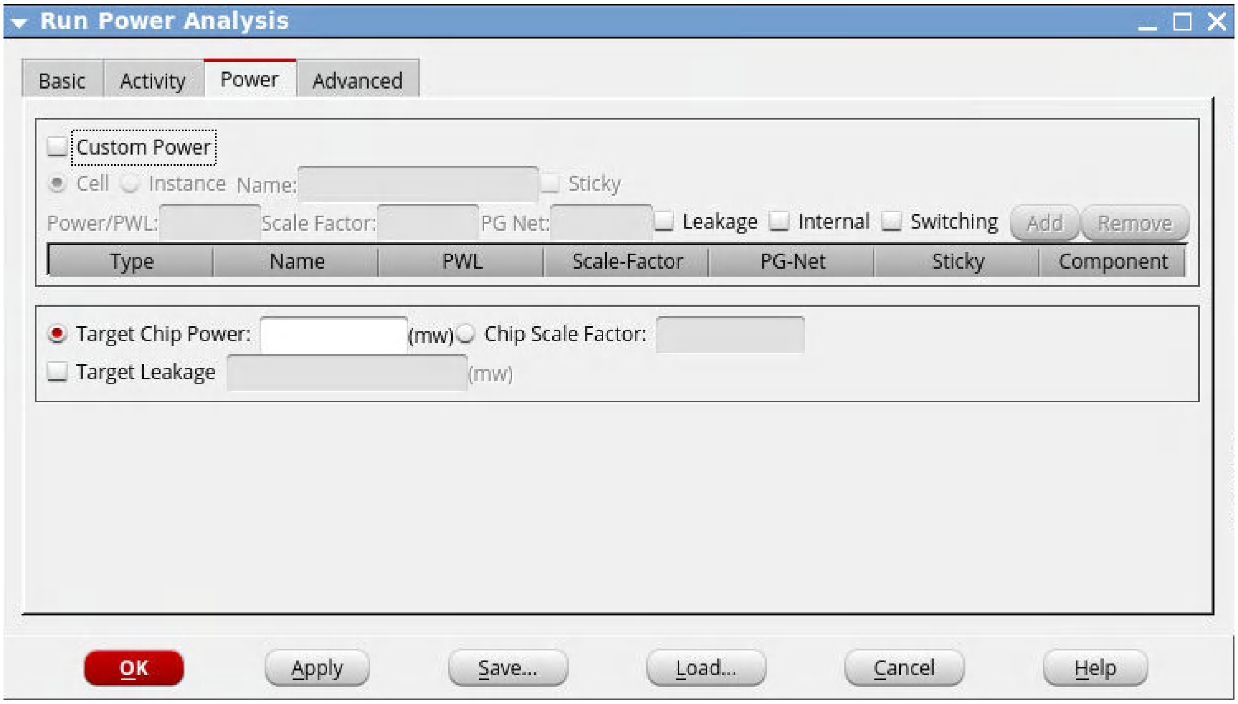
● Activity ○ Transition Density
● Net ○ Pin ○ port

Name: Activity/Td:
Duty: Period:

Type	Comp Type	Name	Value	Duty	Period

□ TCF:
□ SAF: SAF Scope:

9. Global activity can be assigned for specific parts of the hierarchy.
    a. Select *Global Activity* and specify a value.
    b. Select *Hierarchy* if needed and select the *Add* button. Continue until all necessary hierarchy global activities are defined.
10. Specify TCF or SAF file (full or partial), if available.
    a. Select *Activity* or *Transition Density*, depending on how you want to specify the values.
    b. Select *Net*, *Pin*, or *Port* to specify the type of activity that you want to specify.
    c. Specify a *Duty* or *Period,* if needed.
    d. Select the *Add* Button. Continue until all activities are specified.
11. Select the *Power* tab.
    The following form appears:



☐ Custom Power

● Cell ○ Instance Name: ☐ Sticky

Power/PWL: Scale Factor: PG Net: ☐ Leakage ☐ Internal ☐ Switching

Type	Name	PWL	Scale-Factor	PG-Net	Sticky	Component

● Target Chip Power: (mw) ○ Chip Scale Factor:

☐ Target Leakage (mw)

12. Specify *Custom Power* for *Cell*, *Instance Name* if available.
13. Select *Add* and repeat until done.
14. Select *OK* or *Apply*.

TCL Command:

The example TCL command for static power calculation is as follows:

```
set_power_analysis_mode -method static -corner max -off_pg_nets VDDau
    -create_binary_db true -write_static_currents true
set_default_switching_activity -input_activity 0.2 -period 10.0 -seq_activity 0.1
read_activity_file -format tcf design.tcf
set_default_switching_activity -global_activity 0.0 -hier top/block
set_switching_activity -activity 0 -net reset
set_power -type cell BUFX2MTL 0.010w
set_power_output_dir static_power_max
report_power -outfile design.rpt
```

Note: If you do not have a SPEF file, you can use wire load models for load specification which can then be used for static power analysis. The commands to specify wire load models are :

Note: If you have the following commands,

```
set_power_output_dir static_power_max
report_power -outfile design.rpt
```

then `design.rpt` will be dumped to `static_power_max` directory.

If you explicitly specify a directory in `-outfile` option such as below,

```
report_power -outfile my_dir/design.rpt
```

then `design.rpt` is deposited to `my_dir` directory and the path setting by `set_power_output_dir` will be ignored.

MMMC mode default views

If set to MMMC mode, it is recommended to specify the view by one of the below commands,

```
set_power_analysis_mode -analysis_view viewname
```

or

```
report_power -view viewname
```

If you do not, then the default power view is the first setup or hold view depending on the `set_analysis_mode -checkType` 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 -checkType`, 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`.

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
```

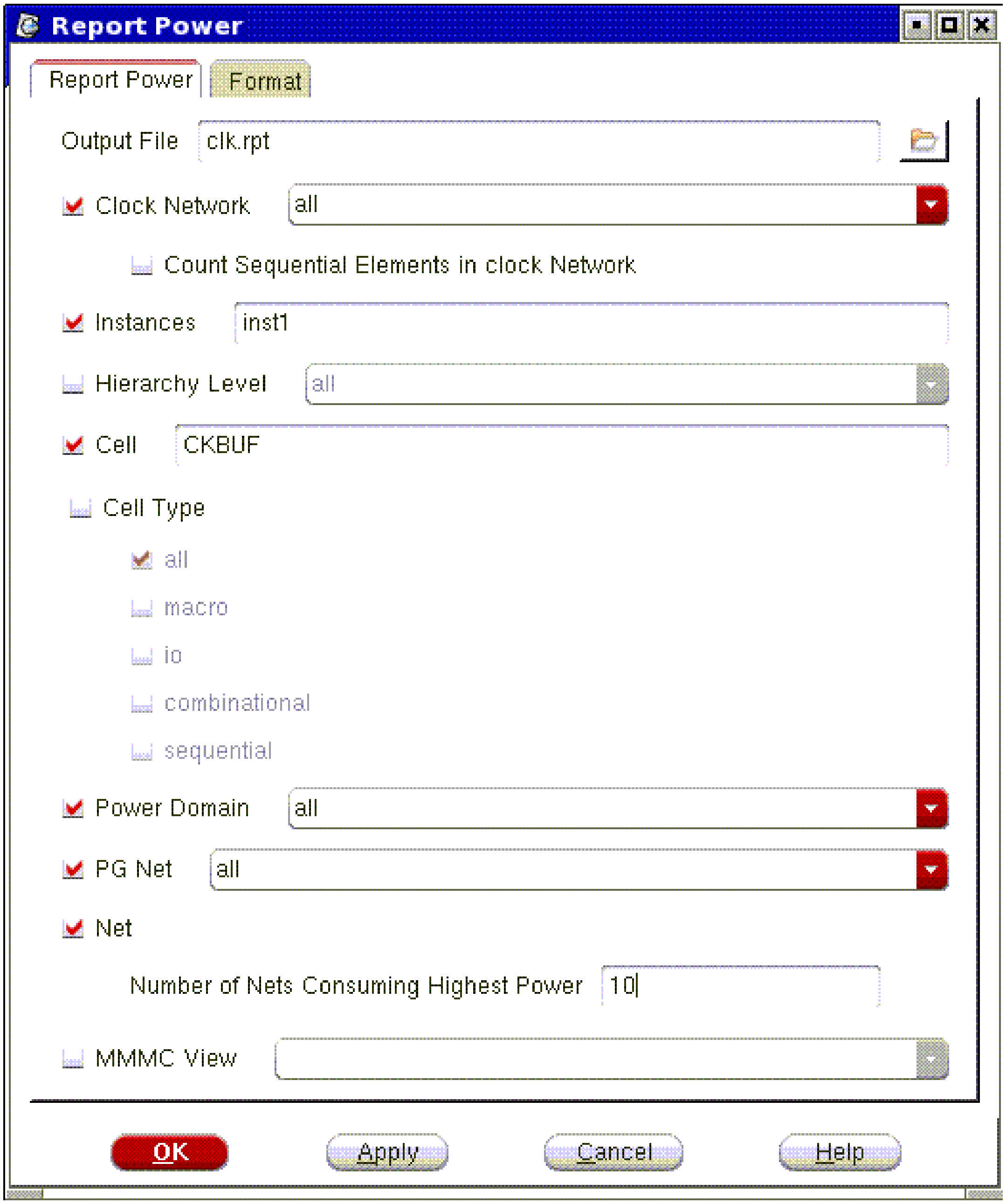
then the power view in this case is the first hold view which is `view3`.

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 the "*Configuring the Setup for Multi-Mode Multi-Corner Analysis*" section in the **Importing and Exporting Designs** chapter of the user guide.

Static Power Reports

The incremental power reports can be generated using *Power - Report - Power* which will bring up the form shown below. Select the items you want to report and *Apply*.



Output File: clk.rpt

☑ Clock Network: all

□ Count Sequential Elements in clock Network

☑ Instances: inst1

□ Hierarchy Level: all

☑ Cell: CKBUF

□ Cell Type

- ☑ all
- □ macro
- □ io
- □ combinational
- □ sequential

☑ Power Domain: all

☑ PG Net: all

☑ Net

Number of Nets Consuming Highest Power: 10

□ MMMC View

TCL Command:

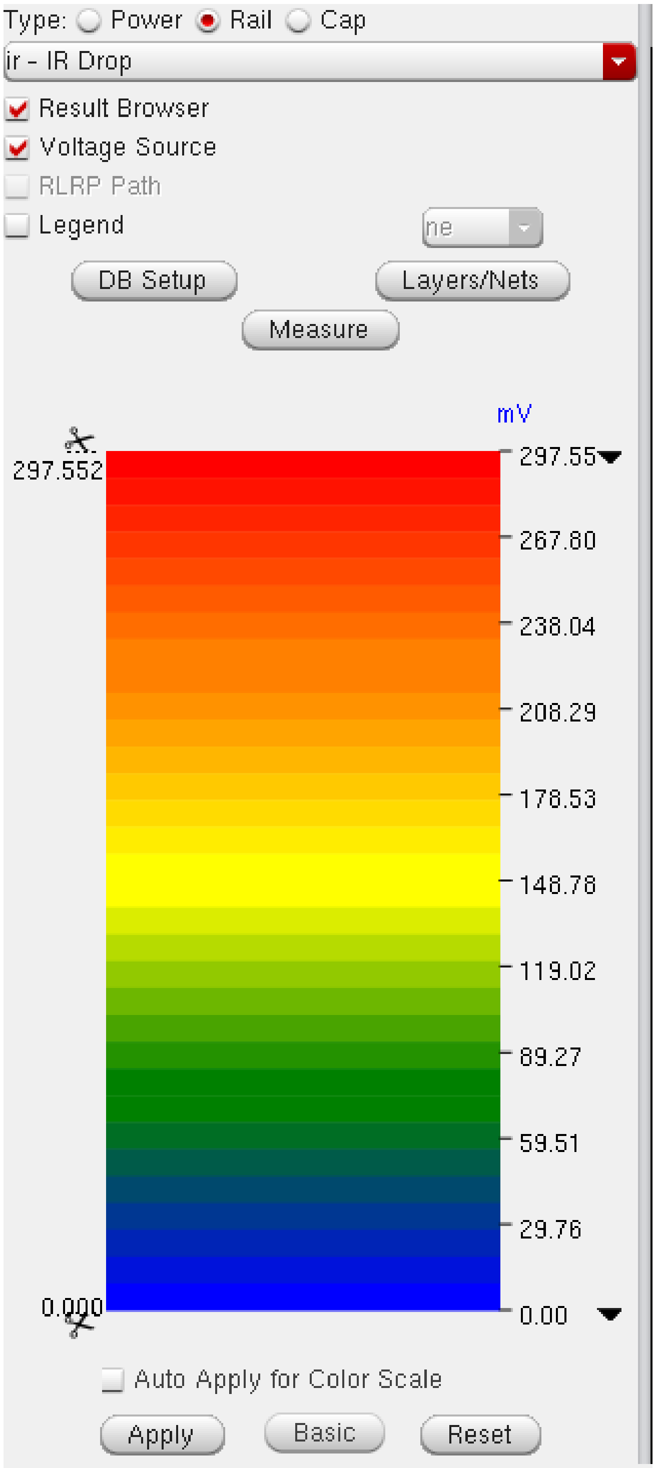
TCL command for generating the incremental report is shown below.

```
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 - Report - Power & Rail Result*.
   The Power & Rail Plots form appears. This form is the main window for displaying the power analysis results.



Type: ○ Power ● Rail ○ Cap

ir - IR Drop

☑ Result Browser

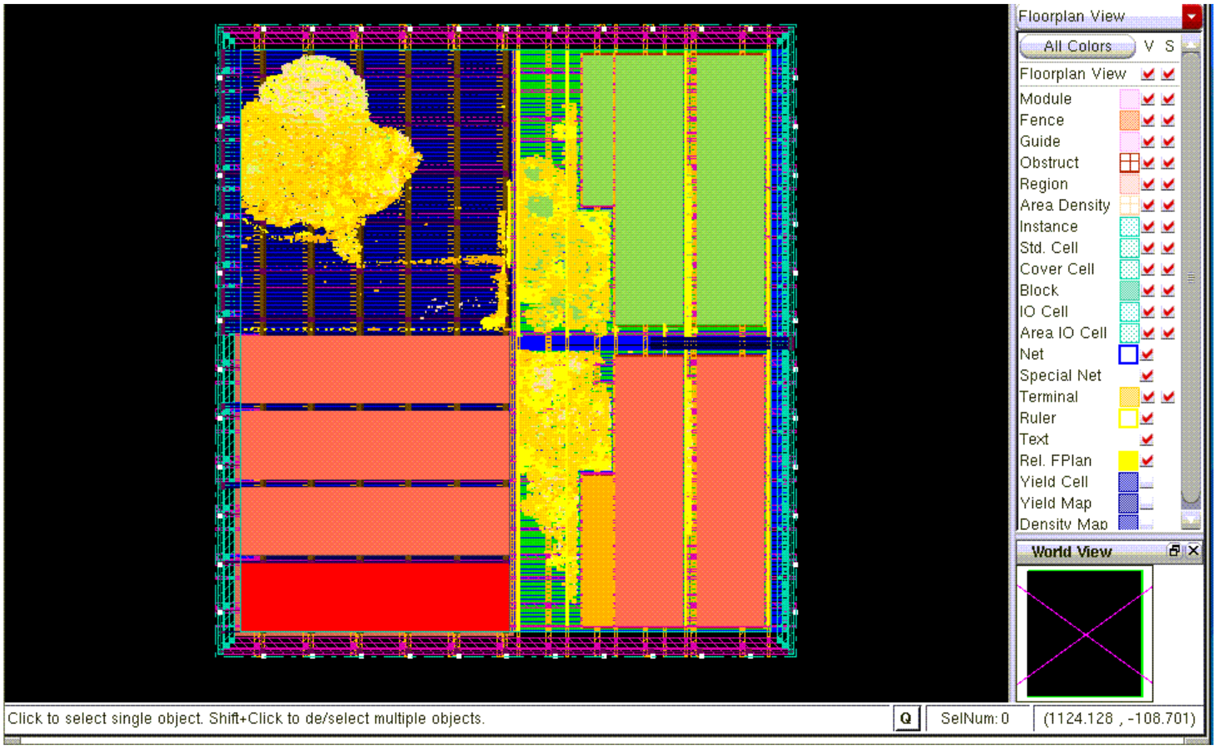
☑ Voltage Source

□ RLRP Path

□ Legend

□ Auto Apply for Color Scale

2. If power is calculated during the session, select the *Power* radio button and select a power analysis type from the drop-down list. See Viewing and Debugging Static Plots for details of the various plot types and how they can be accessed.
   The following figure illustrates the design layout overlaid with power information.



Click to select single object. Shift+Click to de/select multiple objects.

TCL Command:

Some TCL commands for viewing power analysis plots:

```
read_power_rail_results -power_db power.db
set_power_rail_display  -plot ip_s
set_power_rail_display  -plot freq
```

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 GUI:

- **Total Power - ip**
  Total power plots show power distribution of all the instances in the design. It can be used to debug regions of high IRdrop in the design

- **Internal Power - ip_i**
  Internal power is a component of total power and displayed for all instances in the design. It can be used to debug instances that consume unexpectedly a large total power. The internal power is derived from .lib file.
- **Switching Power - ip_s**
  Switching power is a component of total power and displayed for all instances in the design. The switching power plot can be used in conjunction with transition density and loading capacitance plots to understand the switching profile for the design (i.e. high activity and loading capacitance for the instance translates to high switching power).
- **Leakage Power - ip_l**
  Leakage power is a component of total power and displayed for all instances in the design. The leakage power plot can be used to debug high leakage power instances.
- **Frequency Domain - freq**
  The frequency domain plot displays the operating frequency of all instances in the design. The instances operating with multiple clock domains are displayed using the fastest clock frequency for the instance. The power is directly proportional to the frequency, so this plot can be used to debug instances with high power consumption.
- **Transition Density - td**
  Transition density is the product of frequency and activity. It is plotted for all instances in the design. This plot can be used to debug instances with high power consumption and regions of high activity.
  Transition density is plotted for the off domains in power-gated design. However, switching power plot correctly displays that no power is calculated for the off domains
- **Loading Capacitance - load**
  Loading capacitance is directly proportional to the power. It is plotted for all instances in the design. This plot can be used to debug instances that drive large output capacitance resulting in high power consumption.
- **Slack - slack**
  Slack for the instance is derived by Common Timing Engine or external Timing Window File. This plot is primarily used to identify time critical instances and can be useful when performing timing aware decap optimization and IRdrop aware timing analysis.

Interactive Queries of Power Data

The power data can be queried interactively in GUI by selecting an instance using left mouse button and pressing "Q" on keyboard. The Attribute Viewer displays total, internal, switching and leakage power for the instance. In addition it lists frequency domain, transition density and associated power domains of the instance.

Static Power Histograms and Pie-charts

The calculated static power can be debugged using pie-charts and histograms. To see the histograms, perform the following steps:

1. Select *Power* - *Report* - *Power Histogram* menu.
   The Power Debug form appears.
2. Double click on the hierarchy model (at the right of the pie chart) to explode its power distribution. Use up level to go up one level.
3. Selecting the *Histograms* tab will bring up various histograms that you can view. After this form appears, you can select other tabs to view the histograms with various types of data.
   You can search for net(s) in the *By Net*, *Net Toggle*, and *Net Probability* tabs using wildcard entries (*) and filter any net(s) to display in the histogram when debugging static power. This displays 50 nets at a time, starting with the first 50 nets. The net that you select in the histogram gets highlighted in the table, and similarly, the net that you select in the table gets highlighted in the histogram.
   Information about the selected net is displayed across all the three tabs - *By Net*, *Net Toggle*, and *Net Probability*.

Analyzing and Repairing Crosstalk

Overview

Crosstalk is the undesired electromagnetic coupling between signal lines that causes functional failures and delay variation. The effects of crosstalk might slow down or speed up the delay depending on the transition direction of the two coupling nets.

The Innovus™ Implementation System supports signal integrity (SI) operations that include crosstalk prevention during detail routing and analysis and repair afterwards. The crosstalk repair features all optimization techniques currently used for regular base timing postRoute optimization.

Inputs for SI Analysis

The design input files required are the same as needed for regular base postRoute timing optimization:

- Netlist
- SDC (timing information)
- Routed Innovus database or DEF file (placement and routing information)
- LEF file (physical library)
- XILM data (for hierarchical designs)
- Liberty library (.lib)
- Innovus extended capacitance table file
- Quantus QRC standalone extraction technology file and library (optional)

If available you can also supply the following which will be used by the Advanced Analysis Engine (AAE) timing analysis tool for extra accuracy:

- .cdB noise library

Setting Up Innovus for SI Analysis

- RC Extraction Settings
- Noise Analysis Settings
- Static Timing Analysis (STA) Settings
- Advanced Settings for SI Analysis

- Example of Setting Up Innovus for SI Analysis

RC Extraction Settings

The RC extraction settings for SI analysis include the extraction engine and the extraction filters.

Extraction Engine

You can use one of the following postRoute extraction engines:

- PostRoute
- TQuantus
- Integrated Quantus QRC (IQuantus)
- Standalone Quantus QRC

For 65nm technology and above, Innovus software uses PostRoute extraction for postRoute timing and/or optimization. However, for 65nm and below technology, if Quantus QRC technology files are available, the Innovus software uses TQuantus as the default postRoute extraction engine. For superior correlation with signoff extraction, use of TQuantus and IQuantus extraction engine is recommended. The IQuantus extraction engine provides the highest accuracy in implementation flow and is particularly recommended at ECO for incremental extraction.

Note: IQuantus extraction requires a separate QRC license.

To use the TQuantus, IQuantus, or the Standalone Quantus QRC extraction engine (the latter comes with a CPU penalty so is not usually recommended), use the following setting:

```
setExtractRCMode -engine postRoute -effortLevel [medium | high | signoff]
```

Where:

- medium: Invokes the Turbo-Quantus QRC (TQuantus) extraction engine.
- high: Invokes the Integrated-Quantus QRC (IQuantus) extraction engine.
- signoff: Invokes the Standalone Quantus QRC extraction engine.

Extraction Filters

Extraction filters enable you to reduce the total number of parasitic capacitors in the design by grounding some net to net coupling capacitance based on total net capacitance, absolute coupling capacitance size, or relative coupling capacitance size compared to total capacitance.

Effect of RC Extraction Settings on SI Analysis

Extraction coupled capacitance filtering has a significant impact on SI analysis and run time. The Innovus software automatically sets the default values for the RC extraction filters based on the process node specified using the following settings:

```
setDesignMode -process process_node
```

Note: For more information on the default values assigned to the filtering parameters with respect to the specified process node, see the `setDesignMode` command.

If you do not want to use the default filtering values for RC extraction, specify the following parameters of the `setExtractRCMode` command to adjust the coupling capacitance filters:

- `-total_c_th`: Specifies the threshold value (femtoFarads) that determines when the extractor lumps a net's coupling capacitance to ground. The software grounds the coupling capacitances for nets which have a total capacitance value less than the value specified with this parameter.
- `-coupling_c_th`: Specifies the threshold value that determines when the extractor lumps a net's coupling capacitance to ground. The software decouples the coupling capacitance of nets when the total coupling capacitance between the pair of nets is lower than the threshold specified with this parameter.
- `-relative_c_th`: Sets a ratio threshold value that determines when the extractor lumps a net's coupling capacitance to ground. If the total coupling capacitance between a pair of nets is less than the percentage (specified with this parameter) of the total capacitance of the net with the smaller total capacitance in the pair, the coupling capacitance between these two nets will be considered for grounding.

Guidelines for RC Extraction Settings

Use the following guidelines while setting up extraction:

- Note that the detailed extraction engine is significantly faster compared to TQuantus or

IQuantus. However, it trades off accuracy of the extraction results for performance. TQuantus is about 30% faster than IQuantus. TQuantus and IQuantus both support distributed processing and their use is strongly recommended to offset longer runtime. Moreover, the IQuantus engine takes advantage of incremental extraction capability in SI optimization flow to reduce runtime in subsequent cycles.

- Ensure that the filters that you are using in the Innovus software for SI fixing are the same as the ones used for SI signoff analysis.
- The default filtering values set by the Innovus software based on the process node (see section Effect of RC Extraction Settings on SI Analysis) attempt to capture the most significant effects of coupling capacitances on SI analysis. It is strongly recommended that you correlate your RCs using these default filters (set by the Innovus software) with the RCs from your signoff extractor.
- While setting the filtering thresholds, ensure that you retain small coupling capacitors because AAE-SI analysis lumps these together into a single virtual attacker model. Multiple small coupling capacitors can result in a significant virtual attacker.
- Exercise caution while setting low-value filters because this can increase the run time significantly.

Noise Analysis Settings

Noise analysis settings include loading the input noise model, configuring the timing windows, setting the delta delay threshold and specifying the virtual attacker mode.

Timing Models for SI Delay Calculation

Innovus supports NLDM, ECSM, and CCS-based Liberty (.lib) timing libraries for performing SI delay calculation. To help improve ECSM and CCS library loading times, the dotlib files may be converted into the ldb format by using the `write_ldb` command. The cdB noise libraries are also supported.

Note: For hierarchical designs, you need XILM data. For more information on XILM-based SI analysis, see the Top-level Timing Closure Methodologies chapter in the *Innovus User Guide*.

Timing Windows

Timing windows are used to filter out signals that are not switching simultaneously. The internal timing engine computes the timing windows and slew rates automatically.

Delta Delay Threshold

You can set the delta delay threshold for noise-on-delay analysis by using the following command:

```
setSIMode -delta_delay_threshold value
```

If you have a separate delta delay threshold for clock nets, you can use the following command:

```
setSIMode -clock_delta_delay_threshold value
```

Note: This will override any value specified by `-delta_delay_threshold` parameter for the clock nets.

Static Timing Analysis (STA) Settings

Static timing analysis (STA) 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

Analysis Conditions

The important analysis conditions for running SI analysis include:

- Setting Up the OCV Analysis Mode & removing common path pessimism

  Innovus provides different timing analysis modes and performs various calculations for setup and hold checks for each mode. For SI analysis, you can set the analysis mode by using the following command:

  ```
  setAnalysisMode -analysisType onChipVariation
  ```

  The OCV mode assumes that capture clock, launch clock, and data path can have maximum or minimum delays in setup and hold analysis. This is the recommended analysis mode for noise analysis and must be set before AAE analysis can be performed.

To remove the common path pessimism (recommended), you use the following command:

```
setAnalysisMode -cppr both
```

The `-cppr` parameter removes pessimism from clock paths that have a portion of the clock network in common between the clock source and clock destination paths. The pessimism is introduced when the timing analysis tools assume that the common path has different delay values for two different paths in case of on-chip variation.

- Enabling Accurate CPPR Analysis When Incremental Delays are Present

  To enable accurate CPPR analysis in the presence of incremental delays, set the following variable:

  ```
  set_global timing_enable_si_cppr true
  ```

  When this variable is set to `true`, the incremental delay is not included in the CPPR calculation during setup analysis, but is included in the CPPR calculation during hold analysis.

Advanced Settings for SI Analysis

- Multi-CPU Processing Settings
- Path Group Settings for SI Optimization

Multi-CPU Processing Settings

Innovus supports multi-threaded, distributed, and super-threaded noise analysis. Multi-CPU processing support, in general, reduces the noise analysis run time significantly.

The following command considers the multi-CPU processing settings during noise analysis:

- `opt_design -post_route -hold`
- `time_design -post_route -hold`

For information on multi-CPU processing, see "Accelerating the Design Process by Multiple-CPU Processing".

Setting Up Multi-Threading for Noise Analysis

Multi-threading enables you to run noise analysis on multiple CPUs of a single host - the host machine on which you are running Innovus or a different host. You can use the following settings:

- To setup multi-threaded noise analysis on the same host, use the following commands:

  ```
  setMultiCpuUsage -localCpu string
  optDesign -postRoute
  ```

- To setup multi-threaded noise analysis on a different host, use the following commands:

  ```
  setDistributeHost -rsh -add {name}
  setMultiCpuUsage -remoteHost 1 -cpuPerRemoteHostcpu_per_ integer
  optDesign -postRoute
  ```

Setting Up Distributed Processing for Noise Analysis

Distributed processing enables you to divide a noise analysis job between two or more networked computers running concurrently.

- To setup distributed processing using RSH, specify the following commands:

  ```
  setDistributeHost -rsh -add string
  setMultiCpuUsage -remoteHost integer
  optDesign -postRoute
  ```

- To setup distributed processing using LSF, specify the following commands:

  ```
  setDistributeHost -lsf -queue string
  setMultiCpuUsage -remoteHost integer
  optDesign -postRoute
  ```

When setting up distributed processing, ensure that there is a corresponding host computer for each view definition. For example, if the design has four view definitions, divide these between four host computers.

Setting Up Super-Threaded Noise Analysis

Super-threading enables you to run a noise analysis job on both distributed processing and multi-threaded modes; that is, two or more networked computers, each with multiple processors, can be called to concurrently run noise analysis.

- To setup super-threading using RSH, use the following commands:

  ```
  setDistributeHost -rsh -add string
  setMultiCpuUsage -remoteHost integer -cpuPerRemoteHostcpu_per_ integer
  optDesign -postRoute
  ```

- To setup super-threading using LSF, use the following commands:

  ```
  setDistributeHost -lsf -queue string
  setMultiCpuUsage -remoteHost integer -cpuPerRemoteHostcpu_per_ integer
  optDesign -postRoute
  ```

Examples of Setting Up Multi-CPU Processing

- The following settings distribute the analysis & optimization on host1 and host2 machines using RSH:

  ```
  setDistributeHost -rsh -add {host1 host2}
  setMultiCpuUsage -remoteHost 2
  optDesign -postRoute
  ```

- The following multi-threading settings distribute the analysis & optimization on eight threads on the local machine:

  ```
  setMultiCpuUsage -localCpu 8
  optDesign -postRoute
  ```

- The following settings distribute the analysis & optimization on the 5 specified host machines using RSH:

  ```
  setDistributeHost -rsh -add {host1 host2 host3 host4 host5}
  setMultiCpuUsage -remoteHost 5 -localCpu 4
  optDesign -postRoute
  ```

Note: When used together, the `-remoteHost` parameter is given preference over the `-localCpu` parameter, if the number of remote hosts specified is more than the number of CPUs specified with the `-localCpu` parameter. The `-localCpu` parameter is given preference if the number of CPUs specified with the parameter is higher than the number of remote hosts. In this case, the -`localCpu` parameter is ignored. If you use multiple hosts and multiple threads at the same time during SI optimization, use the `-remoteHost` and `-cpu_per_` parameters instead.

- The following super-threading settings distribute the analysis & optimization on host1 and host2 machines using RSH, and run eight threads on each:

  ```
  setDistributeHost -rsh -add {host1 host2}
  setMultiCpuUsage -remoteHost 2 -cpu_per_ 8
  optDesign -postRoute
  ```

  Note: In this case, a total of 16 CPUs will be used.

Path Group Settings for SI Optimization

- The SI optimization flow accounts for any user-specified path groups. If path groups are not defined, the software uses the default groups - `reg2reg`, `reg2cgate`, and `default` (all other paths). This feature enables you to take full advantage of all the path group capabilities during SI optimization.

You can use the following settings:

- `setPathGroupOptions -slackAdjustment` parameter for slack adjustment of any group.
- `setPathGroupOptions -weight` parameter to control the relative optimization priority for each group.

To create, modify, and report path groups, use the following commands:

- `group_path`
- `report_path_groups`
- `createBasicPathGroups`
- `setPathGroupOptions`
- `reportPathGroupOptions`

Example of Setting Up Innovus for SI Analysis

The following example script provides a summary of the settings that are required for performing SI analysis:

```
## Extraction Settings ##

setDesignMode -process 20
setExtractRCMode -engine postRoute -effortLevel high

## SI Settings (unless otherwise recommended it is best to use the defaults & skip
this) ##

setSIMode -individual_attacker_threshold 0.01

## STA settings ##

setAnalysisMode -analysisType onChipVariation -cppr both

## CPPR Removal for Incremental delays (optional)

set_global timing_enable_si_cppr true

## Decide what Timing Tool you will be signing off on & set the engine to match (this
is the default)

setSIMode -analysisType aae

## Ensure SI Delay Cal is on (this is the default)

setDelayCalMode -SIAware true
```

Preventing Crosstalk Violations

To prevent crosstalk violations, here are some recommendations:

1. Set reasonable max transition thresholds and ensure they are fixed before detail routing.
2. Fix transition time violations on the clock tree aggressively.
3. Shield the clock tree root to prevent Clock SI.
4. Run timing and signal integrity driven routing - use the following commands:

   ```
   setNanoRouteMode -route_with_timing_driven true -route_with_si_driven true
   routeDesign option
   ```

See *SI Prevention Techniques for PreRoute using Innovus System* application note for more advanced experiments.

Fixing Crosstalk Violations

- Data Preparation
- Using optDesign to Fix Setup Violations with Effects
- Using optDesign to Fix Hold Violations with Crosstalk Effects
- Using optDesign to Fix Transition Time Violations with Crosstalk Effects

Data Preparation

Extraction:

There are 4 options for postRoute extraction: detailed extraction, Turbo QRC, Integrated QRC & standalone QRC sign-off extraction. To correlate native extraction results with QRC sign-off extraction, compare the SPEF files from basic and sign-off extraction to generate the new scaling factors for total capacitance, cross-coupling capacitance, and resistance. Using these scaling factors, the native extraction results are closer to the sign-off extraction results, while only taking a fraction of the run time required for sign-off extraction. For more information, see "Correlating Native Extraction With Sign-Off Extraction".

Using optDesign to Fix Setup Violations with Effects

By default, using the `optDesign -postRoute` command will fix both setup base and SI timing at the same time. It will also correct glitch violations caused by incremental delays, which occur due to coupling capacitance.

- If you want to run the base timing optimization alone (without the crosstalk incremental delay), set the following:

  ```
  setDelayCalMode -SIAware false
  ```

- If you want to run the SI delay Optimization but turn off SI Glitch Optimization, set the following:

  ```
  setOptMode -opt_post_route_fix_glitch false
  ```

- If you want to turn on the SI Slew Optimization, set the following:

  ```
  setOptMode -opt_post_route_fix_si_transitions true
  ```

  For best results, set up Innovus SI analysis to match Sign-off Tool analysis before using the `optDesign` command.

Using RC Data Generated by an External Tool for SI Fixing

The RC data needs to be regenerated using a third-party tool to perform SI analysis and generate reports after optimization is complete. To perform SI fixing, load the generated RC data generated by using the `spefIn` command.

Note: Ensure that you use the `spefIn` command to load parasitic data for each corner.

Using SDF Data Generated by an External Tool for SI Fixing

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

Fixing Setup Violations Using External SDF

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

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

  ```
  setDelayCalMode -SIAware false
  read_sdf -view viewname1 -overwrite_incremental_delay view1.sdf
  read_sdf -view viewname2 -overwrite_incremental_delay view2.sdf
  ```

Using optDesign to Fix Hold Violations with Crosstalk Effects

By default, the `optDesign -postRoute -hold` command will fix both hold base and SI timing at the same time. If you want to run the base hold timing optimization alone (without the crosstalk incremental delay), set the following:

```
setDelayCalMode -SIAware false
```

Using RC Data Generated by an External Tool for SI Hold Fixing

You can load the RC data generated by an external tool by using the `spefIn` command in Innovus to do limited SI hold fixing. RC data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

Note: Ensure that you use the `spefIn` command to load the parasitic data for each corner.

Using optDesign to Fix Transition Time Violations with Crosstalk Effects

To fix SI induced maximum transition violations, set the following:

```
setOptMode -opt_post_route_fix_si_transitions true
optDesign -postRoute
```

Performing XILM-Based SI Analysis and Fixing

The model-based flow in Innovus involves generating timing accurate interface logic models (ILMs) for hierarchical blocks. To perform SI analysis, the model data generation process should account for the characterized noise library of the blocks, the timing window information of non-ILM timing path nets inside the blocks, and cross-coupling information. This data, which is an extension to ILMs, is called eXtended Interface Logic Model (XILM). An XILM is built with:

- Cells and RC network (including cross-coupling data) connected from each I/O pin to and from the first latch or flip-flop.
- eXtended Timing Window Format (XTWF) file that contains timing window and slew information of non-ILM timing paths inside the block, which might be aggressors to the nets on the ILM timing path or the top-level nets.

Note: For more information on XILM-based SI analysis, see the Top-level Timing Closure Methodologies chapter of the *Innovus User Guide*.

9

Verification Capabilities

- Identifying and Viewing Violations
- Verifying Well Pins and Bias Pins
- Integration with LPA and CCP

Identifying and Viewing Violations

Overview

The verification commands in the Innovus™ Implementation System check and report on the following types of violations:

- Connectivity: Checks for opens, unconnected wires (geometrical antennas), loops, and partial routing.
  Verify the connectivity of the design after the following design step:
- Detailed routing: For more information, see:
  - Verifying Connectivity
  - `verifyConnectivity` in the Verify Commands chapter of the *Innovus Text Command Reference* document.
- Metal density: Checks that the metal density of the metal layers is within the minimum and maximum metal density values specified by the LEF file or the `setMetalFill` command. Also checks the density of the cut layers.
  Verify the metal density after the following design step:
- Inserting metal fill: For more information, see `verifyMetalDensity` and `verifyCutDensity` in the Verify Commands chapter of the *Innovus Text Command Reference* document.
- DRC: 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 `verify_drc` to make sure the design is still DRC clean.
  Verify the DRC of the design after the following design steps:
- Placement
- Power routing
- Detailed routing
- Wire editing: For more information, see
  - Verifying DRC
  - `verify_drc` in the Verify Commands chapter of the *Innovus Text Command Reference* document. This command is currently used for 28nm and above designs.
- Process antennas and unconnected metal segments (floating areas): Checks the charge that builds up on pins caused by routing that does not have a discharge path to a gate.
  The `verifyProcessAntenna` command checks for pin routing that violates the maximum

antenna charge for the pins, and reports violations on pins that have an antenna ratio larger than the maximum allowed antenna ratio specified for the routing layer.

The `verifyProcessAntenna` command also checks for unconnected metal segments that violate the maximum area specified in the LEF file. An unconnected (floating) metal segment is a segment that is not connected to diffusion (or a polysilicon gate) through the same layer or a lower layer.

Verify process antenna and maximum floating area violations after the following design step:

- Detailed routing

  For more information, see `verifyProcessAntenna` in the Verify Commands chapter of the *Innovus Text Command Reference* document.

- AC limit: Checks for AC current violations on signal nets.
  Verify the AC limit after the following design step:
- Detailed routing

  For more information, see `verifyACLimit` in the Verify Commands chapter of the *Innovus Text Command Reference* document.

- Lithography hotspots: The software can interpret hotspot interchange format (HIF) files.
  For more information, see `loadViolationReport` in the Verify Commands chapter of the *Innovus Text Command Reference* document.
- Placement:
  For more information on the types of violations, see `checkPlace` in the "Placement Commands" chapter of the *Innovus Text Command Reference* document.
- Violation markers
  You can use text commands or GUI forms to check the violations and create the reports.

  You create violation markers with the Innovus commands, or import markers from another verification tool, such as Assura™ or Calibre, and view the markers with the Violation Browser. The Innovus software saves the markers with the database.

  For more information, see the Viewing Violations With the Violation Browser section.
  .

Interrupting Verification

The following verification commands support "Interrupt" (`Ctrl+C`):

- `verifyACLimit`
- `verifyConnectivity`
- `verify_drc`
- `verifyMetalDensity`
- `verifyPowerVia`
- `verifyProcessAntenna`

If you press `Ctrl+c` while one of the above verification commands is being executed, the verification process stops and the software displays the Interrupt menu. For information on the Interrupt menu, see "Interrupting the Software" in the Getting Started chapter.

Verifying Connectivity

Verify the connectivity of your design to detect and report conditions such as opens, unconnected pins, dangling wires, loops, and partial routing. You can use the command to create violation markers in the design window and generate a violation report. There is no database impact from using this command unless you save the design, which saves the violation markers.

For regular wires, the Innovus software checks connectivity by using the center line of the wire segments and center of the vias. For special wires, the command checks the whole DRC. 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 DRC loop violations.

Note: The Verify Connectivity feature now uses `setMultiCpuUsage` and other multi-CPU commands for multi-threading.

For more information, see the Multiple-CPU Processing Commands chapter in the *Innovus Text Command Reference* document.

Before You Begin

Before you verify connectivity, the following conditions must be met:

- The design must be routed.
- The design must be loaded into the current Innovus session.

Types of Connectivity Violations Reported

- Antennas (Dangling wires)
  Unconnected wires (dangling wires).
- Opens
  Parts of nets, such as wires or pins, that are connected to each other but are missing a connection to the net as a whole. Marks each part of a net that is missing a connection as an open and displays a violation marker between the parts.

  Violation markers for opens are displayed as polygons that include all wires, pins, and vias that connect to an island.

By default, `verifyConnectivity` checks the connectivity on masterslice layers. If the `-noSoftPGConnect` option is specified, connectivity on these layers is ignored and checking of soft Power/Ground connects is disabled.

- Loops
- Unconnected pins
  Pins that are not connected to any other objects

Note: In releases prior to 7.1, `verifyConnectivity` marked nets with connected pins but without any wiring as unrouted nets. `verifyConnectivity` no longer marks these nets as unrouted, so they do not cause violations.

Note: From the 10.1 release, `verifyConnectivity` recognizes the PG pin with DIRECTION OUT or CLASS CORE/BUMP as a strong connection. So, if you define PG PIN DIRECTION as OUTPUT or any PORT with CLASS CORE/BUMP, `verifyConnectivity` would treat that PIN as strongly connected.

For more information, see *Verify Connectivity* in the *Verify Menu* chapter of the *Innovus Menu Reference* document.

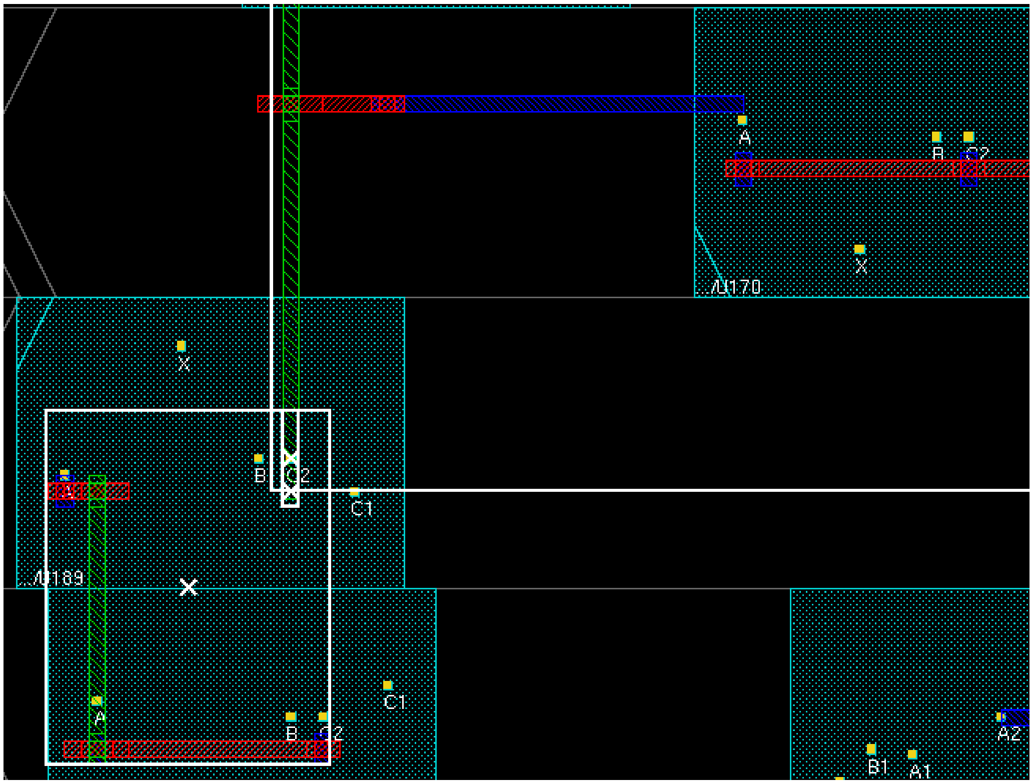
Results

After verifying connectivity, you can use information in the violation report to repair connectivity violations. You can use the Violation Browser for interactive viewing and highlighting of violation markers. You can see incremental results in the Violation Browser.

Debugging Opens Interactively

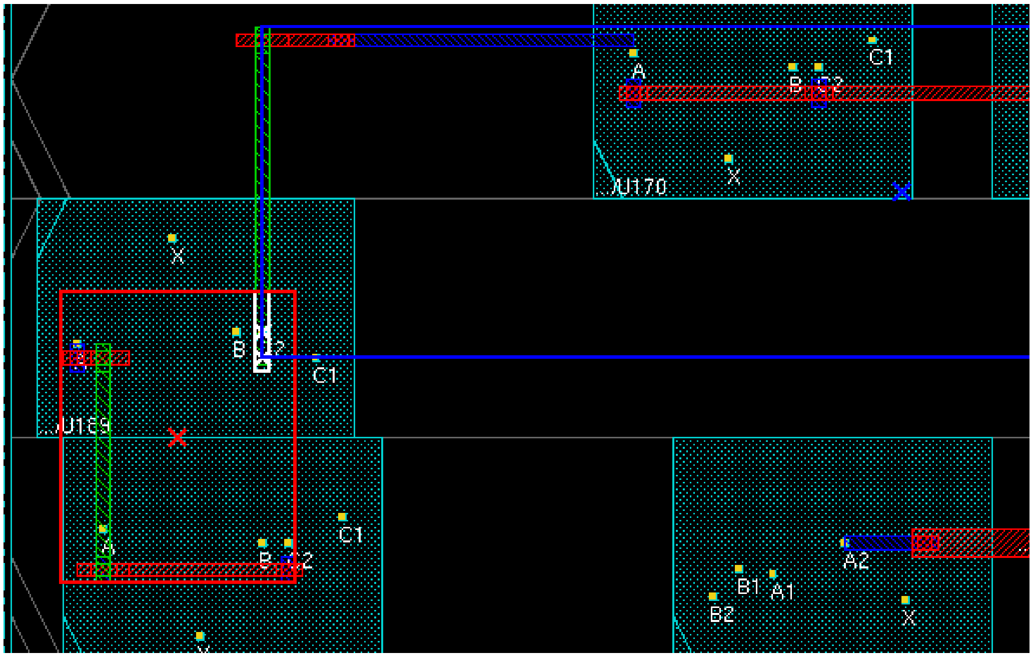
Debugging an open net can be difficult, especially when the island is big. An island is a broken segment of the net including pin, wires, and via The open markers may overlap each other, making it hard to find the island boundary. Starting from the 14.1 release, you can interactively colorize open net markers. This makes it easier to identify the open island boundary.

Suppose your design has multiple Open violations. Initially, all violations are marked in white in the main window by default.



Follow the steps below to use the colorize feature to identify different open island boundaries:

1. In the Violation Browser, click on an Open violation to auto-zoom to the open marker location.
2. Click the open net marker to which you have zoomed into in the main window. Each open net marker is then automatically assigned with a unique color. In this case, the open net markers are colored red, white, and blue.



Note: Clicking any colorized marker again to remove the colors.

3. Press the F12 key or select *Edit > Dim Background* to dim the background and view the open more clearly.
   This helps you identify the cause of the open. Typical causes are missing wires, missing vias, or unconnected pins. You can then take appropriate steps to debug the open net violation.

Verifying Metal Density

Verify the metal density of the design for each routing layer, to ensure that it is within the minimum and maximum density values specified in the LEF file or by the `setMetalFill` command.

Before You Begin

Before you verify metal density, the following conditions must be met:

- Metal density values must be specified in the LEF file or by the `setMetalFill` command. The `setMetalFill` setting overwrites the LEF rules.
- The design must be detail routed.
- The design must be loaded into the current Innovus session.

Metal Density Statements in the LEF File

The following statements in the Layer (Routing) section of the LEF file define the minimum and maximum metal density and how to analyze the density.

- `MINIMUMDENSITY`
- `MAXIMUMDENSITY`
- `DENSITYCHECKWINDOW`
- `DENSITYCHECKSTEP`
- `FILLACTIVESPACING`

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

Results

The verification process generates a report file containing information about the metal density that is not within the minimum and maximum density range.

Verifying Metal Density in Multi-Thread Mode

Accelerate metal density checking by running the software in multi-thread mode. To do so, run the `setMultiCpuUsage` command before running `verifyMetalDensity`. For example:

```
setMultiCpuUsage -localCPU 4
verifyMetalDensity
```

Related Topics

- Accelerating the Design Process By Using Multiple-CPU Processing
- Multiple-CPU Processing Commands chapter in the *Innovus Text Command Reference*

Verifying DRC

Verify the physical layout of the design by checking the width, spacing, internal DRC, and other characteristics of objects. Use the `verify_drc` 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 DRC at the following stages in the design flow:

- After placing the design.
- After adding power stripes and rings and running power routing.
- After running detailed routing.

Before You Begin

- Ensure the following LEF statements are specified:
- `CLEARANCEMEASURE`
- `USEMINSPACING` statements for obstructions and pins

  For more information, see the LEF Syntax chapter in the *LEF/DEF Language Reference*.
- If you run `verify_drc` in multiple-CPU processing mode, use the Innovus multiple-CPU commands or select the appropriate options on the Multiple CPU Processing form. For more information, see the Verifying DRC in Multi-Thread Mode section.
- Route the design.
- Set global parameters for `verify_drc` using the `set_verify_drc_mode` command.

  Note: Check the current global settings for `verify_drc` using the `get_verify_drc_mode` command.

Verify DRC Statements in the LEF File

The following statements in the LEF file can be used to define how to verify DRC.

- `ADJACENTCUTS`

- `CORNERFILLSPACING`
- `CUTCLASS SPACINGTABLE`
- `ENCLOSURE`
- `ENCLOSUREEDGE`
- `ENDOFLINE`
- `EOLENCLOSURE`
- `EOLKEEPOUT`
- `EOLSPACING`
- `EXCEPTRECTANGLE`
- `JOGTOJOGSPACING`
- `MANUFACTURINGGRID`
- `MAXVIASTACK`
- `MINSTEP`
- `MINWIDTH`
- `OPPOSITEEOLSPACING`
- `PINMASK`
- `SPACING`
- `VOLTAGESPACING`
- `WIDTH`

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

Verifying DRC in Multi-Thread Mode

You can accelerate DRC checking by running the software in multi-thread mode. Use one of the following methods:

- On the text command line:

Run the `setMultiCpuUsage` command before running `verify_drc`. For example:

```
setMultiCpuUsage -localCPU 4
verify_drc
```

- In the GUI:

  a. On the Verify DRC - 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 DRC form. When you return to the Verify DRC 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_drc`.

Related Topics

- Accelerating the Design Process By Using Multiple-CPU Processing
- Multiple-CPU Processing Commands chapter in the *Innovus Text Command Reference*
- *Verify DRC - Advanced* in the *Verify Menu* chapter of the *Innovus Menu Reference* document

Spacing Violation Checks

- `verify_drc` uses the minimum dimension of an object to check for spacing violations. The minimum dimension is the width of the object.
- The command does not detect objects with width greater than `WIDTH` and length greater than `LENGTH` that exist within a distance (`WITHIN`) greater than 10 µm for the `MINIMUMCUT` check in the LEF file.
- The command categorizes spacing violations as `SameNet`, `NonDefault`, and `ParallelRun` violations. If it finds a violation caused by a blockage between two instances of different cells, it treats the violation as a `SameNet` violation because it does not belong to a net.
- The command considers `OBS CUT` layer shapes as within the same metal if they are within the

same OBS ROUTING layer shape (the layer above or below). This avoids -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 set_verify_drc_mode -check_implant false 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 *CUT1* and *MET5* layers:

```
LAYER CUT1 TYPE CUT ;
  SPACING 0.42 ;
  SPACING 0.28 LAYER MET5 ;
END CUT1
```

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

Support for Via Rules

verify_drc uses the drc rules in the tech lef file to check for violations caused by vias. The command verify_drc does not flags violations on instance of the via by default. Run verify_drc with the -check_same_via_cell parameter to report viaCell violations for all instances.

Results

verify_drc creates markers corresponding to DRC 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 DRC-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 r epair process antenna violations. You can use the *Tools - Violation Browser* command for interactive viewing and highlighting of violation markers.

Sample Process Antenna Report

The following example shows a section of a detailed process antenna report file:

D1 (2)										
	U0 (BUF) A									
[1]		MET:	Area:	1.10	S.Area:	2.10	G.Area:	100.00	D.Area:	0.00
			Fact:	0.90	PAR:	0.01	Ratio:	0.10	(Area)	
			Fact:	1.00	PAR:	0.02	Ratio:	0.10	(S.Area)	
					CAR:	0.01	Ratio:	0.00	(C.Area)	
					CAR:	0.02	Ratio:	0.00	(C.S.Area)	
[1]		THO:	Area:	0.20	S.Area:	0.00	G.Area:	100.00	D.Area:	0.00
			Fact:	1.00	PAR:	0.00	Ratio:	0.00	(Area)	
					CAR:	0.00	Ratio:	0.00	(C.Area)	
[1]		WMET:	Area:	13.27	S.Area:	51.20	G.Area:	100.00	D.Area:	300.00
			Fact:	1.10	PAR:	0.15	Ratio:	2.50	(Area)	
			Fact:	0.90	PAR:	0.46	Ratio:	0.00	(S.Area)	
					CAR:	0.16	Ratio:	0.00	(C.Area)	
					CAR:	0.48	Ratio:	0.00	(C.S.Area)	

The report uses the following terms:

Area:	Metal area

`S.Area:`	Metal side area
`G.Area:`	Gate area
`D.Area:`	Diffusion area
`Fact:`	Metal (side) area adjusted factor
`PAR:`	Partial antenna ratio
`CAR:`	Cumulated antenna ratio
`Ratio:`	Target antenna ratio
`(Area)`	Metal area
`(S.Area)`	Metal side area
`(C.Area)`	Cumulated metal area
`(C.S.Area)`	Cumulated metal side area

Verifying Well-Process-Antenna Violations

Use the `verifyWellAntenna` command to check for any `CORE` rows that have well-process-antenna violations. Well-process-antenna violations are normally caused by rows with decap cells that do not have any protecting cells in the same row. This command marks with a violation marker any such `CORE` cell. You can then write a Tcl script to put in a well-antenna cell (protection cell) next to the violation marker.

Normally, the decap cells are the only cells that need protection. However, if the standard cell library in your design has one of the `ENDCAP` cells with a well-antenna protection device built-in, and the filler, well-tap, tie-high and tie-low cells are just class `CORE` rather than with the correct `CORE` subclass, you can use the following command:

```
verifyWellAntenna -needToProtect decap* -changeToProtect wellAntEndCap -
changeToNotProtect {filler* welltap* tie*} -report wellant.rpt
```

Sample verifyWellAntenna Report

The following example shows a section of a detailed well-process-antenna report file:

```
###############################################################
# Generated by: Cadence Innovus 15.10-b031_1
# OS: Linux x86_64(Host ID rlno-leenap)
# Generated on: Fri Apr 17 12:25:27 2015
# Design: DTMF_CHIP
# Command: verifyWellAntenna -needToProtect decap* -changeToProtect...
###############################################################

Innovus WellAntenna Verification Report
Design: DTMF_CHIP


Protects needToProtect Name Class [subclass]

1 0 ZLLLN550V15 CORE

1 0 ZHHHN550V15 CORE

1 0 XORSN550V15 CORE

1 0 XORLN550V15 CORE

..........................

...........................
```

```
0 1 EC_RTC10 ENDCAP

1 0 EC_RTC CORE

1 0 EC_RTE10 CORE

1 0 EC_RTE CORE

1 0 EC_LTE10 CORE

1 0 EC_LTE CORE

1 0 EC_BE16 CORE

1 0 EC_BE8 CORE

1 0 EC_BE4 CORE

1 0 EC_BE2 CORE

1 0 EC_BE1 CORE

1 0 EC_BE CORE

0 1 EC_TE16 CORE [FEEDTHRU]

0 1 EC_TE8 CORE [TIEHIGH]

0 1 EC_TE4 CORE [TIELOW]

0 1 EC_TE2 CORE [SPACER]

0 1 EC_TE1 CORE [ANTENNACELL]

1 0 EC_TE CORE

1 0 EC_LE10 CORE

1 0 EC_RE10 CORE

1 0 EC_LE CORE

1 0 EC_RE CORE

1 0 BCMF001N550 CORE
```

Here, cells like `EC_TE16 CORE [FEEDTHRU]` are marked as need to protect cells while cells like `EC_LE10 CORE` are marked as protect cells.

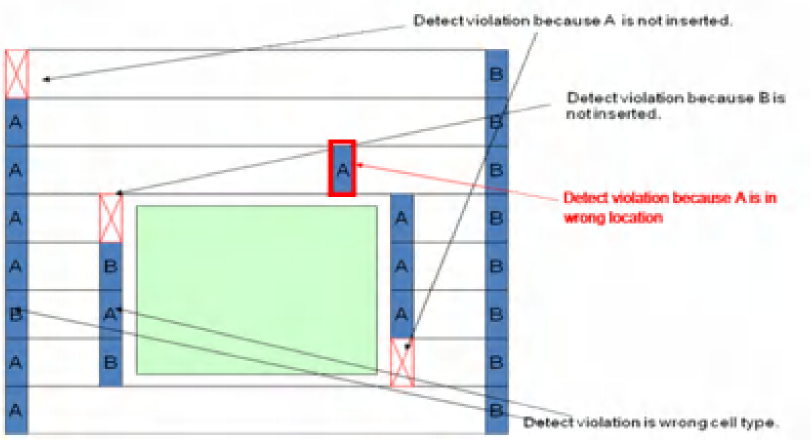
Verifying End Cap Violations

Use the `verifyEndCap` command to verify whether pre/post cap cells are inserted correctly. By default, the command marks the beginning/end of each site row where specified cap cell is 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:

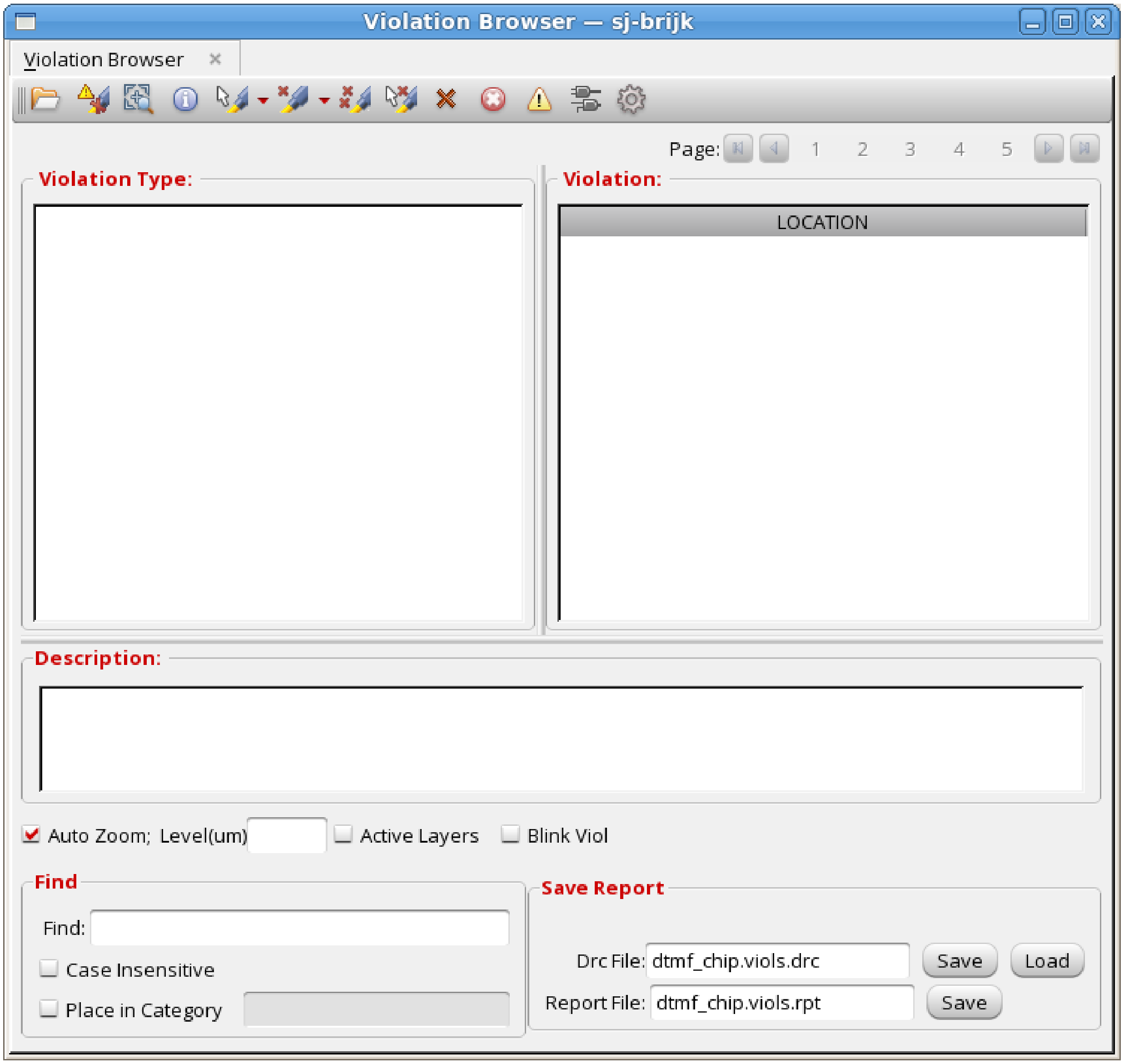
- Specified cap cell is missing. In the diagram, these locations where cap cells were expected but are missing are marked with a red cross enclosed in a rectangle ☒.
- Inserted cap cell is of wrong type, that is cell B is inserted where A is expected and vice versa.
- Cap cell is inserted in wrong location. In the diagram, cell A is inserted in the wrong location in the instance highlighted in red.


Detect violation because A is not inserted.
Detect violation because B is not inserted.
Detect violation because A is in wrong location
Detect violation is wrong cell type.

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.



☑ Auto Zoom; Level(um) □ Active Layers □ Blink Viol

Find:
□ Case Insensitive
□ Place in Category

Drc File: dtmf_chip.viols.drc
Report File: dtmf_chip.viols.rpt

Sample Verify End Cap Report

The following example shows a section of a verify end cap report file:

```
#################################################################
# Generated by: Cadence Innovus 15.10-b031_1
# OS: Linux x86_64(Host ID rlno-leenap)
# Generated on: Fri Apr 17 12:25:27 2015
# Design: DTMF_CHIP
# Command: verifyEndCap
###############################################################

Innovus EndCap Verification Report
Design: DTMF_CHIP

Endcap instance is not found at (-4738.720 4726.800 -4738.160 4732.000) on row ROW_0

 should: RightEdge

Endcap instance is not found at (-2523.360 4726.800 -2522.800 4732.000) on row ROW_0

 should: LeftEdge

Endcap instance is not found at (-2411.920 4726.800 -2411.360 4732.000) on row ROW_0

 should: RightEdge

Endcap instance is not found at (-56.000 4726.800 -55.440 4732.000) on row ROW_0

 should: LeftEdge

Endcap instance is not found at (55.440 4726.800 56.000 4732.000) on row ROW_0

 should: RightEdge

Endcap instance is not found at (2411.360 4726.800 2411.920 4732.000) on row ROW_0

 should: LeftEdge

...............................................................

................................................................

Endcap instance is not found at (55.440 -4732.000 56.000 -4726.800) on row ROW_1819

 should: RightEdge

Endcap instance is not found at (2411.360 -4732.000 2411.920 -4726.800) on row ROW_1819

 should: LeftEdge

Endcap instance is not found at (2522.800 -4732.000 2523.360 -4726.800) on row ROW_1819

 should: RightEdge

Endcap instance is not found at (4738.160 -4732.000 4738.720 -4726.800) on row ROW_1819

 should: LeftEdge
```

Verifying Maximum Floating Area Violations

Verify maximum floating area violations (unconnected metal segments whose area is greater than the maximum area specified in the LEF file) by using the `verifyProcessAntenna` command. The Innovus software checks for maximum floating area violations by default when you run this command. For more information, see `verifyProcessAntenna` in the *Innovus Text Command Reference* document.

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*.

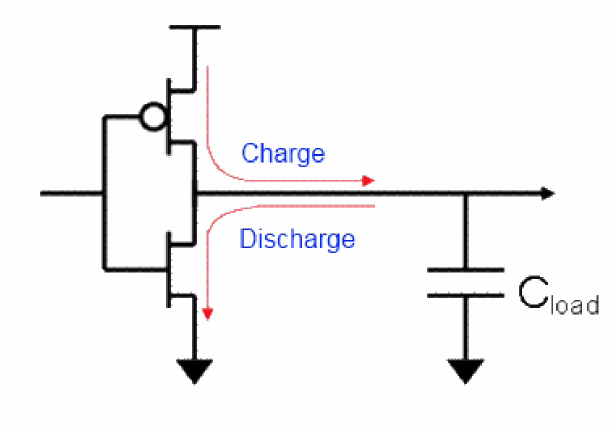
Note: To skip maximum floating area violation verification, but run process antenna verification, type the following command:

```
verifyProcessAntenna -noMaxFloatArea
```

Verifying AC Limit

Overview

The charged particles of conductor in an electric field give up kinetic energy when they collide with atomic ions. The increase in this kinetic energy of the ions manifests itself as heat and a rise in temperature. Wire self-heating or Joule Heating describes a phenomenon where high AC currents flowing through the resistive interconnects causes extreme temperature. The following diagram illustrates signal net experiencing AC current as load capacitance is charged and discharged:



To prevent wire self-heating or AC signal electromigration (EM), signal interconnects should be analyzed for their AC current carrying capacity and measured against the AC current limits specified by foundry.

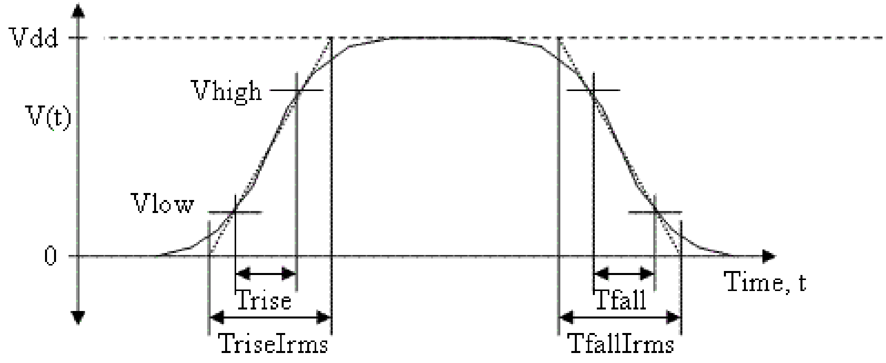
In previous versions, AC current density limits could be defined only in the Technology LEF file and only root-mean-square (RMS) current limit check was supported. From Innovus 11.1 onwards, peak AC current and average AC current limit check are also supported in addition to RMS current limit checks. Peak AC current and average AC limits must be defined in the Quantus Technology file. However, RMS current limits can be defined in both Technology LEF and Quantus 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

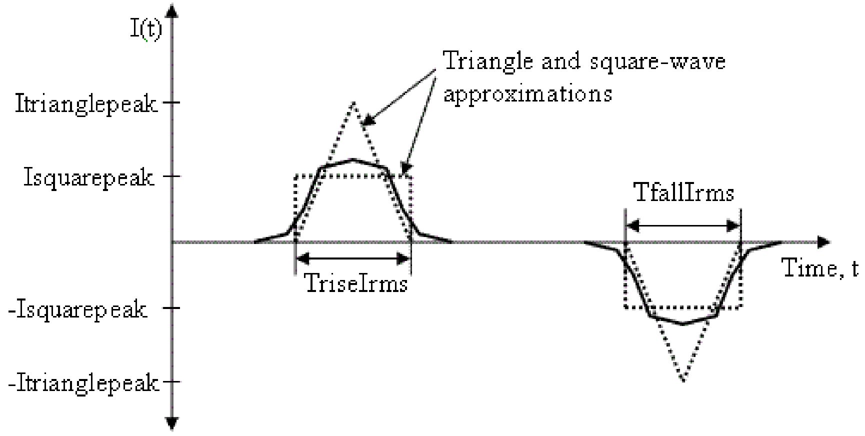
The software estimates the current waveform from the worst-case (fastest) transition time (`Trise`, `Tfall`) given by the delay calculator.

The following diagram shows the `V(t)` square waveform for a given net:



`Trise` and `Tfall` are computed by normal delay calculation between `Vhigh` and `Vlow` thresholds.

The following diagram shows the associated `I(t)` waveform with a square and triangle wave approximation super-imposed:



`TriseIrms` and `TfallIrms` are linear projections of the slew for a full transition from `0` to `Vdd`, or `Vdd` 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

```
Cnet = total capacitance on the net

Vdd = power supply voltage

Vlow = 20% of Vdd

Vhigh = 80% of Vdd
```

then

```
TriseIrms = Trise / (.8 - .2) = Trise * 1.67
```

```
TfallIrms = Trise / (.8 - .2) = Tfall * 1.67
```

and, in order to match the total current for each transition, we have:

```
Itrianglepeak(rise) = 2 * Cnet * Vdd / TriseIrms
```

```
Itrianglepeak(fall) = 2 * Cnet * Vdd / TfallIrms
```

```
Isquarepeak(rise) = Cnet * Vdd / TriseIrms
```

```
Isquarepeak(fall) = Cnet * Vdd / TfallIrms
```

Then, doing the integral for `Irms` for a triangle waveform approximation gives:

$$I_{rms(triangle)} \text{Ť} C_{net}V_{dd}\cdot\sqrt{\frac{4S}{3T_{sw}}\left|\frac{1}{T_{riseIrms}} \text{Ǵ} \frac{1}{T_{fallIrms}}\right|}$$

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. `Tsw` = Period of one switching cycle (rising + falling). And the effective frequency `Feff=S/Tsw`.

The difference is a constant sqrt(4/3) = 15%. You can scale the `Irms` value during analysis (using the `-current_scale_factor` 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 `Irms` accuracy depends on the delay calculator and delay models used, which is controlled by the `setDelayCalMode -engine` option and contents of the .lib files. `verifyACLimit` with `setDelayCalMode -engine feDC`, will use a square-wave current waveform (current is constant for the slew-time extrapolated to a 100% voltage transition) which is normally slightly pessimistic. For long wires with large resistance, it has been measured up to 20-30% worse than Spice.

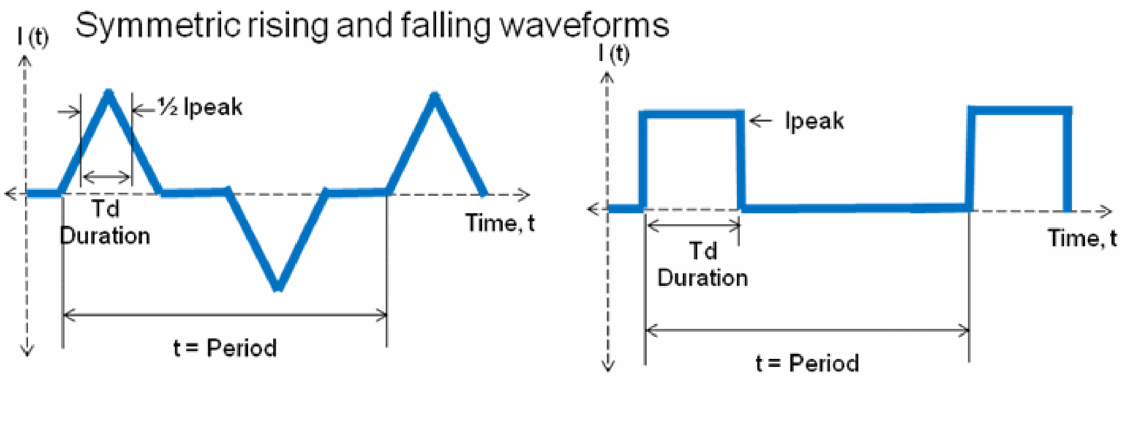
Note: For increased accuracy, you should use either SignalStorm® or AAE delay calculator instead of the default feDC engine to calculate I_{rms}. To enable SignalStorm or AAE calculation, run one of the following command before running `verifyACLimit`:

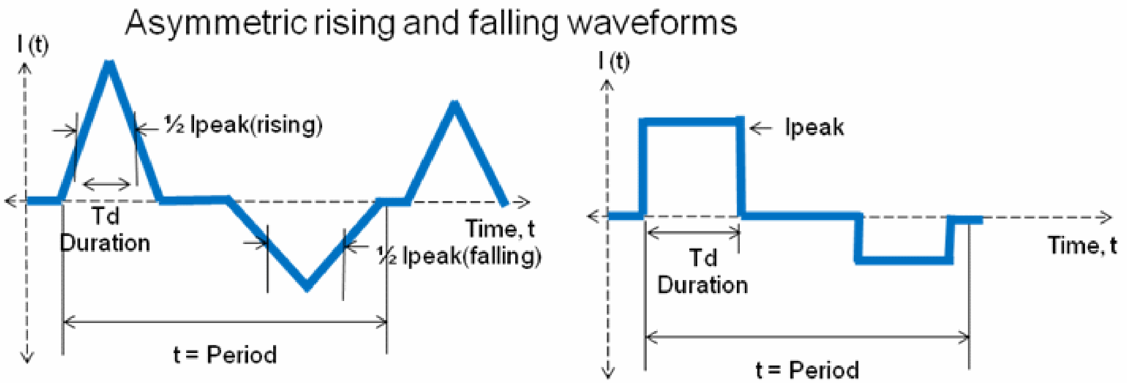
```
setDelayCalMode -engine signalStorm
setDelayCalMode -engine aae
```

Calculating I_{peak} Waveform

The software estimates the peak AC current waveform from the AAE delay calculator. The measured peak AC current, used to check against `Ipeak` provided by foundry, is defined as following:

```
peak_measured = Ipeak * sq. root[r]
```





Where, `r` is the duty ratio which is defined as pulse duration divided by the period.

```
r = Td/t
```

The pulse duration `Td` can be defined for a transition from when it reaches ½ `Ipeak` value to when it declines to ½ `Ipeak` value as shown in the above figures.

The effective duty ratio for asymmetric rising and falling waveforms is computed as follows:

Duty ratio r for I_{peak}(rising)

```
Duty ratio r(rising) = [Td(rising) + (Ipeak(falling)/Ipeak(rising))^2 * Td(falling)] /
t
```

Duty ratio r for I_{peak}(falling)

```
Duty ratio r(falling) = [Td(falling) + (Ipeak(rising)/Ipeak(falling))^2 * Td(rising)] /
t
```

Using the above duty ratios for rising and falling peak waveforms,

```
Ipeak(rising)*sq.root[r(rising)] == Ipeak(falling)*sq.root[r(falling)]
```

Therefore, the software can pick either of these values to compare against the I_{peak} limit provided by the foundry. If the value of the measured peak AC current is larger than the I_{peak} limit, it will be considered as violation. Currently, the I_{peak} limit has to be defined in the Quantus 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_{avg} Waveform

The software uses the AAE delay calculator to calculate the average current considering the recovery factor. For a rising (positive) and falling (negative) waveform, I_{avg} is calculated using the following equation:

Equation1:

```
Iavg. = abs. max (rising, falling) - EM_recovery_factor * abs. min (rising, falling)
```

where, the *`EM_recovery_factor`* is defined in the Quantus Technology file. You can overwrite this value. The default value of the EM recovery factor is `1`, that is, I_{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, `abs.Iavg(rising) = abs.Iavg(falling).`

$$I_{avg(falling)} = \frac{S}{T_{sw}} \int_0^{T_{sw}} I_{(falling)}(t)dt$$

$$I_{avg(rising)} = \frac{S}{T_{sw}} \int_0^{T_{sw}} I_{(rising)}(t)dt$$

where, `S = Switching Factor`

`T`$_{sw}$ `= Period of one switching cycle (rising + falling)`

Therefore, according to Equation1:

```
Iavg =S/Tsw* (C*Vdd - EM_Recovery*C*Vdd) = Feff *(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:

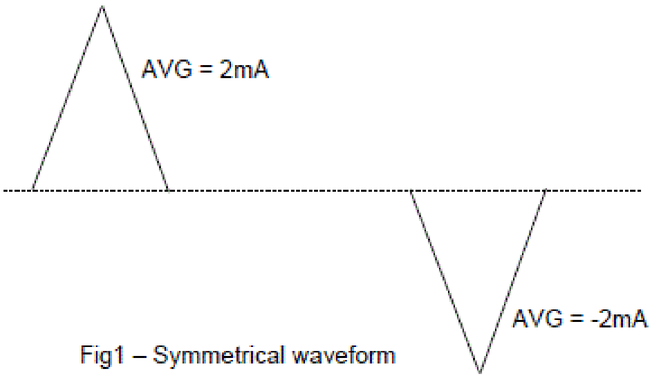


Fig1 – Symmetrical waveform

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*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
```

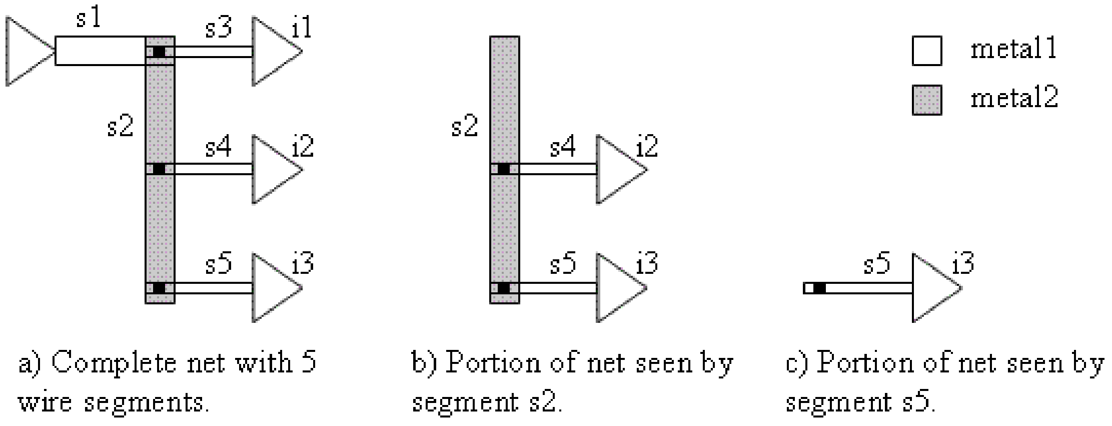
- If using the global activity on all nets is too pessimistic or optimistic, you can set the effective frequency (frequency * activity) using the TCF, or VCD files generated by netlist simulators. Optionally, you can set input and flop activities and propagate them to generate a TCF file as follows:

```
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 \

-avgRecoveryavg_recovery 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:



a) Complete net with 5 wire segments.

b) Portion of net seen by segment s2.

c) Portion of net seen by segment s5.

In this diagram, a) shows that the wire segment `s1` 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 `s1`. In b), the wire segment `s2` only carries current to charge/discharge `s2`, `s4`, `s5` and the pin-caps of `i2` and `i3`. In c), wire segment `s5` only carries current for `s5` and the pin-cap of `i3`.

The total wire capacitance (`Cwire`) 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 (`Cwire`). 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:

`Cwire(total net)` is already known from extraction

`Cnet(total net) = Cwire(total net) + Cpin for all the sinks` (driver's `Cpin` is not included)

I_{rms}`(total net)` is computed from `Cnet`, the slew rate and frequency is as described earlier

`Area(total net) = Area (s1 + s2 + s3 + s4 + s5)`

`Cwire(downstream of s2) = Cwire(total net) * [Area(s2 + s4 + s5) / Area(total net)]`

`Cnet(downstream of s2) = Cwire(downstream of s2) + Cpin(i2) + Cpin(i3)`

I_{rms} is proportional to `Cnet`, so we can estimate I_{rms}`(s2)` as:

I_{rms}`(s2) = `I_{rms}`(total net) * Cnet(downstream of s2) / Cnet(total net)`

Checking the AC Current Limits

You can verify AC current violations on signal nets by using the `verifyACLimit` command. `verifyACLimit` checks for the following types of AC current violations on signal nets:

- Root-mean-square (RMS) current limit violations (I_{rms}violations)
- Peak current limit violations (I_{peak}violations)

In addition, `verifyACLimit` can be used for average current limit (I_{avg}) analysis.

AC current limit violations are sometimes also referred as wire self-heat violations. Design Rule Check (DRC) manuals have these current limits to avoid over-heating a signal-net wire with too much AC current. To prevent wire self-heating or AC signal electromigration, signal interconnects should be analyzed for their AC current carrying capacity and measured against the AC current limits specified by foundry.

Use the `verifyACLimit -method` parameter to specify the waveform calculation method to be used (`rms`, `peak`, or `avg`).

Only the Advanced Analysis Engine (AAE) delay calculation engine supports I_{peak} and I_{avg} calculation. If you specify `-method` as `peak` or `avg` but AAE is not enabled, the tool displays the following error message:

```
**ERROR(ENCVAC-92): verifyACLimit checks for -method peak or avg requires the AAE delay
calculation engine. You must use 'setDelayCalMode -engine aae' in Innovus, before
running verifyACLimit for peak or avg checks.
```

Note: SignalStorm can still be used for I_{rms} calculations, but it does not support I_{peak} or I_{avg} calculations.

RMS/Peak/Avg Current Limit Violations

By default, `verifyACLimit` only checks for I_{rms} violations. The tool calculates the I_{rms} at the driver output and compares it to the `ACCURRENTDENSITY` tables in the LEF file that contain the I_{rms} limits for each routing layer. It generates an error and attaches a violation marker to a net if the calculated I_{rms} for a net exceeds the `ACCURRENTDENSITY` I_{rms} limit for a routing layer or width used by the net.

Note: You can use `verifyACLimit -useQrcTech` to force I_{rms} checks to use the EM rules defined in the Quantus 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 Quantus 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 `0.15um` can carry `10.0 mA/um * .15um = 1.5mA` of current.

The I_{rms} limits in the table can also be scaled for other factors like temperature. For example, the I_{rms} limits table should be computed for a specific "maximum allowed temperature change" (`maxTchange`). The I_{rms} limits are normally proportional to the square-root of the `maxTchange`. Therefore, if the table used a `maxTchange` of 20 degrees, and you want a `maxTchange` of 10 degrees, you would use a scale value of sqrt(10/20) = 0.71. The scale value would be given with the `-current_scale_factor` parameter of the `verifyACLimit` command.

The software checks the `ACCURRENTDENSITY` tables for the following conditions:

- If more than one frequency is given, `verify_AC_limit` will warn that it is only using width values from the first frequency in the table.
- There must be at least two width values, otherwise `verify_AC_limit` gives an error message and quits.
- The widths must be increasing in value, and the current limits must be increasing in value, otherwise the software gives an error message and quits.
- If no table exists for a routing layer, a warning message is given, and the limit is assumed to

be infinite for that layer.

verifyACLimit also checks the ACCURRENTDENSITY tables for the following conditions and takes the following actions:

- If there is no table for a routing layer, the software gives a warning and assumes an infinite limit for the layer.
- If PEAK and AVERAGE tables are present, the software ignores them.

Note: When AAE is used to do I_{rms} check, only hold check is supported as the worst I_{rms} occurs in hold check.

For I_{rms}, signal EM analysis not only supports the rules defined in the technology LEF file, but also the rules defined in the Quantus Technology file. But for I_{peak} and I_{avg}, only the rules defined in the Quantus Technology file are supported. If no EM rule is defined in the Quantus Technology file, the software will stop checking and give an error message.

These rules defined in the Quantus 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 qrcTechFile file in Innovus using the -qx_tech_file parameter of create_rc_corner or update_rc_corner. verifyACLimit processes the limits defined as polynomials inside the qrcTechFile file to determine layer-based AC current density limits. The Quantus Technology file attached to the current timing view (current RC corner) is used.

The qrcTechFile 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 lifetime1scale1 [lifetime2scale2.......]
```

`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 temp1scale1 [temp2scale2 ...]
```

`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 value1area1 | 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 *Quantus 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 detailed routing design step.

For more information, see `setIsolatedCutRule` and `verifyIsolatedCut` in the Verify Commands chapter of the *Innovus Text Command Reference* document.

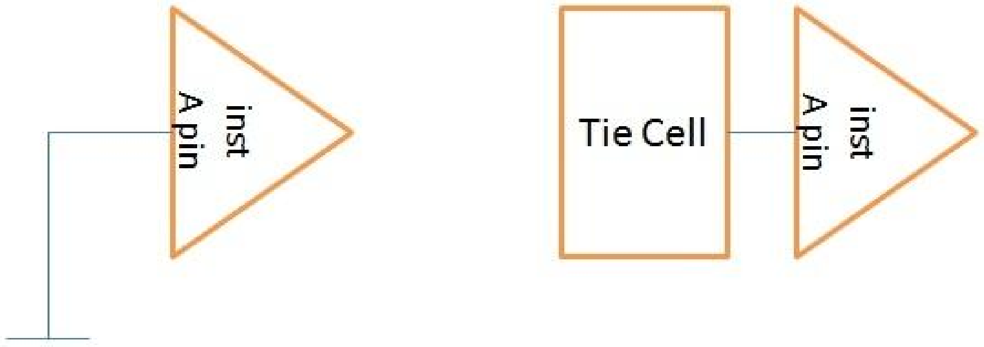
Verifying Tie Cells

Use the `verifyTieCell` command to check whether or not the tie high/low cells specified with `setTieHiLoMode -cell` are connected to the tie high/low nets.

The expected flow is as follows:

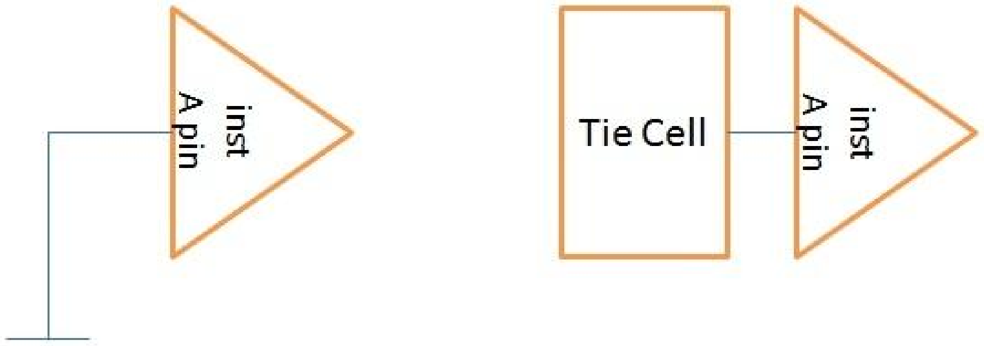
1. Specify the tie cell names and other settings with the `setTieHiLoMode` command.
2. Add tie high/low cells with `addTieHiLo`.
3. Run `verifyTieCell` to check whether tie cell connections are correct.

`verifyTieCell` flags any tie high/low nets that are not connected to tie high/low cells.



The first part of the above figure presents `verifyTieCell` violation, the second presents no violation flagged by `verifyTieCell`.

`verifyTieCell -noTieCell` *`filename`* can be used to verify that pins specified in the file are not connected to tie cells.



The first part of the above figure presents that no violation is flagged by `verifyTieCell -noTieCell`, the second presents that `verifyTieCell -noTieCell` has detected a violation.

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 DRC or Metal Density Violations

The Violation Browser updates violation markers generated by the `verify_drc` and `verifyMetalDensity` commands incrementally in an Innovus session--that is, it displays the markers generated the first time you run either of these commands and adds new markers, or deletes markers, from subsequent runs during the same session. If the software finds violations during a subsequent run that were already found previously, the browser display does not change, as there is no incremental update.

The browser can make the incremental changes because `verify_drc` and `verifyMetalDensity` can check a small area of the design and update the database. As a result of this behavior, the Innovus software saves the information from the first verification run.

Viewing Connectivity, Process Antenna, or AC Limit Violations

`The Violation Browser overwrites violation markers from the` `verifyConnectivity`, `verifyProcessAntenna`, and `verifyACLimit` commands if they are run more than once during an Innovus session. These commands are net-based, not area-based, so the browser does not make incremental updates for connectivity, process antennas, or AC limit. As a result of this behavior, the software does not keep the information from the first verification run.

Viewing Violation Markers From Assura, Calibre, Pegasus, or Other Software Applications

To view violation markers from Assura, Calibre, or Pegasus, or other software applications with the Violation Browser, use the following commands or forms:

- `createMarker`

This command creates markers from data imported from Assura or Calibre. For more information, see `createMarker` in the *Innovus Text Command Reference* document.

- `loadViolationReport` (*Tools - Violation Browser - Load Violation Report*)
  This command loads a report file from Assura, Calibre, Pegasus, or other software applications and converts it to a format that the Innovus software can interpret. For more information, see `loadViolationReport` in the *Innovus Text Command Reference* document.
- `violationBrowser` (*Tools - Violation Browser*)
  This command displays the markers in the Violation Browser. For more information, see `violationBrowser` in the *Innovus Text Command Reference* document.

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 DRC spacing violations.
- An actual value is the current value
- A target value is the value defined in the LEF file.
- Click the `+` or `-` sign to collapse or expand the listings of each violation type.
- Use the *First*, *Previous*, *Next*, *Last*, *Up*, and *Down* buttons to navigate through the list of violations.
- The browser displays the violations in the following hierarchical order:

```
+ tool
    + type
        + subtype
            Description
```

  where the `tool`, `type`, and `subtype` value correspond to the value you specify using the `createMarker` command.
- Use cross probing between the design display area and the Violation Browser.
  To display the details of a violation in the Violation Browser form, double-click the violation marker in the design display area.

- If there are violation markers for overlapped objects, select the top-most marker in the design display area and press the space bar on your keyboard to navigate through the other markers. The type and name of the selected violation is displayed in the lower-left corner of the Innovus main window. Use the `q` keyboard shortcut key to select a violation and highlight it in the Violation Browser form.
- Use the Zoom buttons to change the magnification level of a violation.
- Use any of the following buttons to change the display for a selected violation:
- *Highlight Color*
- *Highlight Violations*
- *De-Highlight Violations*
- *Delete Violations*
- *Mark Violations as False*
- *Mark Violations as True*
- Generate a report file by clicking the *Save* button.
  The report file includes information on the violations shown in the violation browser.
- Limit the number of violations to display by using the *Show Types* panel in the *Settings* page of the form.
- Limit the area to display by using the *Show Area* panel in the *Settings* page of the form.
- Filter the violations to display by using the *Other Filters* panel in the *Settings* page of the form.

Saving Violations

Choose the *Tools* - *Violation Browser* menu item and then click the *Save DRC* button to save the violation markers to a file. The DRC file can be read back with the `loadDrc` command. Alternatively, you can load the DRC file by using the GUI form. To do so, choose the *Tools* - *Violation Browser* menu item and then click the *Load DRC* button.

Clearing Violations

Choose the *Tools - Violation Browser* menu item and then click the *Clear Violation* button to clear the violation markers in your design.

Verifying Well Pins and Bias Pins

With the increase in usage of multiple power supplies, the need to be able to define well-layer information has increased. The software supports defining and verifying well pins and bias pins. The requirements for verification include:

- The `verifyConnectivity` command should detect floating wells, which are wells with no well tap connection.
- The `verify_drc` 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.

Related Information

- High-Level Flow for Verifying Well Pins and Bias Pins
- Adding Information to the Technology and Cell LEF Files
- Specifying Connections of Pins to Wells
- Validating Connections of Pins to Wells
- Exporting the Verilog Netlist
- Important Considerations for Defining Well-Layer Information

High-Level Flow for Verifying Well Pins and Bias Pins

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 `verify_drc` 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.

Related Information

- Important Considerations for Defining Well-Layer Information
- Verifying Well Pins and Bias Pins

Adding Information to the Technology and Cell LEF Files

The following information is required to be provided in the LEF files:

- Add type property to LEF LAYER MASTERSLICE to represent *WELL layers
    - `PROPERTY LEF58_TYPE "TYPE NWELL` ; " ; See Example 1 below
    - `PROPERTY LEF58_TYPE "TYPE PWELL`; "; See Example 2 below

      Note: `TYPE PWELL` is optional. It is added, if required, for triple well and substrate modeling.
- Add MACRO/PIN/PORT shapes for *WELL layers
    - For well tap cells, the layer *WELL shapes will be in the same PIN/PORT to which the physical connection is made. This could be the existing power/ground pin or a special well bias PIN. See Figure 1.

      These PIN/PORTs would need to have connections available to the routing layers. See Figure 2.
    - For regular standard cells, the *WELL layers shapes will be in their own PIN. See Figure 3.

LEF File Example (Row-Based Checking)

Example 1: PROPERTY LEF58_TYPE "TYPE NWELL

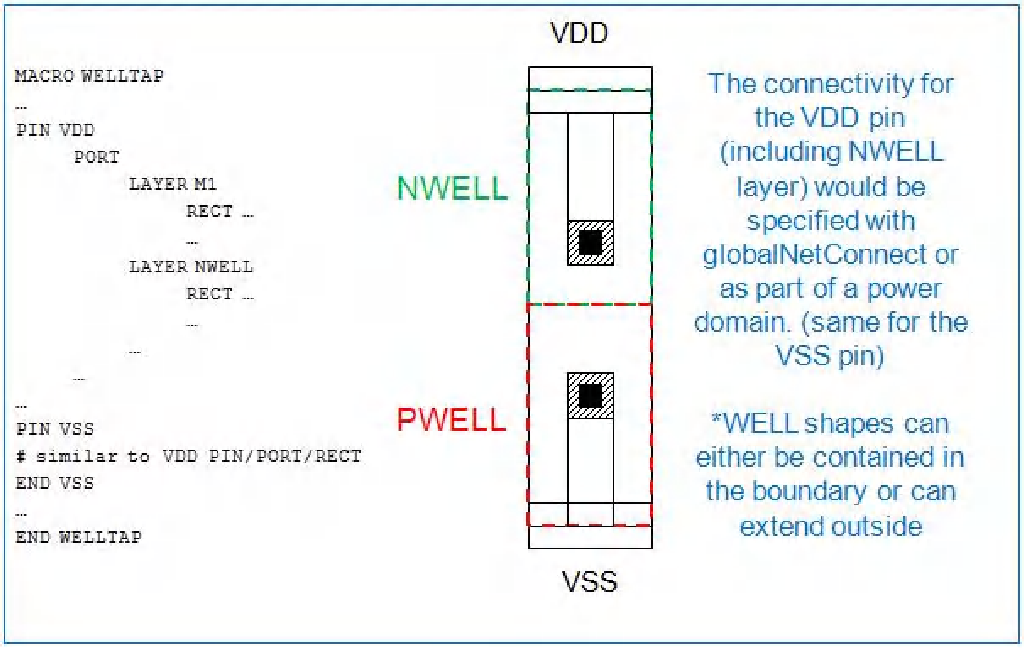
```
LAYER NWELL
```

```
TYPE MASTERSLICE ;

PROPERTY LEF58_TYPE "TYPE NWELL ; " ;

PROPERTY LEF58_SPACING "SPACING … ; " ;

PROPERTY LEF58_WIDTH "WIDTH … ; "]

END NWELL
```

Example 2: PROPERTY LEF58_TYPE "TYPE PWELL

```
LAYER PWELL

TYPE MASTERSLICE ;

PROPERTY LEF58_TYPE "TYPE PWELL ; " ;

PROPERTY LEF58_SPACING "SPACING … ; " ;

PROPERTY LEF58_WIDTH "WIDTH … ; "]

END PWELL
```

Figure 1: LEF File Example for WellTap MACRO (CORE)

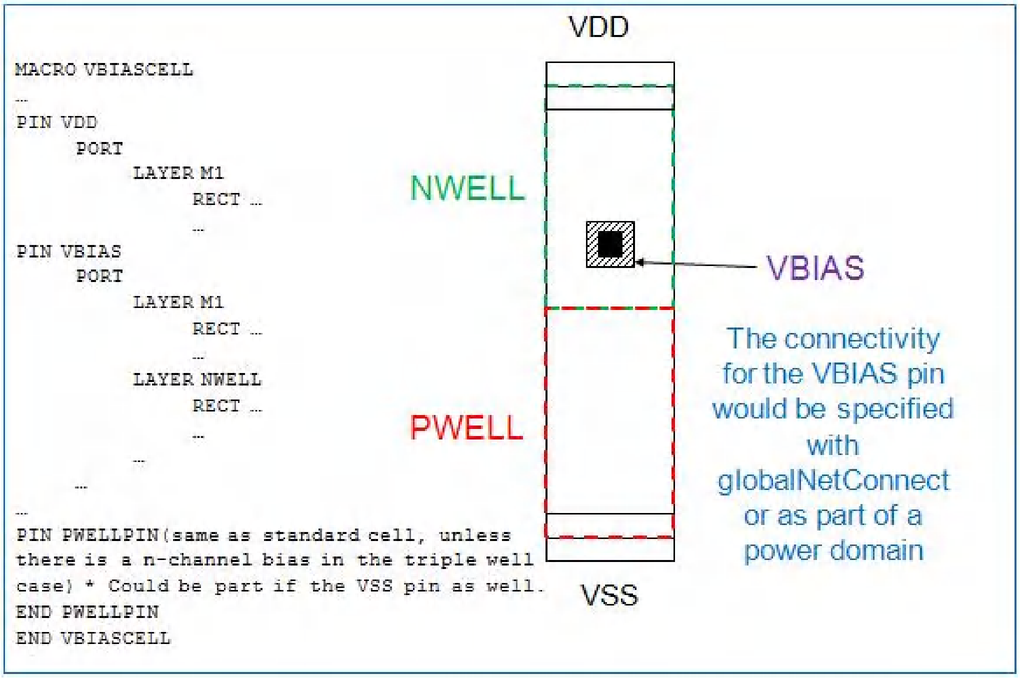


```
MACRO WELLTAP
…
PIN VDD
    PORT
        LAYER M1
            RECT …
            …
        LAYER NWELL
            RECT …
            …
        …
    …
…
PIN VSS
# similar to VDD PIN/PORT/RECT
END VSS
…
END WELLTAP
```

The connectivity for the VDD pin (including NWELL layer) would be specified with globalNetConnect or as part of a power domain. (same for the VSS pin)

*WELL shapes can either be contained in the boundary or can extend outside

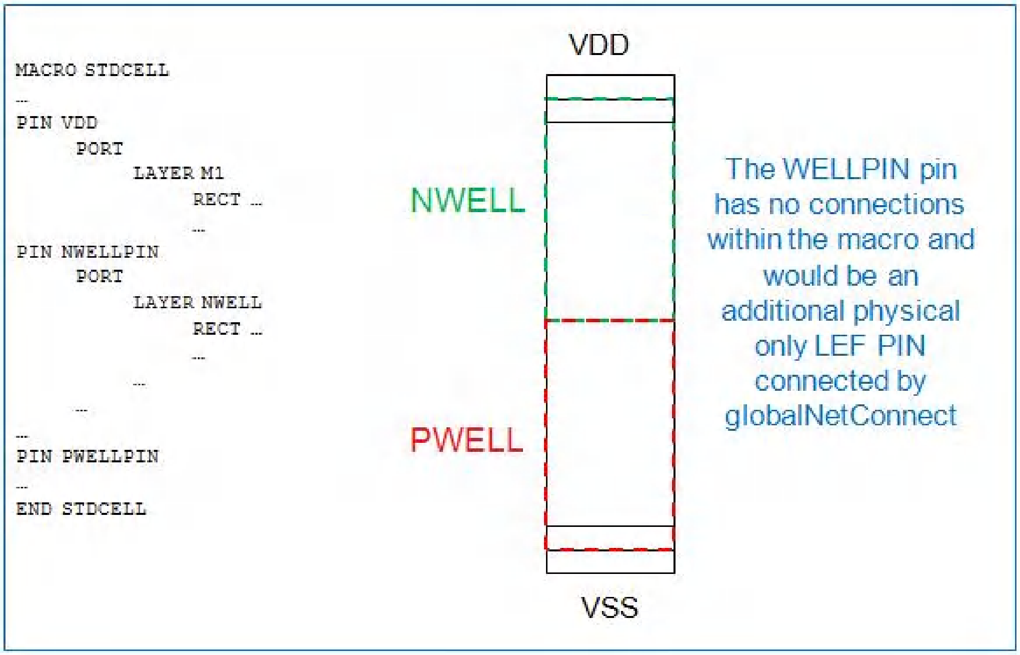
Figure 2: LEF File Example for VBIAS MACRO (CORE)



```
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



```
MACRO STDCELL
...
PIN VDD
     PORT
          LAYER M1
               RECT ...
               ...
PIN NWELLPIN
     PORT
          LAYER NWELL
               RECT ...
               ...
          ...
     ...
...
PIN PWELLPIN
...
END STDCELL
```

The WELLPIN pin has no connections within the macro and would be an additional physical only LEF PIN connected by globalNetConnect

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
```

Related Information

- Verifying Well Pins and Bias Pins
- Important Considerations for Defining Well-Layer Information

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.

Related Information

- Power Planning and Routing
- Connect Global Nets in the Power Menu chapter
- Verifying Well Pins and Bias Pins
- High-Level Flow for Verifying Well Pins and Bias Pins

Validating Connections of Pins to Wells

The width, spacing, and shorts between wells are verified using the `verify_drc` 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.

Validating Width, Spacing, and Shorts

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.

Related Information

- Verifying Connectivity
- Verifying Well Pins and Bias Pins
- High-Level Flow for Verifying Well Pins and Bias Pins

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.

Related Information

- Importing and Exporting Designs

Important Considerations for Defining Well-Layer Information

Consider the following aspects while defining well-layer information:

- The usage is applicable only for power and ground PINs
- Most PINs will be DIRECTION INOUT
- PINs on the *WELL layers do not need "SHAPE" attribute as that it only used to guide the sroute followpin behavior and the *WELL layers are connected by abutment only, no additional routing is added
- The *WELL layer shapes can abut to the boundary of the cell or extend outside
  - Typically derived directly from the layout information (GDSII or OA layout view)
  - In the case of substrate modeling, the PWELL shapes are typically created from an ANDNOT operation from the cell's boundary and NWELL shape
- If all the cells have an implied connection to the substrate and there is no "tap" connection from the topside for the PWELL, then all the PWELL shapes are included in the VSS PINs of all of the CLASS CORE cells (instead of VSS for the tap and PWELLPIN for the non-tap cells)

Related Information

- Verifying Well Pins and Bias Pins
- High-Level Flow for Verifying Well Pins and Bias Pins

Integration with LPA and CCP

Overview

The integration of Litho Physical Analyzer (LPA) and Cadence CMP Predictor (CCP) with Innovus™ Implementation System allows you to perform the foundry-recommended or mandatory lithography and CMP checks at the block and chip level in your design directly from the Innovus GUI, much earlier in the development cycle. You can run LPA during the Routing and Sign-Off phases, and CCP during the Sign-Off phase.

LPA enables you to identify litho hotspots, Design for Manufacturing hotspots, layout optimization opportunities, and predict contours across process windows based on foundry-qualified technology files. It accurately predicts manufacturing variations associated with lithography and etch. Once detected, you can fix these litho hotspots using the NanoRoute routing technology.

Note: To learn more about the Cadence Litho Physical Analyzer tool, refer to the *Litho Physical Analyzer User Guide*.

CCP, on the other hand, allows you to identify the potential yield issues that are due to the variations in interconnect thickness caused by Chemical and Mechanical Polishing (CMP). CCP accurately predicts the thickness of the interconnect and dielectric for any design and any manufacturing process that has been calibrated. The resulting prediction is then used to minimize performance loss and to identify thickness-related yield issues.

Note: To learn more about the Cadence CMP Predictor tool, refer to the *Cadence Chemical Mechanical Polishing Predictor User Guide*.

You use the *DFM* menu of the main Innovus window to configure LPA and CMP runs on the design. However, the *DFM* menu might not appear on your Innovus window or one of its submenu options might be disabled if the prerequisite conditions are not satisfied (See Before You Begin Running LPA and Before You Begin Running CCP).

Results

The output of an LPA or CCP run is an HIF file containing information about all detected hotspots. You can view this HIF file in the Innovus *Violation Browser* and fix the reported hotspots using NanoRoute.

Before You Begin Running LPA

Before you can run LPA from Innovus, the following conditions must be met:

- You must have the LPA license to run litho sign-off from Innovus. However, to run LPA in Routing Layers Only mode, the Innovus DFM Option license is sufficient and no separate LPA license is required.

- You must have either the Encounter Advanced Node GXL Option or Innovus DFM Option license.
- You must be able to launch version 9.2 (or a later version) of LPA from Innovus. In other words, the installation path to LPA must be present in your path variable.
- You must have LPA TechFiles that are compatible with the design technology.

Running LPA from Innovus

The integration of LPA with Innovus allows you to check for litho hotspots and predict contours across process windows earlier in the development cycle, much before the Sign-Off phase. The integration is smooth and easy to configure, and does not require any user intervention to stream out or set up LPA.

You can run LPA from Innovus in two modes:

- Routing Layers Only Mode
- Sign-Off Mode

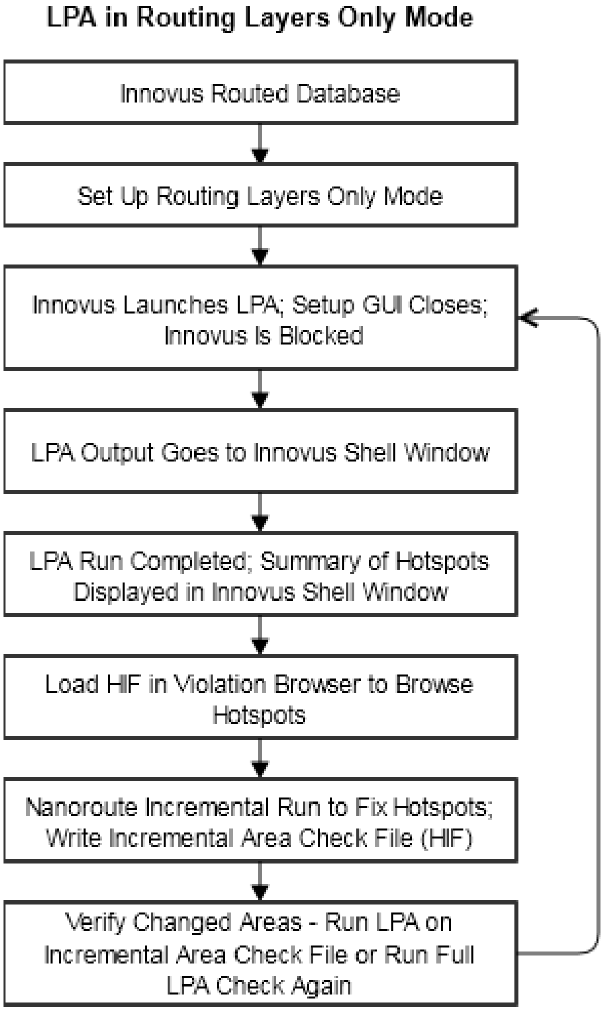
Further, both these modes also support the DRC+ verification methodology from GLOBALFOUNDRIES, in addition to the standard LPA flow. The DRC+ methodology utilizes a foundry-supplied DRC+ Pattern technology file, but the use model is otherwise identical to the standard LPA flow. For guidance on the choice of technology files, consult with your foundry.

Routing Layers Only Mode

Routing Layers Only mode is the fast mode of LPA to flag L1 hotspots in a design at the block level during the Implementation phase. In this mode, LPA runs about 100X faster than Sign-Off Mode and helps you identify and fix most hotspots as routing is completed.

Note: LPA in Routing Layers Only mode is enabled only when you have the Innovus DFM Option license.

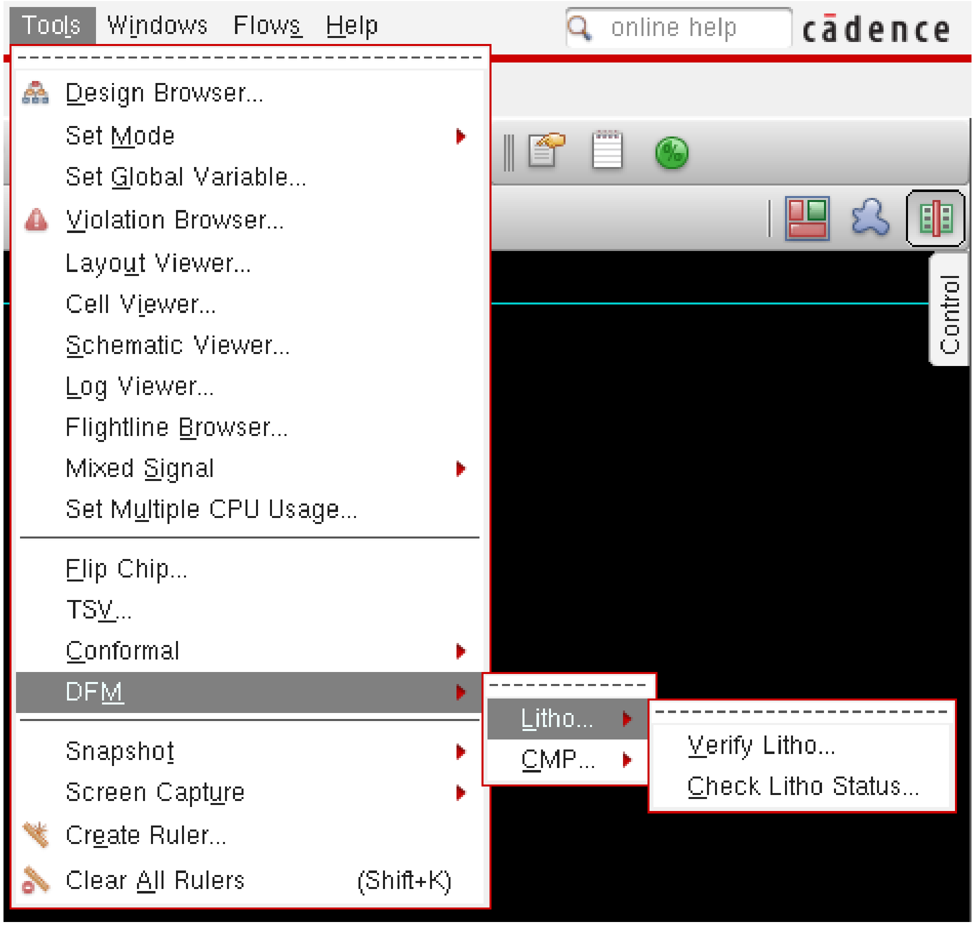
The following diagram shows the design flow for running LPA in Routing Layers Only mode.


LPA in Routing Layers Only Mode
Innovus Routed Database
Set Up Routing Layers Only Mode
Innovus Launches LPA; Setup GUI Closes; Innovus Is Blocked
LPA Output Goes to Innovus Shell Window
LPA Run Completed; Summary of Hotspots Displayed in Innovus Shell Window
Load HIF in Violation Browser to Browse Hotspots
Nanoroute Incremental Run to Fix Hotspots; Write Incremental Area Check File (HIF)
Verify Changed Areas - Run LPA on Incremental Area Check File or Run Full LPA Check Again

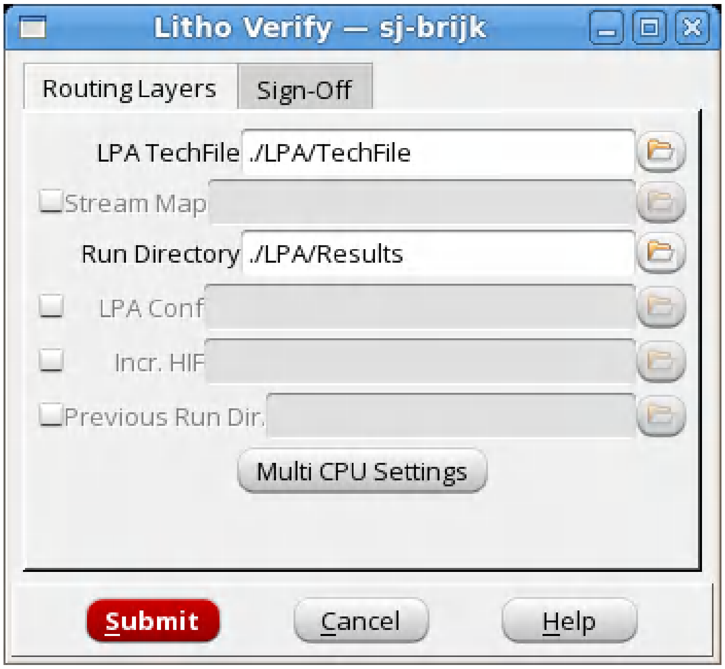
Running LPA in Routing Layers Only Mode from the GUI

To run LPA in Routing Layers Only mode, launch Innovus by using the `innovus` command and load the design. When Innovus is invoked, it automatically loads all the required LPA files from the LPA installation path. Once the Innovus GUI is displayed, perform the following steps:

1. Choose *Tools -> DFM -> Litho -> Verify Litho* from the Innovus menu.



2. This opens the *Litho Verify* form, with the *Routing Layers* tab selected by default. In the *LPA TechFile* field, specify the path and name of the qualified LPA Mx- technology file that includes process-specific hotspot checking options and the LPA model. Alternatively, if you want to enable the GLOBALFOUNDRIES DRC+ flow, you specify the path to the foundry-supplied DRC+ Pattern technology file in the *LPA TechFile* field.



LPA TechFile ./LPA/TechFile

□ Stream Map

Run Directory ./LPA/Results

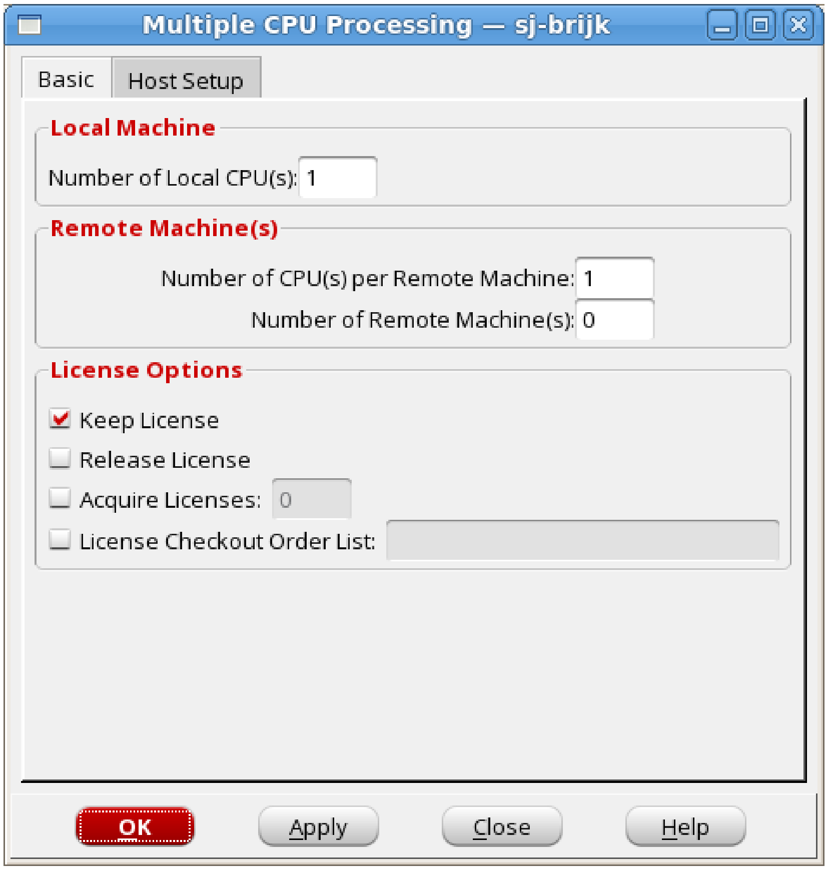
□ LPA Conf

□ Incr. HIF

□ Previous Run Dir.

3. In the *Stream Map* field, specify the stream out map file, which maps the GDS stream to the layers in the Innovus database. Ensure that the GDS layer numbers in the stream out layer map matches the numbers in the LPA layer map.
4. In the *Run Directory* field, specify the LPA output directory that will contain one subdirectory for each layer when LPA is run with the configuration file for that layer.
5. *LPA Conf* is an optional field. Select this check box and specify the name and path of the LPA custom configuration file. This file, if specified, controls all run options of LPA.
6. *Incr. HIF* is also an optional field. Select this check box and specify the name and path of the HIF file that you want to use for incremental validation. This HIF file includes the locations that identify the areas affected by each hotspot fix. LPA reads these locations and performs incremental checking only in these areas. This reduces the time for validation.
7. Another optional field is *Previous Run Dir*. You use this option when the design has been changed but no HIF file that identifies the changed areas is available. In this field, you specify the path to the run directory of a previous Verify Litho run that you want to use for XOR-based incremental validation. When this option is selected, the previous run results are compared to the new design and LPA is run only in locations where the layout has changed and where hotspots previously existed, thereby reducing the overall validation run time.
8. By default, LPA uses the LSF settings from the *Multi-CPU Settings* GUI in Innovus. However, you can change the multi-CPU settings for the LPA run by selecting the *Multi CPU Settings* button and specifying the new settings in the *Multiple CPU Processing* form. Note that this will

also change the distributed options for all Innovus commands.



Local Machine

Number of Local CPU(s): 1

Remote Machine(s)

Number of CPU(s) per Remote Machine: 1

Number of Remote Machine(s): 0

License Options

☑ Keep License

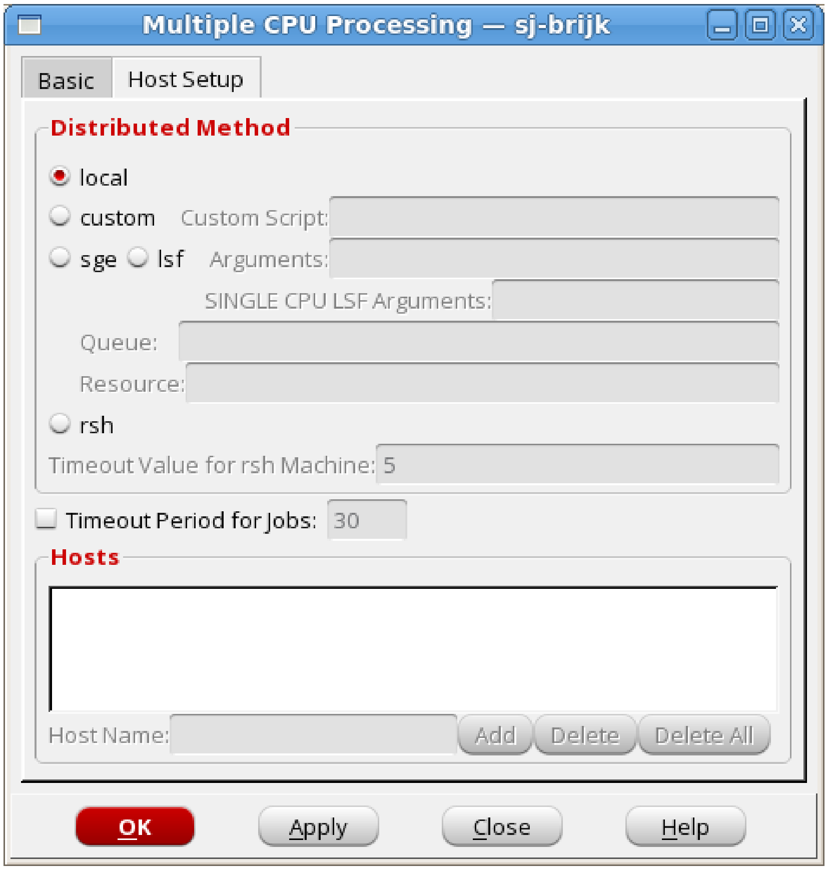
□ Release License

□ Acquire Licenses: 0

□ License Checkout Order List:

9. On the *Basic* tab of the *Multiple CPU Processing* form, if running locally, use the *Number of Local CPU(s)* field to set the number of local CPUs used. If using LSF to run, the field *Number of Remote Machine(s)* field specifies the number of LSF machines that you want to use for the LPA run.

10. On the *Host Setup* tab, select the *lsf* radio button to set the distribution method as LSF and specify the LSF arguments in the *LSF Arguments* field or select the *local* radio button to use local cpu(s) only.
    Note: It is recommended to use local or LSF distribution method for the Innovus-LPA integration.



Distributed Method

● local

○ custom Custom Script:

○ sge ○ lsf Arguments:

SINGLE CPU LSF Arguments:

Queue:

Resource:

○ rsh

Timeout Value for rsh Machine: 5

□ Timeout Period for Jobs: 30

Hosts

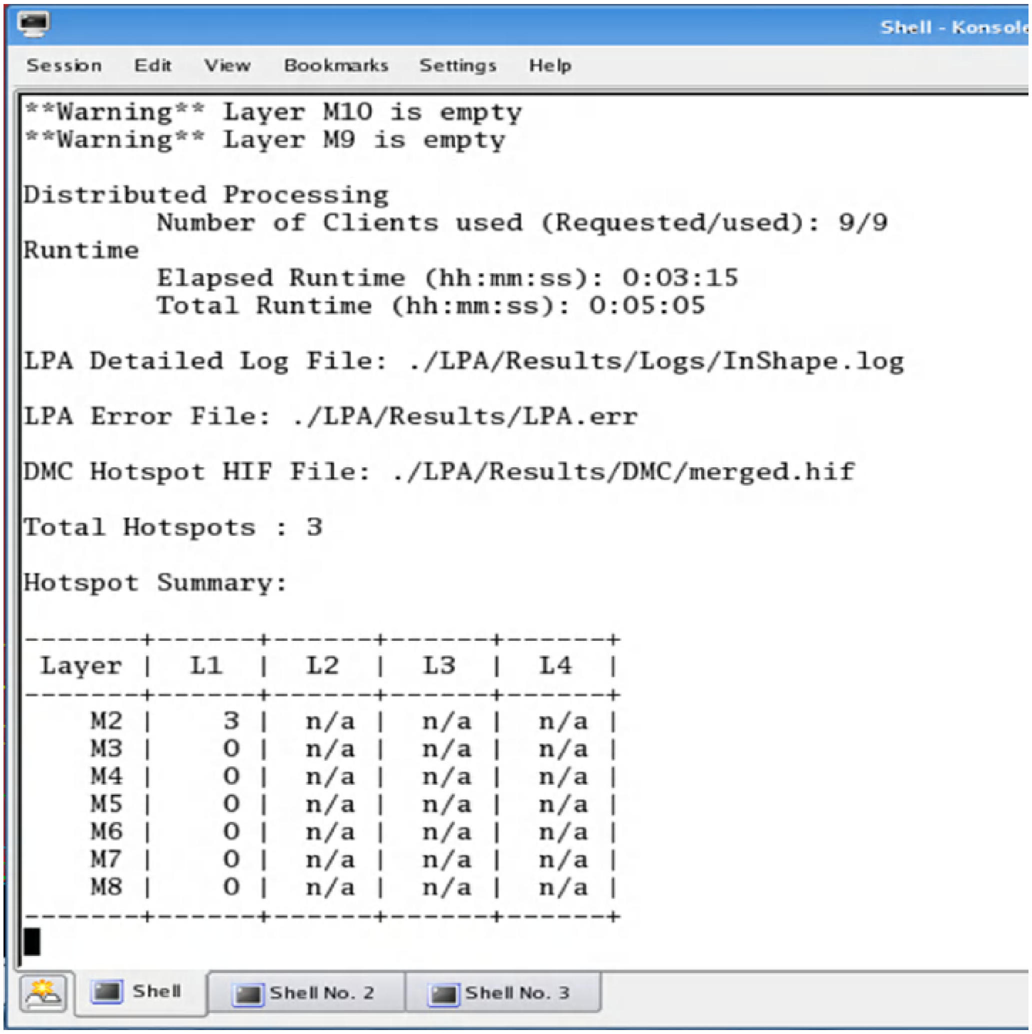
Host Name:

11. Specify the queue and resource string that is needed for the LSF launch in the *Queue* and *Resource* fields.

12. Select the *OK* button to confirm the multiple CPU settings and close the form.

13. Select the *Submit* button in the *Litho Verify* form to launch the LPA run with the specified settings.
    Note: In Routing Layers Only mode, LPA runs in blocking mode. You cannot perform any operations in the Innovus GUI until the LPA run is completed.

During the LPA run, the output is sent to the Innovus *Shell* window. After LPA is successfully completed, the LPA summary file with hotspot count is presented in the Innovus *Shell* window.



```
**Warning** Layer M10 is empty
**Warning** Layer M9 is empty

Distributed Processing
        Number of Clients used (Requested/used): 9/9
Runtime
        Elapsed Runtime (hh:mm:ss): 0:03:15
        Total Runtime (hh:mm:ss): 0:05:05

LPA Detailed Log File: ./LPA/Results/Logs/InShape.log

LPA Error File: ./LPA/Results/LPA.err

DMC Hotspot HIF File: ./LPA/Results/DMC/merged.hif

Total Hotspots : 3

Hotspot Summary:

-------+------+------+------+------+
 Layer |  L1  |  L2  |  L3  |  L4  |
-------+------+------+------+------+
    M2 |    3 |  n/a |  n/a |  n/a |
    M3 |    0 |  n/a |  n/a |  n/a |
    M4 |    0 |  n/a |  n/a |  n/a |
    M5 |    0 |  n/a |  n/a |  n/a |
    M6 |    0 |  n/a |  n/a |  n/a |
    M7 |    0 |  n/a |  n/a |  n/a |
    M8 |    0 |  n/a |  n/a |  n/a |
-------+------+------+------+------+
```

Running LPA in Routing Layers Only Mode from the Command Line

You can also run LPA in the Routing Layers Only mode from the Innovus command line by using the `verifyLitho -routingLayersOnly` option. You must have the LPA techfile and the streamMap file to run LPA in this mode. The `streamMap` file is same as the one used by the Innovus `streamOut` command.

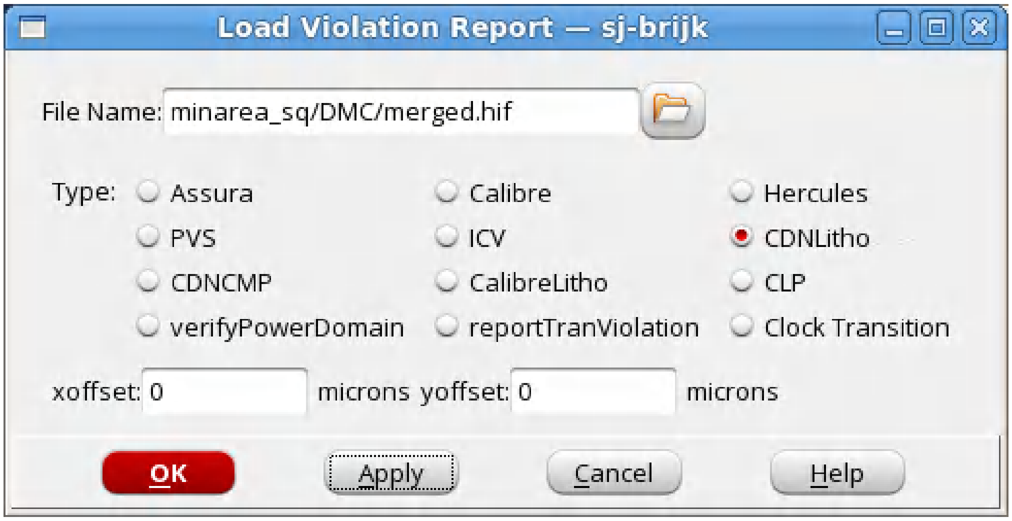
To run run LPA in the Routing Layers Only mode from the command line, launch Innovus by using the `innovus` command and load the design. When invoked, Innovus automatically loads all the required LPA files from the LPA installation path. Once the `innovus>` prompt is displayed, run the following command at the prompt:

```
innovus> verifyLitho -routingLayersOnly -techFile LPATechFile_dir -dCUIirdirection
./LPA/Results
```

Viewing Hotspots

You can load the HIF file (created in the output directory by the LPA run) in the *Violation Browser* directly to view the detected litho hotspots. To do this, perform the following set of steps:

1. Select *Tools -> Violation Browser -> Load Violation Report*.
2. This opens the *Load Violation Report* form. Specify the path and name of the HIF file in the *File Name* field.

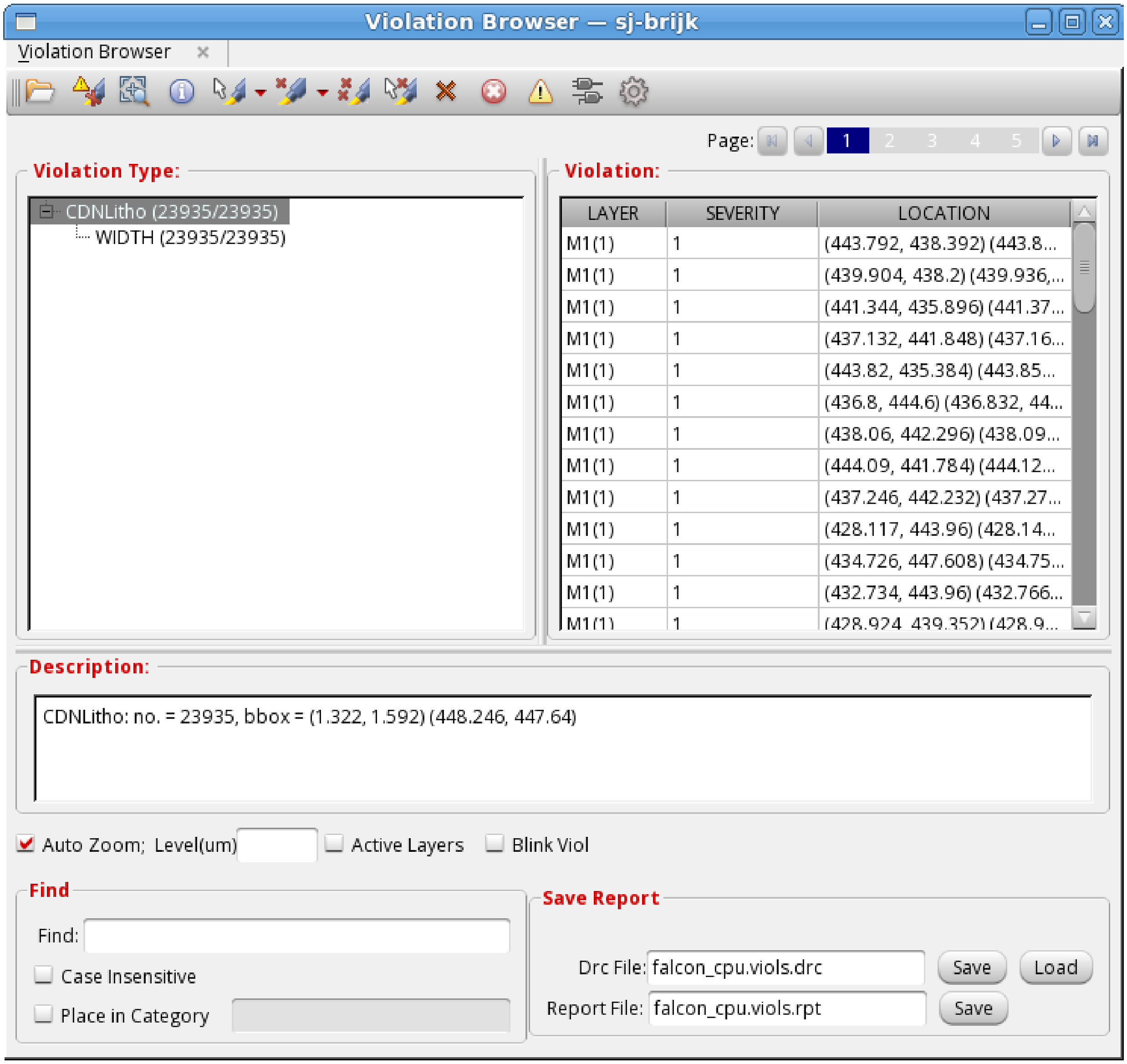


File Name: minarea_sq/DMC/merged.hif

Type: ○ Assura ○ Calibre ○ Hercules
○ PVS ○ ICV ● CDNLitho
○ CDNCMP ○ CalibreLitho ○ CLP
○ verifyPowerDomain ○ reportTranViolation ○ Clock Transition

xoffset: 0 microns yoffset: 0 microns

3. Select the *CDNLitho* radio button to specify the HIF format and select the *OK* button to view the hotspots in the *Violation Browser*.



Violation Browser — sj-brijk

Violation Browser

Page: 1 2 3 4 5

Violation Type:

- CDNLitho (23935/23935)
  - WIDTH (23935/23935)

Violation:

LAYER	SEVERITY	LOCATION
M1(1)	1	(443.792, 438.392) (443.8...
M1(1)	1	(439.904, 438.2) (439.936,...
M1(1)	1	(441.344, 435.896) (441.37...
M1(1)	1	(437.132, 441.848) (437.16...
M1(1)	1	(443.82, 435.384) (443.85...
M1(1)	1	(436.8, 444.6) (436.832, 44...
M1(1)	1	(438.06, 442.296) (438.09...
M1(1)	1	(444.09, 441.784) (444.12...
M1(1)	1	(437.246, 442.232) (437.27...
M1(1)	1	(428.117, 443.96) (428.14...
M1(1)	1	(434.726, 447.608) (434.75...
M1(1)	1	(432.734, 443.96) (432.766...
M1(1)	1	(428.924, 439.352) (428.9...

Description:

CDNLitho: no. = 23935, bbox = (1.322, 1.592) (448.246, 447.64)

☑ Auto Zoom; Level(um) ☐ Active Layers ☐ Blink Viol

Find

Find:

☐ Case Insensitive

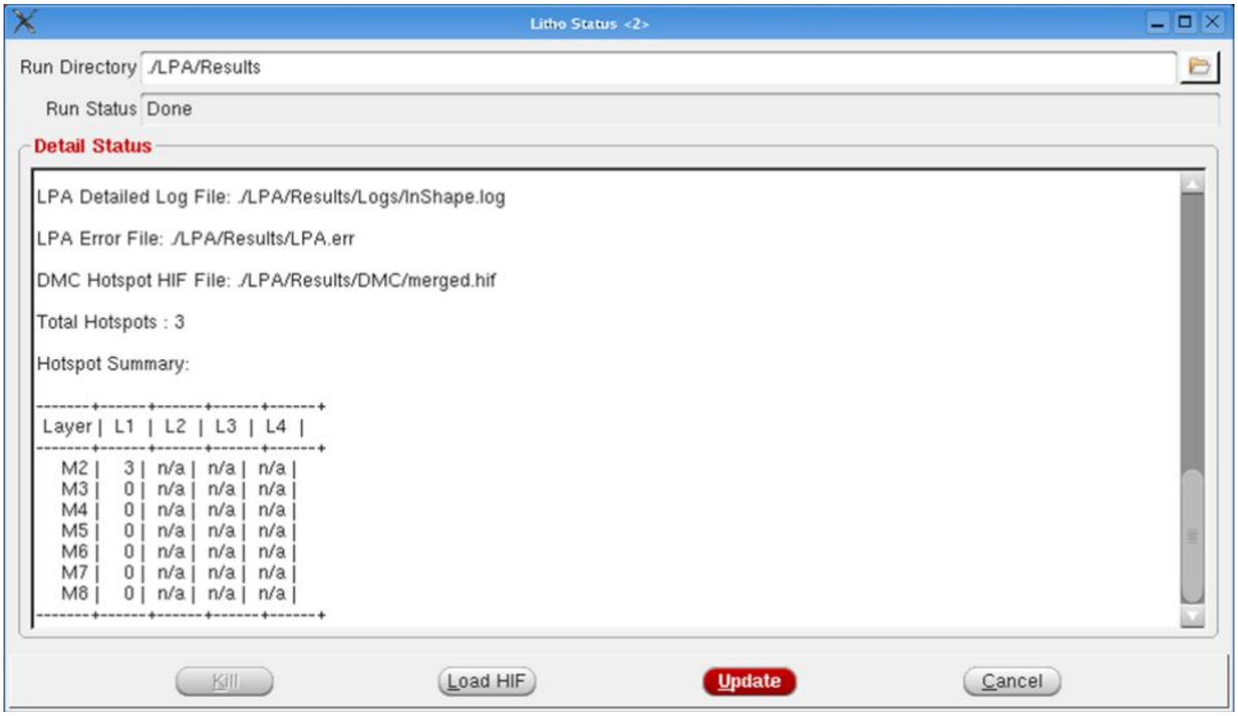
☐ Place in Category

Save Report

Drc File: falcon_cpu.viols.drc

Report File: falcon_cpu.viols.rpt

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.



Run Directory ./LPA/Results

Run Status Done

Detail Status

```
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 |
-------+------+------+------+------+
```

Note: You can also specify a different run directory name in the *Run Directory* field of the *Litho Status* window and then click the *Update* button to check the output of the specified LPA run.

Fixing Hotspots

Next, you fix the hotspots identified by the LPA run by using Nanoroute. There can be several ways to fix hotspots depending on the design technology. Please ask your Cadence representative for technology specific applications notes, which may provide advanced methodologies. To use the standard fixing approach, perform the following set of steps:

1. At the Innovus command line, type the following to set up Nanoroute and run it on the design to repair the litho hotspots.

   ```
   innovus> setNanoRouteMode -droutePostRouteLithoRepair true
   ```

   ```
   innovus> globalDetailRoute
   ```

2. After Innovus has competed `globalDetailRoute`, save the design for verification. This will also save the markers within the design, indicating the areas of change.

   ```
   innovus> saveDesign Litho_Fixed.enc
   ```

3. The Innovus router marks the areas of change and you can run Litho only on these areas of change rather than on the complete design, thereby saving time. Write out the incremental HIF

file that will be used to run Litho only on the changed areas.

```
innovus> writeHif -fCUIile IncrVerify.hif
```

4. Now, run LPA in the incremental area to check for hotspots after Innovus has fixed the litho hotspots. Open the *Litho Verify* form, with the *Routing Layers* tab selected by default. Specify the name of the incremental HIF file in the *Incr. HIF* field and click *Submit* to run LPA only on the changed areas.

If LPA reports no more litho hotspots in this run, the verification task is complete. However, if there are litho hotspots reported, repeat the steps to load the HIF file (created in the output directory by the latest LPA run) in the *Violation Browser*, and fix the hotspots using Nanoroute. The design is marked as verified when the LPA run on the incremental HIF file reports zero litho hotspots.

Selecting and Excluding Nets

While fixing hotspots, you can choose to exclude certain or areas in the design. `verifyLitho` allows you to specify many options for selecting or excluding areas within a design by area, layer identification, or cell. These options can be added to a configuration file which can be passed to `verifyLitho` by file with the `-config` *`config_file_name`* option. For information about these options, refer to the *Litho Physical Analyzer User Guide* in the MVS distribution.

Besides the native `verifyLitho` options for selection and exclusion, you can use Innovus commands to specify which should be affected by routing. To prevent a net from being moved during routing, simply specify that the net should not be touched during routing by using the following command:

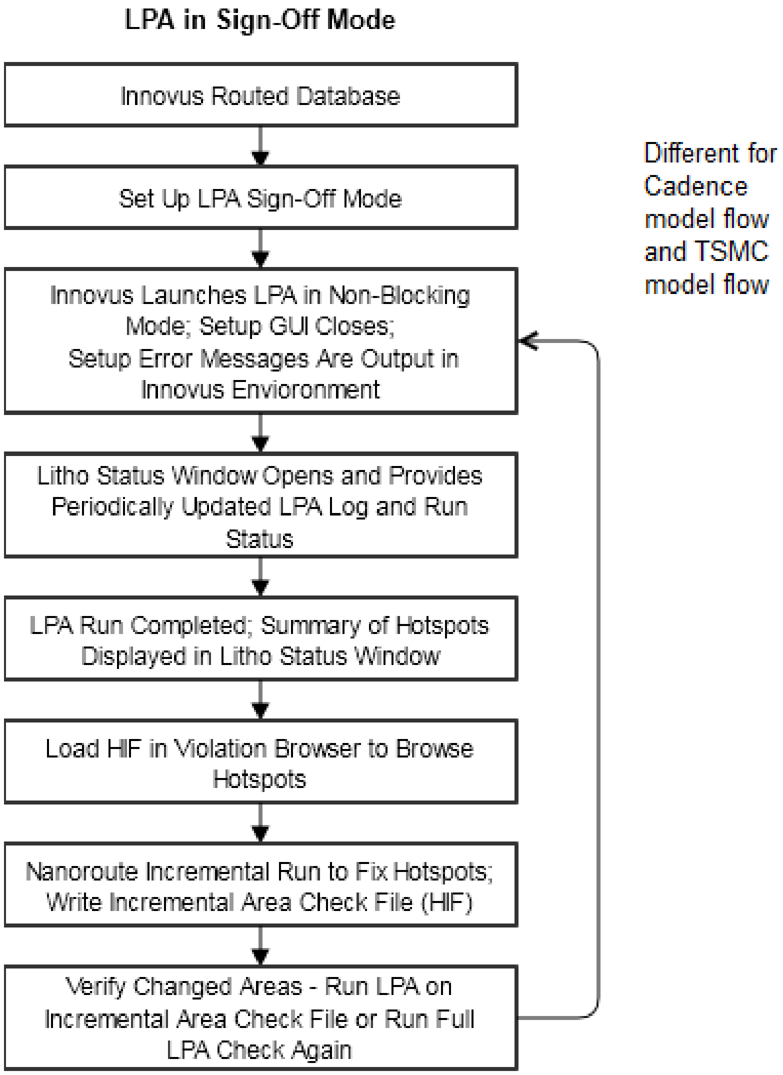
```
setAttribute -nCUIet net_name -skip_routing true
```

This will prevent the net from being modified during detail route. Be sure to turn this attribute to false after litho fixing is complete.

Sign-Off Mode

You run LPA in Sign-Off mode for Litho sign-off, as mandated by foundries. Unlike Routing Layers Only mode where no user input is required, Sign-Off mode requires you to provide the LPA configuration file containing the Techfile and other settings. You must run LPA Sign-Off mode before handing off the design to the top-level or tape-out.

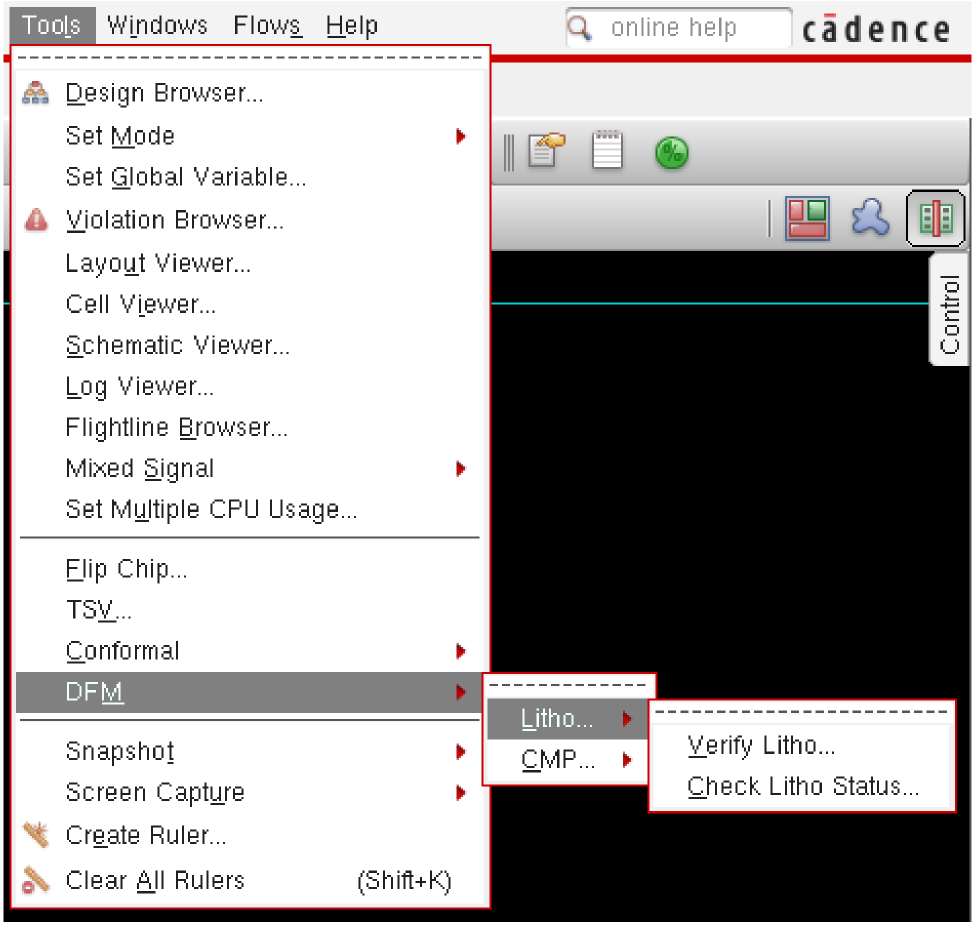
The following diagram shows the design flow for running LPA in Sign-Off mode.


LPA in Sign-Off Mode
Innovus Routed Database
Set Up LPA Sign-Off Mode
Innovus Launches LPA in Non-Blocking Mode; Setup GUI Closes; Setup Error Messages Are Output in Innovus Envioronment
Litho Status Window Opens and Provides Periodically Updated LPA Log and Run Status
LPA Run Completed; Summary of Hotspots Displayed in Litho Status Window
Load HIF in Violation Browser to Browse Hotspots
Nanoroute Incremental Run to Fix Hotspots; Write Incremental Area Check File (HIF)
Verify Changed Areas - Run LPA on Incremental Area Check File or Run Full LPA Check Again
Different for Cadence model flow and TSMC model flow

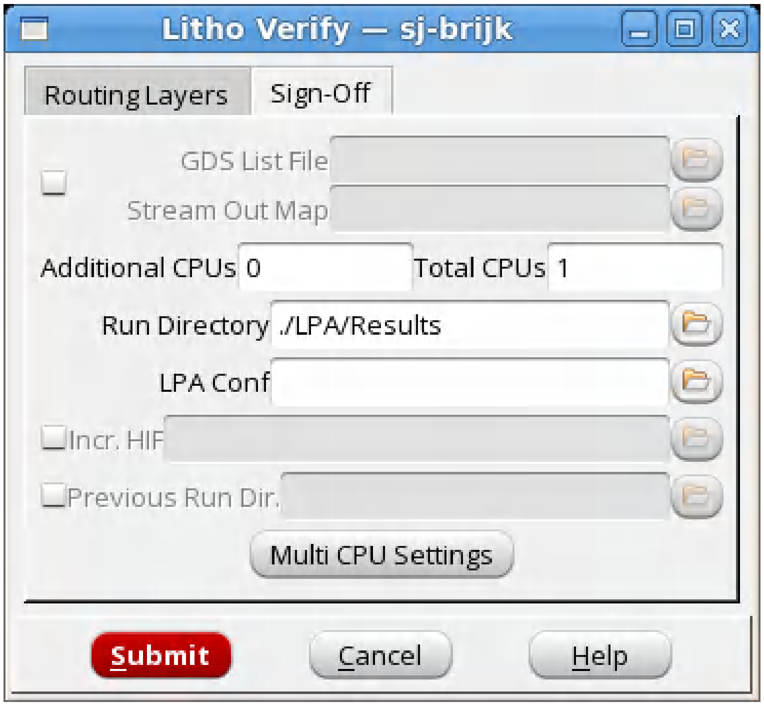
Running LPA in the Sign-Off Mode from the GUI

To run LPA in Sign-Off mode, launch Innovus by using the `innovus` command and load the design. When Innovus is invoked, it automatically loads all the required LPA files from the LPA installation path. Once the Innovus GUI is displayed, perform the following steps:

1. Choose *Tools -> DFM -> Litho -> Verify Litho* from the Innovus menu.



2. This opens the *Litho Verify* form, with the *Routing Layers* tab selected by default. Select the *Sign-Off* tab.



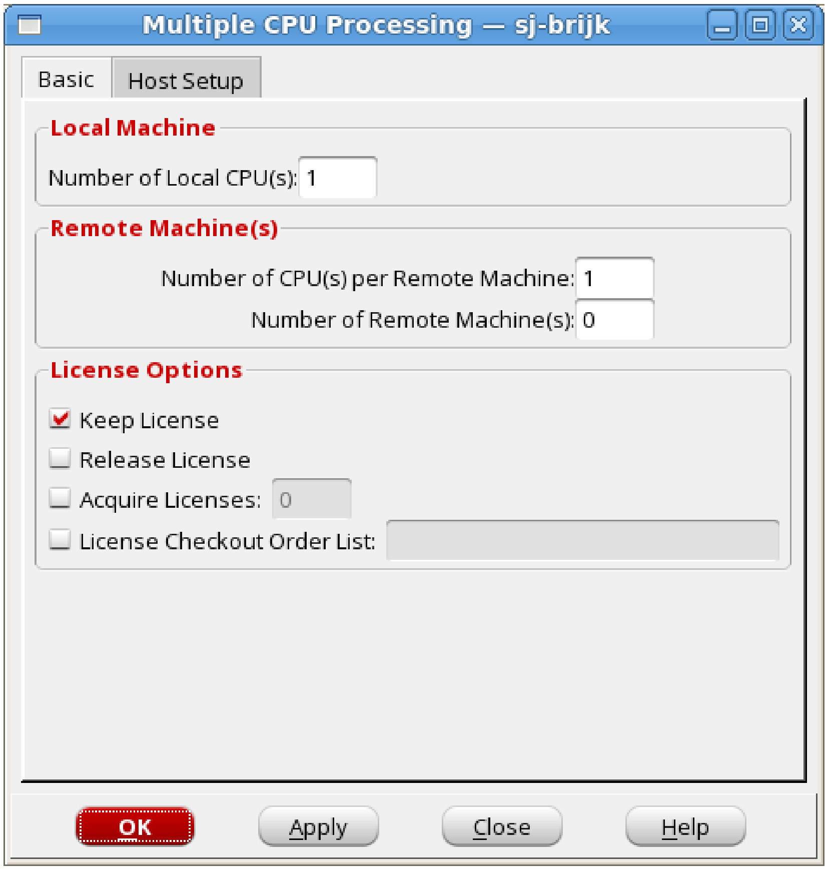
□ GDS List File
Stream Out Map
Additional CPUs 0
Total CPUs 1
Run Directory ./LPA/Results
LPA Conf
□ Incr. HIF
□ Previous Run Dir.

3. Specify the path and name of the LPA output directory in the *Run Directory* field. All output data from the LPA run will be stored under this directory.
4. Specify the name and path of the LPA configuration file in the *LPA Conf* field. This configuration file should contain the Techfile location for all layers and any additional LPA commands that you want to execute during the LPA run.
   Alternatively, if you want to enable the GLOBALFOUNDRIES DRC+ flow, specify the name and path of the configuration file that points to the foundry-supplied DRC+ Pattern technology file.
5. In the *Additional CPUs* field, specify the number of additional CPUs you want to use for the current sign-off LPA run. This number is in addition to the total number of CPUs specified in the *Total CPUs* field. A higher number of additional CPUs results in decreased run time.
6. Optionally, you can specify the name and path of the GDS list file and Stream Out Map file in the *GDS List File* and *Stream Out Map* fields, if you want to run Poly, Diffusion, or Metal1. The GDS list file is a text file containing the list of GDS files for LEF abstracts. The Stream Out Map file is created by Innovus to map the GDS stream to the layers in the Innovus database. By default, LPA runs on the interconnect layers in Sign-Off mode but if the GDS List file and Stream Out Map file are specified, LPA runs on the potential IP blocks defined in these files.
7. If you have already run LPA in Sign-Off mode once and are running it again to check if all detected hotspots have been fixed, specify the name and path of the HIF file that you want to

use for incremental validation in the *Incr. HIF* field. This HIF file includes the locations that identify the areas affected by each hotspot fix. LPA reads these locations and performs incremental checking only in these areas. This reduces the time for validation.

8. If you have already run LPA in Sign-Off mode once and are running it again, but there is no HIF file that identifies the changed areas, you use the *Previous Run Dir* field to specify the path to the run directory of the previous Verify Litho run for XOR-based incremental validation. When this option is selected, the previous run results are compared to the new design and incremental checking is performed only in locations where the layout has changed and where hotspots previously existed, thereby reducing the overall validation run time.

9. By default, LPA uses the LSF settings from the *Multi-CPU Settings* GUI in Innovus. However, you can change the multi-CPU settings for the LPA run by selecting the *Multi CPU Settings* button and specifying the new settings in the *Multiple CPU Processing* form. Note that this will also change the distributed options for all Innovus commands.



Multiple CPU Processing — sj-brijk

Local Machine

Number of Local CPU(s): 1

Remote Machine(s)

Number of CPU(s) per Remote Machine: 1

Number of Remote Machine(s): 0

License Options

☑ Keep License

□ Release License

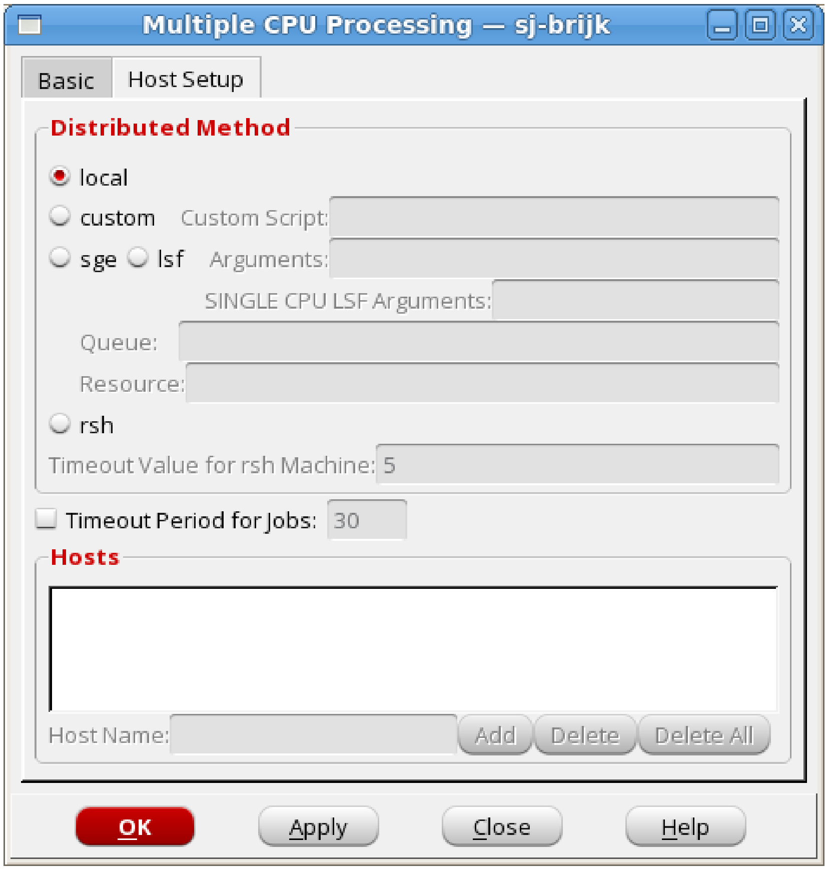
□ Acquire Licenses: 0

□ License Checkout Order List:

10. On the Basic tab of the *Multiple CPU Processing* form, if running locally, use the *Number of Local CPU(s)* field to set the number of local CPUs used. If using LSF to run, the *Number of Remote Machine(s)* field specifies the number of LSF machines that you want to use for the

LPA run.

11. On the *Host Setup* tab, select the the *lsf* radio button to set the distribution method as LSF and specify the LSF arguments in the *LSF Arguments* field or select the *local* radio button to use local CPUs only.
    Note: It is recommended to use local or LSF distribution method for the Innovus-LPA integration.



● local

Timeout Value for rsh Machine: 5

□ Timeout Period for Jobs: 30

12. Specify the queue and resource string that is needed for the LSF launch in the *Queue* and *Resource* fields.

13. Select the *OK* button to confirm the multiple CPU settings and close the form.

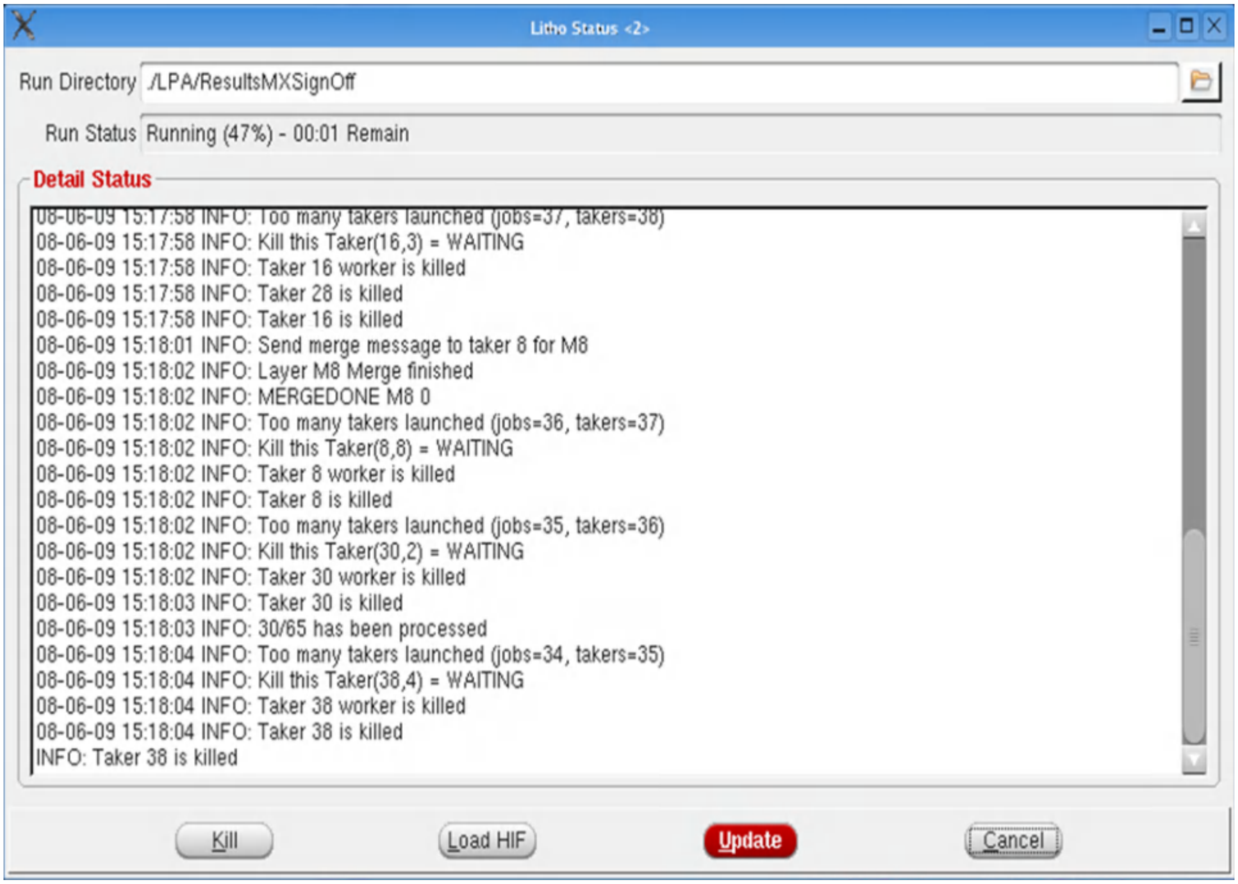
14. Select the *Submit* button in the *Litho Verify* form to launch the sign-off LPA run with the specified settings.
    Note: In Sign-Off mode, LPA runs in non-blocking mode. You can continue working with the Innovus GUI and perform design activities in parallel with Litho sign-off. If you exit Innovus during this period, a warning is displayed informing you that LPA is still running and you can load the results later.

15. On the successful launch of LPA, the *Litho Verify* form closes and the *Litho Status* window is

automatically displayed. The *Run Directory* field of this window is automatically populated from the *Litho Verify* form.

The *Litho Status* window provides updated information about the status of the LPA run.

The *Run Status* field provides information on the completion percentage and approximate remaining time of the LPA run. The run status is periodically updated in the *Detail Status* area. You can click the *Update* button to manually update the status. To terminate the current LPA run, click the *Kill* button.

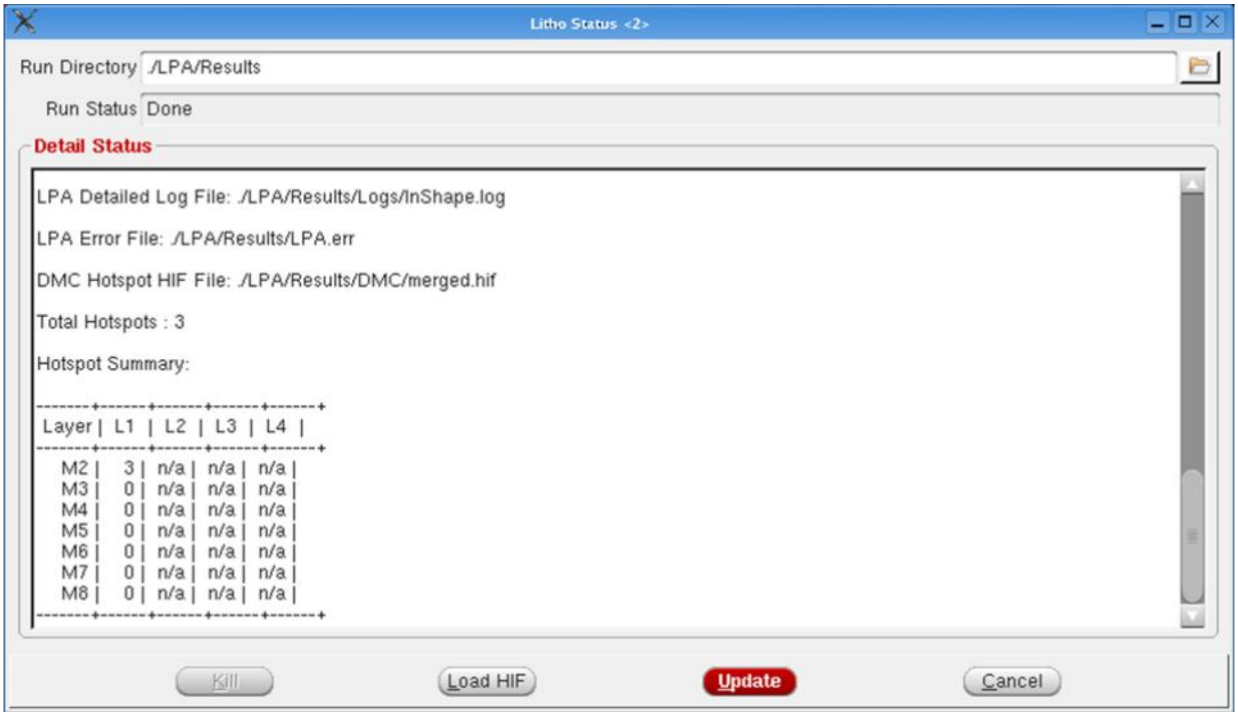


Run Directory ./LPA/ResultsMXSignOff

Run Status Running (47%) - 00:01 Remain

```
08-06-09 15:17:58 INFO: Too many takers launched (jobs=37, takers=38)
08-06-09 15:17:58 INFO: Kill this Taker(16,3) = WAITING
08-06-09 15:17:58 INFO: Taker 16 worker is killed
08-06-09 15:17:58 INFO: Taker 28 is killed
08-06-09 15:17:58 INFO: Taker 16 is killed
08-06-09 15:18:01 INFO: Send merge message to taker 8 for M8
08-06-09 15:18:02 INFO: Layer M8 Merge finished
08-06-09 15:18:02 INFO: MERGEDONE M8 0
08-06-09 15:18:02 INFO: Too many takers launched (jobs=36, takers=37)
08-06-09 15:18:02 INFO: Kill this Taker(8,8) = WAITING
08-06-09 15:18:02 INFO: Taker 8 worker is killed
08-06-09 15:18:02 INFO: Taker 8 is killed
08-06-09 15:18:02 INFO: Too many takers launched (jobs=35, takers=36)
08-06-09 15:18:02 INFO: Kill this Taker(30,2) = WAITING
08-06-09 15:18:02 INFO: Taker 30 worker is killed
08-06-09 15:18:03 INFO: Taker 30 is killed
08-06-09 15:18:03 INFO: 30/65 has been processed
08-06-09 15:18:04 INFO: Too many takers launched (jobs=34, takers=35)
08-06-09 15:18:04 INFO: Kill this Taker(38,4) = WAITING
08-06-09 15:18:04 INFO: Taker 38 worker is killed
08-06-09 15:18:04 INFO: Taker 38 is killed
INFO: Taker 38 is killed
```

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*.



Run Directory ./LPA/Results

Run Status Done

Detail Status

```
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 |
-------+------+------+------+------+
```

Note: You can also specify a different run directory name in the *Run Directory* field of the *Litho Status* window and then click the *Update* button to check the output of the specified LPA run.

Running LPA in the Sign-Off Mode from the Command Line

You can also run LPA in the Sign-Off mode from the Innovus command line by using the `verifyLitho -signOff` option. You must have the LPA techfile and the GDS to LPA layer map file to run LPA in this mode. By default, LPA runs on the interconnect layers in sign-off mode but if the GDS List file and Stream Out Map file are specified, LPA runs on the potential IP blocks defined in these files.

To run run LPA in the sign-off mode from the command line, launch Innovus by using the `innovus` command and load the design. When invoked, Innovus automatically loads all the required LPA files from the LPA installation path. Once the `innovus` prompt is displayed, run the following command:

```
verifyLitho -signOff -dCUIirdirection ./LPA/Results -cpu 1 -config lpa.conf -gdsList
<leaf/*.gds> -mapFile innovusGds.map
```

The GDS to LPA layer map file should have path of the lpa configuration file (lpa.conf). Its syntax is as follows:

```
#userDefined name GDSlayer FAB_techFile_layer_name

M1 31:0 M1
```

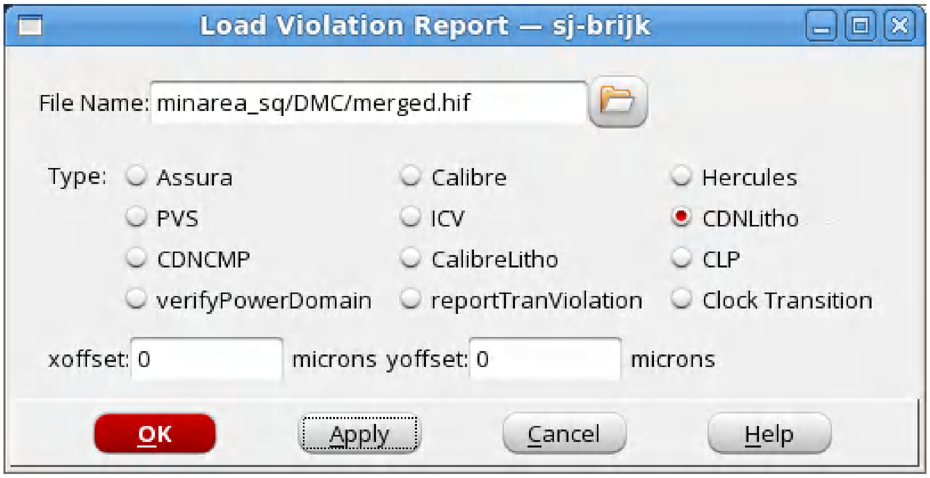
```
M2 32:0 M2
```

For more details, refer to the *Litho Physical Analyzer User Guide*.

Viewing Hotspots

You can load the HIF file (created in the output directory by the LPA run) in the *Violation Browser* directly to view the detected hotspots. To do this, you perform the following set of steps:

1. Select *Tools -> Violation Browser -> Load Violation Report*.
2. This opens the *Load Violation Report* form. Specify the path and name of the HIF file in the *File Name* field.



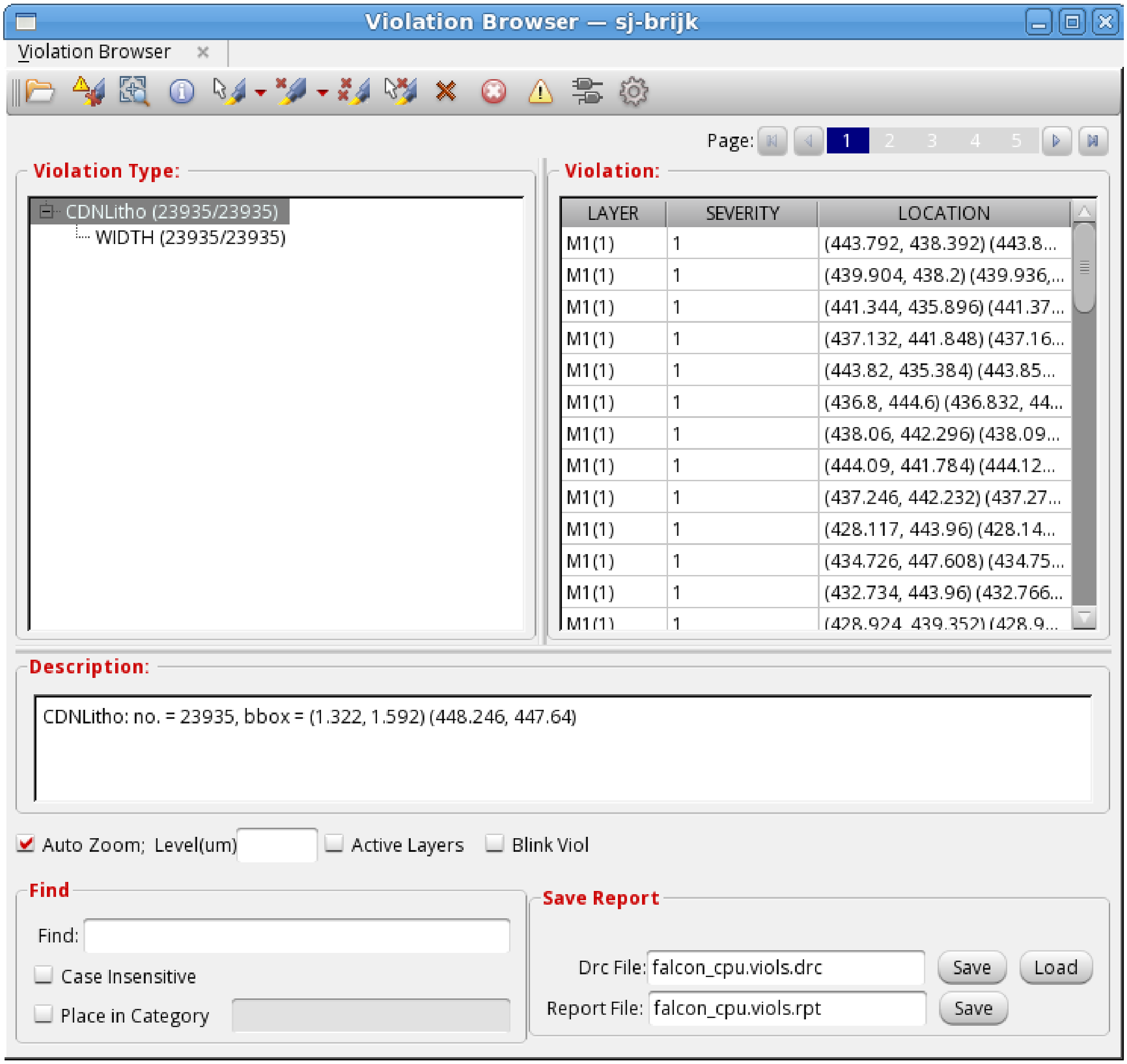
Load Violation Report — sj-brijk

File Name: minarea_sq/DMC/merged.hif

Type: ○ Assura ○ Calibre ○ Hercules
○ PVS ○ ICV ● CDNLitho
○ CDNCMP ○ CalibreLitho ○ CLP
○ verifyPowerDomain ○ reportTranViolation ○ Clock Transition

xoffset: 0 microns yoffset: 0 microns

3. Select the *CDNLitho* radio button to specify the HIF format and select the *OK* button to view the hotspots in the *Violation Browser*.



CDNLitho (23935/23935)
WIDTH (23935/23935)

LAYER	SEVERITY	LOCATION
M1(1)	1	(443.792, 438.392) (443.8...
M1(1)	1	(439.904, 438.2) (439.936,...
M1(1)	1	(441.344, 435.896) (441.37...
M1(1)	1	(437.132, 441.848) (437.16...
M1(1)	1	(443.82, 435.384) (443.85...
M1(1)	1	(436.8, 444.6) (436.832, 44...
M1(1)	1	(438.06, 442.296) (438.09...
M1(1)	1	(444.09, 441.784) (444.12...
M1(1)	1	(437.246, 442.232) (437.27...
M1(1)	1	(428.117, 443.96) (428.14...
M1(1)	1	(434.726, 447.608) (434.75...
M1(1)	1	(432.734, 443.96) (432.766...
M1(1)	1	(428.924, 439.352) (428.9...

CDNLitho: no. = 23935, bbox = (1.322, 1.592) (448.246, 447.64)

☑ Auto Zoom; Level(um) ☐ Active Layers ☐ Blink Viol

Find:

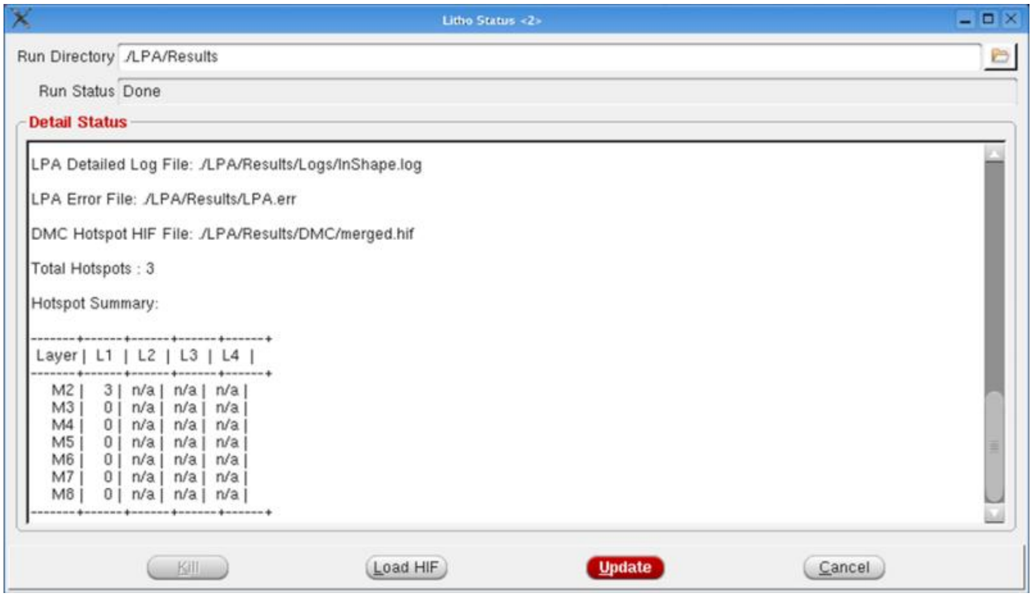
☐ Case Insensitive

☐ Place in Category

Drc File: falcon_cpu.viols.drc

Report File: falcon_cpu.viols.rpt

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.



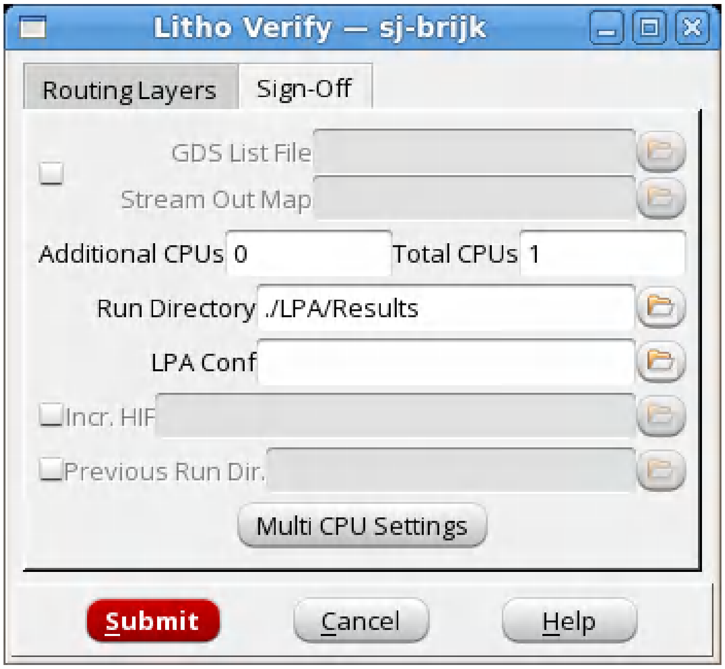
Run Directory ./LPA/Results

Run Status Done

```
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 |
-------+-------+-------+-------+-------+
```

Fixing Hotspots

Next, you fix the hotspots identified by the LPA run by using Nanoroute. The Innovus router marks the areas of change in the design, which you can use to write out the incremental HIF file. The steps to perform this task are similar to the steps performed in the Routing Layers Only mode (see Fixing Hotspots). Once you create the incremental HIF file, use it to run LPA only on the changed areas rather than on the complete design, thereby saving time. Here, you run LPA from the *Sign-Off* tab by specifying the name of the incremental HIF file in the *Incr. HIF* field.



☐ GDS List File
Stream Out Map

Additional CPUs 0 Total CPUs 1

Run Directory ./LPA/Results

LPA Conf

☐ Incr. HIF

☐ Previous Run Dir.

Before You Begin Running CCP

Before you can run CCP from Innovus, the following conditions must be met:

- You must have the CCP license to run CCP from Innovus.
- You must have either the Encounter Advanced Node GXL Option or Innovus DFM Option license.
- You must be able to launch version 9.2 (or a later version) CCP from Innovus. In other words, the installation path to CCP must be present in your path variable.
- You must haveCCP TechFiles that are compatible with the design technology.

Running CCP from Innovus

The integration of CCP with Innovus allows you to check for the potential yield issues that are caused by variations in interconnect thickness for any design and any manufacturing process that has been calibrated. You can run CCP to identify L1 hotspots on full chip or on blocks larger than `1mm x 1mm`.

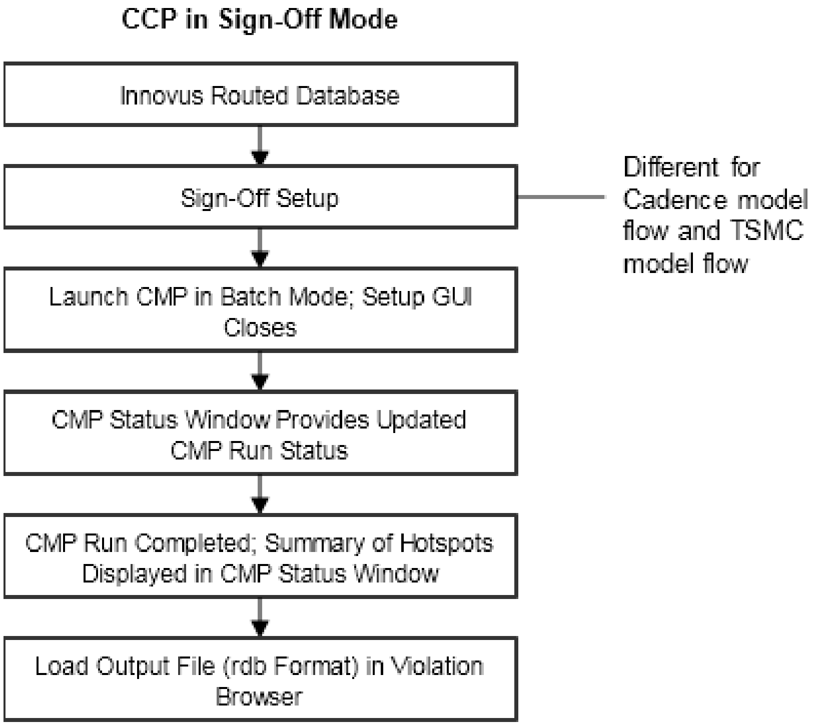
Following are the main advantages of running CCP from within Innovus.

- Supports manufacturability checks of routed blocks
- Supports fast check and sign-off verification
- Eliminates translation and setup hassles
- Runs with default out-of-the-box setup

You need to use the Cadence model flow to verify your design.

CCP Flow in Innovus

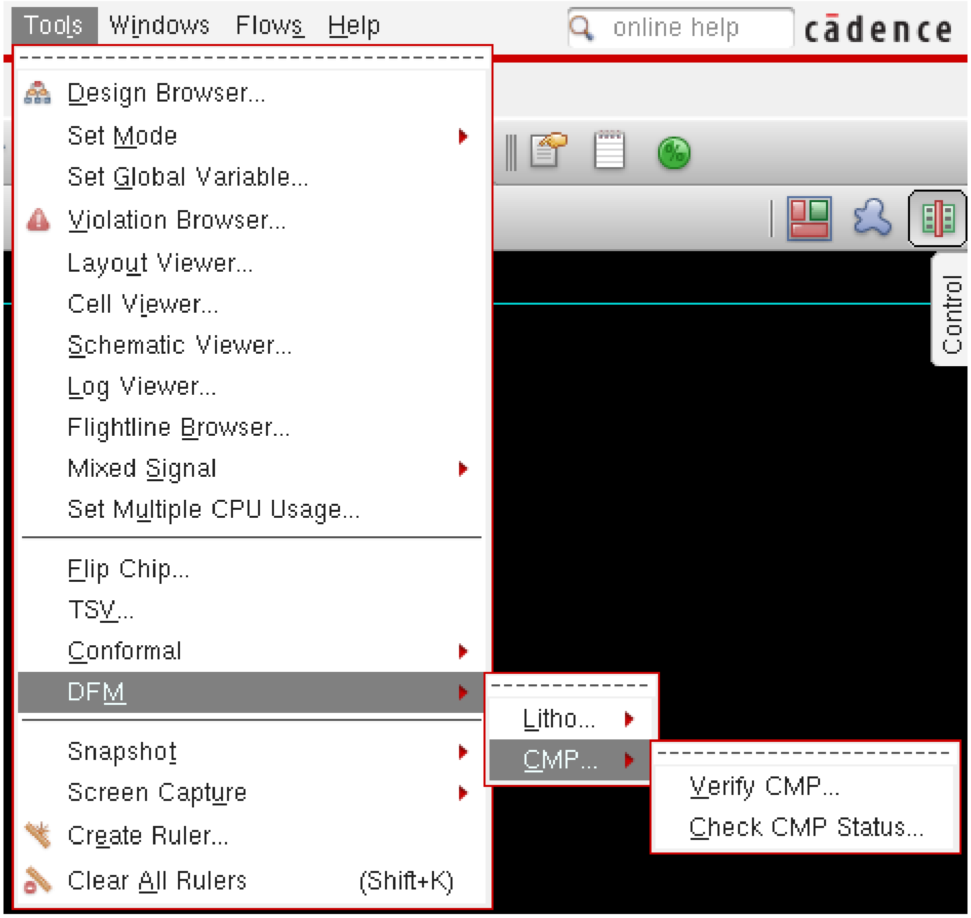
The following diagram shows the design flow for running CCP in Sign-Off mode.


CCP in Sign-Off Mode
Innovus Routed Database
Sign-Off Setup
Different for Cadence model flow and TSMC model flow
Launch CMP in Batch Mode; Setup GUI Closes
CMP Status Window Provides Updated CMP Run Status
CMP Run Completed; Summary of Hotspots Displayed in CMP Status Window
Load Output File (rdb Format) in Violation Browser

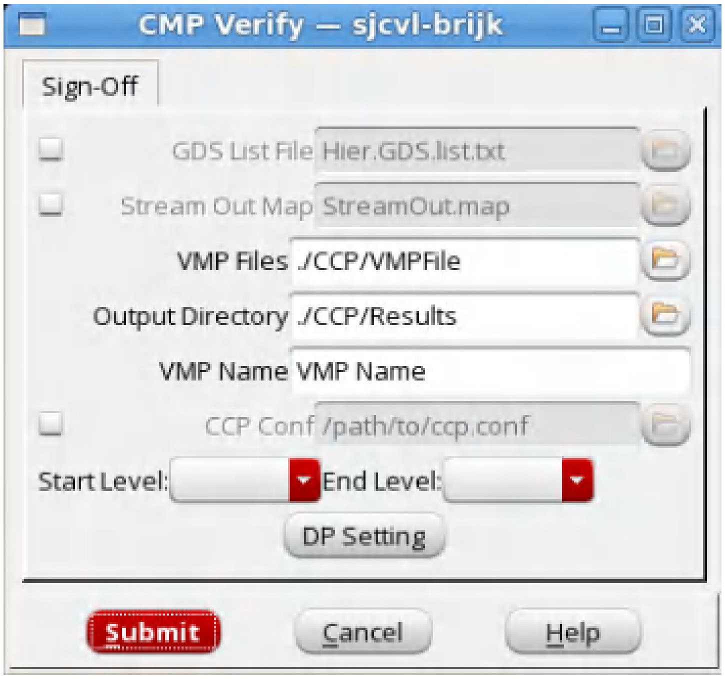
Running CCP in Cadence Model Flow

To run CCP analysis on your design using the Cadence model flow, launch Innovus by using the `innovus` command and load the design. When Innovus is invoked, it automatically loads all the required CCP files from the CCP installation path. Once the Innovus GUI is displayed, perform the following steps:

1. Choose *Tools -> DFM -> CMP -> Verify CMP* from the Innovus menu.



2. This opens the *CMP Verify* form, with the *Sign-Off* tab selected by default. In the *VMP Files* field, specify the name and path of the `vmp.xml` file to be used for the CMP run. These files can be downloaded from the foundries. These files are usually packaged in a DDK kit. The extraction and prediction results of the CMP analysis are based on the process specifications in your `vmp` file.



CMP Verify — sjcvl-brijk

Sign-Off

☐ GDS List File Hier.GDS.list.txt

☐ Stream Out Map StreamOut.map

VMP Files ./CCP/VMPFile

Output Directory ./CCP/Results

VMP Name VMP Name

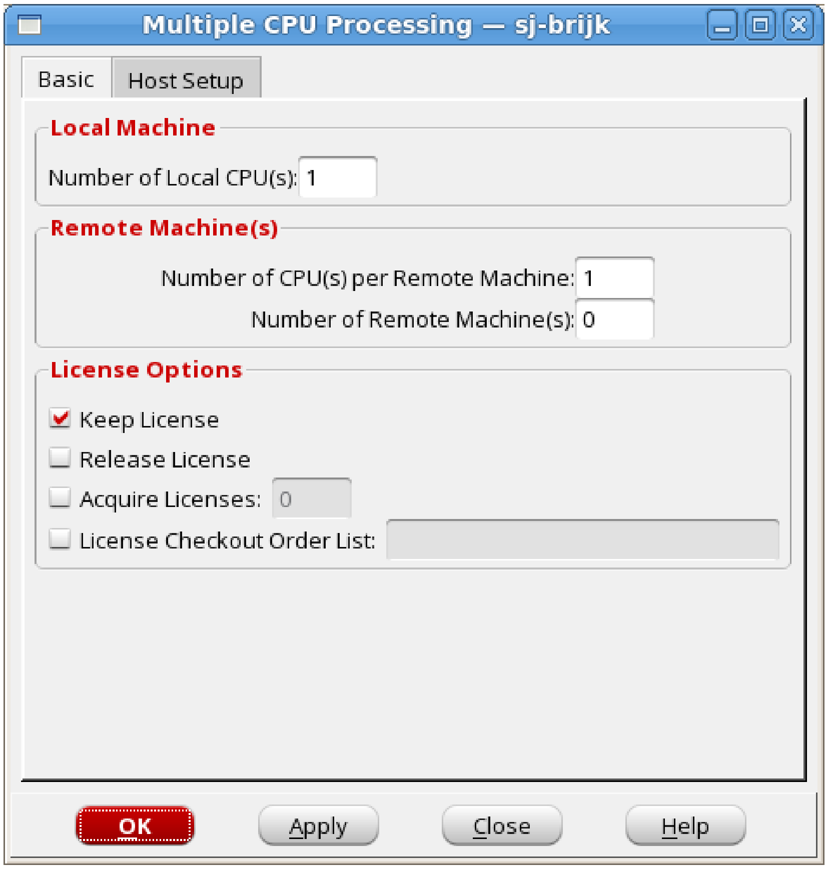
☐ CCP Conf /path/to/ccp.conf

Start Level: End Level:

3. In the *Output Directory* field, specify the CMP output directory that will contain the extraction and prediction results of the CMP run.
4. In the *VMP Name* field, specify the name of the metal scheme from the `vmp.xml` file.
5. Optionally, you can specify the name and path of the CMP configuration file in the *CCP Conf* field. This file, if specified, controls all run options of the CMP simulation.
6. The *Start Level* and *End Level* fields allow you to control the number of layers for prediction. These fields are optional. You can specify the starting metal level and end metal level in the *Start Level* and *End Level* fields for the CMP simulation. You can select them from the drop-down menus associated with these two fields.
7. *GDS List File* and *Stream Out Map* fields are also optional. You can specify the name and path of the GDS list file and Stream Out Map file in these fields. The GDS list file is a text file containing the list of GDS files for LEF abstracts. The Stream Out Map file is created by Innovus to map the GDS stream to the layers in the Innovus database.
   CMP runs on the interconnect layers by default, but if the GDS List file and Stream Out Map file are specified, CMP runs on the potential IP blocks defined in these files.
8. By default, CMP uses the multiple CPU settings from the *Multi-CPU Settings* GUI in Innovus. However, you can change the multi-CPU settings for the CMP run by selecting the *DP Settings* button and specifying the new settings in the *Multiple CPU Processing* form. For example, if you enter `8` in the *Number of Remote Machine(s)* field, eight jobs will be farmed

out to LSF.

Note: If you modify the multi-CPU settings for CMP analysis, this will also change the distributed options for all Innovus commands.



Local Machine

Number of Local CPU(s): 1

Remote Machine(s)

Number of CPU(s) per Remote Machine: 1

Number of Remote Machine(s): 0

License Options

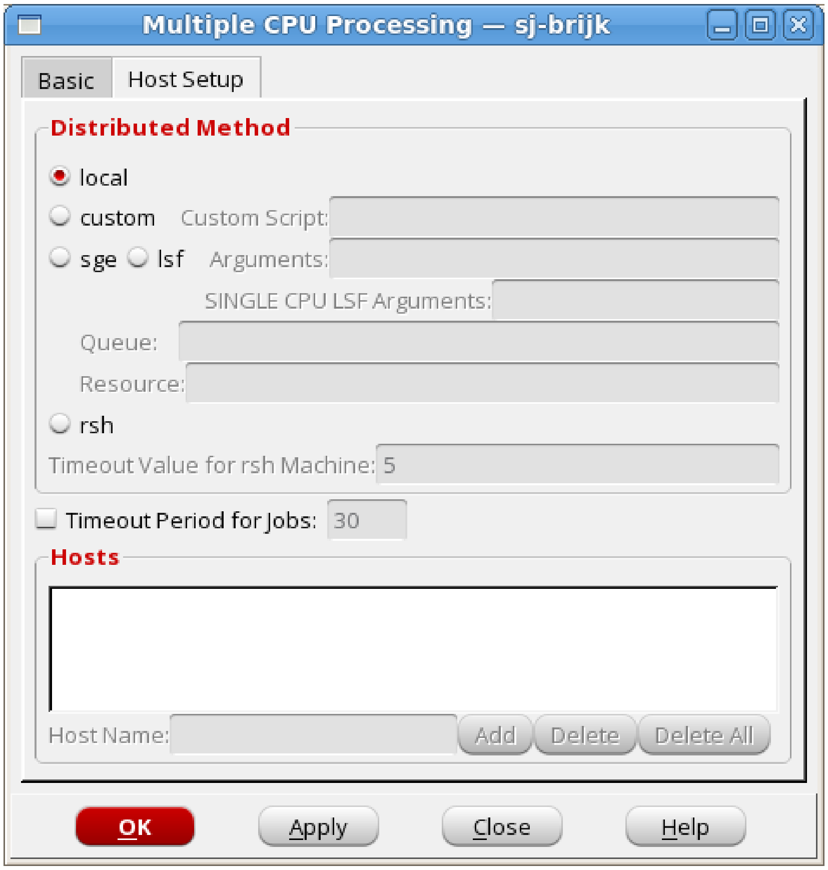
☑ Keep License

☐ Release License

☐ Acquire Licenses: 0

☐ License Checkout Order List:

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.



Distributed Method

● local

○ custom Custom Script:

○ sge ○ lsf Arguments:

SINGLE CPU LSF Arguments:

Queue:

Resource:

○ rsh

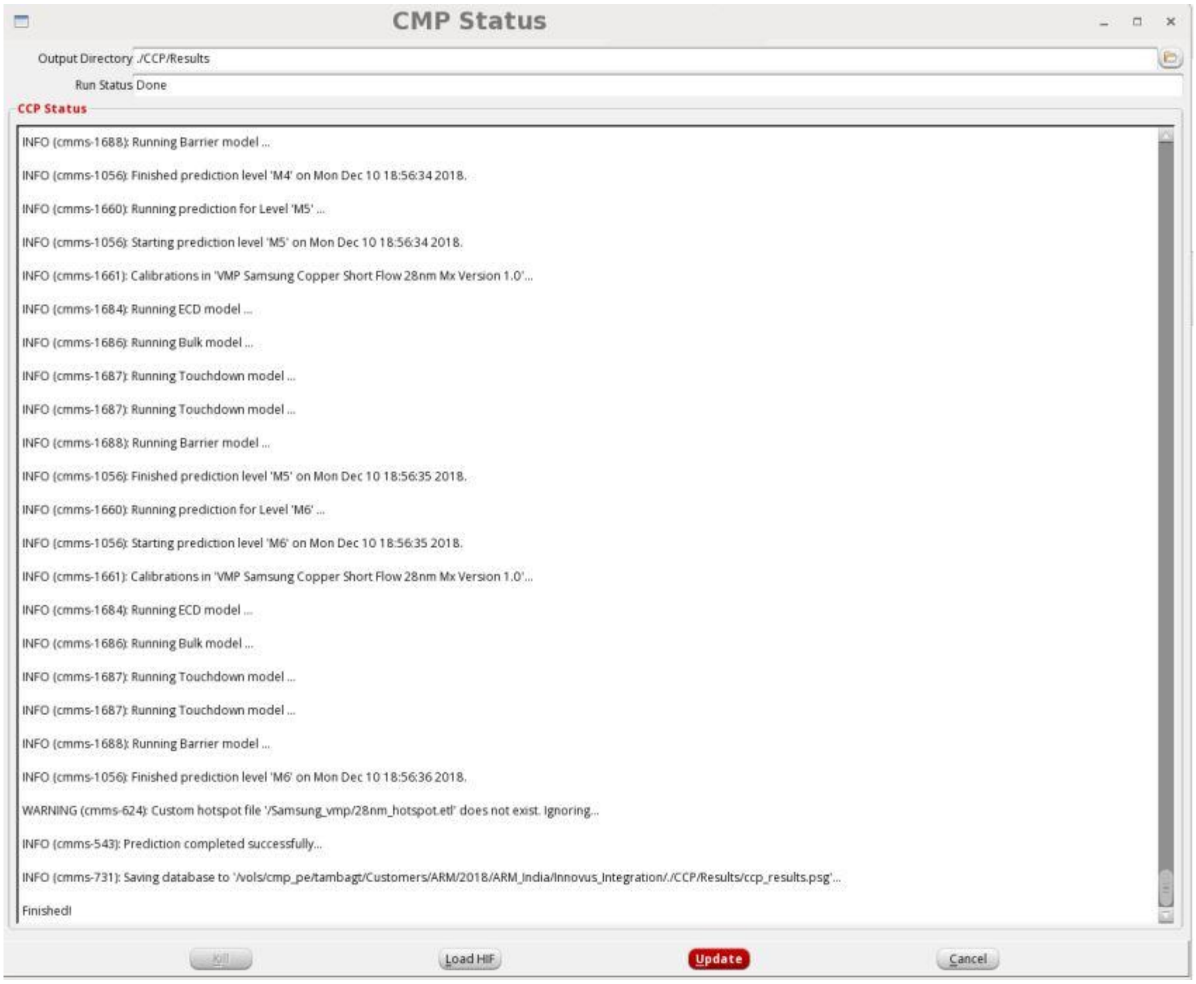
Timeout Value for rsh Machine: 5

□ Timeout Period for Jobs: 30

Hosts

Host Name:

10. Select the *OK* button to confirm the multiple CPU settings and close the form.
11. Select the *Submit* button in the *CMP Verify* form to launch the CMP run with the specified settings.
    Note: CMP runs in non-blocking mode. You can continue working with the Innovus GUI and perform design activities while the CMP simulation is running. If you exit Innovus during this period, a warning will be displayed informing you that CMP is still running and you can load the results later.
    On the successful launch of the CMP simulation, the *CMP Verify* form closes and the *CMP Status* window is automatically displayed. This window provides updated information about the status of the CMP run.



Output Directory ./CCP/Results

Run Status Done

CCP Status

```
INFO (cmms-1688): Running Barrier model ...
INFO (cmms-1056): Finished prediction level 'M4' on Mon Dec 10 18:56:34 2018.
INFO (cmms-1660): Running prediction for Level 'M5' ...
INFO (cmms-1056): Starting prediction level 'M5' on Mon Dec 10 18:56:34 2018.
INFO (cmms-1661): Calibrations in 'VMP Samsung Copper Short Flow 28nm Mx Version 1.0'...
INFO (cmms-1684): Running ECD model ...
INFO (cmms-1686): Running Bulk model ...
INFO (cmms-1687): Running Touchdown model ...
INFO (cmms-1687): Running Touchdown model ...
INFO (cmms-1688): Running Barrier model ...
INFO (cmms-1056): Finished prediction level 'M5' on Mon Dec 10 18:56:35 2018.
INFO (cmms-1660): Running prediction for Level 'M6' ...
INFO (cmms-1056): Starting prediction level 'M6' on Mon Dec 10 18:56:35 2018.
INFO (cmms-1661): Calibrations in 'VMP Samsung Copper Short Flow 28nm Mx Version 1.0'...
INFO (cmms-1684): Running ECD model ...
INFO (cmms-1686): Running Bulk model ...
INFO (cmms-1687): Running Touchdown model ...
INFO (cmms-1687): Running Touchdown model ...
INFO (cmms-1688): Running Barrier model ...
INFO (cmms-1056): Finished prediction level 'M6' on Mon Dec 10 18:56:36 2018.
WARNING (cmms-624): Custom hotspot file '/Samsung_vmp/28nm_hotspot.etl' does not exist. Ignoring...
INFO (cmms-543): Prediction completed successfully...
INFO (cmms-731): Saving database to '/vols/cmp_pe/tambagt/Customers/ARM/2018/ARM_India/Innovus_Integration/./CCP/Results/ccp_results.psg'...
Finished!
```

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*.

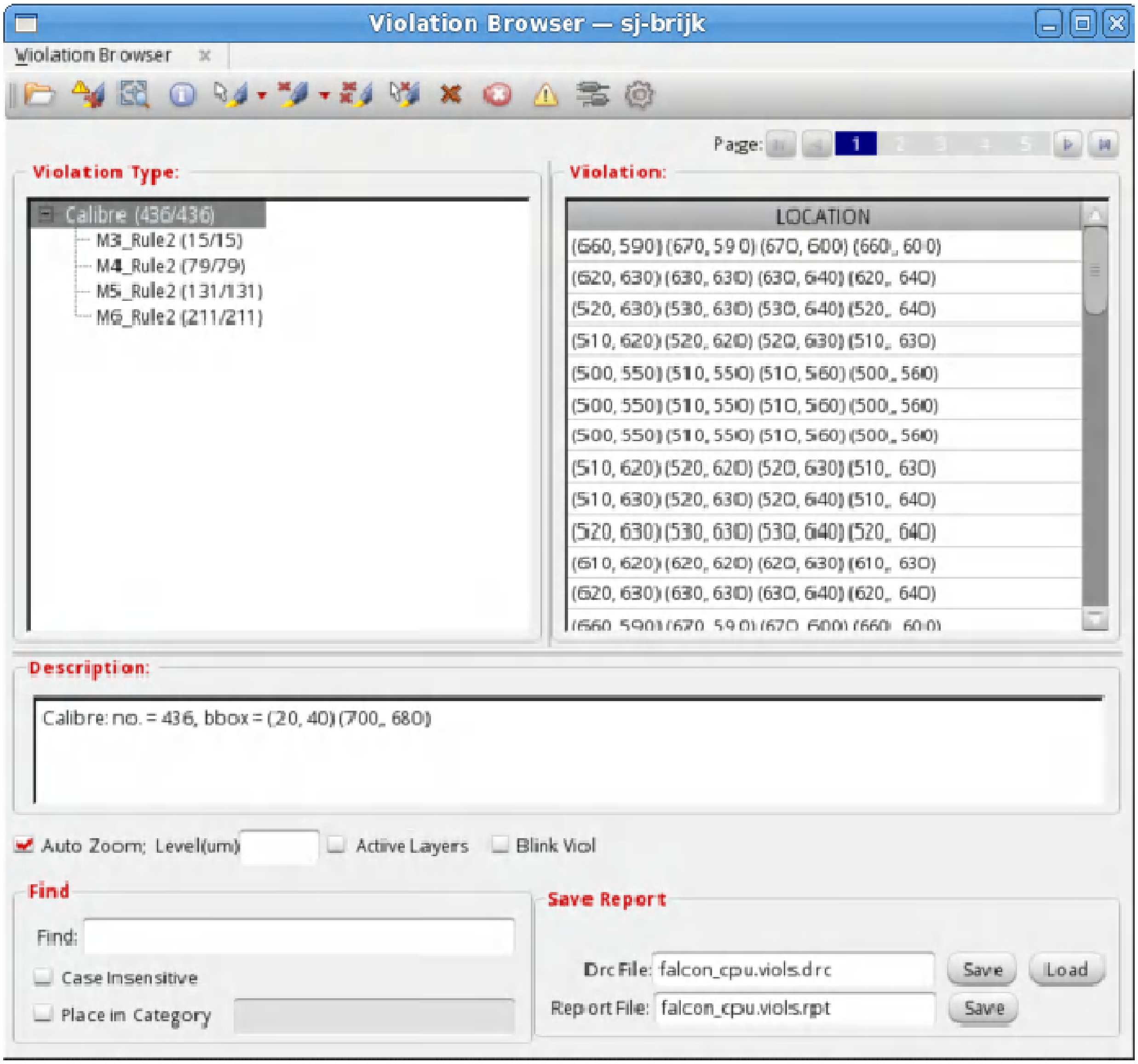
Note: You can also specify a different run directory name in the *Run Directory* field of the *CMP Status* window to check the output of another CMP run.

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.



Calibre (436/436)
- M3_Rule2 (15/15)
- M4_Rule2 (79/79)
- M5_Rule2 (131/131)
- M6_Rule2 (211/211)

LOCATION
(660, 590) (670, 590) (670, 600) (660, 600)
(620, 630) (630, 630) (630, 640) (620, 640)
(520, 630) (530, 630) (530, 640) (520, 640)
(510, 620) (520, 620) (520, 630) (510, 630)
(500, 550) (510, 550) (510, 560) (500, 560)
(500, 550) (510, 550) (510, 560) (500, 560)
(500, 550) (510, 550) (510, 560) (500, 560)
(510, 620) (520, 620) (520, 630) (510, 630)
(510, 630) (520, 630) (520, 640) (510, 640)
(520, 630) (530, 630) (530, 640) (520, 640)
(610, 620) (620, 620) (620, 630) (610, 630)
(620, 630) (630, 630) (630, 640) (620, 640)
(660, 590) (670, 590) (670, 600) (660, 600)

Calibre: no. = 436, bbox = (20, 40) (700, 680)

☑ Auto Zoom; Level(um) ☐ Active Layers ☐ Blink Viol

Find:
☐ Case Insensitive
☐ Place in Category

Drc File: falcon_cpu.viols.drc
Report File: falcon_cpu.viols.rpt

10

ECOs and Interactive Design Editing

ECO Flows

This appendix summarizes the variety of Engineering Change Order (ECO) flows possible with Innovus, and outlines the current approach for each flow.

- Overview
- Pre-Mask ECO Changes from a New Verilog File
- Pre-Mask ECO Changes from a New DEF File
- Pre-Mask ECO Changes from an ECO File
- Post-Mask ECO Changes from a New Verilog Netlist (Using Spare Cells Flow)
- Post-Mask ECO Changes from a New Netlist (Using Gate Array Cells Flow)
- Post-Mask ECO Changes from a New Verilog Netlist (Using Gate Array Filler Cells Flow)

Overview

Many types of ECO flows are possible. The ones outlined in this appendix cover the most common cases. You can use these flows directly, or you can modify them for your design.

For an example ECO file, see the ECO Directives chapter in the *Innovus User Guide*.

Assumptions

The ECO flows in this appendix assume the following:

- The old Verilog netlist and the new Verilog netlist have already been 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 Innovus formats, DEF, and so forth.

Flows

This appendix describes various types of ECO flows:

- Pre-Mask ECO Changes from a New Verilog File
  If you make changes to the netlist, use this flow to 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 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.gz`

  `oldchip.fp.spr.gz`

  Create these files (floorplan, special routing, placement, and routing) by using the `saveFPlan` command.

or

- `root.init (oldchip)`

  `oldchip.globals`

  `oldchip.def`
- Create this file (DEF formats for floorplan, placement, special routing, optionally routing) by using the `defOut` command.
- `newchip.v`

  The new Verilog file is typically created by manually editing the old Verilog netlist.

  Note: If you want to preserve routing, your existing design must contain the antenna diode cells that were added during the previous routing.

Flow

1. Read the new netlist
2. Load old floorplan/placement/routing data
3. Add level shifter or isolation cell for low power design (optional).
4. Remove filler cells and notches (optional)
5. Perform incremental placement
6. Add filler cells (optional)
7. Perform incremental or final route
   Or,

Use the `ecoDesign` command to perform the pre-mask ECO operations.

Steps

1. Read the new netlist.

   ```
   source newchip.globals # same as oldchip.globals except use newchip.v
   init_design
   or
   source oldchip.globals
   set init_verilog "newchip.v"
   init_design
   ```

   The netlist includes old libraries, global power connections, and so forth. It typically uses old timing constraints. However, new timing constraints can be used.

2. Read the old floorplan, special routes, placements and routing from the old floorplan and def files.

   ```
   loadFPlan oldchip.fp ecoDefIn -reportFile inecoDef.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 it's 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.

    ```
    readpowerintent -cpf test.cpf
    commit_power_intent -keepRows
    ```

    During this step, level shifters or isolation cells will be added to new ECO nets that cross the power domain. Retain the old design row definitions. The newly added cells will be unplaced. The `ecoPlace` command will place them in their respective power domain boundaries.

4. Remove filler cells or notch fill (if present).

    ```
    deleteFiller -prefix FILL
    deleteNotchFill
    ```

5. Perform incremental placement.

```
ecoPlace
```

    Unplaced instances are placed, previously placed cells are not moved, and routing is unaffected. You can manually preplace critical cells before using the `ecoPlace` command by placing the cell in the bottom, left corner, selecting it, and then moving it graphically.

    ```
    placeInstance i1/i2/i3 0 0
    selectInst i1/i2/i3
    ```

6. Add filler cells back into the rows.

    ```
    addFiller -cell {FILL4 FILL 2 FILL 1} -prefix FILL
    ```

    Global power connections are performed automatically based on rules loaded from the globals file or floorplan file earlier.

7. Perform incremental or final route.

    ```
    ecoRoute
    ```

    NanoRoute automatically detects modified and new nets and incrementally routes any nets that are incomplete or have violations.
    Or,

    Use the `ecoDesign` command to perform ECO operations. For example, the following command performs a pre-mask ECO:

    ```
    ecoDesign original.enc.dat top_cell newchip.v
    ```

    Use the `ecoDesign -noEcoPlace` or `ecoDesign -noEcoRoute` command to perform ECO operations. The command interrupts before `ecoPlace` or `ecoRoute`. This provides user more flexibility to control separate steps or perform some interactive ECO operations.

Pre-Mask ECO Changes from a New DEF File

In this flow, your design is placed and possibly routed, and you can make a few changes with known cell placements from a new DEF file.

Examples of this flow include:

- Bringing in an external clock tree after placement
- Bringing in external optimizations such as new buffers or cell resizing
- Bringing in external postRoute fixes such as new buffers or cell resizing

Preparation

Before starting the flow steps, you should have the following files available:

- `oldchip.enc`

  `oldchip.enc.dat/`

  Create these files by using the `saveDesign` command after the previous placement, optimization, and routing steps.

  or

  ```
  oldchip.globals
  oldchip.v
  oldchip.def
  ```

  Create these files by using the `saveNetlist` and `defOut` commands after the previous placement, optimization and routing steps.

- `newchip.def`

  The new DEF file is typically created by an external tool, or possibly done manually to fix a few critical postRoute violations with specific placements required. Any necessary physical cells (`+SOURCE DIST`) are expected to also be in the new DEF.

You must start with the old Verilog and update the Verilog modules with new ports and nets as required to match the new DEF netlist. You need to make sure the DEF instance names match the expected Verilog names (for example, a new buffer added to the output of instance `/i1/i2/i3` should have a name such as `/i1/i2/mynewbuf_i100)`, otherwise spurious Verilog ports will be created.

Flow

1. Read the old floorplan/netlist/placements and optionally the old routing.
2. Compare the old netlist to DEF
3. Load the ECO file
4. Write the modified netlist (optional)
5. Read the new placement data
6. Perform incremental or final routing

Steps

1. Read the old Verilog netlist, floorplan, and placement information into Innovus by doing one of the following:

```
restoreDesign oldchip.enc.dat top_name
```

or

```
source oldchip.globals
init_design defIn oldchip.def
```

This step reads in the following information:

- Old libraries and global power connections
- Old timing constraints (could be new constraints if necessary)
- Special routing, placements and optionally old routing
- Old filler cells, end caps, well taps, and other cell information

2. Compare the current old netlist to the new DEF netlist to create an ECO file.

```
ecoCompareNetlist -def newchip.def -outFile oldchip.eco
```

The ECO file has all the changes required to make the current netlist match the new netlist. Physical-only cells are ignored (for example, `+SOURCE DIST` cells such as filler, end caps, and well taps). Examine the ECO file to ensure it is correct.

4. 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.

5. (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 command.

6. 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.
   - Use 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.

7. Perform incremental or final routing.

   ```
   ecoRoute
   ```

   `ecoRoute` automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.

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 can make a small number of changes using an ECO file methodology. The changes are done before the masks are made so it is a pre-mask ECO flow, and there is no need to keep the original poly/diffusion patterns.

For example, you might want to apply a small number of late logical changes, but you want to keep as much of the previous placement, optimization, clock tree, and routing to avoid disturbing previous timing/SI optimization and repair.

Preparation

Before starting the flow steps, you should have the following files available:

- oldchip.enc
  oldchip.enc.dat/
  Create these files by using the `saveDesign` command after the previous placement, optimization, and routing steps.
  or
- oldchip.globals
  oldchip.v
  oldchip.def
  Create the first three files by using the `saveNetlist` and `defOut` commands after the previous placement, optimization and routing steps. The new Verilog file (newchip.v) is typically created by manually editing the old Verilog netlist.

  Note: If you want to preserve routing, your existing design must contain the antenna diode cells that were added during the previous routing.

  or
- `oldchip.eco`
  This file contains the list of changes to be applied to the old netlist. The changes required (see the `loadECO` command syntax) are typically created manually. You might be able to create an ECO file more easily by using `ADDINST` and `DELETEINST` rather than creating a new Verilog file.

Flow

1. Read the old netlist
2. Load the ECO file
3. Write the new netlist (optional)
4. Remove filler cells or notch fill (if present)
5. Perform incremental placement
6. Add filler cells
7. Perform incremental or final routing

1. Read the old Verilog netlist, floorplan, and placement information into Innovus.

   ```
   restoreDesign oldchip.enc.dat top_name
   ```

   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
   select_obj inst: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 -route_with_eco true    # set for incremental routing
   globalDetailRoute
   ```

   NanoRoute automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.
   Or,
   You can instead use the `ecoRoute` command to perform incremental or final routing.

8. Continue with the normal post-routing flow (analysis, repair, add metal fill, notch fill, verify, sign-off, and so forth).

   Note: ECO file contains ECO directive commands. The syntax of these commands is different from that of ECO tcl commands. For detailed information, refer the ECO Directives section of the Innovus Text Command Reference.

Post-Mask ECO Changes from a New Verilog Netlist (Using Spare Cells Flow)

Use this flow when:

- The design is taped out but has errors, or you need to add new features to the taped-out design.
- There is a new Verilog file that has only a few logical changes from the old Verilog file.
- You want to use the pre-existing spare cells so the poly/diffusion and lower layers are not changed, and only the metal and via layer masks need to be modified.

To save mask costs, you can direct the software to perform routing changes only on specific layers.

For this flow, you need the following files:

- `oldchip.enc*` (or `oldchip.globals, oldchip.fp*, oldchip.def`)

- `newchip.v` (this can be output from Conformal ECO Designer)

The original Verilog file already has spare cells because they are typically added by creating spare cells at the top-level or inside a Verilog module(s) just to hold the spare cells. The placer spreads the spare cells evenly throughout the design. If the design is hierarchical, you can add more spare cells inside modules that are likely to change.

Steps

1. Read the new netlist.

   ```
   source newchip.globals
   init_designor
   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 InDefeco.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 `-modifyOnlyLayers` option.
- All unplaced cells are mapped to spare cells during ECO placement in a later step.

3. (Optional) Low power related changes.

```
read_power_intent -cpf test.cpf
commit_power_intent -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.

7. (Optional) Swap spare cells.

   ```
   ecoSwapSpareCell i_9649 spare1
   ```

   At this step of the flow, all the newly added cells should be mapped. In this step, you can use the `ecoSwapSpareCell` command to change the mapping manually.
   `i_9649` is the instance name of the placed cell that `ecoSwapSpareCell` swaps with spare cell spare1 must be a spare cell specified in the previous step. Use the `specifySpareGate` command if necessary.

8. (Optional) Make tie connections.

   ```
   addTieHiLo -postMask [-cell "tieHighCellName tieLowCellName"] [-createHierPort
   {true | false}]
   ```

   During this step, the software reuses the existing tie cells to tie off a newly created spare instance in the design, instead of adding or deleting tie cells.

9. Perform incremental or final routing.

   ```
   ecoRoute -modifyOnlyLayers 2:3
   ```

   You can use the `-modifyOnlyLayers` option to restrict the modifications to a specified range of metal layers. If the `-modifyOnlyLayers` range begins with layer 2, and the spare cell pins are only available from metal 1, then the `ecoRoute` command automatically drops a VIA12 via. This behavior is not available if the `-modifyOnlyLayers` range does not begin with 2.

   The `ecoRoute` command might not be successful if the specified layer range is not sufficient to meet the changes required. You must restore the design from the previous step, then use a different range, such as 2:4, 1:3, and so on.

The unused routing segments of deleted and modified nets will appear in the SPECIALNETS section of the DEF file.

OR,

Use `eco_design` command to perform post-mask ECO operations. The following command and options are used to implement a post-mask ECO:

```
eco_design -post_mask -route_only_layers 2:3 -user_gate_array_cells GACORE -suffix
_spare oldchip.enc.dat top_name newchip.v
```

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 InDefeco.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 `-reportFile` 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 -reportFile 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.

5. (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.

6. (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.

7. 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.

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 -useGACells GA_site top.enc.dat top new.v
```

Post-Mask ECO Changes from a New Verilog Netlist (Using Gate Array Filler Cells Flow)

Currently, `ecoPlace` supports GACells and GAFillerCells flows. GACells flow is GACORE site based filler insertion while GAFillerCells is CORE site based filler insertion. Using the GAFillerCells flow, the tool can insert a GA Filler at any arbitrary grid rather than at the GACORE site grid to provide more available locations for post-mask ECOs.

The following diagrams illustrate the insertion of GA Fillers for GACells flow and GAFillerCells flow.

Figure 2 GA Fillers Insertion for GACells Flow

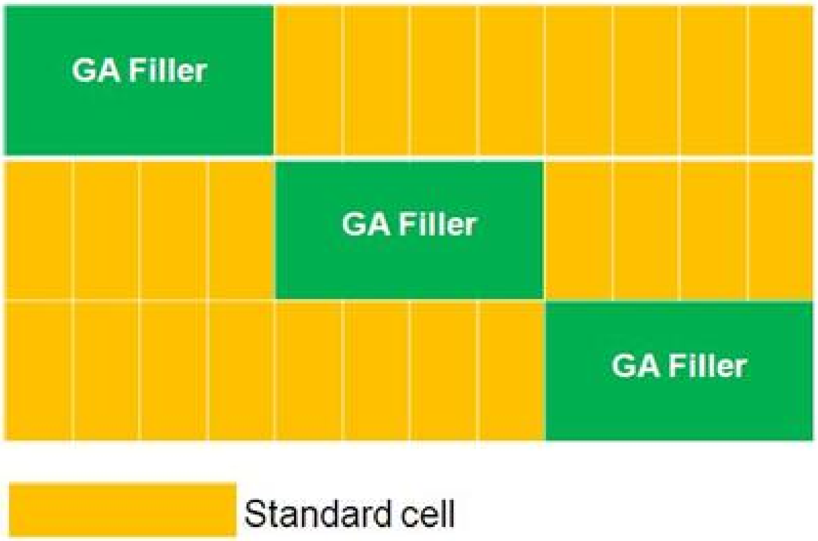
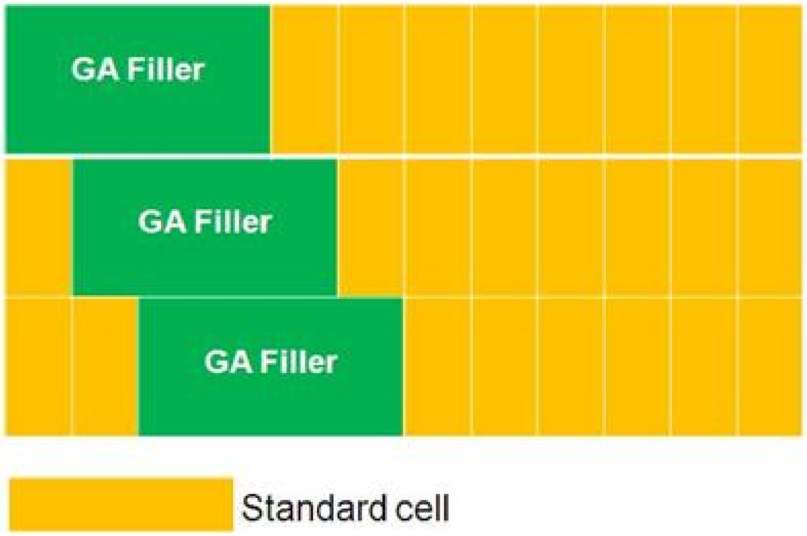


Figure 3



GA Fillers Insertion for GAFillerCells Flow

To perform incremental placement:

```
ecoPlace -useSpareCells true (#optional)
```

Unfix all GAFillers in the design

```
ecoPlace -useGAFillerCells {List of GAFillerCells }
```

This procedure does the following:

- If you need to place standard cells (such as site CORE) and GACells (such as site GACORE), you need to use the `ecoPlace` command twice. First, using the spare cell for unplaced standard cells, and then using the GA filler cells for unplaced GA function cell. Since the tool cannot distinguish standard cells from gate-array cells, you must first specify the GAFillerCell list. Change the GAFillers status from fixed to placed status.
- `ecoPlace` maps unplaced std cells to the same function spare cell (optional).
- `ecoPlace` placed GACells overlap with existing GA Fillers based on the connectivity, deleting the overlapping original GA Fillers and filling in the gap using new smaller GA Fillers.

Figure 3 and 4 illustrate the replacement of GA Fillers:

Figure 4

Before ecoPlace -useGAFillerCells {GAFILL1 GAFILL2 GAFILL3}

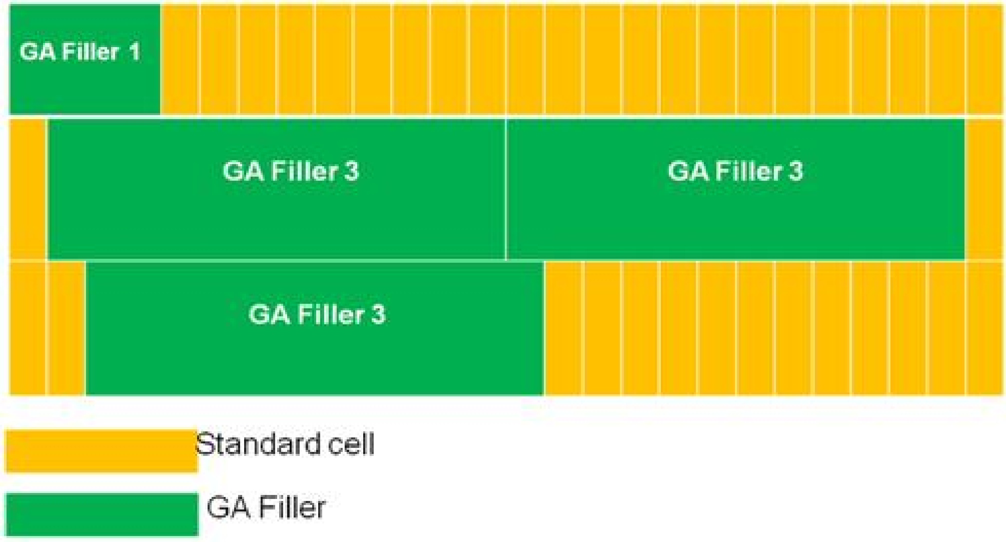
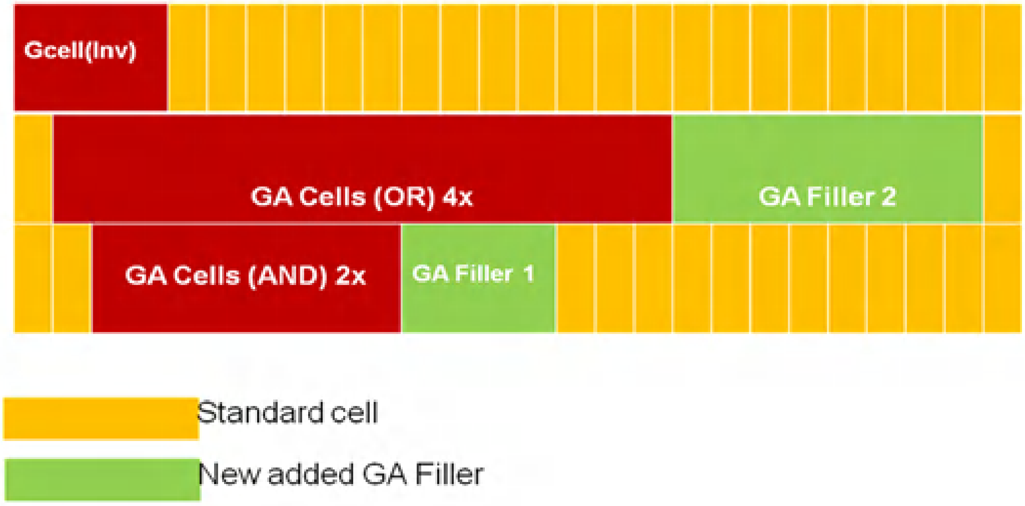


Figure 5

After ecoPlace -useGAFillerCells {GAFILL1 GAFILL2 GAFILL3}



ECO Directives

The ECO directives, presented in alphabetical order, are:

This appendix describes the directives that you specify in an ECO directives file. After you complete the file, you can then read it into the Innovus™ Implementation System (Innovus) by using the `loadECO` command on the Innovus command line. The following command loads `myDirectivesFile`:

```
loadECO myDirectivesFile
```

ⓘ These are ECO directives, not Innovus Tcl commands.

- You can use these directives only in an ECO directives file.
- You cannot use these directives on the Innovus command line.
- You cannot source this file to run the directives.
- You must use the `loadECO` command to read the file.

The names of the directives appear in this appendix in uppercase characters to distinguish them from interactive Innovus commands with the same names; however, the software is case-insensitive.

The ECO File directives do not support Verilog® escape name syntax. For directives that modify or delete existing objects, you cannot specify the name of the object using Verilog escape name syntax.

File format requirements are shown in the section Example ECO File.

ADDHIERINST

`ADDHIERINST`

`instName`

`moduleName`

Creates an instance of a new hierarchical module. You can later use the `ADDINST` directive to add spare cells inside the new hierarchical instance. The software automatically creates new ports for the hierarchical module when you run the `ATTACHTERM` directive. Currently, there are no directives available that allow you to manually create new ports for the new hierarchical module.

Parameters

`instName`	Specifies the name of the new hierarchical instance. If the instance already exists, or if the module containing the instance does not exist, the directive stops and the software displays an error message.
`moduleName`	Specifies the name of the new hierarchical module. If the module already exists, the software uses the module definition and creates a new hierarchy.

Example

- The following directive creates a new hierarchical cell, `sparecell`, and an instance of `sparecell` named `i1/i2/i3/spare1`:

  ```
  ADDHIERINST i1/i2/i3/spare1 sparecell
  ```

  If the instance `i1/i2/i3` does not exist, or instance `i1/i2/i3/spare1` already exists, the directive stops and the software displays an error message.

ADDINST

```
ADDINST [-moduleBased moduleName]
instName cellName nrTerm
INSTTERM
```

termName netName

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 moduleNamecellNameinstName
```

If `ADDINST` is not module-based, the syntax is:

```
ADDINST instName cellName nrTerm addPortAsNeeded INSTTERM termName netName
```

Parameters

cellName	Specifies the master of the instance.
instName	Specifies the name of the instance to add and place.
moduleName	Adds the new instance to the specified verilog module.
netName	Specifies the net name.
termName	Specifies the terminal name.
`nrTerm`	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` *`n1`*`:`*`n2`*	Adds a bussed port to the module. Specify the bus range (the beginning and end of the bus). Use integers to specify the range.
`moduleName` `| '-'`	Specifies the module to which you want to attach the port. To specify the top module, enter `'-'`.
`portName`	Specifies the name of the port to be added. The specified *`portName`* can be a new port name or any of the existing net names to which you want to add the port. The new port name can be the same as the existing net name. This port is attached directly to the existing net name if you specify the port direction (scalar port).

`input |`
`output |`
`bidi`
	Specifies whether the port is input, output, or bidirectional.

Examples

- The following directive creates an input port `p1` on instance `i1/i2/i3`. The port `p1` must be a new port name in instance `i1/i2/i3`. Instance `i1/i2/i3` contains no nets name p1:

  ```
  ADDMODULEPORT i1/i2/i3 p1 input
  ```

- 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` *`startID`*`:`*`endID`*	Creates a bussed Verilog net. The *`startID`* and *`endID`* indicate the first and last bits on the bus. You must separate the start and end IDs with a colon (:).
`-moduleBased` *`verilogModule`*	Adds the new net to the specified verilog module.
`netName`	Specifies the name of the net to add. If the module containing the net does not exist, or if the net already exists, the software displays an error message and the directive stops.
`-physical`	Adds a physical net.

Example

- The following directive adds net `i1/i2/net26` to the netlist:

  ```
  ADDNET i1/i2/net26
  ```

  If the module `i1/i2` does not exist, or if the net `i1/i2/net26` already exists, the software displays an error message and the directive stops.

ATTACHMODULEPORT

```
ATTACHMODULEPORT
{moduleName | '-'}
portName
netName
```

Attaches a port in the specified instance (or top level) to a net.

Parameters

moduleName | '-'	Specifies the module to which you want to attach the port. To specify the top module, enter `'-'`.
netName	Specifies the net to which you want to create to attach to the port.
portName	Specifies the port you want to create on the module.

Naming convention for directive ATTACHMODULEPORT module port net:

1. It does not allow to attach any module port to net inside the model.
2. A module port can be always attached to a net as long as all parent modules of the net do not reside inside the module where the port resides. For example,
   ATTACHMODULEPORT u1 port1 u2/u3/n1
   New hnets and new ports could be created with the similar naming convention to 1).
3. Attach module port to its parent (higher) level net.
   No new hnets are created except for those down hnets associated with new ports
   Port naming convention is similar to 1.

Note:

- Term can be instance term or hterm. for an instance term, the rules are as stated above. For hterm, it will automatically treated as a module port and processing it as stated in section 2 below.
- Net is a flatten net. All newly created hnet in this directive will be hooked on to net.
- Whenever an existing port can be used for the connection, no new ports will be created.
- Whenever new port name has any conflict with current DB objects, naming it to "p", "p_1",... until no name conflicts.
- According to current DB data model, whenever a new port is created, a down hnet with the same name of the port is automatically created to hold the port.

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

instName	Specifies the instance containing the terminal. If the instance name does not exist, the software displays an error message and the directive stops.
`-moduleBased` *verilogModule*	Attaches the terminal to the specified verilog module.
netName	Specifies the name of the net to attach to the terminal. If the net name does not exist, Innovus displays an error message and the directive stops.
`-noNewPort`	Prohibits Innovus from creating hierarchical ports when it attaches a terminal. If the terminal cannot connect to the net through existing ports, Innovus displays an error message and the directive stops. *Default:* If you do not specify this parameter, Innovus creates hierarchical ports as needed.
`-pin` *refInstName refPinName*	Specifies the pin *refPinName* on instance *refInstName* connected to the net that Innovus connects to the terminal. You cannot specify the `-port` parameter if you use this parameter.
`-port` *portName*	Specifies the hierarchical port used to connect the terminal with the net. If you specify this parameter, the hierarchical port must exist in the module that contains the instance. The hierarchical port must connect to the net. This parameter lets you use a specific port to maintain the same netlist topology, which simplifies equivalence checking later in the design flow. You cannot specify the `-pin` parameter if you use this parameter. *Default:* If you do not specify this parameter, Innovus uses existing ports or creates new hierarchical ports as necessary to connect the terminal to the net.
termName	Specifies the name of the terminal that Innovus connects to the specified net. If the terminal name does not exist, Innovus displays an error message and the directive stops.

Naming convention for directive ATTACHTERM term net:
This directive assumes term and flatten net must exist, or error message will be issued.
When term and net are not at the same hierarchical levels, some new port and new hnet nets may be added to complete connection between the term and the net.

1. When attach high level term to lower level hierarchical net, names of new net and new port is:

   `hnet name: U/<net_base_name>` where U is name of the module where the term resides.

   `port name: U1/<net_base_name>, U1/U2/<net_base_name>, ...`

   `U1/U2/.../Un/<net_base_name>` where U1, U2, ... Un are names of modules among the hierarchy from the term down to the net and Un is name of the module where the net resides.

   For example:

   ```
   ATTACHTERM u1/i1 t1 u1/u2/u3/n1
   ```

   Here the net base name is n1. One new hnet module u1 and two new ports at module u2 and u3, respectively, are needed for this connection:

   `hnet name: u1/n1`

   `port names: u1/u2/n1 and u1/u2/u3/n1` (two down hnets with the same name are also internally created to hold these two ports)

   If net is a bus bit u1/u2/u3/n[1], then the blasted name is used, i.e.

   `hnet name: u1/n_1` (for 15.1 or later), `u1/n_1_` (for 14.2 or older)

   `port names: u1/u2/n_1 and u1/u2/u3/n_1` (for 15.1 or later), `u1/u2/n_1_ and u1/u2/u3/n_1_` (for 14.2 or older)

2. When attach lower level term to higher level hierarchical net, names of new net and new port is:

   No hnets are needed to be created expect for those down hnets associated with the new ports Port naming is similar to 1.

Examples

- The following directive attaches terminal `in1` of instance `i1/i2/i3` to net `i1/i2/net26`:

  ```
  ATTACHTERM i1/i2/i3 in1 i1/i2/net26
  ```

  If `i1/i2/i3` does not exist, or if `in1` is not a terminal of `i1/i2/i3`, or if `i1/i2/net26` does not exist, the software displays an error message and the directive stops.
- The following directive attaches terminal `in2` of instance `i1/i2/i3` to net `net27`, using hierarchical port `myPort`:

  ```
  ATTACHTERM i1/i2/i3 in2 net27 myPort
  ```

The `myPort` port must exist in the module definition for `i1/i2`, and `myPort` must already connect to `net27`. If this is not the case, the software displays an error message and the directive stops.

- The following directive attaches terminal `in3` on instance `i1/i2/i3` through existing ports to `net28`:

```
ATTACHTERM -noNewPorts i1/i2/i3 in3 net28
```

If ports do not exist, the software displays an error message and the directive stops.

- The following directive attaches terminal `Y` of instance `testInst` to the verilog module `hier_t3` using the port `testPort`:

```
ATTACHTERM -moduleBased hier_t3 testInst Y testPort
```

CHANGECELL

```
CHANGECELL
instName
newCellName
[oldCellName]
```

Changes the cell type of an instance to a new cell type. You can only change cells that have the same footprint and functionality. If you provide the existing cell type (`oldCellName`), this directive verifies that the specified instance (`instName`) is a current instance of the existing cell type before changing the cell type to a new type (`newCellName`).

If the current instance is already placed, the modified instance keeps the existing placement. Keeping the old placement might cause overlap violations between the modified instance and neighboring instances if the new cell is larger than the existing cell.

Parameters

`instName`	Specifies the name of the instance whose cell type you want to change.
`newCellName`	Specifies the new cell type. The `newCellName` must exist in the library; otherwise, the software displays an error message and the directive stops.
`oldCellName`	Specifies the existing cell type. If the current cell is not `oldCellName`, the software displays an error message and the directive stops.

Example

The following directive changes the instance `i1/i2/i3` of type `and2_1x` to cell type `and2_2x`:

```
CHANGECELL i1/i2/i3 and2_2x and2_1x
```

If `i1/i2/i3` does not exist, the software displays an error message and the directive stops. If `and2_2x` does not exist in the library, or if all terminals of `and2_2x` do not exactly match the terminals of the current cell, or if the current cell is not `and2_1x`, the software displays an error message and the directive stops.

CHANGEINST

```
CHANGEINST
instName
newCellName
oldCellName
nrNewCellFTerm
TERM newTermName oldTermName
```

Changes the cell type (master) for an instance. Unlike the `CHANGECELL` directive, you can exchange the cell type for one with a different footprint but identical functionality. If you provide the existing cell type (`oldCellName`), the `CHANGEINST` directive verifies that the specified instance (`instName`) is a current instance of the existing cell type before changing the cell type to a new type (`newCellName`).

If the current instance is already placed, the modified instance keeps the existing placement. Keeping the old placement might cause overlap violations between the modified instance and neighboring instances if the new cell is larger than the existing cell.

Parameters

`instName`	Specifies the name of the instance you want to change.
`newCellName`	Specifies the new cell type. The `newCellName` must exist in the library and all terminals of `newCellName` must match the terminals of the current cell type exactly; otherwise, the software displays an error message and the directive stops.
`nrNewCellFTerm`	Specifies the number of terminals belonging to the new cell type.
`oldCellName`	Specifies the existing cell type. If the current cell is not `oldCellName`, the software displays an error message and the directive stops.
`TERM`	Specifies the old and new terminal names for the instance. You must specify a TERM directive for every terminal name that must change. • `newTermName`: Specifies the name of the terminal on the changed instance. • `oldTermName`: Specifies the name of the terminal on the existing instance.

Example

The following directives exchange a three-terminal flip-flop master (`FF3`) for a four-terminal master (`FF4`) for instance `i1/i2`.

```
CHANGEINST i1/i2 FF4 FF3 4
TERM D D
TERM Q Q
TERM CK CLK
```

Suppose that `FF3` has input `D` connected to net `n1`, clock terminal `CK` connected to net clock, and output `Q` connected to `n2`. `FF4` has input `D` connected to `n1`, clock terminal `CLK` connected to net clock, output `Q` connected to `n2`. Flip-flop `FF4` also has output `QN`. When the master for `i2/i2` is changed to `FF3`, `QN` is not connected to a net.

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

`instName`	Specifies the name of the instance.
`baseName`	Specifies the new base name to use for the instance.

DELETEBUFFER

```
DELETEBUFFER
instName
keepNetName
[deleteNetName]
```

Deletes a buffer instance after merging the nets on both sides of the buffer into one net.

ⓘ The `ecoDeleteRepeater` command provides the equivalent functionality.

Parameters

deleteNetName	Specifies the name of the net connected to the terminal on *instName* opposite to the terminal that connects to *keepNetName*. If *deleteNetName* is not connected to the terminal on *instName* opposite to the terminal that connects to *keepNetName*, the software displays an error message and the directive stops. For example, if *keepNetName* connects to the input of *instName*, *deleteNetName* specifies the name of the net connecting to the output. The software uses the *deleteNetName* for error checking only. When the `DELETEBUFFER` directive merges the nets, it might detach connections to hierarchical ports, but does not change the direction of hierarchical ports.
instName	Specifies the name of the buffer instance that the `DELETEBUFFER` directive deletes. The instance must have exactly one input terminal and one output terminal, and one of the terminals must connect to *keepNetName*.
keepNetName	Specifies the name of the existing net into which the nets from both of the instance's terminals are merged. The *keepNetName* net must connect to one of the instance's terminals.

Example

- The following directive deletes buffer `i1/i2/i3` and merges the nets from the buffer's two terminals into net `net26`:

  ```
  DELETEBUFFER i1/i2/i3 net26 i1/net25
  ```

  Net `net26` already connects to one of the instance's terminals. Net `i1/net25` connects to the terminal opposite the terminal that connects to net `net26`. If buffer `i1/i2/i3` does not exist, or if net `net26` and net `i1/net25` are not already attached to two terminals of buffer `i1/i2/i3`, the software displays an error message and the directive stops.

DELETEINST

```
DELETEINST
[-moduleBased verilogModule]
instName
```

Deletes an instance after deleting all the instance terminal connections to nets.

Parameters

instName	Specifies the name of the instance to delete. If the specified instance does not exist, the software displays an error message and the directive stops.
-moduleBased *verilogModule*	Deletes the instance from the specified verilog module.

Example

- The following directive deletes instance `i1/i2/i3`:

  ```
  DELETEINST i1/i2/i3
  ```

  If instance `i1/i2/i3` does not exist, the software displays an error message and the directive stops.

- The following directive deletes the instance `insta` from the verilog module `HIER_2`:

  ```
  DELETEINST -moduleBased HIER_2 insta
  ```

DELETEMODULEPORT

```
DELETEMODULEPORT
moduleName| '-'
portName
[netName]
```

Disconnects the specified port from its net and deletes the port.

You can use wildcards (`*?`) to specify the nets you want Innovus to delete.

Parameters

moduleName | '-'	Specifies the module from which you want to delete the port. To specify the top module, enter `'-'`.
netName	Specifies the name of the net from which you want to delete the port.
portName	Specifies the name of the port to be deleted

Example

- The following directive deletes `port1` from the top-level module:

```
DELETEMODULEPORT - port1
```

DELETENET

```
DELETENET
[-moduleBased verilogModule]
netName
```

Deletes a net after deleting all the instance terminal connections to the net. If routing is connected to the net, the routing is deleted.

You can use wildcards (`*?`) to specify the nets you want Innovus to delete.

Parameters

-moduleBased *verilogModule*	Deletes the net from the specified verilog module.
netName	Specifies the name of the net to delete.

Example

- The following directive deletes net `i1/i2/net26`:

```
DELETENET i1/i2/net26
```

If net `i1/i2/net26` does not exist, the software displays an error message and the directive stops.

DETACHMODULEPORT

```
DETACHMODULEPORT
moduleName
portName
```

Detaches the net connected to the specified port on the specified instance.

Parameters

moduleName	Specifies the module from which you want to detach the net. To specify the top module, enter '`-`'.
portName	Specifies the port on the module.

Example

- The following directive detaches port `p1` from `moduleA`:

  ```
  DETACHMODULEPORT moduleA p1
  ```

DETACHTERM

```
DETACHTERM
[-moduleBased verilogModule]
instName
termName
[netName]
```

Disconnects a terminal from a net.

Note: Detaching a terminal that drives an output terminal of a module produces a Verilog violation at the output terminal if you use `DETACHTERM`. Instead, use `ATTACHTERM` to attach the terminal to a new net. The `ATTACHTERM` directive automatically detaches the terminal from the net connecting to the output terminal, then attaches the terminal to the net you specify.

Parameters

instName	Specifies the instance that contains the terminal you want to detach.
-moduleBased *verilogModule*	Detaches the terminal from the verilog module.
netName	Specifies the net that is already connected to the terminal. If the terminal does not connect to the specified net, or if the net does not exist, the software displays an error message and the directive stops. The software uses this parameter for error checking only.
termName	Specifies the terminal to disconnect. If the terminal does not exist on the specified instance, the software displays an error message and the directive stops.

Example

- The following directive disconnects terminal `in1` of instance `i1/i2/i3`:

  ```
  DETACHTERM i1/i2/i3 in1 i1/i2/net26
  ```

  If instance `i1/i2/i3` does not exist, or terminal `in1` is not a terminal of `i1/i2/i3`, the software displays an error message and the directive stops. The net `i1/i2/net26` is specified, so if net `i1/i2/net26` does not exist, or if the terminal is not already connected to net `i1/i2/net26`, the software displays an error message and the directive stops.

INSERTBUFFER

```
INSERTBUFFER
[-noNewPorts]
netName
nrNetTerm
nrBuffer


INST instName cellName nrInstTerm


INSTTERM termName netName [portName]
```

...

```
NETTERM instName termName netName
```

...

Inserts a buffer on a net.

ⓘ This directive has been replaced by `addRepeaterByRule` 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]` *`netNamenrNetTermnrBuffer`* parameters.

Parameters

netName	Specifies the name of the net on which to insert the buffer. You must specify this parameter.	
-noNewPorts	Specifies that Innovus must not add new ports when inserting the buffer. If you try to insert a buffer that needs a new port, Innovus issues an error message and does not insert the buffer.	
nrBuffer	Specifies the number of buffers attached to the net.	
nrNetTerm	Specifies the original number of terminals attached to the net.	
INST	Specifies a buffer instance. You must specify the INST directive for each buffer you want to add.	
	instName	Specifies the name of the buffer instance to insert.
	cellName	Specifies the cell master for the buffer instance.
	nrInstTerm	Specifies the number of instance terminals contained in the buffer. Buffers have one input and one output terminal, so specify 2.
INSTTERM	Specifies a terminal on a buffer instance. The *termName* and *netName* parameters are required. you must specify the INSTTERM directive for each buffer you want to add.	
	termName	Specifies the name of the terminal on the buffer.
	netName	Specifies the net to connect with the terminal.
	portName	Specifies the physical port corresponding to the terminal.
NETTERM	Specifies the net connection between a driver terminal and the added buffer.	
	instName	Specifies the instance containing the terminal.
	netName	Specifies the net connected to the terminal.
	termName	Specifies the terminal connected to the net.

Example

- The following directives insert three buffers, b1, b2, and b3, on net0, which originally connects terminal out on instance i0 to receivers i1, i2, and i3. After the three buffers are added, terminal out drives one terminal: b1/in.

  ```
  INSERTBUFFER net0 4 3

      INST b1 buffer 2
      INSTTERM in net0

      INSTTERM out net1
      INST b2 buffer 2
      INSTTERM in net1
      INSTTERM out net2
      INST b3 buffer 2
      INSTTERM in net1
      INSTTERM out net3
      NETTERM i0 out net0
      NETTERM i1 in net2
      NETTERM i2 in net1
      NETTERM i3 in net3
  ```

  - Buffer `b1` has terminal `in`, connected to `net0` and `out`, connected to `net1`. Buffer `b1` drives buffers `b2` and `b3`, and connects to receiver `i2`.
  - Buffer `b2` has terminal `in`, connected to `b1` through `net1`, and `out`, connected to receiver `i1` through `net2`.
  - Buffer `b3` has terminal `in`, connected to `b2` through `net2`, and `out`, connected to receiver `i3` through `net3`.

Example ECO File

The file format consists of directives, each ending with a newline. The keywords are case insensitive.

Comments must begin with a pound symbol (#) as the leading, non-white space character, and end with a newline.

> ⓘ The first directive in the file must be `FORMATVERSION 2`.

```
#
# FORMATVERSION
#
FORMATVERSION 2
#
# ADDINST: Add at top level, no connectivity
#
ADDINST eco_inst_19 BUFX1

#
# ADDINST: Add at block level, no connectivity
#
ADDINST DTMF_INST/TDSP_CORE_INST/eco_inst_1 BUFX1

#
# ADDINST: Add at top level, with connectivity
#
ADDINST eco_inst_2341 BUFX1 2
INSTTERM A test_mode
INSTTERM Y reset

#
# ADDINST: Add at block level, with connectivity
#
ADDINST DTMF_INST/TDSP_CORE_INST/eco_inst_3 BUFX1 2
```

```
INSTTERM A scan_en

INSTTERM Y reset


#

# 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 Genus - 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. 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).
`RENAME_INST oldInstName newInstName`	Renames instances simultaneously in master/clone design.
`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.
`REMOVE_NET`	Removes the specified net. If there is any logical connection, the net is not removed.

`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
```

Support for Sizing Down the Bus Bit

HECO supports BUS bit trimming for both the top and submodules.

Top Level Bus Bit Port

For the top-level bus bit port trimming:

- Bus bit ports to be deleted must be adjacent with each other and adjacent to either msb or lsb of existing bus.
- Bus bit port to be deleted must be a one pin net.
- Bus bit port net with the same name will be deleted together with its port.

Syntax:

```
REMOVE_PORT <bportName> n:m
```

- The following ECO file removes four top level bits topIn[7], topIn[6], topIn[5], and topIn[4] from existing topIn [7:0]:

  ```
  FORMATVERSION 2
  START_HECO
  REMOVE_PORT topIn 7:4
  END_HECO
  ```

- In the following example, the ECO file tries to remove top-level bits **topIn[5]** from existing **topIn[7:0]**, which results in ERROR as it is not adjacent to msb or lsb:

  ```
  FORMATVERSION 2
  START_HECO
  REMOVE_PORT topIn 5:5
  END_HECO
  ERROR: (xxxx-xxx): Bus bit port %s specified by REMOVE_PORT must be adjacent to
  msb or lsb of existing bus bit port %s.
  ```

  Note: If the bus bit port to be deleted is not connected to a one-pin net, the tool will issue an error:

  ```
  ERROR: (xxxx-xxx): Bus bit port %s cannot be deleted because the net connected to
  it is not floating.
  ```

Submodules Bus Bit Net

For the bus modules bus bit net trimming:

- Bus bit nets to be deleted must be adjacent to each other and adjacent to msb or lsb of the existing bus.
- Bus bit net to be deleted must be floating.

Syntax

REMOVE_NET <bnetName> m:n

- The following ECO file removes two bus bit nets n1[0] and n1[1] on module bot1 (bot1 is instantiated as u0 under model mid) from existing bus n1[0:5]:

  ```
  FORMATVERSION 2
  START_HECO
  CURRENT_INST u1/u0
  REMOVE_NET n1 0:1
  END_HECO
  ```

- In the example below, the ECO file tries to remove two bus bit nets n1[2] and n1[3] on module bot1 (bot1 is instantiated as u0 under model mid) from existing bus n1[0:5], which will result in an ERROR as they are not adjacent to msb or lsb:

  ```
  FORMATVERSION 2
  START_HECO
  CURRENT_INST u1/u0
  REMOVE_NET n1 2:3
  END_HECO
  ERROR: (xxxx-xxx): Bus bit net %s specified by REMOVE_NET must be adjacent to msb
  or lsb of existing bus bit net %s.
  ```

  Note: If the bus bit net to be deleted is not floating at the other side, an ERROR will be issued:

  ```
  ERROR: (xxxx-xxx): Bus bit net %s cannot be deleted because the net is not
  floating.
  ```

Interactive ECO

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 RC extraction on the design, and the timing graph was built before running an ECO, then the RC extraction data and timing graph are incrementally updated.

Before You Begin

Before you can perform interactive ECO, the following conditions must be met:

- You must place and route the design
- You must load the design into the current session

Results

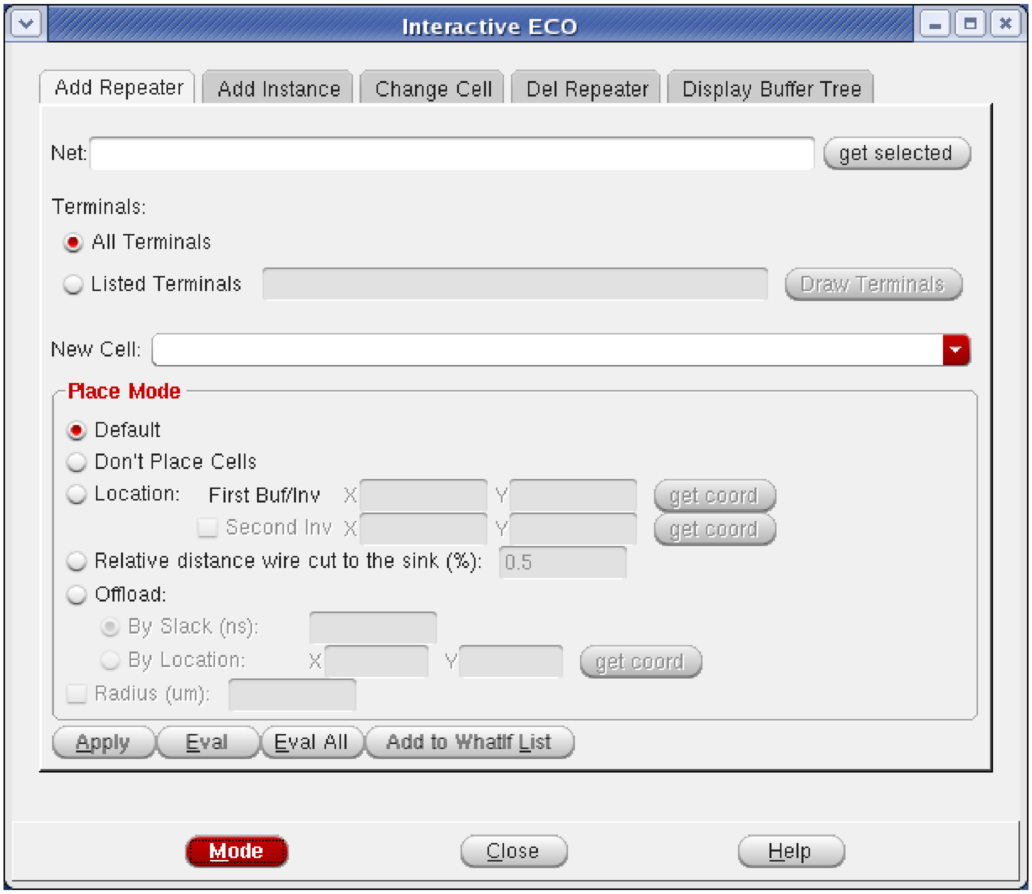
The following output files are generated:

- Updated netlist
- Updated placement

Adding Buffers

You can add a single buffer or a pair of inverters at a time on a net.

1. To open the Interactive ECO form, select *ECO - Interactive ECO* from the Innovus menu. This opens the Interactive ECO form. The *Add Repeater* page is selected.



Terminals: ● All Terminals

Place Mode: ● Default

○ Relative distance wire cut to the sink (%): 0.5

2. Enter the net name in the *Net* field.Type the net name, or click on a displayed net in the design display window and click *get selected.*

3. To select the terminals, choose one of the following:

   - To connect the added buffer to drive all the receivers, specify *All Terminals*. Use this to reduce the delay and output transition time of a weak driver driving a large capacitive load.
   - To connect the added buffer to drive the listed receivers, specify *Listed Terminals*. This provides full flexibility for building an arbitrary buffer connection.
   - *Draw Terminal* button - Allows you to draw an area covering the terminals to which you want to add the buffer.

4. In the *New Cell* field, enter the cell type name of the repeater to add, or click on the arrow to right of the field and select a buffer from the list.

5. In the *Place Mode*pane, you can choose one among several options:

   - *Default*

    The software automatically determines a location and places the new cell.

    - *Don't Place Cells*
      Specifies that the inserted cells should not be placed. Only the logical change in connectivity will be made.
    - *Location*
      Enter the location for the buffer using one of the following methods:
        - You can use the automatically assigned locations, enter the locations, or click on an area in the design display window and click *get coord*.
    - *Relative Distance to the Sink*
      Specifies the location of the buffer based on its distance from the sink or the driver pin. The value is a number between 0 and 1. A low value (0.1) places the buffer near the sink; a high value (0.9) places the buffer near the driver. The fraction is based on the length of the wire.

This option works when one term is provided; it does not work if no term or multiple terms are specified.

    - *Offload*
        a. To connect the added buffer to drive only noncritical receivers, select *By Slack*. This checks the timing graph for noncritical receivers and offloads those from the critical path, and could improve critical path timing by penalizing noncritical path delays.
        b. To add a buffer at a specific location, select *By Location* and enter the x, y coordinates.

- 6. Specify a radius.
    - Specify the radius in which the added instances are free to move. If no legal location can be found in the specified radius, the cells would be placed at the specified location resulting in an overlap with other cells. In that case, you should perform legalization.
- 7. (Optional) To legalize placement of the ECO changes, click *Do Refine Placement*.
- 8. Click *Apply*.
- 9. (Optional) Click the *Eval* button to evaluate the effect on timing if you add a new cell. The values are not applied in the database.
- 10. (Optional) Click the *Eval All* button to evaluate the effect on timing for all the cell types available for the new cell. The timing report shows the effects of all the cell types, enabling

you to select the best cell for your design. The values are not applied in the database.

Note: You can add a buffer around the I/O pin of a block using the `attachIOBuffer` command.

Note: By clicking the Mode button, you can open the Set Interactive Mode form to select different modes that control the behavior of ECO commands.

The following text command and parameters provide equivalent or additional functionality:

- `ecoAddRepeater`

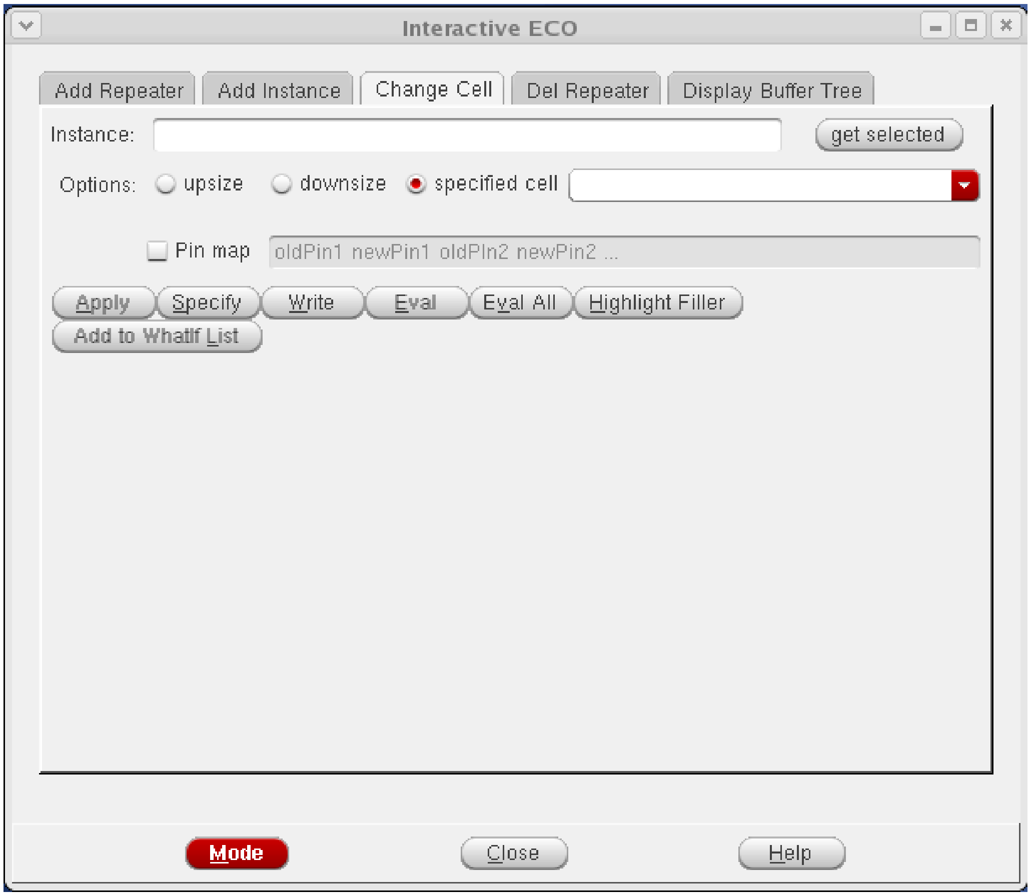
Note: When `ecoAddRepeater` is used in the post-mask ECO, ensure that it is used with the `-noPlace` option to avoid base layer change. Also, call `ecoPlace -useSpareCells` or `ecoSwapSpareCell` to map the new repeater to the spare cell. If you plan to call `ecoRoute -modifyOnlyLayers bottomLayer:topLayer` later, ensure that the added repeater can be connected through the specified layer. Otherwise, please do not use the `ecoAddRepeater`command. You can also use `loadECO <ecofile> -postMask` to add repeaters in the post-mask ECO.

For more information, see Interactive ECO Commands in the *Innovus Text Command Reference*.

Changing the Cell

You can upsize or downsize instances. Upsizing an instance that drives a large load can improve the driver delay and the transition time at the receivers. You can also downsize an instance on the noncritical path to reduce the loading of its driver on the critical path.

1. To open the Interactive ECO form, select *ECO - Interactive ECO* from the Innovus menu, and click the *Change Cell* tab. The Change Cell page is displayed.



2. In the Instance field, enter the hierarchical instance name to be changed. Type the instance name, or click an instance in the design display window and click *get selected.* Select either upsize, downsize, or specified cell. If you select specified cell, enter the replacement cell name in the adjacent field.

3. Type the cell name, or use the pull-down menu to select a cell.

4. (Optional) Specify the pin mapping for the new cell based on the old cell. This field is required if the new master cell has different pin names than the original 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`

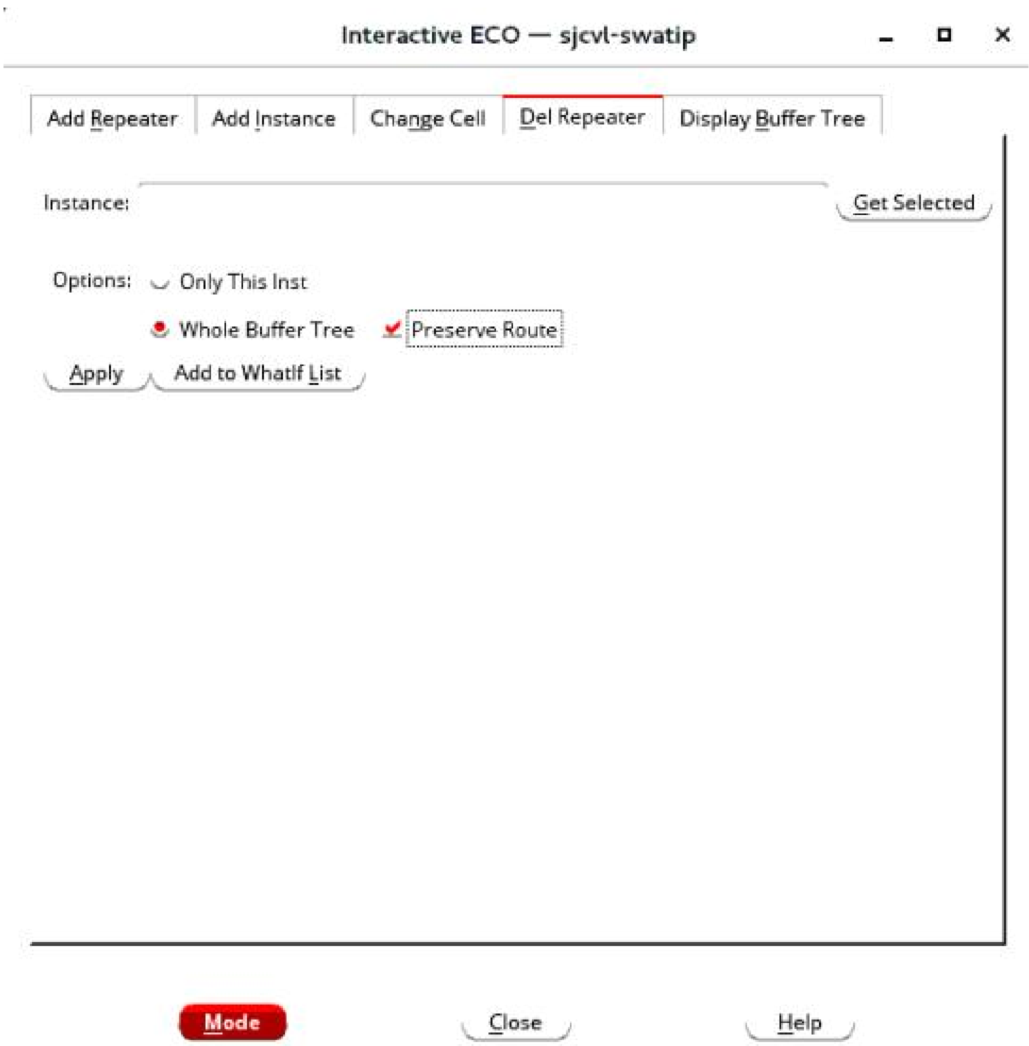
Note: The `ecoChangeCell` command cannot be used in the post-mask ECO. Please use l`oadECO <ecofile> -postMask`to change cells in post-mask ECO.

For more information, see Interactive ECO Commands in the *Innovus Text Command Reference*.

Deleting Buffers

You can delete redundant buffers that cause extra delay. Buffers are typically over-added by synthesis tools based on wireload models.

1. To open the Interactive ECO form, select *ECO* - *Interactive ECO* from the Innovus menu, and click the *Del Repeater* tab. The *Delete Repeater* page is displayed.



Instance:

Options: ○ Only This Inst

● Whole Buffer Tree ☑ Preserve Route

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`

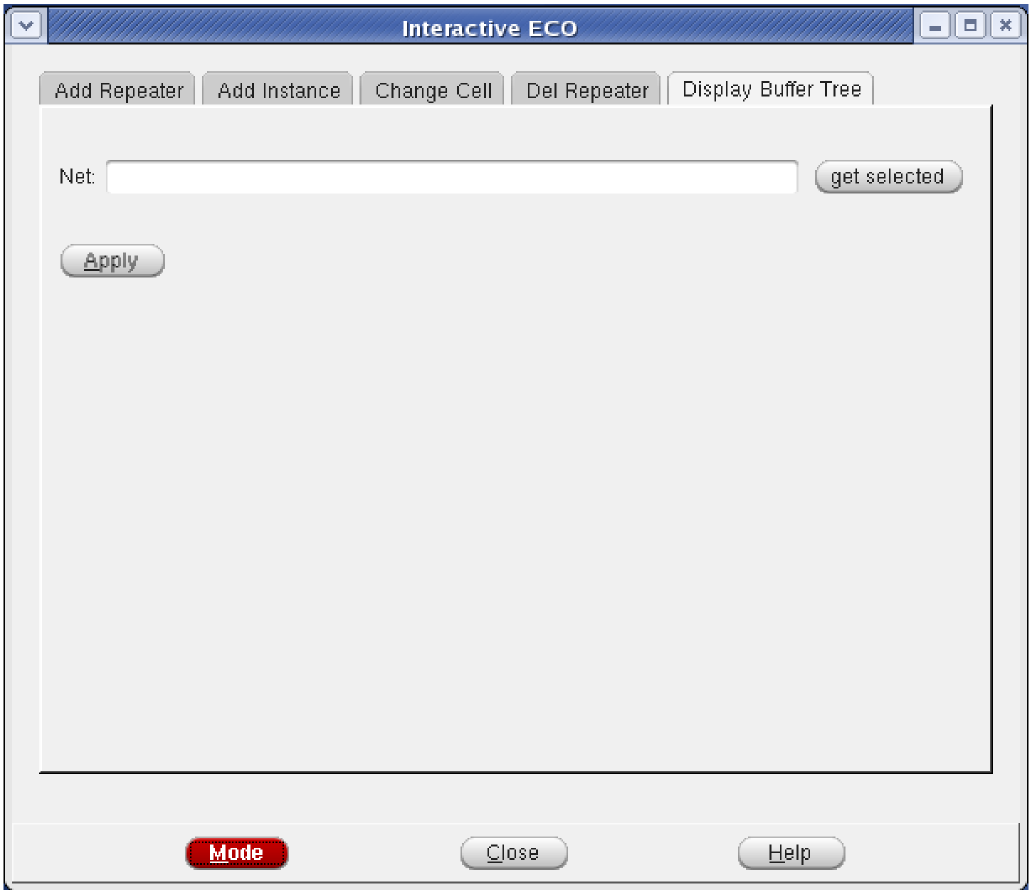
Note: The `ecoDeleteRepeater` command cannot be used in the post-mask ECO. Please use `loadECO <ecofile> -postMask` to delete repeaters in the post-mask ECO.

For more information, see Interactive ECO Commands in the *Innovus Text Command Reference*.

Displaying Buffer Trees

You can inspect the routing topology of the buffer tree after it is created. If the buffer tree requires correction, you can rebuild or modify it through the other three pages in the Interactive ECO form.

1. To open the Interactive ECO form, select *ECO - Interactive ECO* from the Innovus menu, and click the *Display Buffer Tree* tab. The Display Buffer Tree page is displayed.



2. To select a buffer tree, enter the net name in the *Net* field. You can type the net name, or click a net in the design display window and click *get selected.*

3. (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.
4. Click *Apply*.

Note: By clicking the Mode button, you can launch the Set Interactive Mode form to select different modes that control the behavior of ECO commands.

The following text command provides equivalent or additional functionality:

- `displayBufTree`

For more information, see Interactive ECO Commands in the *Innovus Text Command Reference*.

Running ECO Placement

ECO placement updates the placement from a prior Innovus session to reflect the netlist changes, merging the new netlist changes into the prior netlist's placement. The modified netlist can then be imported into an Innovus session so that the result is a new placement that reflects the changes made in the modified netlist.

You can run either an incremental timing or logic change to the design. You can run ECO after running placement, although ECO is usually run after analyzing speed or RC data.

To update the placement with the ECO netlist, complete the following steps:

1. Save the pre-ECO netlist placement data.
2. Start a new Innovus session.
3. Import the (ECO) design.
4. Load the floorplan.
5. Run ECO Placement.

This references the pre-ECO netlist placement data. ECO is performed on the new netlist during the pre-ECO placement. All designs are placed in the resulting placement.

After running ECO successfully, you can 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

Editing Wires

Overview

You can edit the wires and vias in your design manually by using the Edit Route form, the wire editing commands, and a combination of keyboard shortcuts (bindkeys) and tool widgets.

For signal wires, you can perform the following actions:

- Add wires
- Cut wires
- Move wires
- Change the wire to another layer
- Change wire width
- Change vias
- Delete wires
- Merge selected wires
- Force wires to use specified widths
- Add vias
- Trim selected wires
- Copy/paste wires

For power wires, you can perform all of the actions available for signal wires, as well as the following additional actions:

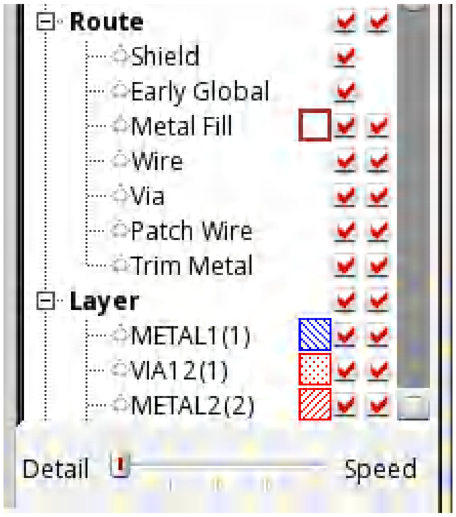
- Split selected wires
- Duplicate selected wires
- Fix wires wider than the maximum width
- Force wires associated with special nets to be created as signal wires

Patch wires are small, rectangular pieces of metal that do not follow the width restrictions of a regular wire. These non-standard wire patches are used to improve routability and fix Design Rule Check (DRC) violations. For instance, to fix a notch or overhang, a regular wire may not be effective and you may need to add a wire of arbitrary width. In such cases, you can use patch wires. As of now, you can perform the following actions on patch wires:

- Create a patch wire
- Select/deselect wires

- Move wires
- Delete wires
- Change the net name of a wire
- Copy/paste wires

You can control the display and selection of patch wires by using the *Visibility* and *Selectability* toggles for the new *Patch Wire* option under *Route* on the control panel.



For description of tabs and fields on the Edit Route form, see the Edit Menu chapter in the *Innovus Menu Reference*.

Before You Begin

Before you can use the wire editing features, load the design into the current Innovus session.

Wire Editor complies with the latest Multi-Mask Patterning (MMP) spec. It dynamically colorizes the wire segment and via being edited by automatically assigning and flipping the color on the active wire segment to avoid color conflict with neighboring objects on the same DPT layer. If multiple color conflicts cannot be resolved by flipping the *maskNum/color*, a color violation marker is flagged immediately. You then have a choice to override the default mask color as needed.

You can use the `setEditMode` parameter `-assign_multi_pattern_color {Auto | Mask1 | Mask2 | Mask3}` to specify how to assign the mask color on the DPT layer of the wire segment to be created. The default value is `Auto`. Alternatively, you can use the *Assign Multi-Pattern Color* drop-down in the *Advanced* tab of the Edit Route form to assign a mask color manually.

Note: After you use the wire editing features, the Innovus software saves the new and modified wires and vias in the database.

Using Keyboard Shortcuts

The Innovus software includes keyboard shortcuts for use with the wire editing features. Type the keyboard shortcuts while the main Innovus window is active and the cursor is in the design display area. Some of the keyboard shortcuts provide functionality that is not available through the Edit Route form or the wire editing commands.

Keyboard Shortcuts That Open Forms

Click in the design display area, then use one of the following shortcuts:

`D`	Opens or closes the Select/Delete/Deselect Route form
`E`	Opens or closes the Edit Route form

Keyboard Shortcuts That Are Equivalent to Tool Widgets

`A`	↖	*Select*
`Shift+A`		*Edit Wire*
`M`	↔	*Move/Resize Wire*
`O`		*Add Via*
`Shift+R`		*Move/Resize/Reshape* (non-connectivity-based move/resize/stretch)
`S`	↔	*Move/Resize Wire*
`Shift+X`	✂	*Cut Wire*

For more information, see "Tool Widgets" in the *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 *Nets* field of the Select/Delete Routes form is also populated. If the queried object is a pin, the layer and width information is set for both horizontal and vertical directions. If the queried object is a wire, the width information is set for both horizontal and vertical directions, but only one of the layers is set. That is, only the horizontal layer is set for a horizontal wire and only the vertical layer is set for vertical wires. This keyboard shortcut does not populate the form with spacing information.
`Ctrl+W`	Deletes the queried segment or via.

These keyboard shortcuts work only while you are in *auto query* mode--they do not work while you are drawing a wire. For more information, see "Auto Query" in the *Innovus Menu Reference*.

Keyboard Shortcuts Used in Edit Wire Mode

`D`	Changes the added wire to the layer below the current layer.
`U`	Changes the added wire to the layer above the current layer.
`Backspace`	Deletes the last segment created in the design area. This allows you to remove one segment of the route at a time.
`Esc`	Removes the entire route.
Number keys	Change the added wire to a specific layer number. If you want the wire to be added to metal layer 1, use the `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, using the `Shift` key and the arrow keys.
`2`	Stretches or reduces horizontal wires from the right and vertical wires from the top, using the `Shift` key and the arrow keys.

For more information, see Stretching Wires.

Keyboard Shortcuts Used to Change Vias

`Shift+N`	Changes the selected via to the next available via.
`Shift+P`	Changes the selected via to the previous available via.

For more information, see Changing Vias.

Selecting Wires

1. Click the *Select By Box* widget in the Tool Widgets area of the Innovus main window or press the `A` keyboard shortcut while the cursor is in the design display area.
2. Click a wire.

> ✓ Tip
> If multiple objects exist at the location of the cursor, press the space bar to toggle the selection among them. To select multiple objects, press the `Shift` key while clicking.

Deleting Wires

To delete a wire without deleting the vias connected to it, complete the following steps:

1. Turn on *Auto Query*. Q
2. Move the cursor over the wire to delete.
3. Use the `N` (next) or `P` (previous) keyboard shortcut to select the correct wire.
4. Press `Ctrl+W` or the `Delete` key.

Note: To delete a wire and the vias connected to it, use the `editDelete` command. To delete a wire without deleting the vias connected to it, you can use the `editDelete` command with the `-wires_only` option.

By default, when you delete a wire or via with `editDelete`, the tool searches and deletes all related DRC markers. If you do not want the related DRC markers to be cleared when you delete a wire or via, set the to `false` before using `editDelete`. This can improve the run time if have to delete a large number of wires and vias.

During post-mask ECO, you can freeze wires and vias by changing their status to `COVER`. The Innovus software does not delete wires or vias with status `COVER`. Type the following commands to freeze the wires and vias on metal layers 1 and 2:

```
deselectAll
editSelect -layer {METAL1 METAL2 VIA12} -object_type {Wire Via}
editChangeStatus -to COVER
```

Moving Wires

You can move wires in the orthogonal direction by using the `editMove` command, mouse, or the keyboard arrow keys (in conjunction with the `Shift` key).

Using the Mouse to Move Wires

1. Click the *Move/Resize Wire* icon in the Tool Widgets area of the Innovus main window.
   The equivalent keyboard shortcut is `M`.
2. Click the wire to be moved.
   The selected wire is highlighted.
3. Move the cursor slightly within the selected wire.
   The cursor changes to a circle shape.
4. Click and release the mouse.
   The wire moves with the cursor in the orthogonal direction (up or down for a horizontal wire, left or right for a vertical wire). Wires connected to the moved wires stretch to maintain connectivity.
   Note: The display area auto pans 20-25% at a time while moving the wire. For example, if you are moving a wire to the right of the area currently visible in the main window, the display area automatically pans to the right as you move the wire.
5. Click the mouse again to place the wire in the new location.
   Note: To cancel the move before you click the mouse, press the `Esc` key. The wire returns to its original location.

Note: If you select the *Snap to Track* option, the wire automatically snaps to the appropriate routing track.

Using Arrow Keys to Move Wires

1. Choose *Edit* - *Wire* - *Edit* from the menu.
   The Edit Route form opens.
   The equivalent keyboard shortcut is `E`.
2. Click the *Misc* tab.
3. Specify a value, in microns, in the *Arrow Increment* field.
   This value defines the distance that the wire is to move each time you press an arrow key while holding the `Shift` key. You can specify either a positive or negative number.
4. Click the *Move/Resize Wire* icon in the Tool Widgets area of the Innovus main window.
   The equivalent keyboard shortcut is `M`.
5. Click the wire to be moved.
   The selected wire is highlighted.
6. Hold the `Shift` key, then press the up or down arrow key for a horizontal wire or the left or right key for a vertical wire.
   The selected wire moves in the direction of the arrow.

Copying Wires

You can copy wires and vias by using the following methods:

- Using the `editCopy` command (all wires and vias)
- Using the mouse (all wires and vias)
- Duplicating and moving (only special wires and vias)

Using the editCopy Command

To copy a wire or via to a specific location, use the `editCopy` command. You can specify whether or not the copied object should retain the net name of the original object. You can also assign a specific net to the copied object. If required, you can copy the selected wires multiple times by using the `-times` parameter.

Using the Mouse to Copy Wires or Vias

To copy wires or vias using the mouse, complete the following steps:

1. Select the wire or via to be copied.
2. Use the `C` bindkey or click the *Copy* icon () to switch to *Copy* mode.
3. (Optional) Press `F3` to open the Copy form. Select the direction in which you want to move the copied object from the *Move Direction* drop-down. You also choose to retain net name of the original object in the copied object. Alternatively, you can assign a specific net to the copied object.
4. Move the mouse over any of the selected objects. The cursor changes to a black dot.
5. Click once to start copying the selected object. A ghost image of the original object will move along with the cursor.
6. Click again to place the copy at the desired location.

Copying and Moving Special Wires or Vias

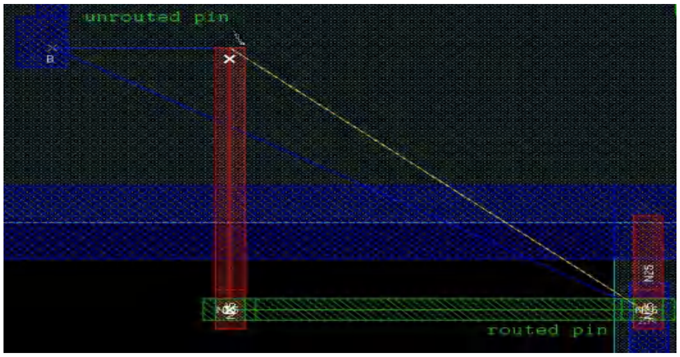
To copy/paste and then move selected special wires or vias, complete the following steps:

1. Select wires or vias.
2. Type the `editDuplicate` command (or use the `C` keyboard shortcut) to copy the objects. The duplicate object is created directly on top of the original object.
3. Use the `Shift+R` keyboard shortcut or click the *Move/Resize/Reshape* icon to switch to *non-connectivity-based move* mode.
4. Move the mouse over any of the selected objects. A black dot appears.
5. Click once to start moving the selected objects.
6. Click again to place the objects in the desired location.

Note: To cut and paste, and move selected special wires or vias, skip step 2.

Adding Wires

You can add one or more wires interactively to single or multiple nets. When you add wires, the flight lines to routed pins are displayed in the pin color (by default, yellow) and flight lines to unrouted pins are displayed in the wire color (by default, blue).



By default, the routing status for newly added signal wires is `FIXED`. A `FIXED` routing status means that the automated routers do not rip up and reroute preroutes. Signal wires that are moved, cut, or otherwise changed by wire editing commands maintain the routing status that was set before the wire editing commands were issued.

Adding a Wire for a Single Net

1. Click the *Edit Wire* widget (or press `Shift+A`).
   This places the Innovus software in the *Edit Wire* mode and the mouse cursor changes to a pencil . In addition, Innovus is automatically placed in the *Auto Query* mode, even if the *Q* widget below the design display area is not enabled.

2. If pins are not visible, select *Pin Shapes*. Pin Shapes ☐☑☐ in the *Cell* group on the Layer Control bar.
3. Place the cursor over the pin or wire at the starting point for the wire to be drawn, and then type `Shift+S` while the cursor is in the design display window.
   This populates the Edit Route form with the net name, layer, and width information that is used in creating new wires.
   Note: If multiple objects exist at a particular point, use the `N` or `P` keyboard shortcut to cycle through the objects.
4. (Optional) Choose *Edit - Wire - Edit* from the menu or use the `E` keyboard shortcut.

The Edit Route form opens, and has been automatically populated with the net name, layers, and widths. The form is not populated with spacing information, which only applies while editing multiple nets.

5. Click the *Basic* tab on the Edit Route form and adjust the values in the *Layer* and *Width* fields.
6. (For special wires only) Select a shape from the *Shape* drop-down menu on the *Basic* tab.
   Note: Shapes are only defined for power/special wires. This value is ignored for signal wires.
7. Click the start point for the wire you want to add, then move the mouse to a new point.
   The wire is drawn interactively as you move the mouse.
8. Click a new location to change the direction of the wire or continue in the same direction with a different segment.
   Note: If there is a layer change, a via is automatically created.

   ✓ Tips

   - Press a number key to change the layer of the wire being added.
     When the software is in the *Edit Wire* mode, number keys can be used as keyboard shortcuts, with the number indicating the layer number of the wire being drawn. For example, if you press the number `2`, the segment is added to metal layer `2`. Alternatively, you can use the `U` or `D` keyboard shortcuts to change the layer of the segment. The U keyboard shortcut changes the segment to the next higher layer, and the `D` keyboard shortcut changes the segment to the next lower layer.
   - Press the `Backspace` key to erase the most recently drawn segment.
     You can do this for as many segments as needed.

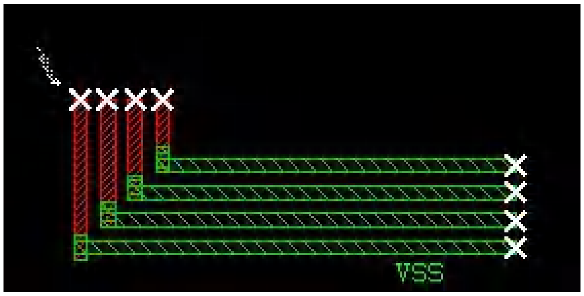
9. Double-click the mouse.
   The wire ends at the location of the cursor.
   Note: After double-clicking, you cannot use the `Backspace` key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the Edit Delete form.

Note: Widths for signal wires depend on the applicable LEF rule, no matter what value is populated in the GUI. To specify a wire width that is different from the default wire width value, select a non-default rule other than *Default* from the *Rule* drop-down menu on the *Basic* tab of the Edit Route form.

Adding Parallel Wires for Multiple Nets

To add parallel wires for multiple nets at the same time, complete the following steps:

1. Click the *Edit Wire* widget (or press `Shift+A`).
   This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil.
2. Choose *Edit - Wire - Edit* from the menu or use keyboard shortcut E.
   The Edit Route form opens.
3. Select the appropriate *Route Action - Create Regular Wire* or *Create Special Wire*.
4. Click the *Basic* tab on the Edit Route form and enter the net names in the *Nets* field.
   Note: You can also specify a file that contains a list of nets. See Adding Wires that Automatically Extend to a Target for more information.
5. (Optional) Select horizontal and vertical layer names and specify width and spacing values.
   Note: To use different width or spacing values for different nets, use the *Multi-Net* tab. See Using Override to Add Wire Groups with Multiple Widths and Spacing for more information.
6. (Optional) Specify a value in the *Drawing Wire* field on the *Multi-Net* tab.
   This specifies which of the nets (specified in the *Nets* field) corresponds to the mouse pointer location. By default, this value is `1`, meaning the mouse position corresponds to the position of the left-most or bottom-most net of the group.
   For example, if the *Nets* field contains `VSS VDD VDDA VSSA,` the `VSS` net is the bottom-most net for horizontal segments, and the left-most net for vertical segments. If the value in the *Drawing Wire* field is set to `1`, the mouse location corresponds to wires on the VSS net.



7. (For special wires only) Click the *Basic* tab and select a shape from the drop-down menu.
   Note: Shapes are only defined for power wires. This value is ignored for signal wires.
8. Click the start point for the wires you want to add, then move the mouse to a new point.
   The wires are drawn interactively as you move the mouse.

9. (Optional) Click a new location to change the direction of the wires or to continue in the same direction with a different segment.
   Note: If there is a layer change, a via is automatically created.

   > Tip
   > Press the `Backspace` key to erase the most recently drawn set of segments. You can do this for as many sets of segments as needed.

10. Double-click the mouse.
    The wires end at the location of the cursor.
    Note: After double-clicking, you cannot use the `Backspace` key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the Edit Delete form.

Adding Wires that Automatically Extend to a Target

To create a wire group for multiple nets that automatically extend to targets, complete the following steps:

1. Click the *Edit Wire* widget .
   This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil.
2. Choose *Edit* - *Wire* - *Edit Route* from the menu.
   The Edit Route form opens.
3. Select the appropriate *Route Action - Create Regular Wire* or *Create Special Wire*.
4. Create a text file that contains the names of multiple nets.
   Make sure that each line in the file contains the name of one net, and that the nets are listed in the order in which you want to create the wire group.
5. Click the *Load* button on the *Basic* tab of the Edit Route form.
   This opens the Open form.
6. Select the file you created in step 4, then click *OK*.
   The *Nets* field now contains the net names in the file.
7. On the *Basic* tab, select horizontal and vertical layer names and specify width and spacing values.

Note: To use different widths or spacing values for different nets, use the *Multi-Net* tab. See Using Override to Add Wire Groups with Multiple Widths and Spacing for more information.

8. Click the *Misc* tab, and select the *Extend Start Wires* and *Extend End Wires* options. These options extend both ends of the wires until they connect to a target.
9. Click the point in the design display area where the left-most or bottom-most wire should start.
   Note: The start point does not have to be at a target.
10. Move the mouse horizontally or vertically.
    The wires are drawn interactively.
11. Double-click the mouse.
    The start point and end point of the wire extend until they connected to a target. If no target is present, the wire does not extend.

Using Override to Add Wire Groups with Multiple Widths and Spacing

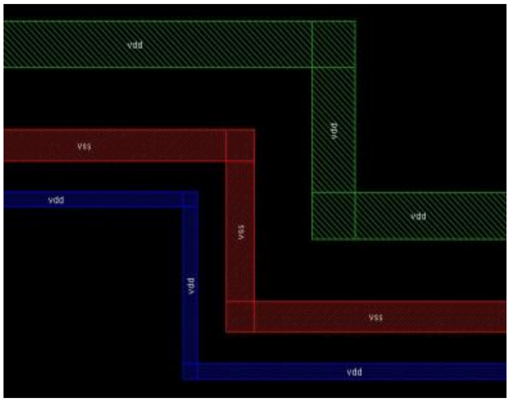
To add pairs of power and ground wires, where wires have different widths and spacing, complete the following steps:

1. Click the *Edit Wire* widget .
   This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil.
2. Choose *Edit - Wire - Edit* from the menu. The Edit Route form opens.
3. Click the *Create Special Wire* option button.
4. Click the *Basic* tab on the Edit Route form and enter the net names into the *Nets* field. For example, "vdd vss vdd" for illustration purpose.
   Note: You can also specify a file that contains a list of nets. See Adding Wires that Automatically Extend to a Target for more information.
5. On the *Basic* tab, select the horizontal and vertical layer names and specify the width and spacing values. These settings will be used for non-overrided nets.
6. Click the *Mutli-Net* tab and then enter a set of width and spacing values for the nets that do not have default width and spacing values.
   For example, if you specify the following values:

```
1 0.1 0.2 1
2 0.2 0.4 2
3 0.3 0.6 3
```

The first line indicates that the first net (`vdd`) has a width of 0.1 microns; the spacing value between the first and second net is `0.2` microns; and the first net is routed only on layer 1. The second line indicates that the second net (`vss`) has a width of `0.2` microns; the spacing value between the second and third net is `0.4` microns; and the second net is routed only on layer 2. To specify the same constraints for multiple nets, use ',' to separate the net numbers. For example, if you specify "`netNum1,netNum2,netNum3 width spacing layer`", all three nets will have the same width, spacing, and layer number.

Note: To specify a value of less than `1`, you must include a `0` before the decimal point. For example, a value of `.6` is not valid, and must be expressed as `0.6`.



7. Close the Edit Route form.
8. Click the point in the design display area where the leftmost stripe should start.
   Note: The start point does not have to be at a target.
9. Double-click the mouse.
   The wires end at the location of the cursor.

Cutting Wires

You can use the *Cut Wire* widget to cut wires and bus guides on the visible layers. There are two operating modes:

- *Cut Wire by Line* (default): When you click this widget, the cursor changes to scissors in the design display area. Move the cursor to draw a draw a line indicating where to cut a wire. The cut must go all the way through the wire. You can cut special wires horizontally and vertically.

The results retain the same direction as the original wire. You can also cut rectangles.

- *Cut Wire by Box*: Select the *Cut by Box* option from the drop-down menu to switch to the *Cut Wire by Box* mode. A square appears on the *Cut Wire* widget (▯) to indicate that you are now in the *Cut Wire by Box* mode. In this mode, you can cut a wire or bus guide by drawing a cut-box to indicate the places from which the wire should be cut. This mode makes it possible to cut off a wire in a single step. Only the wires that are perpendicular to the cut-box edge are cut. In addition, the cut-box edge must pass across the wire, bus guide, or rectangle completely.

You can also use the drop-down menu of the widget to choose between cutting all visible wires or selected wires overlapping with the cut line or box.

Given below are steps for cutting wires using the default *Cut Wire by Line* mode:

1. Click the *Cut Wire* widget ✂.
   The cursor changes to the shape of a scissors, indicating that the Innovus software is in the *Cut Wire* mode.
2. Click the location at which you want to start cutting the shields.
3. Move the mouse so that the drawn line is touching or overlaps the wire orthogonally.
4. Click to complete the cut.
5. Use the `A` keyboard shortcut to enter the *Select* mode.

   The cursor changes to an arrow shape.
6. Click the piece of wire to be deleted.
   The selected piece of wire is highlighted.

   > Tip
   > If multiple objects exist at the location of the cursor, press the space bar to toggle the selection among them. To select multiple objects, press the `Shift` key while clicking.

7. Use the `D` keyboard shortcut to delete the selected objects.

Note: Wires can only be cut in the orthogonal direction. If you cut multiple wires, including wires in the same direction as the cut, the cut only affects wires in the orthogonal direction to the cut. Once cut, signal wire pieces maintain a 1/2 wire width extension, but power wires are not extended.

Alternatively, you can cut wires from the command line by using the `editCutWire` command.

Trimming Antennas on Selected Stripes

If your completed power structure contains stripes in a mesh configuration, physical antennas might remain on some stripes.

1. Use the `D` keyboard shortcut to display the Select/Delete Routes form.
2. Choose *Select* from the *Action* drop-down menu.
3. (Optional) Click *Nets*, then specify one or more nets for the wires to be trimmed.
4. (Optional) Click *Direction*, then click *H* to trim horizontal wires or *V* to trim vertical wires.
5. Click *Shapes*, then select *STRIPE*.
6. Click *Apply*.
   The selected wires are highlighted in the design display area.
7. Use the `Shift+T` keyboard shortcut or click the widget at bottom of the Edit Route form to trim the selected wires.
   The selected power wires are trimmed back to the last connection point and deselected.

Changing Special Wire Width

After running power analysis, you might need to increase the width of some power stripes to alleviate any IR drop or EM issues.

1. Make sure the software is in *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to be widened.
2. Use the `E` keyboard shortcut.
   This displays the Edit Route form without placing the software in the *Edit Wire* mode.
3. Click the *Create Special Wire* option.
4. Click the *Basic* tab on the Edit Route form and enter values in the *Width* fields.
   Specify a width value in the *Horizontal* section for horizontal wires and a width value in the *Vertical* section for vertical wires.
5. Use the `Shift+W` keyboard shortcut or click the widget at bottom of the Edit Route form.
   This changes the width of the selected wire. Any via connected to that the wire is also updated based upon the new width.

Repairing Maximum Wire Width Violations

Violations occur if you specify wires widths greater than the maximum width defined by the `MAXWIDTH` value 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` value and splits up both the wires and the associated vias as minimally as possible while maintaining the same footprint.

Duplicating Special Wires

After running power analysis, you might need to add some power stripes to alleviate any IR drop or EM issues. Instead of creating new wires interactively, you can duplicate existing special wires.

1. Make sure the software is in the *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to duplicate.

2. Click the *Duplicate Selected Wires* widget or use the `C` keyboard shortcut.

   The duplicated wire is automatically selected and placed directly on top of the original wire.

   Note: The width and layer of the duplicated wire are always the same as the original wire. To duplicate a wire and change the layer, use the `editDuplicate` command and specify the layer for the duplicate wire. For more information, see `editDuplicate` in the "Wire Edit Commands" chapter of the *Text Command Reference*.

3. Use the `M` keyboard shortcut.

   This places the software in the *Move* mode, allowing you to use the mouse or the arrow keys (while holding down the `Shift` key) to move the newly created wire segment to the desired location.

Stretching Wires

1. Click the *Select By Box* widget in the Tool Widgets area of the Innovus main window.
   The cursor shape is an arrow, indicating that Innovus software is in the *Select* mode. The equivalent keyboard shortcut is `A`.
2. Click the wire to stretch.
   The selected wire is highlighted.
3. Click the *Move/Resize Wire* widget in the Tool Widgets area of the Innovus main window.
   The equivalent keyboard shortcut is `S`.
4. Move the cursor to the end point of the wire to be stretched. The cursor changes to a T shape .
5. Click the end point, then release the mouse button and move the cursor to a new location and click again. The wire stretches to the new location.

Alternatively, you can use the `Shift` key in conjunction with the arrow keys to stretch or shrink the wire. When the software is in the *Move/Resize Wire* mode, you can use `1` and `2` as keyboard shortcuts to set the edge of the wire to be stretched. By default, the wire is stretched from the top or the right using the arrow keys regardless of the cursor location. To stretch the wire from the bottom or the left, use the `1` keyboard shortcut. The *Move/Resize 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 *Move/Resize Wire* widget changes back to the original picture and the software is in the default stretch mode.

The Floorplan Move/Size/Reshape command (`Shift + R` bindkey) can also be used to resize and stretch wires, in addition to moving, without checking the DRC. Use the `editResize -no_conn` parameter to specify whether the tool should honor wire connectivity and drop vias between wires in different layers. If you set `-no_conn` to:

- `1`: No via will be created or removed.
- `0`: Tool will create or remove via, if necessary.

Currently, resizing or stretching using `Shift+R` is supported only for horizontal and vertical wires and not for 45-degree wires. Only special wires support resizing.

Changing Wire Layers

You may need to change sections of wires to different layers in order to relieve congestion on a specific layer or to fix process antenna violations.

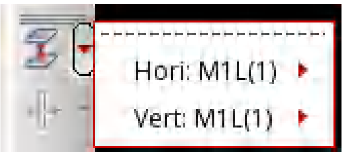
1. Make sure the software is in the *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to be updated.
2. Use the `E` keyboard shortcut.

   This displays the Edit Route form without placing the software in the *Edit Wire* mode.
3. Click the *Basic* tab on the Edit Route form and enter values in the *Layer* fields.

   Specify a layer value in the *Horizontal* section for horizontal wires and a layer value in the *Vertical* section for vertical wires.
4. Use the `Shift+L` keyboard shortcut or click the widget at the bottom of Edit Route form.

   This changes the layer of the selected wire. Any via connected to that wire is also updated automatically based on the new layer.

You can also change the wire layers easily by using the drop-down of the *Change Layer of Selected Wire* widget on the Wire Edit toolbar on the main window. After selecting the wire, choose the required layer from the horizontal or vertical layer submenus in the drop-down of the widget.



Splitting and Merging Special Wires

Stripes that spread over the entire die may need to be altered only in specific locations. In this case, a stripe that is represented as a single piece of wire segment must be split into multiple segments before any local editing. You can split a single stripe into multiple cut stripes at each crossover automatically:

1. Make sure the software is in the *Select* mode (you can use the `A` keyboard shortcut), then click the wire segment to be split.
2. Use the `Ctrl+S` keyboard shortcut.

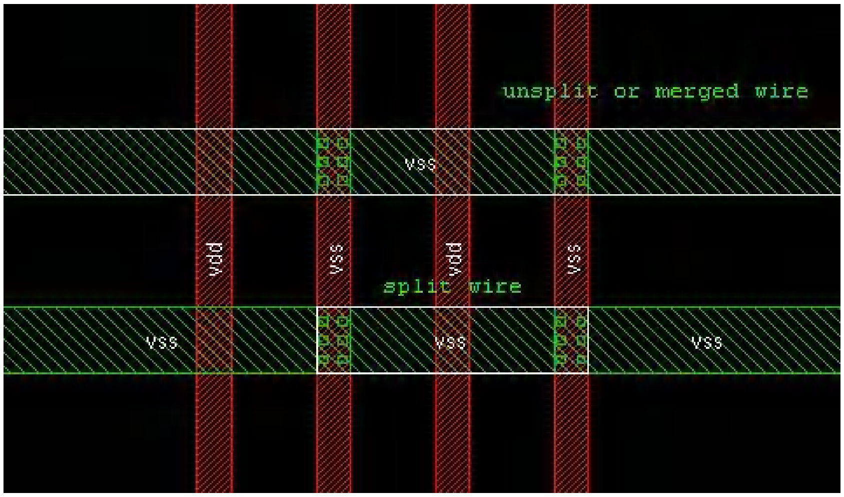
   This automatically splits the single wire segment into multiple segments at points connected to other othogonal wires.

After splitting a wire, you can merge those wire segments that align back into a single segment.

1. Select a single segment.
2. Use the `Shift+M` keyboard shortcut.

   This merges the wire segments into a single segment.

In the following picture, the second horizontal stripe (vss) is split into 3 wire segments due to two crossover points.



Adding Vias

1. Select the *Add Via* widget . The equivalent keyboard shortcut is `O`.
2. Press the `F3` key.

   This displays the Edit Via form.
3. Select *Geometry* in the *Create Via by* field.
4. Fill all of the fields in the form. For more information, see "Edit Via" in Edit Menu chapter of the *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 *Text Command Reference*.

- Using keyboard shortcuts

  You can change one via at a time using keyboard shortcuts.

  a. Place the cursor on the via to be changed in *Auto Query* mode.

  b. Use the `N` (next) or `P` (previous) keyboard shortcuts to select the correct via if multiple vias exist in the same location on different layers.

  c. Without moving the mouse, use the `Shift+N` (next) or `Shift+P`(previous) keyboard shortcut to display a via that has the same LEF rule as the selected via.

     - If a via is available, the display is updated with the new via when you press the keyboard shortcut.

     - If another via is not available, you will hear a warning beep when you press the keyboard shortcut. This can occur when only one via is defined in the LEF file, when the currently queried object is not a via, or when no object is currently queried.

     Note: By default, if the net is routed with a Non Default Rule (NDR), the Wire Editor circles through the list of NDR vias defined in the LEF for that NDR rule *plus* the default vias, each time the *Shift + N* bindkey is used. If you want the Wire Editor to consider only the NDR vias when *Shift + N* is pressed, set the `setEditMode -circle_NDR_vias_only` parameter to `1`. This restricts circle list to NDR vias only.

- Using the Edit Power Vias form

  For information, see Edit Power Vias in the *Innovus Menu Reference*.

Note: You cannot change vias using the Edit Route form.

Moving Vias

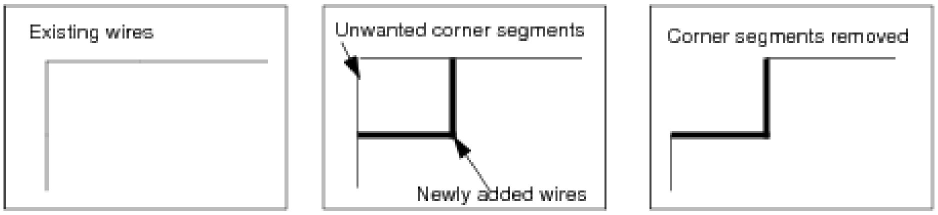
Select vias. Use the Shift+R keyboard shortcut or click the *Move/Resize/Reshape* icon ✥ to switch to *non-connectivity-based move* mode then move the vias. The Innovus software moves vias without considering connectivity.

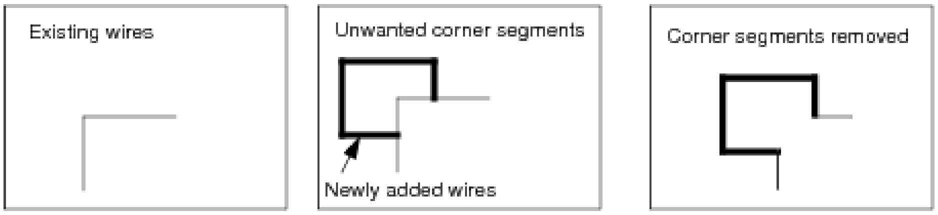
Reshaping Routes

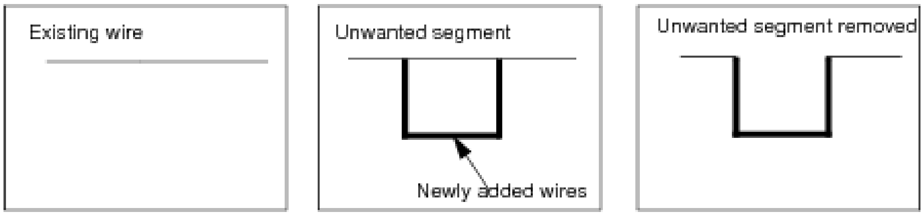
You can reshape routes by specifying that wires at the corner of a route are to be trimmed after adding wires within an area that makes the existing corner wires obsolete. In addition, if you add a wire that circumvents an existing path, the existing route is trimmed after the new wires are added.

1. Click the *Edit Wire* widget .
   This places the Innovus software in the *Edit Wire* mode and changes the mouse pointer to a pencil. In addition, it places the software in the *Auto Query* mode.
   The equivalent keyboard shortcut is `Shift+A`.
2. Choose *Edit - Wire - Edit* from the menu. The Edit Route form opens.
   The equivalent keyboard shortcut is `E`.
3. Click the *Misc* tab.
4. Select the *Reshape* option on the form.
5. Add wires to the design, as described in Adding Wires.

The following illustrations show the results of using the *Reshape* option:







Controlling Cell Blockage Visibility

If you see a spacing violation when adding or editing a via or wire, it might be caused by a cell blockage that is not currently visible.

To see cell blockages, select the *Cell Blkg* option on the *Routing color control* display (click the slidebar to display this option). Alternatively, you can click the *All Colors* button to display the Color Preferences form, then select the *Cell Blockage* visibility option. In addition, depending on whether the blockage is outside or inside a cell, you must do one of the following:

- Cell blockages outside a cell are visible by default. To control the visibility of these blockages for particular layers, click the *Wire Layers* tab of the Color Preferences form. Use the buttons in the fifth column, *Blkg*, to deselect the visibility of blockages for particular layers. By default, all layers are selected.
- Cell blockages within a cell are not visible by default. To see these cell blockages, click the *Wire Layers* tab of the Color Preferences form, then use the buttons in the sixth column, *V Blkg*, to select the visibility of blockages for each cut layer that you want to see. By default, all layers are deselected.

Parallel Editing Capability

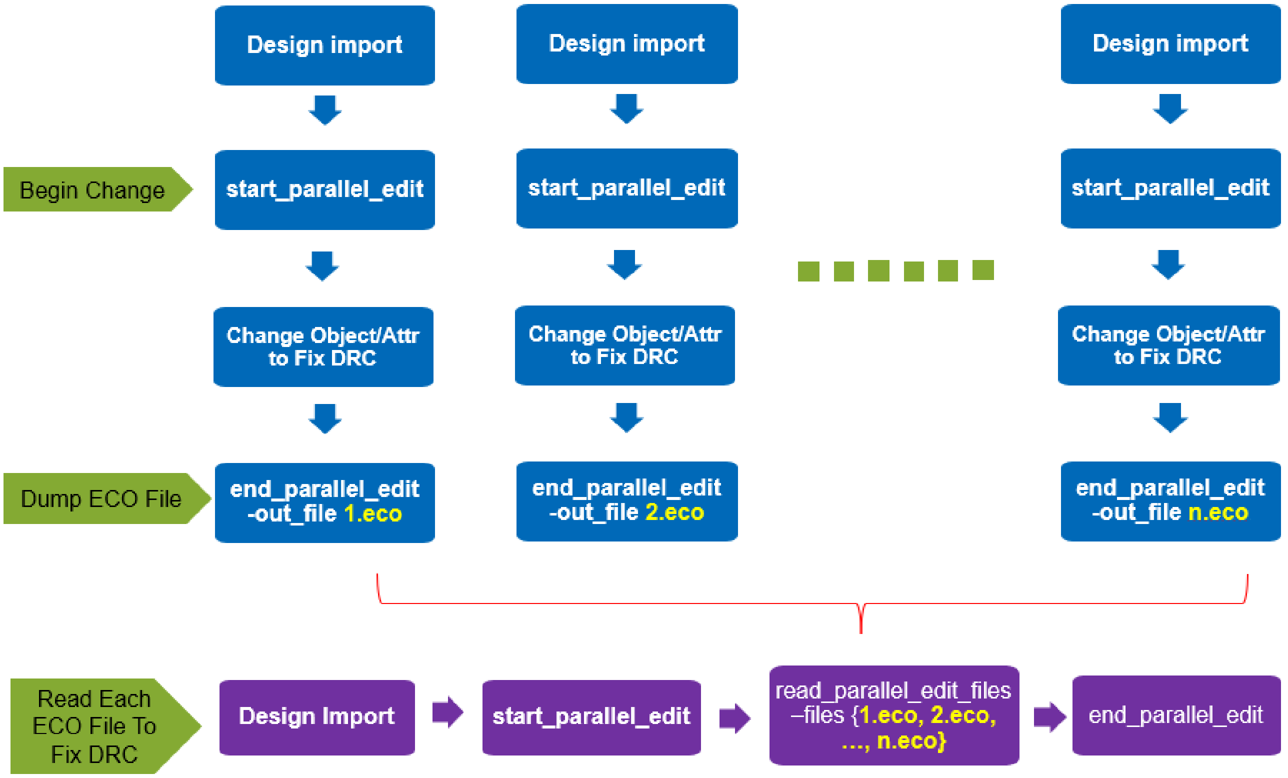
In large designs, it can be challenging to fix all DRC violations by automatic physical implementation tools. The traditional flow in which violations are fixed one at a time by an engineer can consume a lot of time. To address such issues, Innovus provides the parallel editing capability through merge DB. This capability enables multiple users to make wire editing changes in the same database in parallel.
The parallel editing capability can be used to fix DRC violations and ECO in parallel at the sign-off stage in the physical implementation flow.

Parallel Edit Flow

The parallel editing flow can enable multiple users to make physical and logic ECO changes at the same time on different areas of a design. It can support a wide range of basic wire editing and ECO operations, such as modifying or adding a wire or via or modifying logical connections.

This flow consists of three commands: `start_parallel_edit`, `end_parallel_edit`, and `read_parallel_edit_files`. All the physical and logical information modified in the area selected through `start_parallel_edit` is recorded in the related ECO file by `end_parallel_edit`. The `read_parallel_edit_files` command is then used to load all the ECO files to complete the entire ECO operation.

The parallel edit flow can be represented as follows:

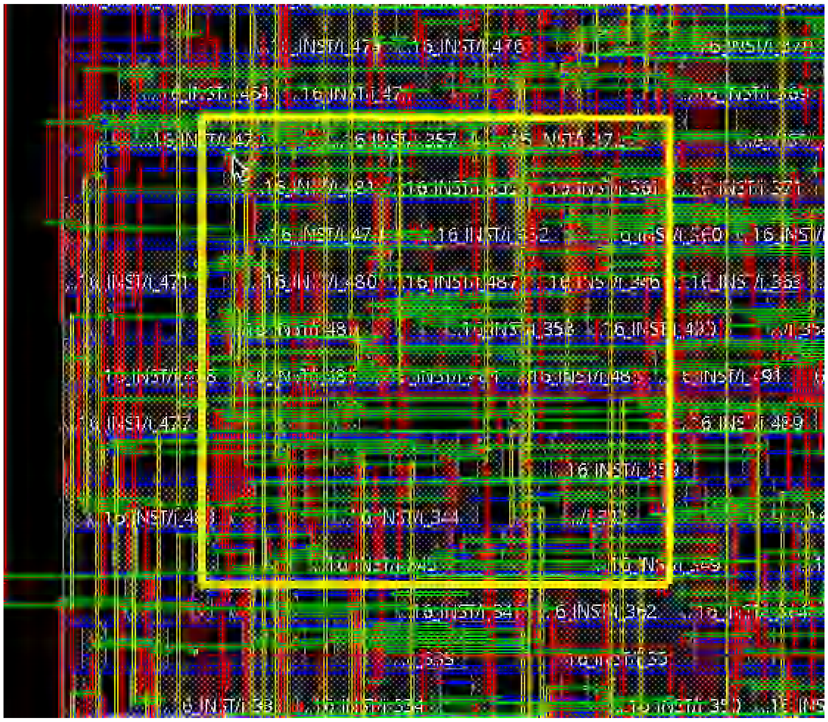

Design import
Design import
Design import
Begin Change
start_parallel_edit
start_parallel_edit
start_parallel_edit
Change Object/Attr to Fix DRC
Change Object/Attr to Fix DRC
Change Object/Attr to Fix DRC
Dump ECO File
end_parallel_edit -out_file 1.eco
end_parallel_edit -out_file 2.eco
end_parallel_edit -out_file n.eco
Read Each ECO File To Fix DRC
Design Import
start_parallel_edit
read_parallel_edit_files –files {1.eco, 2.eco, ..., n.eco}
end_parallel_edit

The steps in the parallel edit flow are detailed below:

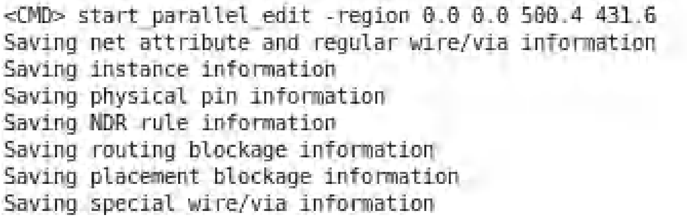
1. Import or restore the design in which you want to make parallel edits along with other users. The other users can start their own Innovus session on the same database in parallel.
2. Initialize parallel editing by using the `start_parallel_edit` command.

   The `start_parallel_edit` command is used to define the area where you need to perform

DRC fixing or ECO operations. The command draws a yellow square on the main window to indicate the edit area and saves the physical data, net attributes, via cell names, and Non-Default Rules (NDRs) to multiple binary files. Use the `-region {`*`x1 y1 x2 y2`*`}` parameter to specify the coordinates of the edit area. Specify the `-area_restricted` parameter to write out only the different objects inside or touching the specified edit area. This is a strict interpretation of the region, and ignores the changes made outside of the region. The other engineers should also initialize `start_parallel_edit` in their sessions separately to specify their own operating areas. Area overlap is not recommended because it might cause ECO conflicts.



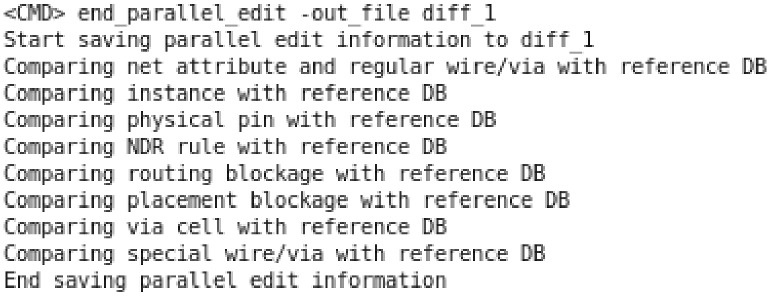
The following example shows the report information generated by `start_parallel_edit`.



```
<CMD> start_parallel_edit -region 0.0 0.0 500.4 431.6
Saving net attribute and regular wire/via information
Saving instance information
Saving physical pin information
Saving NDR rule information
Saving routing blockage information
Saving placement blockage information
Saving special wire/via information
```

3. Make DRC fixes on ECO changes in your assigned area in the design. Multiple engineers can work simultaneously on different areas in the design in the parallel edit mode.

4. After all your changes are done, use the `end_parallel_edit` command in your session to save the ECO file of the edits. The `end_parallel_edit` command compares the data in memory with the original data saved in the binary files by `start_parallel_edit` and writes the parallel edit information to the specified file. The following example shows the report information generated by `end_parallel_edit`.



```
<CMD> end_parallel_edit -out_file diff_1
Start saving parallel edit information to diff_1
Comparing net attribute and regular wire/via with reference DB
Comparing instance with reference DB
Comparing physical pin with reference DB
Comparing NDR rule with reference DB
Comparing routing blockage with reference DB
Comparing placement blockage with reference DB
Comparing via cell with reference DB
Comparing special wire/via with reference DB
End saving parallel edit information
```

   Multiple `end_parallel_edit` commands are allowed after one `start_parallel_edit`. Others users should also issue the `end_parallel_edit` command in their respective sessions after completing their changes. Similarly, other engineers also save their modifications in corresponding files after he corrections for the different areas of the chip are saved in different files.

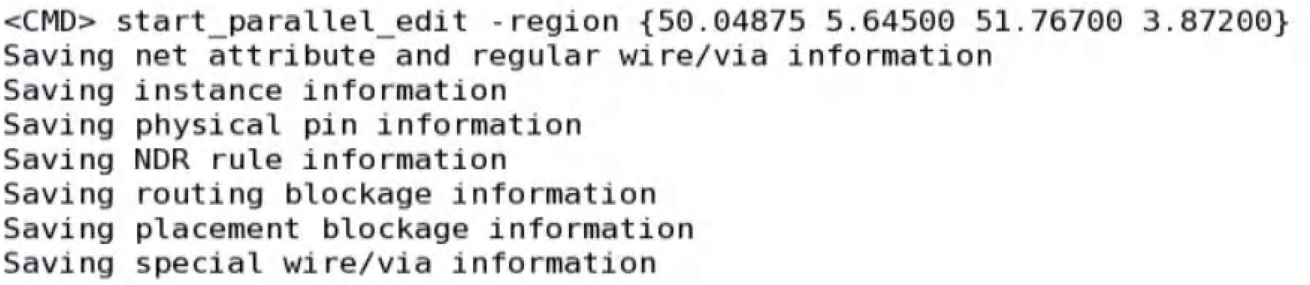
5. Load all parallel edit files by using the `read_parallel_edit_files` command.
   The `read_parallel_edit_files` command loads all the specified parallel edit files and implements the editing changes recorded in them. In case of any conflict in the parallel edit files, the tool implements changes as per the specified conflict mode and writes the conflicting information to a report file.

Parallel Editing Example for DRC Corrections

By using the parallel editing capability, you can divide the entire chip into different areas to enable multiple engineers to correct DRC violations in different areas of the design at the same time. After all the changes are done, you can apply the correction results to the entire chip. The following example shows how DRC corrections are done in a specific area of a real chip in parallel editing mode:

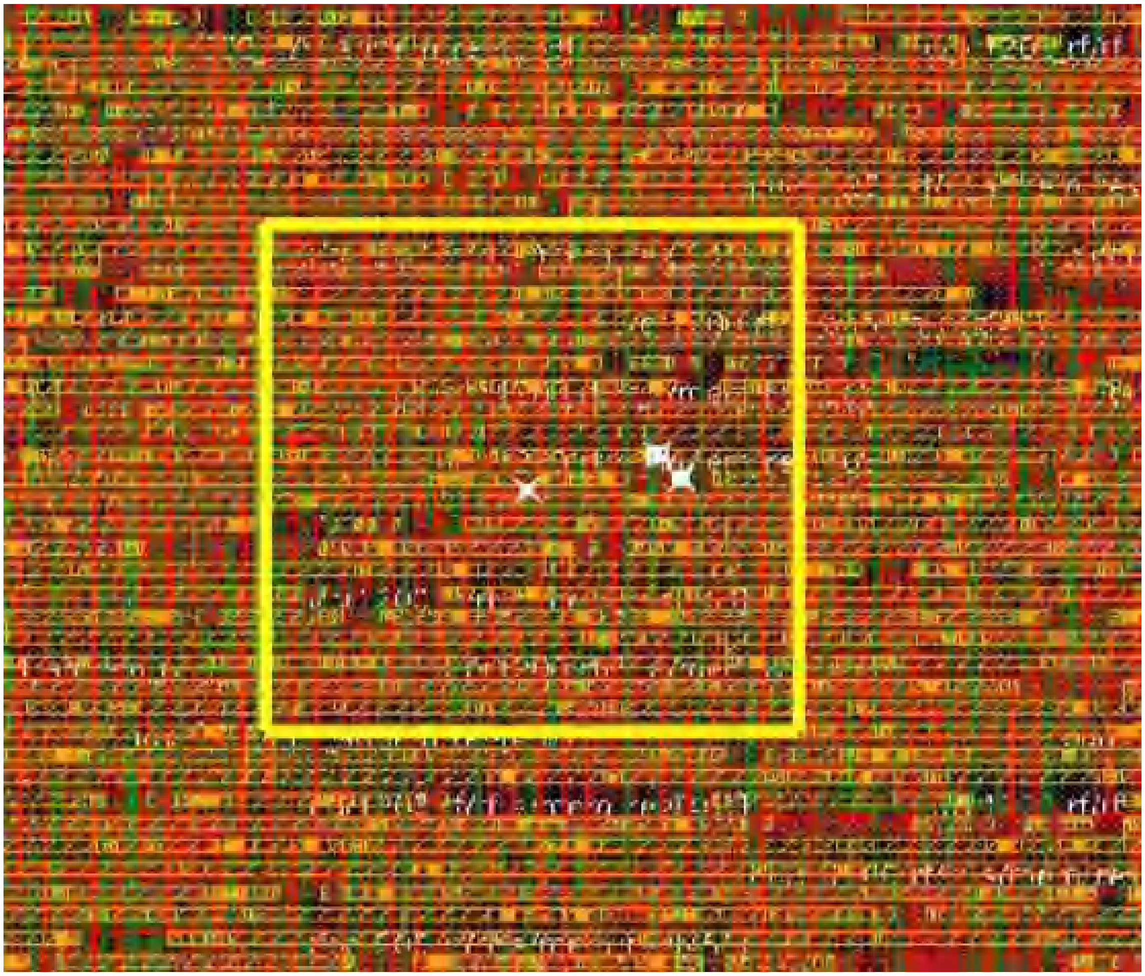
Step 1

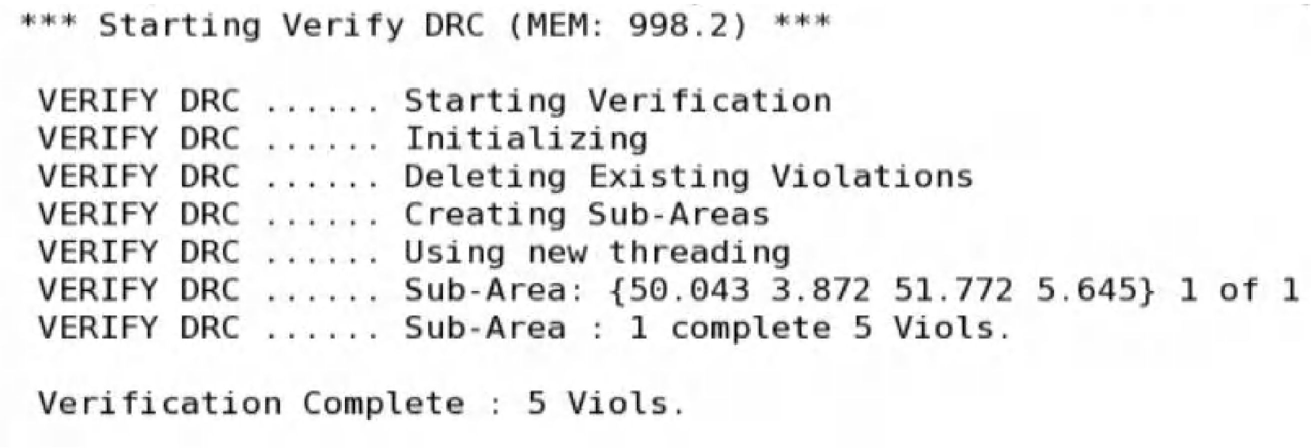
Load the entire chip into Innovus and use `start_parallel_edit -region {`*`x1 y1 x2 y2`*`}` to specify the area in which DRC correction is required.



```
<CMD> start_parallel_edit -region {50.04875 5.64500 51.76700 3.87200}
Saving net attribute and regular wire/via information
Saving instance information
Saving physical pin information
Saving NDR rule information
Saving routing blockage information
Saving placement blockage information
Saving special wire/via information
```

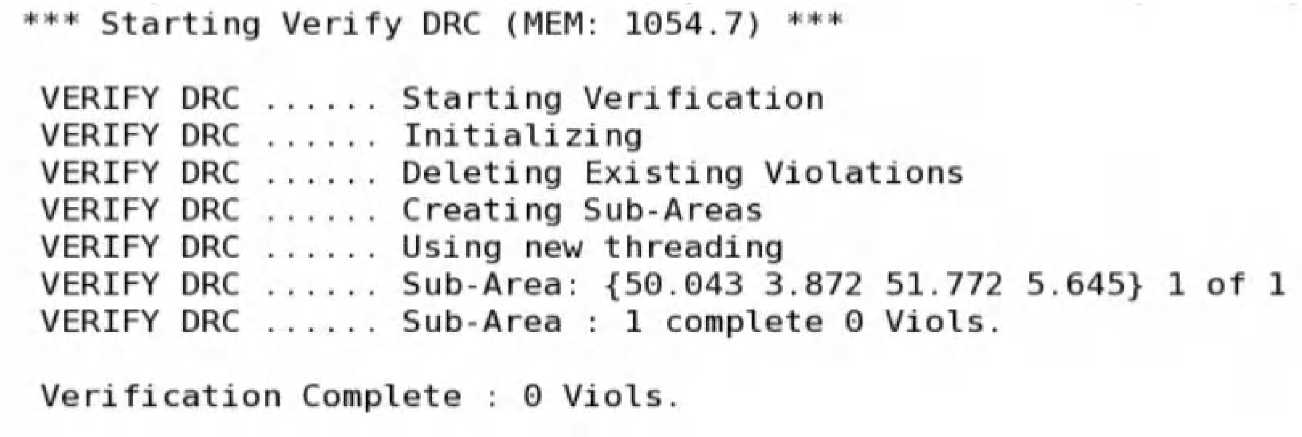
The specified area is highlighted in the GUI at this point. The `verify_drc` command can report all DRC violations in the highlighted area as shown below:





```
*** Starting Verify DRC (MEM: 998.2) ***

  VERIFY DRC ...... Starting Verification
  VERIFY DRC ...... Initializing
  VERIFY DRC ...... Deleting Existing Violations
  VERIFY DRC ...... Creating Sub-Areas
  VERIFY DRC ...... Using new threading
  VERIFY DRC ...... Sub-Area: {50.043 3.872 51.772 5.645} 1 of 1
  VERIFY DRC ...... Sub-Area : 1 complete 5 Viols.

  Verification Complete : 5 Viols.
```

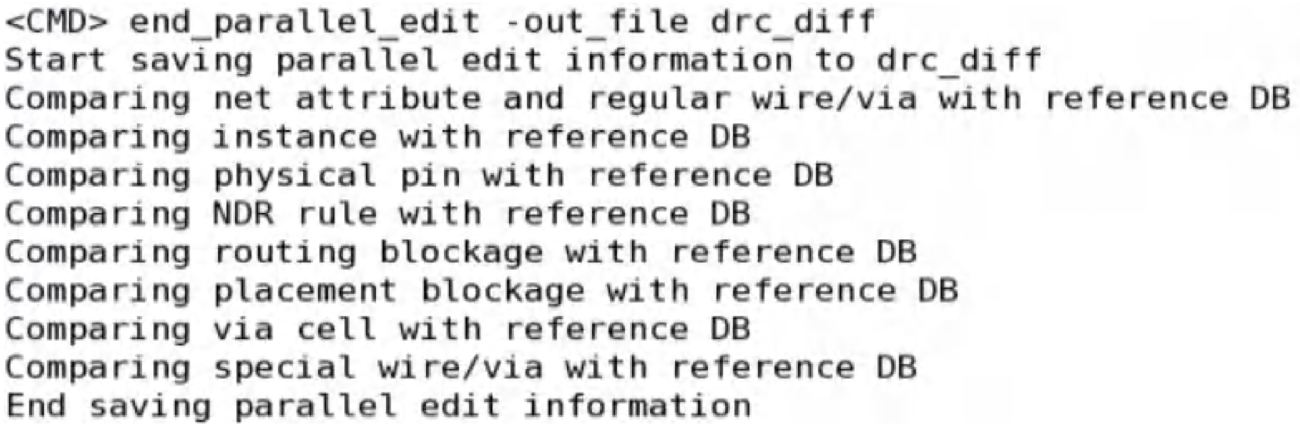
As shown above, there are 5 DRC violations in the defined area. At this point, you can fix the DRC violations manually or through some automated method in the tool. After DRC fixing is complete, use the `verify_drc` command again to check whether or not the DRC violations are completely fixed.



```
*** Starting Verify DRC (MEM: 1054.7) ***

  VERIFY DRC ...... Starting Verification
  VERIFY DRC ...... Initializing
  VERIFY DRC ...... Deleting Existing Violations
  VERIFY DRC ...... Creating Sub-Areas
  VERIFY DRC ...... Using new threading
  VERIFY DRC ...... Sub-Area: {50.043 3.872 51.772 5.645} 1 of 1
  VERIFY DRC ...... Sub-Area : 1 complete 0 Viols.

  Verification Complete : 0 Viols.
```

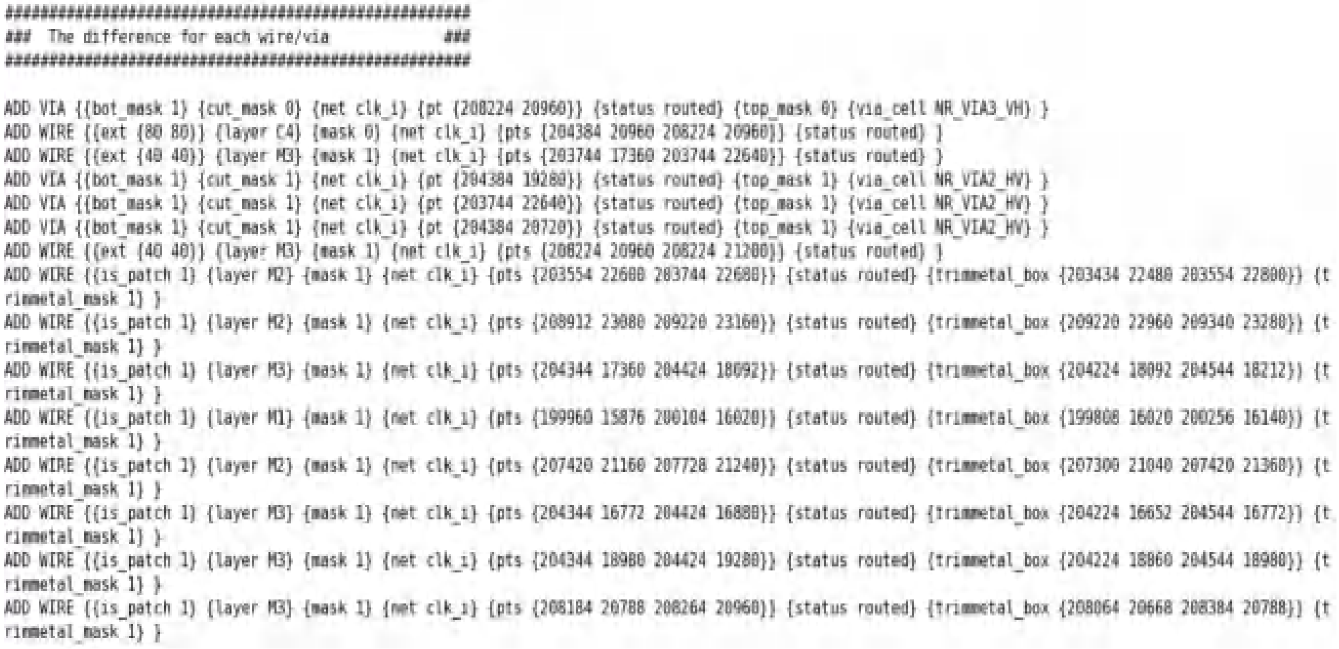
Step 2

After the DRC violations in the specified area are completely fixed, you need to save the changes made to the design. `end_parallel_edit -out_file drc_diff` will save all the changes to the `drc_diff` file as shown below:



```
<CMD> end_parallel_edit -out_file drc_diff
Start saving parallel edit information to drc_diff
Comparing net attribute and regular wire/via with reference DB
Comparing instance with reference DB
Comparing physical pin with reference DB
Comparing NDR rule with reference DB
Comparing routing blockage with reference DB
Comparing placement blockage with reference DB
Comparing via cell with reference DB
Comparing special wire/via with reference DB
End saving parallel edit information
```

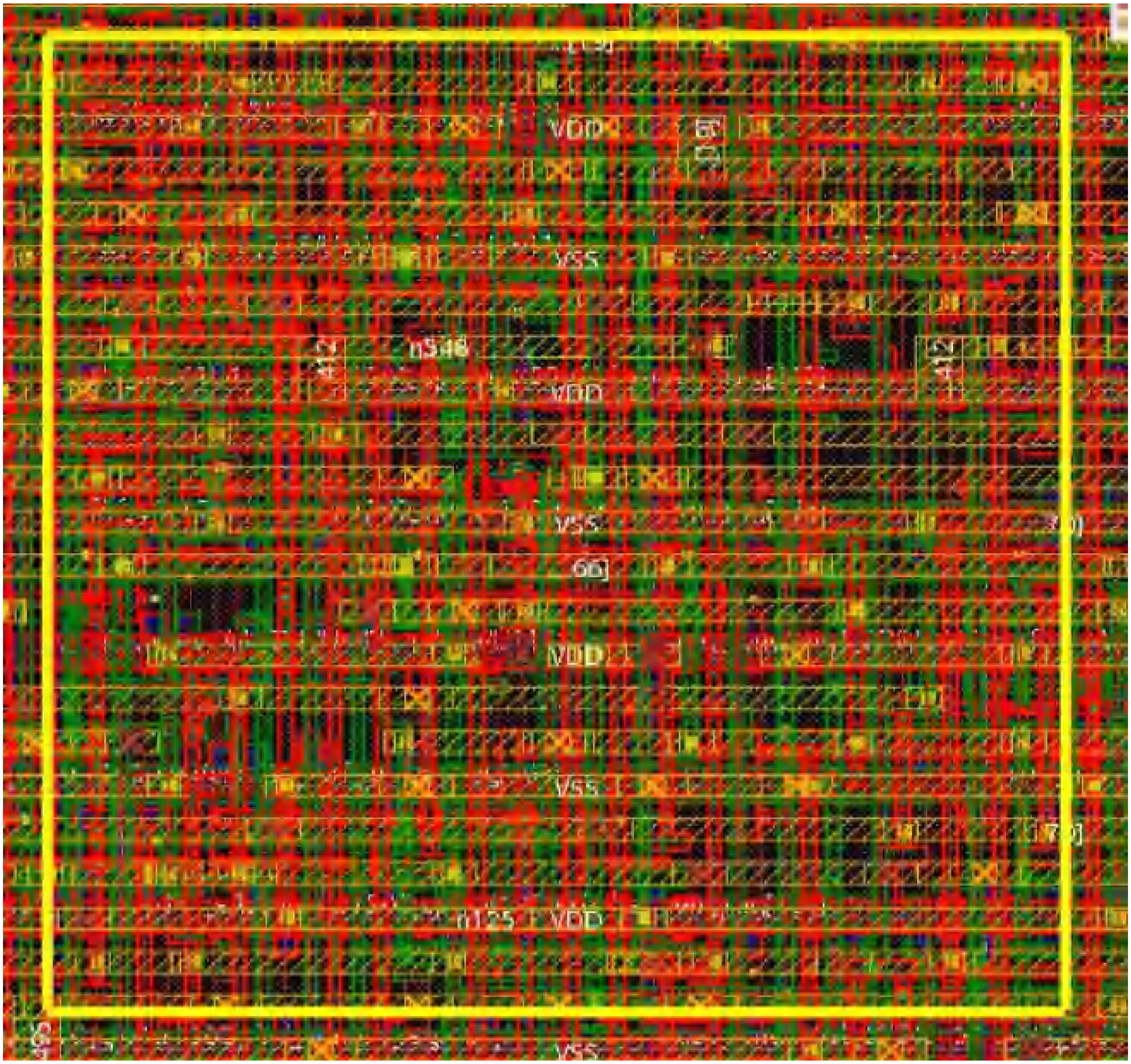
The content of the saved `drc_diff` file is shown below:



```
#######################################################
###  The difference for each wire/via              ###
#######################################################

ADD VIA {{bot_mask 1} {cut_mask 0} {net clk_i} {pt {208224 20960}} {status routed} {top_mask 0} {via_cell NR_VIA3_VH} }
ADD WIRE {{ext {80 80}} {layer C4} {mask 0} {net clk_i} {pts {204384 20960 208224 20960}} {status routed} }
ADD WIRE {{ext {40 40}} {layer M3} {mask 1} {net clk_i} {pts {203744 17360 203744 22640}} {status routed} }
ADD VIA {{bot_mask 1} {cut_mask 1} {net clk_i} {pt {204384 19280}} {status routed} {top_mask 1} {via_cell NR_VIA2_HV} }
ADD VIA {{bot_mask 1} {cut_mask 1} {net clk_i} {pt {203744 22640}} {status routed} {top_mask 1} {via_cell NR_VIA2_HV} }
ADD VIA {{bot_mask 1} {cut_mask 1} {net clk_i} {pt {204384 20720}} {status routed} {top_mask 1} {via_cell NR_VIA2_HV} }
ADD WIRE {{ext {40 40}} {layer M3} {mask 1} {net clk_i} {pts {208224 20960 208224 21200}} {status routed} }
ADD WIRE {{is_patch 1} {layer M2} {mask 1} {net clk_i} {pts {203554 22680 283744 22680}} {status routed} {trimmetal_box {203434 22480 283554 22800}} {t
rimmetal_mask 1} }
ADD WIRE {{is_patch 1} {layer M2} {mask 1} {net clk_i} {pts {208912 23080 209220 23160}} {status routed} {trimmetal_box {209220 22960 209340 23280}} {t
rimmetal_mask 1} }
ADD WIRE {{is_patch 1} {layer M3} {mask 1} {net clk_i} {pts {204344 17360 204424 18092}} {status routed} {trimmetal_box {204224 18092 204544 18212}} {t
rimmetal_mask 1} }
ADD WIRE {{is_patch 1} {layer M1} {mask 1} {net clk_i} {pts {199960 15876 200104 16020}} {status routed} {trimmetal_box {199808 16020 200256 16140}} {t
rimmetal_mask 1} }
ADD WIRE {{is_patch 1} {layer M2} {mask 1} {net clk_i} {pts {207420 21160 207728 21240}} {status routed} {trimmetal_box {207300 21040 207420 21360}} {t
rimmetal_mask 1} }
ADD WIRE {{is_patch 1} {layer M3} {mask 1} {net clk_i} {pts {204344 16772 204424 16880}} {status routed} {trimmetal_box {204224 16652 204544 16772}} {t
rimmetal_mask 1} }
ADD WIRE {{is_patch 1} {layer M3} {mask 1} {net clk_i} {pts {204344 18980 204424 19280}} {status routed} {trimmetal_box {204224 18860 204544 18980}} {t
rimmetal_mask 1} }
ADD WIRE {{is_patch 1} {layer M3} {mask 1} {net clk_i} {pts {208184 20788 208264 20960}} {status routed} {trimmetal_box {208064 20668 208384 20788}} {t
rimmetal_mask 1} }
```

Step 3

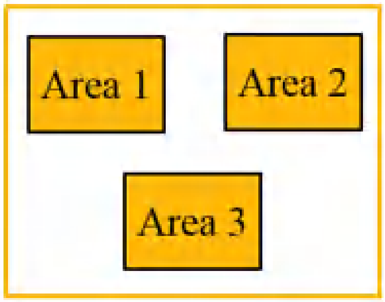
Reload the original chip, and use `read_parallel_edit_files -files drc_diff` to read in the corrections from the previous step. The DRC corrections will be applied to the original chip layout so that DRC violations will be removed from the original area as shown below.



The process described above can also be applied to a multi-user collaboration. The only additional step you need to do is to divide the whole chip into specific areas in the first step and assign them to different engineers.

Parallel Editing Example for ECO Changes

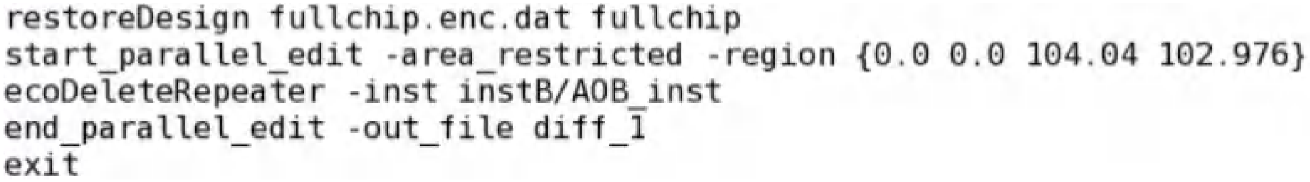
In this example, the chip is divided into three areas as shown below.



Step 1

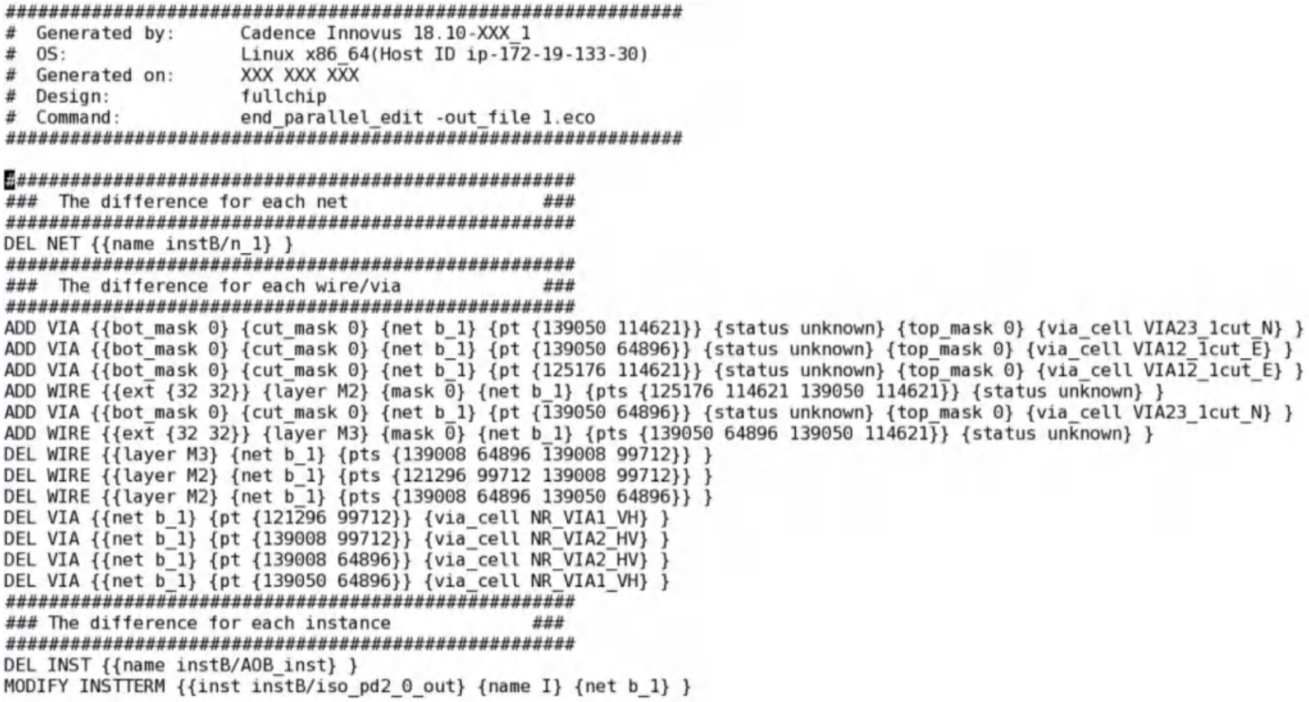
Load the entire chip into Innovus and use `start_parallel_edit -region {Area1_coordinates}` to specify the area in which ECO changes are required. After the relevant operation is completed, use the `end_parallel_edit -out_file 1.eco` command to record the changes to the chip in the `1.eco` file. The specific example script and ECO file are as follows:

Example script



```
restoreDesign fullchip.enc.dat fullchip
start_parallel_edit -area_restricted -region {0.0 0.0 104.04 102.976}
ecoDeleteRepeater -inst instB/AOB_inst
end_parallel_edit -out_file diff_1
exit
```

Example ECO File

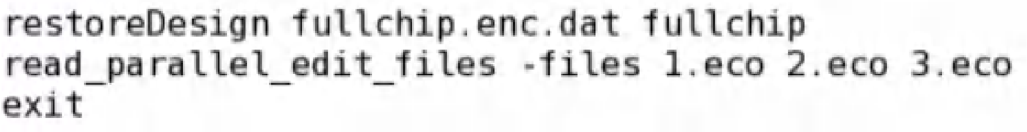


```
################################################################
#  Generated by:      Cadence Innovus 18.10-XXX_1
#  OS:                Linux x86_64(Host ID ip-172-19-133-30)
#  Generated on:      XXX XXX XXX
#  Design:            fullchip
#  Command:           end_parallel_edit -out_file 1.eco
################################################################

######################################################
###  The difference for each net                  ###
######################################################
DEL NET {{name instB/n_1} }
######################################################
###  The difference for each wire/via             ###
######################################################
ADD VIA {{bot_mask 0} {cut_mask 0} {net b_1} {pt {139050 114621}} {status unknown} {top_mask 0} {via_cell VIA23_1cut_N} }
ADD VIA {{bot_mask 0} {cut_mask 0} {net b_1} {pt {139050 64896}} {status unknown} {top_mask 0} {via_cell VIA12_1cut_E} }
ADD VIA {{bot_mask 0} {cut_mask 0} {net b_1} {pt {125176 114621}} {status unknown} {top_mask 0} {via_cell VIA12_1cut_E} }
ADD WIRE {{ext {32 32}} {layer M2} {mask 0} {net b_1} {pts {125176 114621 139050 114621}} {status unknown} }
ADD VIA {{bot_mask 0} {cut_mask 0} {net b_1} {pt {139050 64896}} {status unknown} {top_mask 0} {via_cell VIA23_1cut_N} }
ADD WIRE {{ext {32 32}} {layer M3} {mask 0} {net b_1} {pts {139050 64896 139050 114621}} {status unknown} }
DEL WIRE {{layer M3} {net b_1} {pts {139008 64896 139008 99712}} }
DEL WIRE {{layer M2} {net b_1} {pts {121296 99712 139008 99712}} }
DEL WIRE {{layer M2} {net b_1} {pts {139008 64896 139050 64896}} }
DEL VIA {{net b_1} {pt {121296 99712}} {via_cell NR_VIA1_VH} }
DEL VIA {{net b_1} {pt {139008 99712}} {via_cell NR_VIA2_HV} }
DEL VIA {{net b_1} {pt {139008 64896}} {via_cell NR_VIA2_HV} }
DEL VIA {{net b_1} {pt {139050 64896}} {via_cell NR_VIA1_VH} }
######################################################
### The difference for each instance             ###
######################################################
DEL INST {{name instB/AOB_inst} }
MODIFY INSTTERM {{inst instB/iso_pd2_0_out} {name I} {net b_1} }
```

In the above step, ECO changes were made in `Area1` in the parallel editing mode. Similarly, ECO changes are made in `Area2` and `Area3`, and then `end_parallel_edit` is used to write the modified physical and logical information in these areas to the `2.eco` and 3.eco files, respectively.

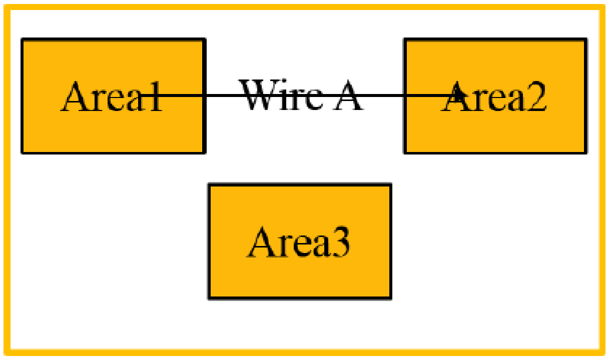
Step 2

Load the entire chip first and then read the eco files generated in the previous step by using the `read_parallel_edit_files` command. All the changes in different areas will be reflected in the chip. The script example for this step is as follows:



```
restoreDesign fullchip.enc.dat fullchip
read_parallel_edit_files -files 1.eco 2.eco 3.eco
exit
```

Sometimes, there may be a conflict between the changes in two eco files. In such cases, `read_parallel_edit_files` will follow the first eco file and will issue a warning message when a conflicting eco file is read in. For example, Wire A belongs to two different areas in the design below: `Area1` and `Area2`.



Suppose, both users have corrected this wire. Assuming that `1.eco` and `2.eco` are saved at the end of the first step, then changes to Wire A in `Area1` are ignored when the following command is run:

```
read_parallel_edit_files -files 2.eco 1.eco
```

For more details on handling conflict, see the Handling Conflicts in Parallel Edit Files section.

Parallel Edit File Content

You can record changes to the following objects/attributes in a parallel edit file:

Object Name	Attribute Name	ACTION		
		add	delete	modify
Net	Name	y	y	y

	dont_touch			y
	rule			y
	skip_routing			y
	weight			y
	bottom_preferred_routing_layer			y
	top_preferred_routing_layer			y
	shield_net			y
Wire	net	y	y	y
	ext	y		
	pts	y	y	y
	layer	y	y	y
	mask	y		y
	rule	y		
	status	y		y
	is_patch	y		y
	trimmetal_box	y		y
	trimmetal_mask	y		y
sWire	net	y	y	y
	box	y	y	y
	layer	y	y	y
	mask	y		y
	status	y		y
	shape	y		y
	shield_net	y		

	type	y		
	width	y		
	subclass	y		
	style_id	y		
	poly	y	y	y
	is_patch	y		y
	trimmetal_box	y		y
	trimmetal_mask			y
	dont_touch	y		
Via Cell	name	y		
	rule	y		
	top_layer	y		
	cut_layer	y		
	bot_layer	y		
	is_colored	y		
	p_cell	y		
	cut_pattern	y		
	type	y		
	is_regular	y		
	contact_id	y		
	cut_width	y		
	cut_height	y		
	x_pitch	y		
	y_pitch	y		

	x_times	y		
	y_times	y		
	cut_box	y		
	top_rects	y		
	bot_rects	y		
	cut_rects	y		
	mask_array	y		
Via	net	y	y	y
	pt	y	y	y
	status	y		y
	via_cell	y	y	y
	top_mask	y		y
	cut_mask	y		y
	bot_mask	y		y
sVia	net	y	y	y
	pt	y	y	y
	status	y		y
	orient	y		
	via_cell	y	y	y
	dont_touch	y		
	shape	y		y
	shield_net	y		
	top_mask	y		y
	cut_mask	y		y

	bot_mask	y		y
	subclass	y		
NDR	name	y		y
	hard_spacing	y		y
	layer	y		y
	min_cut	y		y
	via_cell	y		y
	via_rule	y		y
	via_cut_class	y		y
Instance	name	y	y	y
	cell	y	y	
	pt	y	y	
	is_physical_only	y	y	
	place_status	y	y	
	orient	y	y	
	mask_shift	y	y	
Instance Term	name			y
	inst			y
	net			y
Pin	name	y	y	y
	net	y	y	
	is_special	y		
	direction	y		
	type	y		

	place_status	y	y	
	port	y		
Pin Shape	name	y	y	
	layer	y	y	
	mask	y		
	box	y	y	
	poly	y	y	
Routing Blockage	layer	y	y	y
	name	y		y
	mask	y		y
	slots	y		y
	fills	y		y
	pushdown	y		y
	except_pgnet	y		y
	pgnet_only	y		y
	inst	y		y
	spacing	y		y
	width	y		y
	box	y	y	y
	poly	y	y	y
Placment Blockage	box	y	y	y
	pushdown	y		y
	inst	y		y
	name	y		y

	type	y		y
	density	y		y
Area	box			y

Parallel Edit File Format

Each entry in the parallel edit file has the following format:

```
<action command> <object type> <{{attribute value} ...}>
```

`action command` specifies the type of action to be carried out on the given object-attributes combination. Three types of action commands are supported: `ADD`, `DEL`, and `MODIFY`.

For example:

```
MODIFY NET {{Attribute1 Value1} {Attribute2 Value2} ...}
```

Sample Parallel Edit File

Here's an extract from a sample parallel edit file:

```
###############################################################

# Generated by: Cadence Innovus 17.10-d111_1

# OS: Linux x86_64(Host ID noi-leenap)

# Generated on: Wed Mar 8 16:46:37 2017

# Design: design_top

###############################################################

#####################################################
### The difference for each wire/via ###
#####################################################

MODIFY WIRE {{is_patch 1} {layer M1} {mask 1} {net clk_i} {pts {10904 275220 11048
275300}} {status routed} {trimmetal_box {10752 275300 11200 275420}} {trimmetal_mask 1}
}
MODIFY WIRE {{is_patch 1} {layer M1} {mask 1} {net clk_i} {pts {18968 279060 19112
279140}} {status routed} {trimmetal_box {18816 279140 19264 279260}} {trimmetal_mask 1}
}
MODIFY WIRE {{is_patch 1} {layer M1} {mask 1} {net clk_i} {pts {18968 278300 19112
278380}} {status routed} {trimmetal_box {18816 278180 19264 278300}} {trimmetal_mask 1}
}
MODIFY WIRE {{is_patch 1} {layer M1} {mask 1} {net clk_i} {pts {14488 274260 14632
274340}} {status routed} {trimmetal_box {14336 274340 14784 274460}} {trimmetal_mask 1}
}
MODIFY WIRE {{is_patch 1} {layer M1} {mask 1} {net clk_i} {pts {6872 137940 7016
138020}} {status routed} {trimmetal_box {6720 138020 7168 138140}} {trimmetal_mask 1} }
...
....
```

Handling Conflicts in Parallel Edit Files

During parallel editing, there is a possibility of a conflict if the same attribute of an object is changed in different parallel edit files.

Some cases in which a conflict can occur are listed below:

- Two users edit the layer of the same wire segment.
- Two users edit the width of the same NDR rule.
- One user cuts the wire segment, and another user changes the status of the same wire.

In the following examples of parallel edits, there is no conflict:

- One user edits the layer of wire, and another user edits its mask.
- One user edits the width of an NDR rule, and another user edits its spacing.

Whenever a conflict occurs in parallel editing, the way the conflict is addressed depends on the `read_parallel_edit_files -conflict_mode` setting, if specified. The two possible settings for this parameter are:

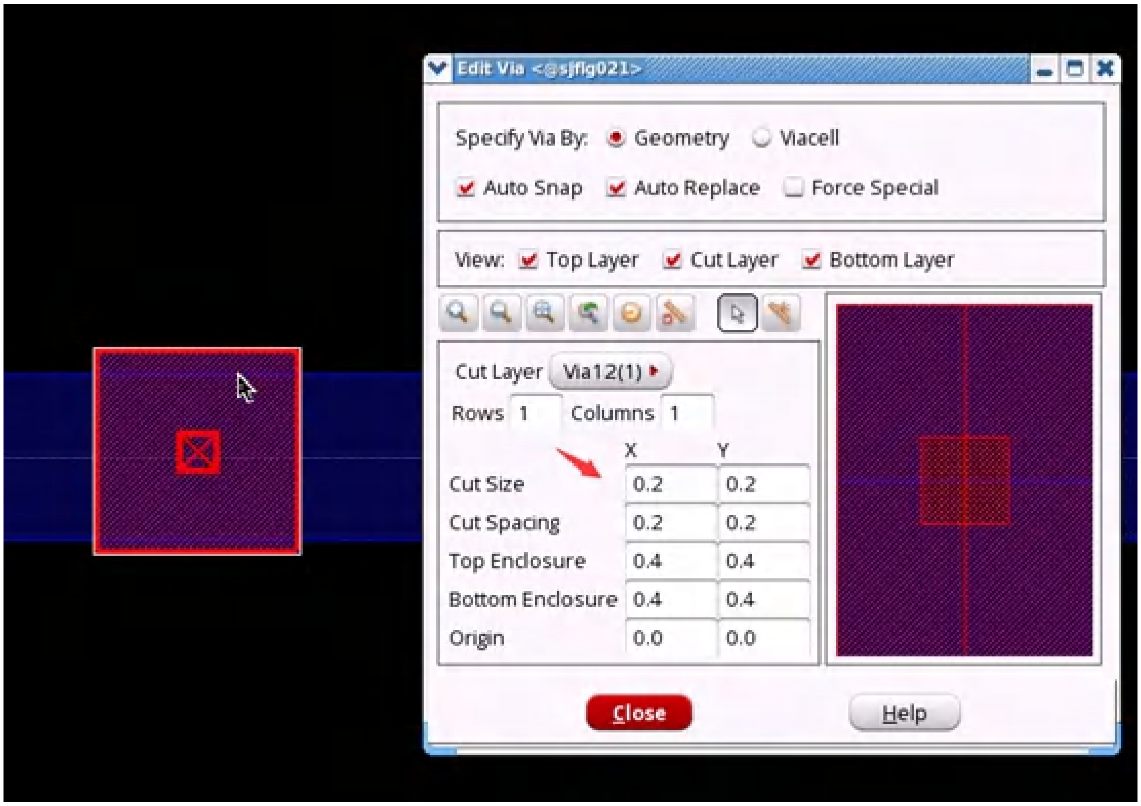
- `only_one` - In case of conflicting commands, one of the command is implemented and the other command is recorded in the conflict report.
- `stop_all` - In case of conflicting commands, the tool implements neither of the conflicting commands and records both of them in the conflict report.

Consider the following examples.

Example 1

Two parallel editing files, `pEdit_1.txt` and `pEdit_2.txt`, have have different definitions for the same via cell *`V12_1X1`*:

Via Cell *V12_1x1* in `pEdit_1.txt`



Specify Via By: ● Geometry ○ Viacell

☑ Auto Snap ☑ Auto Replace □ Force Special

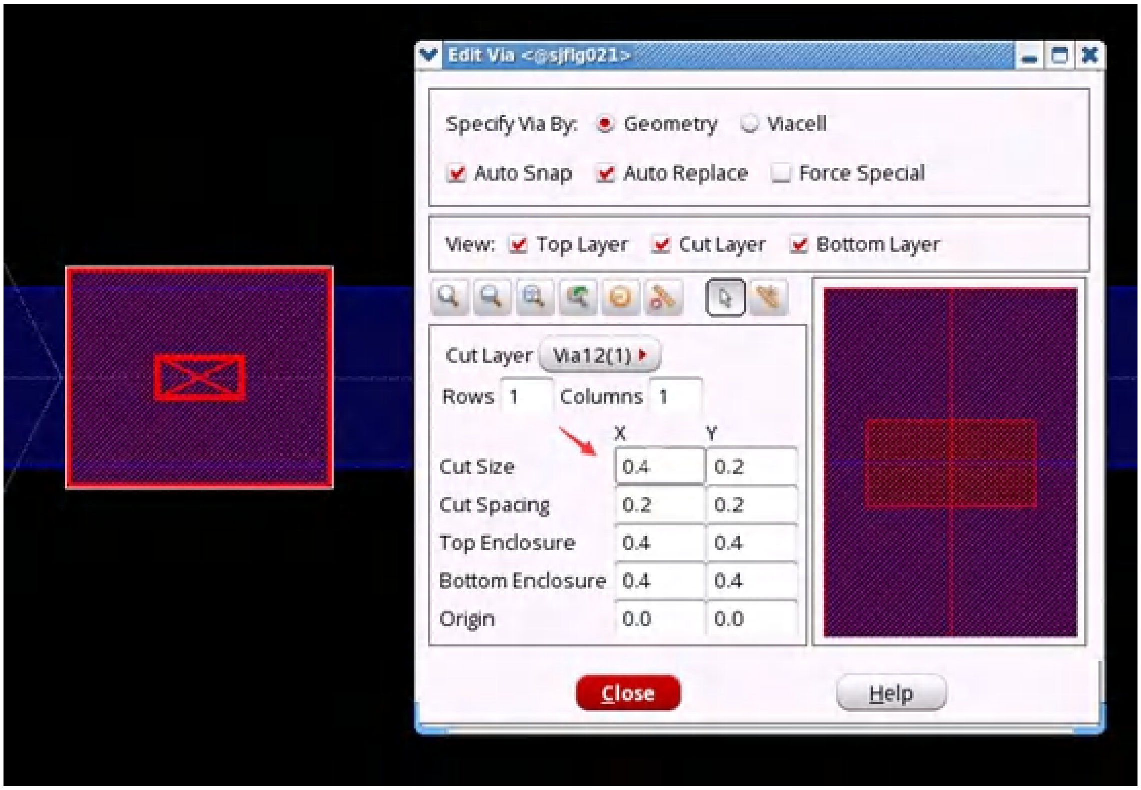
View: ☑ Top Layer ☑ Cut Layer ☑ Bottom Layer

Cut Layer Via12(1)

Rows 1 Columns 1

	X	Y
Cut Size	0.2	0.2
Cut Spacing	0.2	0.2
Top Enclosure	0.4	0.4
Bottom Enclosure	0.4	0.4
Origin	0.0	0.0

Via Cell *V12_1x1* in `pEdit_2.txt`



Edit Via <@sjflg021>

Specify Via By: ● Geometry ○ Viacell

☑ Auto Snap ☑ Auto Replace □ Force Special

View: ☑ Top Layer ☑ Cut Layer ☑ Bottom Layer

Cut Layer Via12(1)

Rows 1 Columns 1

	X	Y
Cut Size	0.4	0.2
Cut Spacing	0.2	0.2
Top Enclosure	0.4	0.4
Bottom Enclosure	0.4	0.4
Origin	0.0	0.0

When both files are read, the tool will keep the via cell name in `pEdit_1.txt` and rename the via cell name to *`V12_1x1_1`* in `pEdit_2.txt`. The tool will also print the following message on the console and record it in the conflict report file:

```
Warning: Rename the via cell name fromV12_1x1 to V12_1x1_1 in file pEdit_2.txt.
```

Example 2

Two parallel editing files, `pe2` and `pe3`, modify the same attribute of a wire.

If the `read_parallel_edit_files -conflict_mode` parameter is specified and set to `only_one` when reading the files, the tool implements the changes in `pe2` but gives the following error:

```
**ERROR: (IMPDBPE-104): 2 files 'pe2:17, pe3:19' modify the same wire. Only command in
file pe2 will be executed. Following commands didn't get executed:
pe3:15
pe3:16
pe3:14
pe3:19
pe3:13
pe3:17
pe3:18
```

If the `read_parallel_edit_files -conflict mode` parameter is set to `stop_all`, the tool does not implement either of the conflicting commands and records both of them in the conflict report in *`File_name`*`:`*`line_num`* format:

```
**ERROR: (IMPDBPE-105): 2 files 'pe2:17, pe3:19' modify the same wire. Following
commands didn't get executed:
pe2:16
pe2:17
pe2:18
pe2:13
pe2:14
pe2:19
pe2:15
pe3:13
pe3:17
pe3:15
pe3:16
pe3:19
pe3:18
pe3:14
```

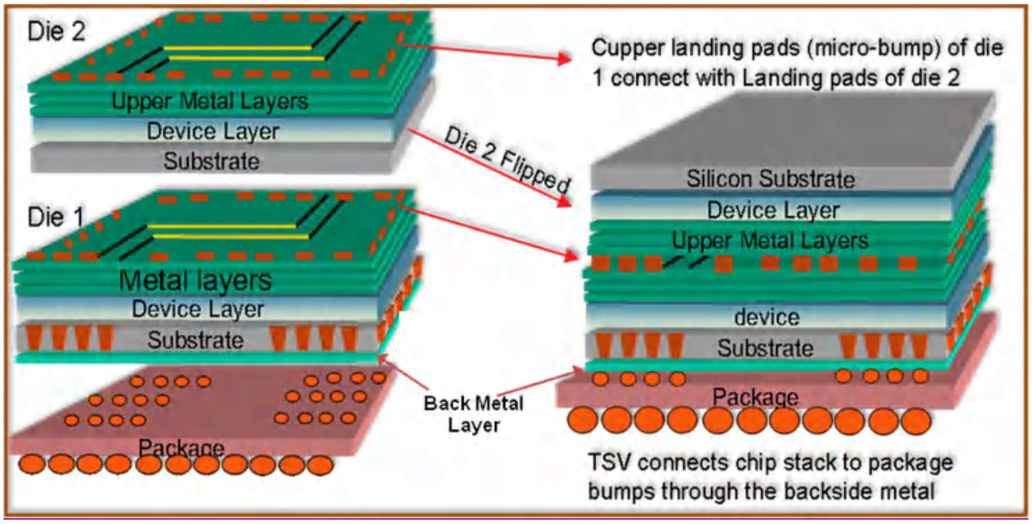
11

Design Methodology for 3D IC with Through Silicon Via

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 front-side metal and the back side RDL. Finally, when the dies are stacked, the aligned bumps between them constitute the data path from one die to the other.

Figure 1 Scheme of 3D IC Profile



Innovus supports TSV designs. In Innovus, all the dies of the 3D ICs are divided into several tiers. Each tier contains several dies, and each die can be flipped, rotated and with offset related to the package. With Innovus, the designers are able to specify the multi-die system configuration (the interconnection between dies, the related position of each die), manipulate TSVs and bumps, perform co-design, and sync-up the interface among all the dies.

Related Topics

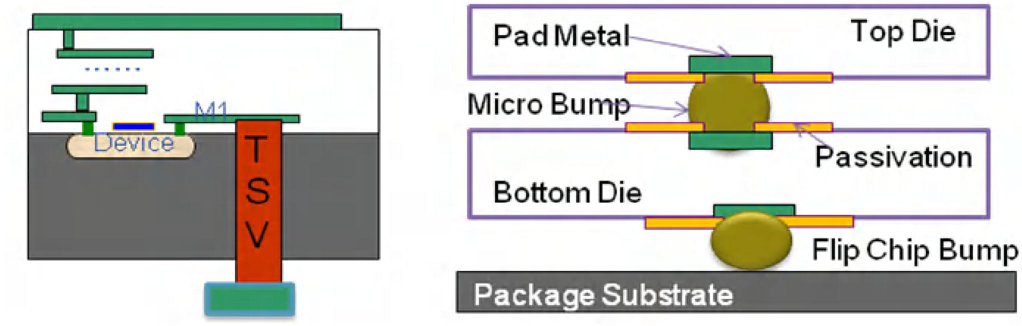
- "Through Silicon Via Design Commands" chapter of the *Innovus Text Command Reference*.
- "TSV Tool Box" section of the Tools Menu chapter in the *Innovus Menu Reference*.

TSV/Bump/Back Side Metal Modeling in Innovus

Back Side Metal (MB), TSV, and Micro bump are introduced to establish 3D stacked interconnection between dies. MB is the Redistribution Layer on the back side of the substrate. TSV is the via which penetrates the silicon substrate. The top cap layer of TSV is usually the first normal routing layer (1) and the bottom cap layer of TSV is MB, therefore, the back side metal and the 1st metal layer is connected through TSV.

To connect the die with the others, some solder balls (or pillars) are placed on the top metal layer or back side metal layer. These solder balls and the underneath metal pad are called bumps in Innovus. The aligned bump between dies is called micro bump or landing pad. The cross die signal or power goes to/comes from the adjacent die through the micro bump. The bump between the die and the package substrate is called flip chip bump.

Figure 2 TSV and Bump Cross-Section



TSV and Bump Cross-Section

All the physical information for back side metal, TSV, and bump is defined in the LEF file (the version for the LEF file should be LEF 57 or later). MB is modeled as ROUTING layer before the first normal metal, and a LEF property "BACKSIDE" is assigned to MB. TSV is model as a CUT layer with a LEF property "TYPE TSV". Below is an example of the additional information in LEF file for TSV and back side metal definition.

The pad metal of the bump is also described in the LEF file. In the LEF file, the bump is modeled as a cell. The bump cell has a pin which has the same shape and layer with the pad metal of the bump. The solder ball information is not described in the LEF. The other way to categorize the bump is based on the bump pad layer. If a bump is on top metal layer, it could be called as front bump, and if a bump is on back side metal layer it is back bump. Micro Bump could be either front bump or back bump depending how the die is stacked.

Example

The statement in LEF to describe the TSV is given below:

```
PROPERTYDEFINITIONS

    LAYER LEF58_BACKSIDE STRING ;

    LAYER LEF58_TYPE STRING ;

 …

END PROPERTYDEFINITIONS

 LAYER MB     # Back side metal layer

 TYPE ROUTING ;

 PROPERTY LEF58_BACKSIDE "BACKSIDE ; " ;

 …

END MB


LAYER TSV     # TSV cut layer between 1 & MB

 TYPE CUT ;

 PROPERTY LEF58_TYPE "TYPE TSV ; " ;

 SPACING 20.0 LAYER OVERLAP ;

 …

END TSV

 LAYER METAL1     # 1

 TYPE ROUTING ;

    …

END METAL1

 …

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 METAL1 ;

     RECT -7.0 -7.0 7.0 7.0 ;

END VIAB1
```

Defining Keep Out Area in Hard Macros

You can define keep-out area constraints inside macros to avoid overlap between bumps/adjacent die edges and these areas. Innovus will report warning if creating bumps inside the keep-out area, and this keep-out area information can be passed to the adjacent die.

Blockage area can be specified in hard macros by creating the following layers of OBS in the macro LEF file:

- Passivation layer in front side
- Passivation layer in back side
- Master slice layer for top die
- Master slice layer for bottom die

Use the passivation layers in the LEF file for defining bump keep-out area, and Master Slice layers in LEF for defining die edge checking. The statement in the LEF file to define blockage for die edge checking and bump keep-out area is given below:

Layer for back bump keep out region

```
LAYER BACKPASSIV
    TYPE CUT ;
    PROPERTY LEF58_TYPE "TYPE PASSIVATION ; " ;
    PROPERTY LEF58_BACKSIDE "BACKSIDE ; " ;
END BACKPASSIV
```

Layer for front bump keep out region

```
LAYER TOPPASSIV
    TYPE CUT ;
    PROPERTY LEF58_TYPE "TYPE PASSIVATION ; " ;
END TOPPASSIV
```

A new layer for top die edge

```
LAYER TOPDIE
    TYPE MASTERSLICE ;
    PROPERTY LEF58_TYPE "TYPE ABOVEDIEEDGE ; " ;
END TOPDIE
```

A new layer for bottom die edge

```
LAYER BOTTOMDIE
    TYPE MASTERSLICE ;
    PROPERTY LEF58_TYPE "TYPE BELOWDIEEDGE ; " ;
END BOTTOMDIE
```

Then, define the OBS layers in the macro for these constraints. The sample syntax for OBS layer definition is given below:

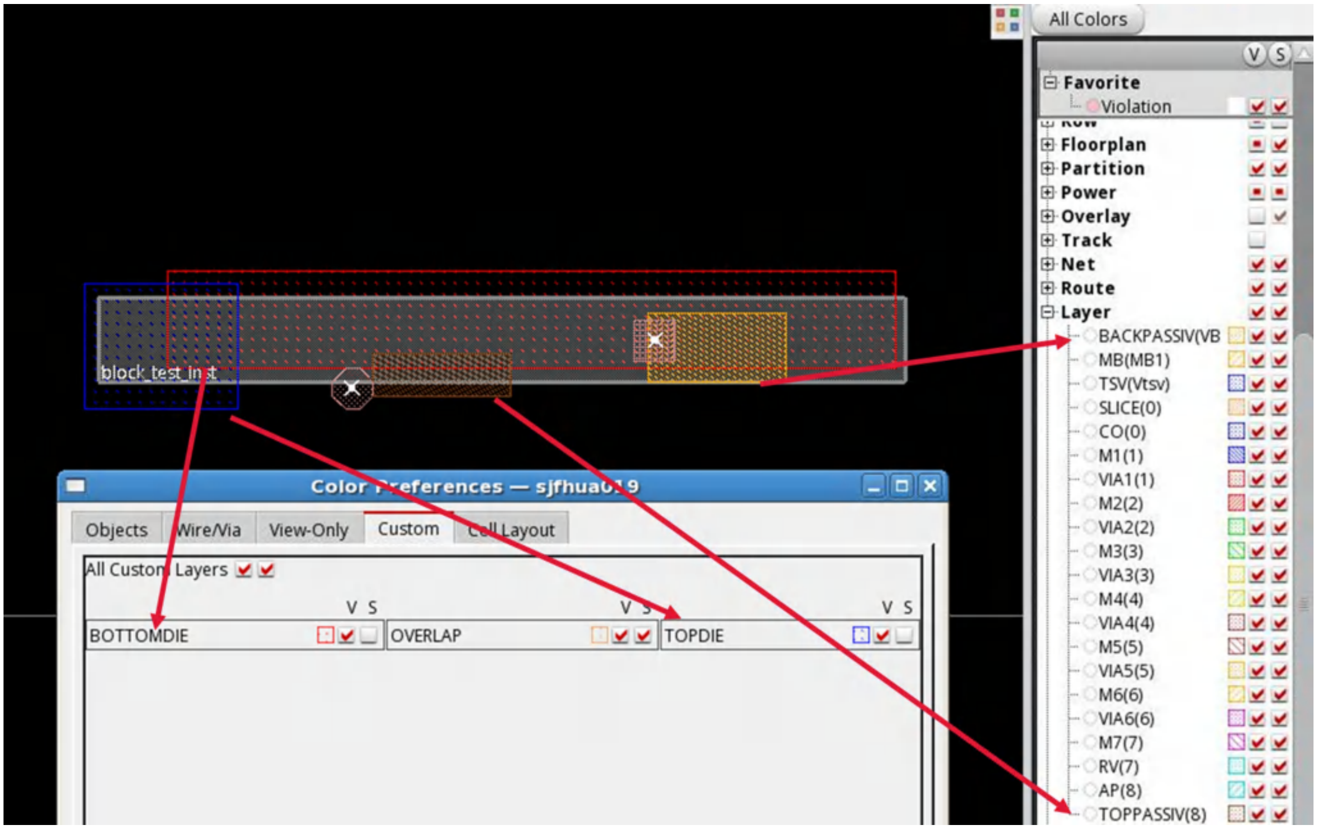
……

```
OBS

    LAYER TOPDIE
    SPACING 10 ;
    RECT -10 -20 100.940 70.885 ;
    LAYER BOTTOMDIE
    SPACING 5 ;
    RECT 50 10 580 80.885 ;
    LAYER TOPPASSIV
    SPACING 20 ;
    RECT 200 -10 300 20 ;
    LAYER BACKPASSIV ;
    RECT 400 0 500 50 ;
```

To control the visibility of the OBS layers, click the *Wire/Via* tab of the *Color Preferences* form (*Options* - *All Colors*) and select these layers.

To control the visibility of the OBS layers on BACKPASSIV and TOPPASSIV, click the *Layer* tab of the *Color Preferences* form, and select these layers. To control the visibility of the OBS layers on BOTTOMDIE and TOPDIE, click the *All Colors* button, select Custom tab, and then select these layers.

Figure 3 OBS Layers Example



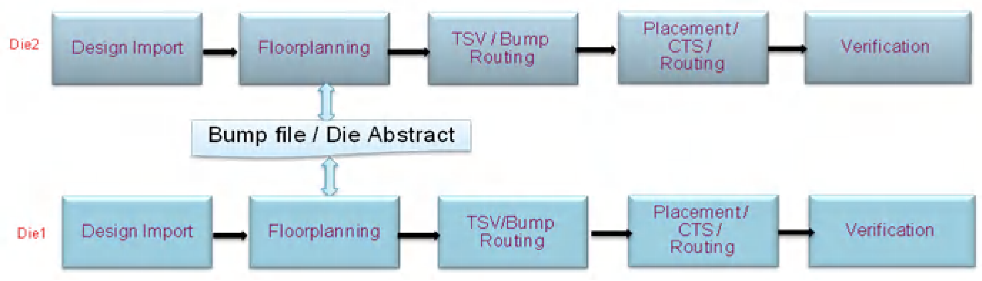
Check Bump Keep Out Area Violation

eUse the `verify_stacked_die -check_type` {bump_keepout} command parameter to check the overlap between the OBS layer and bumps (both front and back bumps). You can use the `verify_stacked_die -check_type {die_edge}` command parameter for die edge violation checking between adjacent dies. Once you define an OBS layer inside a macro, the adjacent die edge should not overlap this OBS area. If there is any violation between macro OBS and adjacent die edge, it will be marked in the GUI window. The `verify_stacked_die` command must be used after `readDieAbstract` to input adjacent die’s die abstract file. Both top and bottom die edge violation is checked at the same time.

3D IC Flow in Innovus

The following figure shows the general Innovus 3D IC design flow. Only 2 dies are included in this flow chart, and it can be extended to multiple dies.

Figure 4 3D IC Design Flow in Innovus

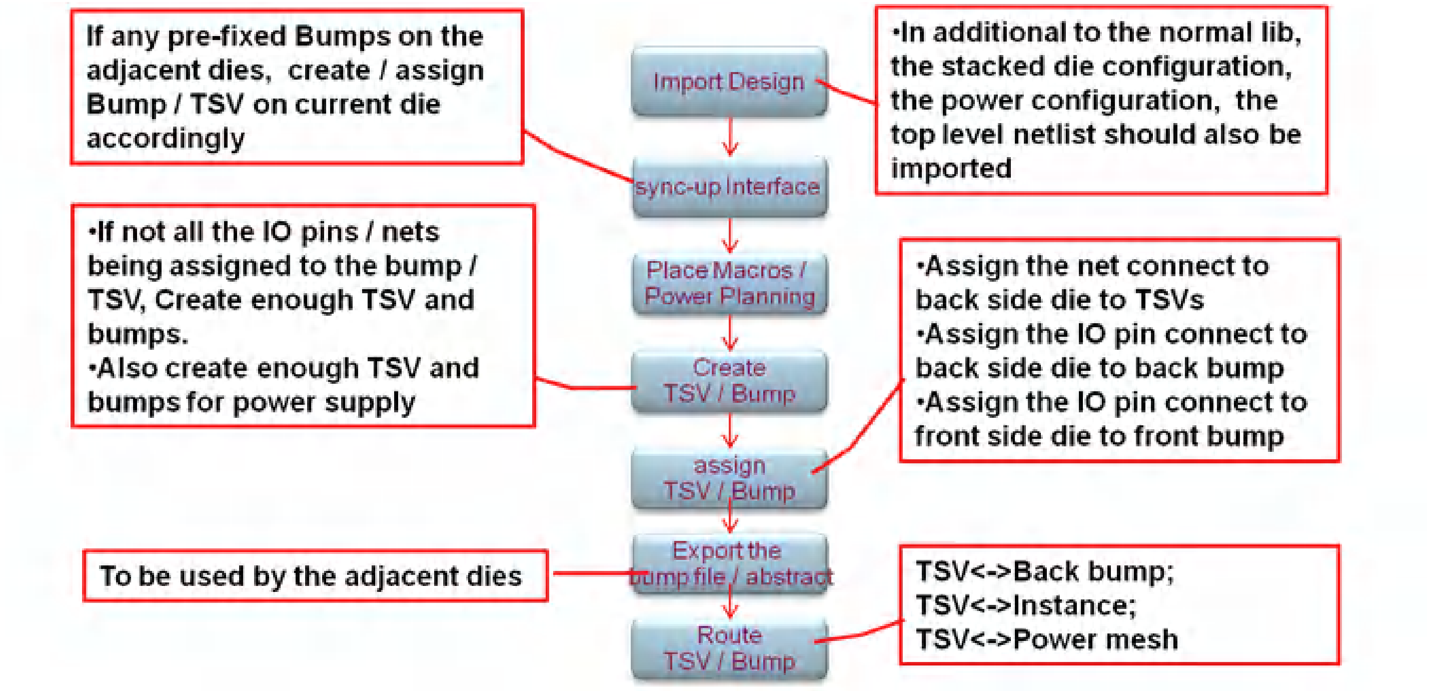


Innovus designs one die at a time. When designing one die, the micro bump and the instance on the adjacent die are honored and the micro bump on the current die is synchronized and optimized accordingly. The design flow for each die is quite similar to the normal design, except the following steps:

1. In the design import stage, additional configuration for stacked die should be imported.
2. In the floorplan stage, interface between each dies is ensured to be synchronized, and TSV/Bumps is created and assigned.
3. After floorplanning, the TSV and bump should be routed.
4. In the verification stage, the connectivity between dies should be verified.

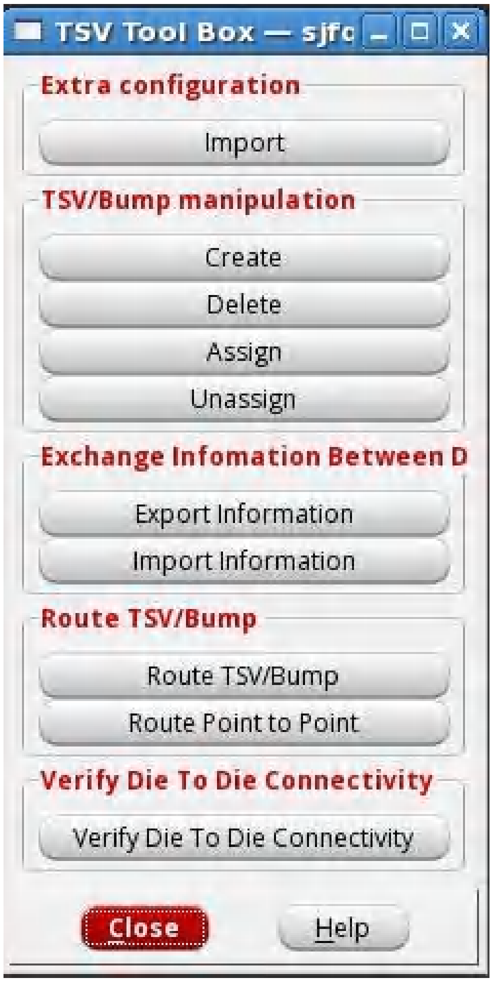
Figure 5 shows the detailed floorplan flow for a 3D IC design.

Figure 5 Detailed Floorplan Flow for One of the Dies


If any pre-fixed Bumps on the adjacent dies, create / assign Bump / TSV on current die accordingly
•If not all the IO pins / nets being assigned to the bump / TSV, Create enough TSV and bumps.
•Also create enough TSV and bumps for power supply
To be used by the adjacent dies
Import Design
sync-up Interface
Place Macros / Power Planning
Create TSV / Bump
assign TSV / Bump
Export the bump file / abstract
Route TSV / Bump
•In additional to the normal lib, the stacked die configuration, the power configuration, the top level netlist should also be imported
•Assign the net connect to back side die to TSVs
•Assign the IO pin connect to back side die to back bump
•Assign the IO pin connect to front side die to front bump
TSV<->Back bump;
TSV<->Instance;
TSV<->Power mesh

Innovus provides *TSV Tool Box* to perform various 3D IC flow task. It contains the forms used in the 3D IC flow, including TSV/Bump manipulation, data exchange, TSV/Bump routing, and design verification. Choose *Tools -TSV* to open it.

Figure 6 GUI Form of TSV Tool Box



Related Topic

- "TSV Tool Box" section of the Tools Menu chapter in the *Innovus Menu Reference*.

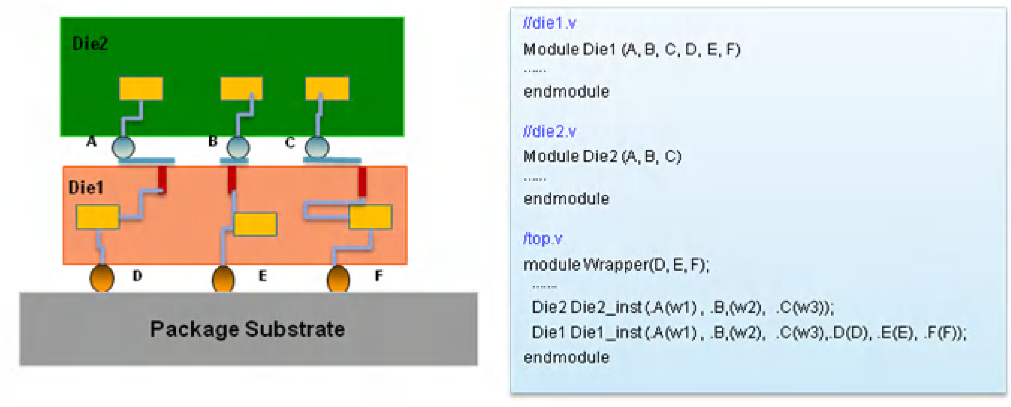
Design Import

Innovus takes the conventional and the additional information for stacked dies as inputs for each die design.

Stacked IC Verilog Input

Innovus inputs an additional top-level net list to describe the die-to-die interconnection and die to package interconnection. In the top-level netlist, package is set as the top module; each die is instantiated and the connectivity among dies is described. Figure 7 shows an example of the top-level netlist.

Figure 7 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
```

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. `init_top_cell` should be set as the module name of the die to be designed. Considering Figure 7 as an example, the following setting in the design configuration is for Die1 design.

```
set init_top_cell {Die1}

set init_verilog { Die1.v top.v }
```

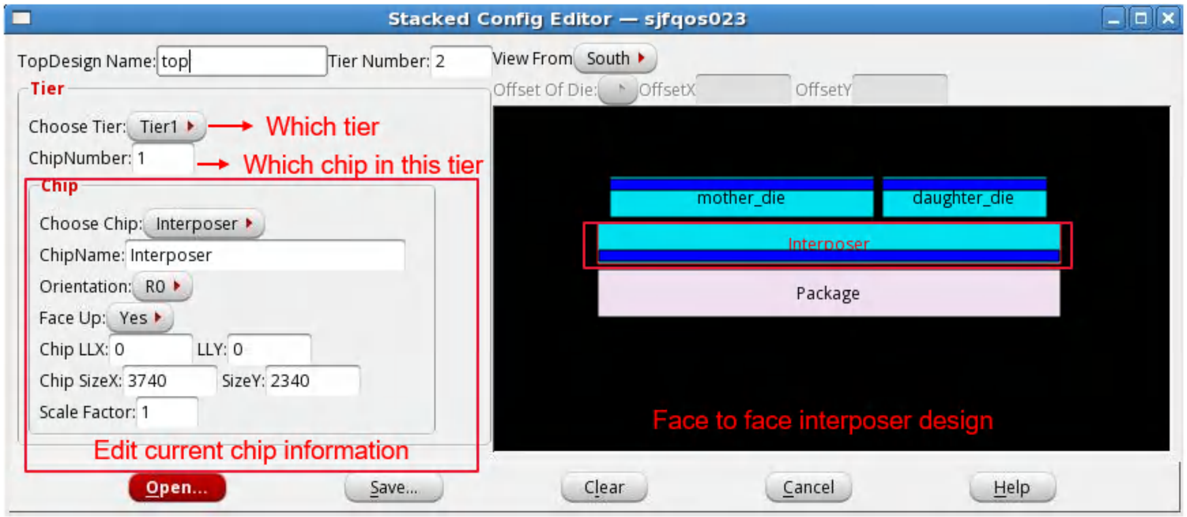
Stack Configuration Input

Innovus inputs an xml file that describes the position, flipping, and the orientation for each die related to the package. The xml file could be loaded by the `readTSVConfig` command directly or generated and edited through the *Stacked Config Editor* form.

```
<StackChip>

        <TopDesign name="top" offsetX=0.0 offsetY=0.0 orientation="R0" />

        <Tier number=1>

                <Chip name="Interposer" orientation="R0" faceUp="yes" llx=0 lly=0
sizeX=3740 sizeY=2340 scaleFactor=1 />
```

```
        </Tier>

        <Tier number=2>

                <Chip name="mother_die" orientation="R0" faceUp="no" llx=100 lly=100
sizeX=2140 sizeY=2140 scaleFactor=0.8 />

                <Chip name="daughter_die" orientation="R0" faceUp="no" llx=2300 lly=500
sizeX=1340 sizeY=1340 scaleFactor=1 />

        </Tier>

</StackChip>
```

Figure 8 Stack Config File Example



Stacked Config Editor — sjfqos023

TopDesign Name: top
Tier Number: 2
View From: South
Offset Of Die: OffsetX OffsetY

Tier
Choose Tier: Tier1
ChipNumber: 1

Chip
Choose Chip: Interposer
ChipName: Interposer
Orientation: R0
Face Up: Yes
Chip LLX: 0 LLY: 0
Chip SizeX: 3740 SizeY: 2340
Scale Factor: 1

For more information, see the "*Stacked Config Editor*" form in the Tools Menu chapter of the *Innovus Menu Reference Guide*.

Power Connectivity Input

A text file is used to set up P/G net configuration on each die.

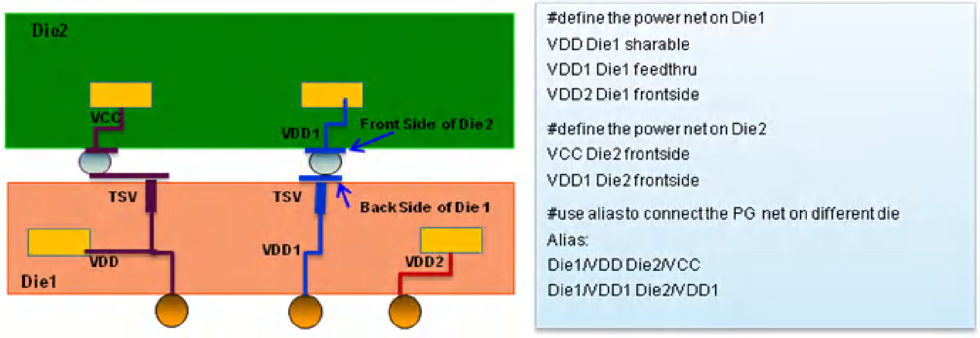
There are four types of P/G nets: frontside, backside, sharable, feedthru.

- frontside: The P/G net is connected to the front side bump only
- backside: The P/G net is connected to the back side bump only
- sharable: The P/G net is connected to both front and back side bumps and other instance in the chip
- feedthru: The P/G net is connected to front and back side bumps directly, without the

connection to any instance of current chip; use alias to define the same P/G net on different dies

Figure 9 shows an example of a power connectivity file.

Figure 9 Example of Power Configuration File



```
#define the power net on Die1
VDD Die1 sharable
VDD1 Die1 feedthru
VDD2 Die1 frontside
#define the power net on Die2
VCC Die2 frontside
VDD1 Die2 frontside
#use alias to connect the PG net on different die
Alias:
Die1/VDD Die2/VCC
Die1/VDD1 Die2/VDD1
```

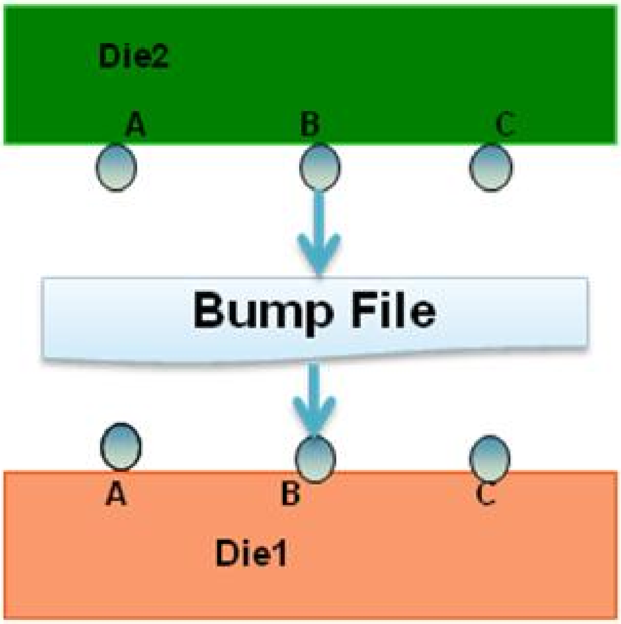
Related Information

- `readTSVConfig` command in the "Through Silicon Via Design Commands" chapter of the *Innovus Text Command Reference*.
- "`Load Stacked Die Config`" form in "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference*.

Interface Synchronization and Information Exchange between Dies

In 3D ICs, the micro bump is the data/power path between dies. As shown in Figure 2, the pad of micro bump should be exactly aligned to build the connection between the dies. Innovus uses the bump file to transfer the bump information from one die to the adjacent die. The bump file contains all the bump information of a die. Firstly, the bump file is dumped out by `writeBumpLocation` command from one die. Then `readBumpLocation` command should be called on the adjacent die. The `readBumpLocation` command creates and assigns micro bumps on the current die according to the bumps on the adjacent die, as shown in the following picture.

Figure 10 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. The `readDieAbstract` command imports the adjacent die information. This information is displayed and honored by the die being designed.

Related Information

- `writeBumpLocation`, `readBumpLocation`, `writeDieAbstract`, and `readDieAbstract` commands in the "Through Silicon Via Design Commands" chapter of the Innovus Text Command Reference.
- "Exchange Information Between Dies" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference Guide*.

TSV and Bump Manipulation

Innovus provides the capability to create/delete and assign/unassign TSV and bumps.

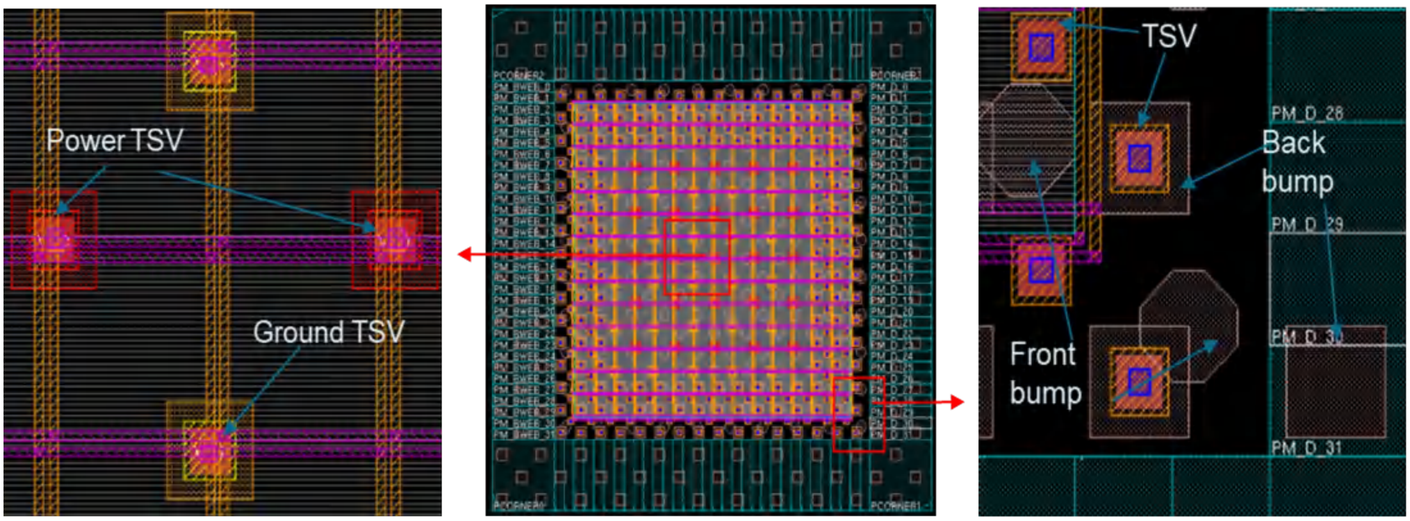
TSV/Bump Generation

There are multiple ways to create TSV/Bumps in Innovus.

- Create TSV/Bumps according to bumps on adjacent die
  If there are some micro bumps fixed on adjacent dies, user could export the bump file for the adjacent die with `writeBumpLocation` command. Then in the current die, import the bump file with `readBumpLocation` command and specify `-frontBump`, `-backBump`, and `-tsvViaName` option in read_bump_location to create corresponding front side bump, back side bump and TSV.
- Create TSV/Bumps on current die by command
  If there's no micro bump fixed on adjacent dies, run `addTSV -addTSV`, `-frontBump`, and `-backBump` to create corresponding cells and specify `-lowerLeftLoc`, `-pitchxy`, and `-upperRightLoc` options to create expected TSV/Bump array.

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.

Figure 11 TSV/Bumps in GUI



TSV/Bump Assignment

After creating TSV/Bump, user can now assign signal and PG TSV, frontside and backside bumps.

- Assign TSV/Bumps according to bumps on adjacent die

If there are some micro bumps fixed on adjacent dies, read_bump_locations help user create and assign bumps automatically.

- Create TSV/Bumps on current die by command

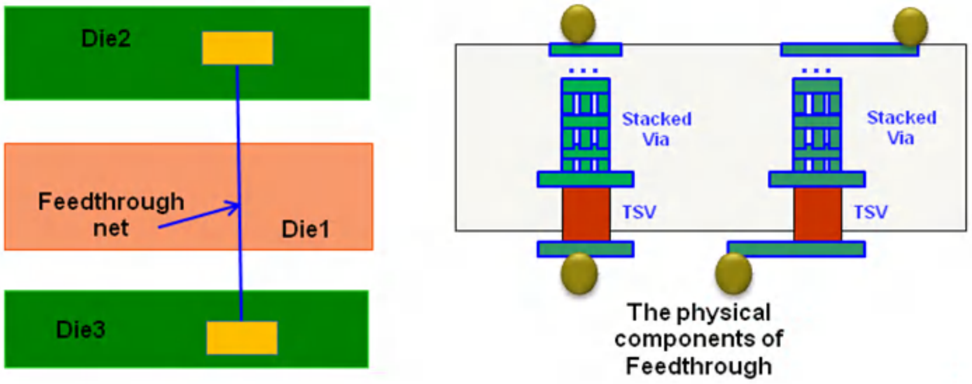
Run `assignTSV` to assigns nets to TSVs, front side and back side bumps, and/or feedthrus.

- `assignTSV -frontBump` help user assign front side bumps with the IO pins.
- `assignTSV -backBump` help user assign back side bumps and TSVs.
- `assignTSV -tsvViaName` help user assign specified TSV cell only.

Feedthru Handling

In a 3D IC 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 12.

Figure 12 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. The `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`. The `assignTSV` will not assign this back side pin to TSV.

Related Information

- `addTSV`, `deleteTSV`, `assignTSV`, and `unassignTSV` commands in the "Through Silicon Via Design Commands" chapter of the *Innovus Text Command Reference*.
- "TSV/Bump Manipulation" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference*.

TSV and Bump Routing

In a 3D IC design, user usually needs to route TSV/Bump to bumps, PG stripes, instance pins, and IO pads routing. Innovus provides several route engines to support routing for TSV/Bumps.

TSV to IO Pads/ Bumps/ PG Stripes Routing

1. Route TSV to IO Pad. Use `fcroute` with `aio` mode to support TSV to IO pad routing. For example:

   ```
   fcroute -type signal -designStyle aio -connectTsvToPad -routeWidth 3 -
   layerChangeTopLayer METAL4 -layerChangeBotLayer METAL1
   ```

2. Route TSV to bump. Innovus cannot route front side bump and back side bump at the same time, which needs to route separately.For example, to route TSV and back side bump, set both top and bottom routing layer as back side layer and define extra configuration `srouteExcludeBumpType`. Use the following commands:

   ```
   fcroute -type signal -designStyle aio -routeWidth 8 -layerChangeTopLayer MB -
   layerChangeBotLayer MB -connectTsvToBump -extraConfig ./conf/backside.extraConf
   ```

   In the `backside.extraConf`, user needs to define "`srouteExcludeBumpType FRONT_BUMP`", "`FRONT_BUMP`" is the cell name of the front side bump.

   To route TSV and front side bump, set both top and bottom routing layer as front side layer and define extra configuration `srouteExcludeBumpType`:

   ```
   fcroute -type signal -designStyle aio -routeWidth 8 -layerChangeTopLayer METAL4 -
   layerChangeBotLayer METAL1 -connectTsvToBump -extra_config
   ./conf/frontside.extraConf
   ```

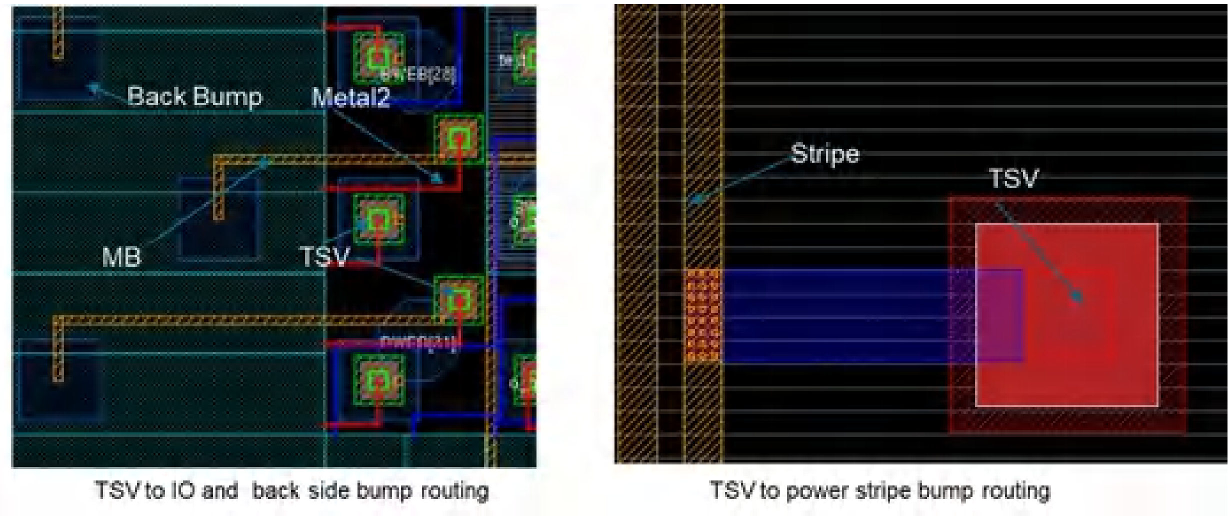
   In the `frontside.extraConf`, user need to define `"srouteExcludeBumpType BACK_BUMP"`, `"BACK_BUMP"` is the cell name of the back side bump.

3. Route TSV to power ground stripe. Use `fcroute` with type `-connectTsvToRingStripe` to route TSV and power groundstripes, for example:

   ```
   fcroute -type power -connectTsvToRingStripe -routeWidth 6 -layerChangeTopLayer
   METAL2 -layerChangeBotLayer METAL1
   ```

   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.

Figure 13 TSV/Bumps Routing



TSV to IO and back side bump routing

TSV to power stripe bump routing

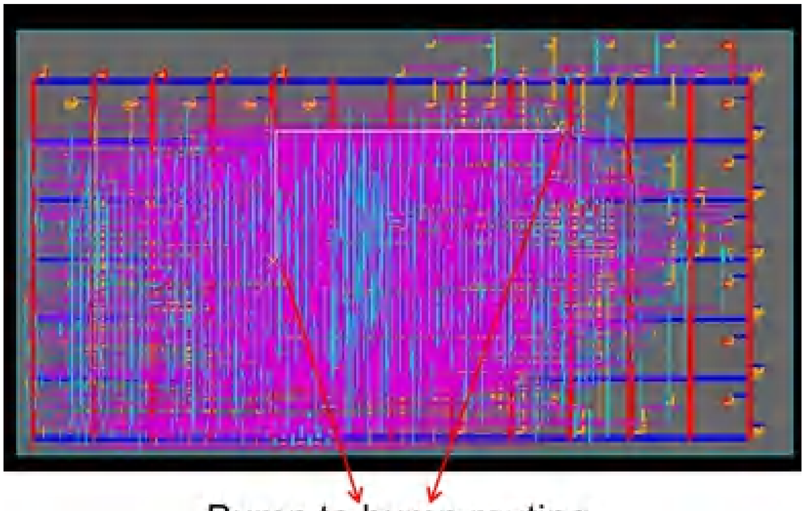
Bump to Bump Routing

Front Bump to Front Bump Routing

An interposer design has no instance in the design and requires front bump connect to front bump directly. You can use nano route to route them. Use the following setting and commands:

```
setNanoRouteMode -route_selected_net_only

setNanoRouteMode -route_connect_to_bumps

routeDesign
```

Figure 14 Bump to Bump Routing



Bump to bump routing

Front Bump to Front Bump Bus Routing

In interposer design that connects HBM dies and SoC dies, usually bus routing is required to connect HBM dies and SoC dies to ensure the signal transmission performance. This section illustrates the routing methodology for bus routing in interposer design.

Routing Steps:

1. Create Non-Default Rule (NDR) and Bus Routing Constraints.
    a. Following command defines the NDR (2 um width and 3 um spacing on layer METAL2 and METAL4) to be used on bus routing:
    ```
    add_ndr -width {METAL2 2.0 METAL4 2.0} -spacing {METAL2 3 METAL2 3} -name
    bus_ndr
    ```
    b. Following command defines bus net constraints to be used in routing. It requires tool using only METAL2 as bus routing layer and bus_ndr as routing rule. Tool may switch layers of last few wire segments when connecting to bump pins.
    ```
    setIntegRouteConstraint -type bus -topLayer METAL2 -bottomLayer METAL2 -rule
    bus_ndr -net {bus_net_1 bus_net_2 …}
    ```
2. Set interlayer shielding constraints
   In some scenarios, user may want to create shielding wires between signal routing layers. It is named as “interlayer shielding”.
    a. Interlayer Shielding offset
       Following example defines the offset (2.8 um) between centerline of signal wires and centerline of interlayer shielding wires on METAL1 and METAL3 layer.

```
setNanoRouteMode -route_interposer_interlayer_shielding_offsets {METAL1 2.8
METAL3 2.8}
```

b. Interlayer Shielding Width
   Following example defines routing width (2 um) of interlayer shielding wires on layer METAL1 and METAL3.

```
setNanoRouteMode -route_interposer_interlayer_shielding_widths {METAL1 2.0
METAL3 2.0}
```

c. Interlayer shielding layers
   Following example defines interlayer shielding wires are routed below M2 and M4 layer, respectively.

```
setNanoRouteMode -route_interposer_interlayer_shielding_layers {METAL2 bottom
METAL4 bottom}
```

d. Interlayer shielding net name
   Following example defines shielding net name is VSS for shielding wires in the layer range of METAL1 to METAL4

```
setNanoRouteMode -route_interposer_interlayer_shielding_nets {METAL1:METAL4
VSS}
```

3. Set same-layer interleaved shielding constraints
   In some scenarios, user may want to create shielding wires between signal routing wires on same layer. It is named as “interleaved shielding”.

   a. `setFlipChipMode -constraintFile fc_shield.const`

   b. `fc_shield.const` file content.
      Following example defines shielding width to be 1.0 um, shielding-to-signal spacing to be 2.0 um and shielding net is VSS

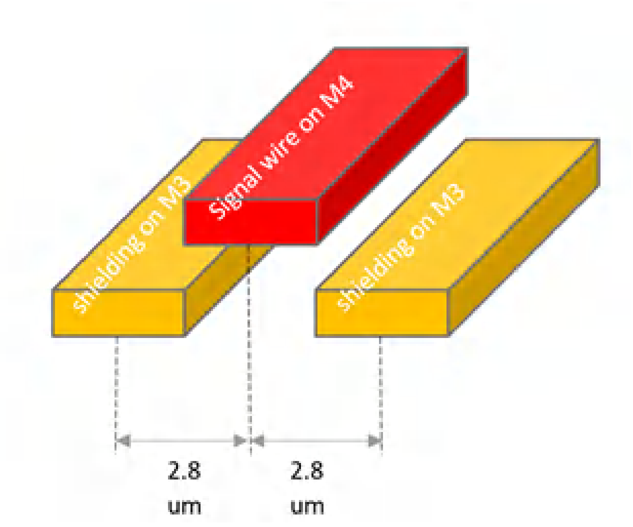
```
SHIELDWIDTH 1.0
SHIELDGAP 2.0
SHIELDSTYLE c
SHIELDNET VSS
```

4. Create PG stripes over PG bumps

5. Route constrained Bus nets and shielding

```
routeDesign -bump
```

Figure 15 Shielding Pattern



Front Bump to TSV-Bump Feedthru Net Routing

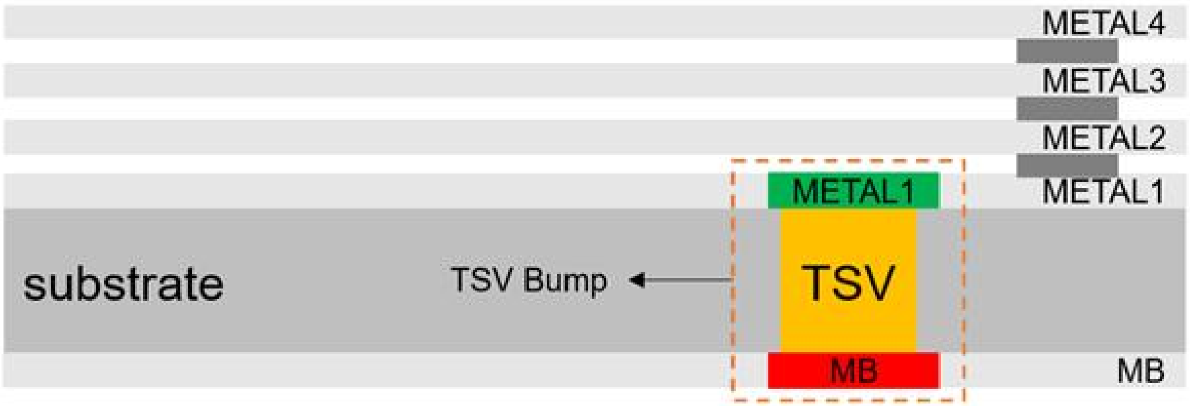
In some situations, TSV via and back side bump are packed together and formed a new bump cell. It is named TSV bump. TSV bump usually has multiple geometries on both front-side, TSV layer and back-side. `fcroute` has the capability to route from front bump to TSV-bumps by using option `-connectTsvToBump` and constraint BUMP_AS_PAD.

```
fcroute -type connect_bump_to_pad  -connectTsvToBump -layerChangeTopLayer METAL4 -
layerChangeBotLayer METAL1 --constraintFile fc.const -nets “net1 net2 …”
```

In the flipchip.const, user need to define "BUMP_AS_PAD" constraint, "TSV_bump_cell_name" is the cell name of TSV-bumps.

```
BUMP_AS_PAD

<TSV_bump_cell_names>

END BUMP_AS_PAD
```

Figure 16 TSV-Bump cell



TSV/Bump to Instance Pin Routing

In 3D stacked die design, sometimes there is no IO pads. TSVs and front bumps connect to instance pins directly. Innovus supports it by using nano route. Use the following setting and commands:

```
setNanoRouteMode -route_selected_net_only true

setNanoRouteMode -route_connect_to_bumps true

routeDesign
```

Related Information

- fcroute command in the "Flip Chip Commands" chapter of the *Innovus Text Command Reference*.
- "*Route TSV/Bump*" form in the "TSV Tool Box" section of the "Tools Menu" chapter in the *Innovus Menu Reference*.
- "Flip Chip Methodology" chapter of the *Innovus User Guide*.
- `add_ndr` and `routeDesign` commands in the "Route Commands" chapter of the *Innovus Text Command Reference*.
- `setIntegRouteConstraint` command in the "Mixed Signal Commands" Chapter of the *Innovus Text Command Reference*.

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`. The violation will be shown on the violation browser, and the violation marker will be displayed on the layout window.

`readBumpLocation -checkAlignment` can also be used for checking bump alignment.

Related Information

- `verifyConnectivity` command in the "Verify Commands" chapter of the *Innovus Text Command Reference*.
- `readBumpLocation` command in the "TSV Design Commands" chapter of the *Innovus Text Command Reference*.
- "*Verify Connectivity*" form in the "*Verify Menu*" section of the Tools Menu chapter in the *Innovus Menu Reference*.
- "Identifying and Viewing Violations" chapter in the *Innovus User Guide*.

Export Files

After design is finished, innovus could export extra files for 3D IC designs excepted conventional files like DEF, GDSII etc. The extra files are used for downstream analysis tools for purposes of STA, extraction and rail analysis.

- For STA
  Innovus is able to create a top-level SPEF that describes the connectivity between dies. The RC is 0 in this SPEF file, but only inter-die connectivity is written. This SPEF file, together with the SPEF for each die, is the input of Tempus for static timing analysis. Use following commands:

  ```
  createTSVNoLoadSPEF
  ```

- For RC Extraction and Rail Analysis
  Innovus is able to write micro bump mapping file for Quantus and Voltus. It describes the connections of micro bumps between adjacent dies.

  ```
  writeMicroBumpMappingFile
  ```

Related Information

- `createTSVNoLoadSPEF and writeMicroBumpMappingFile` commands in the "Through Silicon Via Design Commands" chapter of the *Innovus Text Command Reference*.

12

Syntax and Scripts

CCOpt Properties

- cell_halo_x
- cell_halo_y
- check_route_follows_guide
- check_route_follows_guide_min_length
- clock_gate_buffering_location
- clock_gating_cells
- clock_gating_depth
- clock_gating_depth_top_down
- clock_gating_only_optimize_above_flops
- clock_period
- clock_source_cells
- clock_tree
- clock_tree_generator_sink_is_leaf
- clock_tree_source_group
- clock_trees
- clone_clock_gates
- clone_clock_logic
- cloning_inst_name_suffix
- cloning_inst_name_suffix_source_group_assignment
- compatibility_warning
- consider_during_latency_update
- consider_power_management
- constrains
- cts_clock_gate_movement_limit
- cts_merge_clock_gates
- cts_merge_clock_logic
- def_lock_clock_sinks_after_routing
- delay_cells
- detailed_cell_warnings
- effective_clock_halo_x
- effective_clock_halo_x_source
- effective_clock_halo_y
- effective_clock_halo_y_source
- effective_clock_period
- effective_clock_period_sources

- effective_sink_type
- enable_all_views_for_io_latency_update
- enable_nanoroute_layer_check_failure
- error_on_problematic_slew_violating_nets
- error_on_problematic_slew_violating_nets_max_printout
- exclusive_sinks_rank
- exit_if_stage_delay_sigma_target_over_constrained
- extract_balance_multi_source_clocks
- extract_clock_generator_skew_group_name_prefix
- extract_clock_generator_skew_groups
- extract_clock_group_skew_group_name_prefix
- extract_cts_case_analysis
- extract_faster_sdc_clocks_as_clock_trees
- extract_network_latency
- extract_no_exclude_pins
- extract_pin_insertion_delays
- extract_skew_group_sinks_at_clock_node_timing_endpoints
- extract_source_latency
- extract_through_multi_output_cells_with_single_clock_output
- extracted_from_clock_name
- extracted_from_constraint_mode_name
- extracted_from_delay_corners
- filter_cell_lists_for_frequency_dependent_max_cap_constraints
- final_cell
- flexible_htree
- flexible_htree_placement_legalization_effort
- force_all_virtual_delay_updates
- force_clock_objects_to_propagated
- force_clock_tree
- force_update_io_latency
- frequency_dependent_max_cap_usability_check_max_cap_fanout_factor
- generated_by_sinks
- htree_sinks
- hv_balance
- ideal_net

- ignore_pins
- ignore_problematic_skew_as_result_of_dont_touch_nets
- image_directory
- implicit_sink_type
- include_source_latency
- insertion_delay
- insertion_delay_sources
- inst_name_prefix
- inverter_cells
- inverting
- is_active
- isolated
- is_genus_clock_gate
- is_sdc_clock_root
- layer_density
- leaf_buffer_cells
- leaf_inverter_cells
- legalized_on_clock_spine
- load_capacitance_cells
- library_trace_through_to
- lock_on_clock_spine
- log_precision
- log_special_case_cell_selections
- logic_cells
- long_path_removal_cutoff_id
- long_path_removal_percentile
- manage_power_management_illegalities
- max_buffer_depth
- max_cell_height
- max_clock_cell_count
- max_driver_distance
- max_fanout
- maximum_insertion_delay
- max_root_distance
- max_source_to_sink_net_length

- max_source_to_sink_net_resistance
- mixed_fanout_net_type
- mode
- move_clock_gates
- move_logic
- move_middle_cell_first_when_adding_wire_delay
- net_name_prefix
- net_type
- net_unbufferable
- node_type
- omit_symmetry
- opt_ignore
- original_names
- override_minimum_max_trans_target
- override_minimum_skew_target
- override_vias
- override_zero_placeable_area
- parents
- partition_boundary_polarity
- partition_groups
- pin
- pin_capacitance_sources
- pin_insertion_delay_histogram_bin_size
- pin_route_type
- pin_route_type_propagation
- pin_target_max_trans
- place_driver_in_center_of_fanout
- post_conditioning
- post_conditioning_enable_drv_fixing
- post_conditioning_enable_drv_fixing_by_rebuffering
- post_conditioning_enable_routing_eco
- post_conditioning_enable_skew_fixing_by_rebuffering
- power_weight
- preserve_from_deletion
- primary_delay_corner

- primary_reporting_skew_groups
- primary_reporting_skew_groups_log_min_max_sinks
- pro_enable_drv_fixing_by_rebuffering
- recluster_ignore_pins
- remove_bufferlike_clock_logic
- rename_clock_tree_nets
- repair_congestion
- report_only_skew_group_with_target
- report_only_timing_corners_associated_with_skew_groups
- route_balancing_buffers_with_default_rule
- route_type
- route_type_autotrim
- routing_override
- routing_preferred_layer_effort
- routing_top_fanout_count
- routing_top_min_fanout
- routing_top_transitive_fanout
- schedule
- sink_grid
- sink_grid_box
- sink_grid_exclusion_zones
- sink_instance_prefix
- sink_grid_sink_area
- sink_type
- sink_type_reasons
- sinks
- sinks_active
- size_clock_gates
- size_clock_source
- size_logic
- skew_band_size
- skew_group_report_columns
- skew_group_report_histogram_bin_size
- skew_groups_active
- skew_groups_active_sink

- skew_groups_ignore
- skew_groups_constraining
- skew_groups_constraining_sink
- skew_groups_sink
- skew_groups_source_pin
- skew_passes
- skew_passes_ideal_mode
- skew_passes_per_cluster
- source_driver
- source_group_clock_trees
- source_input_max_trans
- source_latency
- source_max_capacitance
- source_output_max_trans
- source_pin
- spec_config_create_reporting_only_skew_groups
- stack_via_rule
- stack_via_rule_required
- sources
- stop_at_sdc_clock_roots
- target_insertion_delay
- target_insertion_delay_wire
- target_max_stage_delay_sigma
- target_max_capacitance
- target_max_trans
- target_max_trans_sdc
- target_multi_corner_allowed_insertion_delay_increase
- target_skew
- target_skew_wire
- timing_connectivity_based_skew_groups
- timing_connectivity_based_skew_groups_balance_master_clocks
- timing_connectivity_info
- top_buffer_cells
- top_inverter_cells
- trace_bidi_as_input

- trace_through_to
- transitive_fanout
- trunk_cell
- trunk_override
- update_io_latency
- use_estimated_routes_during_final_implementation
- use_inverters
- use_receiver_model_capacitance_for_drv
- use_macro_model_pin_cap_only
- useful_skew_implementation_cache_hold_slacks
- useful_skew_clock_gate_movement_limit
- useful_skew_max_delta
- useful_skew_min_delta
- useful_skew_post_implement_db
- useful_skew_pre_implement_db
- virtual_delay

To set the value of any CCOpt property, use the `set_ccopt_property` command:

```
set_ccopt_property use_inverters -clock_tree ct1 true
```

To get the current value of any CCOpt property, use the `get_ccopt_property` command:

```
get_ccopt_property property_name
```

Example:

```
get_ccopt_property cell_density
```

The software returns the following information:

```
1
```

To get a description of any CCOpt property, use the following command:

```
get-ccopt_property -help property_name
```

For more information, see "CCOpt Property System" in the Clock Tree Synthesis chapter of the *Innovus User Guide*.

add_driver_cell

Specify a driver cell to add for clock tree after root.

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`".

add_exclusion_drivers

By default, this property is set to `true`, which causes CCOpt to add extra drivers above exclude pins to remove them from the clock tree.

Default: `true`

This global property does not use additional arguments.

add_port_driver

Specifies a cell type or pair of cell types, so that a cell instance can be added above an output port or below an input port. The port is specified by the argument pin. If the pin specified is not a design IO pin or it is not in the clock network then CCOpt will issue a warning and will not add cell instances at that position. No more than two cell types may be specified. A warning will be emitted if no AON cells are specified.

Default: `{}`

Applicable arguments: "`-pin <`*`name`*`>`". Required: "`-pin <`*`name`*`>`".

add_port_driver_max_distance_from_port

This property specifies the allowable distance, in um if no units suffix is specified, that a port driver is allowed to be placed from the port being buffered. If the driver cannot be added close enough to the port, it will not be added at all. This property, if set, overrides any value set in the property `add_port_driver_max_distance_from_port_rows`.
Default: `auto`

This global property does not use additional arguments.

add_port_driver_max_distance_from_port_rows

This property specifies the allowable distance, in standard cell row heights, that a port driver is allowed to be placed from the port being buffered. If the driver cannot be added close enough to the port it will not be added at all. The property, `add_port_driver_max_distance_from_port`, if set, overrides this value.

Default: `50`

This global property does not use additional arguments.

additional buffer_depth

Relaxes the `max_buffer_depth` constraint by additional N stage depths, but only for the shortest paths.

Allows the constraint on the maximum number of buffers that CTS can add on a source to sink path to be relaxed by adding an additional `additional_buffer_depth` stage depths to the constraint, but only for the shortest paths in a skew group. This can help make power/area savings while still fixing skew.

Refer to the help for the property, `additional_buffer_depth_skew_group_fraction` for how to control the number of paths where this property is applied.

Default: auto

Optional applicable arguments: "`-skew_group <`*`name`*`>`".

additional_buffer_depth_skew_group_fraction

When using `additional_buffer_depth`, apply the relaxed constraint to specified fraction of the skew group's shortest paths. The number of paths to apply additional buffering to is found by multiplying the number of paths in a skew group by this fraction, and then selecting the shortest N paths.

Valid values: Numbers in the range 0.0 to 1.0.

Default: 0.05

This global property does not use additional arguments.

adjacent_rows_legal

Defines the clock halo in the y direction. If set to `true`, the y direction clock halo is zero and other clock instances are allowed in adjacent rows.

The following properties can be used to assign y direction clock halos within CCOpt:

- `cell_halo_y`
- `adjacent_rows_legal`
- `cell_halo_rows`

Only one of these properties is used to determine the clock halo in the y direction. The following rules determine which:

- If `cell_halo_y` is set to a non-auto value, then this defines the y direction clock halo. The properties `adjacent_rows_legal` and `cell_halo_rows` have no effect.
- If `cell_halo_y` is set to `auto` and `adjacent_rows_legal` is set to a non-auto value then `adjacent_rows_legal` defines the clock halo in the y direction. The property `cell_halo_rows` has no effect.

- If both `cell_halo_y` and `adjacent_rows_legal` are set to `auto` then `cell_halo_rows` defines the clock halo in the y direction.

Valid values: `true false`

See also:
. `cell_halo_y`
. `cell_halo_rows`

Default: `false`

This global property does not use additional arguments.

adjust_sink_grid_for_aspect_ratio

By default, this property is set to `true` and it adjusts the sink grid for the aspect ratio of the sink grid box.

Valid values: `true false`

Default: true

Optional applicable arguments: "`-flexible_htree` *`name`*".

allow_non_standard_inputs_clock_gate

If this property is set, CTS will allow the use of clock gates with non-standard pins. CCOpt considers the following pin types to be standard: clock pins, enable pins, test enable pins, retention pins and power gating pins. Before starting CTS CCOpt will emit a warning, indicating which pin(s) it considers non-standard.

Type: `boolean`

Default: `false`

This global property does not use additional arguments.

annotated_delay_to

Overrides any clock tree timing engine-computed cell arc or net arc delays to the specified pin, in a manner similar to that of SDC `set_annotated_delay` command.

For cell arcs: The format is dict where the keys are pins and the values are a rise fall delay pair. For example:
`"<from pin 1> {<from rise delay 1> <from fall delay 1>} <from pin 2> {<from rise delay 1> <from fall delay 2>}"`

For net arcs: The format is dict where the keys are pins and the values are the delay. For example:
`"<from pin 1> <delay 1> <from pin 2> <delay 2>}"`

For convenience, set the property using a single value that will apply to all available delays from pins and rise/fall from edges. For example, `"0.100"`.

Default: `""`

Applicable arguments: `"-delay_corner <name>", "-pin <name>", "-early", "-late", "-rise" and "-fall". Required: "-pin <name>".`

annotated_transition

Overrides any clock tree timing engine-computed transition at the specified pin, in a manner similar to that of SDC `set_annotated_transition` command.

Default: `a`uto

Applicable arguments: `"-delay_corner <name>", "-pin <name>", "-early", "-late", "-rise" and "-fall". Required: "-pin <name>".`

auto_limit_insertion_delay_factor

CCOpt attempts to keep the insertion delays of each clock tree a fixed multiple of the longest insertion delay that would result from a global skew approach. This multiple can be modified by design timing and the presence of other clock trees, but will start at a fixed fraction above the global skew insertion delay. This property specifies that fixed fraction.

Valid values: `real`

See also: `auto_limit_insertion_delay_factor_skew_group`

Default: `1.5`

This global property does not use additional arguments.

auto_limit_insertion_delay_factor_skew_group

CCOpt uses this property to generate a maximum insertion delay for any skew groups on which it is set. This is done by multiplying the insertion delay for that skew group that would result from a global skew approach by the specified factor. For any skew groups where this property is not set, a limit based on the longest global skew insertion delay across all skew groups is used; see the `auto_limit_insertion_delay_factor` property.

Valid values: real

See also: `auto_limit_insertion_delay_factor`

Default: `auto`

Optional applicable arguments: "`-skew_group <`*`name`*`>`".

auto_limit_insertion_delay_max_increment

CCOpt attempts to keep the insertion delays of each clock tree to the longest insertion delay that would result from a global skew approach plus the given increment. This property specifies that fixed duration. If this property is set to `auto`, the value specified using the `auto_limit_insertion_delay_factor` property is used. The increment is specified in library time units, which are often but not always nanoseconds. When setting the property you can optionally specify the units to make sure the value given is interpreted correctly. For example:

```
set_ccopt_property auto_limit_insertion_delay_max_increment 200ps

get_ccopt_property auto_limit_insertion_delay_max_increment 0.2

set_ccopt_property auto_limit_insertion_delay_max_increment 0.25ns

get_ccopt_property auto_limit_insertion_delay_max_increment 0.25

set_ccopt_property auto_limit_insertion_delay_max_increment 0.1

get_ccopt_property auto_limit_insertion_delay_max_increment 0.1
```

Valid values: real

See also: `auto_limit_insertion_delay_max_increment_skew_group`

Default: `auto`

This global property does not use additional arguments.

auto_limit_insertion_delay_max_increment_skew_group

CCOpt uses this property to generate a maximum insertion delay for any skew groups on which it is set. This is done by adding the insertion delay for that skew group that would result from a global skew approach with the specified increment. For any skew groups where this property is not set, a limit based on the longest global skew insertion delay across all skew groups is used; see the `auto_limit_insertion_delay_max_increment` property.

Valid values: `real`

See also: `auto_limit_insertion_delay_max_increment`

Default: `auto`

Optional applicable arguments: "`-skew_group <`*`name`*`>`".

balance_mode

Replaces the CCOpt mode setting, `set_ccopt_mode -cts_opt_type {full | cluster | trial}`. If not set to `full`, causes CCOpt and CCOpt CTS to halt before final completion of the clock tree creation to facilitate clock tree inspection.

The possible values for this property are as follows:

- `full` - default value, a full CTS is performed.
- `cluster` - a cluster-only CTS is performed. The clock tree has no balancing delay applied.
- `trial` - The clock has only virtual (numeric annotation) balancing delays applied.

Type: `string`

Default: `full`

This global property does not use additional arguments.

blackbox_default_driver_base_pin

Base pin that will be used for all timing and capacitance modeling for clock roots with a source pin at a blackbox output.

Default: `{}`

This global property does not use additional arguments.

blackbox_default_load_base_pin

Base pin that will be used for all timing and capacitance modeling for clock sinks at blackbox inputs. Any capacitance_override property will be used in preference to this.

Default: `{}`

This global property does not use additional arguments.

buffer_cells

Specifies the buffer cells for CTS. If none are specified CCOpt will choose buffers from the libraries. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names.

If set explicitly, CCOpt will ignore any don't use settings for the cells specified. Different buffer cells may be specified for any combination of clock tree and power domain.

To use different buffers for each net type set the `top_buffer_cells` and `leaf_buffer_cells` properties.

Some examples are provided below:

To specify buffer cells for all clock trees and all power domains:

```
set_ccopt_property buffer_cells {bufAX* bufBX*}
```

To specify buffer cells for a particular clock tree and all power domains:

```
set_ccopt_property -clock_tree clk buffer_cells {bufX20 bufX18}
```

To specify buffer cells for a particular clock tree and power domain:

```
set_ccopt_property -clock_tree clk -power_domain pd buffer_cells {bufX12 bufX8}
```

Valid values: a list of library cell names, or a list of patterns to expand to library cell names

Default: `{}`

Optional applicable arguments: "`-clock_tree <`*`name`*`>`" and "`-power_domain <`*`name`*`>`".

cannot_clone_reason

A list of reasons to explain why the instance could not be cloned. This property will not be set if properties `clone_clock_gates`/`clone_clock_logic` are `false`. The property returns a list of reasons why the instance cannot be cloned.

The reasons why an instance cannot be cloned can be divided into three categories:

1. The inst is marked as dont_touch
2. Reasons specific to the clock node
3. General reasons

The possible dont_touch reasons are:

dont_touch.add_driver_cell	Set if the cell was added below the root via property `add_driver_cell`
dont_touch.clock_root	Set during clock tree extraction if identified as having the root pin
dont_touch.clock_sink	Set on cell if any pin is a clock sink in a clock tree
dont_touch.clock_tree_generator_path	Set if nodes and wires are in clock tree generator paths
dont_touch.clock_wire	Set on cell if clock input wire is user dont_touch
dont_touch.clockgate_no_power_domain	Set if clockgate/clocklogic is a clocknode in a no clockgate/clocklogic power domain

dont_touch.dft_observability_test_mode_pin	Set if a cell has the `dft_observability_test_mode_pin`
dont_touch.drives_multi_driver_net	Set if the cell drives wires with other drivers
dont_touch.external_skew_group_pin	Set if a pin on the cell is a skew group sink, source or ignore pin for a skew group created by the user
dont_touch.flexible_htree	Set if the cell was added to a flexible H-tree
dont_touch.internal_skew_group_pin	Set if a pin on the cell is a skew group sink, source or ignore pin for a skew group created by CCOpt
dont_touch.neg_edge_clock_gate	Set if a clock gate gates the falling edge (`extract_enable_neg_edge_clock_gate_handling` is `true`)
dont_touch.non_flop_clock_gating	Set by SetDontTouchBlackBoxGating
dont_touch.non_standard_inputs_clock_gate	Set if a cell is a clock gate with 'non-standard' inputs, to avoid disconnecting them, or if property `consider_all_clock_gates_to_have_non_standard_inputs` is `true`
dont_touch.observability_clock_gate	Set if a cell is a clock gate with an observability output pin
dont_touch.placer.lock	Set if a cell is user locked or locked by DEF
dont_touch.power_management	Set if a cell is a power management cell
dont_touch.sdc	Set if constrained by SDC timing
dont_touch.sdc_path_group	Set if there is an SDC path group start/endpoint on one of the pins of this cell
dont_touch.sub_block	Set if cell is in dont_touch module
dont_touch.unmergeable_composite_clock_gate	Set if some of the clock gate is dont_touch
dont_touch.user	Set by user
Contact Cadence Support if any of the following reasons are listed:	
dont_touch.cannot_understand_clock_gate	Set if clock gate not recognized, `extract_enable_neg_edge_clock_gate_handling` is `false`
dont_touch.composite_clock_gate_enable_test_or_gate	
dont_touch.initial_netlist	
dont_touch.output_wire	
dont_touch.port_preservation_prevents_cg_opt	
dont_touch.prevent_assign	

The possible reasons specific to the clock node are:

PowerManagement	Set if cell is power management cell and property `clone_power_management_cells` is `false`
PowerManagementInconsistency	Set if a disconnection of the cell from the netlist would cause a power management inconsistency

PreservedUMPB	Set if cell is a User Module Port Bit which is preserved
Contact Cadence Support if any of the following reasons are listed:	
ClockGatesAlways	Set if cell is clock gate, considered always dont_touch
ClockLogicAlways	Set if cell is clock logic, considered always dont_touch
NodeIsRoot	Set if the cell is the clock root (cannot be cloned)
NoRoot	Set if the clock node has no root
RootIgnored	Set if the clock root is ignored
SpineCell	Set if cell is a clock spine cell and is marked as dont_touch

The possible General Reasons are:

DrivesAcrossPowerDomains	The cell drives across power domain boundaries
NodeHasSGConstraints	The cell has user mode skew group constraints in default mode and property `clone_clock_cells_with_skew_group_constraints` is `false`
InvertingClockGate	The cell is an inverting clock gate and cloning of these is not supported
ClockDriverCannotCloneInverter	
ClockDriverCreatedByBuffLongNets	Need to keep cell as created by 'Buffering long nets' and property `clustering_leave_cmf_drivers_alone` is `true`
ClockDriverInverterCloningDisabled	The cell is an inverter and cloning inverters is disabled (property `clone_inverters` is `false`)
Contact Cadence Support if any of the following reasons are listed:	
ClockLogicMultiOutputCell	
InIgnoredTree	
IsUnclonable	
NonIntegratedClockGate	
NotTreeViewNode	
OutputPinCellPinNull	
ClockDriverNeedToKeep	

Valid values: list string

Default: `{}`

Applicable arguments: `"-inst <name>"`. Required: `"-inst <name>"`.

cannot_fix_pre_route

Mark a net as having violations that cannot be fixed with current opt settings.

Default: `false`

Applicable arguments: "`-net <name>`". Required: "`-net <name>`".

cannot_merge_reason

A list of reasons to explain why the instance could not be merged. The reasons why an instance cannot be merged can be divided into two categories:

1. The merge is prevented by user settings
2. The instance could not be merged with another inst due to mismatched attributes

The possible user-controlled reasons are:

IsDontTouch	Set if the cell is marked as dont_touch
ClockGateMergingDisabledOnTree	Set if `cts_merge_clock_gates` is `false` on the clock tree

The possible mismatched attributes are:

UniqueUnderParent	Set if two clock nodes have the same enables but different parents
DifferentNumberOfParents	Set if two clock gates have different number of parents
HasSkewGroupConstraints	Set if two clock nodes have different skew group constraints
DifferentSkewGroupForInputs	Set if two clock nodes have different skew group constraints for inputs
DifferentSkewGroupForOutputs	Set if two clock nodes have different skew group constraints for outputs
NoClockOutputPin	Set if two clock nodes have no clock output pin
DifferentNonStandardInputs	Set if two clock gates have different non standard inputs
MismatchingPowerDomains	Set if two clock gates with non standard inputs have mismatching power domains
DifferentCellFamilies	Set if two clock gates with non standard inputs have different cell families
DifferentNumberOfInputPins	Set if two clock gates with non standard inputs have different number of input pins
MismatchingNonClockInputs	Set if two clock gates with non standard inputs have mismatching non clock inputs
MismatchingClockInputs	Set if two clock gates with non standard inputs have mismatching clock inputs
IncompatibleRestrictedRegions	Set if two clock nodes have incompatible restricted regions

Valid values: list string

Default: `{}`

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

capacitance_override

Specifies an overridden capacitance value for this pin. Can be specified in pF, fF, F (default pF). Can set per delay corner (delay_corner) and with reference to an event (rise|fall). If set to `auto`, CCOpt will calculate the capacitance value using the library information.

Valid values: `Library dependent`

Default: `auto`

Applicable arguments: "`-delay_corner <name>`", "`-pin <name>`", "`-rise`" and "`-fall`". Required: "`-pin <name>`".

case_analysis

Specifies a constant value to be used as pin signal when analyzing timing within CTS. Only applies to input pins of cells in the clock tree. If set to '`none`' (the default), CTS will consider the input as non-constant.

Valid values: `0 1 none`

Default: `auto`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

ccopt_auto_limit_insertion_delay

If set to `true`, CCOpt will automatically limit the insertion delay of clock trees. The limits are determined by looking at the value of the `ccopt_auto_limit_insertion_delay_factor` property, and also taking into considering the timing of the design and the insertion delay of other clock trees in the design.

Type: boolean

Default: `true`

This global property does not use additional arguments.

ccopt_auto_limit_insertion_delay_factor

If the `ccopt_auto_limit_insertion_delay` property is true, CCOpt will attempt to keep the insertion delays of each clock tree a fixed multiple of the longest insertion delay that would result from a global skew approach. This multiple can be modified by design timing and the presence of other clock trees, but will start at a fixed fraction above the global skew insertion delay. This property gives that fixed fraction.

Type: real

Default: 1.5

This global property does not use additional arguments.

ccopt_merge_clock_gates

If set to `true`, clock gate merging is enabled. If this is `false`, merging of all clock gates is disabled, including clock gates which may have been cloned by CTS.

This property also impacts the early clock flow.

Note that this property has no impact on '`ccopt_design -cts`'. See also property `cts_merge_clock_gates`.

Valid values: `true false`

Default: `true`

Optional applicable arguments: "`-clock_tree <name>`".

ccopt_merge_clock_logic

If set to `true`, clock logic merging is enabled. If this is `false`, merging of all clock logics is disabled, including clock logics which may have been cloned by CTS.

Note that this property has no impact on '`ccopt_design -cts`'. See also property `cts_merge_clock_logic`.

Valid values: `true false`

Default: `true`

Optional applicable arguments: "`-clock_tree <name>`".

ccopt_worst_chain_report_timing_too

When CCOpt runs the worst chain report, additionally run `report_timing`.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

cell_density

Specifies the clock halo in the x direction. Any x direction clock halo defined by this property is proportional to the cell width.

The constant of proportionality is defined by the property value. For example:

- If `cell_density = 0.25` then the x direction clock halo equals 3 * cell width.
- If `cell_density = 0.5` then the x direction clock halo equals cell width.
- If `cell_density = 0.75` then the x direction clock halo equals cell width / 3.
- If `cell_density = 1` then the x direction clock halo equals zero.

This property can specify the x direction clock halo for all clock trees via:

```
set_ccopt_property cell_density 0.8
```

This property can specify the x direction clock halo per-clock tree via:

```
set_ccopt_property -clock_tree clk cell_density 0.9
```

The following properties can be used to assign x direction clock halos within CCOpt:

- `cell_halo_x`
- `cell_density`
- `cell_halo_sites`

Only one of these properties is used to determine the clock halo in the x direction. The following rules determine which:

- If `cell_halo_x` is set to a non-auto value, then this defines the x direction clock halo. The properties `cell_density` and `cell_halo_sites` have no effect.
- If `cell_halo_x` is set to auto and cell_density is set to a non-auto value then cell_density defines the clock halo in the x direction. The property `cell_halo_sites` has no effect.
- If both `cell_halo_x` and cell_density are set to auto then `cell_halo_sites` defines the clock halo in the x direction.

Valid values: `0.01 to 1`

See also:
. `cell_halo_x`
. `cell_halo_sites`

Default: `0.75`

Optional applicable arguments: `"-clock_tree <name>"`.

cell_halo_mode

Specifies how clock halos are used to determine the minimum legal separation between a pair of clock instances. There are two possible modes, '`max`' and '`sum`'. When set to '`max`' the minimum legal separation is the larger of the two clock halos. When set to '`sum`' the minimum legal separation is the sum of the two clock halos.

Valid values: `max sum`

See also:
. `cell_halo_x`
. `cell_halo_y`
. `cell_density`
. `adjacent_rows_legal`
. `cell_halo_sites`
. `cell_halo_rows`
. `effective_clock_halo_x`
. `effective_clock_halo_y`
. `effective_clock_halo_x_source`
. `effective_clock_halo_y_source`

Default: `max`

This global property does not use additional arguments.

cell_halo_rows

Specifies the clock halo in the y direction in rows for all clock cells.

The following properties can be used to assign y direction clock halos within CTS:

- `cell_halo_y`
- `adjacent_rows_legal`
- `cell_halo_rows`

Only one of these properties is used to determine the clock halo in the y direction. The following rules determine which:

- If `cell_halo_y` is set to a non-auto value, then this defines the y direction clock halo. The properties `adjacent_rows_legal` and `cell_halo_rows` have no effect.
- If `cell_halo_y` is set to auto and `adjacent_rows_legal` is set to a non-auto value then `adjacent_rows_legal` defines the clock halo in the y direction. The property `cell_halo_rows` has no effect.
- If both `cell_halo_y` and `adjacent_rows_legal` are set to auto then `cell_halo_rows` defines the clock halo in the y direction.

See also:
. `cell_halo_y`
. `adjacent_rows_legal`

Default: `1`

This global property does not use additional arguments.

cell_halo_sites

Specifies the clock halo in the x direction in sites for all clock cells.

The following properties can be used to assign x direction clock halos within CTS:

- `cell_halo_x`
- `cell_density`
- `cell_halo_sites`

Only one of these properties is used to determine the clock halo in the x direction. The following rules determine which:

- If `cell_halo_x` is set to a non-auto value, then this defines the x direction clock halo. The properties cell_density and cell_halo_sites have no effect.
- If `cell_halo_x` is set to auto and cell_density is set to a non-auto value then `cell_density` defines the clock halo in the x direction. The property `cell_halo_sites` has no effect.
- If both `cell_halo_x` and `cell_density` are set to auto then cell_halo_sites defines the clock halo in the x direction.

See also:
`.cell_halo_x`
`.cell_density`

Default: 4

This global property does not use additional arguments.

cell_halo_x

Specifies the clock halo distance in the x direction. The default value of this property is auto.

The following properties can be used to assign x direction clock halos within CTS:

- `cell_halo_x`
- `cell_density`
- `cell_halo_sites`

Only one of these properties is used to determine the clock halo in the x direction. The following rules determine which:

- If cell_halo_x is set to a non-auto value, then this defines the x direction clock halo. The properties cell_density and cell_halo_sites have no effect.
- If cell_halo_x is set to auto and cell_density is set to a non-auto value then cell_density defines the clock halo in the x direction. The property cell_halo_sites has no effect.
- If both cell_halo_x and cell_density are set to auto then cell_halo_sites defines the clock halo in the x direction.

See also:
`. cell_density`
`. cell_halo_sites`

Default: `auto`

Optional applicable arguments: `"-clock_tree <name>", "-power_domain <name>",` and `"-cell <name>"`.

cell_halo_y

Specifies the clock halo distance in the y direction. The default value of this property is auto.

The following properties can be used to assign y direction clock halos within CCOpt:

- `cell_halo_y`
- `adjacent_rows_legal`
- `cell_halo_rows`

Only one of these properties is used to determine the clock halo in the y direction. The following rules determine which:

- If `cell_halo_y` is set to a non-auto value, then this defines the y direction clock halo. The properties `adjacent_rows_legal` and `cell_halo_rows` have no effect.
- If `cell_halo_y` is set to auto and `adjacent_rows_legal` is set to a non-auto value then `adjacent_rows_legal` defines the clock halo in the y direction. The property `cell_halo_rows` has no effect.
- If both `cell_halo_y` and `adjacent_rows_legal` are set to auto then `cell_halo_rows` defines the clock halo in the y direction.

See also:
`. adjacent_rows_legal`
`. cell_halo_rows`

Default: `auto`

Optional applicable arguments: `"-clock_tree <name>", "-power_domain <name>",` and `"-cell <name>"`.

check_route_follows_guide

Specifies whether to run diagnostic checks to ensure that clock nets routed by NanoRoute follow route guides. The diagnostic checks compare lengths of estimated routes to the final detailed wiring.

The `check_route_follows_guide_min_length` property lets you specify a minimum route length for diagnostic checks. Only nets longer than this minimum are checked.

Valid values: `true false`

See also:
`. check_route_follows_guide_min_length`

Default: `true`

This global property does not use additional arguments.

check_route_follows_guide_min_length

Minimum net length to perform guide route diagnostics when `check_route_follows_guide` is set to `true`.

Valid values: `double` (in microns) `| auto`

Auto: computed automatically based on design

See also:
`. check_route_follows_guide`

Default: `auto`

This global property does not use additional arguments.

clock_gate_buffering_location

Specifies where CCOpt should place delay buffers relative to clock gates during clock tree synthesis.

If set to `below`, delay buffers will be put underneath clock gates. This is best for power, but may be detrimental to timing.

If set to `above`, delay buffers will be put above clock gates; this is best for timing.

Valid values: `above below`

Default: `above`

Optional applicable arguments: `"-clock_tree <`*`name`*`>"`.

clock_gating_cells

Specifies the clock gates for CTS. If none are specified CCOpt will choose clock gates from the libraries. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names.

If set explicitly, CCOpt will ignore any dont_use settings for the cells specified.

A list of lists can be used to specify separate clock gate sub-lists. All CCOpt transforms will resize clock gates within the clock gate sub-list they are initially part of. If an incoming clock gate is not part of any clock gate sub-list, it will be resized to the first clock gate sub-list.

Different clock gates may be specified for any combination of clock tree and power domain.

Some examples follow:

To specify clock gates for all clock trees and all power domains:

```
set_ccopt_property clock_gating_cells {cgAX* cgBX*}
```

To specify clock gates for a particular clock tree and all power domains:

```
set_ccopt_property -clock_tree clk clock_gating_cells {cgX20 cgX18}
```

To specify clock gates for a particular clock tree and power domain:

```
set_ccopt_property -clock_tree clk -power_domain pd clock_gating_cells {cgX12 cgX8}
```

To specify separate clock gate sub-lists:

```
set_ccopt_property clock_gating_cells {{cgX12A1 cgX8A1} {cgX12A2 cgX8A2}}
```

Valid values: `a list of library cell names, or a list of patterns to expand to library cell names, a list of lists to specify separate clock gate sub-lists`

Default: `{}`

Optional applicable arguments: "`-clock_tree <`*`name`*`>`" and "`-power_domain <`*`name`*`>`".

clock_gating_depth

The depth of gating at and below a clock gate, counted either top-down or bottom-up as determined by the value of the `clock_gating_depth_top_down` property. This property has a value of 0 for any cell that is not a clock gate in a defined clock tree.

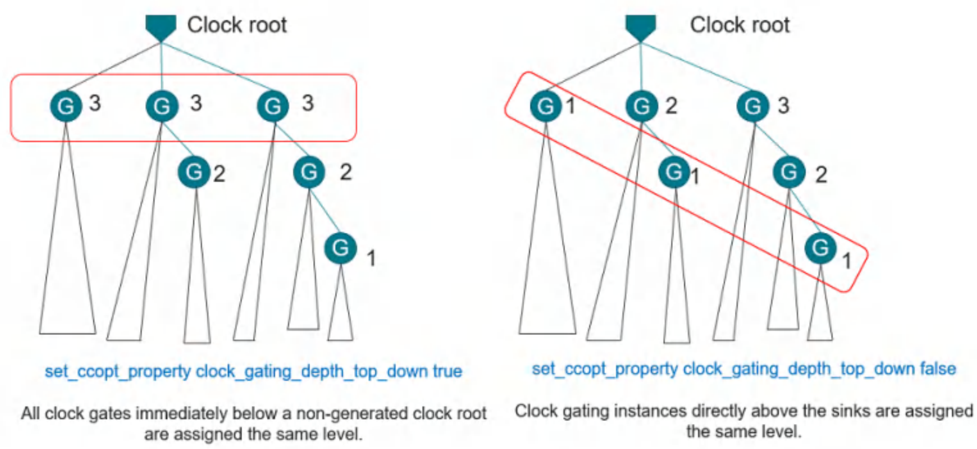
Valid values: `int`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

clock_gating_depth_top_down

Determines whether the `clock_gating_depth` counts gates top-down or bottom-up. In the former case, all gates immediately below a non-generated clock root are assigned the same level. In the latter, all gates immediately above sinks are assigned the same level. In both cases, lower numbers are closer to sinks, and higher numbers closer to roots. Both true and false scenarios for this property are illustrated in the figure below.


Clock root
G 3
G 3
G 3
G 2
G 2
G 1
set_ccopt_property clock_gating_depth_top_down true
All clock gates immediately below a non-generated clock root are assigned the same level.
Clock root
G 1
G 2
G 3
G 1
G 2
G 1
set_ccopt_property clock_gating_depth_top_down false
Clock gating instances directly above the sinks are assigned the same level.

Type: boolean

Default: `true`

This global property does not use additional arguments.

clock_gating_only_optimize_above_flops

Allows CCOpt to work around spurious warnings and failures in downstream tools. If set to true, CCOpt does not optimize clock gating above RAMs, latches, black boxes, or any other clock tree sink that is not a flop. Power optimizations are achieved by transferring flops to a clone of the affected clock gating path. (Note that if CCOpt is run with `-cg_opt off`, flops are not transferred to a cloned clock gating path.)

Setting to false increases power saving opportunities, but may cause spurious formal equivalence issues with some third-party tools.

Type: boolean

Default: `true`

This global property does not use additional arguments.

clock_period

Setting a non-auto value for this property annotates a clock period on this pin (e.g. setting a period of '1ps' will annotate the pin with that period). Normally, clock tree extraction (performed by running `create_ccopt_clock_tree_spec`) will annotate appropriate clock_periods on clock roots. These clock periods will propagate downwards from the clock roots. The `effective_clock_period` property can be queried to see the minimum clock_period that applies at any clock pin.

Valid values: 'auto' or a time value with optional units

Default: `auto`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

clock_source_cells

Specifies the cells available for CTS to size clock sources if the property 'size_clock_source' is set to `true`. If none are specified CCOpt will choose cells from the libraries. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names.

If set explicitly, CCOpt will ignore any don't use settings for the specified cells.

Different cells may be specified for clock trees or power domains. Only clock sources that are buffers, inverters, logic, and clock gating cells with a single output can be resized.

Valid values: `a list of library cell names, or a list of patterns to expand to library cell names`

Default: `{}`

Optional applicable arguments: "`-clock_tree <`*`name`*`>`" and "`-power_domain <`*`name`*`>`".

clock_tree

The clock tree to which this object belongs. Flops do not belong to a clock tree, but their clock pins do.

Valid values: `clock_tree`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <`*`name`*`>`". Required: "`-pin <`*`name`*`>`".

clock_tree_generator_sink_is_leaf

Controls whether the input pin of generator sinks is considered to be a leaf for clock tree routing rules and slew constraints.

Type: boolean

Default: `false`

This global property does not use additional arguments.

clock_tree_source_group

Specifies the clock tree source group to which this clock tree belongs.

Valid values: `list clock_tree_source_group`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-clock_tree <`*`name`*`>`".

clock_trees

A list of clock trees the pin is contained within. This includes parents of generated clock trees and all relevant parents when clock trees overlap.

Valid values: `list clock_tree`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

clone_clock_gates

Specifies whether CCOpt should clone clock gates in an attempt to improve timing. This is bad for power, congestion and utilization, but may in some cases improve clock gate enable timing. The `clock_gate_buffering_location` property controls whether CCOpt always clones (if it is "above"), never clones (if it is "below").

Note: In multi-tap CTS, cloning is enabled by default. This includes cloning across the taps, which is only the cloning necessary to spread the sinks efficiently across the taps.

Set the global property to disable clock gate cloning for all clock trees:

```
set_ccopt_property clone_clock_gates false
```

Set the per-clock tree property to disable clock gate cloning for a particular clock tree:

```
set_ccopt_property clone_clock_gates false -clock_tree clk
```

Valid values: `true false`

See also:
. `clock_gate_buffering_location`
. `clone_clock_logic`

Default: `false`

Optional applicable arguments: "`-clock_tree <name>`".

clone_clock_logic

Specifies whether CCOpt should clone clock logic in an attempt to improve timing. This is bad for power, congestion and utilization, but may in some cases improve clock gate enable timing. The clock_gate_buffering_location property controls whether CCOpt always clones (if it is "above"), never clones (if it is "below").

Set the global property to disable clock logic cloning for all clock trees:

```
set_ccopt_property clone_clock_logic false
```

Set the per-clock tree property to disable clock logic cloning for a particular clock tree:

```
set_ccopt_property clone_clock_logic false -clock_tree clk
```

Valid values: `true false`

See also:
. `clock_gate_buffering_location`
. `clone_clock_gates`

Default: `false`

Optional applicable arguments: "`-clock_tree <name>`".

cloning_inst_name_suffix

If set, the suffix will be used generally for all insts cloned by CTS, else the suffix will default to "clone".

Default: `clone`

This global property does not use additional arguments.

cloning_inst_name_suffix_source_group_assignment

Specifically controls the suffix used for multi tap cloning. If this is set to a non-empty string, that string will be used as a suffix when naming cloned instances created by multi-tap cloning. Otherwise, the name suffix will be controlled by the cloning_inst_name_suffix property.

Default: `{}`

This global property does not use additional arguments.

compatibility_warning

Compatibility warning.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

consider_during_latency_update

Determines whether to consider the sink when updating the IO latencies.

Valid values: `true false auto`

`true`: Consider this pin during latency update.

`auto`: Consider this pin during latency update only if it is an active sink of a non-reporting only skew group. This is the default value.

`false`: Do not consider this pin during latency update. This setting is appropriate for sinks that have a large user pin insertion delay applied.

Default: `auto`

Applicable arguments: "`-pin <`*`name`*`>`". Required: "`-pin <`*`name`*`>`".

consider_power_management

Enables checking of effective power domain for all pins in the clock tree. Setting this property to `false` causes CTS to ignore effective power domain violations when constructing the clock tree.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

constrains

Specifies how this skew group constrains the balancing of sinks during CCOpt.

If set to "default", this skew group will constrain sinks during "ccopt_design -cts" and during the initial global balancing step of a regular "ccopt_design" run.

If set to “all”, then this skew group will constrain both "ccopt_design -cts" and the whole of "ccopt_design”.

If set to "none", this specifies that the skew group will only be used for reporting purposes.

Valid values: default all none

Default: default

Optional applicable arguments: "-skew_group <name>".

cts_clock_gate_movement_limit

Each clock gate is restricted to a Manhattan ball centered on its original location with the CTS flow. The radius of the ball is a multiple of the clock gate height. This property controls the default value of that multiple.

Default: 10

This global property does not use additional arguments.

cts_merge_clock_gates

If set to true, clock gate merging is enabled. If this is false, merging of all clock gates is disabled, including clock gates which may have been cloned by CTS.

This property only impacts 'ccopt_design -cts'. See also property ccopt_merge_clock_gates.

Note: You can also use the name, merge_clock_gates for this property. The un-prefixed version is a synonym for the corresponding cts_* version.

Valid values: true false

Default: false

Optional applicable arguments: "-clock_tree <name>".

cts_merge_clock_logic

If set to true, clock logic merging is enabled. If this is false, merging of all clock logics is disabled, including clock logics which may have been cloned by CTS.

This property only impacts 'ccopt_design -cts'. See also property ccopt_merge_clock_logic.

Note: You can also use the name, merge_clock_logic for this property. The un-prefixed version is a synonym for the corresponding cts_* version.

Valid values: true false

Default: false

Optional applicable arguments: "-clock_tree <name>".

def_lock_clock_sinks_after_routing

If set to true, we will DEF lock clock tree sinks after routing in addition to any clock node locking (fixed).

If set to `soft`, we will DEF lock clock tree sinks after routing in addition to any clock node soft_locking (softFixed).

Valid values: `true false soft`

Default: `false`

This global property does not use additional arguments.

delay_cells

Specifies the delay cells available for CTS. If none are specified CCOpt will not use delay cells. Setting this property to the string 'auto' means that CCOpt will choose delay cells from the libraries to use.
Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names. If set explicitly, CCOpt will ignore any dont_use settings for the cells specified.

Different delay cells may be specified for any combination of clock tree and power domain, or by omitting those arguments a global setting can be applied.

Some examples follow:

To specify delay cells for all clock trees and power domains:

```
set_ccopt_property delay_cells {delayAX* delayBX*}
```

To specify delay cells for a particular clock tree and all power domains:

```
set_ccopt_property -clock_tree clk delay_cells {delayX1 delayX2}
```

To specify delay cells for a particular clock tree and power domain:

```
set_ccopt_property -clock_tree clk -power_domain pd delay_cells {delayX2 delayX3}
```

Valid values: a list of library cell names, or a list of patterns to expand to library cell names, or the string 'auto'

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

detailed_cell_warnings

If set to true, CCOpt outputs detailed cell warning diagnostics when it encounters issues with library cell selection, power domains and/or signal levels.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

effective_clock_halo_x

This read only property shows the x component of the clock halo The units are um. The following properties can be used to assign x direction clock halos within CCOpt:

- `cell_halo_x`
- `cell_density`
- `cell_halo_sites`

See also:

- `effective_clock_halo_x_source`
- `effective_clock_halo_y_source`
- `effective_clock_halo_y`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

effective_clock_halo_x_source

This read only property shows which property defines the x component of the clock halo. The following properties can be used to assign x direction clock halos within CCOpt:

- `cell_halo_x`
- `cell_density`
- `cell_halo_sites`

See also:

- `effective_clock_halo_x`
- `effective_clock_halo_y_source`
- `effective_clock_halo_y`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

effective_clock_halo_y

This read only property shows the x component of the clock halo. The units are um. The following properties can be used to assign y direction clock halos within CCOpt:

- `cell_halo_y`
- `adjacent_rows_legal`
- `cell_halo_rows`

See also:

- `effective_clock_halo_x_source`
- `effective_clock_halo_y_source`
- `effective_clock_halo_x`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

effective_clock_halo_y_source

This read only property shows which property defines the y component of the clock halo. The following properties can be used to assign y direction clock halos within CCOpt:

- cell_halo_y
- adjacent_rows_legal
- cell_halo_rows

See also:

- effective_clock_halo_x
- effective_clock_halo_y
- effective_clock_halo_x_source

Read-only: This property cannot be modified by set_ccopt_property or unset_ccopt_property.

Applicable arguments: "-inst <name>". Required: "-inst <name>".

effective_clock_period

The effective_clock_period for a pin corresponds to the clock_period property for the pin, if the user explicitly sets the clock_period property at the pin. Otherwise, the effective_clock_period is computed by taking the minimum clock_period of all clocks that are present at the pin and that possess a clock_period annotated at their root pin.

A value of 'auto' for this property indicates either that the pin is not in the clock tree network, or that no clock_period annotations were found, either at the pin or for any clocks present at the pin.

Read-only: This property cannot be modified by set_ccopt_property or unset_ccopt_property.

Applicable arguments: "-pin <name>". Required: "-pin <name>".

effective_clock_period_sources

This property lists the set of pins that define the effective_clock_period at a given pin (i.e. which clock_period property settings defined this pin's effective_clock_period).

Read-only: This property cannot be modified by set_ccopt_property or unset_ccopt_property .

Applicable arguments: "-pin <name>". Required: "-pin <name>".

effective_sink_type

Indicates how CCOpt will treat a given pin, taking into account both its implicit_sink_type, and any sink_type settings. Setting a non-default value for the sink_type property will override the implicit_sink_type property.

Read-only: This property cannot be modified by set_ccopt_property or unset_ccopt_property.

Applicable arguments: "-pin <name>". Required: "-pin <name>".

enable_all_views_for_io_latency_update

If set, enables all views before updating IO latencies, and restores the set of enabled views after.

Valid values: true false

Default: true

This global property does not use additional arguments.

enable_nanoroute_layer_check_failure

Before running CCOpt, a check is made to verify that the routing configuration selects preferred layer ranges that are wholly contained within the routing layer range specified in the NanoRoute configuration. If the check fails then an error message is emitted.

This validation ensures that the estimated routes used during CTS can in principle be realised during clock net routing. Estimates constructed using routing layers outside of the range specified in the NanoRoute configuration will likely cause a routing correlation issue.

If this property is set to true, and the check fails, CCOpt will not continue running.

This check can be disabled by setting this property to false. In this case the error messages are still emitted but CCOpt will continue to run.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

error_on_problematic_slew_violating_nets

Post slew-fixing, CCOpt will identify nets with slew violations that probably can not be fixed and return an error. Set the property to false to bypass this error. Common problems include setting a net that requires buffering as constant. Review your design carefully before setting this property, as it may lead to poor results.

Type: `boolean`

Default: `false`

This global property does not use additional arguments.

error_on_problematic_slew_violating_nets_max_printout

Early on on the CCOpt flow nets that are marked as e.g. don't touch but have violations are identified. Set this property to control how many nets with each kind of violation will be printed. Setting the property to -1 will print all violating nets.

Type: `integer`

Default: `5`

This global property does not use additional arguments.

exclusive_sinks_rank

The rank of this skew_group. Sinks will be considered as a sink of all skew groups with a rank equal to the greatest rank of all skew groups of which they are defined as sinks.

Valid values: `integer`

Default: `0`

Optional applicable arguments: "`-skew_group <name>`".

exit_if_stage_delay_sigma_target_over_constrained

If set, allows any max stage delay sigma target to be set. If the target is too low, it is likely to lead to runtime problems.

Default: `true`

This global property does not use additional arguments.

extract_balance_multi_source_clocks

Causes `create_ccopt_clock_tree_spec` to set up a clock tree source group for SDC clocks with multiple source pins. If this property is set to `active`, `create_ccopt_clock_tree_spec` defines one clock tree for each source pin and then uses the `create_ccopt_clock_tree_source_group` command to collect those clock trees together, so that CTS can allocate sinks among clock trees. If this property is set to `inactive` or `true`, the source groups are also created during `create_ccopt_clock_tree_spec`, but they will be defined as `inactive`.

Only active source groups will take part in sink allocation.

By default, it is set to `false`, which means that no source groups are extracted from SDC multiple pin clocks.

Default: `false`

This global property does not use additional arguments.

extract_clock_generator_skew_group_name_prefix

This property controls the skew group name prefix used for skew groups generated to balance generator flops with their adjacent flops. Default is `"_clock_gen"`. The default has a start underscore at the beginning to cause listings of skew groups ordered by name to collect such skew groups together at the end of a list.

Type: `string`

Default: `_clock_gen`

This global property does not use additional arguments.

extract_clock_generator_skew_groups

This property will cause the `create_ccopt_clock_tree_spec` command to create skew groups for sequential generators and their adjacent registers. Such skew groups will be specified with the same highest rank so that they can be balanced from the other normal skew groups that share some sinks of them. The adjacent registers of a generator are registers that have a datapath timing path to talk with the generator directly. When this property is set to `true`, one skew group will be created per sequential generator instance, master clock and generated clock tree triple. The resulting skew groups will by default be named in the pattern:

```
_clock_gen_<master_clock_name>_<generator_local_name><_optional_number>/<constraint_mode_name>.
```

For example, for a pair of generators, with the same local name "`reg_clkgen`", CCOpt creates generated clock trees from the same master clock named "`fclk`" in a constraint mode named "`func`" the skew groups emitted into the clock tree specification file would be named:

```
_clock_gen_fclk_reg_clkgen_1/func
_clock_gen_fclk_reg_clkgen_2/func
```

The prefix for the names of such skew groups is controlled by the `extract_clock_generator_skew_group_name_prefix` CCOpt property and defaults to "`_clock_gen`" (the start underscore is used to group such skew groups at the end of any skew group listing ordered by name).

Type: boolean

Default: true

This global property does not use additional arguments.

extract_clock_group_skew_group_name_prefix

This property controls the skew group name prefix used for skew groups derived from SDC set_clock_group and similar assertions. Default is "_clock_group". The default has a start underscore at the beginning to cause listings of skew groups ordered by name to collect such skew groups together at the end of a list.

Type: string

Default: _clock_group

This global property does not use additional arguments.

extract_cts_case_analysis

During create_ccopt_clock_tree_spec, set case_analysis properties for any clock tree cell inputs determined to have constant values. CTS considers timing through cells according to the case_analysis properties of the cell's input pins. If this property is set to false, create_ccopt_clock_tree_spec will not set any case_analysis properties and CTS will assume worst-case timing.

Type: boolean

Default: true

This global property does not use additional arguments.

extract_faster_sdc_clocks_as_clock_trees

If set to true, this property causes create_ccopt_clock_tree_spec to create a separate clock tree for SDC clocks that are faster than the parent clock.

Type: boolean

Default: true

This global property does not use additional arguments.

extract_network_latency

When true, this property causes create_ccopt_clock_tree_spec to use the clock network latency as the insertion delay target for the representative clock tree. An SDC command of the form set_clock_latency <clock> <delay>; will result in the create_ccopt_clock_tree_spec command setting the -target_insertion_delay on the corresponding create_ccopt_skew_group command.

Type: boolean

Default: true

This global property does not use additional arguments.

extract_no_exclude_pins

When the clock tree extraction algorithm finds a pin on the clock tree that has no outgoing timing arcs (for example a design output, black box input, or flop D input), it will normally exclude that pin from the clock tree. The pin will still be clocked, but the clock tree will include as a sink a point higher up in the clocking structure that drives this excluded pin, rather than the pin itself. If this property is `true`, the clock tree will instead include as sinks all pins with no outgoing timing arcs, which may lead to the extraction algorithm following convoluted paths in the clock tree with the subsequent effect that CCOpt will attempt to meet the clock tree's slew target on these paths.

Type: `boolean`

Default: `false`

This global property does not use additional arguments.

extract_pin_insertion_delays

If set to `true`, `create_ccopt_clock_tree_spec` will extract pin insertion delay settings from SDC `set_clock_latency` assertions on clock sinks, if the clock network latency specified for the sink differs from the network latency specified for the corresponding clock.

Type: `boolean`

Default: `true`

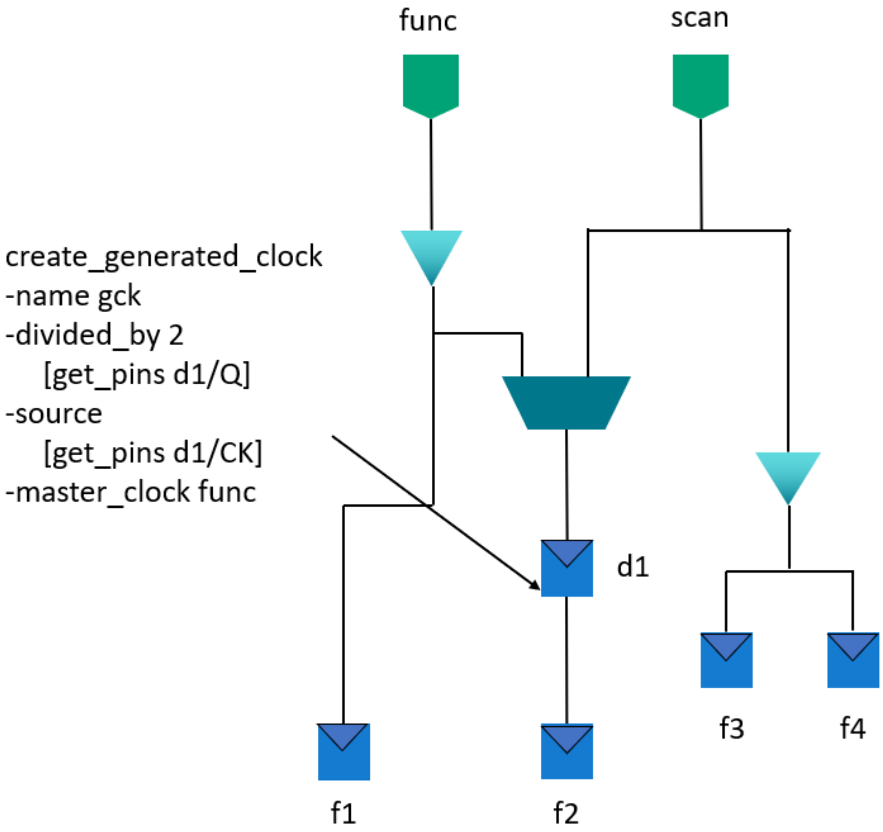
This global property does not use additional arguments.

extract_skew_group_sinks_at_clock_node_timing_endpoints

If set, `create_ccopt_clock_tree_spec` will emit `skew_group` sinks at clock node inputs which are timing endpoints for a SDC clock. If not set, then `create_ccopt_clock_tree_spec` will emit `skew_group` ignores at such inputs and they will not be considered for balancing. Typical examples of clock node timing endpoints include flops that are used as generators for one clock (typically the functional clock) but are strictly sinks for another clock (typically a scan clock). Usually such endpoints are fine to be balanced as skew group ignore pins since there are normally relatively few of them and they have loose timing constraints. In addition relatively few designs have the correct logic structures above such timing endpoints to balance them correctly as sinks, hence the default value of `false`.

For example, in the image below, d1 is the generator for clock func and a sink (timing endpoint) for clock scan.

If this property is set to `false` (default), then skew group scan includes f3 and f4. If it is set to `true`, skew group scan includes d1, f3, and f4.



Type: `boolean`

Default: `false`

This global property does not use additional arguments.

extract_source_latency

When `true`, this property causes `create_ccopt_clock_tree_spec` to use the clock source latency as the source delay for the representative clock tree. An SDC command of the form `set_clock_latency -source <clock> <delay>` will result in the `create_ccopt_clock_tree_spec` command setting the `source_latency` property.

Type: `boolean`

Default: `true`

This global property does not use additional arguments.

extract_through_multi_output_cells_with_single_clock_output

This property controls whether and how `create_ccopt_clock_tree_spec` will extract through cells that have multiple outputs but where only one of those outputs has a clock emanating from it; call these types of instance multi-output single-clock instances.

If the feature is disabled, spec creation will not trace through multi-output single-clock instances. The resulting clock spec will specify a generated clock tree on the clock-carrying outputs of every multi-output instance. Clock-carrying inputs that are connected to each clock output by way of timing arcs will be generator inputs to the corresponding generated clock tree. The presence of these generated clock trees may impact CTS operations on the instance such as sizing, moving, and cloning.

If the feature is enabled, the resulting clock spec will include trace_through_to settings for multi-output single-clock instances. These settings will allow the instance to be treated as a normal clock instance, most commonly as a clock logic. In this case there will be no need for the generated clock tree on the clock-carrying output. The treatment of instances with multiple clock-carrying outputs is unchanged: on such an instance each clock-carrying output will take a generated clock trees, as before.

The feature may be enabled conditionally, such that it is only applied to multi-output single-clock instance which are combinational. In this case, multi-output single-clock instances which are sequential will continue to take a generated clock tree on the clock output.

The possible values for this property are as follows:

- `false` - The feature is completely disabled.
- `true` - The feature is enabled for all cell types; both combinational and sequential.
- `combinational` - The feature is enabled for combinational cells, but disabled for sequential cells.

Valid values: `false true combinational`
Default: `combinational`

This global property does not use additional arguments.

extracted_from_clock_name

This contains the name of the SDC clock that this skew group has been created to represent the balancing constraints in CCOpt.

Valid values: `string`

Default: `{}`

Optional applicable arguments: "`-skew_group <name>`".

extracted_from_constraint_mode_name

This contains the name of the constraint mode that this skew_group has been created to represent the balancing constraints in CCOpt.

Valid values: `string`

Default: `{}`

Optional applicable arguments: "`-skew_group <name>`".

extracted_from_delay_corners

This contains the delay corners associated with this `skew_group` that has been created to represent the balancing constraints in CTS.

Valid values: `string`

Default: `{}`

Optional applicable arguments: "`-skew_group <name>`".

filter_cell_lists_for_frequency_dependent_max_cap_constraints

This property can be used to enable or disable filtering out cells that have low max cap values at some frequencies used in the design. CCOpt will then not use those cells.

Valid values: `true` or `false`
Default: `true`

This global property does not use additional arguments.

final_cell

The library cell to use for the H-tree sinks.

Valid values: `string`
Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

flexible_htree

Returns the name of the flexible H-tree, if any, associated with the given object. This property is read-only, which means it cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

flexible_htree_placement_legalization_effort

The legalization effort for finding placement unblocked points on the synthesis grid for flexible H-trees. High placement legalization effort can avoid having to relax placement constraints when implementing H-trees but may lead to increased runtime.

Valid values: low, high
Default: `low`

This global property does not use additional arguments.

force_all_virtual_delay_updates

If this is set to true, any call to set or reset a virtual delay, even if that call doesn't change the stored value, will push through into the timing engine. Normally we'll avoid calling into CTE if the annotation is left unchanged - saves runtime.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

force_clock_objects_to_propagated

Determine whether or not to put the propagated_clock assertion on all clocks within ccopt_design.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

force_clock_tree

Can be used to make sure a clock tree sink pin gets driven by a particular clock tree. The clock tree can alternatively be specified as the tree's root pin, in case extraction renumbers the clock trees defined from a single SDC clock. By default CCOpt automatically selects appropriate pins.

Valid values: `clock_tree | list of pins`

Default: `{}`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

force_update_io_latency

If false (default), I/O latencies are updated if and only if all SDC clocks are in ideal mode. If true, run I/O latency updates even when clocks are in propagated mode.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

frequency_dependent_max_cap_usability_check_max_cap_fanout_factor

The minimum number of instances of the same type of cell a cell needs to be able to drive at all used frequencies.

Valid values: `numbers > 1.0`

Default: `4`

This global property does not use additional arguments.

generated_by_sinks

The list of parent clock tree sinks for which timing to this clock tree should be considered, if any.

Valid values: `list pin`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-clock_tree <name>`".

htree_sinks

Specifies H-tree sinks as approximate rectangular areas for locations of final cells (given by -final_cell) or pins to wire to.

Valid values: `list {pin | {xmin ymin xmax ymax}}`
*Defau*lt: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

hv_balance

Specifies whether horizontal and vertical wires can only be balanced against other wires of the same orientation (true), or whether any wire can be balanced against any other wire (false).

Valid values: `true false`
Default: `true`

Optional applicable arguments: "`-flexible_htree <name>`".

ideal_net

If set the clock tree timing engine will consider this net as ideal.

Valid values: `true false`
Default: `false`

Required argument: "`-net <name>`"

ignore_pins

A list of ignore pins for this skew group.

Valid values: `list pin`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-skew_group <name>`".

ignore_problematic_skew_as_result_of_dont_touch_nets

If set to `true`, sinks directly connected to nets that are causing unfixable skew problems will be ignored for skew balancing.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

image_directory

Specifies the name of the directory to which the images are written.

The images are PNG files describing the tree as generated by the H-tree synthesis algorithm

White: Unobstructed edges of the synthesis grid

Red: Grid points that are blocked for trunk cell placement in all modules

Orange: Grid points that are blocked for final cell placement in all modules

Red orange: Grid points that are blocked for trunk and final cell placement in all modules

Yellow circle: The grid point of the source

Yellow crosses: Candidate grid points of H-tree sinks

Yellow rectangle: The sink area containing target grid point candidates of H-sinks, adjusted to the synthesis grid

Brown: If specified, the sink grid box snapped to the synthesis grid

Green/blue: The edges of the synthesized H-tree

Purple: H-tree repeaters

Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

implicit_sink_type

Indicates the type of sink that CCOpt classified this pin as. Note that the sink_type property can override these internal classifications.

Possible values are:

`exclude` Indicates that this pin is a sink which represents a non-clock pin.
`ignore` Indicates that CCOpt has determined not to search for more clock tree through this pin. Additionally, this pin will not be balanced.
`stop` Indicates that CCOpt has determined not to search for more clock tree through this pin.

An empty value for this property indicates either that this pin is either not a sink, or that it is a sink that is not implicitly exclude or ignore or stop.

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

include_source_latency

Specifies whether clock tree source latency should be included when timing the skew group.

Valid values: `true false`

Default: `false`

Optional applicable arguments: "`-skew_group <name>`".

insertion_delay

Specifies the amount of insertion delay under this pin. Clock tree synthesis will attempt to make the insertion delay to this pin less than that to other sinks in the same skew group by this amount if a positive value is set. A negative value should be used if you would like the insertion delay to this pin to be greater than that to other sinks. The value 'auto' means there is no insertion delay offset for the pin.

Valid values: `double | ignore | min | auto`

Default: `auto`

Applicable arguments: "`-delay_corner <name>`", "`-pin <name>`", "`-early`", "`-late`", "`-rise`", "`-fall`", "`-max`" and "`-min`". Required: "`-pin <name>`".

insertion_delay_sources

The amount of insertion delay under this pin, broken down by the source of the pin insertion delay (PID). The value of this property is a Tcl dictionary of PID values, keyed on the source of that PID. This provides an 'exploded' view into the total pin insertion delay that is present, categorizing the various contributions by their origin.

Valid Tcl dict keys:

`user` Asserted by user Tcl scripting or restored from an older DB.
`sdc` Extracted from SDC set_clock_latency by spec creation.
`ilm` Records the mean clock latency of the ILM clock network that is rooted at this pin.
`useful_skew` A useful skew generated by pre-CTS optimization.
`incremental` An incremental offset, used to specify useful skews to incremental CTS.
`total` The formal sum of the above contributions. This value is the 'overall' pin insertion delay accessible via the insertion_delay property.

Valid Tcl dict values: `double | ignore | min | auto`

Reading the property returns a dict value that exposes the current state of the PID, recording all the PID sources and the associated PID value of each one. Sources whose value are auto are omitted. If there are no non-auto sources, the returned dict is empty. For convenience, the total is always included in the dict if it is non-auto.

Writing a dict value to the property will update each of the specified sources with the specified PID value. The total PID value will then be updated. Sources in the dict that are assigned a value of 'auto' will be reset. Sources omitted from the dict are left unchanged from their current value:
`assigning an empty dict is a no-op.`

Resetting the property will clear all the sources and the total back, resetting them to the default value of 'auto', thereby completely removing the PID from the pin.

Valid values: A Tcl dictionary of PID values, keyed on PID source.

Default: `{}`

Applicable arguments: "`-delay_corner <`*`name`*`>`", "`-pin <`*`name`*`>`", "`-early`", "`-late`", "`-rise`" and "`-fall`". *Required*: "`-pin <`*`name`*`>`".

inst_name_prefix

The name prefix of instances created by CTS. The default value is "CTS". The default names of instances are CTS_*.

Valid values: string

Default: CTS

This global property does not use additional arguments.

inverter_cells

Specifies the inverter cells available for CTS. If none are specified CCOpt will choose inverters from the libraries. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names. If set explicitly, CCOpt will ignore any don't use settings for the cells specified.

Different inverter cells may be specified for any combination of clock tree and power domain.

To use different inverters for each net type set the `top_inverter_cells` and `leaf_inverter_cells` properties.

Some examples are provided below:

To specify inverter cells for all clock trees and all power domains:

```
set_ccopt_property inverter_cells {invAX* invBX*}
```

To specify inverter cells for a particular clock tree and all power domains:

```
set_ccopt_property -clock_tree clk inverter_cells {invX20 invX18}
```

To specify inverter cells for a particular clock tree and power domain:

```
set_ccopt_property -clock_tree clk -power_domain pd inverter_cells {invX12 invX8}
```

Valid values: a list of library cell names, or a list of patterns to expand to library cell names

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

inverting

Specifies whether the flexible H-tree will invert its input or not.

Valid values: `true, false`

Default: false

Optional applicable arguments: "`-flexible_htree <`*`name`*`>`".

is_active

Determines whether the clock tree source group is active or not. Only active source groups will take part in sink allocation.

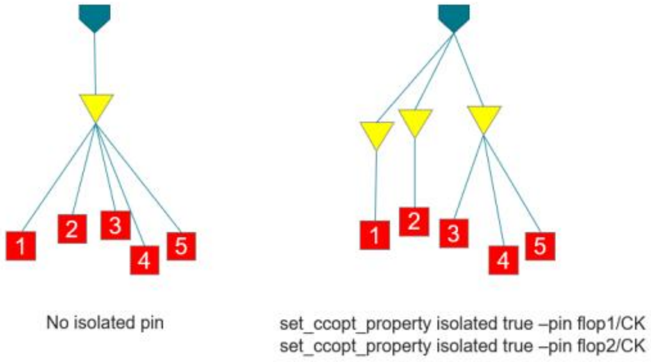
Valid values: `true false`.

Default: `true`

Optional applicable arguments: "`-clock_tree_source_group <name>`".

isolated

Indicates whether this is an isolated pin. CTS does not cluster isolated pins with any other pins. This is shown in the figure below.


1
2
3
4
5
No isolated pin
1
2
3
4
5
set_ccopt_property isolated true –pin flop1/CK
set_ccopt_property isolated true –pin flop2/CK

Valid values: `true false`

Default: `false`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

is_genus_clock_gate

Set by the tool in the iSpatial flow to indicate that this integrated clock gating instance was added by Genus clock gating.

Default: `false`

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

is_sdc_clock_root

Specifies whether the given pin is the root (source pin) of an SDC clock. The create_ccopt_clock_tree_spec populates this property with the location of the SDC clock root (source) pins.

This property controls the behavior of clock tree definition commands, `create_ccopt_clock_tree` and `create_ccopt_generated_clock_tree`, when the `-stop_at_sdc_clock_roots` argument is specified. In such a case, pins and ports for which this property is true will be treated as being SDC clock root pins.

Valid values: `true false`

Default: `false`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

layer_density

The assumed layer density used to compute the parasitics for timing estimates of H-tree nets during H-tree synthesis based on the non default rule for top nets.

Valid values: Any float in the range 0 to 1.
Default: 1

Optional applicable arguments: "`-flexible_htree <name>`".

leaf_buffer_cells

Specifies the buffer cells available for CTS to use on leaf nets. If none are specified, CCOpt will use the same buffers as on trunk nets - as specified in the `buffer_cells` property. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names. If set explicitly, CCOpt will ignore any don't use settings for the cells specified.

Different leaf buffer cells may be specified for any combination of clock tree and power domain.

Valid values: a list of library cell names, or a list of patterns to expand to library cell names

Default: `{}`

Optional applicable arguments: "`-clock_tree <`*`name`*`>`" and "`-power_domain <`*`name`*`>`".

leaf_inverter_cells

Specifies the inverter cells available for CTS to use on leaf nets. If none are specified, CCOpt will use the same inverters as on trunk nets - as specified in the `inverter_cells` property. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names. If set explicitly, CCOpt will ignore any don't use settings for the cells specified.

Different leaf inverter cells may be specified for any combination of clock tree and power domain.

Valid values: a list of library cell names, or a list of patterns to expand to library cell names

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

legalized_on_clock_spine

The name of the clock spine to which the specified pin has been assigned by CCOpt, if any. Pins which have been aligned with a clock spine report the name of that spine in this property. Clock spine cell placement inside CCOpt CTS will record the clock spine that it chose to place the pin on (or near, in the case of placement failure).

When setting the property if a clock spine name is specified which does not correspond to an existing clock spine then the command will fail.

When setting the property if a clock spine name is specified which is not in the corresponding list in the lock_on_clock_spine property then the clock spine name will be added to that property as well.

Valid values: `string`

Default: `{}`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

load_capacitance_cells

Specifies the load capacitance cells available for CTS. CTS will use cells from this collection for load capacitance optimizations. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names.

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

library_trace_through_to

Clock tree definition will, by default, not continue through certain types of cell arc (for instance, the clock to Q arc in a DFF). This property allows you to override the default behavior by specifying the output library pin to which the clock tree should propagate, when it arrives at an instance of the given input library pin. The specified output library pin must be another pin on the same library cell as the given input pin. The configured value is applicable to all instances of the library cell in the netlist.

The output pin may be specified either by its fully qualified name (i.e. inclusive of the library cell name), or else simply by its local (cell-relative) name.

Note: If both trace_through_to and `library_trace_through_to` are applicable at a given netlist instance pin, the `trace_through_to` value takes precedence.

Valid values: `lib_pin`

Default: {}

Optional applicable arguments: "`-lib_pin <name>`".

lock_on_clock_spine

A set of clock spine names restricting where the pin should be located. If the set contains more than one item, the pin may be placed on any of the specified clock spines. Clock tree nets above pins locked on a clock spine pins will automatically be routed as top nets.

When setting the property if a clock spine name is specified which does not correspond to an existing clock spine then the command will fail.

When setting the property if the list of clock spine names no longer contains the clock spine referred to by the property `legalized_on_clock_spine` for this pin then that property will be unset.

Valid values: `list string`

Default: `{}`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

log_precision

The number of significant figures printed in floating point numbers written to the log

Type: integer

Default: `3`

This global property does not use additional arguments.

log_special_case_cell_selections

When specified, CCOpt will provide additional log information on the lists of cells that it will consider using when determining if a cell can be resized. This information is only provided for the more "specialized" cases.
Specialized cases include logic cells and any cells marked (either by the user or by an internal constraint) as dont touch.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

logic_cells

Specifies the clock logics for CTS. If none are specified CCOpt will choose clock logics from the libraries. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names. If set explicitly, CCOpt will ignore any dont_use settings for the cells specified.

Different logic cells may be specified for any combination of clock tree and power domain.

Some examples follow:

To specify logic cells for all clock trees and all power domains:

```
set_ccopt_property logic_cells {and* mux*}
```

To specify logic cells for a particular clock tree and all power domains:

```
set_ccopt_property -clock_tree clk logic_cells {andX20 andX18}
```

To specify logic cells for a particular clock tree and power domain:

```
set_ccopt_property -clock_tree clk -power_domain pd logic_cells {andX12 andX8}
```

Valid values: `a list of library cell names, or a list of patterns to expand to library cell names`

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

long_path_removal_cutoff_id

A per-skew-group control for the "Artificially removing long paths" algorithm. For a skew group where it is set, the paths with the greatest insertion delay will be artificially shortened by means of a pin insertion delay, preventing CTS from increasing the average ID towards these long paths, which are assumed to be acceptable outliers.

The property gives the ID beyond which all sinks will be removed as outliers.

Valid values: Numbers in the range `0-inf`.

Default: auto

Optional applicable arguments: "`-skew_group <name>`".

long_path_removal_percentile

A per-skew-group control for the "Artificially removing long paths" algorithm. For a skew group where it is set, the paths with the greatest insertion delay will be artificially shortened by means of a pin insertion delay, preventing CTS from increasing the average ID towards these long paths, which are assumed to be acceptable outliers.

The property gives the percentile of the sink with the maximum acceptable ID. Sinks with IDs longer than this maximum ID will have insertion delay offsets set so that the sink ID becomes equal to the new maximum. In a skew group with a 100 sinks, setting this property to 0.95 would select the sink with the 5th longest delay (the 95th sink) to be the new maximum, with sinks 96, 97, 98, and 99 having IDs now equal to the 95th.

Valid values: Numbers in the range 0-1.0.

Default: `auto`

Optional applicable arguments: "`-skew_group <name>`".

manage_power_management_illegalities

If this property is set, the CTS algorithm will work around power management illegalities in the clock tree, as opposed to failing with an error when it encounters them. This allows the clock tree to be synthesized, but any power management illegalities will remain in the exported design.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

max_buffer_depth

Constrains CTS to ensure that at most this many buffers are present along each path from source to sink in the skew group.

Default: `auto`

Optional applicable arguments: "`-skew_group <name>`".

max_cell_height

When selecting cells for use, CTS will filter the selection to remove all buffers, inverters and clock gates taller than this many row heights. This helps quickly eliminate very large cells that are unlikely to be useful.

Default: 3

This global property does not use additional arguments.

max_clock_cell_count

The maximum number of clock cells taken from a library before a warning is printed. Parts of CCOpt can exhibit poor runtime performance when the list of possible library cells to consider becomes too large. A warning is printed when the list of cells exceeds the limit set in this property. It may indicate that the number of cells specified should be reduced. If done properly, this will have little effect on results.

Valid values: Any positive number or 0 to disable.

Type: integer

Default: `15`

This global property does not use additional arguments.

max_driver_distance

When specified, the software ignores DRVs and uses the given value as the maximum length of the nets connecting the H-tree drivers.

Valid values: `float`

Default: *auto*

Optional applicable arguments: "`-flexible_htree <name>`".

max_fanout

The maximum fanout at any point in the clock tree.

Valid values: integer ranged between `2` and `1000` inclusive

Default: `100`

Optional applicable arguments: "`-lib_pin <name>`".

maximum_insertion_delay

For instances in the clock tree, specifies a maximum desired insertion delay beneath that instance. For instances not in the clock tree, this property has no effect.

Because it relies on the existence of the clock tree, this property can only be set after clock trees have been created.

Valid values: `double`

Default: `-1.79769e+308`

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

max_root_distance

If specified, the software ignores DRVs and uses the given value as the maximum length of the net connecting the root and the first driver of the H-tree. This value overrides the property `max_driver_distance` of this net and can only be specified if `create_ccopt_flexible_htree -max_driver_distance` is also specified.

Valid values: `float`

Default: *auto*

Optional applicable arguments: "`-flexible_htree <name>`".

max_source_to_sink_net_length

The maximum routing length in microns between driving source pin and driven sink pin on each net that clock tree synthesis should observe. This constraint can be applied to either a pin, a clock tree, or a net type. By default (if this property is not set) no explicit clock tree net length constraint is enforced. However, other clock tree constraints such as maximum slew (transition) and maximum capacitance will indirectly limit the maximum net length.

Valid values: `double`

Default: `auto`

Optional applicable arguments: "`-clock_tree <name>`", "`-lib_pin <name>`" and "`-net_type <name>`".

max_source_to_sink_net_resistance

The maximum routing resistance (in resistance library units) between source and sink that clock tree synthesis should observe.

Valid values: `double`

Default: `auto`

This global property does not use additional arguments.

mixed_fanout_net_type

Controls how CCOpt considers nets that have fanout consisting partially but not entirely of sinks. By default, having any sinks, for example DFFs, will make the net be considered leaf, but when set to trunk, a net has to drive only sinks to be considered as leaf. For example, a net driving a clock gate and a DFF would no longer count as a leaf net.

Valid values: `leaf trunk`

Default: leaf

This global property does not use additional arguments.

mode

Specifies the driver insertion mode for the H-tree.

If set to `drv`, the algorithm inserts drivers to avoid DRVs.
If set to `distance`, the properties `max_driver_distance` and `max_root_distance` determine the maximum net lengths allowed. Transitions and delays are not computed in this case.

Valid values: `drv distance`

Default: `drv`

Optional applicable arguments: "`-flexible_htree <name>`".

move_clock_gates

If this property is set, the CTS algorithm will move clock gates that appear in the clock tree. Setting this property may cause the clock tree to have a lower insertion delay, but might break datapath timing. During optimization, this is not a problem, because the timing will be automatically recovered during the optimization process. If this property is false, CTS permits small movements of ICGs for legalization.

Type: boolean

Default: `true`

This global property does not use additional arguments.

move_logic

If this property is set, the CTS algorithm will move logic that appears in the clock tree. "Logic" does not include clock gates, buffers, and inverters in a clock tree, which are always moved unless they are locked, or clock generators that are above the root of the clock tree. Usually, this will affect multiplexers used for selecting one of a number of clocks, or for switching between a test clock and the main clock. Setting this property may cause the
clock tree to have a lower insertion delay, but might break datapath timing. During optimization, this isn't a problem, because the timing will be automatically recovered during the optimization process.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

move_middle_cell_first_when_adding_wire_delay

When moving three cells to add wire delay, move middle cell first, followed by the other added cell and then lastly the parent or child of the middle cell.

Default: `false`

This global property does not use additional arguments.

net_name_prefix

The name prefix of instances/nets created by CTS. The default value is "CTS". The default names of nets are CTS, CTS_1, CTS_2... ...The default names of instances are CTS_*.

Valid values: `string`

Default: `CTS`

This global property does not use additional arguments.

net_type

For a net, return the CTS net type. It will be one of the following:

- `"top"` - A clock net that has fanout above routing_top_transitive_fanout.
- `"trunk"` - A CTS clock net that is not a "top" net nor a "leaf" net.
- `"leaf"` - A CTS clock net that has fanout to only clock sinks.

- `"unknown"` - A CTS clock net that has not been analyzed.
- `""` - Not a CTS clock net.

See also the `get_ccopt_clock_tree_nets` command.

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-net <`*`name`*`>`". Required: "`-net <`*`name`*`>`".

net_unbufferable

This property contains a list of reasons why CCOpt was not able to buffer the clock net attached to the specified pin.

Valid values: `string`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

node_type

For a pin, return the CTS node type. One of the following will be returned:

`""`, `"sink"`, `"source"`, `"generator"`, `"inverter"`, `"buffer"`, or `"clock_gate"`.

A value of "" indicates the pin is not in a clock tree.

See also the `get_ccopt_clock_tree_sinks` and `get_ccopt_clock_tree_cells` commands.

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

omit_symmetry

Controls the omission of symmetry features to balance a flexible H-tree.

Symmetry drivers are drivers that are added to balance pin capacitances at branch points. For example, if buffer pairs are inserted at branch points, one of these buffers may not drive any fanout and is inserted to match other buffer pairs at the same level of the flexible H-tree. Omitting symmetry drivers can reduce the power of the H-tree but increase its skew. Drivers that are added to terminate symmetry branches do not count as symmetry drivers and are never omitted.

Similarly, symmetry branches are branches that are needed to balance the wire load at branch points. They are added to match other branchpoints at the same level of the flexible H-tree. By default, symmetry branches and drivers are added. Symmetry branches are always terminated with drivers, which are not counted as symmetry drivers.

Possible values for this property:

`false` Add symmetry branches and drivers
`true` Omit both symmetry drivers and branches
`drivers` Omit symmetry drivers
`branches` Omit symmetry branches
`{drivers branches}` Omit both symmetry drivers and branches
`{}` Add symmetry branches and drivers

Default: auto

Optional applicable arguments: "-flexible_htree <name>".

opt_ignore

Specifies whether CCOpt will balance this clock tree. If set to true, CCOpt will not balance or optimize this clock tree. The default is false.

Valid values: true false

Default: false

Optional applicable arguments: "-clock_tree <name>".

original_names

Specifies for a clock gate or clock logic which has been merged or is a clone, a list of names from the original netlist which are equivalent to the clock gate/clock logic. For example:

- If A and B are merged to form C then original_names for C is { A B }.
- If D_clone is a clone of D then original_names for D_clone is { D }.
- If E is a clone of C then original_names for E is { A B } (remembering C was a merger of A and B).

Valid values: list string

Default: {}

Applicable arguments: "-inst <*name*>". Required: "-inst <*name*>".

override_minimum_max_trans_target

Specifies that CCOpt should allow any slew target to be used, even if it is likely to lead to runtime problems. By default, CCOpt computes a minimum slew target which should not lead to runtime problems, and does not allow any slew target to be set lower than this minimum. This property overrides that check, and allows any slew target to be used.

Valid values: true false

Default: false

This global property does not use additional arguments.

override_minimum_skew_target

Specifies that CCOpt should allow any skew target to be used, even if it is likely to lead to poor results. By default, CCOpt computes a minimum skew target which should not lead to excessive buffering, and does not allow any skew target to be set lower than this minimum. This property overrides that check, and allows any skew target to be used.

Valid values: true false

Default: false

This global property does not use additional arguments.

override_vias

When specified, this property defines the vias to be used in RC extraction from routing estimates. The listed vias will be used in place of those configured on the routing rules for the clock network. The order of the list is irrelevant. The vias may also be overridden for each non-default rule (NDR). In this case, the name of the NDR is given as the first element in the list with subsequent entries being the via names. Multiple NDRs can be specified, e.g. {via1d {NDR1 via2d} {NDR2 via2d}} will override via1d in the default rule, and both via1d and via2d in NDR1 and NDR2.

Valid values: `list via_call`

Default: `{}`

This global property does not use additional arguments.

override_zero_placeable_area

If set CCOpt will allow CTS to run on a design which has zero placeable area. Setting this property may be useful to temporarily work-around problems with row definition and/or blockages causing placeable area to be zero.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

parents

The list of parent clock trees from which this clock tree is generated, if any.

Valid values: `list clock_tree`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-clock_tree <name>`".

partition_boundary_polarity

Specifies polarity of the clock signal with regard to the source of the flexible H-tree when entering partitions. The following values can be specified:

- `non_inverting`: When specified, allows only non-inverting polarity at partition entry points.
- `inverting`: When specified, allows only inverting polarity at partition entry points.
- `ignore`: When specified, allows inverting and non-inverting polarity at partition entry points.

Valid values: `non_inverting inverting ignore`

Default: `non_inverting`

Optional applicable arguments: "`-flexible_htree` *`<name>`*".

partition_groups

Specifies the groups in which partitions are clustered in channelless designs. Nested lists imply allowed crossings between groups. Each group has zero or one input port and each partition must only be specified once. Optionally, a maximum pre-route net length from the boundary of a partition group can be specified. The maximum pre-route net length must be specified as the second parameter of a nested partition group. Optionally, the next argument specifies the clock phase when entering the partition group in relation to the root pin of the H-tree. Allowed values are 'inverting' and 'non_inverting'. If no value is specified, the clock phase is unconstrained. The clock phase is unconstrained when crossing the boundaries of partitions within the same group. All specified clock phases must be either inverting or non_inverting.

For example, if the following partitions are specified:

```
{{A} {{C D} 50 non_inverting {{E F}} {{G}}}}
```

It means the following:

- The H-tree starts in partition A and descends into group C/D.
- From partition group C/D, the tree descends into groups E/F and G.
- Any clustering of sinks inside the C/D and E/F groups is allowed, potentially crossing internal partition boundaries several times.
- Partition A has no clock input port and one clock output port
- Partition group C/D has one clock input port and two clock output ports
- Partition group E/F and G have one clock input port and no clock output port
- The maximum net length from the entry point into partition group C/D is 50um (pre-route)
- The clock phase is non_inverting when entering parition group C/D. The clock phase is unconstrained when entering other partition groups.

Valid values: string

Default: `{}`

Optional applicable arguments: "`-flexible_htree <`*`name`*`>`".

pin

The pin under which the H-tree is created. This pin must be part of a clock tree at the time of synthesis.

Valid values: `pin`
Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

pin_capacitance_sources

Consists of a list of the places the capacitance could be retrieved from. Requesting this property for a pin not in the clock tree will result in an error. The possible places are:

- `library` - CTS will retrieve capacitance values from the library for one or more timing_corner/event combinations.
- `capacitance_override` - CTS retrieves capacitance values from the `capacitance_override` property for one or more timing_corner/event combinations.
- `blackbox_default_load_base_pin` - CTS retrieves capacitance values from the `blackbox_default_load_base_pin` property

for one or more timing_corner/event combinations.

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

pin_insertion_delay_histogram_bin_size

This property specifies the preferred size of the histogram bins that should be used when reporting the distribution of pin insertion delays. The bin size is specified in time units and must be greater than zero. A value of auto (the default) leaves it to the tool to calculate a bin size.

This property affects pin insertion delay histograms that are logged during various flow commands. It also determines the preferred histogram bin size that is used by the `report_ccopt_pin_insertion_delays` command when an explicit `-bin_size` argument has not been specified.

Valid values: auto, time value in library units

Default: auto

This global property does not use additional arguments.

pin_route_type

This property, when set on a pin, forces all the routing of the specified `net_type` generated by buffering the net driven by the specified pin to be routed with the given `route_type`. This property has no effect when set on non-output pins.

For example, if there exists a clock gate, XXX with an output pin, ECK driving some flops that need buffering during CTS to meet the DRV limits, then the following setting would cause the leaf nets generated by buffering the fanout of the `clock gate` to be routed with a `route_type` named `rt_a` and the trunk nets to be routed with a route_type named rt_b:

```
set_ccopt_property pin_route_type -pin XXX/ECK -net_type leaf rt_a
set_ccopt_property pin_route_type -pin XXX/ECK -net_type trunk rt_b
```

If a `net_type` is unspecified, the `route_type` will apply to nets of any `net_type`, for example the following setting will cause all nets generated by buffering pin, `YYY/ECK` to have `route_type rt_c`, irrespective of whether they are classified as leaf or trunk:

```
set_ccopt_property pin_route_type -pin YYY/ECK rt_c
```

Default: `default`

Applicable arguments: "`-pin <name>`" and "`-net_type <name>`". Required: "`-pin <name>`".

pin_route_type_propagation

Describes how the `pin_route_type` property should be applied to the net driven by the specified pin. The possible values are:

- `none` - it applies to net driven by the pin and only that net, irrespective of whether the net is buffered or not.
- `net` - it applies to all nets generated by buffering the pre-CTS net driven by pin.

Default: `net`

This property overrides any global setting for the specified pin. This property, set on a pin would do nothing if the pin has no `pin_route_type` setting.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

pin_target_max_trans

The per-pin target slew used for clock tree synthesis. This overrides any target set by `target_max_trans`. If set to auto (the default) the transition target used for optimization falls back the clock_tree, global target, liberty `max_transition` or the value of `target_max_trans_sdc`.

Valid values: `double` or `auto`

Default: `auto`

Applicable arguments: "`-delay_corner <name>`", "`-pin <name>`", "`-early`" and "`-late`". Required: "`-pin <name>`".

place_driver_in_center_of_fanout

When set, this property tells the CTS clustering code to put drivers in the center of the bounding box of their fanout. There are multiple methods for calculating the cluster center available.

Valid values: `true, false, leaf_only`

Default: `leaf_only`

This global property does not use additional arguments.

post_conditioning

Enable post-conditioning optimization after clock nets are routed.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

post_conditioning_enable_drv_fixing

If set to `false`, post-conditioning will skip its DRV-fixing step.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

post_conditioning_enable_drv_fixing_by_rebuffering

If set, post-conditioning will fix DRVs by adding buffers.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

post_conditioning_enable_routing_eco

If set to false, post-conditioning will skip its ECO-routing step.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

post_conditioning_enable_skew_fixing_by_rebuffering

If set to `true`, post-conditioning will attempt skew-fixing using rebuffering.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

power_weight

The power versus insertion delay trade-off. Valid values are between 0 and 1, specifying the weight that is put on power optmization during the synthesis of flexible H-trees.

Default: 1

Optional applicable arguments: "`-flexible_htree <name>`".

preserve_from_deletion

Do not allow this instance to be deleted. This property can be used to build clone-only CTS flows where a clock tree structure is pre-inserted in combination with multi-tap/Flex-h and then cloned for DRVs.

When this property is set to `true`, the specified instance is preserved from any operation in CTS that would remove it, except when `delete_clock_tree_repeaters -force` is set. The instance is also prevented from merging. To ensure symmetry across multiple passes of merging and cloning, when cloning this instance, CTS will leave the value of this property as `false`.

Valid values: `true false`

Default: `false`

Applicable arguments: "`-inst <name>`". Required: "`-inst <name>`".

primary_delay_corner

This specifies the delay corner in which clock tree balancing applies the slew and insertion delay targets. If more than one timing corner is defined, this must be set before running CCOpt. By default, this is set to the first defined delay corner.

Valid values: `corner name`, or `empty`

Default: {}

This global property does not use additional arguments.

primary_reporting_skew_groups

Specifies the primary skew groups used for reporting. By default, the value is specified as auto that automatically takes the skew group with maximum number of sinks as the primary reporting skew group. For invalid values, the default (auto) will be considered.

Valid values: `a list of existing skew group names`, `auto` or `none`

Default: `auto`

This global property does not use additional arguments.

primary_reporting_skew_groups_log_min_max_sinks

If set to `on`, the sinks with the shortest and longest paths in each primary reporting skew group will be logged. If set to `logv`, they will be logged only to the logv file.

Valid values: `on, off, logv`

Default: `logv`

This global property does not use additional arguments.

pro_enable_drv_fixing_by_rebuffering

If set, PRO will fix DRVs by adding buffers.

Default: `false`

This global property does not use additional arguments.

recluster_ignore_pins

Enable a reclustering step to find and fix ignore pins which have broken slew.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

remove_bufferlike_clock_logic

When set to `true`, CCOpt will identify and remove clock tree logic that is logically equivalent to buffering.

Valid values: `true false`

Default: `false`

Optional applicable arguments: "`-clock_tree <name>`".

rename_clock_tree_nets

Tells CCOpt to rename all clock tree nets for easy identification later in the flow.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

repair_congestion

If set, congRepair is called during CTS. This can reduce congestion and ease later routing issues.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

report_only_skew_group_with_target

The skew groups report (run using the `report_ccopt_skew_groups` command) displays insertion delay, skew, and min/max path information for different combinations of skew group, timing corner, and early/late path. Set the `report_only_skew_group_with_target` property to `false` (the default) to report on all skew group/timing corner/path combinations regardless of whether a skew target has been set. Set the `report_only_skew_group_with_target property` to `true` to report only on skew group/timing corner/path combinations where a skew target has been set (either explicitly or using the 'auto' setting).

Type: boolean

Default: `false`

This global property does not use additional arguments.

report_only_timing_corners_associated_with_skew_groups

Specifies whether the skew groups report displays all skew groups, in all corners or only corners associated with the timing configs from which the skew group was extracted.

Type: boolean

Default: `false`

This global property does not use additional arguments.

route_balancing_buffers_with_default_rule

If set CCOpt will always use the default routing rule for balancing buffers.

Valid values: `true false`

Default: `false`

This global property does not use additional arguments.

route_type

Specifies the route type. Setting this property binds an existing user-defined route_type to one or more types of clock tree nets. Binding a route_type to a type of clock tree nets means that all nets of that type (including the nets created by CTS) will be routed according to the specification of that route_type.

In the most common usage, the route_type is bound to one of the three types of clock tree nets (top, trunk, or leaf) with the optional

-net_type argument. Omitting the -net_type argument causes the route_type to be bound to all three types of clock tree nets. The optional -clock_tree *<pattern>* argument limits the binding to the clock trees whose name matches <pattern>. Omitting the -clock_tree argument causes the binding to apply to all clock trees.

For a route_type to be used in CTS, it must be bound to at least one net type. If net type is not bound to any route_type, a default route_type will be created for that net type at the start of CTS.

Examples:

- The following commands will cause all ‘trunk’ type clock tree nets to be routed using route_type ‘CTS_2w2s’ and all ‘leaf’ type clock tree nets to be routed using route_type ‘CTS_1w2s’:

  ```
  set_ccopt_property route_type -net_type trunk CTS_2w2s
  set_ccopt_property route_type -net_type leaf CTS_1w2s
  ```

- The following command will cause all ‘trunk’ clock tree nets for clock trees matching pattern “MAIN*” to be routed with route_type ‘NDR_xtra_wide’:

  ```
  set_ccopt_property route_type -net_type trunk -clock_tree MAIN* NDR_xtra_wide
  ```

Valid values: internal

Default: default

Optional applicable arguments: "-clock_tree <name>" and "-net_type <name>".

route_type_autotrim

Enable/disable autotrimming for all route types. If set, the allowed range of layers to use for routing may be restricted to a subset of the user-defined range in order to improve performance.

Valid values: true false

Default: true

This global property does not use additional arguments.

routing_override

The value auto means the pin has no routing override (either the user has not set an override, or the pin does not support routing overrides).
The value top means the pin has a routing override in place, forcing the net attached to the pin to be treated as a top net.
The value trunk means the pin has a routing override in place, forcing the net attached to the pin to be treated as a trunk net.
The value leaf means the pin has a routing override in place, forcing the net attached to the pin to be treated as a leaf net.

Valid values: auto top trunk leaf

Default: auto

Applicable arguments: "-pin <name>". Required: "-pin <name>".

routing_preferred_layer_effort

Controls preferred layer effort for clock routing. The following settings may be specified:

- standard (default) - No change made to current early global route settings.
- high - enables a package of early global route settings to increase layer adherence, possibly at the expense of wire length

and via count.

- `auto` - maps to high effort only if the effective routing effort level for the vast majority of nets of `net_type` trunk, for all clock trees resolves to high. Otherwise, the auto setting maps to `standard`.

Default: `standard`

This global property does not use additional arguments.

routing_top_fanout_count

The number of clock sinks this sink counts for when applying the top routing rules. Note that this property is only valid for sink pins, and it returns auto for non-sink pins. For a sink pin, a non-auto value means that this sink is counted as though it were multiple sinks, for the purposes of determining which nets should have top routing. An auto value for a sink pin means that the sink counts as a single sink.

Valid values: `integer > 0`

Default: `auto`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

routing_top_min_fanout

Minimum number of transitive fanout in the clock tree for a net to be routed as a top net. Nets with at least this many sinks in their transitive fanout in the clock tree will have the special routing rules applied to them.

Valid values: integer

Default: `unset`

Optional applicable arguments: "`-clock_tree <name>`".

routing_top_transitive_fanout

The number of clock sinks in the transitive fanout of the pin as counted for applying the top routing rules. This property is very similar to the `transitive_fanout property` but counts sink fanout using the `routing_top_fanout_count` property instead of always counting sinks as a single item of fanout.

Requesting this property for a pin not in the clock tree will result in an error.

Valid values: `integer`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

schedule

Controls what CCOpt can do with the schedule for this pin.

Allowed values are:

`auto:` Move the sink to make timing better during CCOpt optimization, and to make the clock tree QoR better where possible.

`timing_only:` Move the sink for timing if necessary, but otherwise leave it near the original schedule.

`off:` Leave the sink as close as possible to the original schedule, even if this breaks timing.

Valid values: `auto timing_only off`

Default: `auto`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

sink_grid

Specifies the columns and rows of a grid of H-tree sinks.

Valid values: `{columns rows}`

Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

sink_grid_box

The box describing the area that the grid of H-tree sinks should cover. This property only has an effect if the `sink_grid` property of the flexible H-tree is set.

Valid values: `{xmin ymin xmax ymax}`

Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

sink_grid_exclusion_zones

Specifies zones from which sinks of the sink grid are excluded if the zones completely cover the respective sink area. This property only has an effect if the `sink_grid` property of the flexible H-tree is set.

Valid values: `list {xmin ymin xmax ymax}`
Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

sink_instance_prefix

Prefix used for instance names of final cells (given by `-final_cell`). The name of the cell will be *`<prefix>_<htree_name>_<id>`*, where id is a running index.

Valid values: `string`

Default: `{}`

Optional applicable arguments: "`-flexible_htree <`*`name`*`>`".

sink_grid_sink_area

The approximate size of the rectangle describing valid locations for final cells (given by `-final_cell`) per H-tree sink in the grid. This property only has an effect if the `sink_grid` property of the flexible H-tree is set.

Valid values: `{width height}`

Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

sink_type

The type of sink this pin represents. For this property to take effect, set it before running `create_ccopt_clock_tree_spec.`

Valid values are as follows:

`auto:` The pin type will be automatically determined by CCOpt
`through`: Through pin. Trace the clock tree through this pin.
`stop:` Stop pin. When defining clock trees, CCOpt stops searching for parts of the clock tree at stop pins.
`ignore:` Ignore pin. CCOpt stops searching for parts of the clock tree at ignore pins and it does not attempt to balance the insertion delay of ignore pins.
`min:` Min pin. Keep the pin at minimal insertion delay.
`exclude:` Exclude pin. Exclude this pin from the clock tree.

Valid values: `auto through stop ignore min exclude`

Default: `auto`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

sink_type_reasons

Provides the reasons why the specified pin has the given `sink_type`. This property is configured by the `create_ccopt_clock_tree_spec` command in order to record the reasons that the `sink_type` property has been configured for the given pin.

This property is a list of values. Valid values are as follows:

`auto`: The sink_type property is set to 'auto'
`implicit`: The pin is an implicit sink (flop/latch)
`user`: The user has set the sink_type property
`design_io`: This pin is a design I/O
`multiple_outputs`: This pin is on an instance with multiple outputs and the trace_through_to property has not been set
`set_disable_timing`: SDC set_disable_timing stops the clock at this pin
`set_case_analysis`: SDC set_case_analysis stops the clock at this pin
`generated_clock_tree`: This pin is the generator input to an SDC generated clock
`no_sdc_clock`: The SDC clock is stopped at this pin for other reasons
`ilm`: The `create_ccopt_clock_tree_spec` command has detected an ILM below this pin

Valid values: `auto, implicit, user, design_io, multiple_outputs, set_disable_timing, set_case_analysis, generated_clock_tree, no_sdc_clock, ilm`

Default: `auto`

Applicable arguments: `"-pin <`*`name`*`>".`

Required: `"-pin <`*`name`*`>".`

sinks

A list of sinks for this skew group.

Valid values: `list pin`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-skew_group <name>`".

sinks_active

Returns the list of active sinks for this skew group. All sink pins that have this skew group in the `skew_groups_active_sink` property are included.

Note that only endpoint sink pins are included. Non-sink pins through which this skew group passes are not included.

Valid values: `list pin`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-skew_group <name>`".

size_clock_gates

When set to true (the default), the CTS algorithm sizes clock gates that appear in the clock tree. Setting this property may cause the clock tree to have a lower insertion delay, but might change the cell types of logic gates in the clock tree, which in turn may require them to be moved slightly to find a legal location for the new cell.

Default: `true`

This global property does not use additional arguments.

size_clock_source

When set to true, CTS will try to size the clock source. Only clock sources that are buffers, inverters, logic, and clock gating cells with a single output will be sized. The cells available for CTS to size clock sources can be set specified using the property '`clock_source_cells`'.

Note: Clock sources may be prevented from sizing, for example, if their placement status is 'fixed' or if they are connected to pre-routed. However, the software will ignore cells whose placement status was set 'fixed' by a previous step, such as H-tree synthesis, and it will also reset the `skip_routing` attribute for nets connected to clock tree sources for which the `size_clock_source` property is `true`.

Valid values: `true false`

Default: `false`

Optional applicable arguments: "`-clock_tree <name>`".

size_logic

When set to true (the default), the CTS algorithm sizes logic that appears in the clock tree. "Logic" does not include clock gates, buffers, and inverters in a clock tree, which are always sized unless they are locked, or clock generators that are above the root of the clock tree. Usually, this affects multiplexers used for selecting one of a number of clocks, or for switching between a test clock and the main clock. Setting this property may cause the clock tree to have a lower insertion delay, but might change the cell types of logic gates in the clock tree, which in turn may require them to be moved slightly to find a legal location for the new cell.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

skew_band_size

Internal property to control `ccopt_design` flow.

Valid values: `internal`

Default: `0.1`

This global property does not use additional arguments.

skew_group_report_columns

A Tcl list of columns to include in skew group reports produced by the skew group report. You can use this property to specify the columns you would like to include in the skew group report, and the order in which the columns should appear. Most of the legal values are straightforward. However, the set of legal values of the following form deserve further explanation:

`summaryType_summaryLocation[_event]`

These values let you report the delay value for other paths that go through the pin.

`summaryType` is one of: max (show the longest delay), min (show the shortest delay), or skew (show the skew, that is the difference between the longest and the shortest delay).

`summaryLocation` is one of: above (show the delay/skew above this pin), below (show the delay/skew below this pin), or through (show the delay/skew for paths through this pin).

`event` is one of: rise (show the delay/skew for the rise event at this pin), fall (show the delay/skew for the fall event at this pin), or both (show the delay/skew for both events at this pin).

Valid values:

```
capacitance
distance
event
fanout
increment
lib_cell
location
name
slew
status
time
```

Default: `{name lib_cell event increment time slew capacitance location distance fanout status}`

This global property does not use additional arguments.

skew_group_report_histogram_bin_size

When set to a numeric value, that numeric value will be used as the histogram range size (in library units). For example, if the library time units are set to 1 nanosecond, a value of 0.010 for `report_skew_groups_histogram_bin_size` will result in histogram ranges of 10 picoseconds.

When set to auto, the size of the histogram ranges are dependent on the skew targets that are set. If a skew target is set for a given half corner and skew group combination, then the histogram range size will be 10% of the skew target for that half corner and skew target combination. If no skew target is set for a half corner and skew group combination but a skew target is set for the primary half corner and skew group combination, then the histogram range size will be 10% of the skew target for the primary half corner and skew group combination.

In the event that no skew targets are set and report_skew_groups_histogram_bin_size is set to auto, a default value of 10 picoseconds will be used for the histogram range size.

Valid values: `auto | string`

Default: `auto`

This global property does not use additional arguments.

skew_groups_active

Returns the list of active skew groups for this pin. For sink pins, this property lists both skew groups that pass through this pin and skew groups for which this sink is an endpoint. For non-sink pins, shows skew groups that pass through this pin.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

skew_groups_active_sink

Returns the list of skew groups for which this pin is an active sink. For sink pins, this property lists the skew groups for which this sink is an endpoint. Skew groups that pass through this pin are not included. For non-sink pins, this property always returns null.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

skew_groups_ignore

The list of skew groups for which paths through this pin are ignored.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

skew_groups_constraining

The list of delay-constraining skew groups which are active at this pin.

This property reports similar data to skew_groups_active. The only difference is that the reporting-only skew groups are not included in this property's value.

For sink pins, this property lists both delay-constraining skew groups that pass through this pin and delay-constraining skew groups for which this sink is an endpoint.

For non-sink pins, shows delay-constraining skew groups that pass through this pin.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

skew_groups_constraining_sink

The list of delay-constraining skew groups for which this pin is an active sink. This property reports similar data to skew_groups_active_sink. The only difference is that the reporting-only (`constrains none`) skew groups are not included in this property's value.

For sink pins, this property lists the delay-constraining skew groups for which this sink is an endpoint. Skew groups that pass through this pin are not included.

For non-sink pins, this property always returns null.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

skew_groups_sink

The list of skew groups for which this pin is a sink.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

skew_groups_source_pin

The list of skew groups for which this clock tree or pin is specified as a source.

Valid values: `list skew_groups`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

skew_passes

Internal property to control ccopt_design flow.

Valid values: `internal`

Default: `250`

This global property does not use additional arguments.

skew_passes_ideal_mode

Internal property to control ccopt_design flow.

Valid values: `internal`

Default: `250`

This global property does not use additional arguments.

skew_passes_per_cluster

Internal property to control ccopt_design flow.

Valid values: `internal`

Default: `25`

This global property does not use additional arguments.

source_driver

Specifies the library pin which is assumed to drive this clock tree. It is either a single lib_pin (in which case all arcs to that lib_pin shall be used when timing the clock tree root) or a pair of lib_pins (in which case only arcs from the first specified pin to the second will be considered). By default this is generated from clock tree extraction.

Valid values: `lib_pin | {lib_pin lib_pin}`

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`".

source_group_clock_trees

A list of the clock trees relevant to this source group.

Valid values: `list clock_trees`

Default: `{}`

Optional applicable arguments: "`-clock_tree_source_group <name>`".

source_input_max_trans

The slew which will be assumed at the input of the root driver.

Valid values: `double`

Default: `0`

Optional applicable arguments: "`-delay_corner <name>`", "`-clock_tree <name>`", "`-early`" and "`-late`".

source_latency

Specifies a delay value between the global clock source and this clock tree. This additional delay will be included in all timing analysis involving skew groups for which this clock tree is a source. The default is 0.

Valid values: `double`

Default: `0`

Optional applicable arguments: "`-delay_corner <name>`", "`-clock_tree <name>`", "`-early`", "`-late`", "`-rise`" and "`-fall`".

source_max_capacitance

The maximum capacitive load which this clock tree is permitted to drive.

Valid values: `double | auto`
Auto: from clock tree extraction

Default: `auto`

Optional applicable arguments: "`-clock_tree <name>`".

source_output_max_trans

If non-zero, the slew which will be assumed at the output of the root driver. This overrides the value from SDC.

Valid values: `double`

Default: `0`

Optional applicable arguments: "`-delay_corner <name>`", "`-clock_tree <name>`", "`-early`" and "`-late`".

source_pin

The source pin for this clock_tree.

Valid values: `pin`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-clock_tree <name>`".

spec_config_create_reporting_only_skew_groups

Controls whether spec creation will synthesize reporting-only skew groups. A reporting-only skew group is a skew group whose constrains property is set to 'none'. Such a skew group imposes no clock balancing constraint and will not be considered by CTS. The typical example is a generated clock that is synchronous to its master clock. The relationship between the two clocks implies that the sinks of both clocks should be balanced together. To model this, a clock spec created for such a design will have the skew group corresponding to the master clock span the domain of both the master clock and also the generated clock. The skew group of the generated clock is then a subset of the master clock skew group: as such it is a redundant constraint, and may be safely omitted. If it is synthesized, such a skew group will be created as a reporting-only skew group.

When this property is set to `false`, reporting-only skew groups are completely omitted from the generated spec. When set to `true`, reporting-only skew groups are included in the generated spec, but are marked with the `constrains` property set to `none`. Either way, they impose no balancing constraint.

Valid values: `true false`

Default: `true`

stack_via_rule

The preferred stack via rule for terminal connections. This property helps guide the choice of stack via rule (via pillar) used for connecting routes to netlist terminals. If the specified value names a valid candidate for terminal in question (it is a member of list the candidate rules associated with the terminal's cell pin), then it will be used as the preferred stack via rule for connecting to that terminal.

Valid values: '`auto`' or `stack via rule name`

Default: `auto`

Optional applicable arguments: "`-clock_tree <`*`name`*`>`", "`-lib_pin <`*`name`*`>`" and "`-net_type <`*`name`*`>`".

stack_via_rule_required

Specifies the pin-specific required field for stack via rule connections.

Default: `false`

Optional applicable arguments: "`-clock_tree <`*`name`*`>`", "`-lib_pin <`*`name`*`>`" and "`-net_type <`*`name`*`>`".

sources

A list of sources for this skew group.

Valid values: `list pin`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Optional applicable arguments: "`-skew_group <name>`".

stop_at_sdc_clock_roots

If specified, stop searching for parts of the clock tree through SDC clock roots when defining generated clock trees for H-tree sinks.

Default: `false`

Optional applicable arguments: "`-flexible_htree <name>`".

target_insertion_delay

The target insertion delay used for clock tree synthesis. This may be set to the following values:

`auto` - Allow the minimum clustered insertion delay to be pushed up a little (around 5%) to facilitate clock tree power reduction.

`A numeric value` - Attempt to balance the clock tree to the specified insertion delay (specified in library units). CTS will attempt to have a longest clock path delay of no more than this value plus half of the skew target, and a shortest path delay of no less than this value minus half the skew target.

Valid values: `auto | double`

Default: `auto`

Optional applicable arguments: "`-skew_group <name>`".

target_insertion_delay_wire

The target wire insertion delay used for clock tree synthesis.

Valid values: `auto | double`

Default: `auto`

Optional applicable arguments: "`-skew_group <name>`".

target_max_stage_delay_sigma

The max per-stage SOCV delay sigma target to use for CTS.

Default: `auto`

Optional applicable arguments: "`-delay_corner <name>`", "`-early`" and "`-lat`e".

target_max_capacitance

The target maximum capacitive load to allow during clock tree synthesis. This property specifies a maximum (combined pin and wire) capacitance that the clock tree synthesis algorithm will allow any given library pin to drive in a given clock tree when driving a given net_type. It is specified in library units. It currently only constrains the primary delay corner capacitance values - other delay corners can be specified but will not be constrained. This property is applied in addition to the `max_capacitance` constraints read from the liberty library data - the tightest (lowest) of the constraint specified by this property and the constraint present in the liberty data will be used. It also doesn't apply at the root pins of clock trees - to constrain those nets the `source_max_capacitance` CCOpt property should be used instead.

Valid values: `auto | double`

Default: `auto`

Optional applicable arguments: "`-delay_corner <name>`", "`-clock_tree <name>`", "`-lib_pin <name>`", "`-net_type <name>`", "`-early`" and "`-lat`e".

target_max_trans

The target slew used for clock tree synthesis. This property specifies a maximum slew time that the clock tree synthesis algorithm will allow in this clock tree, in library units. 'default' means 'auto' in primary half corner and 'ignore' in other half corners. If set to 'auto', CTS picks an appropriate value based on the collection of allowed buffer sizes and library parameters, although this may not give optimal quality of results. If set to 'ignore', CTS does not constrain the corner. The tightest (lowest) of the constraint specified by this property and the constraint present in the liberty data (max_transition) will be used.

Valid values: `default | auto | ignore | double`

Default: `default`

Optional applicable arguments: "`-delay_corner <`*`name`*`>`", "`-clock_tree <`*`name`*`>`", "`-power_domain <`*`name`*`>`", "`-net_type <`*`name`*`>`", "`-early`" and "`-late`".

target_max_trans_sdc

If non-zero, the target slew used for clock tree synthesis, overriding the SDC. This property specifies a maximum slew time that the clock tree synthesis algorithm will allow, in library units obtained from SDC.

Valid values: `double`

Default: `0`

Optional applicable arguments: "`-delay_corner <name>`", "`-clock_tree <name>`", "`-net_type <name>`", "`-early`" and "`-late`".

target_multi_corner_allowed_insertion_delay_increase

This slack is the factor by which we will permit the insertion delay target of this skew group to increase for the purposes of multi corner balancing in general, and wire/cell delay balancing in particular. If a wire/cell delay balance is not possible even after increasing insertion delays by the allowed amount, we will not permit further insertion delay increases.

Values less than 1.0 are not permitted.

A value of 1.0 means "do not allow insertion delay to increase at all".

A value of 2.0 means "allow the insertion delay to at most double."

A value of infinity means "allow any amount of insertion delay increase".

An undefined value ("`auto`") is treated as infinite.

Valid values: `auto | double`

Default: `auto`

Optional applicable arguments: "`-skew_group <name>`".

target_skew

This specifies the target skew for clock tree balancing. This may be set to a numeric value, or one of '`auto`', '`ignore`' or '`defaul`t'.

If set to '`auto`' this indicates that an appropriate skew target should be computed.

If set to '`ignore`' this indicates that skew should not be balanced for this corner/path combination.

If unspecified then the value of this property is '`default`'.

If the value of the property is 'default' the target skew for late delays in the primary delay corner is interpreted as 'auto' and as '`ignore`' otherwise.

Valid values: `default | auto | ignore | double`

Default: default

Optional applicable arguments: "`-skew_group <name>`", "`-delay_corner <name>`", "`-early`" and "`-late`".

target_skew_wire

This specifies the target wire skew for clock tree balancing. This may be set numeric value, or one of 'auto', 'ignore' or 'default'.

If set to '`auto`' this indicates that an appropriate skew target should be computed.

If set to '`ignore`' this indicates that skew should not be balanced for this corner/path combination.

If unspecified then the value of this property is '`default`'.

If the value of the property is 'default' the target skew for late delays in the primary delay corner is interpreted as 'auto' and as 'ignore' otherwise.

Valid values: `default | auto | ignore | double`

Default: `default`

Optional applicable arguments: "`-skew_group <name>`", "`-delay_corner <name>`", "`-early`" and "`-late`".

timing_connectivity_based_skew_groups

SDC false path assertions may render a generated SDC clock asynchronous to its master clock. Under such a condition, the sinks of the generated clock need not be balanced against the sinks of the master clock: there are no timing paths between the two clocks that can be affected by inter-clock skew. This balancing 'relaxation' is realized by adjusting the skew groups that are generated for the two clocks.

This property specifies which SDC assertions are considered when deciding whether a generated clock is asynchronous to its master. Valid values for this property are as follows:

- off: In this mode, every generated SDC clock is treated as being synchronous to its master SDC clock.
- clock_false_path: In this mode, only clock/clock set_false_path and set_clock_group assertions are considered.

Valid values: `off clock_false_path`

Default: `off`

This global property does not use additional arguments.

timing_connectivity_based_skew_groups_balance_master_clocks

This property has no effect if `timing_connectivity_based_skew_groups` is set to `'off'`. With timing connectivity based skew groups enabled, this property controls how CCOpt will address the synchrony of disjoint (i.e. non-overlapping) master (i.e. non-generated) SDC clocks.

When this property is set to `false`, all disjoint master clocks are assumed to be mutually asynchronous. The skew groups generated by `create_ccopt_clock_tree_spec` will not constrain the sinks of two such clocks to be balanced together.

When this property is set to `true`, the synchrony of disjoint master clocks is determined by consulting the SDC assertions in accordance with the value of property `timing_connectivity_based_skew_groups`. Additional skew groups will be generated by `create_ccopt_clock_tree_spec`, such that the sinks of each pair of synchronous disjoint master clocks will be balanced together by CTS.

Valid values `true false`

Default: `false`

This global property does not use additional arguments.

timing_connectivity_info

This property is populated by clock spec creation when timing connectivity based skew groups are enabled. Configuration of this property will be written to the generated spec. It documents the clock/clock balancing relationships that were determined for the production of timing connectivity based skew groups. This property is documenting only.

If timing connectivity based skew groups are not enabled, this property is not populated by spec creation.

The value of the property is a set of nested Tcl dictionaries, four levels in total. The top-level dictionary indicates the information category. Currently there is only one such category: 'clock_relationships'. The second level is keyed on constrained mode name; the lowest level dictionaries are both keyed on SDC clock name. These keys taken together identify two SDC clocks in a given constraint mode. The leaf value specifies the resolved balancing relationships between the two SDC clocks. It an enum with the following members:

- `direct` - The two clocks are determined to belong to the same clock group and must be balanced together.
- `indirect` - The two clocks do not belong to the same clock group; however by transitive closure they must balance together.

Clocks in a direct or indirect balancing relationship will share one or more skew groups so as to ensure that their sinks are balanced together by CTS.

Missing entries should be taken as 'need not balance'; i.e. the pair of SDC clocks have neither a direct nor indirect balancing relationship.

The indirect balancing relationship can be explored in more detail using this same Tcl dictionary to probe out the transitive closure over the direct balancing relationships that link two indirect balance clocks together. For example if clock A directly balances with clock B only, and clock B directly balances with clock C only; then clock A indirectly balances with clock C.

Below is an example dict value, corresponding to clock groups {clkA clkB} {clkB clkC} in constraint mode cm1, and clock groups {clkP clkQ} {clkR clkS} in constraint mode cm2.

```
{clock_relationships \
  {cm1 {clkA {clkB direct clkC indirect} \
        clkB {clkA direct clkC direct} \
        clkC {clkA indirect clkB direct}} \
   cm2 {clkP {clkQ direct} \
        clkQ {clkP direct} \
        clkR {clkS direct} \
        clkS {clkR direct}}}}
```

Notice that although they are in separate clock groups, `clkA` and `clkC` nevertheless indirectly balance. This is due to fact that clkB is present in both clock groups. Note also that we omit information about the relationship between `clkP` and `clkS`: they are 'need not balance'.

Default: `{}`

This global property does not use additional arguments.

top_buffer_cells

Specifies the buffers cells available for CTS to use on top nets. If none are specified, CCOpt will use the same buffers as on trunk nets - as specified in the `buffer_cells` property. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names. If set explicitly, CCOpt will ignore any don't use settings for the specified cells.

Different top buffer cells may be specified for any combination of clock tree and power domain.

Valid values: a list of library cell names, or a list of patterns to expand to library cell names

Default: `{}`

Optional applicable arguments: "`-clock_tree <`*`name`*`>`" and "`-power_domain <`*`name`*`>`".

top_inverter_cells

Specifies the inverter cells available for CTS to use on top nets. If none are specified, CCOpt will use the same inverters as on trunk nets - as specified in the `inverter_cells` property. Cell names may be specified as a Tcl list of names, or as a Tcl list of patterns to be expanded to match names.If set explicitly, CCOpt will ignore any don't use settings for the cells specified.

Different top inverter cells may be specified for any combination of clock tree and power domain.

Valid values: a list of library cell names, or a list of patterns to expand to library cell names

Default: `{}`

Optional applicable arguments: "`-clock_tree <name>`" and "`-power_domain <name>`".

trace_bidi_as_input

Trace bi-directional pins as input pins during `create_ccopt_clock_tree_spec`.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

trace_through_to

Clock tree definition will, by default, not continue through certain types of cell arc (for instance, the clock to Q arc in a DFF). This property allows you to override the default behavior by specifying the output pin to which the clock tree should propagate, when it arrives at a given input pin. The specified output pin must be another pin on the same instance as the given input pin.

The output pin may be specified either by its fully qualified name (i.e. inclusive of the instance name), or else simply by its local (cell-relative) name.

Note: If both `trace_through_to` and `library_trace_through_to` are applicable at a given netlist instance pin, the `trace_through_to` value takes precedence.

Valid values: pin

Default: `{}`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

transitive_fanout

The number of clock sinks in the transitive fanout of the pin, within the clock tree. Requesting this property for a pin not in the clock tree will result in an error.

Valid values: `int`

Read-only: This property cannot be modified by `set_ccopt_property` or `unset_ccopt_property`.

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

trunk_cell

The library cell to use inside the H-tree.

Valid values: `string`
Default: `{}`

Optional applicable arguments: "`-flexible_htree <name>`".

trunk_override

Prefer trunk routing rules for this pin. Only applies to clock tree sinks.

Valid values: `true false`

Default: `false`

Applicable arguments: "`-pin <name>`". Required: "`-pin <name>`".

update_io_latency

Determine whether to update IO latencies within ccopt_design.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

use_estimated_routes_during_final_implementation

Use route estimates, instead of NanoRoute during the final implementation clock routing phase.

Valid values: `true` or `false`

Default: `false`

This global property does not use additional arguments.

use_inverters

Specifies whether clock tree synthesis should prefer to use inverters rather than buffers when balancing the clock tree. If set to true, CTS will use inverters for the clock tree balancing process. If set to false, CTS will use the minimum number of levels of inverters required to maintain logical correctness. If set to auto (the default) CTS will use what it considers to be the best combination of buffers and inverters to get optimal quality of results.

Valid values: `auto true false`

Default: `auto`

Optional applicable arguments: "`-clock_tree <name>`".

use_receiver_model_capacitance_for_drv

When this property is set to `auto`, CCOpt will use receiver model capacitance for drv if global variable, timing_report_use_receiver_model_capacitance is true. When set to true, CCOpt will use receiver model capacitance for drv even if timing_report_use_receiver_model_capacitance is false.

By default, this property is set to false, which means that CCOpt will not use receiver model capacitance for drv.

Valid values: `auto true false`

Default: `false`

This global property does not use additional arguments.

use_macro_model_pin_cap_only

This property is for translating the Macro Model capacitance constraint from FE-CTS spec format Macro Model constraints to CCOpt. The default is false, implying that the Macro Model pin capacitance will be added to the library pin capacitance. If the property is set to true only the Macro Model pin capacitance will be used.

Type: boolean

Default: `false`

This global property does not use additional arguments.

useful_skew_implementation_cache_hold_slacks

If set, CCOpt will gather hold slacks for constructing implementation slack windows. Setting this to `true` will temporarily switch to the hold analysis mode and gather hold slacks for the purposes of building implementation slack windows that factor in a contribution from the hold views. This incurs extra timing updates.

Valid values: `true false`

Default: `true`

This global property does not use additional arguments.

useful_skew_clock_gate_movement_limit

Each clock gate is restricted to a Manhattan ball centered on its original location with CCOpt flow. The radius of the ball is a multiple of the clock gate height. This controls the default value of that multiple.

Default: `10`

This global property does not use additional arguments.

useful_skew_max_delta

The maximum delta for useful skew.

Type: `real`

Default: `1000`

This global property does not use additional arguments.

useful_skew_min_delta

The minimum non-zero delta which useful skew will apply.

Valid values: `double | auto`

Default: `auto`

This global property does not use additional arguments.

useful_skew_post_implement_db

Filename to save design state to after useful skew implementation.

Type: file

Default: `{}`

This global property does not use additional arguments.

useful_skew_pre_implement_db

Filename to save design state to prior to useful skew implementation.

Type: file

Default: `{}`

This global property does not use additional arguments.

virtual_delay

The amount of virtual delay that has been applied under this pin.

Valid values: `double`

Default: `0`

Applicable arguments: "`-delay_corner <name>`", "`-pin <name>`", "`-early`", "`-late`", "`-ris`e" and "`-fall`". Required: "`-pin <name>`".

Creating the ICT File

The first step involved in modeling the parasitic interconnect capacitance and resistance of your design is to specify the fabrication process information in an Interconnect technology (ICT) file by using the syntax defined in this section. You can use any text editor to enter this information.

Note: Although there are no file-naming restrictions for ICT files, you should name your ICT file by using the process name with the `.ict` file extension, as follows:

`process_name.ict` (ICT file)

Fabrication process information consists of the following requirements:

- Minimum spacing and minimum width of the conductors as specified in the design rules for the conductor layers
- Thicknesses of the conductor layers
- Heights of the conductor layers above the substrate (measuring height from the field) or as a delta from a previously defined lower-level conductor layer
- Resistivities of the conductor layers
- Interlayer planar dielectric constant, its height above the substrate (measuring height above the field), and its thickness
- Names of the top conductor layer of a via, the bottom conductor or diffusion layer of the via, and the contact resistance of the via
- Names of the wells

The following sections below describe 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 *Quantus Techgen Reference* manual.

Format

Lines in the ICT file are in the following general format:

```
command name {argument_list}
```

where `argument_list` is a list of field-value pairs. The fields in this syntax are separated by white space. ViewICT, IceCaps, and RCgen ignore blank lines.

Note: A backslash (\) is generally required for line continuation, but it is not required if you are using braces ({}) to define a list.

Data

All data entered into the ICT file should be the actual physical fabrication process information, not the drawn data.

Comments

A pound-sign character (#) at the beginning of a line indicates text that ViewICT, IceCaps, and RCgen are treated as comments.

Case Sensitivity

All keywords in the ICT file are case-insensitive. However, the arguments are case-sensitive. Keywords consist of all command and field names.

Warnings and Errors

The ViewICT utility displays all errors, warnings, and informational messages on screen and writes them in a log file. Warnings and errors include the corresponding line number.

Invalid Layer Names

The "NX" string is an invalid layer name.

- ICT File Commands
- Sample ICT File

ICT File Commands

This topic describes the commands available in the ICT file. All command fields are enclosed in braces `({})`.

- Process
- Well
- Conductor
- Dielectric
- Passivation
- Via

Process

The `process` command specifies the background dielectric constant. Use it only once in the ICT file.

Syntax

```
process name {background_dielectric_constant value}
```

or

```
      process name {

         background_dielectric_constant value

}
```

This syntax contains the following parameters:

- *`name`*
  Specifies the name of the process.
- `background_dielectric_constant` *`value`*
  Specifies the dielectric constant for the region above the top passivation layer or last dielectric layer. This field is required.

Example

```
process "Process_Example" {

          background_dielectric_constant 1.0

}
```

Well

The `well` command which defines the well layers is an optional command that you can use to differentiate capacitance to a well from capacitance to the substrate.

Syntax

```
well name { }
```

`name` specifies the name of the well layer.

Anything placed in the brackets is ignored.

Example

```
well nwell { }

well pwell { }
```

Conductor

The `conductor` command defines conductor layers.

You can specify the height of a conductor layer in three ways:

- Height (absolute)
- Delta height (relative)
- Upto (maximum top down)

You can use more than one of these methods per conductor definition, as long as the numbers are valid.

All measurements are in microns, unless otherwise specified.

Syntax

```
conductor name {field1 value1 ... fieldN valueN}

or

conductor name {

    field1 value1

    ...

    fieldN valueN
}
```

You can specify the `field-value` pairs in any order.

This syntax contains the following parameters:

- `name`

  Specifies the name of the conductor layer.
- `min_spacing` *`value`*

  Specifies the minimum spacing permitted by the technology between two conductors (wires) on a layer.
- `min_width` *`value`*

  Specifies the minimum width of a conductor.
- `height` *`value`*

  Specifies the layer's height above the substrate.
- `delta_height` *`value`*

  Specifies the layer's height relative to the top of another layer. This parameter must be used with `delta_layer`.
- `delta_layer layer_name`

  Specifies the reference layer for `delta_height`. It must be a layer that has already been defined. The reference layer must be a conducting layer, a dielectric layer, or a passivation layer. This parameter must be used with `delta_height`.
- `thickness` *`value`*

  Specifies the layer's thickness.
- `upto` *`value`*

  Specifies the layer's top surface height above the substrate. This value is equal to the height plus the thickness. You only need to specify two of the three or four parameters (`height`, {`delta_height`, `delta_layer`}, `thickness`, `upto`) to complete the geometrical definition of a conductor layer.
- `resistivity` *`value`*`|[`*`value width`*`]+`

  Specifies the layer's sheet resistance, in ohms per square. You can enter the resistivity value as a constant, or you can enter value-width pairs as a piecewise linear function. You may want to use the value-width pairs to account for width-dependent resistivity.

  If you enter value-width pairs, the syntax is as follows:

  `resistivity` *`value1 width1 value2 width2 ... valuen widthn`*

  If the width of the wire is less than the minimum width, width1, use the minimum value, value1. If the width of the wire is greater than the maximum width, widthn, use the maximum value, valuen. For widths between value-width pairs, the resistivity value through linear interpolation.

  Note: The width in the value-width pair refers to the silicon width of the wire.
- `rho`
  - `rho_widths W1 ... Wn`
  - `rho_spacings S1 ... Sm`
  - `rho_values R11 .... R1n`

    ```
    ....
    Rm1 ... Rmn
    ```

    This parameter is for specifying resistivity as a function of both width and spacing. For values that fall between specified points, linear interpolation is applied. When values are outside of the boundary values, it uses the boundary values.

- `gate_forming_layer [true|false]`

  Specifies that this layer forms the gate. The polycide or polysilicon layer is a typical gate-forming conducting layer.

- `field_poly_diffusion_spacing` *`value`*

  Specifies the lateral spacing between field polycide and diffusion for transistor-level parasitic extraction. There is no lateral separation between gate polycide and diffusion.

- `PnR_widths [value]+, PnR_spacings [value]+`

  Allows you to provide design widths and spacings used in the layout to the technology file generation program. These are not necessary for accurate extraction of parasitics if the design widths and spacings are within small perturbations of the minimum process widths and spacings. However, if the design widths and spacings are routinely different from the specified process parameters, it is recommended that you provide these values to the technology file generator.

- `capacitor_only_layer_to` *`layer_name`*

  Specifies that the current layer be used solely to create high capacitance values in the design and that it is located a few angstroms above or below the *`layer_name`* layer. Layers having the `capacitor_only_layer_to` keyword set are not extracted.

- `wire_top_enlargement_c` E_{top}

  `wire_top_enlargement_r` E_{top}

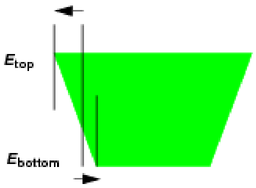
  Specifies the enlargement, either positive or negative, of the top edge of the wire. Specify values for both R and C to account for different bias values for R and C. See Figure 1 below.

- `wire_bottom_enlargement_c` $\mathrm{E_{bottom}}$

  `wire_bottom_enlargement_r` $\mathrm{E_{bottom}}$

  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 the figure below.

Figure 1: Trapezoidal Wire Shape Resulting from Manufacturing Processes



- `wire_edge_enlargement | wire_edge_enlargement_[r|c]`

  `wee_widths` $\mathrm{W_1}$ `...` $\mathrm{W_n}$

  `wee_spacings` $\mathrm{S_1}$ `...` $\mathrm{S_m}$

  `wee_adjustments` $\mathrm{E_{11}}$ `...` $\mathrm{E_{1n}}$

  .

  .

  .

`Em`$_1$ `...` `Em`$_n$

Models the effect of wire-edge enlargement, if the `wire_edge_enlargement`, `wee_width`, `wee_spacings`, and `wee_adjustments` keywords are specified. The wee_adjustments table describes the amount of enlargement applied when certain spacings and widths are observed. For example, the wire is enlarged by E_{ij} for spacing S_i and width W_j. Positive enlargements oversize and negative enlargement undersize the wire. A piecewise constant interpolation is used to obtain enlargements for intermediate spacings and widths. For width/spacings outside of the boundary width/spacing points, the boundary values are used.

Wire_edge_enlargement_r and wire_edge_enlargement_c can be used if one wants to specify different values for resistance and capacitance.

- `wee_widths` W_1 `...` W_n

  Specifies the widths of the wires in the design. Typically, variation is only seen for widths less than 1.5 microns.
- `wee_spacings` S_1 `...` S_m

  Specifies the spacings of the wires in the design. Typically, variation is only seen for spacings less than 1.5 microns.
- `wee_adjustments` $E_{11}...E_{1n}$ `...` $E_{m1}...E_{mn}$

  Specifies the enlargement, either positive or negative, of the wire edge.

See Wire-Width Values for information on the wire-width values to use.

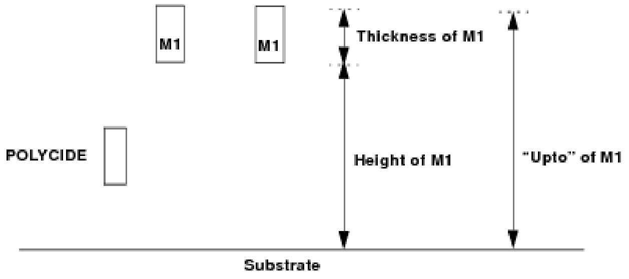
Required Conductor Command Fields

The required fields in this syntax are `min_spacing, min_width, resistivity, gate_forming_layer, min_net_fill_spacing, X_fill_fill_spacing, Y_fill_fill_spacing, unit_fill_region`, and two of the following three parameters:

- `height (or delta_height and delta_layer)`
- `thickness`
- `upto`

The figure below illustrates these parameters.

Figure 2: Geometric Fields in a Conducting Layer



Wire-Width Values

You can use the `wire_edge_enlargement` statement with the `wire_top_enlargement` statement or the `wire_bottom_enlargement` statement, or both in the ICT file. If you use the `wire_edge_enlargement` statement with either or both of these statements, the width of the wires defined by `wee_widths` must be biased as follows:

`drawn_width + ((top + bottom) / 2)`

When calculating resistivity as a function of width, you must use the `wire_top_enlargement` and `wire_bottom_enlargement` values to correct the resistance-width pairs. If a table of wire-edge enlargement values is available, the RC extractor uses the wire widths in the table, which always include biasing and wire-edge enlargement. If this table is not available, the resistance is calculated as follows:

`rho* L / (drawn_width + (top + bottom) / 2 + (top + bottom) / 2)`

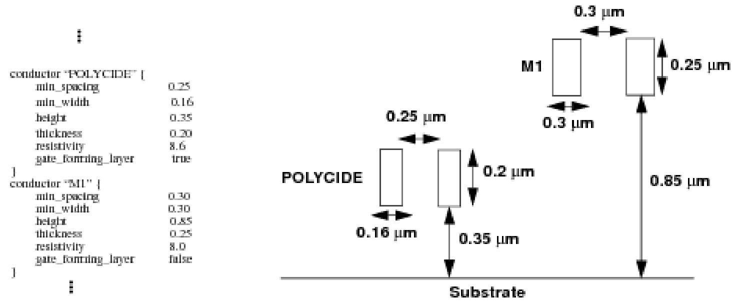
where *`rho`* is the sheet resistivity.

Wire-width values are used in the following order:

1. Drawn width
2. Biased width
3. Edge-enlarged width
4. Resistivity as a function of width

The figure below illustrates the defining of the conductor layer.

Figure 3: Example Conductor Definition



```
        ⋮
conductor "POLYCIDE" {
    min_spacing            0.25
    min_width              0.16
    height                 0.35
    thickness              0.20
    resistivity            8.6
    gate_forming_layer     true
}
conductor "M1" {
    min_spacing            0.30
    min_width              0.30
    height                 0.85
    thickness              0.25
    resistivity            8.0
    gate_forming_layer     false
}
        ⋮
```

Example File for Conductor Definition

```
conductor "POLYCIDE" {

    min_spacing             0.25

    min_width               0.16

    height                  0.35

    upto                    0.55

    resistivity             8.6

    gate_forming_layer      true

}

conductor "M1" {

    min_spacing             0.30
    min_width                 0.30
    delta_layer               POLYCIDE
    delta_height              0.30
    thickness                 0.25
    resistivity               8.0
    gate_forming_layer        false
    wire_top_enlargement 0.01
    wire_bottom_enlargement -0.01
    wire_edge_enlargement {
      wee_widths      0.18  0.00   0.26   0.30   0.34
      wee_spacings    0.18 0.00   0.26   0.30   0.34   0.38
      wee_adjustments 0.00 0.00 -0.10 -0.10 -0.20
                      0.00 0.00   0.00 -0.10 -0.20
                      0.10 0.00   0.00   0.00 -0.10
                      0.10  0.10  0.00   0.00    0.00
                      0.20  0.20   0.10  0.00  0.00
                      0.30  0.20   0.20   0.10   0.00
    }

}
```

Dielectric

The `dielectric` command defines dielectric layers.

All measurements are in microns unless otherwise specified.

Syntax

```
dielectric name {conformal value field1 value1 ... fieldN valueN}
```

or

```
dielectricname{
    conformalvalue
   field1 value1
    ...
   fieldN valueN
}
```

You can specify the *`field-value`* pairs in any order.

The syntax for planar dielectrics contains the following parameters:

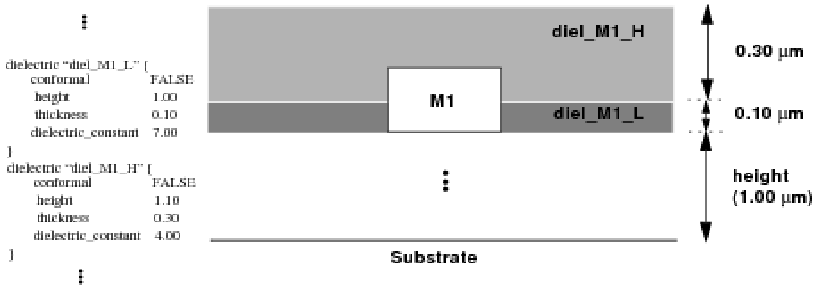
- `name`
  Specifies the name of the dielectric layer.
- `conformal false`
  Specifies that the dielectric is planar. This field is required.
- `height value`
  Specifies the layer's height above the substrate.
- `thickness value`
  Specifies the layer's thickness.
- `dielectric_constant value`
  Specifies the dielectric constant for this material.
- `delta_height value`
  Specifies the layer's height relative to the top of another layer.
- `delta_layer layer_name`
  Specifies the reference layer for `delta_height`. It must be a layer that has already been defined. A reference layer can be a conducting layer or a dielectric layer.
- `upto value`
  Specifies the layer's top surface height above the substrate. This value is equal to the height plus the thickness. You only need to specify two of the three parameters (`height` (or {delta_height, delta_layer}), thickness, upto) to complete the geometrical definition of a dielectric layer.

The required fields in the specification for planar dielectrics are `conformal`, `dielectric_constant`, and two of the following three parameters:

- `height` (or {delta_height and delta_layer})
- `thickness`
- `upto`

The figure below illustrates the planar dielectric syntax.

Figure 4: Planar Dielectric Syntax



```
        ⋮
dielectric "diel_M1_L" [
    conformal            FALSE
     height              1.00
     thickness           0.10
    dielectric_constant  7.00
]
dielectric "diel_M1_H" [
    conformal            FALSE
     height              1.10
     thickness           0.30
    dielectric_constant  4.00
]
        ⋮
```

Passivation

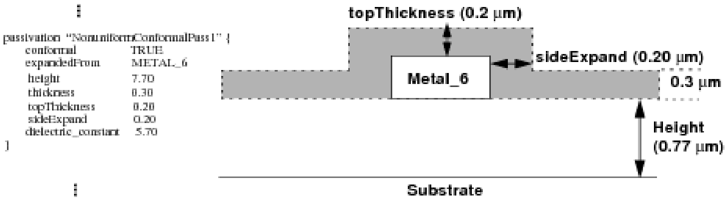
The `passivation` command defines passivation layers. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

Syntax

The syntax of this command is the same as that of the `dielectric` command, except that *`name`* specifies the name of the passivation layer. See the "Dielectric" section for information on this syntax. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

The figure below illustrates the defining of the passivation layer.

Figure 5: Passivation Syntax



```
        ⋮
passivation "NonuniformConformalPass1" {
    conformal           TRUE
    expandedFrom        METAL_6
     height             7.70
     thickness          0.30
     topThickness        0.20
     sideExpand          0.20
    dielectric_constant   5.70
}
        ⋮
```

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

A sample ICT file is provided below.

```
#

# Copyright (c) 2003 Cadence Design Systems, Inc.

#

##########################################################################

# Process declaration.

##########################################################################

process "DIFFERENT_KINDS_OF_DIELECTRIC" {

    background_dielectric_constant 1.0

}

##########################################################################

# Well declarations.
```

```
##########################################################################
well nwell {}
well pwell {}
##########################################################################
# Diffusion declarations.
##########################################################################
diffusion "N_SOURCE_DRAIN" {
# Tox is (height of POLYCIDE - thickness of diffusion) = (0.35 - 0.3455) = 0.0045um
    thickness 0.3455
    resistivity 7.7
}
diffusion "P_SOURCE_DRAIN" {
    thickness 0.3455
    resistivity 8.3
}
##########################################################################
# Conducting layer declarations.
##########################################################################
conductor "POLYCIDE" {
    min_spacing 0.25
    min_width 0.16
    height 0.35
    thickness 0.20
    resistivity 8.6
    gate_forming_layer true
}
conductor "METAL_1" {
    min_spacing 0.23
    min_width 0.23
    height 1.05
    thickness 0.53
    resistivity 0.086
    gate_forming_layer false
}
conductor "METAL_2" {
    min_spacing 0.28
    min_width 0.28
    height 2.38
    thickness 0.53
    resistivity 0.086
# The key words TRUE and FALSE are not case sensitive.
    gate_forming_layer FALSE
}
conductor "METAL_3" {
    min_spacing 0.28
    min_width 0.28
    height 3.71
    thickness 0.53
    resistivity 0.086
    gate_forming_layer false
}
conductor "METAL_4" {
    min_spacing 0.28
    min_width 0.28
# delta_height + delta_layer is an alternative to height.
    delta_height 0.80
    delta_layer METAL_3
# "height" is then redundant but it's okay to specify.
#    height 5.04
    thickness 0.53
    resistivity 0.086
    gate_forming_layer false
}
conductor "METAL_5" {
    min_spacing 0.28
    min_width 0.28
    height 6.37
    thickness 0.53
    resistivity 0.086
    gate_forming_layer false
}
conductor "METAL_6" {
    min_spacing 0.46
    min_width 0.44
    height 7.70
    thickness 0.99
    resistivity 0.035
    gate_forming_layer false
}
#########################################################################
# Dielectric and passivation layer declarations.
#########################################################################
#########################################################################
# Base dielectric from substrate...
#########################################################################
dielectric "First_dielectric" {
# Starts at height zero.
    conformal FALSE
    height 0.00
    thickness 0.35
    dielectric_constant 3.90
}
# Simple planar dielectric starts at the bottom of POLYCIDE
# and ends at 1.08um which is 0.03um above the bottom of M1.
dielectric "SimplePlanar1" {
# Starts at height of Poly
    conformal FALSE
    height 0.35
    thickness 0.73
    dielectric_constant 4.00
}
#########################################################################
# M1 level...
#########################################################################
# Now a planar intra-metal (M1) dielectric starts 0.03um above from the
# bottom of M1.
dielectric "PlanarIntraMetal1" {
    conformal FALSE
#
# Starts at height of M1
    height 1.08
```

```
# Laterally intersect with M1
    thickness 0.03
    dielectric_constant 7.00
}
# The second intra-metal dielectric across M1
# and on top of "PlanarIntraMetal1".
dielectric "PlanarIntraMetal2" {
# Yet another intra-metal planar dielectric layer.
    conformal FALSE
    height 1.11
    upto 1.15
# OR
#    thickness 0.04
    dielectric_constant 3.00
}
# A conformal dielectric.
# When specifying a conformal dielectric (whether it is uniform or
# non-uniform, we must use "conformal TRUE", "expandedFrom", "sideExpand",
# and "topThickness" together.
#
# 1. "conformal" must be set to TRUE.
# 2. "expandedFrom" can be a metal layer or a dielectric/passivation layer.
# The conformal dielectric layer must be expanded from its immediate
# lower (metal/dielectric/passivation) layer. It cannot be expanded
# from a planar dielectric layer.
# 3. "thickness" is the bottom dielectric thickness.
# 4. "sideExpand" specifies the side thickness.
# 5. "topThickness" is the thickness of the dielectric above the
# top of the "expandedFrom" layer.
dielectric "conformalAtTopOFM1" {
# Conformal above M1
    conformal TRUE
    expandedFrom METAL_1
# and starts from the top of "PlanarIntraMetal2"
    height 1.15
# Base/Bottom thickness of the conformal dielectric.
    thickness 0.43
```

```
# The thickness of the dielectric above the "expandedFrom" object, i.e. M1.
    topThickness 0.43
# This is the side thickness of the dielectric.
    sideExpand 0.43
    dielectric_constant 4.10
}
dielectric "SimplePlanar2" {
# From top of M1 to bottom of M2
    conformal FALSE
    height 1.58
    thickness 0.80
    dielectric_constant 4.00
}
#########################################################################
# M2 level...
#########################################################################
# An uniform conformal dielectric starting from the bottom of M2.
dielectric "UniformConformal1" {
    conformal TRUE
    expandedFrom METAL_2
# Height of M2
    delta_height 0.00
        delta_layer SimplePlanar2
#    height 2.38
    thickness 0.50
    topThickness 0.50
    sideExpand 0.50
    dielectric_constant 3.00
}
# A nonuniform conformal dielectric is one when any one of "thickness",
# "sideExpand", and "topThickness" are different.
dielectric "NonuniformConformal1" {
    conformal TRUE
    height 2.88
    thickness 0.10
    expandedFrom UniformConformal1
    sideExpand 0.03
    topThickness 0.05
    dielectric_constant 7.00
}
dielectric "SimplePlanar3" {
    conformal FALSE
    height 2.98
    thickness 0.73
    dielectric_constant 4.10
}
#########################################################################
# M3 level...
#########################################################################
# A special case of conformal dielectric.
dielectric "NonuniformConformal2" {
# Humps over M3 with side and top thicknesses equal to 0.17 um and 0.50 um, respectively.
    conformal TRUE
    expandedFrom METAL_3
    height 3.71
# Note that the bottom thickness is thicker than M3!
    thickness 0.90
    topThickness 0.50
    sideExpand 0.17
    dielectric_constant 4.10
}
dielectric "SimplePlanar5" {
    conformal FALSE
    height 4.61
# Upto the bottom of M4.
    upto 5.04
    dielectric_constant 3.00
}
#########################################################################
# M4 level...
#########################################################################
dielectric "NonuniformConformal3" {
    conformal TRUE
    expandedFrom METAL_4
```

```
# Height of M4
    height 5.04
    thickness 0.30
    topThickness 0.30
    sideExpand 0.10
# Special case. See SimplePlanar6.
    dielectric_constant 4.10
}
dielectric "PlanarIntraMetal3" {
# Planar intrametal dielectric.
    conformal FALSE
    height 5.34
    upto 5.44
    dielectric_constant 3.10
}
dielectric "PlanarIntraMetal4" {
# Top off the top of NonuniformConformal3.
    height 5.44
    upto 5.87
    dielectric_constant 3.00
}
dielectric "SimplePlanar6" {
    conformal FALSE
    height 5.87
# Upto the bottom of M5.
    upto 6.37
# NOTE that it has the same dielectric constant as NonuniformConformal3.
# This makes "NonuniformConformal3" a special case.
    dielectric_constant 4.10
}
#########################################################################
# M5 level...
#########################################################################
dielectric "UniformConformal3" {
    conformal TRUE
    expandedFrom METAL_5
    height 6.37
    thickness 0.10
    topThickness 0.10
    sideExpand 0.10
    dielectric_constant 7.20
}
dielectric "PlanarIntraMetal5" {
# Special planar dielectric which intersects "UniformConformal3"
    conformal FALSE
    height 6.47
    thickness 0.40
    dielectric_constant 3.00
}
dielectric "PlanarIntraMetal6" {
# Another special planar dielectric which intersects "UniformConformal3"
    conformal FALSE
    height 6.87
    thickness 0.10
    dielectric_constant 4.00
}
dielectric "PlanarIntraMetal7" {
# Yet another special planar dielectric which intersects "UniformConformal3"
    conformal FALSE
    height 6.97
    thickness 0.03
    dielectric_constant 7.00
}
########################################################################
passivation "PlanarPass1" {
# From top of M5 to bottom of M6.
    conformal FALSE
    height 7.00
    thickness 0.70
    dielectric_constant 4.00
}
########################################################################
# M6 level...
########################################################################
```

```
passivation "NonuniformConformalPass1" {

    conformal TRUE

    expandedFrom METAL_6

    height 7.70

    thickness 0.30

    topThickness 0.20

    sideExpand 0.20

    dielectric_constant 5.70

}

passivation "PlanarPass2" {

    conformal FALSE

    height 8.00

    upto 8.89

    dielectric_constant 4.30

}

#########################################################################

passivation "PlanarPass3" {

    conformal FALSE

     height 8.89

    thickness 1.00

    dielectric_constant 3.00

}

#########################################################################

# Contacts and Via declarations.

#########################################################################

via "CONT" {

    top_layer METAL_1

    bottom_layer POLYCIDE

    contact_resistance 7.8

}

via "CONT" {

    top_layer METAL_1

    bottom_layer N_SOURCE_DRAIN

    contact_resistance 11

}

via "CONT" {

    top_layer METAL_1
    bottom_layer P_SOURCE_DRAIN
    contact_resistance 10
}
via "VA1" {
    top_layer METAL_2
    bottom_layer METAL_1
    contact_resistance 7.9
}
via "VA2" {
    top_layer METAL_3
    bottom_layer METAL_2
    contact_resistance 8.2
}
via "VA3" {
    top_layer METAL_4
    bottom_layer METAL_3
    contact_resistance 8.1
}
via "VA4" {
    top_layer METAL_5
    bottom_layer METAL_4
    contact_resistance 8.0
}
via "VA5" {
    top_layer METAL_6
    bottom_layer METAL_5
    contact_resistance 4.0
}
```

Related Information

- Creating the ICT File

Supported CPF 1.0 Commands

Note: The following commands are supported unless otherwise noted.

Command Name	Option	Notes

		N/A = not available in this release
create_analysis_view		
	-name	
	-mode	
	-domain_corners	
create_bias_net		
	-net	
	-driver	N/A
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_ground_nets		
	-nets	
	-voltage	N/A
	-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_volage_range	
	-direction	
	-output_voltage_input_pin	N/A
	-input_power_pin	
	-output_power_pin	
	-ground	
	-valid_location	
define_open_source_input_pin		
	-cells	
	-pin	
	-library_set	
define_power_clamp_cell		N/A
	-cells	
	-data	
	-ground	
	library_set	
	-power	
define_power_switch_cell		

	-cells	
	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-power_switchable	
	-power	
	-ground	
	-ground_switchable	
	-on_resistance	Accessible by ::CPF::getCpfPsoCell
	-stage_1_saturation_current	Accessible by ::CPF::getCpfPsoCell
	-stage_2_saturation_current	Accessible by ::CPF::getCpfPsoCell
	-leakage_current	Accessible by ::CPF::getCpfPsoCell
define_state_retention_cell		
	-cells	
	-library_set	
	-always_on_pin	N/A
	-clock_pin	N/A
	-restore_function	
	-restore_check	N/A
	-save_function	
	-save_check	N/A
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_library_set		

	-name	
	-libraries	
end_design		
identify_always_on_driver		N/A
identify_power_logic		
	-type	
	-instances	
set_array_naming_style		
set_cpf_version		
set_design		
	-ports	
	module	
set_hierarchy_separator		
set_instance		
	-port_mapping	
	-merge_default_domains	
	hier_instance	
set_power_target		N/A
set_power_unit		N/A
set_register_naming_style		
set_switching_activity		
	-all	
	-pins	
	-instances	
	-hierarchical	
	-probability	
	-toggle_rate	
	-clock_pins	N/A
	-toggle_percentage	N/A
	-mode	N/A
set_time_unit		

update_isolation_rules		
	-names	`
	-location	
	-cells	
	-library_set	
	-prefix	
	-combine_level_shifting	N/A
	-open_source_pins_only	
update_level_shifter_rules		
	-names	
	-location	
	-cells	
	-library_set	
	-prefix	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-name	
	-internal_power_net	
	-internal_ground_net	
	-min_power_up_time	N/A
	-max_power_up_time	N/A
	-pmos_bias_net	N/A
	-nmos_bias_net	N/A
	-user_attributes	Accessible by ::CPF::getCpfUserAttr
	-rail_mapping	N/A
	-library_set	
update_power_mode		
	-name	

	-activity_file	N/A
	-activity_file_weight	N/A
	-sdc_files	
	-peak_ir_drop_limit	N/A
	-average_ir_dropt_limit	N/A
	-leakage_power_limit	N/A
	-dynamic_power_limit	N/A
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-library_set	
	-prefix	
	-peak_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
	-average_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
update_state_retention_rules		
	-name	
	-cell_type	
	-cell	
	-library_set	

CPF 1.0 Script Example

The following section contains an example of the CPF 1.0 file using a sample design and library.

For list of supported CPF commands and options within Innovus product family, see "Supported CPF 1.0 Commands".

```
set_cpf_version 1.0


####################################################

# #
# Technology portion of the CPF: #
# Defining the special cells for low-power designs #
# #
####################################################

################################
#### High-to-Low level shifters
################################

define_level_shifter_cell -cells LVLH2L* \
-input_voltage_range 0.8:1.0:0.1 \
-output_voltage_range 0.8:1.0:0.1 \
-direction down \
-output_power_pin VDD \
-ground VSS \
-valid_location to

##########################################
#### Always-on High-to-low level shifters
##########################################

define_level_shifter_cell -cells AOLVLH2L* \
-input_voltage_range 0.8:1.0:0.1 \
-output_voltage_range 0.8:1.0:0.1 \
-direction down \
-output_power_pin TVDD \
-ground VSS \
-valid_location to


################################
#### Low-to-High Level Shifters
################################
define_level_shifter_cell -cells LVLL2H* \
-input_voltage_range 0.8:1.0:0.1 \
-output_voltage_range 0.8:1.0:0.1 \
-input_power_pin VDDI \
-output_power_pin VDD \

-direction up \

-ground VSS \

-valid_location to


##########################################################

#### Low-to-High level shifting plus isolation combo cells

##########################################################

define_level_shifter_cell -cells LVLCIL2H* \

-input_voltage_range 0.8:1.0:0.1 \

-output_voltage_range 0.8:1.0:0.1 \

-output_voltage_input_pin ISO \

-input_power_pin VDDI \

-output_power_pin VDD \

-direction up \

-ground VSS \

-valid_location to


####################

#### Isolation cells

####################


define_isolation_cell -cells LVLCIL2H* \

-power VDD \

-ground VSS \

-enable ISO \

-valid_location to


#################################

#### Power switch cells: headers

#################################


define_power_switch_cell -cells {HEADERHVT HEADERAOPHVT} \

-power_switchable VDD -power TVDD \

-stage_1_enable !ISOIN1 \

-stage_1_output ISOOUT1 \

-stage_2_enable !ISOIN2 \
-stage_2_output ISOOUT2 \
-type header

#################
#### SRPG cells
#################

define_state_retention_cell -cells { SRPG2Y } \
-clock_pin CLK \
-power TVDD \
-power_switchable VDD \
-ground VSS \
-save_function "SAVE" \
-restore_function "!NRESTORE"

################################################
#### Always-on cells: buffers and level shifters
################################################

define_always_on_cell -cells {AOBUFF2Y AOLVLH2L*} \
-power_switchable VDD -power TVDD -ground VSS
#################################################
# #
# Design part of the CPF #
# #
#################################################

set_design top

set_hierarchy_separator "/"

set constraintDir ../CONSTRAINTS

set libdir ../LIBS

####################################################
#### create the power and ground nets in this design
```

```
####################################################

### VDD will connect the power follow-pin of the instances in the always-on
#power domain
### VDD_core_SW will connect the power follow-pin of the instances in the
#switchable power domain and is the power net that can be shut-off
### VDD_core_AO is the always-on power net for the switchable power domain

create_power_nets -nets VDD -voltage {0.8:1.0:0.1}
create_power_nets -nets VDD_core_AO -voltage 0.8
create_power_nets -nets VDD_core_SW -internal -voltage 0.8
create_power_nets -nets AVDD -voltage 1.0

create_ground_nets -nets VSS
create_ground_nets -nets AVSS

#####################################################################
#### Creating three power domains:
#### AO is the default always-on power domain
#### CORE is the switchable power domain
#### PLL is another always-on power domain
### Also specifying the power net-pin connection in each power domain
######################################################################

#########################
### For power domain "AO"
#########################

create_power_domain -name AO -default
update_power_domain -name AO -internal_power_net VDD

create_global_connection -domain AO -net VDD -pins VDD
create_global_connection -domain AO -net VSS -pins VSS
create_global_connection -domain AO -net VDD_core_SW -pins VDDI

###########################
### For power domain "CORE"
```

```
###########################

create_power_domain -name core -instances CORE_INST \
-shutoff_condition {PWR_CONTROL/power_switch_enable}
update_power_domain -name core -internal_power_net VDD_core_SW

create_global_connection -domain CORE -net VSS -pins VSS
create_global_connection -domain CORE -net VDD_core_AO -pins TVDD
create_global_connection -domain CORE -net VDD_core_SW -pins VDD

###########################
### For power domain "PLL"
###########################

### PLL conatins a single PLL macro and five top-level boundary ports which
#connect to the PLL macro directly

create_power_domain -name PLL -instances PLLCLK_INST -boundary_ports \
{refclk vcom vcop ibias pllrst}
update_power_domain -name PLL -internal_power_net AVDD

create_global_connection -domain PLL -net AVDD -pins avdd!
create_global_connection -domain PLL -net AVSS -pins agnd!
create_global_connection -domain PLL -net VDD -pins VDDI
create_global_connection -domain PLL -net AVDD -pins VDD
create_global_connection -domain PLL -net AVSS -pins VSS

########################################
########################################

set lib_1p1_wc "$libdir/technology45_std_1p1.lib"
set lib_1p3_bc "$libdir/technology45_std_1p3.lib"
set lib_1p0_bc "$libdir/technology45_std_1p0.lib"
set lib_0p8_wc "$libdir/technology45_std_0p8.lib"


set lib_ao_wc_extra "\
$libdir/technology45_lvll2h_1p1.lib \
"
set lib_ao_bc_extra "\
$libdir/technology45_lvll2h_1p3.lib \
"
set lib_core_wc_extra "\
$libdir/technology45_lvlh2l_0p8.lib \
$libdir/technology45_headers_0p8.lib \
$libdir/technology45_sprg_ao_0p8.lib \
"
set lib_core_bc_extra "\
$libdir/technology45_lvlh2l_1p0.lib \
$libdir/technology45_headers_1p0.lib \
$libdir/technology45_srpg_ao_1p0.lib \
"

set lib_pll_wc "\
$libdir/pll_slow.lib \
$libdir/ram_256x16_slow.lib \
$libdir/rom_512x16_slow.lib \
"
set lib_pll_bc "\
$libdir/pll_fast.lib \
$libdir/ram_256x16_fast.lib \
$libdir/rom_512x16_fast.lib \
"
########################
#### Define library sets
########################
define_library_set -name ao_wc_0p8 -libraries "$lib_0p8_wc $lib_ao_wc_extra"
define_library_set -name ao_bc_1p0 -libraries "$lib_1p0_bc $lib_ao_bc_extra"

define_library_set -name ao_wc_1p1 -libraries "$lib_1p1_wc_base $lib_ao_wc_extra"
define_library_set -name ao_bc_1p3 -libraries "$lib_1p3_bc_base $lib_ao_bc_extra"

define_library_set -name core_wc_0p8 -libraries "$lib_0p8_wc $lib_core_wc_extra"
define_library_set -name core_bc_1p0 -libraries "$lib_1p0_bc $lib_core_bc_extra"
define_library_set -name pll_wc_1p1 -libraries "$lib_pll_wc"

define_library_set -name pll_bc_1p3 -libraries "$lib_pll_bc"


#############################

#### Create operating corners

#############################


create_operating_corner -name BC_PVT_AO_L \

-process 1 -temperature 0 -voltage 1.0 \

-library_set ao_bc_1p0


create_operating_corner -name WC_PVT_AO_L \

-process 1 -temperature 125 -voltage 0.8 \

-library_set ao_wc_0p8


create_operating_corner -name BC_PVT_AO_H \

-process 1 -temperature 0 -voltage 1.3 \

-library_set ao_bc_1p3


create_operating_corner -name WC_PVT_AO_H \

-process 1 -temperature 125 -voltage 1.1 \

-library_set ao_wc_1p1


create_operating_corner -name BC_PVT_CORE \

-process 1 -temperature 0 -voltage 1.0 \

-library_set core_bc_1p0


create_operating_corner -name WC_PVT_CORE \

-process 1 -temperature 125 -voltage 0.8 \

-library_set tdsp_wc_0p8



create_operating_corner -name BC_PVT_PLL \

-process 1 -temperature 0 -voltage 1.3 \

-library_set core_bc_1p3
```

```
create_operating_corner -name WC_PVT_PLL \
-process 1 -temperature 125 -voltage 1.1 \
-library_set tdsp_wc_1p1


#########################################
#### Create and update nominal conditions
#########################################

create_nominal_condition -name high_ao -voltage 1.1
update_nominal_condition -name high_ao -library_set ao_wc_1p1

create_nominal_condition -name low_ao -voltage 0.8
update_nominal_condition -name low_ao -library_set ao_wc_0p8

create_nominal_condition -name low_core -voltage 0.8
update_nominal_condition -name low_core -library_set core_wc_0p8

create_nominal_condition -name high_pll -voltage 1.1
update_nominal_condition -name high_pll -library_set pll_wc_1p1

create_nominal_condition -name off -voltage 0

######################################
#### Create and upDate four power modes
######################################

create_power_mode -name PM_HL_FUNC \
-domain_conditions {AO@high_ao CORE@low_core PLL@high_pll} \
-default
update_power_mode -name PM_HL_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_HL_TEST \
-domain_conditions {AO@high_ao CORE@low_core PLL@high_pll}
update_power_mode -name PM_HL_TEST -sdc_files ${constraintDir}/top_test.sdc

create_power_mode -name PM_HO_FUNC \
-domain_conditions {AO@high_ao CORE@off PLL@high_pll}
update_power_mode -name PM_HO_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_LO_FUNC \
-domain_conditions {AO@low_ao CORE@off PLL@high_pll}
update_power_mode -name PM_LO_FUNC -sdc_files ${constraintDir}/top_slow.sdc

#################################
##### Creating ten analysis views
#################################

create_analysis_view -name AV_HL_FUNC_MIN_RC1 -mode PM_HL_FUNC \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MIN_RC2 -mode PM_HL_FUNC \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}

create_analysis_view -name AV_HL_FUNC_MAX_RC1 -mode PM_HL_FUNC \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MAX_RC2 -mode PM_HL_FUNC \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_HL_SCAN_MIN_RC1 -mode PM_HL_TEST \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_SCAN_MAX_RC1 -mode PM_HL_TEST \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_HO_FUNC_MIN_RC1 -mode PM_HO_FUNC \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HO_FUNC_MAX_RC1 -mode PM_HO_FUNC \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_LO_FUNC_MIN_RC1 -mode PM_LO_FUNC \
-domain_corners {AO@BC_PVT_AO_L CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_LO_FUNC_MAX_RC1 -mode PM_LO_FUNC \
-domain_corners {AO@WC_PVT_AO_L CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

#########################################################
```

```
#### Creating and updating the rules for the insertion

#### of power switch, level shifter, isolation cell

#########################################################


###########################

##### One power switch rule

###########################


create_power_switch_rule -name PWRSW_CORE -domain CORE \

-external_power_net VDD_core_AO


update_power_switch_rule -name PWRSW_CORE \

-cells HEADERHVT \

-prefix CDN_SW_ \

-acknowledge_receiver SIWTCH_ENOUT

#######################################################################

##### One isolation rule using level-shifting and isolation combo cells

#######################################################################


create_isolation_rule -name ISORULE -from CORE \

-isolation_condition "!PWR_CONTROL/isolation_enable" \

-isolation_output high


update_isolation_rules -names ISORULE -location to -cells LVLCIL2H2Y


################################

##### Three level shifting rules

################################


#### For signals from AO to CORE


create_level_shifter_rule -name LSRULE_H2L -from AO -to CORE \

-exclude {PWR_CONTROL/power_switch_enable PWR_CONTROL \

        /state_retention_enable PWR_CONTROL/state_retention_restore}

update_level_shifter_rules -names LSRULE_H2L -cells LVLH2L2Y -location to


#### Only for the control signals from AO to CORE
```

```
create_level_shifter_rule -name LSRULE_H2L_AO -from AO -to CORE \
-pins {PWR_CONTROL/power_switch_enable PWR_CONTROL/state_retention_enable\ PWR_CONTROL/state_retention_restore}
update_level_shifter_rules -names LSRULE_H2L_AO -cells AOLVLH2L2Y -location to

#### For signals from PLL to AO

create_level_shifter_rule -name LSRULE_H2L_PLL -from PLL -to AO
update_level_shifter_rules -names LSRULE_H2L_PLL -cells LVLH2L2Y -location to

####################
##### One SRPG rule
####################

create_state_retention_rule -name SRPG_CORE \
-domain CORE \
-restore_edge {!PWR_CONTROL/state_retention_restore} \
-save_edge {PWR_CONTROL/state_retention_enable}

update_state_retention_rules -names SRPG_CORE \
-cell SRPG2Y \
-library_set tdsp_wc_0v792

end_design
```

Supported CPF 1.0e Commands

Note: The following commands and options are supported unless otherwise noted.

Command Name	Option	Notes
create_analysis_view		
	-name	
	-mode	
	-domain_corners	
	-group_views	
	-user_attributes	
create_assertion_control		

	-name	Unsupported
	-assertions	Unsupported
	-domains	Unsupported
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-net	
	-driver	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_power_domain		
	-name	
	-instances	
	-boundary_ports	
	-default	
	-shutoff_condition	
	-external_controlled_shutoff	
	-default_isolation_condition	
	-default_restore_edge	
	-default_save_edge	
	-default_restore_level	Supported
	-default_save_level	Supported
	-power_up_states	Unsupported
	-active_state_condition	Unsupported

	-secondary_domains	
create_ground_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_isolation_rule		
	-name	
	-isolation_condition	
	-no_condition	Unsupported
	-pins	
	-from	
	-to	
	-exclude	
	-isolation_target	
	-isolation_output	
	-secondary_domain	
create_level_shifter_rule		
	-name	
	-pins	
	-from	
	-to	
	-exclude	
create_mode_transition		
	-name	
	-from	
	-to	
	-start_condition	

	-end_condition	
	-cycles	
	-clock_pin	
	-latency	
create_nominal_condition		
	-name	
	-voltage	
	-ground_voltage	
	-state	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
create_operating_corner		
	-name	
	-voltage	
	-ground_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
	-process	
	-temperature	
	-library_set	
create_power_mode		
	-name	
	-default	
	-group_modes	
	-domain_conditions	
create_power_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes

	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_power_switch_rule		
	-name	
	-domain	
	-external_power_net	
	-external_ground_net	
create_state_retention_rule		
	-name	
	-domain	
	-instances	
	-exclude	
	-restore_edge	
	-save_edge	
	-restore_precondition	
	-save_precondition	
	-target_type	
	-secondary_domain	
define_always_on_cell		
	-cells	
	-library_set	
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_isolation_cell		
	-cells	
	-library_set	
	-always_on_pins	
	-power_switchable	
	-ground_switchable	

	-power	
	-ground	
	-valid_location	
	-enable	
	-no_enable	Unsupported
	-non_dedicated	
define_level_shifter_cell		
	-cells	
	-library_set	
	-always_on_pins	
	-input_voltage_range	
	-output_voltage_range	
	-ground_output_voltage_range	Unsupported
	-groung_output_voltage_range	
	-direction	
	-input_power_pin	
	-output_power_pin	
	-input_ground_pin	Unsupported
	-output_ground_pin	Unsupported
	-ground	
	-power	Unsupported
	-enable	
	-valid_location	
define_library_set		
	-name	
	-libraries	
	-user_attributes	cdb: specify cdb libraries for the library set aocv: specify aocv libraries for the library set

define_power_clamp_cell		
	-location	Unsupported
	-within_hierarchy	Unsupported
	-cells	Unsupported
	-prefix	Unsupported
define_power_switch_cell		
	-cells	
	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-power_switchable	
	-power	
	-ground_switchable	
	-ground	
	-on_resistance	Supported (for use with addPowerSwitch)
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-leakage_current	Supported (for use with addPowerSwitch)
define_state_retention_cell		
	-cells	
	-library_set	

	-cell_type	
	-always_on_pins	
	-clock_pin	
	-restore_function	
	-save_function	
	-restore_check	
	-save_check	
	-always_on_components	Unsupported
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
end_design		
end_macro_model		
end_power_mode_control_group		
get_parameter		
include		
identify_power_logic		
	-type	Only "isolation" is supported for the -type
	-instances	Supported
	-module	Supported
set_array_naming_style		
set_cpf_version		
set_hierarchy_separator		
set_design		
	-ports	
	-honor_boundary_port_domain	
	-parameters	
set_equivalent_control_pins		

	-master	Unsupported
	-pins	Unsupported
	-domain	Unsupported
	-rules	Unsupported
set_floating_ports		
set_input_voltage_tolerance		
	-ports	Unsupported
	-bias	Unsupported
set_instance		
	-design	
	-model	
	-port_mapping	
	-domain_mapping	
	-parameter_mapping	
set_macro_model		
set_power_mode_control_group		
	-name	
	-domains	
	-groups	Unsupported
set_power_target		
	-leakage	Unsupported
	-dynamic	Unsupported
set_power_unit		
set_register_naming_style		
set_switching_activity		
	-all	Supported
	-pins	Supported
	-instances	Supported
	-hierarchical	Supported
	-probability	Supported
	-toggle_rate	Supported

	-clock_pins	Unsupported
	-toggle_percentage	Unsupported
	-mode	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
	-open_source_pins_only	Supported
update_level_shifter_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-name	
	-primary_power_net	
	-primary_ground_net	
	-pmos_bias_net	Unsupported
	-nmos_bias_net	Unsupported
	-user_attributes	Supported: query getCPFUserAttributes
	-transition_slope	Unsupported
	-transition_latency	Unsupported

	-transition_cycles	Unsupported
update_power_mode		
	-name	
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-sdc_files	
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
	-leakage_power_limit	Unsupported
	-dynamic_power_limit	Unsupported
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-gate_bias_net	Unsupported
	-prefix	
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
update_state_retention_rules		
	-names	
	-cell_type	
	-cells	
	-set_rest_control	Unsupported

CPF 1.0e Script Example

The following section contains an example of the CPF 1.0e file using a sample design and library.

For list of supported CPF commands and options within Innovus product family, see "Supported CPF 1.0e Commands".

```
#-----------------------------------------------------------------------------------------------
```

```
# setting
#-------------------------------------------------------------------------------
set_cpf_version 1.0e
set_hierarchy_separator /

#-------------------------------------------------------------------------------
# define library_set/cells
#-------------------------------------------------------------------------------
define_library_set -name wc_0v81 -libraries { \
../LIBS/timing/library_wc_0v81.lib }
define_library_set -name bc_0v81 -libraries { \
../LIBS/timing/library_bc_0v81.lib }
define_library_set -name wc_0v72 -libraries { \
../LIBS/timing/library_wc_0v72.lib }
define_library_set -name bc_0v72 -libraries { \
../LIBS/timing/library_bc_0v72.lib }

define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
VDD -power TVDD -ground VSS

define_isolation_cell -cells { LVLLH* } -power VDD -ground VSS -enable \
NSLEEP -valid_location to
define_isolation_cell -cells { ISOHID* ISOLOD* } -power VDD -ground VSS \
-enable ISO -valid_location to

define_level_shifter_cell -cells { LVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { PTLVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHCD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \
-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 { BLSRPG* } -cell_type ballon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS

#-------------------------------------------------------------------------------
# macro models
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# top design
#-------------------------------------------------------------------------------
set_design top

create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0

create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0

create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0

create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0

create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \

-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0


create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0

create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0


create_nominal_condition -name nom_0v81 -voltage 0.81

create_nominal_condition -name nom_0v72 -voltage 0.72


#-------------------------------------------------------------------------------

# create power domains

#-------------------------------------------------------------------------------

create_power_domain -name PDdefault -default

create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifsip \

IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \

-external_controlled_shutoff -shutoff_condition io_shutoff_ack

create_power_domain -name PDpll -instances { INST/PLLCLK_INST \

IOPADS_INST/Pibiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip \

IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } -boundary_ports { ibias reset \

refclk vcom vcop pllrst }

create_power_domain -name PDram_virtual

create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \

-shutoff_condition !INST/PM_INST/power_switch_enable \

-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 ballon_latch

#--------------------------------------------------------------------------------
# end
#--------------------------------------------------------------------------------
end_design
```

Supported CPF 1.1 Commands

Note: The following commands and options are supported unless otherwise noted.

Command Name	**Option**	**Notes**
asset_illegal_domain_configurations		
	-domain_conditions	
	-group_modes	
	-name	
create_analysis_view		
	-domain_corners	
	-group_views	
	-mode	
	-name	
	-user_attributes	
create_assertion_control		
	-assertions	Unsupported
	-domains	Unsupported
	-exclude	
	-name	Unsupported
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-average_ir_drop_limit	
	-driver	
	-net	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
create_global_connection		
	-domain	
	-instances	
	-net	

	-pins	
create_ground_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_isolation_rule		
	-exclude	
	-from	
	-isolation_condition	
	-isolation_output	
	-isolation_target	
	-name	
	-no_condition	
	-pins	
	-secondary_domain	
	-to	
	-force	
create_level_shifter_rule		
	-exclude	
	-from	
	-name	
	-pins	
	-to	
	-force	
create_mode_transition		
	-clock_pin	
	-cycles	

	-end_condition	
	-from	
	-latency	
	-name	
	-to	
	-start_condition	
create_nominal_condition		
	-ground_voltage	
	-name	
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-state	Unsupported
	-voltage	
create_operating_corner		
	-ground_voltage	Unsupported
	-library_set	
	-name	
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-process	
	-temperature	
	-voltage	
create_power_domain		
	-active_state_condition	Unsupported
	-base_domains	
	-boundary_ports	
	-default	
	-default_isolation_condition	
	-default_restore_edge	
	-default_restore_level	Supported

	-default_save_edge	
	-default_save_level	Supported
	-external_controlled_shutoff	
	-instances	
	-name	
	-power_up_states	Unsupported
	-shutoff_condition	
create_power_mode		
	-default	
	-domain_conditions	
	-group_modes	
	-name	
create_power_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_power_switch_rule		
	-domain	
	-external_ground_net	
	-external_power_net	
	-name	
create_state_retention_rule		
	-domain	
	-exclude	
	-instances	
	-name	
	-restore_edge	

	-restore_precondition	
	-save_edge	
	-save_precondition	
	-secondary_domain	
	-target_type	
define_always_on_cell		
	-cells	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
define_isolation_cell		
	-always_on_pins	
	-cells	
	-enable	
	-ground	
	-ground_switchable	
	-library_set	
	-no_enable	
	-non_dedicated	
	-power	
	-power_switchable	
	-valid_location	
define_level_shifter_cell		
	-always_on_pins	
	-cells	
	-direction	
	-enable	
	-ground	
	-ground_input_voltage_range	

	-ground_output_voltage_range	
	-input_ground_pin	
	-input_power_pin	
	-input_voltage_range	
	-library_set	
	-output_ground_pin	
	-output_power_pin	
	-output_voltage_range	
	-power	
	-valid_location	
define_library_set		
	-libraries	
	-name	
	-user_attributes	cdb: specify cdb libraries for the library set aocv: specify aocv libraries for the library set
define_power_clamp_cell		
	-cells	Unsupported
	-data	Unsupported
	-power pin	Unsupported
	-ground	Unsupported
	-library_set	
define_power_switch_cell		
	-cells	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-ground	
	-ground_switchable	

	-leakage_current	Supported (for use with addPowerSwitch)
	-library_set	
	-power	
	-power_switchable	
	-stage_1_on_resistance	
	-stage_2_on_resistance	
	-stage_1_enable	
	-stage_1_output	
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_enable	
	-stage_2_output	
	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-type	
define_state_retention_cell		
	-always_on_components	Unsupported
	-always_on_pins	
	-cell_type	
	-cells	
	-clock_pin	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
	-restore_check	
	-restore_function	
	-save_check	

	-save_function	
end_design		
end_macro_model		
end_power_mode_control_group		
get_parameter		
include		
identify_power_logic		
	-instances	Supported
	-module	Supported
	-type	Only "isolation" is supported for the -type
set_array_naming_style		
set_cpf_version		
set_hierarchy_separator		
set_design		
	module	
	-ports	
	-honor_boundary_port_domain	
	-parameters	
set_equivalent_control_pins		
	-domain	Unsupported
	-master	Unsupported
	-pins	Unsupported
	-rules	
set_floating_ports		
set_input_voltage_tolerance		
	-bias	Unsupported
	-ports	Unsupported
set_instance		
	-design	

	-domain_mapping	
	-model	
	-parameter_mapping	
	-port_mapping	
set_macro_model		
set_power_mode_control_group		
	-domains	
	-groups	Unsupported
	-name	
set_power_target		
	-dynamic	Unsupported
	-leakage	Unsupported
set_power_unit		
set_register_naming_style		
set_switching_activity		
	-all	Supported
	-clock_pins	Unsupported
	-hierarchical	Supported
	-instances	Supported
	-mode	Supported
	-pins	Supported
	-probability	Supported
	-toggle_percentage	Unsupported
	-toggle_rate	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-cells	
	-location	
	-names	

	-open_source_pins_only	Supported
	-prefix	
	-within_hierarchy	
update_level_shifter_rules		
	-cells	
	-location	
	-names	
	-prefix	
	-within_hierarchy	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-equivalent_ground_nets	
	-equivalent_power_nets	
	-name	
	-nmos_bias_net	Unsupported
	-pmos_bias_net	Unsupported
	-primary_ground_net	
	-primary_power_net	
	-transition_cycles	Unsupported
	-transition_latency	Unsupported
	-transition_slope	Unsupported
	-user_attributes {{disable_secondary_domains {list of domains}}}	Supported: query getCPFUserAttributes
update_power_mode		
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-average_ir_drop_limit	
	-dynamic_power_limit	Unsupported

	-leakage_power_limit	Unsupported
	-name	
	-peak_ir_drop_limit	
	-sdc_files	
update_power_switch_rule		
	-acknowledge_reciever_1	
	-acknowledge_reciever_2	
	-average_ir_drop_limit	
	-cells	
	-enable_condition_1	
	-enable_condition_2	
	-gate_bias_net	Unsupported
	-name	
	-peak_ir_drop_limit	
	-prefix	
update_state_retention_rules		
	-cell_type	
	-cells	
	-names	
	-set_rest_control	Unsupported

CPF 1.1 Script Example

The following section contains an example of the CPF 1.1 file using a sample design and library.

For list of supported CPF commands and options within Innovus product family, see "Supported CPF 1.1 Commands".

```
#---------------------------------------------------------------------------------------------
# setting
#--------------------------------------------------------------------------------
set_cpf_version 1.1
set_hierarchy_separator /

#--------------------------------------------------------------------------------
# define library_set/cells
```

```
#------------------------------------------------------------------------------
define_library_set -name wc_0v81 -libraries { \
../LIBS/timing/library_wc_0v81.lib }
define_library_set -name bc_0v81 -libraries { \
../LIBS/timing/library_bc_0v81.lib }
define_library_set -name wc_0v72 -libraries { \
../LIBS/timing/library_wc_0v72.lib }
define_library_set -name bc_0v72 -libraries { \
../LIBS/timing/library_bc_0v72.lib }

define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
VDD -power TVDD -ground VSS

define_isolation_cell -cells { LVLLH* } -power VDD -ground VSS -enable \
NSLEEP -valid_location to
define_isolation_cell -cells { ISOHID* ISOLOD* } -power VDD -ground VSS \
-enable ISO -valid_location to

define_level_shifter_cell -cells { LVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { PTLVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHCD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \
-input_power_pin VDDL -output_power_pin VDD -ground VSS -valid_location to

define_power_switch_cell -cells { HEADERHVT1 HEADERHVT2 } \
-stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 -stage_2_enable \
NSLEEPIN2 -stage_2_output NSLEEPOUT2 -type header -power_switchable VDD \
-power TVDD -stage_1_on_resistance 10 - stage_2_on_resistance 10
```

```
define_state_retention_cell -cells { MSSRPG* } -cell_type \

master_slave -clock_pin CP -restore_check !CP -save_function !CP \

-always_on_components { DFF_inst } -power_switchable VDD -power TVDD \

-ground VSS

define_state_retention_cell -cells { BLSRPG* } -cell_type ballon_latch \

-clock_pin CP -restore_function !NRESTORE -save_function SAVE \

-always_on_components { save_data } -power_switchable VDD -power TVDD \

-ground VSS


#-------------------------------------------------------------------------------

# macro models

#-------------------------------------------------------------------------------


#-------------------------------------------------------------------------------

# top design

#-------------------------------------------------------------------------------

set_design top


create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \

0 -library_set bc_0v72

create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \

0 -library_set bc_0v81

create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \

125 -library_set wc_0v81

create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \

125 -library_set wc_0v72


create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0

create_power_nets -nets VDD_EQ -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0\

-average_ir_drop_limit 0

create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \

-peak_ir_drop_limit 0 -average_ir_drop_limit 0

create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0

create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \

-average_ir_drop_limit 0
```

```
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-------------------------------------------------------------------------------
# create power domains
#-------------------------------------------------------------------------------
create_power_domain -name PDdefault -default
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifsip \
IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
-external_controlled_shutoff -shutoff_condition io_shutoff_ack
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
IOPADS_INST/Pibiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip \
IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } -boundary_ports { ibias reset \
refclk vcom vcop pllrst }
create_power_domain -name PDram_virtual
create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \
-shutoff_condition !INST/PM_INST/power_switch_enable \
-base_domains { PDram_virtual }
create_power_domain -name PDtdsp -instances { INST/RAM_128x16_TEST_INST1 \
INST/DSP_CORE_INST0 INST/DSP_CORE_INST1 } -shutoff_condition \
!INST/PM_INST/power_switch_enable -base_domains { PDdefault }

#-------------------------------------------------------------------------------
# set instances
#-------------------------------------------------------------------------------
set_instance INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -domain_mapping \
{ {RAM_DEFAULT PDtdsp} }

set_macro_model ram_256x16A

create_power_domain -name RAM_DEFAULT -boundary_ports { A* D* CLK CEN WEN Q* } \
-default -external_controlled_shutoff

create_state_retention_rule -name RAM_ret -instances { mem* } -save_edge !CLK

update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
-primary_ground_net VSS

end_macro_model ram 256x16A

#-------------------------------------------------------------------------------
# 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 -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 ballon_latch


#--------------------------------------------------------------------------------

# end

#--------------------------------------------------------------------------------

end_design
```

Supported SAI Commands

The Soc Architecture Information (SAI) methodology is a powerful and self-contained design planning capability. It provides an ideal design/floorplan hand-off mechanism between front-end and back-end teams. Designers can turn a high-level block diagram into a real netlist and timing constraint (.sdc) ready for floorplanning and timing analysis much earlier without having a netlist or just with a partial netlist.

With SAI version 2.0, designers are able to specify floorplan constraints to guide module and/or macro placement, report floorplan quality index, and any constraint violations.

The following are the supported SAI Commands:

- `add_clock`
- `add_macro`
- `connect`
- `constrain`
- `convertLefToSAI`
- `create_module`
- `delete_macro`
- `delete_module`
- `insert_boundary_flops`
- `report_sai_constraint`
- `set_floorplan`
- `set_ref_flop`
- `set_ref_gate`
- `set_ref_macro`
- `set_ref_memory`
- `set_sai_version`

Note: You can use these commands in the SAI interactive mode or write in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

add_clock

```
add_clock
[-help]
clockName
-buffer string
[-default]
[-inst string]
-period string
[-waveform string]
```

Allows the software to add new clocks. The `add_clock` command creates clock root in netlist and adds the `create_clock` constraint in generated SDC along with connecting the clock root to clock port of hierarchical instances. If SDC is provided before SAI, then the `add_clock` command only connects clock root to clock port of hierarchical instances.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`clockName`	Specifies the clock name.
`-buffer` *`string`*	Specifies the buffer cell name. You can also specify “-” for clock port. • If buffer cell name is specified, instance of the cell is inserted as clock root in top level. • If “-” is specified, clock root is created for the external port on top level.
`-default`	Specifies the default clock that will be used by the `connect` command.
`-help`	Outputs a brief description that includes type and default information for each `add_clock` parameter.
`-inst` *`string`*	Specifies the hinst in which the clock buffer is to be added.
`-period` *`string`*	Specifies the clock period.
`-waveform` *`string`*	Specifies the clock waveform.

Related Information

- Supported SAI Commands

add_macro

```
add_macro
[-help]
[ref_counts]
-cell targetModule
[-memory string]
[-names string]
[-prefix string]
[-update_memory_bit_count string]
```

Allows the software to add macro instances to the netlist.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-cell` *`targetModule`*	Specifies target module to add macros.
`-help`	Outputs a brief description that includes type and default information for each `add_macro` parameter.

`-memory` *`string`*	Specifies the reference memory name created using the `set_ref_memory` command.
	For example,
	`set_ref_memory mem1 -bit_size 1024 -area_per_bit 1.1 -aspect_ratio 0.75`
	`create_module iBlock -cell mBlock -memory_bit_count 4096`
	`add_macro -cell mBlock -memory mem1`
	then, 4096 / 1024 = 4 “mem1” macros are added in mBlock module.

`-names` *`string`*	Specifies the names for adding memory and macro.
`-prefix` *`string`*	Specifies the prefix for adding memory and macro.
`ref_counts`	Specifies a list of macro cell name and count pair. For example, `{RAM1 12 RAM2 24 ...}`
`-update_memory_bit_count` *`string`*	Updates memory bit count for adding reference memory further.

Example

```
add_macro -cell ryon_mod1 {ram_128x16A 2} ...(A)
add_macro -cell ryon_mod1 {ram_128x16A 4} ...(B)
```

```
add_macro (A) creates
XM_PORT0/ram_128x16A_0
XM_PORT0/ram_128x16A_1
```

```
add_macro (B) creates
XM_PORT0/ram_128x16A_3
XM_PORT0/ram_128x16A_4
```

```
XM_PORT0/ram_128x16A_5
XM_PORT0/ram_128x16A_6
```

Related Information

- Supported SAI Commands

connect

```
connect
[-help]
src
[-bus_width integer]
[-clock string]
[-insert_driver_flop {auto | none | force}]
[-insert_receiver_flop {auto | none | force}]
[-pipeline_stages integer]
-to string
[-to_clock string]
[-weight integer]
```

Creates a net connection by connecting the source and destination clocks. You can use this command to connect top-module ports.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

The `connect` SAI command supports bus split and accepts the bus bit order in the msb:lsb format. It supports the following bus-oriented operations:

- msb:lsb format bus order:

  ```
  connect M3/out[0:63] -to M4/in[63:0] -clock clk
  ```

- Bus split:

  ```
  connect M4/out[0:15] -to M5/in[0:15] -clock clk
  connect M4/out[16:31] -to M8/in[0:15] -clock clk
  ```

  Note: The range of M4/out [0:15] and [16:31] should be continuous.

Parameters

`-bus_width` *`integer`*	Specified the bus width of connection. If not specified, single connection is created.
`-clock` *`string`*	Specifies clock name of source and destination flops. **Note**: The `connect` command uses the default clock (`add_clock -default`) if the clock information is not specified. It displays a warning if a default clock is not defined.
`-help`	Outputs a brief description that includes type and default information for each `connect` parameter.
`-insert_driver_flop {auto | none | force}`	Inserts source boundary flops. • `auto`: If the specified source module is created by SAI, then inserts source boundary flops automatically. • `none`: Does not insert source boundary flops. • `force`: Enforces the insertion of source boundary flops. *Default*: `auto`
`-insert_receiver_flop {auto | none | force}`	Inserts destination boundary flops. • `auto`: If the specified destination module is created by SAI, then inserts destination boundary flops automatically. • `none`: Does not insert destination boundary flops. • `force`: Enforces the insertion of destination boundary flops. *Default*: `auto`
`-pipeline_stages` *`integer`*	Specifies the number of pipeline flops between source flops and destination flops. If not specified, source flops and destination flops are connected directly.
`src`	Specifies the source hierarchical port or macro pin name.
`-to` *`string`*	Specifies the destination hierarchical port or macro pin name.
`-to_clock` *`string`*	Specifies clock name of destination flops. If not specified, destination flops are connected to *clock_name1*.
`-weight` *`integer`*	Specifies the net weight.

Related Information

- Supported SAI Commands

constrain

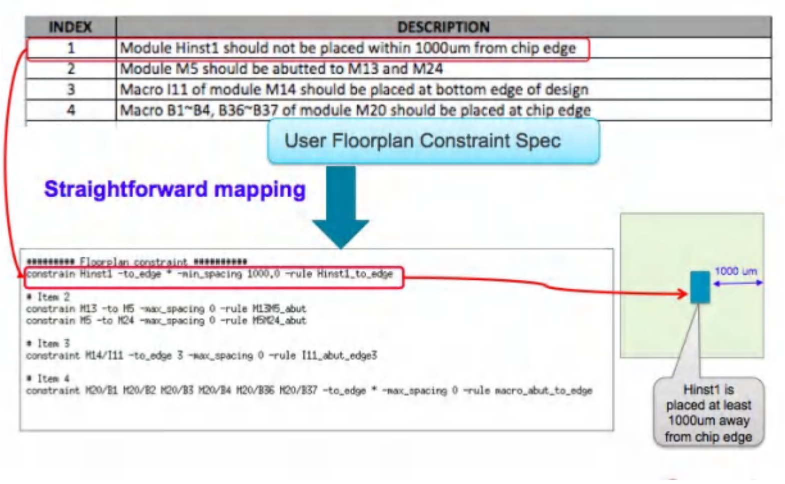
```
constrain
[-help]
from_names
[-max_spacing float]
[-min_spacing float]
[-order {none clockwise counterclockwise both}]
[-rule ruleName]
[-weight float]
[-to objectNameList | -group | -to_edge string | -area float | {[-min_area float] [-max_area float]}]
```

The constrain SAI command enables you to set the following constraints/rules:

- Place specified macros along the boundary.
- Place modules that have a minimum spacing to core box.
- Place modules at the minimum spacing from each other.
- Modules/macros abutment

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

The specified constraint from a spreadsheet can be mapped directly to a constrain command. The following illustration explains the mapping process.



INDEX	DESCRIPTION
1	Module Hinst1 should not be placed within 1000um from chip edge
2	Module M5 should be abutted to M13 and M24
3	Macro I11 of module M14 should be placed at bottom edge of design
4	Macro B1~B4, B36~B37 of module M20 should be placed at chip edge

User Floorplan Constraint Spec
Straightforward mapping
1000 um
Hinst1 is placed at least 1000um away from chip edge

```
######### Floorplan constraint ##########
constrain Hinst1 -to_edge * -min_spacing 1000,0 -rule Hinst1_to_edge

# Item 2
constrain M13 -to M5 -max_spacing 0 -rule M13M5_abut
constrain M5 -to M24 -max_spacing 0 -rule M5M24_abut

# Item 3
constraint M14/I11 -to_edge 3 -max_spacing 0 -rule I11_abut_edge3

# Item 4
constraint M20/B1 M20/B2 M20/B3 M20/B4 M20/B36 M20/B37 -to_edge * -max_spacing 0 -rule macro_abut_to_edge
```

Parameters

`-area` *`float`*	Specifies the area constraint of the specified module(s).
`from_names`	Specifies module or macro name(s). It does not support mixed type objects.
`-group`	Applies the specified constraint to all the specified objects in the list/group.
`-help`	Outputs a brief description that includes type and default information for each `constrain` parameter.
`-max_area` *`float`*	Specifies the maximum area constraint of the specified module(s)
`-max_spacing` *`float`*	Specifies the maximum spacing between specified objects. Objects will be abutted when the maximum spacing is 0.
`-min_area` *`float`*	Specifies minimum area constraint of the specified module(s)
`-min_spacing` *`float`*	Specifies the minimum spacing between specified objects.
`-order {none clockwise counterclockwise both}`	Applies constraints to objects in the specified order. The order can be clockwise, counter clockwise, or either of them specified using the both option. *Default*: `none`
`-rule` *`ruleName`*	Specifies the rule name.
`-to` *`objectNameList`*	Applies the specified constraint from the *`from_names`* to *`objectNameList.`*
`-to_edge` *`string`*	Constrains *from_names* to the specified core boundary edge. **Note:** Use * to constrain the specified module or macro to any core edge.
`-weight` *`float`*	Specifies the rule weight.

Example

- The following command specifies that Module B2 should not be placed within 770 microns from the core boundary:

```
constrain B2 -to_edge * -min_spacing 770.0
```

- The following command specifies that Modules B1, B2, B3, B4 should be placed far away from each other (-group) with a minimum distance of 2000um.

```
constrain B1 B2 B3 B4 -group -min_spacing 2000.0
```

- The following command abuts macro H1, H2, H3, H4 to the right edge (edge 2):

```
constrain H1 H2 H3 H4 -to_edge 1 -max_spacing 0 -order clockwise
```

- The following command abuts modules B3 B23 B24 in either clockwise or counterclockwise order:

```
constrain B3 B23 B24 -group -max_spacing 0 -order both
```

- The following command abuts macro H1, H2, H26~H28 to core boundary:

```
constrain H1 H2 H26 H27 H28 -to_edge * -max_spacing 0 \
-order clockwise -edge_orientation {0 R0 1 R0 2 MY 3 MX}
```

Related Information

- Supported SAI Commands

convertLefToSAI

```
convertLefToSAI
[-help]
-cells string
-clock string
-ref_flop string
```

Converts the LEF macro cells to SAI netlist.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-cells` *`string`*	Specifies the LEF macro cells to convert.
`-clock` *`string`*	Specifies the clock net name to attach the flops.
`-help`	Outputs a brief description that includes type and default information for each `convertLeftoSAI` parameter.
`-ref_flop` *`string`*	Specifies the flop cell name to be inserted along the boundary.

Related Information

- Supported SAI Commands

create_module

```
create_module
[-help]
name
[-aspect_ratio_range { min max }]
[-cell moduleName]
[-flop_count string]
[-flop_ref_clock clkName]
[-gate_count string]
[-memory_bit_count string]
[-util float]
[-incremental]
```

Generates hierarchical instance and creates the associated module if it does not exist yet. It automatically creates intermediate level instances and modules if missing.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-aspect_ratio_range {`*`min max`*`}`	Specifies the aspect ratio range for `proto_design`. These aspect ratios will appear in the *sai_proto)design.seed* file.
`-cell `*`moduleName`*	Specifies the module name. A module is created if the specified module name does not exist. If the module name is not specified, then the local hierarchical instance name is used as the module name. For example, `add_macro A/B/C/D` creates module D and its hierarchical instance A/B/C/D.
`-flop_count `*`string`*	Specifies the number of flops to insert as intermediate flops. It uses the cell set for `set_ref_flop`. To add flops with multiple clock domains, specify `{`*`numFlops1 clkName1 numFlops2 clkName2`*` ... }`. **Note**: You can also specify the flop count using an integer value. However, if you specify an integer, then you must specify the `-flop_ref_clock `*`clkName`* option. In this case, all added flops' clock pins are connected to the *`clkName`* port.
`-flop_ref_clock `*`clkName`*	Specifies the flop's reference clock
`-gate_count `*`string`*	Specifies the number of instance to insert. It uses the cell set for `set_ref_gate`.
`-help`	Outputs a brief description that includes type and default information for each `create_module` parameter.
`-incremental`	Specifies to incrementally fill incomplete modules that might have only port definition or have some logic inside.
`-memory_bit_count `*`string`*	Specifies the number of memory bits in the module.
`name`	Specifies hierarchical instance name to generate.
`-util `*`float`*	Specifies the target utilization value for `proto_design`. This target utilization appears in the *sai_proto_design.seed* file.

Related Information

- Supported SAI Commands

delete_macro

```
delete_macro
[-help]
[macroInstanceOrCellName]
[-cell moduleName]
```

Delete macro instances. It allows the software to delete macros created using the `add_macro` command.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-cell` *`moduleName`*	Specify the cell name that has the macro inside. Deletes all the macros that the *`macroInstanceOrCellName`* in module specified by *`moduleName`*.
`-help`	Outputs a brief description that includes type and default information for each `delete_macro` parameter.
`macroInstanceOrCellName`	Deletes the specified macro instance name or the macro's cell name if `-cell` is specified.

Related Information

- Supported SAI Commands

delete_module

```
delete_module
[-help]
hinstName
[-cell]
```

Deletes hierarchical instance and module. It allows the software to delete modules inserted using the `create_module` command.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-cell`	If specified, Verilog module of *`hinstName`* is also deleted.
`-help`	Outputs a brief description that includes type and default information for each `delete_module` parameter.
`hinstName`	Specifies hierarchical instance name to delete.

Related Information

- Supported SAI Commands

insert_boundary_flops

```
insert_boundary_flops
[-help]
-clock string
-module string
[-reset]
-terms string
```

Allows insertion of boundary flops for empty modules/blocks (instantiated in the top level) through SAI. The command options specify the clock net name, the module name and the module pin name.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-clock` *`string`*	Specifies clock net name.
`-help`	Outputs a brief description that includes type and default information for each `insert_boundary_flops` parameter.
`-module` *`string`*	Specifies the module name.
`-reset`	Deletes all flops.
`-terms` *`string`*	Specifies module pin name.

Related Information

- Supported SAI Commands

report_sai_constraint

```
report_sai_constraint
[-help]
[-area_weight float]
[-spacing_weight float]
[-order_weight float]
[-congestion_weight float]
[-rule {all | rule_name | rule_name_list}]
```

You can use the `report_sai_constraint` command to validate a floorplan result and to check whether the specified SAI constraints are met.
This enables front-end designers to easily evaluate a floorplan provided by their physical designers against the specified floorplan rules without requiring them to have much knowledge about using the Innovus System.

The `report_sai_constraint` command reports:

- The general floorplan violations such as macro-to-macro overlap, fence-to-fence overlap, macro overlaps with fence, and/or if macros are placed outside their parent module boundary.
- SAI 2.0 constraint violations
- The Floorplan Quality Index that enables you to estimate whether a floorplan is ready to be used. Floorplan Quality Index is similar to the total negative slack (TNS) concept of timing analysis
- Support area, spacing, net weight, order, and congestion quality indexes; and their index weights.

Besides generating a report, it also creates a violation marker if violations are detected. These violation markers will be displayed in the Innovus artwork window and can be queried from the Innovus violation browser.

Based on the reported floorplan quality index, you can decide whether this floorplan is good to go or not. If the floorplan quality index does not meet your requirements, you can further:

- Adjust the wire length, congestion, and/or overlap cost factors and rerun `proto_design` to produce new floorplan result(s).
- Improve the current floorplan by manual adjustment.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-area_weight` *`float`*	Specifies the weight of the area index. *Default*: 1.0
`-congestion_weight` *`float`*	Specifies the weight of the congestion index. *Default*: 1.0
`-help`	Outputs a brief description that includes type and default information for each `report_sai_constraint` parameter.
`-order_weight` *`float`*	Specifies the weight of the order index. *Default*: 1.0
`-rule {all` | *`rule_name`* | *`rule_name_list`*`}`	
	Checks and reports the violation(s) of the specified rule(s). *Default*: all (All SAI rules of the current design)
`-spacing_weight` *`float`*	Specifies the weight of the area index. *Default*: 1.0

Related Information

- Supported SAI Commands

set_floorplan

```
set_floorplan
[-help]
[-aspect_ratio float]
[-height float]
[-side_spacing float]
[-util float]
[-width float]
```

Defines specifications for the `floorplan` command before netlist import. The aspect ratio and utilization values will be passed to the `floorplan` command.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-aspect_ratio` *float*	Specifies the core dimensions of the chip as the ratio of the height divided by the width. If a value of 1.0 is used, a square chip is defined. A value of 2.0 will define a rectangular chip with height dimensions that are twice the width dimension.
`-height` *float*	Specifies the height of the die.
`-help`	Outputs a brief description that includes type and default information for each `set_floorplan` parameter.
`-side_spacing` *float*	Specifies the distance from all the sides to the core.
`-util` *float*	Specifies the target utilization for the core.
`-width` *float*	Specifies the width of the die.

Related Information

- Supported SAI Commands

set_ref_flop

```
set_ref_flop
[-help]
reference_flop_cell_name
[-clock_input_pin string]
[-data_input_pin string]
[-data_output_pin string]
```

Defines the reference flop cell. The reference flop cell is used for boundary flop (inserted by `connect` command) and dummy flops (inserted by `create_module` command).

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-clock_input_pin` *`string`*	Specifies the cell CLK port name.
`-data_input_pin` *`string`*	Specifies the cell D port name.
`-data_output_pin` *`string`*	Specifies the cell Q port name.
`-help`	Outputs a brief description that includes type and default information for each `set_ref_flop` parameter.
`reference_flop_cell_name`	Specifies the cell name of the reference flop.

Related Information

- Supported SAI Commands

set_ref_gate

```
set_ref_gate
[-help]
reference_unit_gate_cell
```

Defines reference gate cell. The reference gate cell is used for dummy gates (inserted by `create_module` command).

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-help`	Outputs a brief description that includes type and default information for each `set_ref_gate` parameter.
`reference_unit_gate_cell`	Specifies the cell name of the reference gate.

Related Information

- Supported SAI Commands

set_ref_macro

```
set_ref_macro
[-help]
name
[-edge_orientation string]
-height float
[-symmetry string]
[-type string]
-width float
```

Defines a reference macro. It helps specify the macro size, macro type, and symmetry information. This command generates a LEF macro with 'SYMMETRY X Y R90'. This command supports OA based design import/export.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-edge_orientation` *`string`*	Specifies orientation of the macro that will be used when placing it on left, bottom, right, top core boundary respectively.
`-height` *`float`*	Specifies the height size.
`-help`	Outputs a brief description that includes type and default information for each `set_ref_macro` parameter.
`name`	Specifies the macro name.
`-symmetry` *`string`*	Specifies the macro symmetry. *Default*: `"X Y"`
`-type` *`string`*	Specifies the macro type. *Default*: `BLOCK`
`-width` *`float`*	Specifies the width size.

Related Information

- Supported SAI Commands

set_ref_memory

```
set_ref_memory
[-help]
name
-area_per_bit float
[-aspect_ratio float]
[-bit_size float]
[-symmetry string]
```

Allows the software to set reference memory cells in SAI mode. This command creates LEF MACRO. Reference memory is used for macro insertion by `create_module` and `add_macro` command. This command supports OA based design import/export.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-area_per_bit` *float*	Specifies the area per bit.
`-aspect_ratio` *float*	Specifies the aspect ratio of the memory cell. *Default*: 1.0

`-bit_size` *float*	Specifies the bit size. This size is used with `create_module` and `add_macro` command.
	For example,
	`set_ref_memory mem1 -bit_size 1024 -area_per_bit 1.1 -aspect_ratio 0.75`
	`create_module iBlock -cell mBlock -memory_bit_count 4096`
	`add_macro -cell mBlock -memory mem1`
	then, 4096 / 1024 = 4 “mem1” macro are added in mBlock module.

`-help`	Outputs a brief description that includes type and default information for each `set_ref_memory` parameter.
name	Specifies the reference memory cell name. It generates *`name.lef`* on the Innovus working directory.
`-symmetry` *string*	Specifies the memory symmetry.

Related Information

- Supported SAI Commands

set_sai_version

```
set_sai_version
[-help]
version
```

Sets the version of the SoC Architecture Information (SAI) mode.

Note: Use this command in the SAI interactive mode or write it in the SAI file. The `read_sai` command can be used to read the SAI file in batch mode.

Parameters

`-help`	Outputs a brief description that includes type and default information for each `set_sai_version` parameter.
`version`	Specifies the SAI format version.

Related Information

- Supported SAI Commands

Supported UFC Commands

The Unified Floorplan Constraint (UFC) methodology is a powerful Innovus capability that can define several rules on floorplan objects, check specified rules, and report the violations with rule name as well as severity. The violations can be shown in the violation browser. UFC also provides the ability to automatically fix specific violations. It can add routing blockages, snap macro location, adjust core shape to fix corresponding violations. In addition, UFC also provides the ability to exclude rules from checking for specified objects.

The following are the supported UFC Commands:

- `set_area_rule`
- `set_width_rule`
- `set_spacing_rule`
- `set_merge_and_reshape_spacing_rule`
- `set_halo_rule`
- `set_same_length_site_rule`
- `set_track_rule`
- `set_parallel_run_length_rule`
- `set_reshape_object_rule`
- `set_white_area_extension_rule`
- `set_reshape_available_sites_rule`
- `set_dont_use_base_cell_rule`
- `exclude_rule`

Note: You can write the supported UFC commands in a UFC file and then use the `check_ufc` command to check the floorplan based on the rules defined in the file.

exclude_rule

```
exclude_rule
-name "ruleName"
[-obj string]
[-obj1 string]
[-ref string]
[-reason string]
[-same_base_cell]
[-check_orientation {none | mirror | partial | align}]
```

Excludes the specified rules if the object triggers the exclusion condition.

Note:

- The `exclude_rule` command should be specified before the other UFC rules (`set_*_rule`).
- The `exclude_rule` command only supports macro, routing_blockage and available_sites as `-obj`, `-obj1` and `-re`f objects.

Parameters

`-check_orientation {none | mirror | partial | align}`	
	Specifies the spacing check is valid when the `-obj` and `-ref` objects are mirror, partial or align. *Default*: `none` *Data_type*: enum, optional
`-name "`*`ruleName`*`"`	Specifies the rule name.
`-obj` *`string`*	Specifies the object to be excluded for this rule.
`-obj1` *`string`*	Specifies the object to be excluded for this rule.
`-reason` *`string`*	Specifies the reason for exclusion of the object.
`-ref` *`string`*	Specifies the reference object to be excluded for this rule.
`-same_base_cell`	When specified, the rule will be only valid when both `-obj` and `-ref` objects are hard macros and identical.

Example

-
  ```
  exclude_rule -name "GBL_ MACRO_SPACING" -obj [macros -if {.base_cell.name == “XYZ”}] \
       -reason “my spacing is included inside the macro”
  ```

  ```
  set_spacing_rule {.spacing >=10.0} -name "GBL_ MACRO_SPACING" -obj [macros ] -ref [macros] \
       -description {the spacing between macro to macro must be greater than 10.0.}
  ```

Related Information

- Supported UFC Commands

set_area_rule

```
set_area_rule
expression_of_.area
-obj shapeObj
-name "ruleName"
[-description string]
[-severity string]
```

Specifies area constraints for specific object(s).

Parameters

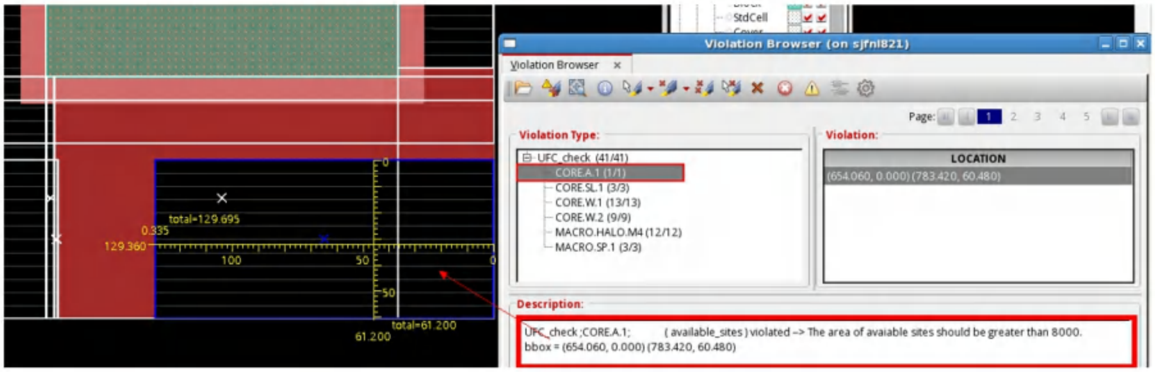
`-description string`	Specifies the description of this rule.
`expression_of_.area`	Specifies the dimension expression of .area to check for the valid area.
`-name "ruleName"`	Specifies the rule name.
`-obj shapeObj`	Specifies the target object to check against this rule.
`-severity string`	Specifies the severity.

Example

- The following command specifies the rule name, the expression of .area to check for the valid area and the severity of the rule.

  ```
  set_area_rule -name "AREA_X1" {.area >= 8000.0}  -obj [available_sites]}] -severity GUIDELINE
  ```

  Violations of area rule:

  ```
  UFC_check ;CORE.A.1;          ( available_sites ) violated --> The area of avaiable sites should be greater than 8000.
  bbox = (654.060, 0.000) (783.420, 60.480)
  ```

Related Information

- Supported UFC Commands

set_dont_use_base_cell_rule

```
set_dont_use_base_cell_rule
-name "ruleName"
-obj string
[-description string]
[-severity string]
```

Specifies the cells which cannot be used in the design.

Parameters

`-description` *`string`*	Specifies the description of this rule.
`-name "`*`ruleName`*`"`	Specifies the name of the rule that is shown on DRC marker.
`-obj` *`string`*	Specifies the list of cell names to be defected.
`-severity` *`string`*	Specifies the severity.

Related Information

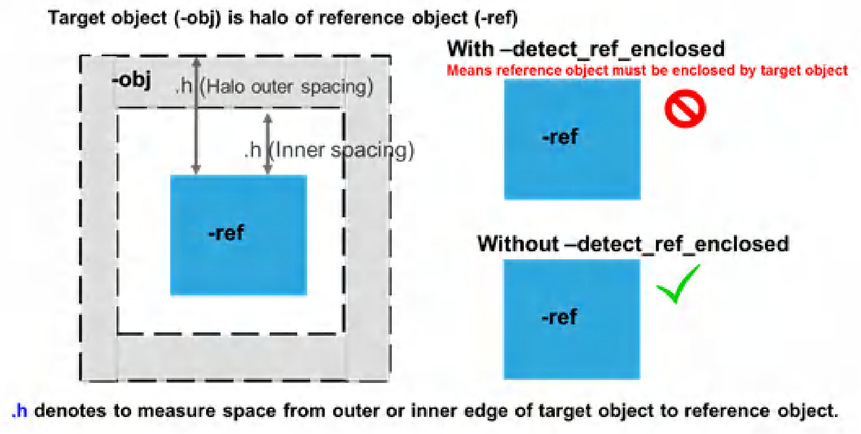
- Supported UFC Commands

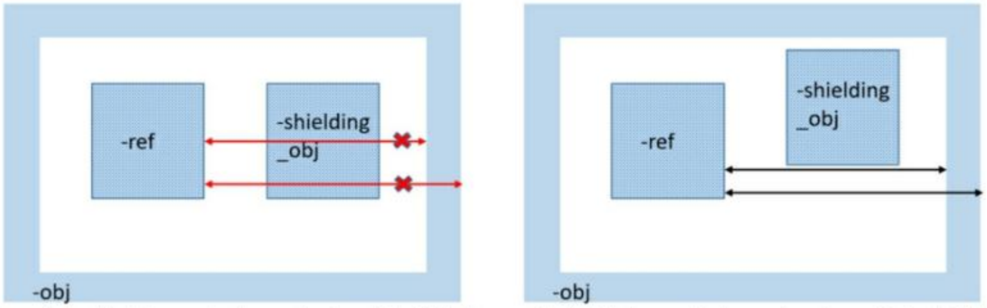
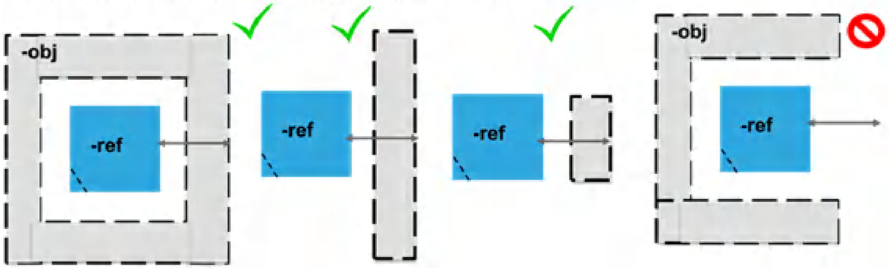
set_halo_rule

```
set_halo_rule
expression_of_.halo
-obj shapeObj1
-ref shapeObj2
-name "ruleName"
[-side {all | top | bottom | left | right | vertical | horizontal}]
[-type {inner | outer}]
[-detect_ref_enclosed {0|1}]
[-description string]
[-severity string]
[-shielding_obj shapeObj3]
```

Specifies halo constraints for specific object(s).

Parameters

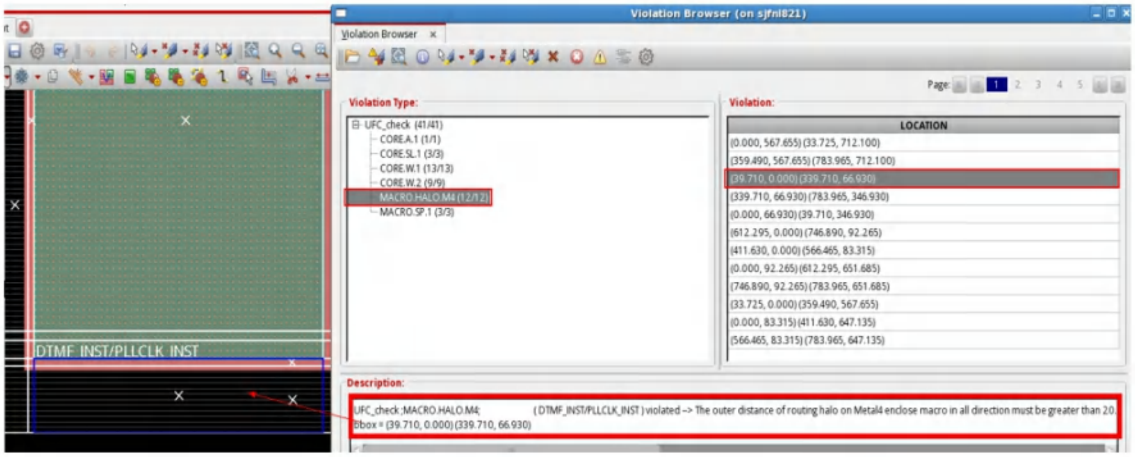
`-description` *`string`*	Specifies the description of this rule.
`-detect_ref_enclosed {0|1}`	Checks if the reference object is completed enclosed by the main object.  Target object (-obj) is halo of reference object (-ref) -obj .h (Halo outer spacing) .h (Inner spacing) -ref With –detect_ref_enclosed Means reference object must be enclosed by target object -ref Without –detect_ref_enclosed -ref .h denotes to measure space from outer or inner edge of target object to reference object. **Note:** With -detect_ref_enclosed 1, only checks existence of specified edge. *Default*: 0
`expression_of_.halo`	Specifies the dimension expression of .halo to check for valid spacing.
`-name "`*`ruleName`*`"`	Specifies the rule name.
`-obj` *`shapeObj1`*	Specifies the target object to check against this rule.
`-ref` *`shapeObj2`*	Specifies the reference object to check against this rule.
`-severity` *`string`*	Specifies the severity.

`-shielding_obj` *`shapeObj3`*	Specifies the shielding object that the rule will waive off if the object is between `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`*. **With -shielding_obj**  -ref -shielding _obj -obj -ref -shielding _obj -obj **Does not check halo inner and outer spacing since fully shielded.** **Checks halo inner and outer spacing since not fully shielded**
`-side {all | top | bottom | left | right | vertical | horizontal}`	
	Specifies the side of the cell to use for checking. The halo rule is checked only on the specified side. **Only right side halo is defined: set_halo_rule –side right**  *Default*: `all`
`-type {inner | outer}`	Specifies the type of halo to be checked. The halo check types can be outer or inner. *Default*: `inner`

Example

-
  ```
  set_halo_rule -name "MACRO.HALO.M4" -obj [route_blockages -if { .layer.name == Metal4}] -ref [macros] -type
  outer -side all -detect_ref_enclosed {.halo >= 20.0} \
  -description {The outer distance of routing halo on Metal4 enclose macro in all direction must be greater
  than 20.0 }
  ```

  Violations of halo rule:

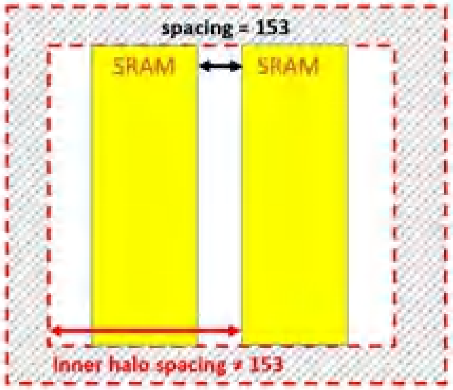
  UFC_check (41/41)
  - CORE.A.1 (1/1)
  - CORE.SL.1 (3/3)
  - CORE.W.1 (13/13)
  - CORE.W.2 (9/9)
  - MACRO.HALO.M4 (12/12)
  - MACRO.SP.1 (3/3)

LOCATION
(0.000, 567.655) (33.725, 712.100)
(359.490, 567.655) (783.965, 712.100)
(39.710, 0.000) (339.710, 66.930)
(339.710, 66.930) (783.965, 346.930)
(0.000, 66.930) (39.710, 346.930)
(612.295, 0.000) (746.890, 92.265)
(411.630, 0.000) (566.465, 83.315)
(0.000, 92.265) (612.295, 651.685)
(746.890, 92.265) (783.965, 651.685)
(33.725, 0.000) (359.490, 567.655)
(0.000, 83.315) (411.630, 647.135)
(566.465, 83.315) (783.965, 647.135)

  ```
  UFC_check ;MACRO.HALO.M4;            ( DTMF_INST/PLLCLK_INST ) violated --> The outer distance of routing halo on Metal4 enclose macro in all direction must be greater than 20.
  Bbox = (39.710, 0.000) (339.710, 66.930)
  ```

-
  ```
  set_halo_rule -name "DMTB.X3" -obj [available_sites] -ref [base_cells SRAM] -side left -type inner -
  detect_ref_enclosed \
        -shielding_obj [base_cells SRAM] {.halo in {0.153}} \
        -description {set_halo_constraint -label DMTB.X3 -class1 useful_sites -lib_cell2 SRAM -halo_side left -
  edge_type inner -must_enclosure true -shielding_lib SRAM -white_list 0.153}
  ```

  Shielding halo rules:

Related Information

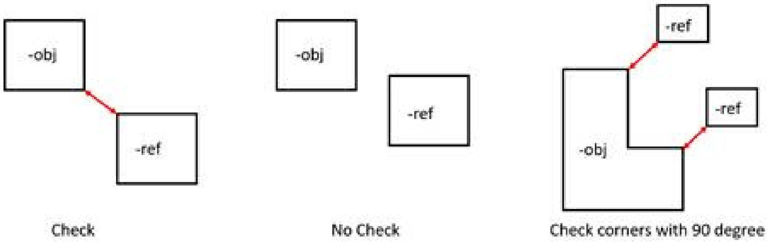
- Supported UFC Commands

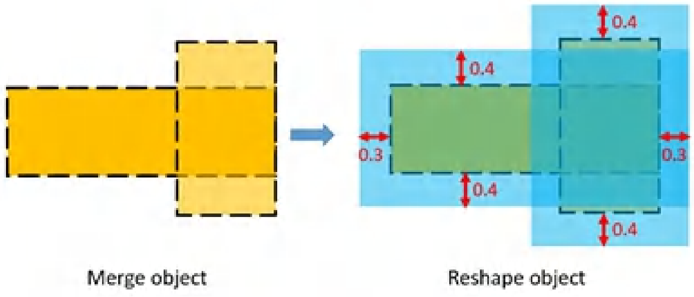
set_merge_and_reshape_spacing_rule

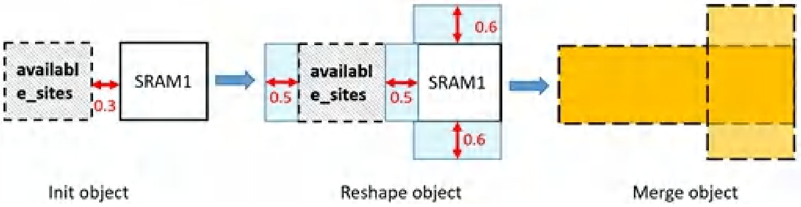
```
set_merge_and_reshape_spacing_rule
expression of .spacing
-name "ruleName"
-obj {[obj1], [obj2]}
-ref {[ref1], [ref2]}
-parallel_run_length float
-reshape_obj_element [index direction value]
-reshape_ref_element [index direction value]
-reshape_obj [direction value]
-reshape_ref [direction value]
-direction {all|horizontal|vertical|corner_distance|top|bottom|left|right}
```

Specifies spacing between merged and reshaped objects. This rule is used to check the spacing between merged objects.

Parameters

`-direction {all|horizontal|vertical|corner_distance|top|bottom|left|right}`	
	Specifies the direction of the edge or the target object to perform the spacing check. For rectilinear objects, all edges of the specified direction are checked. **Note:** The corner-to-corner spacing check with the -direction corner_distance option is only valid when the real parallel run length between target and reference object is less than zero. When parallel run length is larger than zero, there is no need to check the corner distance. **Note:** The rule only checks the corner space between 90 degrees.  -obj -ref Check -obj -ref No Check -ref -ref -obj Check corners with 90 degree
expression_of_.halo	Specifies the expression of .spacing to check for valid spacing.
`-name "ruleName"`	Specifies the rule name.

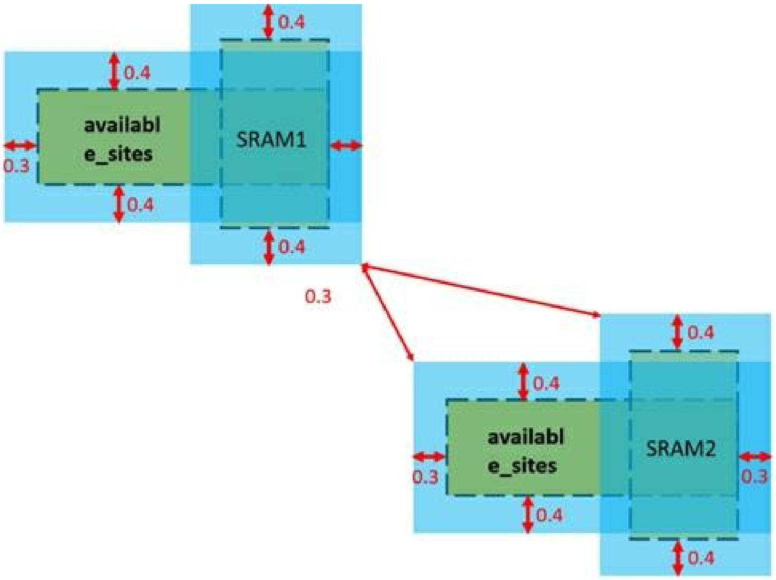
`-obj {[`*`obj1`*`], [`*`obj2`*`]}`	Specifies the target object to check against this rule. **Note:** The -obj and -ref parameters are used to specify which objects should be merged as one target object and one reference object respectively. **Note:** Before merging objects, objects should be reshaped using the `-reshape_obj_element` and `-reshape_ref_element` parameters to determine whether the specified objects can be merged. Only objects which overlap with each other after reshaping can be merged. The merged objects can still be reshaped by the `-reshape_obj` and -reshape_ref parameters. The reshaped objects will be used to do spacing check.
`-parallel_run_length` *`float`*	Specifies the minimal parallel run length required to enable spacing check. It is invalid when checked direction is `corner_distance`.
`-ref {[`*`ref1`*`], [`*`ref2`*`]}`	Specifies the reference object to check against this rule. **Note:** This parameter is used to specify which objects should be merged as one reference object.
`-reshape_obj [`*`direction value`*`]`	Specifies to reshape objects specified with the `-obj` parameter after they are merged. The `-reshape_obj` parameter works on the `-obj` parameter. Its format is: `{{`*`direction value`*`} {`*`direction value`*`} …}.` **Note:** The reshaped and merged target or reference objects can still be reshaped before spacing check. The `-reshape_obj` and `-reshape_ref` parameters are used to reshape target and reference objects. • *direction*: Specifies the direction of reshape. The valid value is vertical, horizontal, left, right, top and bottom. • *value*: Specifies the reshape value. A positive value means extend and a negative value means shrink. For example, “-obj {[available_sites], [base_cells -if {.name == SRAM1}]} -reshape_obj_element {{0 horizontal 0.5} {1 vertical 0.6}} -reshape_obj {{horizontal 0.3} {vertical 0.4}}” means target object should extend 0.3 in horizontal and extend 0.4 in vertical. The following diagram shows a reshaped merged object used as a target object to do spacing check:  0.4 0.4 0.3 0.3 0.4 0.4 Merge object Reshape object
`-reshape_obj_element [`*`index direction value`*`]`	

	Specifies to reshape objects specified with the `-obj` parameter and then merge them into one object. - *index*: Specifies the objects element. The valid value is from 0 to 1. For example, for index “-obj {[available_sites], [base_cells -if {.name == SRAM1}]}”, 0 stands for available_sites and 1 stands for block SRAM1. - *direction*: Specifies the direction of reshape. The valid value is vertical, horizontal, left, right, top and bottom. - *value*: Specifies the reshape value. A positive value means extend and a negative value means shrink. For example, “-obj {[available_sites], [base_cells -if {.name == SRAM1}]} -reshape_obj_element {{0 horizontal 0.5} {1 vertical 0.6}}” means available_sites extends 0.5 in horizontal and block SRAM1 extend 0.6 in vertical. Only reshaped object elements overlap with each other can be merged. *Reshape and merge object elements* The following diagram shows the process of reshape object elements and then merge them as one target object. For this case, if the initial spacing between available_sites and SRAM1 is larger than 0.5, then they will not be merged after extension 0.5 in horizontal since the reshaped objects do not abut or overlap with each other.  availabl e_sites 0.3 SRAM1 0.5 availabl e_sites 0.5 0.6 SRAM1 0.6 Init object Reshape object Merge object
`-reshape_ref [`*`direction value`*`]`	Specifies to reshape objects specified with the `-ref` parameter after they are merged. The `-reshape_ref` parameter works on the -ref parameter. Its format is: `{{`*`direction value`*`} {`*`direction value`*`} …}.`
`-reshape_ref_element [`*`index direction value`*`]`	
	Specifies to reshape objects specified with the `-ref` parameter and then merge them into one object.
`-severity` *`string`*	Specifies the severity.

Example

-
  ```
  set_merge_and_reshape_spacing_rule -name “DTMB.S1” -obj {[available_sites] , [base_cells -if {.name ==
  SRAM1}]} -ref {{[available_sites] , [base_cells -if {.name == SRAM2}]} -reshape_obj_element {{0 horizontal
  0.5} {1 vertical 0.6}} -reshape_ref_element {{0 horizontal 0.5} {1 vertical 0.6}} -reshape_obj {{horizontal
  0.3} {vertical 0.4}} -reshape_ref {{horizontal 0.3} {vertical 0.4}} -direction corner_distance {.spacing >=
  5} \
  -description {set_composite_space_constraint -lable “DTMB.S1” -object1 {{class useful_site} {lib_cell SRAM1}}
  -object2 {{class useful_site} {lib_cell SRAM2}} -extension_obj_element {{0 horizontal 0.5} {1 vertical 0.6}}
  -extension_ref_element {{0 horizontal 0.5} {1 vertical 0.6}} -extension_obj {{horizontal 0.3} {vertical 0.4}}
  -extension_ref {{horizontal 0.3} {vertical 0.4}} -direction corner -min 5}
  ```

  Example of rule DTMB.S1



Related Information

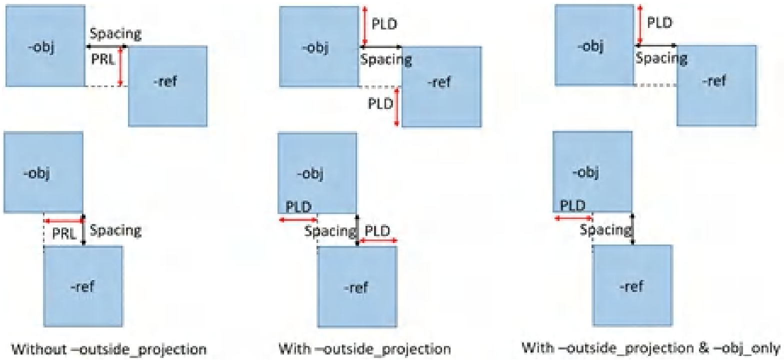
- Supported UFC Commands

set_parallel_run_length_rule

```
set_parallel_run_length_rule
expression_of_.prl
-name "ruleName"
[-obj string]
[-ref string]
[-direction {all | horizontal | vertical}]
[-max_triggering_spacing float]
[-outside_projection [-triggering_prl float] [-obj_only]]
[-description string]
[-severity string]
```

Checks the projected edge length of macro or available_sites.

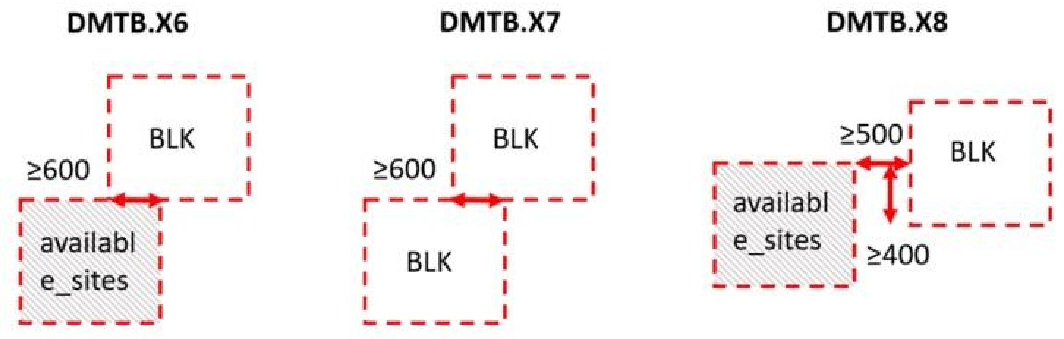
Parameters

`-description` *`string`*	Specifies the description of this rule.
`-direction {all | horizontal | vertical}`	
	Specifies the check direction. *Default*: `all`
`-max_triggering_spacing` *`float`*	Specifies that the rule will only be checked when the spacing between two objects is smaller than or equal to the specified value. *Default*: Infinite, that is all the parallel run lengths are checked in the design.
`-name "`*`ruleName`*`"`	Specifies the rule name.
`-obj` *`string`*	Specifies the target object to check against this rule.
`-obj_only`	When specified (true), only the side of object specified by the `-obj` parameter is checked with the projection rule. If `-obj_only` is not specified (false), both sides of objects specified by the `-obj` and `-ref` parameters are checked for projection rule. *Default:* false **Note:** It must be used with the `-outside_projection` parameter.
`-outside_projection`	When specified, it checks the outside projection between two objects instead of checking parallel run length. • If `-outside_projection` is `false`, the parallel run length is checked. • If `-outside_projection` is `true`, projection length is checked.  -obj Spacing PRL -ref PLD Without –outside_projection With –outside_projection With –outside_projection & –obj_only
`-ref` *`string`*	Specifies the reference object to check against this rule.
`-severity` *`string`*	Specifies the severity.
`-triggering_prl` *`float`*	Specifies that the projection rule will only be checked when the prl between two objects is larger than or equal to this value. *Default:* 0 **Note:** It must be used with the `-outside_projection` parameter.

Example

- `set_parallel_run_length_rule -name "DMTB.X6" -obj [base_cells BLK] -ref [available_sites] -direction vertical -max_triggering_spacing 0 {.prl >= 0.6} \`
  `[-description {set_parallel_run_length_contraint -label "DMTB.X6" -lib_cell1 BLK -class2 useful_site -direction vertical -space 0 -min 0.6}]`
- `set_parallel_run_length_rule -name "DMTB.X7" -obj [base_cells BLK] -ref [base_cells BLK] -direction vertical -max_triggering_spacing 0 {.prl >= 0.6} \`
  `[-description {set_parallel_run_length_contraint -label "DMTB.X7" -lib_cell1 BLK -lib_cell2 BLK -direction vertical -space 0 -min 0.6}]`
- `set_parallel_run_length_rule -name "DMTB.X8" -obj [base_cells BLK] -ref [available_sites] -direction horizontal -max_triggering_spacing 0.5 {.prl >= 0.4} \`
  `[-description {set_parallel_run_length_contraint -label "DMTB.X8" -lib_cell1 BLK -class2 useful_site -direction horizontal -space 0.5 -min 0.4}]`

Example of parallel run length rules:

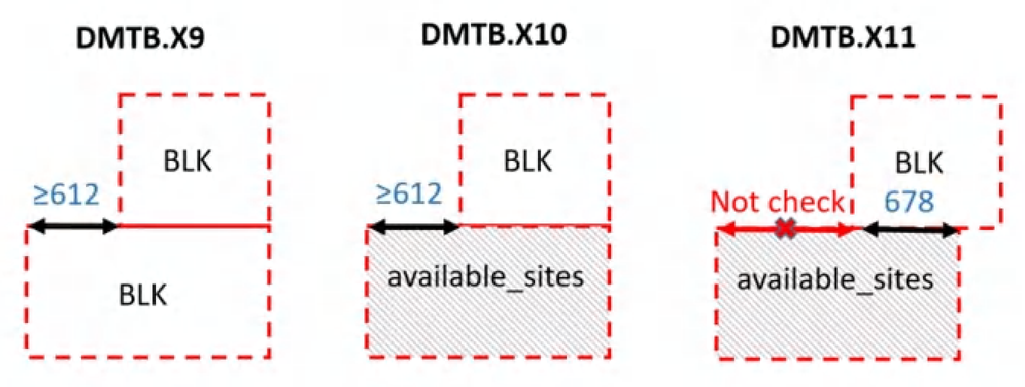


-
  ```
  set_parallel_run_length_rule -name "DMTB.X9" -obj [base_cells BLK] -ref [base_cells BLK] \
     -direction vertical -max_triggering_spacing 0 -outside_projection {.prl ≥ 0.612} \
     [-description {set_projection_length_difference_constraint -label "DMTB.X9" -lib_cell1 BLK -lib_cell2 BLK -direction vertical -space 0 -min 0.612}]
  ```

-
  ```
  set_parallel_run_length_rule -name "DMTB.X10" -obj [base_cells BLK] -ref [available_sites] \
     -direction vertical -max_triggering_spacing 0 -outside_projection {.prl ≥ 0.612} \
     [-description {set_projection_length_difference_constraint -label "DMTB.X10" -lib_cell1 BLK -class2 useful_site -direction vertical -space 0 -min 0.612}]
  ```

-
  ```
  set_parallel_run_length_rule -name "DMTB.X11" -obj [base_cells BLK] -ref [available_sites] \
     -direction vertical -max_triggering_spacing 0 -outside_projection -triggering_prl 0.8 {.prl ≥ 0.612} \
    [-description {set_projection_length_difference_constraint -label "DMTB.X10" -lib_cell1 BLK -class2 useful_site -direction vertical -space 0 -prl_value 0.8 -min 0.612}]
  ```

Example of outside_projection length difference rules:



Note: This rule supports using extended objects. It enables the `set_reshape_object_rule` command to extend objects. The extend objects honor orientation.

For example:

```
set_reshape_object_rule -name "DTMB.X11" -obj obj -direction horizontal -size 1
set_parallel_run_length_rule -obj [base_cells BLK] …
```

Related Information

- Supported UFC Commands

set_reshape_available_sites_rule

```
set_reshape_available_sites_rule
-obj {available_sites | available_sites_no_merge}
-skip_cells cell_name
```

Recalculates the available_sites or available_sites_no_merge by skipping the specified cells. It takes the row sites under the cells into account.

Note:

- The `set_reshape_available_sites_rule` command should be specified before the other `set_*_rule` rules.
- The `set_reshape_available_sites_rule` command only supports blocks as skip_cells.

Parameters

`-obj {available_sites | available_sites_no_merge}`	
	Specifies the object which will be reshaped. It only can be available_sites or available_sites_no_merge. *Data_type*: enum, required
`-skip_cells` *`cell_name`*	Specifies the cells that are skipped while calculating the available_sites or available_sites_no_merge. It take the row sites under the cells into account. This parameter supports wildcards (`*`). *Data_type*: list, required

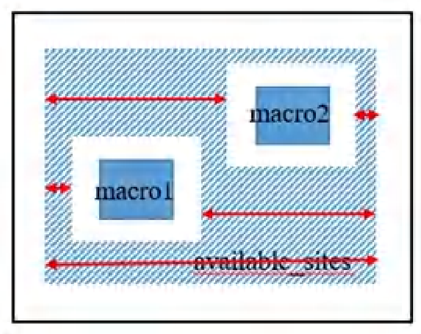
Example

- The following command specifies the available_sites' width check if macro1 is ignored.

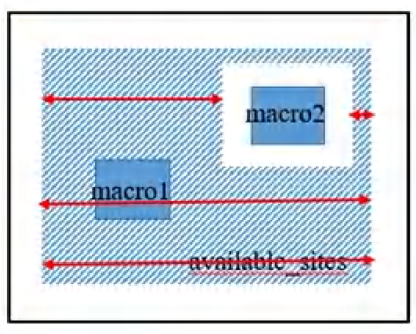
  ```
  set_reshape_available_sites_rule -obj available_sites -skip_cells macro1
  ```

  In the following, the Diagram-A shows that the available_sites will be cut into many parts if macro1 and macro2 are taken into account. The Diagram-B shows the available_sites only be cut by macro2 since macro1 is skipped by `set_reshape_available_sites_rule`.

A



B



Related Information

- Supported UFC Commands

set_reshape_object_rule

```
set_reshape_object_rule
-name ruleName
-obj { all | obj |ref }
-direction { horizontal | vertical | top | bottom | left | right}
-size float
```

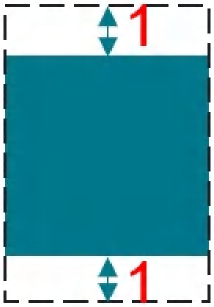
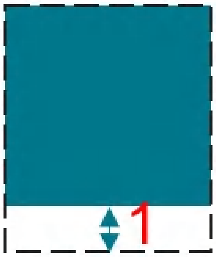
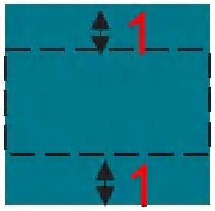
Reshapes the specified objects in specified rules.

Note: This rule can only work on `set_spacing_rule` and `set_parallel_run_length_rule`.

Parameters

`-name "ruleName"`	Specifies the rules whose –obj or –ref objects will be reshaped.
`-obj { all | obj |ref }`	Specifies the objects in the specified rules to reshape. - `all`: Objects specified by both the `-obj` and -ref options will be reshaped. - `obj`: Only objects specified by the `-obj` option in the specified rules will be reshaped. - `ref`: Only objects specified by the `-ref` option in the specified rules will be reshaped.
`-direction { horizontal | vertical | top | bottom | left | right}`	

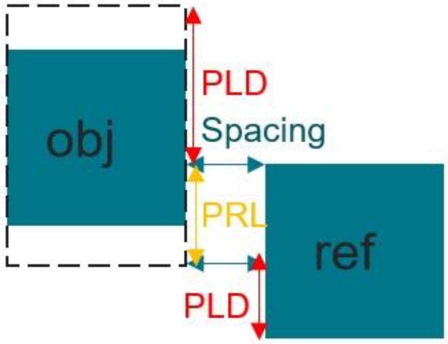
	Specifies the reshape direction. - `horizontal`: Objects are reshaped in the horizontal direction. - `vertical:` Objects are reshaped in the vertical direction. - `top`: Objects are reshaped in the top direction. - `bottom`: Objects are reshaped in the bottom direction. - `left`: Objects are reshaped in the left direction. - `right`: Objects are reshaped in the right direction.

`-size` *`float`*	Specifies the reshape value. Objects are extended when a positive value is specified and shrunk when a negative value is specified. Example: • `-direction vertical -size 1`  • `-direction bottom -size`  `1` • `-direction vertical -size -1`  •

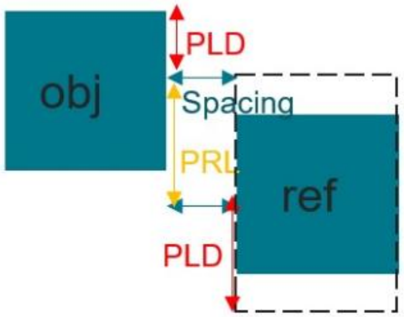
Example

- The following example shows objects are reshaped based on PRL/PLD rule.

```
set_reshape_object_rule -name "rule_name" -obj obj -direction vertical -size 1
```



```
set_reshape_object_rule -name "rule_name" -obj ref -direction vertical -size 1
```



Related Information

- Supported UFC Commands

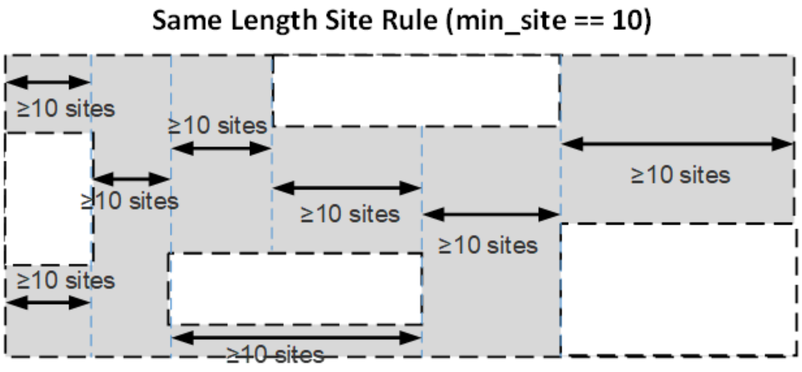
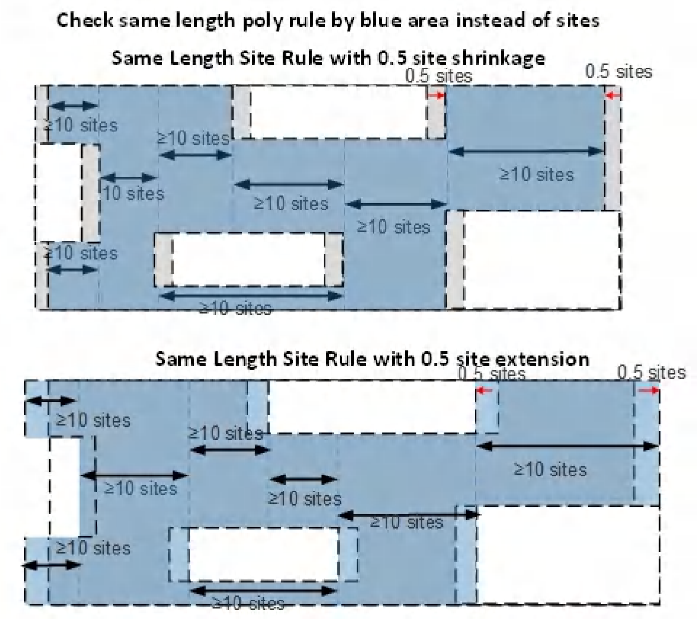
set_same_length_site_rule

```
set_same_length_site_rule
[-obj shapeObj]
[-name "ruleName"]
[-min_site]
[-site_extension]
[-description string]
[-severity string]
```

Sets the same length site rule in a group of continuous poly stripes with the same length.

Parameters

`-description` *`string`*	Specifies the description of this rule.

`-min_site`	Specifies the minimum number of poly for a group of continuous poly with the same length. *Default*:10  **Note:** The same length site rule is already supported with the minimal number. However, UFC methodology also provides the capability to shrink/extend 0.5 site on available_sites. The following example shows the same length site rule with 0.5 site shrinkage/extension.  Check same length poly rule by blue area instead of sites Same Length Site Rule with 0.5 site shrinkage 0.5 sites ≥10 sites Same Length Site Rule with 0.5 site extension 0.5 sites ≥10 sites
`-name "ruleName"`	Specifies the rule name.
`-obj shapeObj`	Specifies the target object to check against this rule. *Default*: available_sites
`-severity string`	Specifies the severity.

`-site_extension`	Specifies the extension site number to be extend or shrink. If the specified value is negative, it will shrink the same length site.

Example

-
  ```
  set_same_length_site_rule -name "SL.CORE.1" -min_site 20 -site_extension 0.5 \
  -description {The same length site must be greater than 20 with 0.5 site extension.}
  ```

Related Information

- Supported UFC Commands

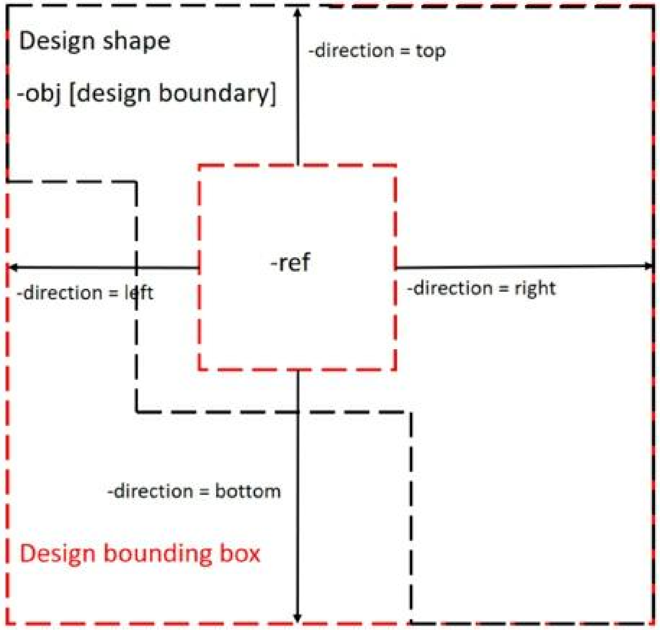
set_spacing_rule

```
set_spacing_rule
expression_of_.spacing
-obj shapeObj1
{-ref shapeObj2 [-check_edge {opposite | same | both }]}
-name "ruleName"
[-parallel_run_length float | -enclosure [-detect_ref_enclosed]]
[-no_overlap]
[-direction {all | vertical | horizontal | any | top | bottom | right | left | corner_distance | manhattan}]
[-description string]
[-severity string]
[-shielding_obj shapeObj3]
[-same_base_cell]
[-check_orientation {none | mirror | partial | align}]
[-bounding_box {none | both | obj | ref} [-ignore_orientation {none |both | obj | ref}]]
[-same_object]
```

Specifies the spacing or enclosure constraints between specific objects.

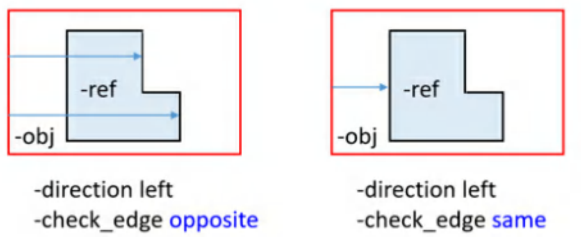
Parameters

```
-bounding_box {none | both | obj | ref}
```

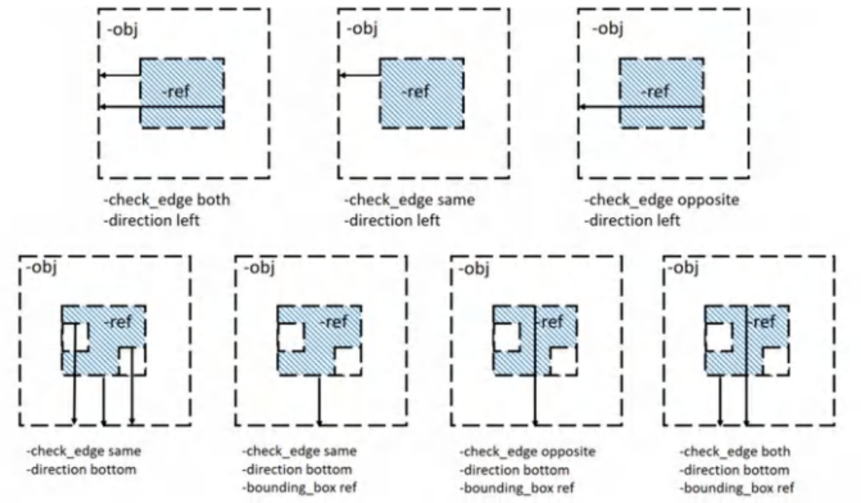
	When specified, uses extended bounding box instead of the actual shape to check the spacing. • `none`: The actual shapes of the target and reference objects are used to check the spacing. • `both`: Uses the bounding box of both the target object and the reference object to check the spacing. • `obj`: Uses the bounding box of the target object and the actual shape of the reference object to check the spacing. • `ref`: Uses the the actual shape of the target object and the bounding box of the reference object to check the spacing. This ref option value is normally used for checking spacing between design bounding box and macro. The following example shows the use of the design bounding box as target object box with option `-bounding_box obj`: `-obj [design_boundary] -bounding_box obj`  Design shape -obj [design boundary] -direction = top -ref -direction = left -direction = right -direction = bottom Design bounding box *Default*: `none` *Data_type*: enum, optional
`-check_edge {opposite | same | both }`	

Specifies that the spacing check is between edges of the same, opposite or both sides of the target and reference objects.

For example, the following diagram shows the spacing check between the same and opposite sides.



The `-check_edge` parameter is used to specify if all of `-ref object` edges should be checked from. The following diagram shows how the `-bounding_box`, `-check_edge` and `-direction` parameters determine the checked spacing.



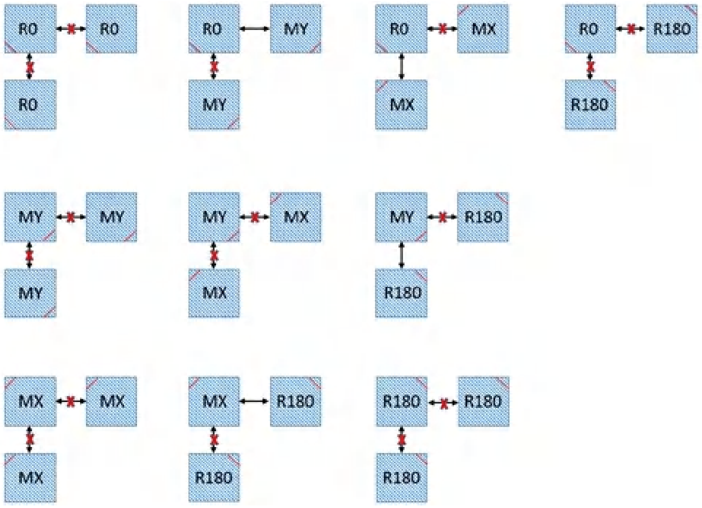
Default: `opposite`

Data_type: enum, optional

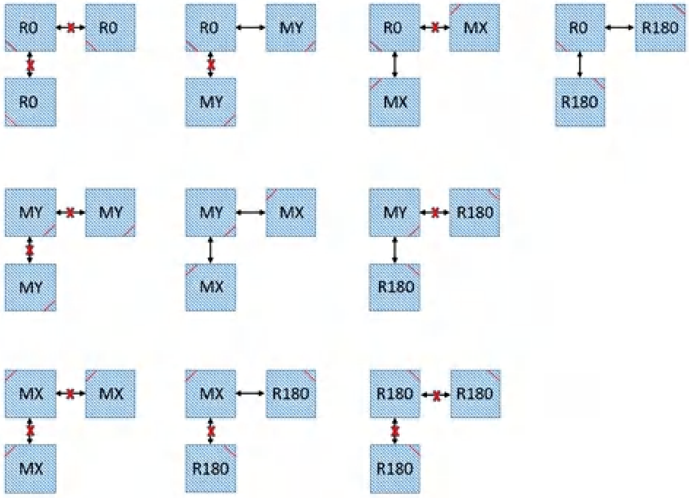
`-check_orientation {none | mirror | partial | align}`

Specifies that the spacing check is valid when `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* have mirror, partial or align orientations. It checks the spacing between specified orientations.

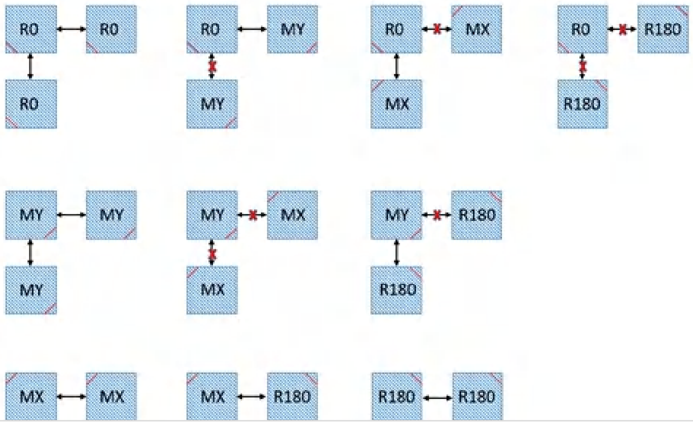
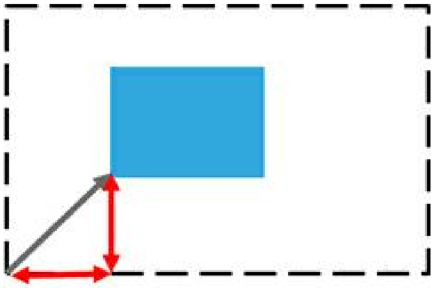
- `none`: Checks the spacing between `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* without taking orientation into account.
- `mirror`: Checks the spacing between `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* with mirror symmetry.
  Example: `-check_orientation mirror`

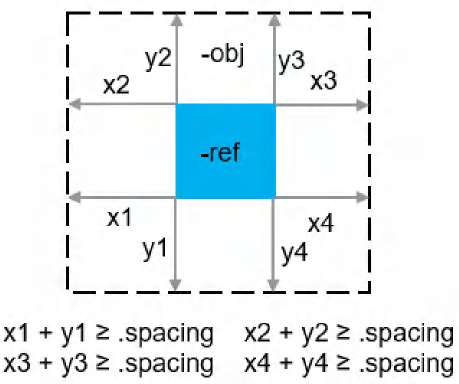


- `partial`: Checks the spacing between the same edges of `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* without taking edges' direction into account. It is similar to `mirror` but the checked edges can have opposite direction.
  Example: `-check_orientation partial`

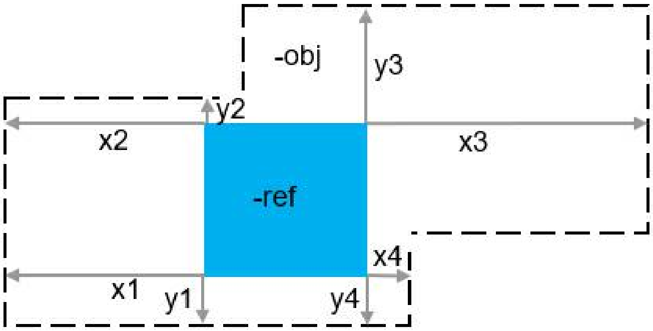


- `align`: Checks the spacing between `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* if they align in the check direction.
  Example: `-check_orientation align`

<table>
<tr><td></td><td> </td></tr>
<tr><td><code>-detect_ref_enclosed</code></td><td>Checks if the reference object is completely enclosed by the target object.
<b>Note:</b> With the <code>-enclosure</code> parameter, the spacing check is only performed when the target object encloses the reference object.
<b>Note:</b> With <code>-detect_ref_enclosed</code>, the target object must enclose the reference object. Otherwise, it is a violation. Without <code>-detect_ref_enclosed</code>, no checking is done and no violation is reported.
<i>Default</i>: 0</td></tr>
<tr><td><code>-description <i>string</i></code></td><td>Specifies the description of this rule.</td></tr>
<tr><td colspan="2"><code>-direction {all | vertical | horizontal | any | top | bottom | right | left | corner_distance | manhattan}</code></td></tr>
<tr><td></td><td>Specifies the direction of the edges or the target object to perform the spacing check. For rectilinear objects, all edges of the specified direction will be checked.
• <code>corner_distance</code>: Corner distance is be modeled as spacing (<code>corner_distance</code>)

$\sqrt{{d_x}^2 + {d_y}^2} = .s$
• <code>manhattan</code>: Manhattan spacing means the <code>-ref <i>shapeObj2</i></code> must be enclosed by the <code>-obj <i>shapeObj1</i></code> and the sum of the distance between <code>-ref <i>shapeObj2</i> corner</code> and <code>-obj <i>shapeObj1</i> boundary</code> in X and Y direction must be equal or greater than the specified value.
In the following example, the Manhattan spacing is only for enclosure check:</td></tr>
</table>

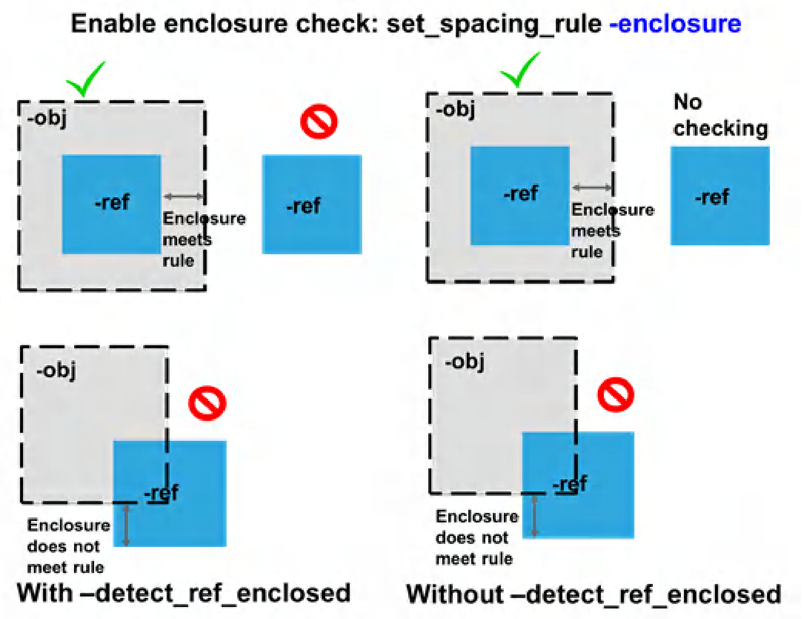


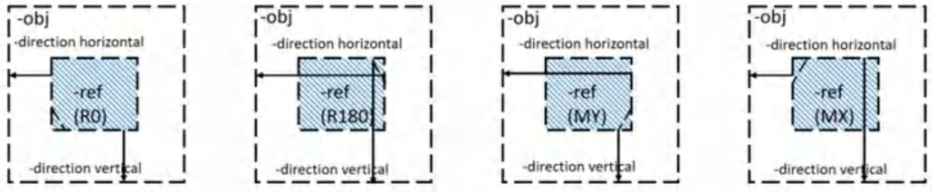
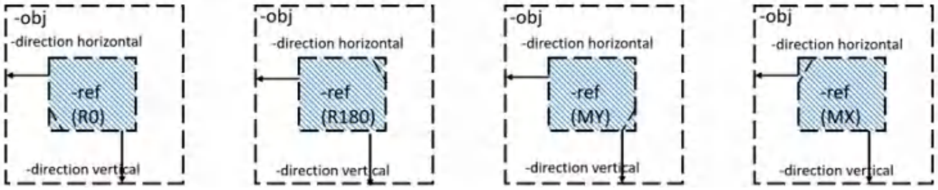
In the following example, manhattan spacing is checked if `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* are rectilinear.

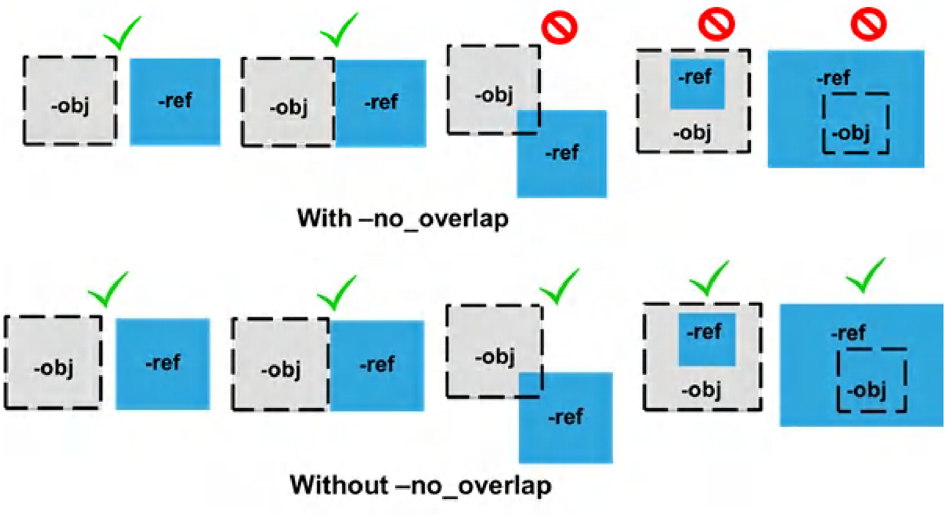
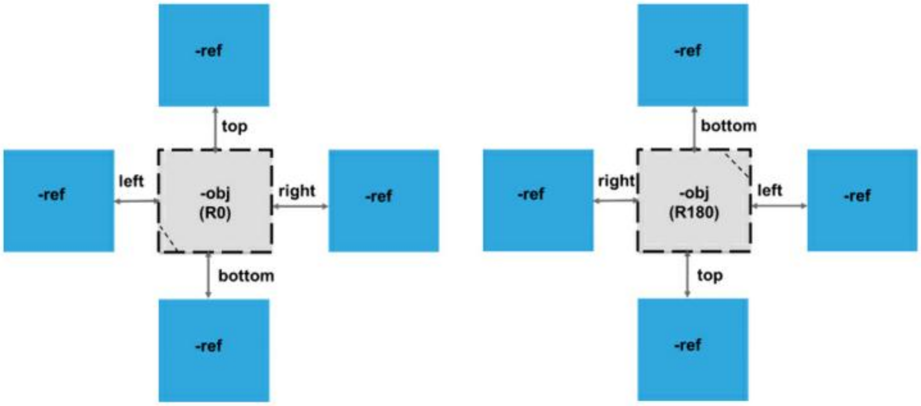


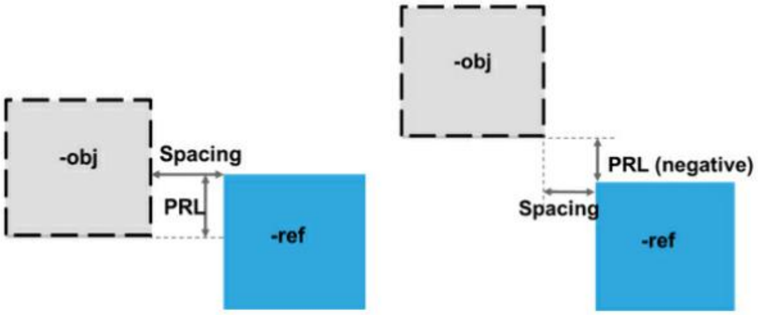
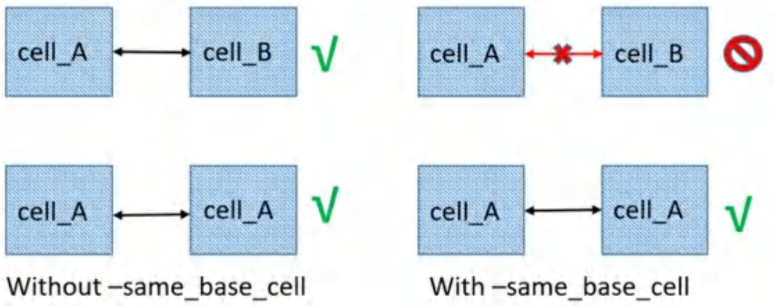
Default: `all`

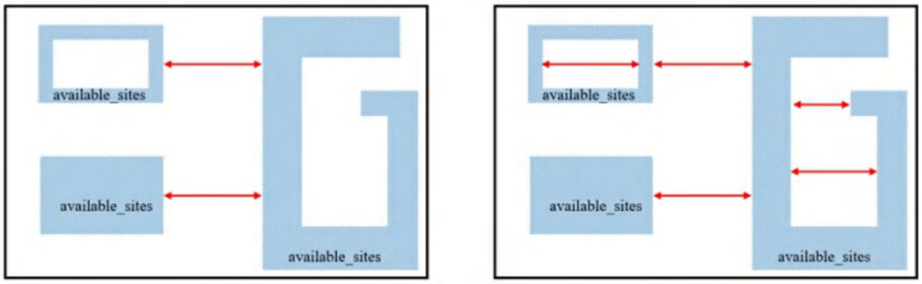
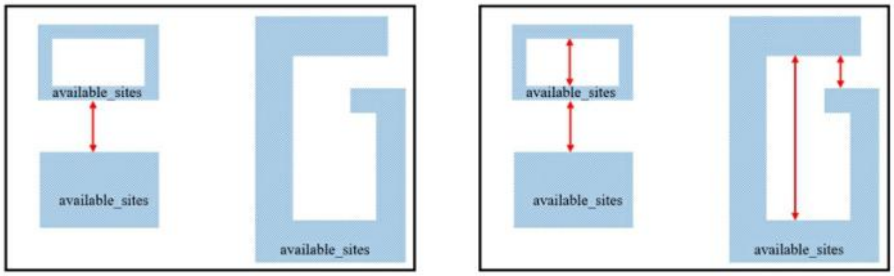
Data_type: enum, optional

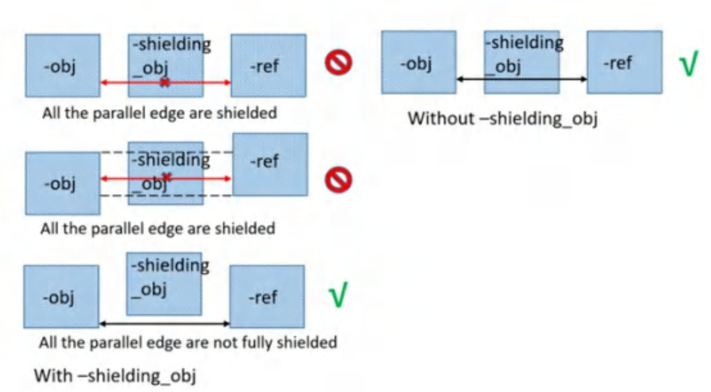
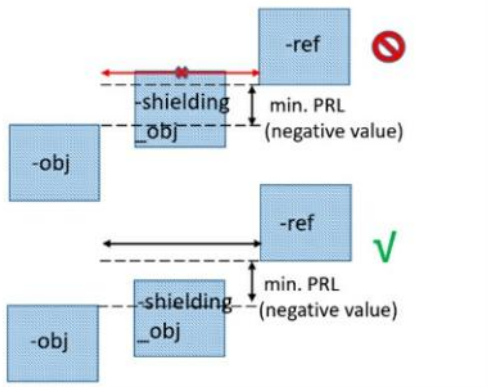
`-enclosure`	When specified, spacing check is only performed when the target object encloses the reference object. **Note:** With `-detect_ref_enclosed`, the target object must enclose the reference object. Otherwise, it is a violation. Without `-detect_ref_enclosed`, no checking is done and no violation is reported.  Enable enclosure check: set_spacing_rule -enclosure ✓ -obj -ref Enclosure meets rule 🚫 -ref ✓ -obj -ref Enclosure meets rule No checking -ref -obj 🚫 -ref Enclosure does not meet rule With –detect_ref_enclosed -obj 🚫 -ref Enclosure does not meet rule Without –detect_ref_enclosed *Default*: 0
`expression_of_.spacing`	Specifies the expression of .spacing to check for valid spacing.
`-ignore_orientation {none |both | obj | ref}`	

	When specified, the checked edge of the specified object should not rotate even if the specified object rotates. For example, `-ignore_orientation ref` means that the checked edge of the reference object should not rotate even if the reference object rotates. **Note:** This parameter can only be used with `-bounding_box` parameter. When `-ref` *`object`* is a macro, `-check_edge` parameter is not `both`, and `-ignore_orientation` is `none`, then the rotation of the macro is considered while checking the spacing rules. The following diagram shows that the checked edge of the reference object should also rotate if the reference object rotates.  -obj -direction horizontal -ref (R0) -direction vertical -obj -direction horizontal -ref (R180) -direction vertical -obj -direction horizontal -ref (MY) -direction vertical -obj -direction horizontal -ref (MX) -direction vertical If `-ignore_orientation` is `ref`, spacing check does not consider the macros' rotation. The following diagram shows that the checked edge of the reference object does not rotate if reference object rotates.  -obj -direction horizontal -ref (R0) -direction vertical -obj -direction horizontal -ref (R180) -direction vertical -obj -direction horizontal -ref (MY) -direction vertical -obj -direction horizontal -ref (MX) -direction vertical *Default*: `none` *Data_type*: enum, optional
`-name "`*`ruleName`*`"`	Specifies the rule name.

`-no_overlap`	When enabled, overlapping between the target and reference object is flagged.  *Default*: 0
`-obj` *`shapeObj1`*	Specifies the target object to check against this rule. **Note:** This rule is not checked on objects (`-obj` *`object`*) that have R90 or R270 orientation.  -ref top left -obj (R0) right bottom -obj (R180)

`-parallel_run_length` *`float`*	Specifies the minimal parallel run length to enable spacing check. This spacing check is valid only for horizontal, vertical, orthogonal spacing.  -obj Spacing PRL -ref -obj PRL (negative) Spacing -ref
`-ref` *`shapeObj2`*	Specifies the reference object to check against this rule.
`-same_base_cell`	When specified, the rule will be only valid when both `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* are hard macros and identical. It checks the spacing between same cells only.  cell_A cell_B √ cell_A cell_B cell_A cell_A √ cell_A cell_A √ Without –same_base_cell With –same_base_cell

`-same_object`	When specified, the spacing between between `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* is checked along with the spacing for objects themselves. **Note:** This rule check is valid only when both `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`* are available sites. For example, the following diagram shows the spacing check in the *horizontal* direction with and without option `-same_object` when both `-obj` and `-ref` are specified as available sites:  available_sites available_sites available_sites available_sites available_sites available_sites without option –same_object with option –same_object The following diagram shows the spacing check in the *vertical* direction with and without option `-same_object` when both `-obj` and `-ref` are specified as available sites:  available_sites available_sites available_sites available_sites available_sites available_sites without option –same_object with option –same_object
`-severity` *`string`*	Specifies the severity.

`-shielding_obj` *`shapeObj3`*	Specifies the shielding object that the rule will waive off if the object is between `-obj` *`shapeObj1`* and `-ref` *`shapeObj2`*. **Note:** With option `-parallel_run_length` and a negative value, the spacing between `-obj` and `-ref` is not checked if the parallel edges are fully shielded. Spacing is checked if parallel edges are not fully shielded. Example: With `-shielding_obj`  -obj -shielding _obj -ref All the parallel edge are shielded -obj -shielding _obj -ref Without –shielding_obj -obj -shielding _obj -ref All the parallel edge are shielded -obj -shielding _obj -ref √ All the parallel edge are not fully shielded With –shielding_obj Example: With `-shielding_obj` and `-parallel_run_length`  -ref -shielding _obj min. PRL (negative value) -obj -ref √ min. PRL (negative value) -shielding _obj -obj

Example

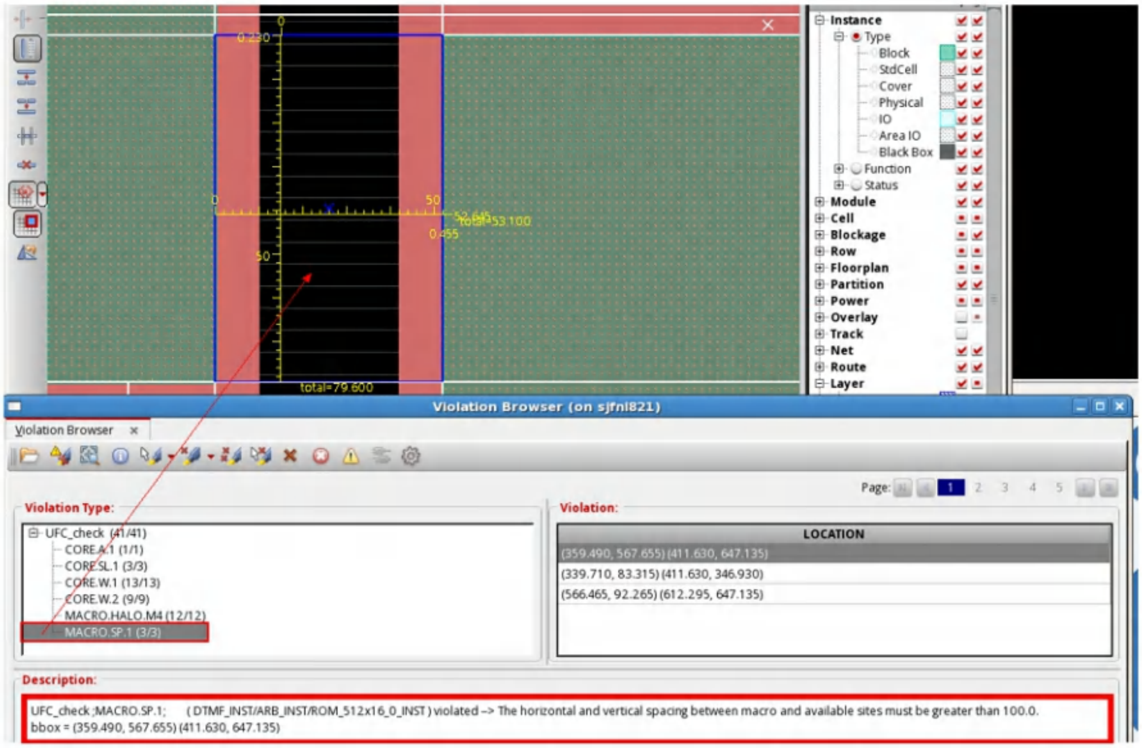
-
  ```
  set_spacing_rule -enclosure -name "EN.CORE.1" -obj [design_boundary] -ref [available_sites] -direction all
  {.spacing >= 80.0} \
  -description {The available sites to design boundary spacing in horizontal and vertical direction must be
  greater than 80.0.}
  ```

-
  ```
  set_spacing_rule -enclosure -name "EN.CORE.1" -obj [design_boundary] -ref [macros] -direction vertical -
  detect_ref_enclosed {.spacing >=150.0} \
  ```

```
-description {The macro to design boundary spacing in vertical direction must be greater than 150.0.}
```

-
  ```
  set_spacing_rule -name "MACRO.SP.1" -obj [macros] -ref [available_sites] -direction all -parallel_run_length
  20 {.spacing >= 100.0} -severity ERROR-02\
  -description {The horizontal and vertical spacing between macro and available sites must be greater than
  100.0 when the PRL is greater than 20.}
  ```

Violations of spacing rule:



Violation Type:

```
UFC_check (41/41)
  CORE.A.1 (1/1)
  CORE.SL.1 (3/3)
  CORE.W.1 (13/13)
  CORE.W.2 (9/9)
  MACRO.HALO.M4 (12/12)
  MACRO.SP.1 (3/3)
```

Violation:

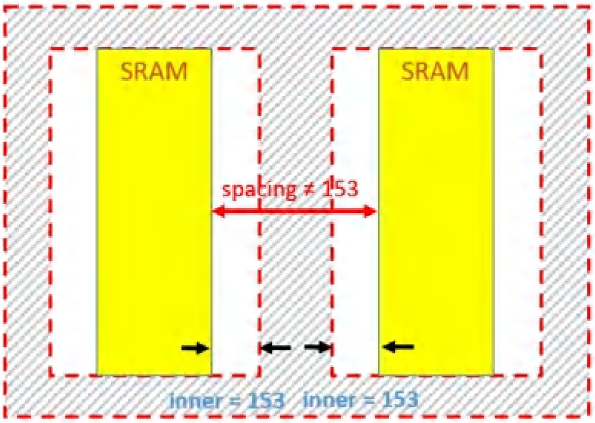
LOCATION
(359.490, 567.655) (411.630, 647.135)
(339.710, 83.315) (411.630, 346.930)
(566.465, 92.265) (612.295, 647.135)

Description:

```
UFC_check ;MACRO.SP.1;     ( DTMF_INST/ARB_INST/ROM_512x16_0_INST ) violated -> The horizontal and vertical spacing between macro and available sites must be greater than 100.0.
bbox = (359.490, 567.655) (411.630, 647.135)
```

-
  ```
  set_spacing_rule -name "DMTB.X2" -obj [base_cells SRAM] -ref [base_cells  SRAM] -direction left -
  parallel_run_length -0.42 \
        -shielding_obj [available_sites] {.spacing in {0.153}} \
        -description {set_space_constraint -label “DMTB.X2” -lib_cell1 SRAM -lib_cell2 SRAM -direction left -
  prl -0.42 -shielding_class useful_site -white_list 0.153}
  ```

Shielding spacing rules:

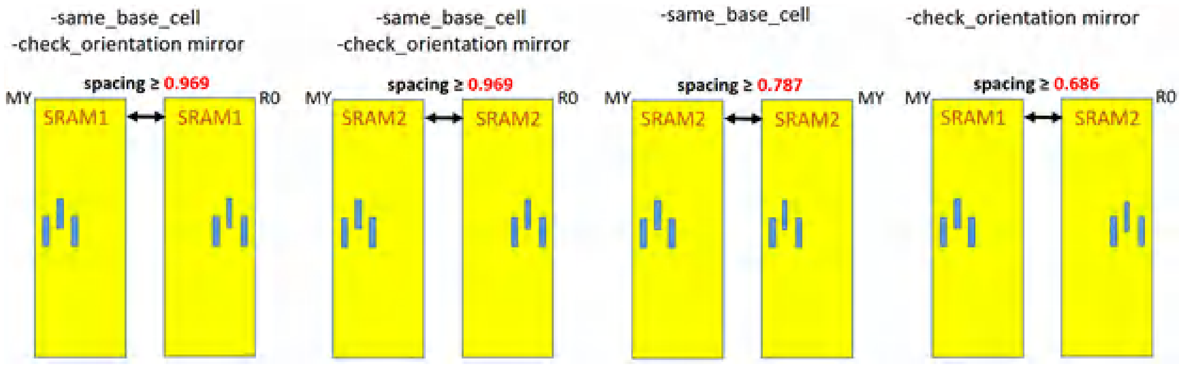


-
  ```
  set_spacing_rule -name "DMTB.X4" -obj [base_cells SRAM1 SRAM2] -ref [base_cells SRAM1 SRAM2] -
  same_base_cell -check_orientation mirror
         -direction left -parallel_run_length -0.42 {.spacing >= 0.969} \
         -description {set_space_constraint -label "DMTB.X4" -lib_cell1 {SRAM1 SRAM2} -lib_cell2 {SRAM1 SRAM2}
  -identical true -mirror true -direction left -prl -0.42 -min 0.969}
  ```

-
  ```
  set_spacing_rule -name "DMTB.X4" -obj [base_cells SRAM1 SRAM2] -ref [base_cells SRAM1 SRAM2] -same_base_cell
  \
          -direction left -parallel_run_length -0.42 {.spacing >= 0.787} \
         -description {set_space_constraint -label "DMTB.X4" -lib_cell1 {SRAM1 SRAM2} -lib_cell2 {SRAM1 SRAM2}
  -identical true -direction left -prl -0.42 -min 0.787}
  ```

-
  ```
  set_spacing_rule -name "DMTB.X4" -obj [base_cells SRAM1 SRAM2] -ref [base_cells SRAM1 SRAM2] -
  check_orientation mirror \
         -direction left -parallel_run_length -0.42 {.spacing >= 0.686} \
         -description {set_space_constraint -label "DMTB.X4" -lib_cell1 {SRAM1 SRAM2} -lib_cell2 {SRAM1 SRAM2}
  -mirror true -direction left -prl -0.42 -min 0.686}
  ```

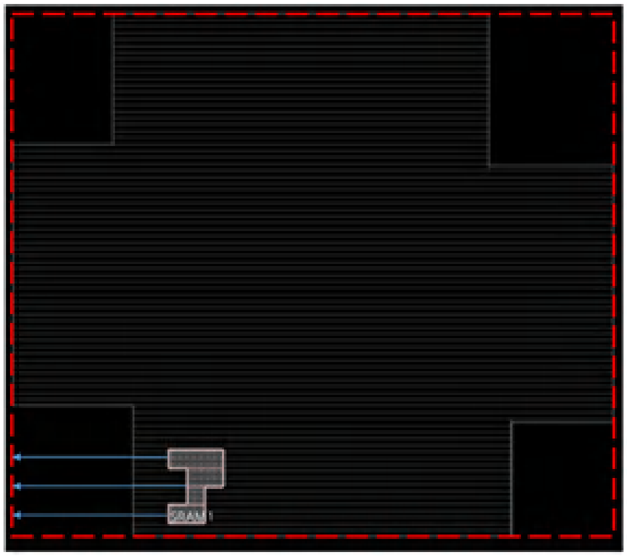
Identical and mirror spacing rules:



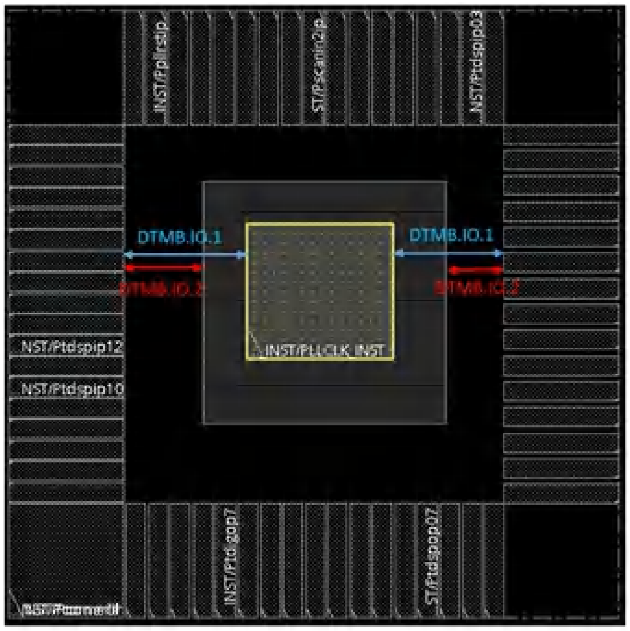
-
  ```
  set_spacing_rule -name "SRAM.L1" -obj [design_boundary] -ref [base_cells SRAM}] -check_edge same -
  bounding_box obj -enclosure -direction left {.spacing >= 10} \
         -description {set_location_constraint -label "SRAM.L1" -lib_cell SRAM -origin bottom_left -direction
  ```

```
horizontal -location_type bottom_left -min 10}
```

Here, the spacings between the left edges of macro SRAM1 and the left edge of design bounding box will be checked.



Spacing check between IO pads and other objects (macros and available_sites) in horizontal direction



-
  ```
  set_spacing_rule -name “DMTB.IO.1” -obj [base_cells pllclk] -ref [base_cells PDIDGZ] -direction horizontal
  {on_grid(.spacing-0.1,0.2)}\
     -description {set_space_constraint -label “DMTB.IO.1” -lib_cell1 pllclk -lib_cell2 PDIDGZ -direction
  horizontal -offset 0.1 -pitch 0.2}
  ```

-
  ```
  set_spacing_rule -name “DMTB.IO.2” -obj [available_sites] -ref [base_cells PDIDGZ] -direction horizontal
  {on_grid(.spacing-0.2,0.3)}\
     -description {set_space_constraint -label “DMTB.IO.2” -class1 useful_site -lib_cell2 PDIDGZ -direction
  ```

```
horizontal -offset 0.2 -pitch 0.3}
```

Related Information

- Supported UFC Commands

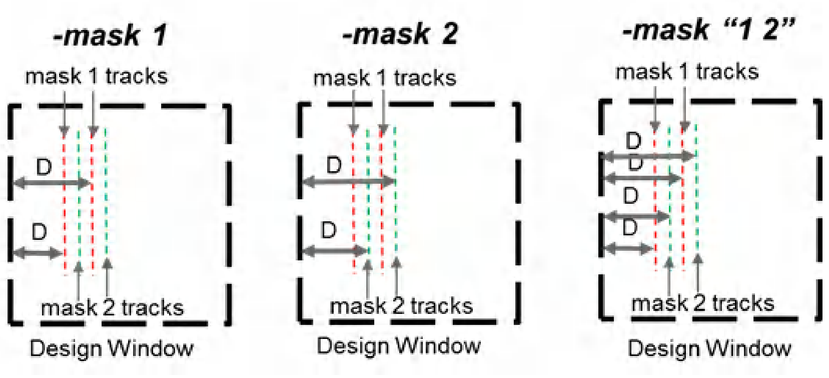
set_track_rule

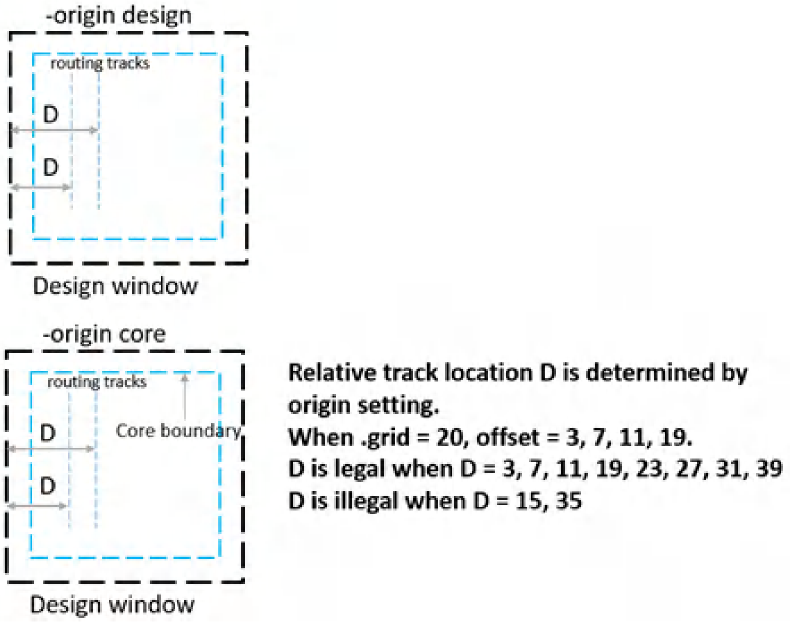
```
set_track_rule
-name "ruleName"
[-layer string]
[-mask string]
[-direction {horizontal | vertical}]
[-origin {core | design}]
[-grid float]
[-offset float]
[-description string]
[-severity string]
```

Checks the grid and offset of routing tracks on specified metal layers.

Parameters

`-description string`	Specifies the description of this rule.
`-direction`	Specifies the direction of track.
`-grid`	Specifies the repeated grid of tracks. **Note:** For track with grid 1 and offset 0, locations 0, 1, 2, 3, 4, 5, 6, 7… are ok for tracks. It is also acceptable for the track to only have locations 1, 2, 4, or 5. The UFC capability only checks whether the current tracks are on legal locations. It does not check whether all the legal locations have tracks on it.
`-layer string`	Specifies the metal layer of routing track to be checked.

`-mask` *`string`*	Specifies the mask color of routing track to be checked. **Note:** Multiple mask colors can be specified on DPT layer.  -mask 1 mask 1 tracks D D mask 2 tracks Design Window -mask 2 mask 1 tracks D D mask 2 tracks Design Window -mask "1 2" mask 1 tracks D D D D mask 2 tracks Design Window
`-name "`*`ruleName`*`"`	Specifies the rule name.
`-offset` *`float`*	Specifies the legal offset values of track location. **Note:** Offset can be a list of numbers to denote legal values of relative track location % grid.

`-origin {core | design}`	Specifies the origin of track. Origin used to determine track offset calculation from design window or core boundary.  -origin design routing tracks D D Design window -origin core routing tracks Core boundary D D Design window **Relative track location D is determined by origin setting.** **When .grid = 20, offset = 3, 7, 11, 19.** **D is legal when D = 3, 7, 11, 19, 23, 27, 31, 39** **D is illegal when D = 15, 35**
`-severity` *`string`*	Specifies the severity.

Example

- `set_track_rule -name "DMTB.TRACK.1" -layer M1 -direction horizontal -origin core -grid 0.5 -offset {0 1 3 5 7}`

Related Information

- Supported UFC Commands

set_white_area_extension_rule

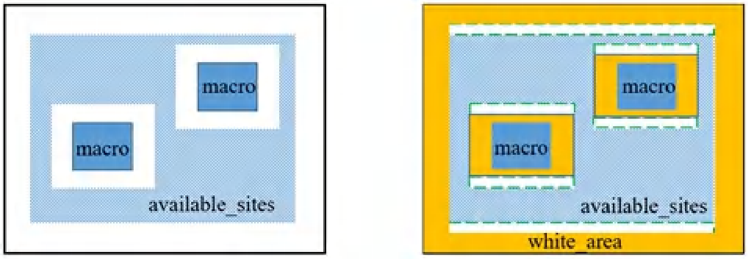
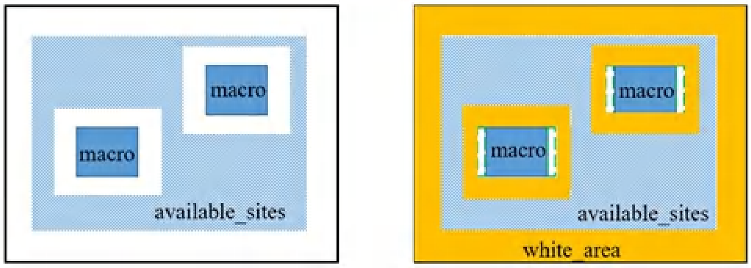
```
set_white_area_extension_rule
-obj string
-direction {horizontal | vertical | top | bottom | left | right}
-size float
[-description string]
[-severity string]
```

Specifies the object extension to calculate white_area object. The default white_area is the core area except available_sites and macros.

Notes:

- The `set_white_area_extension_rule` command should be specified before the other `set_*_rule` rules.
- The `set_white_area_extension_rule` command only supports available_sites and macros as -obj objects.
- The enclosure, spacing, and width rules support white_area. The white_area can be enclosed by the design boundary, spacing between white_area and white_area and the width of white_area.

Parameters

<table>
<tr><td colspan="2">-direction {horizontal | vertical | top | bottom | left | right}</td></tr>
<tr><td></td><td>Specifies the extend direction.
The following diagram shows white_area after extending available_sites in vertical direction:

The following diagram shows white_area after extending macro in horizontal direction:
 </td></tr>
<tr><td>-description <i>string</i></td><td>Specifies the description of this rule.</td></tr>
<tr><td>-obj <i>string</i></td><td>Specifies the target object to extend.
<b>Note:</b> The objects can only be available_sites and macros.</td></tr>
<tr><td>-size <i>float</i></td><td>Specify the extend size. A positive value means extend and a negative value means shrink.</td></tr>
<tr><td>-severity <i>string</i></td><td>Specifies the severity.</td></tr>
</table>

Example

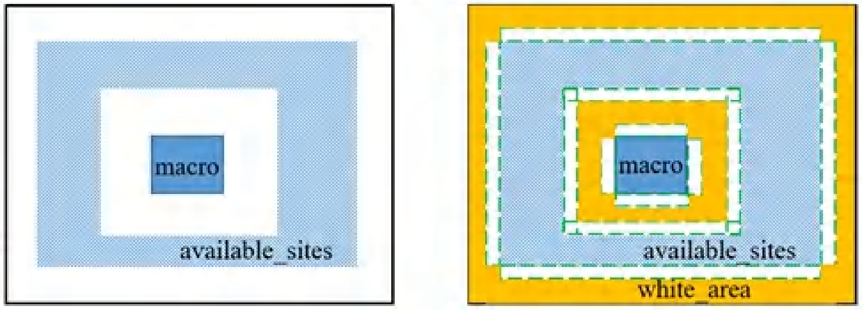
-
  ```
  set_white_area_extension_rule -obj [available_sites] -direction horizontal -size 10 \
        -description {set_white_space_object_extension -class useful_site -extend_side horizontal -extension 10}
  ```

-
  ```
  set_white_area_extension_rule -obj [available_sites] -direction vertical -size 10 \
        -description {set_white_space_object_extension -class useful_site -extend_side vertical -extension 10}
  ```

-
  ```
  set_white_area_extension_rule -obj [macros] -direction horizontal -size 10 \
        -description {set_white_space_object_extension -class macro -extend_side horizontal -extension 10}
  ```

-
  ```
  set_white_area_extension_rule -obj [macros] -direction vertical -size 10 \
        -description {set_white_space_object_extension -class macro -extend_side vertical -extension 10}
  ```

White_area after defining the above rules:



Related Information

- Supported UFC Commands

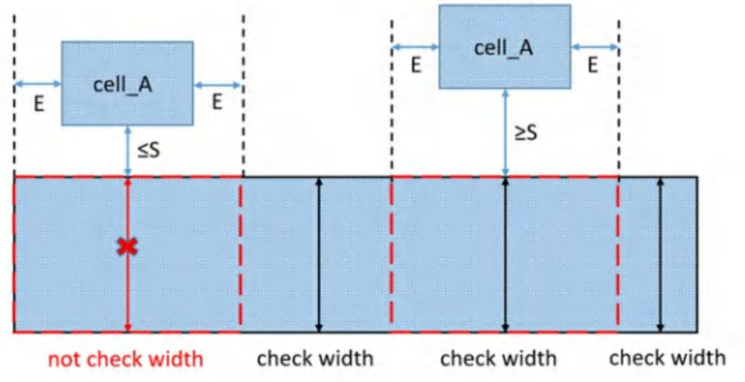
set_width_rule

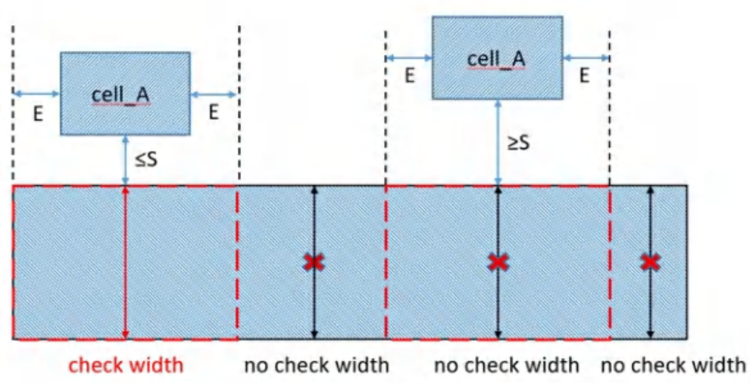
```
set_width_rule
expression_of_.width
-obj shapeObj
-name "ruleName"
[-direction {all | vertical | horizontal | corner_distance}]
[-type {simple |concave | jog | incorner}]
[-description string]
[-severity string]
[{-exception_triggering_base_cell shapeObj | -triggering_base_cell shapeObj}
[-triggering_spacing float]
[-effective_extension float]]
```

Specifies width constraints for specific object(s).

Parameters

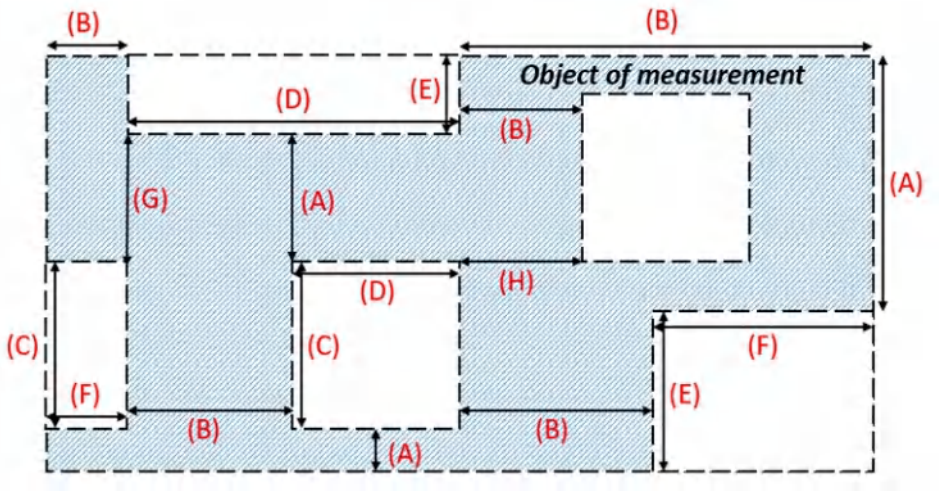
`-description` *`string`*	Specifies the description of this rule.
`-direction {all	vertical	horizontal	corner_distance}`	
	Specifies the check direction.

`-effective_extension` *`float`*	Specifies the effective extension value relative to the object or exception object. The extension direction is perpendicular to the direction specified by the `-direction` parameter. *Default*: 0 The `-effective_extension` parameter must be used with the `-exception_triggering_base_cell` or `-triggering_base_cell` parameters at the same time. The following diagram shows the relationship between them. For example: `-exception_triggering_base_cell cell_A -triggering_spacing S -effective_extension E`  cell_A E E ≤S cell_A E E ≥S not check width check width check width check width
`-exception_triggering_base_cell` *`shapeObj`*	
	Specifies the base cell name that triggers the width check exception.
`expression_of_.width`	Specifies the dimension expression of .width to check for the valid width.
`-name "`*`ruleName`*`"`	Specifies the rule name.
`-obj` *`shapeObj`*	Specifies the target object to check against this rule.
`-severity` *`string`*	Specifies the severity.
`-triggering_base_cell` *`shapeObj`*	Specifies the base cell name that triggers the width check.

`-triggering_spacing float`	Specifies the maximum spacing from the excepted or included width to the triggering base cell. The spacing direction is the same as the direction specified by the `-direction` parameter. **Note:** It must be used with the `-exception_triggering_base_cell` or `-triggering_base_cell` parameters at the same time. *Default*: 0 The `-triggering_spacing` parameter must be used with the `-exception_triggering_base_cell` or `-triggering_base_cell` parameters at the same time. The following diagram shows the relationship between them. For example: `-triggering_base_cell cell_A -triggering_spacing S -effective_extension E`  cell_A E E ≤S cell_A E E ≥S check width no check width no check width no check width
`-type {simple | concave | jog | incorner}`	
	Specifies the type of width to be checked.

Understanding width types and direction

The following image can be used as a quick reference for understanding the width direction and type.



Where,

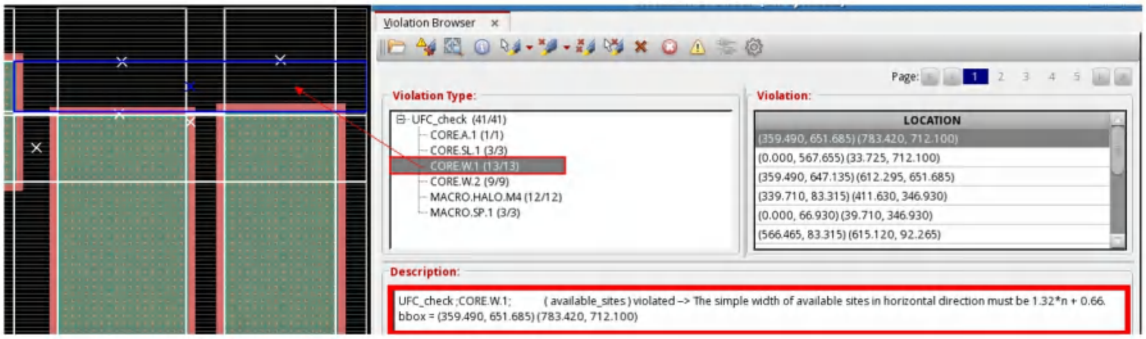
A	Simple width in vertical direction. (Width between two horizontal edge directly project to each other)
B	Simple width in horizontal direction.
C	In-corner width in vertical direction. (Width between two 270-degree corners)
D	In-corner width in horizontal direction.
E	Jog width in vertical direction. (Width between one 90-degree corner and one 270-degree corners)
F	Jog width in horizontal direction.
G	Concave width in vertical direction. (Width between two horizontal edges PRL = 0)
H	Concave width in horizontal direction.

Example

- The following command specifies the rule name, the expression of .width to check for the valid width and the type, direction and description.

```
set_width_rule -name "WIDTH.CORE.1" -obj [available_sites] -type simple -direction horizontal {on_grid
(.width-0.66, 1.32)} \
-description {The simple width of available sites in horizontal direction must be 1.32*n+0.66.}
```

  Violations of width rule:



- UFC_check (41/41)
  - CORE.A.1 (1/1)
  - CORE.SL.1 (3/3)
  - CORE.W.1 (13/13)
  - CORE.W.2 (9/9)
  - MACRO.HALO.M4 (12/12)
  - MACRO.SP.1 (3/3)

LOCATION
(359.490, 651.685) (783.420, 712.100)
(0.000, 567.655) (33.725, 712.100)
(359.490, 647.135) (612.295, 651.685)
(339.710, 83.315) (411.630, 346.930)
(0.000, 66.930) (39.710, 346.930)
(566.465, 83.315) (615.120, 92.265)

UFC_check ;CORE.W.1;   (available_sites) violated -> The simple width of available sites in horizontal direction must be 1.32*n + 0.66.
bbox = (359.490, 651.685) (783.420, 712.100)

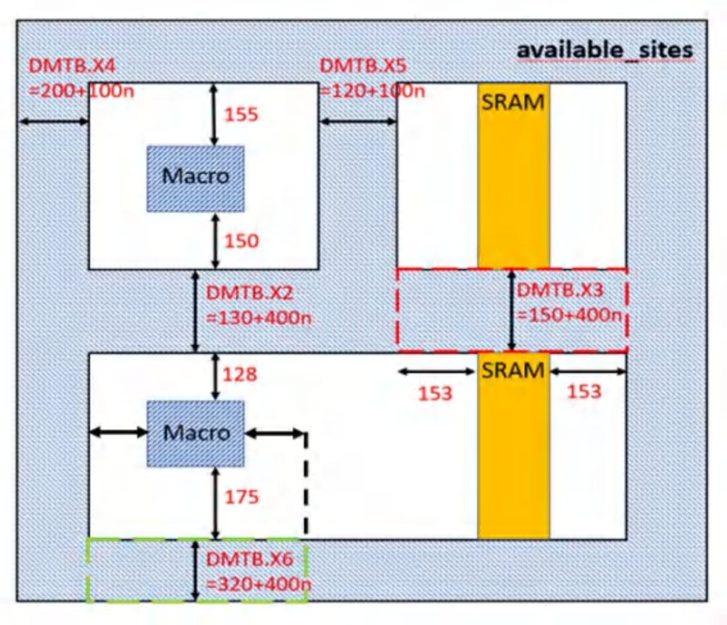
-
  ```
  set_width_rule -name "DMTB.X2" -obj [available_sites] -type simple -direction vertical -
  exception_triggering_base_cell [base_cells SRAM}] \
      -effective_extension  0.153 {on_grid(.width-0.13,0.4)} \
      -description {set_width_constraint -label "DMTB.X2" -class useful_site -width_type simple -direction
  vertical -except_project_lib SRAM -project_extension 0.153 -grid 0.4 -offset 0.13}
  ```

-
  ```
  set_width_rule -name "DMTB.X3" -obj [available_sites] -type simple -direction vertical {on_grid(.width-
  0.15,0.4)} \
       -description {set_width_constraint -label "DMTB.X3" -class useful_site -width_type simple -direction
  vertical -grid 0.4 -offset 0.15}
  ```

-
  ```
  set_width_rule -name "DMTB.X4" -obj [available_sites] -type simple -direction horizontal -
  exception_triggering_base_cell [base_cells SRAM] {on_grid(.width-0.2,0.1)} \
       -description {set_width_constraint -label "DMTB.X4" -class useful_site -width_type simple -direction
  vertical -except_project_lib SRAM -grid 0.1 -offset 0.2}
  ```

-
  ```
  set_width_rule -name "DMTB.X5" -obj [available_sites] -type simple -direction horizontal {on_grid(.width-
  0.12,0.1)} \
       -description {set_width_constraint -label "DMTB.X5" -class useful_site -width_type simple -direction
  vertical -except_project_lib SRAM -grid 0.1 -offset 0.12}
  ```
- The following command uses the -exception_triggering_base_cell, -effective_extension and -triggering_spacing width rules:
  ```
  set_width_rule -name "DMTB.X6" -obj [available_sites] -type simple -direction vertical -
  exception_triggering_base_cell [base_cells Macro}] \
       -effective_extension 0.15 -triggering_spacing 0.16 {on_grid(.width-3.2,0.4)} \
       -description {set_width_constraint -label "MTB.X6" -class useful_site -width_type simple -direction
  vertical -except_project_lib Macro -project_extension 0.15 -project_spacing 0.16 -grid 0.4 -offset 3.2}
  ```

  Here, for DMTB.X6, the spacing between the below macro and available_sites is 0.175 which is larger than 0.16 specified by -triggering_spacing. So the available_sites area with extension 0.15 specified by -effective_extension will be checked:



Related Information

- Supported UFC Commands