cadence®

Mixed Signal (MS) Interoperability Guide

Product Version 22.13
Last Updated in July 2023

© 2023 Cadence Design Systems, Inc.
Printed in the United States of America.

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

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

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

All other trademarks are the property of their respective holders.

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

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

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. 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. Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

Contents

About This Manual 14
Audience 14
How to Use This Document 14
Related Documents 16
Innovus Product Documentation 16
Innovus Stylus Common UI Product Documentation 17
Virtuoso Documentation 18
1 19
Getting Started 19
1.1 Product and Installation Information 19
1.2 Setting the Run-Time Environment for Innovus 20
1.2.1 Supported and Compatible Platforms 20
1.2.2 64-Bit Version of Innovus Applications 20
1.3 Temporary File Locations 21
1.4 OpenAccess 21
1.5 Launching the Console 21
1.6 Accessing Documentation and Help in Innovus 23
1.6.1 Launching Cadence Help From the Command Prompt 23
1.6.2 Using the Innovus man and help Commands on the Text Command Line 23
1.7 Starting the Virtuoso Studio Design Environment 27
1.8 Innovus and Virtuoso Release Compatibility Information 28
2 30
Overview of Mixed Signal Interoperability 30
2.1 Mixed Signal Solution - Introduction 30
2.1.1 Design Methodologies Supported by the Cadence Mixed Signal Solution 31
3 34
Technology Data Preparation 34
3.1 Software Requirements 34
3.2 Technology Data Preparation 35
3.2.1 Library and Technology Requirements 35
3.2.2 Preparing the Technology Library 35

3.2.3 Determining Whether Your MSOA PDK Is Traditional or Rapid 49
3.2.4 Preparing the IP Library 50
3.2.5 Creating a Rapid MSOA PDK with Multiple foundry_innovus Constraint Groups 52
3.2.6 Preparing a Technology Library with Multiple LDRSs 53
3.2.7 Rapid PDK ASCII Techfile Structure 59
4 70
Design Data Preparation 70
4.1 Special Settings/Instructions To Enable the Mixed-Signal Flow 71
4.2 Name Space Mapping in the OpenAccess Interoperable Flow 73
4.3 Converting an Existing LEF/DEF-based Design into OpenAccess 73
4.3.1 Converting a LEF/DEF-based design into OpenAccess by using oaOut 75
4.4 Migrating a LEF and Floorplan File-based Flow to OpenAccess 78
4.5 Running Innovus in the OpenAccess Mode 80
4.5.1 Initializing Innovus for Implementing a Digital Block in OpenAccess 80
4.5.2 Initializing Innovus with the OpenAccess Database of a Design Created in Virtuoso XL 81
4.6 Interoperability of Constraint Groups and Non-Default Rules between Virtuoso and Innovus 83
4.6.0.1 Example 83
4.7 Generic Guidelines To Run the Netlist-Driven Mixed-Signal Flow 84
5 85
Schematic-Driven Mixed Signal Design Flow 85
5.1 Schematic-Driven Mixed Signal Flow 85
5.1.1 Overview 85
5.1.2 Top-Level Early Floorplanning 87
5.1.3 The Analog Block Implementation Sub-Flow 92
5.1.4 Custom Digital Implementation Flow 95
5.1.5 Digital Block Implementation Flow 97
5.1.6 OpenAccess Based Interoperable Flow between Innovus and Virtuoso 105
5.1.7 Top-Level Block/Chip Assembly flow 105
5.1.8 Virtuoso Top-Level Assembly 107
5.1.9 Virtuoso Chip-Level Layout Assembly 108
5.2 Flow Example: Power PushDown Flow for Digital Blocks in AoT Designs 110
5.2.1 Performing Top-Level Floorplanning in Virtuoso 110
5.2.2 Pushing Power Shapes into the Digital Block 125
5.2.3 Implementing the Digital Block in Innovus 131
5.2.4 Bringing Back the Digital Block into Virtuoso for Design Assembly 135

6 136
Virtuoso Digital Implementation 136
6.1 Overview 136
6.2 Invoking the VDI Environment 138
6.2.1 Starting Innovus Using a Verilog Netlist 141
6.2.2 Starting Innovus Using an OpenAccess Database from Virtuoso 145
6.2.3 Starting Innovus Using an Existing Script 148
6.2.4 Loading Existing Form Configuration 149
6.3 Configuring Power Planning and Power Routing 151
6.4 Creating the View Definition File 158
6.4.1 Implementing a Digital Block in the Single Timing Analysis Mode 159
6.4.2 Implementing a Digital Block in the BcWc or Min/Max Analysis Mode 161
6.4.3 Implementing a Digital Block in the MMMC Analysis Mode 162
6.5 Specifying Physical Cells 169
6.6 Specifying Optional Plugin Scripts 171
6.7 Generating the Innovus Script 171
6.7.1 With Customization 171
6.7.2 Without Customization 172
6.8 Completing the Implementation 173
6.9 Sample Files 175
6.9.1 Sample View Definition File 175
6.9.2 Sample SDC File 179
6.9.3 Generated Innovus Script Sample 180
6.10 Helpful Hints 183
7 184
Quick Abstract Inference 184
7.1 Overview of Quick Abstract Inference 184
7.2 Why Abstracts Are Still Required for Standard Cells 185
7.3 How QAI Works 185
7.3.1 Rules for Abstract Inference 187
7.3.2 Detailed Description of Data Objects 190
7.3.3 Blockage Modeling in QAI 190
7.3.4 QAI Pin Modeling 197
7.3.5 Antenna Annotation Utility for Creating Accurate Antenna Information for the Innovus Flow 199
8 201

Netlist-Driven Mixed Signal Design Flow 201
8.1 Overview 201
8.2 Technology and IP Library Preparation 203
8.3 Verilog Netlist Creation 203
8.4 General comments 203
8.5 Floorplanning 204
8.5.1 Floorplanning of Verilog Netlist Using Blackboxes 205
8.5.2 Generate From Source for Soft Analog Block Layout Using Virtuoso 209
8.5.3 Load Physical View to Merge Optimized Pin Locations and Block Boundary 213
8.5.4 Physical Implementation of Soft Analog Blocks Using Virtuoso 216
8.5.5 Physical Implementation of Soft Digital Blocks Using Innovus 217
8.6 Top-level Analog Net and Power Routing 218
8.7 Top-level Design Implementation 219
8.8 Final Chip Integration and Sign-Off 220
8.9 ECO Flows 220
9 221
OpenAccess Wiring Terminology 221
9.1 Creating an Interoperable Digital Block Floorplan in Virtuoso Layout Suite XL 222
9.2 Understanding Symbolic and Geometric Routing 230
9.2.1 Differences Between Symbolic and Geometric Routing 230
9.2.2 Identifying Symbolic Routing and Geometric Routing 231
9.2.3 Identifying the Route Status for Symbolic and Geometric Routing 232
9.2.4 Ways of Creating Wires in Virtuoso and Innovus Environment 233
9.3 Locking a Net Using Virtuoso Space-Based Router (VSR) 234
9.3.1 Locking a partially routed net 234
9.3.2 Locking a net that is completely unrouted 235
9.3.3 Locking a completely routed net 235
9.3.4 Unlocking Nets 235
9.3.5 Highlighting Locked Nets 235
9.4 Editing Net Attributes in Virtuoso 235
9.5 Wire/Net Mapping Between Virtuoso and Innovus 236
9.6 Handling of Wires from Virtuoso in Innovus 237
9.6.1 Conversion of Wires to DEF SPECIALNETS in Innovus 237
9.6.2 Implications of the Conversion in Innovus 239
9.6.3 Addressing the Conversion Issue 240
9.6.4 How To Prevent the Conversion Issue 240

9.7 Wiring Connectivity in Innovus 242
9.8 Wiring Extraction in Innovus 242
10 243
Trim Metal Interoperability 243
10.1 Introduction to Trim Metal 243
10.1.1 The TRIMMETAL LAYER 243
10.1.2 General Requirements for N7, N5, and N3 Designs 244
10.1.3 What Is Trim Metal? 244
10.2 Technology Requirements for Trim Metal Insertion 245
10.3 Trim Metal Insertion Flow in Virtuoso 247
10.3.1 Trim Shape Rules 247
10.3.2 Trim Shape Extension Models 248
10.4 Supporting Tool Enhancements 251
11 252
Routing Constraint Interoperability 252
11.1 An Overview of Routing and Constraint Interoperability 253
11.2 Steps for Creating an Interoperable NDR in Virtuoso 256
11.3 Creating Interoperable Shielding Constraints in Virtuoso 259
11.4 Routing Constraints Understood by NanoRoute (Digital Router in Innovus) 261
11.5 Creating/Viewing Interoperable Routing Constraints in Virtuoso 262
11.5.1 Creating Interoperable Constraints Using Virtuoso Constraint Manager 263
11.5.2 Checking OpenAccess Database with Constraints Prior to Bringing a Design into OA DB Checker System 263
11.6 Creating Interoperable Routing Constraints Using Innovus 264
11.7 Hierarchical Propagation of Constraints 267
11.8 Creating/Adding Mixed-Signal Routing Constraints on IP Blocks 269
11.9 Routing Constraints Interoperability between NanoRoute and VSR 271
11.9.1 Getting a List of Nets with skip_routing Attribute 271
11.9.2 Checking Routing Results Against the Routing Constraints 272
11.9.3 Creating Interoperable Library between Innovus and Virtuoso 273
11.10 Trying the Constraint Interoperability Environment and the run_vsr Command in Innovus for the First Time 274
12 278
Working with Buses in OpenAccess 278
12.1 Settings for Automatic Annotation of Bus Bits During Abstract Generation using Virtuoso

Abstract Generator 278
12.2 The Interface Bit Setting of a Terminal 279
12.2.1 Checking for the Presence of the Interface Bit 279
12.2.2 Running verilogAnnotate 280
13 281
High Frequency Router In Innovus 281
13.1 Overview 281
13.2 Invoking NRHF 282
13.3 Controlling NRHF Routing 283
13.4 Supported Routing Styles 284
13.4.1 Non-default Routing 284
13.4.2 Shielding 285
13.4.3 Differential Pair 286
13.4.4 Match Length 287
13.4.5 Resistance Match 288
13.4.6 Layer Match 289
13.4.7 Bus Routing 289
13.4.8 Length Control 291
14 292
Working with Vias in OpenAccess 292
14.1 Overview 292
14.2 Types of Via Definitions 294
14.2.1 Custom Vias or customViaDefs 294
14.2.2 VIARULE or standardViaDefs 295
14.2.3 standardViaVariants 296
14.3 How Tools Find and Use Vias 298
14.3.1 generateVias 300
14.3.2 Summary Table 300
14.3.3 cdsFixedVias 302
14.4 Examples 306
14.4.1 Setting the LDRS in Virtuoso Layout Suite 308
14.4.2 Checking Vias in the Tech File 310
14.5 Directing the Router to Use Only the Vias in the ITDB in Innovus 314
15 315
Placing Standard-cell-based Designs from Virtuoso 315

15.1 Overview 315
15.2 Guidelines for Placing a Standard-cell-based Design from Virtuoso in Innovus 316
15.3 What are Sites and Rows in Innovus and Why They Are Needed 318
15.4 Site Property in the Cell View of Standard Cells 322
15.5 Site Definition in the Technology Library 324
15.6 Standard Row Creation by Innovus During init_design 327
15.7 Switching the Orientation of the Standard Rows 328
16 330
Using NanoRoute To Route a Design from Virtuoso 330
16.1 Overview 331
16.2 What Are Routing Tracks? 331
16.3 Guidelines for Using Innovus To Route a Design from Virtuoso 334
16.4 Interoperability of Routing Tracks 335
16.4.1 Supported Use Models for the Interoperability of Routing Tracks 336
16.4.2 Role of add_tracks in the Interoperability of Routing Tracks 339
16.4.3 Role of setOaxMode -updateMode in the Interoperability of Routing Tracks 339
16.5 Track Alignment in Innovus 340
16.5.1 Relationship Between Wire Width, Routing Track Pitch, and Minimum Spacing Requirement 340
16.5.2 Alignment of I/O Pins with Routing Tracks 342
16.5.3 Alignment of I/O Pins with Non-Uniform-Pitch Tracks in Lower Technology Nodes 344
16.5.4 Access to I/O Pins with Double Patterning Layers in Lower Technology Nodes 345
16.5.5 Access to Standard-Cell I/O Pins in Lower Technology Nodes 346
16.5.6 Access to Standard-Cell I/O Pins with Double Patterning Layers 350
16.5.7 Track Patterns with Non-Uniform Pitch 352
16.5.8 Track Patterns with Non-Uniform Pitch and Non-Default Width 353
17 355
Design and Core Boundary in Innovus 355
17.1 Design and Core Boundary Definitions 355
17.2 Matching the Orientation of Rows and Standard Cells 358
17.3 Handling Gaps Between Standard Cells and the Design Boundary 362
17.3.1 Gap Between the Left Edge of the Leftmost Standard Cell and the Design Boundary 362
17.3.2 Gaps Between the Left and Bottom Edges of Standard Cells and the Design Boundary 367
17.4 Creating Double-Height Rows 370
17.5 Creating Routing Tracks with Respect to coreBox 371

17.6 Resizing the PRBoundary Object in Virtuoso 375
18 377
Static Timing Analysis for Mixed Signal Designs 377
18.1 Overview 379
18.2 OpenAccess Compatibility for Mixed-Signal STA Flow 380
18.2.1 Basic Design Requirements 380
18.2.2 Requirements for Correct Connectivity Propagation 386
18.2.3 Requirements for OpenAccess Compatibility 388
18.2.4 Useful Utilities and Information 393
18.3 Handling Schematic-Driven Designs Implemented Using a Non-Interoperable PDK 398
18.3.1 Checking the Technology Database Graph of a Specific Design Library 398
18.3.2 Opening a Design Attached to a Non-Interoperable PDK in Innovus 400
18.3.3 Ways To Switch a Design Library from a Non-Interoperable to an Interoperable PDK 401
18.4 Running STA by Flattening the Design 407
18.4.1 Steps for Running STA Using the Flatten Approach 410
18.5 Running STA by Using the FTM 415
18.5.1 Generating the FTM for Each Block 419
18.5.2 Running STA on Top-level Design with FTM of Blocks 422
18.6 Difference between Flat and FTM Approaches 426
18.7 Guidelines for Running STA Flow on Mixed-Signal Design 427
18.8 Creating a Quick Timing Model for Analog IPs in Innovus 427
18.9 Different Ways of Running assembleDesign 430
18.9.1 Running assembleDesign in the Batch Mode 430
18.9.2 Running assembleDesign in the Incremental Mode 433
18.9.3 Differences between the Batch and Incremental Modes of assembleDesign 436
18.9.4 Running assembleDesign with the -allTimingBlocks Option 436
18.9.5 Tips on Running assembleDesign -allTimingBlocks 438
18.9.6 Caveats for and Limitations of assembleDesign -allTimingBlocks 439
18.9.7 Running assembleDesign in the Batch Mode and Subsequently in the Incremental Mode 439
18.9.8 Limitations of assembleDesign 440
18.10 Parasitic RC Extraction for Running MS-STA 446
18.10.1 Running Quantus Extraction with postRoute Engine for a Routed Design 447
18.10.2 Running Quantus Extraction with Signoff Effort Level 447
18.10.3 Using the Quantus Layer Map File 448

18.10.4 Auto-creation of the Quantus Layer Map File from Innovus 449
18.10.5 Running Signoff Quantus Extraction in the OpenAccess Mode 449
18.10.6 Auto-creation of Input Files from Innovus To Run Standalone Quantus QRC 449
18.10.7 Running Standalone Quantus QRC and Loading the Resulting SPEF Files Back to Innovus 450
18.11 Tips for Debugging the No Constrained Timing Path Issue Generated by report_timing 451
18.11.1 Case Studies 452
18.11.2 Basic Conditions for Reporting the Timing on a Path 458
18.11.3 Common No Constraint Situations 458
18.11.4 Checking the Validity of a Timing Constraint 466
18.11.5 Checking the Existence of a Design Object in the Layout 471
18.11.6 Common Invalid Timing Path Situations 472
19 487
Chip Finishing and ECO Flows 487
19.1 Overview 487
19.2 Virtuoso-Based ECO Flow 488
19.3 Innovus-Based ECO Flow 490
19.3.1 Overview 490
19.3.2 Pre-Mask ECO Flow Steps 491
19.3.3 Post-Mask ECO Flow Steps 493
19.3.4 Example Post-Mask ECO Scenarios 495
20 497
OpenAccess Database Interoperability Checker 497
20.1 Overview 497
20.2 Running the OA DB Checker 497
20.3 Check Library - Innovus Interoperability Library Checker 499
20.3.1 Technology DB Checker 499
20.3.2 Library DB Checker 501
20.3.3 Check Pins between Two Cellviews for the Remaster Instance 501
20.3.4 Report File Name 501
20.4 Check Design - Innovus Interoperability Design Checker 503
20.4.1 CellView(s) 503
20.4.2 Report File Name 503
20.4.3 Checks 506
20.5 Create Check File - Create Check Design File 516
20.6 Viewing OA DB Checker Violation Markers in the Annotation Browser in Virtuoso 517

20.7 Run Innovus - Innovus Launch GUI 518
20.8 OA DB Checker Use Case Scenarios for Virtuoso Users 519
21 520
oaZip Utility 520
21.1 The oazip Utility to Compress/Decompress Databases 520
21.1.1 Command Syntax 520
21.1.2 Arguments 521
22 524
Voltage Dependent Rule Interoperability 524
22.1 Overview 524
22.2 VDR Syntax 525
23 527
Useful Tips 527
23.1 Removing Pipe Character from Instance Names 528
23.2 Viewing PCells in Innovus 528
23.3 Global Net Name Collision Resolution 529
23.4 NanoRoute Support for Nets 529
23.5 Saving Blackboxes in OpenAccess 529
23.6 Importing the Power and Ground Nets Connections Using Verilog Netlist 530
23.7 Turning Off Power and Ground Connections in a Netlist 530
23.7.1 verilogAnnotate Command Syntax 531
23.8 Generating Power and Ground Pins 532
23.9 Using Abstract Views in Innovus 532
23.10 Performing Power Routing Outside Innovus 532
23.11 Generating Abstracts with Antenna Information 532
23.12 Mapping Virtuoso Bind Keys to Innovus Bind Keys 534
23.13 SKILL to TCL Mapping 536
23.14 Generating a Verilog Netlist for Innovus from a Virtuoso Schematic 538
23.14.1 Netlisting Options 538
23.14.2 Schematic and Hierarchy Editor Setup 538
23.14.3 Netlist Generation 539
23.14.4 Problems and Solutions 539
23.14.5 Filtering Power and Ground Nets in Verilog 540
23.14.6 Netlisting Options 540
23.14.7 Schematic Modifications 540

23.14.8 Limitation due to Split Bus and Bundle with `Merge All' 543
23.15 Creating a Non-Default Rule in Virtuoso and then Using it in Innovus 545
23.15.1 Using Virtuoso 545
23.15.2 Using Innovus 546
23.16 Troubleshooting Common Errors in Innovus OpenAccess Flow 547
23.16.1 IMPOAX-717 Error 547
23.17 Working with Old Versions of OpenAccess Data 547

About This Manual

This manual describes how to implement the digital mixed-signal flow.

Audience

This manual is written for designers with experience in both digital and custom design environment. 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.

This document applies to the Innovus user interface. See the Stylus Common UI Mixed Signal (MS) Interoperability Guide (Limited Access) document for the Innovus Stylus user interface.

How to Use This Document

The Mixed Signal Interoperability Guide covers many different Mixed signal related topics and has detailed descriptions on various OpenAccess-based flows supported by the tools involved in this flow. This portion of the document attempts to guide you to the pertinent content in this documentation, based on what you wish to accomplish so that the study of the documentation becomes a task-based approach. You can go directly to the recommended sections of the document to complete the task at hand.

Task: Become familiar with the supported Mixed Signal flows.

Chapter 2 of the document, Overview of Mixed Signal Interoperability, has a brief description of the various supported flows. Once you have decided which flow to use, go to the chapter for that specific flow to view its detailed description. For instance, if you decide to exercise the Analog-on-Top (AoT) flow, go to Chapter 5 - Schematic-Driven Mixed Signal Design Flow.

Task: Set up the various libraries and support files for a Mixed Signal flow.

Chapter 3 - Technology Data Preparation has most of the information needed to prepare the libraries needed for this flow.

If there is a need for special handling of vias in the design, Chapter 13 - Working with Vias in OpenAccess is highly recommended.

Task: Make any special setting required to make the software work properly for Mixed Signal users.

Study the section "*Special Settings/Instructions To Enable the Mixed-Signal Flow*" in Chapter 4 - Design Data Preparation. Pay special attention to the `setOaxMode` command in Innovus because

this command controls many of the Innovus Mixed Signal features. For more information on the `setOaxMode` command, refer to the *Innovus Text Command Reference*.

Task: Bring a design implemented in Virtuoso into Innovus for editing or analysis.

Note that this flow requires an interoperable PDK to be in place before the design can be opened in Innovus. After the interoperable PDK is available, the design can be loaded in Innovus by using the following settings/commands after launching Innovus:

```
set init_design_netlisttype {OA}    # This tells Innovus that the netlist connectivity
for the design is coming through an OA DB, as opposed to a Verilog netlist

set init_oa_design_lib {mylib}      # This is the name of the library that contains the
design to be opened in Innovus

set init_oa_design_cell {top}       # This specifies the name of the top level cell in
the design. In this example, the cell name is "top".

set init_oa_design_view {layout}    # This tells Innovus which view of the cell "top" to
open. In this example, the layout view is opened.

set init_oa_ref_lib {gsclib045}     # This tells Innovus which reference/technology is
being used for this design. In this example, gsclib045 is the interoperable PDK to be
used.

set init_pwr_net {VDD AVDD}         # Power nets being used in the design are defined.
In this example VDD and AVDD are being used as power nets  in our design.

set init_gnd_net {GND AGND}         # Ground nets being used in the design are defined.
In this example GND and AGND are being used as ground nets in our design.

set init_mmmc_file {viewDefinition.tcl}  # This setting is optional as it is used if
you are bringing the design into Innovus to perform timing analysis or optimization. It
points to the view definition files that are used to set up the environment for Multi
Mode Multi Corner (MMMC) analysis and implementation.

init_design                         # This is the actual command that will load the
design into Innovus
```

Refer to the Static Timing Analysis for Mixed Signal Designs for more details on how Virtuoso-based designs can be imported into Innovus.

Task: Understand the details of the specific flow that you have decided to use.

If you are going to be using the Analog-on-Top (AoT) flow, refer to Chapter 5 - Schematic-Driven Mixed Signal Design Flow and Chapter 6 - Virtuoso Digital Implementation.

If you are going to be using the Digital-on-Top (DoT) flow, refer to Chapter 8 - Netlist-Driven Mixed Signal Design Flow.

If you will be bringing the entire Mixed Signal design from one implementation platform to another (Virtuoso to Innovus, or Innovus to Virtuoso), you will be exercising the Mixed-Signal-on-Top flow (MSoT). MSoT flow supports Mixed Signal STA, Mixed Signal Floorplaning, Mixed Signal Routing, and so on.

Task: Run Static Timing Analysis on a Mixed Signal design coming from Virtuoso with digital content

Refer to Chapter 20 - OpenAccess Database Interoperability Checker, followed by Chapter 18 - Static Timing Analysis for Mixed Signal Designs.

Task: Floorplan the design in Virtuoso and Innovus, and share the floorplan information between the two cockpits

Refer to Chapter 7 - Quick Abstract Inference.

Refer to Chapter 5 - Schematic-Driven Mixed Signal Design Flow and study the section titled “Controlling Floorplanning Information Read in from the OpenAccess Cellview”.

Related Documents

For information on installing Cadence products, see *Cadence Installation Guide*.

For information on library structure, the library definitions file, and name mapping for data shared by multiple Cadence tools, see *Cadence Application Infrastructure User Guide*.

Innovus Product Documentation

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

- What's New in Innovus
  Provides information about new and changed features in this release of the Innovus family of products.
- Innovus User Guide
  Describes how to install and configure the Innovus software, and provides strategies for implementing digital integrated circuits.
- 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 the Innovus database access commands and provides a brief description of syntax and usage.
- Innovus Foundation Flows User Guide
  Describes how to use the scripts that represent the recommended implementation flows for digital timing closure with the Innovus software.
- 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.

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

Innovus Stylus Common UI Product Documentation

- Innovus Stylus Common UI Migration Guide
  Provides information on migrating from legacy to the Stylus Common UI version of the Innovus software.
- 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 User Guide
  Describes how to install and configure the Innovus Stylus Common UI software, and provides strategies for implementing digital integrated circuits.
- 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.

Virtuoso Documentation

- For information on the design environment n Virtuoso, see *Virtuoso Studio Design Environment User Guide*.
- For information on database SKILL functions, including data access functions, see *Virtuoso Studio Design Environment SKILL Reference*

Virtuoso Studio Design Environment Tools

For information on the analog design editing tools, see the following documents:

- *Virtuoso Schematic Editor User Guide*
- *Design Data Translator's Reference*

Virtuoso Layout Suite Tools

- For information on how to perform design tasks with the Virtuoso Layout Suite tools, see the following documents:
  - *Virtuoso Layout Suite XL: Basic Editing User Guide*
  - *Virtuoso Layout Suite GXL Reference*
  - *Virtuoso Layout Suite XL User Guide*
- For information on custom layout SKILL functions, see *Virtuoso Layout Suite SKILL Reference*.
- For information on how to use the Virtuoso custom digital placement capability, see *Virtuoso Custom Digital Placer User Guide*.

1

Getting Started

- Product and Installation Information
- Setting the Run-Time Environment for Innovus
  - Supported and Compatible Platforms
  - 64-Bit Version of Innovus Applications
- Temporary File Locations
- OpenAccess
- Launching the Console
- Accessing Documentation and Help in Innovus
  - Launching Cadence Help From the Command Prompt
  - Using the Innovus man and help Commands on the Text Command Line
- Starting the Virtuoso Studio Design Environment
- Innovus and Virtuoso Release Compatibility Information

1.1 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 the Product and Licensing Information chapter in the *Innovus User Guide*.

For information about Virtuoso licenses, see *Virtuoso Software Licensing and Configuration User*

Guide on support.cadence.com.

1.2 Setting the Run-Time Environment for Innovus

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.

1.2.1 Supported and Compatible Platforms

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

1.2.2 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.

1.3 Temporary File Locations

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

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

The name of the tmp_dir will look like:

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

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

```
innovus_temp_10233_farm254_bob_nfp9ez
```

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

1.4 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`.

1.5 Launching the Console

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

```
innovus>
```

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

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

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.

1.6 Accessing Documentation and Help in Innovus

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

1.6.1 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.

1.6.2 Using the Innovus man and help Commands on the Text Command Line

1.6.2.1 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
  ```

1.6.2.2 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 information for each getAllLayers parameter.

  For a detailed description of the command and all of its parameters, use the man command:

  man getAllLayers

  type Specifies the type of the layer. Innovus supports six types of layers, 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)
```

1.6.2.3 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.
  ```

1.6.2.4 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.

1.7 Starting the Virtuoso Studio Design Environment

Virtuoso Studio Design Environment software is built into binary files--or *workbenches*--containing blocks of executable computer code. Each workbench contains the code to run several related applications.

Note: From IC614, `virtuoso` is the recommended workbench.
The `layout` (and `layoutPlus)` workbenches are being phased out from IC614, but will still be accessible via symlinks in the short-to-medium term. It is however recommended that you now use the `virtuoso` executable.

When you start the Virtuoso® Studio Design Environment, the Command Interpreter Window (CIW) opens. From the CIW, you can access Cadence applications that you are licensed to run.

To start a workbench, type the workbench binary name at the system prompt. For example, to start the `virtuoso` workbench, type `virtuoso` at the system prompt. Your system administrator can tell you the command to type to run your particular set of applications.

> The installation procedure puts your Cadence executables in your_install_dir/tools/dfII/bin. You must not move any of the Cadence executables from this location or they will not run.

You can run Virtuoso Studio Design Environment software in graphics or nongraphics mode:

- When you start the software in graphics mode, the Command Interpreter Window (CIW) appears. From the CIW, you can start individual Cadence applications.
- In nongraphics mode, you can type Cadence SKILL commands or do any other work that does not require graphic display of designs or the graphical user interface. You can start Cadence software in nongraphics mode using the `-nograph` command-line option.

When Virtuoso has launched, the time taken to successfully checkout the 111 (Virtuoso Framework) license will be shown in the CIW, for example:

```
\o Virtuoso Framework License (111) was checked out successfully. Total checkout time
was 0.32s.
```

You can now go on to access some applications by selecting a menu item (such as from the *Tools* menu in the CIW) while other applications start automatically when you open a design cellview (using *File - Open*). For example, when you open a schematic, your schematic editor starts automatically.

For more information on using the Virtuoso Studio Design Environment, search for *Virtuoso Studio*

Design Environment User Guide on support.cadence.com.

1.8 Innovus and Virtuoso Release Compatibility Information

You can pair the following Innovus and Virtuoso releases for the Mixed Signal flow:

- INVS19.14 + ICADVM18.1ISR10 / IC618ISR10
- INVS19.15 + ICADVM18.1ISR12 / IC618ISR12
- INVS19.16 + ICADVM18.1ISR13 / IC618ISR13
- INVS19.17 + ICADVM20.1ISR15 / IC618ISR15
- INVS 20.10 + ICADVM18.1ISR9 / IC618ISR9
- INVS 20.11 + ICADVM18.1ISR12 / IC618ISR12
- INVS 20.12 + ICADVM18.1ISR13 / IC618ISR13
- INVS 20.13 + ICADVM20.1ISR15 / IC618ISR15
- INVS 20.14 + ICADVM20.1ISR17 / IC618ISR17
- INVS 20.15 + ICADVM20.1ISR19 / IC618ISR19
- INVS 20.16 + ICADVM20.1ISR21 / IC618ISR21
- INVS 21.10 + ICADVM20.1ISR16 / IC618ISR16
- INVS 21.11 + ICADVM20.1ISR19 / IC618ISR19
- INVS 21.12 + ICADVM20.1ISR21 / IC618ISR21
- INVS 21.13 + ICADVM20.1ISR23 / IC618ISR23
- INVS 21.14 + ICADVM20.1ISR25 / IC618ISR25
- INVS 21.15 + ICADVM20.1ISR27 / IC618ISR27
- INVS 21.16 + ICADVM20.1ISR29 / IC618ISR29
- INVS 21.17 + ICADVM20.1ISR30 / IC618ISR30
- INVS 21.18 + ICADVM20.1ISR33 / IC618ISR33
- INVS 22.10 + ICADVM20.1ISR26 / IC618ISR26

- INVS 22.11 + ICADVM20.1ISR29 / IC618ISR29
- INVS 22.12 + ICADVM20.1ISR31 / IC618ISR31
- INVS 22.13 + ICADVM20.1ISR33 / IC618ISR33

Note: IC61***x***ISR***y*** is same as IC6.1.***x***(-64b).500.***y***, returned by `virtuoso -W`.

2

Overview of Mixed Signal Interoperability

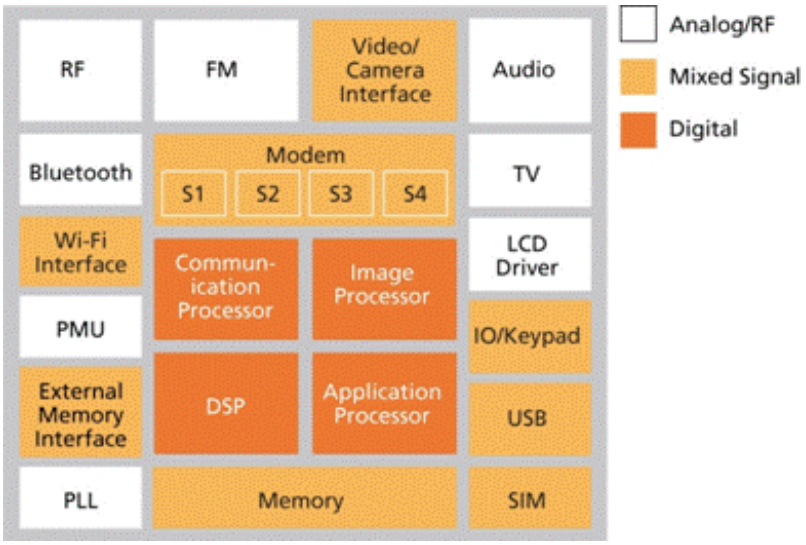
- Mixed Signal Solution - Introduction
  - Design Methodologies Supported by the Cadence Mixed Signal Solution

2.1 Mixed Signal Solution - Introduction

Success in today's electronics market place requires highly integrated and low-cost solutions for wireless, consumer, computer, and automotive applications. At the same time, advanced process nodes now make it possible to manufacture analog and RF circuits down to 45 nm and below. Consequently, analog and mixed signal IP content is significantly increasing in system-on-chip (SoC) devices that in the past contained mostly digital circuitry. This situation creates new challenges for design, integration, and verification.

Most SoCs currently being developed have analog or mixed signal blocks, such as SerDes cores, UARTs, DACS, ADCs, PLLs, and other transceivers. Since analog does not scale as well as digital, these blocks might represent a substantial portion of the SoC. Moreover, many so-called analog blocks actually have digital-control logic. As such, an increasing amount of analog IP is really mixed signal, and with rapidly increasing SoC capacity, a single IP block might represent an extremely complex mixed signal function.

The hypothetical SoC in the figure below shows analog blocks in white and digital blocks in orange. Light orange represents mixed signal blocks with both analog and digital circuitry. In earlier process generations, some of these mixed signal blocks might have been the entire chips.



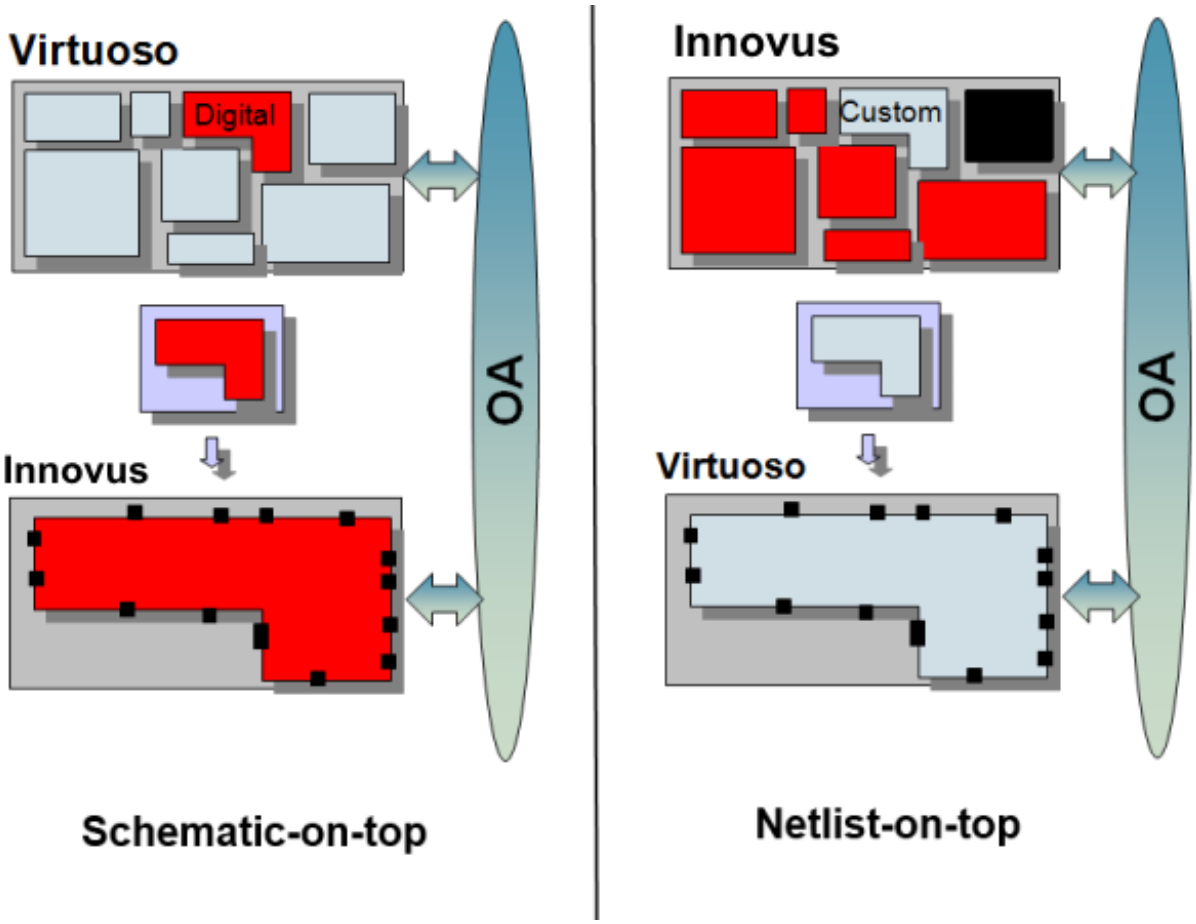
Currently, a sizable part of mixed signal chip implementation planning is done manually, which is a slow and laborious process that can lead to design errors and numerous iterations. During final assembly, the completed blocks are also placed and routed using a semi-manual process, without the aid of design rule-correct automation.

Big analog designs with digital blocks start with a schematic design in Virtuoso and then iterate over the floorplanning steps by manually placing the blocks as per the design architecture, and assigning pins for the blocks which could be analog as well as digital. Additionally, in the beginning of the design cycle, you just have the top-chip floorplan and do not have much insight into the design-implementation details.

2.1.1 Design Methodologies Supported by the Cadence Mixed Signal Solution

Schematic-driven mixed signal and netlist-driven mixed signal are two of the design methodologies supported by the Cadence mixed signal solution. Schematic-driven mixed signal and netlist-driven mixed signal are block-based methodologies sufficient for many designs when functionality can be contained in blocks with few analog-digital interfaces not critical for design performance. Schematic-driven mixed signal is a methodology used to implement mixed signal designs that have a large analog content and a small digital content (also known as big A little D - A/d). On the other hand, netlist-driven mixed signal is a methodology used to implement designs that have a large digital content and a smaller analog content (also known as big D little A - D/a).

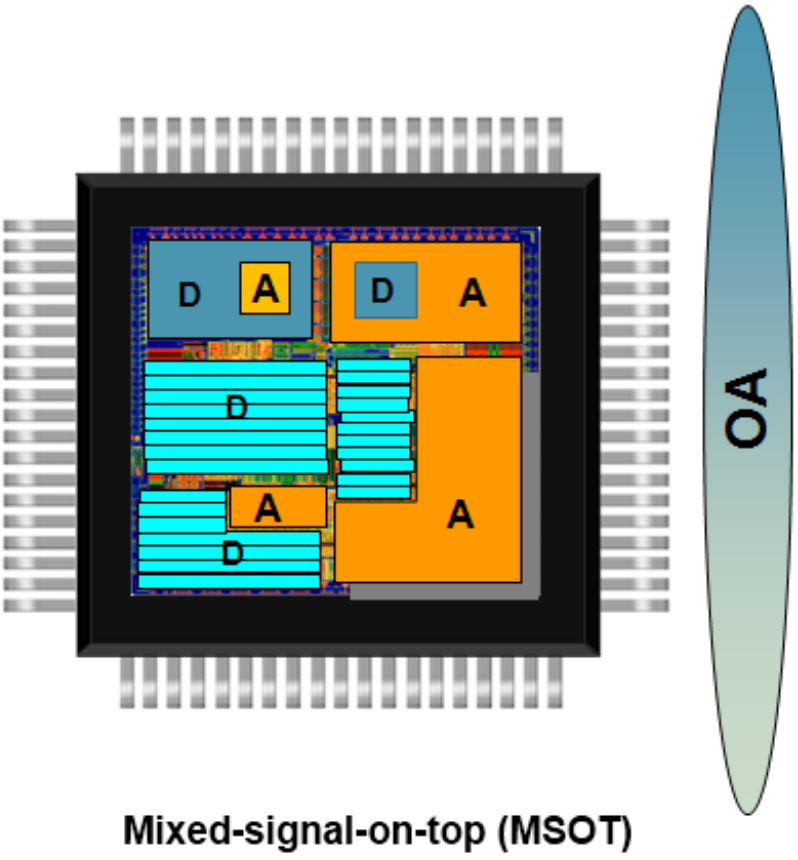
For the Schematic-driven mixed signal flow, the top level of a design is typically represented as schematics, while the Netlist-driven mixed signal's top level is typically represented as a Verilog netlist. That is the reason why some designers refer Schematic-driven mixed signal methodology as a the schematic-on-top methodology, and netlist-driven mixed signal as the netlist-on-top methodology.



In the Cadence environment, if schematics are used for representing the top level, the design cockpit of choice is generally the Virtuoso environment, However, if a Verilog netlist is used for the top-level design, the Innovus™ Implementation System software is generally used as the design cockpit.

Traditionally, both the schematic-driven mixed signal and netlist-driven mixed signal flows have partitioned out the part of a design that is targeted for implementation in a different cockpit. For example, in the case of a schematic-driven mixed signal flow, the top level of the design is predominantly analog and is represented as a schematic containing one or more smaller digital blocks. In this case, the digital content is taken to Innovus, and once implemented and verified, brought back and re-mastered in the top-level design in Virtuoso.

The recent trend towards the implementation of high performance mixed signal designs has created a requirement for supporting flows/methodologies that allow mixing of digital and analog content throughout the design hierarchy. The underlining OpenAccess infrastructure in the Cadence mixed signal solution makes it possible to support this design methodology through a flow that takes advantage of the functionality existing in either Innovus or Virtuoso cockpit, no matter whether the top-level design is represented as a schematic or a netlist. This methodology is referred to as concurrent mixed signal.



Mixed-signal-on-top (MSOT)

Cadence is addressing this challenge by introducing a new flow which draws on the strengths of the schematic-driven and netlist-driven mixed signal flows, and the interoperable Open Access database. This will allow users to easily migrate their entire design from one platform to another and take advantage of the unique features of each environment. For instance, in the new floorplanning flow, both analog and digital blocks can be floorplanned, much earlier in the design cycle when the details of these blocks are not available. These blocks are defined as blackboxes for the purpose of starting the design. The design is taken from one platform to another to enable all analysis steps, possible in the two platforms, thereby enabling users to make adjustment and more intelligent trade-offs. For example, after the top-level schematic is ready, you can run OpenAccess-based applications to generate Verilog for the top level, stitch the Verilog of the available modules, import in Innovus for blackbox-based high capacity, congestion, and timing-driven floorplanning. This helps them arrive at a higher quality block placement, pin locations, routing interface nets, and thereby lowering routing congestion. This new flow reduces the number of iterations between analog and digital designs, particularly during floorplanning, chip integration and early-and-late small/big ECOs.

3

Technology Data Preparation

- Software Requirements
- Technology Data Preparation
  - Library and Technology Requirements
  - Preparing the Technology Library
  - Determining Whether Your MSOA PDK Is Traditional or Rapid
  - Preparing the IP Library
  - Creating a Rapid MSOA PDK with Multiple foundry_innovus Constraint Groups
  - Preparing a Technology Library with Multiple LDRSs
  - Rapid PDK ASCII Techfile Structure

3.1 Software Requirements

The following installations are required to build the flow. There are older versions of Innovus and Virtuoso which have also been qualified for the mixed-signal flow.

- Innovus Implementation System 17.1 or later with IC 6.1.6 ISR6 or later

3.2 Technology Data Preparation

This section contains the following:

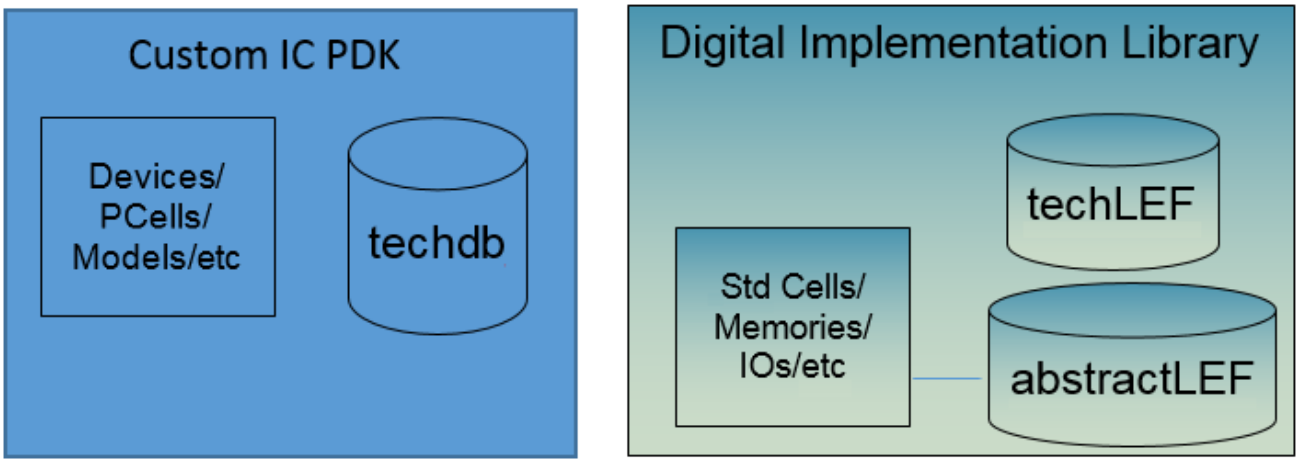
- Library and Technology Requirements
- Preparing the Technology Library
- Preparing the IP Library
- Preparing a Technology Library with Multiple LDRSs

3.2.1 Library and Technology Requirements

- Technology and IP data on OpenAccess 2.2 Data Model 4 format
- A common (interoperable) Process Design Kit (PDK), which will have all the necessary technology information for Innovus and Virtuoso. For instruction on how to create a common PDK, please refer to the Preparing the Technology Library section.
- Liberty timing library for standard cells and IP blocks and chip-level SDC file. If you do not have Liberty files and the chip-level SDC file, it will not be possible to perform static timing analysis of the top-level design.
- Extended FE capacitance table and QRC technology file to support extraction in Innovus.
- Power analysis libraries for enabling VoltageStorm analysis in Innovus.

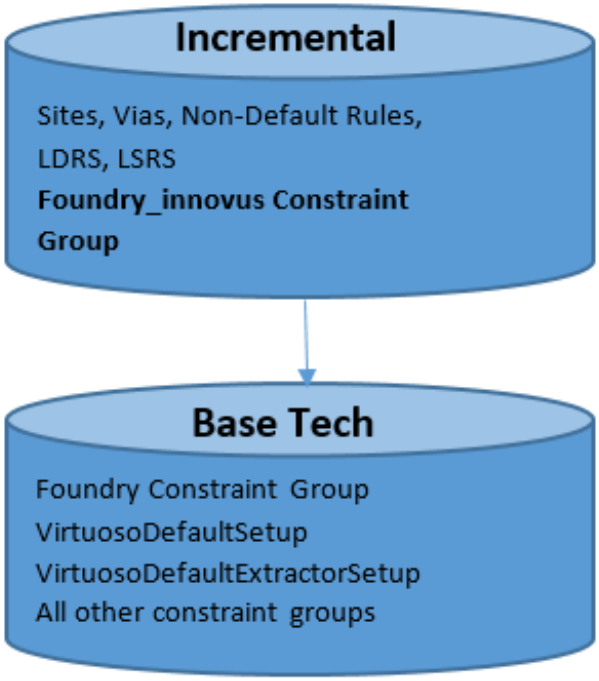
3.2.2 Preparing the Technology Library

Historically, most digital designs have relied on the use of a LEF file to specify the required technology information for the complete implementation of the design. These LEF technology files typically contain a complete set of rules for the metal and cut layers, and a very small subset of rules on the masterslice layers: `poly`, `well`, and `diffusion`.



The use of the OpenAccess-based interoperable mixed-signal flow requires the implementation of a common OpenAccess Process Design Kit (MSOA PDK). The MSOA PDK contains the information typically found in a base PDK in addition to that found in the technology LEF. The base PDK, which is the PDK used for Virtuoso, is expected to have at least the foundry rules for the masterslice layers and will typically include the foundry rules for all the metal and cut layers. The LEF would have unique information not present in the PDK: Routing-specific information, such as default routing rules for most nets and non-default rules for selected nets. Default and non-default rules must include the list of valid routing layers and vias. The LEF language requires that routing pitch and width should be included. The LEF file also has antenna checking rules and current density limits, which could be defined in the PDK but is typically not found in a base PDK.

The following diagram shows the desired structure that must be built using the custom IC PDK and the digital implementation library shown above.



3.2.2.1 Implementing an Interoperable/MSOA PDK

As previously mentioned, interoperable PDKs are required to take advantage of OpenAccess-based interoperability for mixed signal designs. Starting from the 18.1 release of Innovus, you now have multiple options for creating an MSOA PDK:

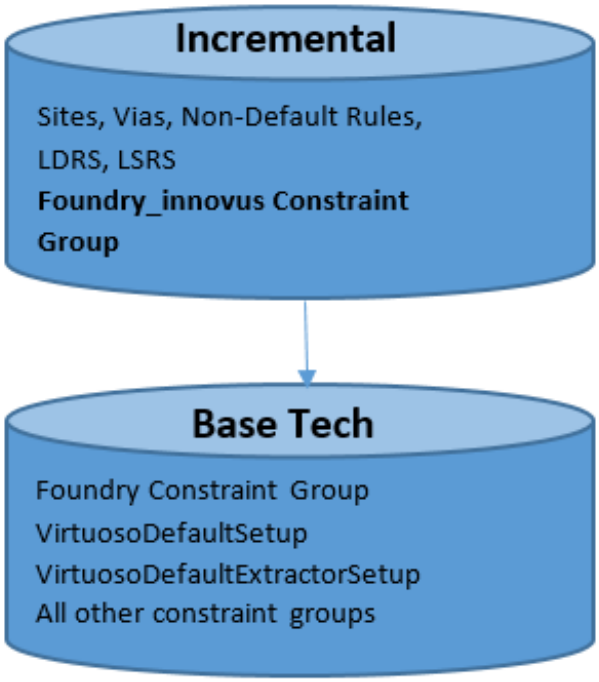
- Rapid MSOA PDK (Available in 18.1 and later releases)
- Traditional MSOA PDK (Available prior to 18.1 and described above, these MSOA PDKs are supported only for 28nm and older nodes. If you are implementing a new MSOA PDK, Cadence recommends the implementation of a Rapid MSOA PDK.)
- Innovus-only MSOA PDK (Useful for experimenting with Innovus in the OpenAccess mode)

As discussed earlier, the process of creating an MSOA PDK involves populating the tech file used by Virtuoso (base tech) with specific information that is typically only found in a tech LEF (used for the P&R environment).

Implementing a Rapid MSOA PDK

This is the easiest method for creating an MSOA PDK that could be used by Innovus and Virtuoso to implement mixed signal designs. The Rapid MSOA infrastructure removes the necessity for all DRC rules to be consistent between tech LEF and Virtuoso tech for a certain process. In other words, there is no need to change the Virtuoso tech file to align the DRC rules in it with those found in the tech LEF. Virtuoso obtains all the DRC rules from the base tech (the same tech used by Virtuoso in the past), while Innovus gets DRC rules from an ITDB layer that contains all the rules found in the tech LEF.

The main difference between the traditional MSOA PDK and the rapid MSOA PDK is the presence of a new foundry group in the incremental tech file, which will be used exclusively by Innovus for DRC rules.



The table below provide guidelines on the use of rapid MSOA PDKs:

	Rapid	**Traditional****
Same as LEF from an Innovus point of view?	Yes	Yes
Cellviews are interoperable with Virtuoso?	Yes	Yes

OpenAccess tech supports Design Rule Driven (DRD) design and Virtuoso Space-based Router (VSR) with the same rules as LEF?	No	Yes
Target processes?	New process nodes that are often changing	Process nodes that are stable
User who is expected to create the PDK?	End users who receive frequent LEF updates from their library vendor	Library vendors

**Traditional MSOA PDKs are covered in detail in the next section.

You can follow the steps outlined below to implement a rapid MSOA PDK:

1. Read LEF into Innovus using `init_design`:
   - A dummy/empty Verilog netlist can be used for this step
   - Dump out canonical LEF files using `write_lef_library -tech_only` and `-macro_only`
   - Files created: `TECH.lef` and `MACRO.lef`

```
set init_lef_file { <list_of_lef_files> }
set init_verilog {list_of_verilog_files_or_dummy}
set init_top_cell <top_cell_name>
init_design
write_lef_library -tech_only TECH.lef
write_lef_library -macro_only MACRO.lef
```

2. Determine if the `LAYER`/`CUTCLASS`/`NDR` names in the LEF are consistent with those in the base PDK. If not, use `sed` to create temp LEF file(s) that align to the `LAYER`/`CUTCLASS`/`NDR` names in the base PDK.
   Note: Details and examples for this step are provided later in this document.
3. Use `lef2oa -useFoundryInnovus` to process the `TECH.lef` (or the version created through the optional `sed` step).

   If Step 2 was not required, you can use the following command:
   `lef2oa -useFoundryInnovus -lef TECH.lef -lib RapidPDK -techRefs BasePDK`
   If you had to use `sed` to rename certain items in Step 2, use the following command (here, the `TECH_OA_NAMES.lef` has been created using `sed`):
   `lef2oa -useFoundryInnovus -lef TECH_OA_NAMES.lef -lib RapidPDK -techRefs BasePDK`

4. Use `lef2oa` to process the `MACRO.lef`:
   `lef2oa -lef MACRO.lef -lib RapidPDK` (If `sed` was not used)
   `lef2oa -lef MACRO_OA_NAMES.lef -lib RapidPDK` (If `sed` was used)
   While importing the standard cell LEFs, ensure that the standard cells library does not have any existing cells with the same names. `lef2oa` cannot override physical data in the standard cell abstract views by default, so you might see the following message:
   ```
   WARNING: (OALEFDEF-50057): stdcell.lef (107464-107522) : MACRO XXXXXX: The macro attempts to redefine obstructions, but updating physical data is not supported. The new obstructions were ignored. Use the -overwrite option to create the designs from scratch.
   ```
   If you want to recreate the abstracts for these cells, you need use either the `-overwrite` option of `lef2oa` or delete the cellview for these cells. However, note that it is possible to import the antenna information for an existing cell from a different LEF file by using `lef2oa`.
   In some process nodes, the standard cells that are electrically equivalent (EEQ) come in multiple macro LEF files. For example, the `macroB.lef` file has a `OR_SIZE1A_H100` standard cell that has the following EEQ statement:
   ```
   EQ OR_SIZE1_H100 ;
   ```
   while the cell `OR_SIZE1_H100` is in the `macroA.lef` file.
   To convert these macro LEFs into abstract, concatenate the two macro LEF files and then use `lef2oa` to read the concatenated file:
   ```
   cat macroA.lef macroB.lef > macroAB.lef
   ```
   `lef2oa -lef macroAB.lef -lib RapidPDK` (If `sed` was not used)
   Otherwise, if you use `lef2oa` to convert the `macroB.lef` file directly (`lef2oa -lef macroB.lef -lib RapidPDK`), `lef2oa` will fail and issue the following message when reading `macroB.lef`:
   ```
   ERROR: (OALEFDEF-50055): macroB(74-75) : MACRO OR_SIZE1A_H100: The macro requires that the EEQ master cell OR_SIZE1_H100 be defined before macro use. This attribute was ignored.
   ```
5. Read the OpenAccess library data (from Steps 3 and 4) into Innovus by using `init_design`:
   - A dummy/empty Verilog netlist can be used for this step
   - Dump out canonical LEF files using `write_lef_library -tech_only` and `-macro_only`
   - Files created: `TECH_RapidPDK.lef` and `MACRO_RapidPDK.lef`

```
set init_oa_ref_lib { RapidPDK }
set init_verilog { <list_of_verilog_files_or_dummy> }
set init_top_cell <top_cell_name>
init_design
write_lef_library -tech_only TECH_RapidPDK.lef
write_lef_library -macro_only MACRO_RapidPDK.lef
```

6. Compare the LEF files from Step 5 against the ones from Step 1 or 2. The only differences should be that additional layers from the base PDK that were not in the original LEF will be present:
    - If Step 2 was required, compare to the files from that step instead of Step 1. Use:
      ```
      diff TECH.lef TECH_RapidPDK.lef > TECH_RapidPDK.diff
      diff MACRO.lef MACRO_RapidPDK.lef > MACRO_RapidPDK.diff
      ```
    - If Step 2 was required, use:
      ```
      diff TECH_OA_NAMES.lef TECH_RapidPDK.lef > TECH_RapidPDK.diff
      diff MACRO_OA_NAMES.lef MACRO_RapidPDK.lef > MACRO_RapidPDK.diff
      ```

    Except the additional layers from the base PDK, there should be no differences, which indicates that Innovus should be able to produce identical results as compared to the flows using the the original LEFs.

As you can see, Steps 1-4 are for creating the rapid PDK, and Steps 5 and 6 are for checking the rapid PDK.

Details of Step 2 (using `sed` to create temp LEF file(s) that align to the `LAYER`/`CUTCLASS`/`NDR` names in the base PDK):

Simple `sed` syntax is used to make sure that only complete names are mapped. For example, `M1` should be mapped to `METAL1`, but `CM1` should not be affected by that mapping. The `sed` approach relies on the white space separated style that is used by the LEF/DEF syntax. It maps any complete name, so it will affect the names inside the property string values as well.

There are some limitation to the use of `sed`:

- If the LEF file contains names that need to be mapped that are also keywords, then extra processing is required.
  Example: Mapping a layer called `PWELL` to `PW` would affect the `TYPE PWELL` as well.
- If the LEF and OpenAccess names are overlapping in some way:
  Example 1: LEF has `M0`, `M1`, `M2`, ..., `M8` and OpenAccess has layers `M1`, `M2`, `M3`..., `M9`
  To handle this case, the order of statements in the `sed` file is important to prevent the layer `M0` from becoming `M1`, then `M2`, then `M3`, and so on, as the `sed` file is processed (which happens sequentially). This is an unlikely perverse case, but is still easy to handle.

  Example 2: LEF has CUTCLASS names `SQ` and `RE`, while the respective names in OpenAccess are `VSINGLECUT` and `VDOUBLECUT`. In addition, LEF has layers `M1` and `VIA1`, while the respective OpenAccess layers are called `METAL1` and `VIA12` as shown in the example below.

Example:

```
s/\<M1\>/METAL1/g
s/\<VIA1\>/VIA12/g
s/\<RE\>/VDOUBLECUT/g
s/\<SQ\>/VSINGLECUT/g
```

In this case, the following syntax shoud be used:

```
s/\<lef_name\>/oa_name/g
```

The `\<` and `\>` are used to prevent partial matching.

To additionally handle the case where layer `PWELL` needs to be mapped to `PW`, the following syntax is required:

(*lef_names_to_oa_names*.sed)

```
s/\<TYPE PWELL\>/TYPE PWELL_temp/g

s/\<PWELL\>/PW/g

s/\<M1\>/METAL1/g

s/\<VIA1\>/VIA12/g

s/\<RE\>/VDOUBLECUT/g

s/\<SQ\>/VSINGLECUT/g

s/\<TYPE PWELL_temp\>/TYPE PWELL/g
```

Using `sed` to map OpenAccess names to LEF names:

To additionally handle the case where layer `PW OR` needs to be mapped to `PWELL` in LEF, the following syntax is required:

(oa_names_to_lef_names.sed)

```
s/\<TYPE PWELL_temp\>/TYPE PWELL/g

s/\<PW\>/PWELL/g

s/\<METAL1\>/M1/g

s/\<VIA12\>/VIA1/g

s/\<VDOUBLECUT\>/RE/G

s/\<VSINGLECUT\>/SQ/g

s/\<TYPE PWELL_temp\>/TYPE PWELL/g
```

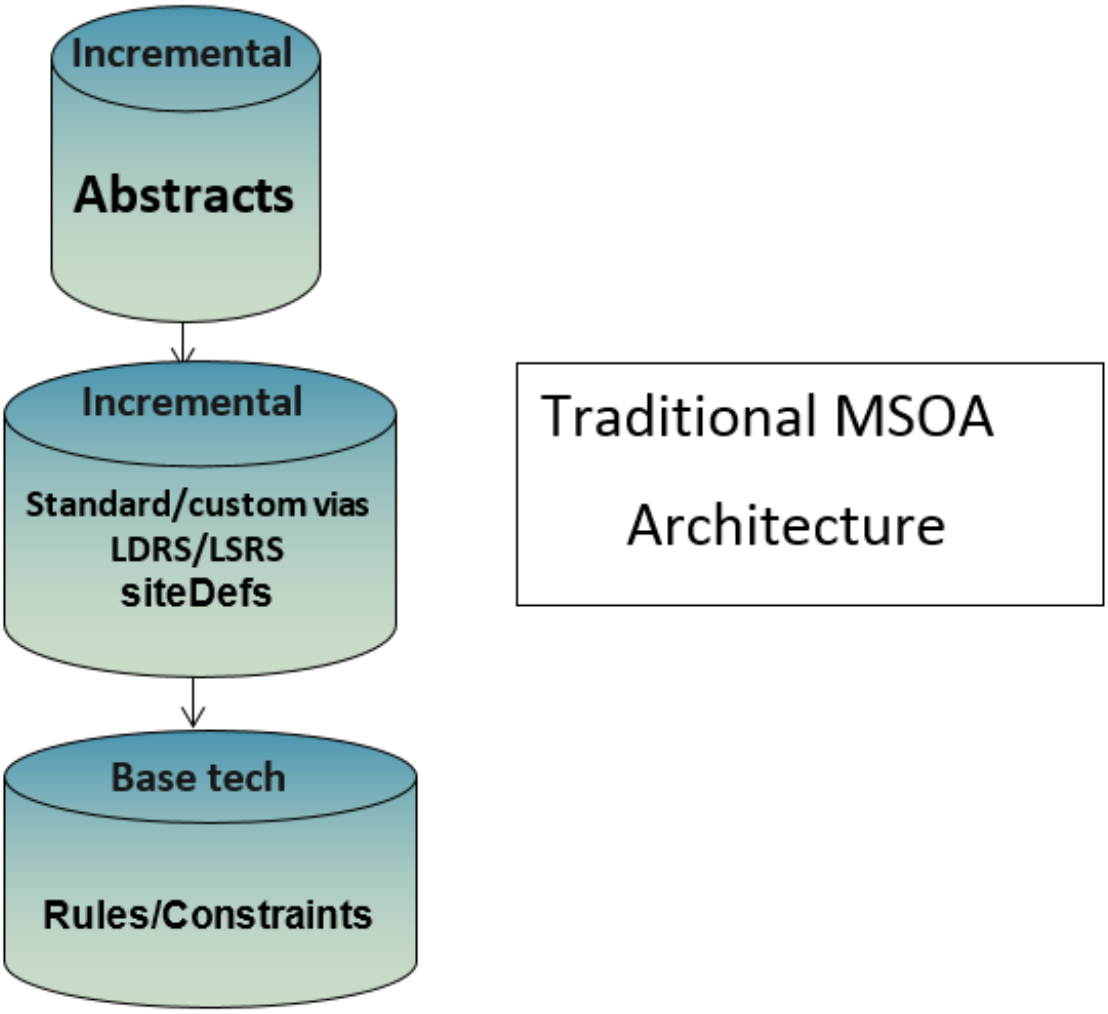
The lines in Italic font are not actually required because the reverse mapping does not have the collision in this specific case.

The following are not covered by `sed`-based mapping:

- Special case mapping for LEF and OpenAccess semantic differences, such as:
  - LEF `CM1 MASK1` becoming `CM1A` in OpenAccess
  - `R1` (region layer) handling
  - `*_P48` marker layer support

  Existing OpenAccess/OAX support is still required for these (OALAYERMAP techParam, and so on)
- There is no scriptable workaround for the case where the same `CUTCLASS` name is used for different layers, but the OpenAccess side has different names for each layer. The names need to be manually synced up in this case.

Creating a Traditional MSOA PDK



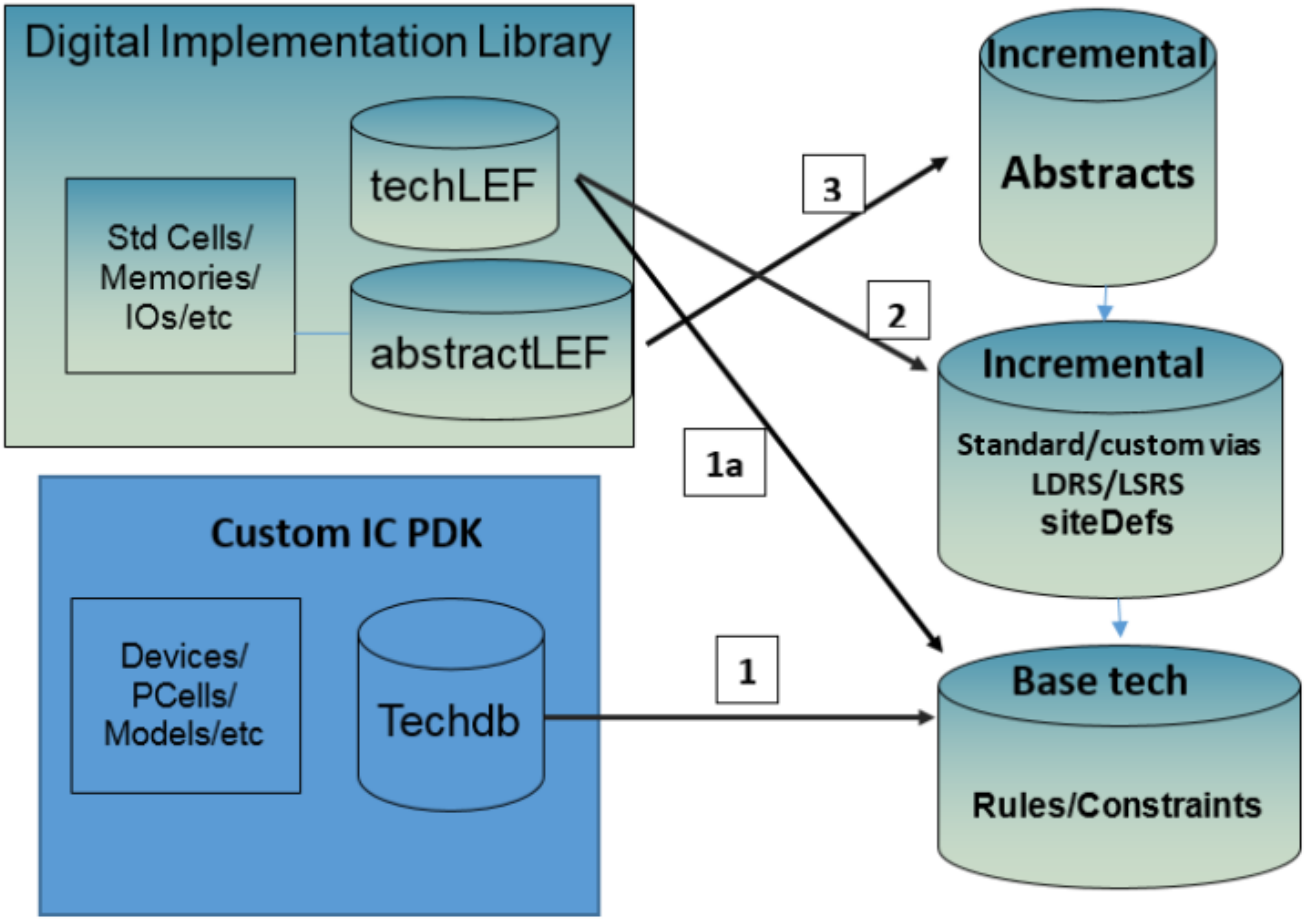
In the traditional MSOA architecture, the base library contains all the rules and constraints required to run the Virtuoso implementation environment. The same DRC rules will be used by Innovus, when it uses this technology library. An incremental technology library is created, which contains only the information needed to run the place and route (P&R) environment, such as any custom vias, the LefDefaultRouteSpec (LDRS) and the LefSpecialRouteSpec(LSRS), and the site definition for standard cells in the library. In order to run the P&R environment, Innovus uses abstracts of the standard cells. The standard cell OpenAccess abstracts are the additional incremental technology layer in the library.

Given that Innovus and Virtuoso will obtain technology information from the "Base tech" in the above diagram, there might be a need to reconcile any difference between the tech LEF and the tech DB prior to building the library illustrated above. The resulting MSOA PDK is referred to as a “Traditional MSOA PDK”.

It is recommended that you seek assistance from Cadence when creating MSOA PDKs for use in the Mixed Signal flow. The PDK factory team at Cadence is available to provide assistance and answer any questions you might have in creating and testing such PDKs. The process for creating traditional MSOA PDKs differs slightly based on the technology node for which the MSOA PDK is being implemented.

Creating a traditional MSOA PDK for process nodes 40nm or older

The process of creating a traditional MSOA PDK involves transferring data from the tech LEF and the Virtuoso tech to the MSOA PDK by following the arrows in the diagram below:



The numbers in the above diagram represent the sequence of actions needed to implement the traditional MSOA PDK.

For the above process to work well, you might be required to make updates to the following items in the Virtuoso technology file to reconcile any differences between the tech LEF and the Virtuoso tech file: Layer name and purposes, LPPs, capacitance and resistance information, current density information (if existing in Virtuoso tech), and advanced routing constraints. This is denoted as Step

1 and Step 1a in the above diagram.

Note that many foundries have started delivering Virtuoso tech files that are ready to go through Steps 2 and 3. You should contact the particular foundry to ask for an Innovus-ready Virtuoso tech file.

This method uses the `lef2oa` command in the Innovus hierarchy, and is targeted only at process nodes 40nm or older. Following is an example of how this command can be used to create an MSOA PDK:

```
lef2oa -lef  <name_of_the_LEF_technology_file> -lib <the_MSOA_PDK_to_be_created>  -techRefs <the_base_PDK> -pnrLibDataOnly
```

The above step extracts routing information, such as metal pitch, site definition, NDRs, vias, and so on, from the tech LEF and populates the information in the resulting MSOA PDK. In the diagram above, it is represented as Step 2.

Now you need to create an Incremental Technology Database (ITDB) structure for the standard cells and any macro libraries to be used with the MSOA PDK. This is also done using the `lef2oa` command as shown below:

```
lef2oa -lib  <the_OA_stdcell_lib_to_be_created> -lef  stdcell.lef  -techRefs <the_MSOA_PDK_created_from_the_previous_step>
```

```
lef2oa -lib <the_marco_library_to_be_created> -lef  macro/lef -techRefs <the_MSOA_PDK_created_from_the_previous_step>
```

Creating a traditional MSOA PDK for process nodes 28nm and below

The second method is to be used for 28nm and below. It involves the use of the `write_oa_techfile` command in Innovus to dump out an equivalent OpenAccess representation of the technology information found in the LEF technology file:

```
write_oa_techfile  <filename>
```

After that is done, you can manually add the LEF information to the existing base PDK.

It is common to see a particular foundry update their existing 40nm or older LEF technology files to include rules that are typically found in newer processes. Pay special attention to warning and error messages when using the `lef2oa` command. If messages such as the one below are reported, you are advised to use the `write_oa_techfile` method to implement your interoperable MSOA PDK.

```
INFO: (OALEFDEF-XXXX): <path to the LEF file>(<line numbers in the LEF file>): A LEF58 property named 'XXXXXX' with value 'yyyy ;' was found in this line range. This property syntax is not supported by the LEF parser. This property string is kept, but is not converted into an OA rule constraint.
```

As can be seen from the above message, there will be a loss in the translation process to bring in the particular LEF information into the OpenAccess PDK.

Creating an Innovus-only MSOA PDK

Very often, users who have previously used Innovus in the LEF/DEF mode wish to try running the tool in the OpenAccess mode to better understand how the data is stored when Innovus is run in the OpenAccess mode. When Innovus is run in OpenAccess, the difference in the flow is basically in the initialization steps of the flow; the actual flow scripts that go through all the steps in the flow are identical to the LEF/DEF flow. In the OpenAccess mode, Innovus saves the data for the design into the same lib/cell/view structure that is used by Virtuoso to save the design data. This provides the end user with a unique interoperable environment, where a particular cell view could be opened in either Virtuoso or Innovus, edited, and the edits made available to the other tool.

If the end user is not interested in interoperating the design data between Innovus and Virtuoso and would only like to run Innovus in the OpenAccess mode, the MSOA interoperable PDK could be created using only the tech LEF data.

The detailed process for implementing an MSOA PDK for use only by Innovus is as follows:

1. Invoke EDI or Innovus with any design that uses the LEF technology information in which you are interested. Then, invoke `write_oa_techfile`. This command writes out a DFII style ASCII technology file that can later be given to the `techLoadDump` executable to create an equivalent OpenAccess technology database.
2. In the same EDI/Innovus session, run the following command:
   `write_lef_library -tech_only`
   This command creates a LEF file that contains only the LEF technology information. This file will be used in later tasks in this process.
3. In the same EDI/Innovus session, run the following command:
   `write_lef_library -macro_only`
   This command will write out the LEF information for macros in the LEF library. This information will get used by later tasks in this process.
4. The next command is available in the Virtuoso hierarchy. Ensure that you have access to the installation you want to use for this step. Invoke the following command with the file created from Step 1:
   `techLoadDump -l -createLib <`*`library_name`*`> <`*`tech_file_from_Step1`*`>`.
   This command will create an OA library with LEF layers and foundry rules, from the ASCII technology file that was created by the `write_oa_techfile` command.
   Note: For advanced node processes (14nm and below), the `techLoadDump` utility from a Virtuoso Advanced Node (ICADV*) Release must be used.

You can use the `techLoadDump` command from the UNIX prompt for both loading or dumping a technology library.

```
$ which techLoadDump
/INSTALL/IC618ISR/tools/dfII/bin/techLoadDump

$ techLoadDump
Virtuoso Framework License (111) was checked out successfully. Total
checkout time was 0.09s.
Usage: techLoadDump [ -l | -d ] [-createLib [-libPath libPath]] techLibName
techFileName
```

For creating a technology library, use the `-l` parameter.

For dumping a techlib, use the `-d` parameter.

You can also load or dump a tech file for a given library by using the *Tools -> Technology File Manager* command from CIW in Virtuoso. This opens the Techology File Manager form, which provides options to load or dump a technology file.

5. The next command is available in the EDI/Innovus installation hierarchy. You will be using the output of Step 2 as input for this command. Invoke:

   `lef2oa -pnrLibDataOnly -lef <file_from_Step2> -techRefs <library_created_from_Step4> -lib <library_name>`.

   This command maps only library-specific place and route information to an incremental tech, ignoring foundry rules. The incremental technology database references a technology database that has complete foundry information. LEF constructs DIRECTION and PITCH, along with WIDTH, OFFSET, and WIREEXTENSION (if specified) are mapped to constraints in the `LEFDefaultRouteSpec` (LDRS) type constraint group. VIAS and VIARULES defined are added to the validRoutingVias constraint in the `LEFDefaultRouteSpec`. NONDEFAULTRULES are mapped to OpenAccess constraint groups, SITES are mapped to oaSiteDefs, and MACROS are mapped to abstract oaDesigns. For more information on this command, refer to the *Innovus Text Command Reference* manual.

6. Now that the library has all the rules, the abstracts need to be added to the library to complete the process. In EDI/Innovus, invoke the following command:

   `lef2oa -lef <file_from_step3> -lib <library_created_in_Step5>`

3.2.2.2 Recommended Method for Checking the Integrity of the Common/Interoperable PDK for Innovus

After you have implemented an interoperable PDK, it is recommended that you perform the following steps to ensure that the LEF technology related information contained in the PDK is the same as the information contained in the initial LEF file that was used to create the interoperable PDK. The verification method involves writing the LEF related information out of the PDK and comparing it to the initial LEF. The steps are as follows:

1. Start Innovus with the interoperable OA PDK, which contains the LEF technology information.
2. Use the Innovus command `write_lef_library` to output a LEF file from the interoperable OA PDK. Note that this command outputs a LEF directly from the Innovus DB, and is an exact representation of how the LEF technology data is seen by Innovus.
3. Start Innovus again with the original LEF that was used to make the interoperable PDK.
4. Use the Innovus command `write_lef_library` to output a LEF file.
5. As the output of `write_lef_library` is in a canonical format, you can now use the Unix command `tkdiff` to compare the two.

You can also dump out the PDK technology into an ASCII file from Virtuoso to check if all the rules names and values are defined correctly in the appropriate sections. This will enable you to check the mapping of the rules from the LEF file to their corresponding OpenAccess structure. To do so:

1. In the Virtuoso Command Interpreter Window (CIW), choose *Tools -> Technology File Manager -> Dump*.
2. From the *Technology Library* drop-down list, select the interoperable PDK and then select the *Select All* check box to indicate you want to dump out all rules.
3. Specify a name for the file in the *ASCII Technology File* text box and click *OK*.

For more information on the ASCII techfile syntax and the Rapid PDK ASCII syntax in particular, refer to the Rapid PDK ASCII Techfile Structure section.

3.2.3 Determining Whether Your MSOA PDK Is Traditional or Rapid

The Innovus command `report_oa_lib` can be used to determine the type of MSOA PDK technology library being used in the Innovus session.

```
innovus> report_oa_lib <libName>
```

The output of this command will be similar to the following:

```
Library Name : FEOAreflib cdsinto.tag exists : No Compression Level : 0 Rapid PDK : Yes
Constraint Groups : 6 Library Name: FEOAreflib LEFDefaultRouteSpec
(LEFDefaultRouteSpec, default) validLayers: M1 M2 M3 M4 M5 M6 AP validVias: VIAGEN12
VIAGEN23 VIAGEN34 VIAGEN45 minSpacing minNumCut minProtrusionNumCut
oaMinViaSpacing_FEOAreflib LEFSpecialRouteSpec Technology Graph: FEOAreflib
```

If the MSOA PDK is a Rapid MSOA PDK, the report will contain the following line:

```
Rapid PDK             : Yes
```

On the other hand, if the MSOA PDK you are using is traditional, the report will contain the following line:

```
Rapid PDK : No
```

3.2.4 Preparing the IP Library

1. For opening a design in Innovus, ensure that the IP library contains the abstract views of all IP blocks used at the top level in the design.

   While opening an existing OpenAccess design database, Innovus uses the `init_oa_ref_lib` list and the `init_abstract_view` variable values given in the configuration file to read the existing abstract view.

2. If the `SYMMETRY` attribute is not set, you can set the symmetry of a cell using the following SKILL commands in the CIW window of Virtuoso XL:

   - `cv = geGetWindowCellView(hiGetCurrentWindow())`
   - `dbSetCellViewSymmetry(cv "XYR90")`
   - `dbGetCellViewSymmetry(cv)`

3. If you require the analog property, add the analog property `oacSigTypeAnalog` on analog nets so that Innovus maintains such nets as `dbIsAnalog true`. This is an optional step.
   To add the analog property (signal type) in Virtuoso, follow these steps:

   a. Right-click the top-most toolbar to invoke the Navigator and the Property Editor.
   b. Expand Navigator Nets to view all the nets.
   c. Select the net for which you want to change the signal type.
   d. Save the design.

4. To generate abstracts for analog IPs that are finally used by Innovus, use Virtuoso Abstract Generator. Virtuoso Abstract Generator preserves the `oacSigTypeAnalog` on terminal nets in the OpenAccess database.
5. Save the OpenAccess database of IP abstracts to be used by the digital toolsets. Use verilogAnnotate, a stand-alone UNIX utility, to annotate the bus terminal list and order the distributed terms in the physical abstract. The verilogAnnotate utility can be found in the Virtuoso installation hierarchy.
   Run the following command:
   ```
   verilogAnnotate -refLibs <> -verilog <>
   ```
   If using Virtuoso Layout XL, use the following settings instead of the `verilogAnnotate` command:
   ```
   envSetVal("layoutXL" "createImplicitBusTerminals" 'boolen t)
   ```
   When you create abstracts in Virtuoso Abstract Generator, use:
   ```
   absSetOption( "AnnotateBusInAbstract" "true")
   ```
   Use `verilogAnnotate` only if you have some legacy IPs that cannot not be modified by any other way.
6. Unset the environment variable `OA_HOME`. Ensure that the `OA_HOME` environment variable is not set before running Virtuoso XL/GXL or Innovus. Source the Virtuoso path setting file so that you can get Virtuoso XL/GXL in your `PATH` environment variable.
7. Follow these steps to open the design with parameterized cell (PCell) in Virtuoso XL/GXL, and regenerate the PCell cache:
   a. Choose *Tools-Express Pcell Manager*.
   b. Enter the details and enable caching of *Pcell* check box using the *Auto Save* option.
   c. Click *Save Copy* to save the PCell layout cache.
      This step enables interoperation of data between Innovus and Virtuoso platforms. Close the layout window and purge your data from Virtuoso so that the Virtuoso file lock is released. For more information, see the *Virtuoso XL/GXL User Guide*.

3.2.5 Creating a Rapid MSOA PDK with Multiple foundry_innovus Constraint Groups

A foundry may release multiple tech LEF files corresponding to the same Virtuoso PDK. In the past, you had to create a Rapid MSOA PDK for each tech LEF. The MSOA PDK creation process has been enhanced in 21.13 release of Innovus to allow for the creation of a single Rapid MSOA PDK, containing multiple `foundry_innovus` constraint groups, each corresponding to a tech LEF.

ⓘ This process is not recommended if the tech LEF files have any of the following conditions:

- Different siteDefs with the same name exist in the tech LEF files.
- Vias with the same name, but different geometries, exist in the tech LEFs.
- LEF NDRs with the same name, but different width, spacing, or other attributes, exist in the tech LEFs.
- Cut classes with the same name, but different sizes, exist in the tech LEFs.

The process for creating such a Rapid MSOA PDK is as follows:

1. Create the base technology library, if it does not already exist, from the base ASCII tech file using either the Technology File Manager in Virtuoso or the `techLoadDump` executable. Refer to the Creating an Innovus-only MSOA PDK section for details of this process.
2. Run the following `lef2oa` commands to create a multi-tech Rapid PDK from `techLEF_6` and `techLEF_9`. This step can be extended to any number of LEF files.

   a. `lef2oa -lef techLEF_6.lef -lib myfoundry_innovus -techRefs myfoundry -logFile lef2oa_run1.log -useFoundryInnovus -suffix _6`
   This first `lef2oa` command uses `-techRefs` to reference the base PDK; any subsequent `lef2oa` runs do not need the `-techRefs` option. The above `lef2oa` command creates the first `foundry_innovus` constraint group with the name `foundry_innovus_6`.

   b. `lef2oa -lef techLEF_9.lef -lib myfoundry_innovus -logFile lef2oa_run2.log -useFoundryInnovus -suffix _9`
   The above `lef2oa` command creates a second `foundry_innovus` constraint group with the name `foundry_innovus_9`. This constraint group corresponds to `techLEF_9`.

   The previous two steps create a Rapid PDK `myfoundry_innovus`, which has the respective constraint groups named with the specified suffix `foundry_innovus_6`,

`foundry_innovus_9`, `LEFDefaultRouteSpec_6` and so on.

3. Convert the standard-cell and macro LEFs to OpenAccess while referencing to the Rapid PDK created in the above step. For more details on this step, see the Implementing a Rapid MSOA PDK section.
4. Set the following commands in Innovus to read the corresponding tech from the multi-tech Rapid PDK. For example, to read `foundry_innovus_6` from the Rapid MSOA PDK, set the following commands while importing the design into Innovus. Similarly, for `foundry_innovus_9`, replace `_6` with the `_9`.
   a. `set init_oa_default_rule LEFDefaultRouteSpec_6`
   b. `set init_oa_foundry_rule foundry_innovus_6`
   c. `set init_oa_special_rule LEFSpecialRouteSpec_6`
5. Check the library with `report_oa_lib` to make sure that you can see all the `foundry_innovus` constraint groups that needed to be created.
6. Validate using the `write_lef_library` output to check if the corresponding technology information is read correctly.

ⓘ While creating a multi-tech Rapid MSOA PDK, the ITDB may have multiple cut classes, not all of which will be required for a single tech set. Starting from the 21.17 release, a new constraint `cutClassNames` is added in each `foundry_innovus_<suffix>` to store the sizes of the cutClasses needed for a specific layer. Only the cutClasses specified in this constraint will be read into Innovus.

3.2.6 Preparing a Technology Library with Multiple LDRSs

⚠ This process will work only for a traditional MSOA PDK. You should not follow these instructions if the MSOA PDK being created is a Rapid MSOA PDK.

As previously discussed in this document, a LEFDefaultRouteSpec (LDRS) is a constraint group that gets added to the Virtuoso tech file for use by Place and Route tools, such as the Virtuoso Space Based Router, the Innovus Placement Engine, and the Innovus Router NanoRoute. The LDRS contains the same information as that can be found in a tech LEF file. The LDRS can contain information on routing layers, default vias, pitch, routing direction and width, layer offset, and wire

extension rule. The rest of the rules are read from the foundry constraint group in the tech.

In some cases, you may wish to have one tech file with multiple LDRSs. For example, you may want certain designs to use different widths or routing directions for optimal results while all other designs follow the regular LDRS. In this case, the original LDRS and the modified LDRS could both exist in the tech file. When the tech file is used with Innovus, you can instruct Innovus to pick the correct LDRS by using the following setting:

```
init_oa_default_rule    ruleName
```

Note that the second LDRS should be given a unique name because that is what tells Innovus which LDRS to choose for a particular Innovus session. The tech file needs to be edited to add the second LDRS. If the contents of the tech file is dumped to ASCII, the LDRS will look similar to this example:

```
;*******************************
; CONSTRAINT GROUPS
;*******************************
constraintGroups(

;( group [override] [definition] [operator] )
;( ----- ---------- ------------ ---------- )
( "LEFSpecialRouteSpec"                     nil           "LEFSpecialRouteSpec"

interconnect(
( validVias (M2_M1 M3_M2 M4_M3 M5_M4 M6_M5 M7_M6 M8_M7 M9_M8 M10_M9 M11_M10 ) )
) ;interconnect
) ;LEFSpecialRouteSpec

;( group [override] [definition] [operator] )
;( ----- ---------- ------------ ---------- )
( "LEFDefaultRouteSpec"             nil                 "LEFDefaultRouteSpec"

routingDirections(
( Poly "none" )
( Metal1 "horizontal" )
( Metal2 "vertical" )
( Metal3 "horizontal" )
( Metal4 "vertical" )
( Metal5 "horizontal" )
( Metal6 "vertical" )
( Metal7 "horizontal" )
( Metal8 "vertical" )
( Metal9 "horizontal" )
( Metal10 "vertical" )
( Metal11 "horizontal" )
```

```
) ;routingDirections

spacings(
( minWidth "Cont" 0.06 )
) ;spacings

routingGrids(
( horizontalPitch "Metal1" 0.2 )
( verticalPitch "Metal1" 0.19 )
( horizontalOffset "Metal1" 0.1 )
( verticalOffset "Metal1" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal1" 0.06 )
( minWidth "Via1" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal2" 0.2 )
( verticalPitch "Metal2" 0.19 )
( horizontalOffset "Metal2" 0.1 )
( verticalOffset "Metal2" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal2" 0.08 )
( minWidth "Via2" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal3" 0.2 )
( verticalPitch "Metal3" 0.19 )
( horizontalOffset "Metal3" 0.1 )
( verticalOffset "Metal3" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal3" 0.08 )
( minWidth "Via3" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal4" 0.2 )
( verticalPitch "Metal4" 0.19 )
( horizontalOffset "Metal4" 0.1 )
( verticalOffset "Metal4" 0.095 )
```

```
) ;routingGrids

spacings(
( minWidth "Metal4" 0.08 )
( minWidth "Via4" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal5" 0.2 )
( verticalPitch "Metal5" 0.19 )
( horizontalOffset "Metal5" 0.1 )
( verticalOffset "Metal5" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal5" 0.08 )
( minWidth "Via5" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal6" 0.2 )
( verticalPitch "Metal6" 0.19 )
( horizontalOffset "Metal6" 0.1 )
( verticalOffset "Metal6" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal6" 0.08 )
( minWidth "Via6" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal7" 0.2 )
( verticalPitch "Metal7" 0.19 )
( horizontalOffset "Metal7" 0.1 )
( verticalOffset "Metal7" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal7" 0.08 )
( minWidth "Via7" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal8" 0.2 )
( verticalPitch "Metal8" 0.19 )
```

```
( horizontalOffset "Metal8" 0.1 )
( verticalOffset "Metal8" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal8" 0.08 )
( minWidth "Via8" 0.07 )
) ;spacings

routingGrids(
( horizontalPitch "Metal9" 0.2 )
( verticalPitch "Metal9" 0.19 )
( horizontalOffset "Metal9" 0.1 )
( verticalOffset "Metal9" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal9" 0.08 )
( minWidth "Via9" 0.18 )
) ;spacings

routingGrids(
( horizontalPitch "Metal10" 0.5 )
( verticalPitch "Metal10" 0.19 )
( horizontalOffset "Metal10" 0.6 )
( verticalOffset "Metal10" 0.095 )
) ;routingGrids

spacings(
( minWidth "Metal10" 0.22 )
( minWidth "Via10" 0.18 )
) ;spacings

routingGrids(
( horizontalPitch "Metal11" 0.5 )
( verticalPitch "Metal11" 0.475 )
( horizontalOffset "Metal11" 0.6 )
( verticalOffset "Metal11" 0.57 )
) ;routingGrids

spacings(
( minWidth "Metal11" 0.22 )
) ;spacings

interconnect(
( validLayers (Metal1 Metal2 Metal3 Metal4 Metal5 Metal6 Metal7 Metal8 Metal9 Metal10
Metal11 ) )
```

```
( validVias (M2_M1_HV M2_M1_VV M2_M1_VH M2_M1_HH M2_M1_2x1_HV_E M2_M1_2x1_HV_W
M2_M1_1x2_HV_N M2_M1_1x2_HV_S M3_M2_VH M3_M2_HH M3_M2_HV M3_M2_VV M3_M2_M_NH M3_M2_M_SH
M3_M2_2x1_VH_E M3_M2_2x1_VH_W M3_M2_1x2_VH_N M3_M2_1x2_VH_S M4_M3_HV  ) )
) ;interconnect
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

If the user wishes to add a second LDRS, the ASCII tech file needs to be edited to include the name of the additional LDRS and any specific data that is typically found in the LDRS. In the example below, `LEFDefaultRouteSpec_3H` is a new LDRS added to an existing tech that already contains a default LDRS.

```
;*******************************
; CONSTRAINT GROUPS
;*******************************
constraintGroups(

;( group [override] [definition] [operator] )
;( ----- ---------- ------------ ---------- )
( "LEFSpecialRouteSpec"                    nil                   "LEFSpecialRouteSpec"

interconnect(
( validVias (M2_M1 M3_M2 M4_M3 M5_M4 M6_M5 M7_M6 M8_M7 M9_M8 M10_M9 M11_M10 ) )
) ;interconnect
) ;LEFSpecialRouteSpec

;( group [override] [definition] [operator] )
;( ----- ---------- ------------ ---------- )
( "LEFDefaultRouteSpec"                    nil              "LEFDefaultRouteSpec"

.

.

.

..

  ) ;LEFDefaultRouteSpec


( "LEFDefaultRouteSpec_3H"                    nil

.

.

.
```

```
) ;LEFDefaultRouteSpec_3H
```

Note that the second LDRS (`LEFDefaultRouteSpec_3H`) did not need to be defined as a LEFDefaultRouteSpec, as Innovus will search the tech for the name specified using the following command:

```
init_oa_default_rule  LEFDefaultRouteSpec_3H
```

3.2.7 Rapid PDK ASCII Techfile Structure

The basic sections of the ASCII technology file are mentioned below. The Rapid PDK will not have all the sections defined fully as it will create only the technology data from the LEF and use basic data, such as layer names, from the base PDK.

- `controls`: This section has the general information for the PDK, such as the grid and technology reference libraries, along with the parameters that can be used across the technology library.
- `layerDefinitions`: This section has the user-defined layer and purpose definitions along with the display setup. Most of the layer definitions are part of the base PDK.
- `layerRules`: This section contains the layers function, the routing direction, the cutClasses and so on among other attributes for user-defined layers. The layer function is defined in the base PDK, but the cutClasses are defined in both the base and the Rapid PDK.
- `viaDefs`: This section contains all the standard and custom via definitions. Both the base and the Rapid PDK can have their own via definitions.
- `constraintGroups`: This section stores most of the technology rules for the layers. While Virtuoso reads in most of the technology rules from the `foundry` section in the base PDK, the Rapid PDK creates the `foundry_innovus` section, which has most of the LEF technology rules to be read in by Innovus.
- `siteDefs`: This section has the site definitions for the PDK. The sites can be defined in either the base or Rapid PDK.

Click **here** to view a sample of an ASCII techfile skeleton.

3.2.7.1 Key Subsections in the Rapid PDK ASCII Technology file

The primary sections mentioned above have various subsections to manage the information easily. Let's look at some of the key subsections in the Rapid PDK ASCII technology file in more detail.

- The `refTechLibs` subsection in the `controls` section: The name of the reference technology library of the Rapid PDK is seen in the `refTechLibs` subsection of the `controls` section. This subsection contains the name of the base PDK that is being referred by the Rapid PDK. All the layer names, layer functions, and display patterns for the layer purpose pairs are picked up from this base PDK. Example:

  ```
  refTechLibs(
  ; techLibName
  ; ---------
   "basePDKName"
  ) ;refTechLibs
  ```

- The `LEFDefaultRouteSpec` constraint group in the `constraintGroups` section: This constraint group captures the plance-and-route (PnR) information, such as specific information on the valid routing layers and the default via definitions needed by Innovus. The valid vias can also have the standard via definitions which can be used by Innouvs to create the required variants. This section also has the following technology rules specific to these PnR layers:
  - Layer direction for the routing layers. The routing direction can also be pre-defined in the base PDK in the `routingDirections` subsection of the `layerRules` section.
  - The minWidth for the routing layers
  - The routing pitch for the routing layers
  - The routing offset for the routing layers
  - The wire extension for the routing layers
  - The width and pitch for the 45-degree angle style routing

  Here's an example of the `LEFDefaultRouteSpec` constraint group in the `constraintGroups` section:

  ```
  constraintGroups(;( group [override]    [definition]    [operator] );( ----- ----------    ------------    ---------- )(
  "LEFDefaultRouteSpec" nil "LEFDefaultRouteSpec"   routingDirections(    ( M1 "horizontal" )    ( M2 "vertical" )...   )
  ;routingDirectionsroutingGrids(  ( horizontalPitch "M1" 0.2 )  ( verticalPitch "M1" 0.19 )  ( horizontalOffset "M1" 0.1 )  (
  verticalOffset "M1" 0.095 ) ) ;routingGridsspacings(  ( minWidth "M1" 0.06 ) ) ;spacings.....interconnect(( validLayers ( M1
  M2 M3 M4 M5 M6 ) )( validVias ( M2_M1 M3_M2 M4_M3 M5_M4 M6_M5 ))) ; interconnect) ;LEFDefaultRouteSpec
  ```

You can have multiple `LEFDefaultRouteSpec` constraint groups in the tech graph and specify the one from which rules are to be read. Use the following variable to control the name of the `LEFDefaultRouteSpec` to be read:

```
set init_oa_default_rule <LDRS_CG_name>
```

- The `foundry_innovus` constraint group in the `constraintGroups` section:
  The `foundry_innovus` constraint group captures all the technology rules for all the layers defined in the LEF file apart from the rules that are defined in the `LEFDefaultRouteSpec`. This section basically recreates all the LEF technology rules in the ITDB for use of Innovus without any dependency on the foundry section of the base PDK. All the LEF58 layer and library properties from the LEF file are also stored within this section.
  Innovus reads this section to populate the technology rules in its database.
  The `lef2oa` process might create several other constraint groups, such as `minSpacing`, `minDualExtension`, and so on and add them as member constraint groups within the `foundry_innovus` group.
  On the other hand, Virtuoso, by default, reads the technology rules from the `foundry` constraint group of the tech graph for all its applications.
  You can have multiple `foundry_innovus` constraint groups in the tech graph and specify the one from which rules are to be read. Use the following variable to control the name of the `foundry_innovus` constraint group to be read:

  ```
  set init_oa_foundry_rule <foundry_CG_name>
  ```

  Here's an example of the `foundry_innovus` constraint group in the `constraintGroups` section:

  ```
  ;( group [override]    [definition]    [operator] )

  ;( ----- ----------    ------------    ---------- )

   ( "foundry_innovus" nil "foundry_innovus"...spacings(...  ( layerProp   "M1" 'propName
  "LEF58_ANTENNAGATEPLUSDIFF" "ANTENNAGATEPLUSDIFF OXIDE1 0.350 ; ANTENNAGATEPLUSDIFF OXIDE2
  0.350 ;" )  ( layerProp   "M1" 'propName "LEF58_MINSTEP" "MINSTEP 0.10 MAXEDGES 1 MINADJACENTLENGTH 0.10
  ;" )  ( layerProp   "M1" 'propName "LEF58_RECTONLY" "RECTONLY ;" )  ( layerProp   "M1" 'propName
  "LEF58_RIGHTWAYONGRIDONLY" "RIGHTWAYONGRIDONLY ;" )  ( minWidth    "M1"   0.024 )  ( maxWidth    "M1"
  0.25 )  ( minArea    "M2"   0.00288 )  ( minEndOfLineSpacing   "M2"  (0.0405 0.0135 0.06) ) ) ;spacings

  memberConstraintGroups(...  "minSpacing"...  ); memberConstraintGroupsspacingTables(  ;( constraint        layer1
  [layer2]  ; (( index1Definitions  [index2Defintions])  [defaultValue] )  ; ( table) )  ;( ----------) (
  minDensity        "M1"...  ( maxDensity        "M1"  ) ) ;spacingTables
  ```

- The `LEFSpecialRouteSpec` constraint group in the `constraintGroups` section:
  The `LEFSpecialRouteSpec` (LSRS) is used primarily for power routing. It is used to list the

viaDefs that should be used at a particular width intersection. It is equivalent to the list of LEF VIARULE statements in LEF.
However, Innovus can use any standardViaDef, even the ones not listed in the LSRS for routing.
Earlier, the via generator used to determine the viaDef to be used on the basis of the order in the LSRS but now priority is given to the cut area and the viaDef that creates the largest cutArea is used.
To restrict the via choice, `setViaGenMode -viarule_preference <list of valid vias>` should be used.
Here's an example of the `LEFDefaultRouteSpec` constraint group in the `constraintGroups` section:

```
;( group [override]    [definition]    [operator] );( ----- ----------    ------------    ---------- )( "LEFSpecialRouteSpec" nil "LEFSpecialRouteSpec"

 interconnect(  ( validVias (       (( "layer1Width" nil nil "layer2Width" nil nil )       'viaLayerDirection       (.....)
;LEFSpecialRouteSpec
```

You can have multiple `LEFSpecialRouteSpec` constraint groups in the tech graph and specify the one from which rules are to be read. Use the following variable to control the name of the `LEFSpecialRouteSpec` to be read:

```
set init_oa_special_rule <LSRS_CG_name>
```

- The non-default rules (NDRs) in the `constraintGroups` section: Separate constraint groups are created for all the NDRs defined in the LEF files and these NDRs are available for use in Innovus. Example:

  ```
  ;( group  [override] ) ;( -----  ---------- ) ( "CTS_Rule" nil   spacings(...  ( minSpacing "M2" 0.13 'soft )  ( minSpacing "M1" 0.13 'soft )  ( minWidth "M1" 0.13 )  ( minWidth "M2" 0.14 )... ) ;spacingsinterconnect( ( validLayers (M1 M2 M3 M4 M5 ) ) ( validVias (VIA12_HV VIA23_VH VIA34_VH VIA45_VH ) )) ;interconnectspacingTables(;( constraint       layer1 [layer2]; (( index1Definitions [index2Defintions]) [defaultValue] ); ( table) );( ----------) ( minNumCut "V1"   (( "minNumCut" nil nil ) 1)...  )...  ) ;spacingTables) ;CTS_Rule
  ```

- The cutClass definitions in the `layerRules` section: All the cutClasses defined in the LEF file are added to the ITDB structure. The difference in name with the cutClasses in the base PDK does not matter as long as the width, length and numCuts parameters are in sync with the base PDK. Example:

  ```
  cutClasses( ;( layerName ) ;( (cutClassName                (width length)) ) ;( ---------- )(V1   (VSQ   'numCuts   1      (0.016 0.016))   (VREC   'numCuts   2      (0.016 0.03)))(V2   (VSQ   'numCuts   1      (0.016 0.016))   (VREC   'numCuts   2      (0.016 0.03)))......) ;cutClasses
  ```

  Note: Even if the name or number of the cutClasses is different in the base PDK and the

Rapid PDK, Innovus would always be reading the cutClasses defined in the Rapid PDK. Therefore, a LEF dump from Innovus with the Rapid PDK would show up the same cutClasses as in the technology lef file. However, to avoid any complications, it is recommended that you should keep the names and definitions of the cutClasses in the technology LEF file the same as those in the base PDK while creating the Rapid PDK.

- The via definitions in the `viaDefs` section: The via definitions from the LEF file, both custom and standard, are also created in the ITDB structure. However, the standard vias from both the base PDK and the Rapid PDK are available to be used in Innovus. Example:

```
viaDefs(

standardViaDefs(

;( viaDefName layer1 layer2 (cutLayer cutWidth cutHeight [resistancePerCut])

; (cutRows cutCol (cutSpace) [(l_cutPattern)])

; (layer1Enc) (layer2Enc) (layer1Offset) (layer2Offset) (origOffset)

; [implant1 (implant1Enc) [implant2 (implant2Enc) [well/substrate]]])

;( ------------------------------------------------------------ )

( M2_M1          M1      M2        ("V1" 0.07 0.07 5.0)

 (1 1 (0.07 0.07))

 (0.005 0.03) (0.005 0.03) (0.0 0.0) (0.0 0.0) (0.0 0.0)

)

( M3_M2          M2      M3        ("V2" 0.07 0.07 5.0)

(1 1 (0.07 0.07))

(0.005 0.03) (0.005 0.03) (0.0 0.0) (0.0 0.0) (0.0 0.0)

)

...

) ;standardViaDefs

customViaDefs(;( viaDefName libName  cellName viewName layer1 layer2 resistancePerCut);( ---------- -------  -------- --------
------ ------ ----------------) ( M2_M1     RapidPDK  M2_M1     via     M1     M2    0.0) ( M3_M2     RapidPDK  M3_M2     via
M1     M2    0.0)......) ;customViaDefs
```

- The site definitions in the `siteDefs` section: The site definitions for the standard cells are also saved in the `siteDefs` section of the ITDB techfile. Example:

  siteDefs(scalarSiteDefs(;(siteDefName type width height symInX symInY symInR90) ;(---------- ---- ----- ------ ------ ------ ------) (CoreSite core 0.2 1.71 nil t nil) (PadSite pad 240.0 240.0 nil nil nil)) ;scalarSiteDefs) ;siteDefs

  Note: The site definitions are read in from all the libraries specified in the reference library list during the `init_design` step in Innovus. So, if some of the sites are not defined in the technology graph (ITDB structure) but are read from any reference library, they would be saved in the design library. However, during `restoreDesign`, it is expected that the site definitions are part of the technology graph and are not read from the reference libraries.
- The `FIXEDMASK` statement in the `foundry_innovus` constraint group in the `constraintGroups` section: The `FIXEDMASK` statement from the LEF file is mapped to the `colorControl` section of the ASCII tech file in the foundry group. It is set in the `foundry_innovus` constraint group in case the base PDK foundry constraint group does not have this set.

  ("foundry_innovus" nil ... colorControl((integrationColorModel "locked")) ;colorControl ...) ;foundry_innovus

3.2.7.2 Mapping for Other Key Rapid PDK Constructs

Mapping for the LEF Library Properties

The LIBRARY PROPERTYDEFINITIONS from the LEF file are mapped to `libProp` in the `foundry_innouvs` subsection of the ASCII techfile. They are also set as properties on the Rapid PDK library.

Note: If any `libProp` definitions are found in the Rapid PDK while reading the OpenAccess data to Innovus, the properties set on the library are ignored.

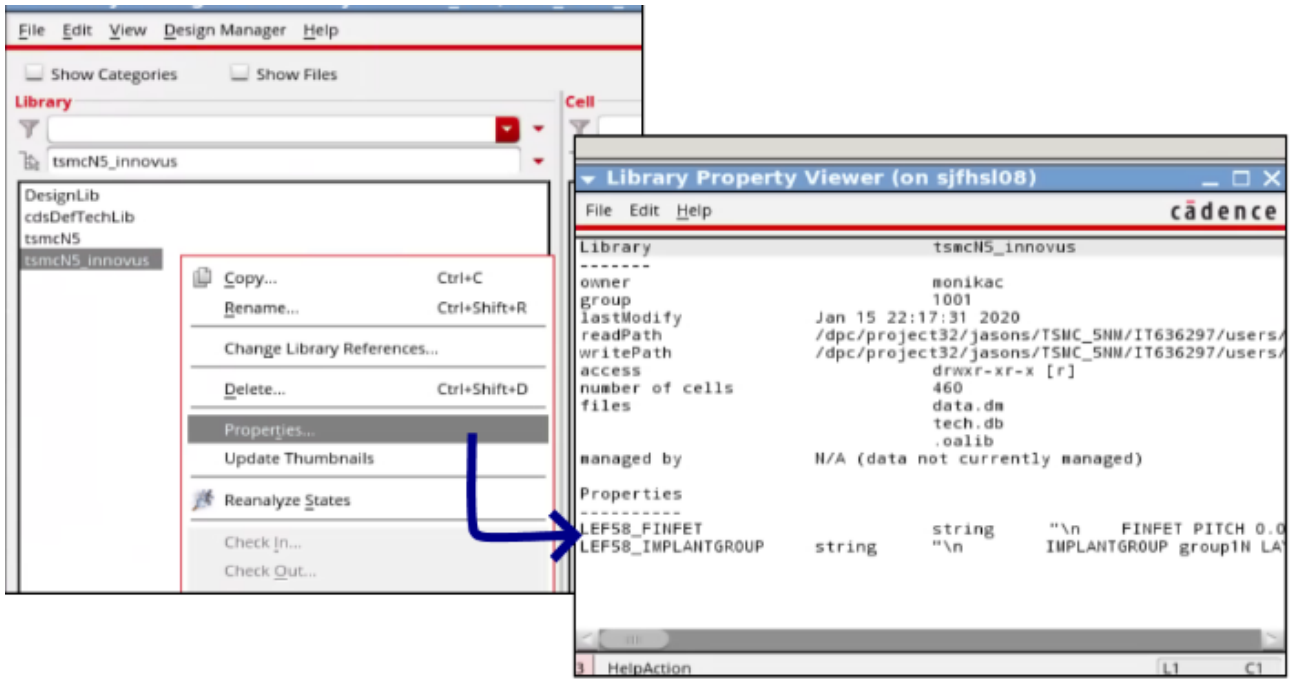
Here's a sample mapping of the LIBRARY PROPERTYDEFINITIONS in the LEF file to the ASCII techfile:

LEF File Section	ASCII Techfile Syntax

PROPERTYDEFINITIONS LIBRARY LEF58_FINFET STRING " FINFET PITCH 0.056 OFFSET 0.0 HORIZONTAL NOCORECELL ; " ; END PROPERTYDEFINITIONS	("foundry_innovus" nil "foundry_innovus" ... (libProp 'propName "LEF58_FINFET" " FINFET PITCH 0.056 OFFSET 0.0 HORIZONTAL NOCORECELL ; ") ...) ;foundry_innovus

You can check the LIBRARY PROPERTYDEFINITIONS from the Library Manager in Virtuoso. To do so:

1. Choose *CIW -> Library Manager*.
2. In the listing of the libraries, right-click the name of the Rapid PDK tech library and select *Properties* from the context menu.



Note: All the properties in the base and Rapid PDKs are read into Innovus. However, if a property with the same name exists in both the base and Rapid PDKs, only the version from the Rapid PDK is read into Innovus.

Checking for Layer Names, Definitions, and Number of Masks

- The layer names between the base PDK and the technology LEF file must match. In case of any mismatches, fix the technology LEF using a sed script or the vi editor to match the names with the base PDK before creating the Rapid PDK structure.
- This layer name change would also be needed for the standard cell and NDR LEF files.
- All the layer definitions must exist in the base PDK; any new layers defined in the LEF file will be dropped during the `lef2oa` process.
- If a layer has multiple masks, ensure that the mask number has the same value defined in both the base PDK and the technology LEF file. In case of a mismatch, the number of masks defined in the base PDK will be kept and the LEF mask number will be ignored.

Here is a sample section from the base PDK with the layer definitions and number of masks per layer values:

```
layerDefinitions(techLayers(;( techLayer number abbreviation [#Masks] );( --------- ------ ------------ -------- )... ( M1 31 M1 2 ) ( M2 32 M2 2 )...) ;techLayers...) ;layerDefinitions
```

Note: Pseudo-material layers, such as routing, cut, and trimmetal, for the region-based rules (Example: `M1_FB1`) do not need to be defined in the base PDK; these are created in the *`rapidPDK_techLibName`*.

Checking for Routing Layer Usage and Routing Direction

- Ensure that there is no mismatch between the LEF file and the base PDK in the layer type definition or usage (for example, whether a layer is used as metal, cut, masterslice and so on). `lef2oa` will give an error if the type in the LEF file does not match the type specified in the base PDK.
  Note: However, if the base PDK has a particular type set, the LEF can be used to update the subclasses of the same type. For example, if the base PDK has a layer with type CUT, the LEF can have the TYPE as IMPLANT and that will be updated. Similarly, for a type routing in the base PDK, the type can be overridden in the tech LEF file.
- Ensure that the routing direction for the metal layers is also the same between the LEF file and the base PDK.

Here is a sample section from the base PDK with the layer functions:

```
layerRules( functions(;( layer function [maskNumber] );( ----- -------- ------------ )... ( M1 "metal" 104 ) ( VIA1 "cut" 107 ) ( M2 "metal" 108 )... ) ;functions routingDirections( ;( layer direction ) ;( ----- --------- )... ( M1 "vertical" ) ( M2 "horizontal" )... ) ;routingDirections...) ;layerRules
```

Checking for the OALAYERMAP Definition for Layers with Multiple Masks

The `LEF58_OALAYERMAP` construct defined in the techfile can be used to map a layer with multiple masks to different layer names. This is supported only for trim metal layers. You need to make sure that the layermap definition is not repeated in the technology LEF file in case it is already defined in the base PDK.

Consider an example in which the `MASK 1` shapes on LEF layer `CL1` would go to the OpenAccess layer `CL1A` while the `MASK 2` shapes on `CL1` would go to the OpenAccess layer `CL1B`. In this case:

- Ensure that the layer `CL1` is not defined in the base PDK as that can cause issues with the tool since the shapes will be created on different layers based on the mask number.
- The layer `CL1` needs to be defined in the technology LEF file, whereas the layers `CL1A` and `CL1B` need to be defined in the base PDK, as shown below.

LayerMap in the Base Virtuoso techfile	controls(... techParams(;(parameter value) ;(---------- -----) (LEF58_OALAYERMAP_RapidPDK "OALAYERMAP CL1A LAYER CL1 MASK 1 ; OALAYERMAP CL1B LAYER CL1 MASK 2 ; ")...) ;controls
LEF File Definition	LAYER CL1 TYPE MASTERSLICE ; MASK 2 ; ...END CL1

3.2.7.3 ASCII Techfile Skeleton

; Technology File myTechLib; Generated on MM DD HR:MIN:SEC YYYY; with @(#)$CDS: techLoadDump version BUILDVERSION MM/DD/YYYY HH:MM (machineName) $controls(...

techParams(

...

) ; techParams ...) ; controls

layerDefinitions(

```
        ...
        techLayers(          ...
        ) ;techLayers
        ...
) ;layerDefinitions
layerRules(
    ...
    functions(
    ...
    ) ;functions
    ...
    cutClasses(
    ...
    ) ; cutClasses
    ...
) ; layerRules

viaDefs(
    ...
    standardViaDefs(
    ...
    ) ; standardViaDefs
    ...
    customViaDefs(
    ...
    ) ; customViaDefs
) ;viaDefs
```

```
constraintGroups(
        ...
        ( "foundry"
         ...
        ) ;foundry
...
) ; constraintGroups

siteDefs(
    ...
    scalarSiteDefs(
    ...
    ) ; scalarSiteDefs
) ;siteDefs
```

4

Design Data Preparation

- Special Settings/Instructions To Enable the Mixed-Signal Flow
- Name Space Mapping in the OpenAccess Interoperable Flow
- Converting an Existing LEF/DEF-based Design into OpenAccess
  - Converting a LEF/DEF-based design into OpenAccess by using oaOut
- Migrating a LEF and Floorplan File-based Flow to OpenAccess
- Running Innovus in the OpenAccess Mode
  - Initializing Innovus for Implementing a Digital Block in OpenAccess
  - Initializing Innovus with the OpenAccess Database of a Design Created in Virtuoso XL
- Interoperability of Constraint Groups and Non-Default Rules between Virtuoso and Innovus
- Generic Guidelines To Run the Netlist-Driven Mixed-Signal Flow

4.1 Special Settings/Instructions To Enable the Mixed-Signal Flow

⚠ Cadence does not support a mixed signal flow that combines LEF and OpenAccess in the same flow. The flows used should be either entirely LEF-based or entirely OpenAccess. The OpenAccess flow has the advantage of full interoperability with Virtuoso, and enables many features such as floorplanning, routing constraint interoperability and Mixed Signal Static timing analysis. The instructions for starting Innovus in the OpenAccess mode are included in this section of the documentation.

- To facilitate the use of the mixed-signal flow, Cadence has taken steps to prevent problems that could arise from the use of non-interoperable or bad data originating from the Virtuoso platform. Cadence strongly recommends you to use a data and technology checker whenever Virtuoso-created data is to be brought into Innovus. The OpenAccess Database Interoperability Checker (oaDBChecker) provided by Cadence will flag any technology or data-related issues that could cause problems later in the flow. The checker can be run directly from Virtuoso, where it is available as a plug-in. For steps to load the oaDBChecker, refer to the OpenAccess Database Interoperability Checker chapter.
- Innovus recognizes parameterized cells (PCells) already created by Virtuoso. However, it cannot create PCells by itself.
- To view PCells in Innovus™ Implementation System, run the following commands at the UNIX prompt before starting the Innovus session:

  ```
  setenv CDS_ENABLE_EXP_PCELL TRUE

  setenv CDS_EXP_PCELL_DIR <directory path>
  ```

  The default directory name for the PCells cache is `.expressPcells`.
- To import an OpenAccess database created from Virtuoso that contains objects that are not understood by Innovus (such as MPPs), set the following variable in Innovus:

  ```
  setOaxMode -updateMode true
  ```

  This setting will retain any objects not understood by Innovus, and will save the objects back to the OpenAccess database that is output from Innovus.

  If you are not sure whether the OpenAccess database created from Virtuoso contains objects that are not understood by Innovus, set the following variable in Innovus:

  ```
  setOaxMode -updateMode auto
  ```

This setting enables update mode internally **if** the software finds objects that are not supported in Innovus.

- If you are performing power routing in Innovus using `sroute` in the OpenAccess flow, you should control `sroute` so that it does not create custom vias. This is important because saving a design with custom vias in Innovus will require write access to the technology library, and in many cases this library is Read-Only. To prevent `sroute` from creating non-symmetrical vias, you should have this setting in Innovus prior to running `sroute` in your script:

  ```
  setViaGenMode -symmetrical_via_only true
  ```

- By default, the DEF-based flow is used for invoking Quantus extraction. To use the OpenAccess interface instead for invoking Quantus extraction, set the following variable in Innovus:

  ```
  setExtractRCMode -useQrcOAInterface true
  ```

- To create SPECIALNETS for Innovus (nets that would be in the DEF SPECIALNETS section), geometric routes need to be created in Virtuoso. To enable GUI access for creating geometric routes, include the following in the .cdsinit file when working in the Virtuoso environment.

  ```
  ; Enable the Create->Geometric Wire functionality

  envSetVal( "we" "mixedSignalWireEditingEnvironment" 'boolean t )
  ```

- Set the compression level of a library, during the library creation process in Innovus, to `0` using the `setOaxMode` command. Innovus supports the compression of theOpenAccess database and does it by default. However, Virtuoso 6.1.5 version does not support a compressed database as the input. So, in order to create an OpenAccess library that can interoperate with Virtuoso 6.1.5, the compression in Innovus needs to be turned off by using the following Innovus command:

  ```
  setOaxMode -compressLevel 0
  ```

  Cellviews written to a library will follow the compression level of the library and not the `setOaxMode -compressLevel` setting at the time when the cellview is being created.

4.2 Name Space Mapping in the OpenAccess Interoperable Flow

A name space refers to the conventions that a tool uses for denoting such things as bus notation, hierarchical delimiter, and so on. Given the nature of place-and-route (P&R) and custom tools, name space differences exist between Innovus and Virtuoso. While the Innovus name space is aligned with LEF/DEF, Virtuoso aligns to the C-level Database Access (CDBA) name space. Some examples of namespace differences:

- The hierarchical delimiter in Innovus is the `/` character, while Virtuoso uses the `` ` `` character to denote hierarchy. So `InstxV/instyV` in Innovus is seen as `InstxV`instyV` in Virtuoso.
- Similarly, the bus notation of `[]` in Innovus will be seen as `<>` in Virtuoso. So `da_in[0]` in Innovus is seen as `da_in<0>` in Virtuoso.

Both Virtuoso and Innovus are capable of outputting the data in the name space format where the data is going to be consumed. For instance, when a DEF file is generated from the data inside Virtuoso, Virtuoso internally maps the CDBA name space to DEF while outputting the DEF file. Similarly, when a Verilog netlist is written from Virtuoso, the CDBA name space gets mapped to the Verilog name space by Virtuoso.

In the OpenAccess interoperable flow, the mapping of the name spaces happens automatically. When an OpenAccess design is saved in Innovus and later opened in Virtuoso, you DO NOT need to make any changes in the name space seen in Virtuoso. If the same cell view is edited and saved in Virtuoso, the next time that cell view is opened in Innovus, you will see the Innovus name space.

4.3 Converting an Existing LEF/DEF-based Design into OpenAccess

You may have a placed and routed Innovus design that you would like to convert to OpenAcess in order to exercise the many interoperability related features of OpenAccess or simply to run the Innovus flow in OpenAccess. Conversion to OpenAccess is possible only if an interoperable mixed-signal OpenAccess (MSOA) PDK has already been created for the particular technology and metal stack used by the design. To convert the Innovus design into OpenAccess, use the following steps:

1. Restore the existing LEF/DEF-based design:
   ```
   restoreDesign PNRComplete.dat myDesign
   ```
2. Set the `init_lef_file` global variable to null to instruct Innovus that a LEF technology file will

not be used in this session:

```
set init_lef_file ""
```

3. Specify the OpenAccess technology and standard cell libraries and any OpenAccess macro libraries:

   ```
   set init_oa_ref_lib {techLib stdcellLib macroLib}
   ```

   In this example, `techLib` is the interoperable MSOA PDK, which has been created by merging some of the technology information found in a tech LEF with the base PDK used by Virtuoso, as explained in the "Implementing an Interoperable/MSOA PDK" section in the Technology Data Preparation chapter.

4. Write out the LEF-equivalent information from the LEF/DEF-based design that was loaded into memory using `restoreDesign` in Step 1:

   ```
   write_lef_library before_conversion.lef
   ```

   The file created in this step can be compared to the LEF that will later be created from the OpenAcess representation of the same design. This step is highly recommended as it enables you to detect possible issues with the MSOA PDK, early in the flow.
   The `write_lef_library` command has an option for writing out only the technology information (using the `-tech_only` option) or the macro data (using the `-macro_only` option). Cadence recommends that both type of data in the LEF be compared; however, if you prefer to compare the DRC rules separately, the command could be run twice, with one option at a time, to create two different files that can later be used for comparison. Note that this command could have also been run directly after the `restoreDesign` command, because the setting of variables after `restoreDesign` does not change the LEF data seen by this command.

5. Save the design into an OpenAccess lib/cell/view format:

   ```
   saveDesign -cellview {myDesignLib myDesign PNRcomplete}
   ```

   In this example, `myDesignLib` is the OpenAccess design library that will contain the design data, the specific cell name is `myDesign`, and the name of the view is `PNRcomplete`. This step would have not been possible without the MSOA tech file that was loaded earlier using the `set init_oa_ref_lib` command in Step 3.

At this point, the LEF/DEF-based design has been saved the design into OpenAccess. Now, you need to make sure that the technology and macro information is the same between the LEF and OpenAccess versions of the design. To do so, you need to restore the converted OpenAccess design in a different session of Innovus and write out the equivalent LEF information so that you can compare it with the LEF written out to the `before_conversion.lef` file in Step 4. To restore the design in a different session, a `cds.lib` file needs to exist in the current working directory where Innvous is getting invoked. This file enables Innovus to find the directory where the design data is

stored. The syntax of the `cds.lib` file could be as follows:

```
DEFINE myDesignLib /usr/A/myDesignLib
```

In this example, the `/usr/A/myDesignLib` directory will contain a directory named `myDesign`, which is the top-level cell name in the design.

In a new Innovus session, perform the following steps:

1. Restore the converted OpenAccess Design:
   ```
   restoreDesign -cellview {myDesignLib myDesign PNRcomplete}
   ```
2. Write out the LEF information from the converted design:
   ```
   write_lef_library after_conversion.lef
   ```
   This file will contain the LEF data created from the OpenAccess technology and macro information loaded in Innovus when the design is read in using the `restoreDesign` command.

Now, you can run the Unix `diff` command on the `before_conversion.lef` and `after_conversion.lef` files. If the conversion happened correctly, there will be no differences in the two files from a DRC point of view. The two files should also have the same set of macros.

The following differences in the two LEF files will not cause any problems when the design is run in Innovus:

- Extra masterslice/implant layers in `after_conversion.lef`
- Extra vias/sites in `after_conversion.lef`

If the routing and cut layers are the same between the two LEFs and the DRC rules match, the OpenAccess PDK is in sync with the LEF from a technology point of view.

4.3.1 Converting a LEF/DEF-based design into OpenAccess by using oaOut

You can use the `oaOut` command in Innovus to send the design data in the OpenAccess format to either Virtuoso or a third party tool capable of reading OpenAcess. For `oaOut` to work, you need to first create an interoperable MSOA PDK containing the design rules for the specific process used by the design. Follow steps 1 and 2 below to create the technology library to be used in this process:

1. With the design loaded in Innovus, execute the following two commands:
   a. `write_lef_library -tech_only TECH.lef`
   b. `write_lef_library -macro_only MACRO.lef`

(If the design has no macros, skip this step.)

2. Run the `lef2oa`command. `lef2oa` is a stand-alone UNIX command that is shipped with the Innovus software. You can found it in the Innovus hierarchy:
   a. `lef2oa -useFoundryInnovus -lef TECH.lef -lib RapidPDK -techRefs BasePDK`
   b. `lef2oa -lef MACRO.lef -lib RapidPDK`
      (If the design has no macros, skip this step.)
   c. `lef2oa -lef stdcell.lef -lib RapidPDK`
      (This will convert the std cell libraries to the OpenAccess format, referencing the technology library.)

   Here, `RapidPDK` is the name of the new tech file that will have all the technology information for both Innovus and Virtuoso. `BasePDK` is the technology file used to run the Virtuoso environment. The above process builds an Incremental Technology Database (ITDB) on top of the existing technology file for Virtuoso.
3. Next, create an OpenAccess design library to hold the design data to be sent to Virtuoso by using the following command:
   a. `createLib myDesignLib -referenceTech {RapidPDK}`
4. Now, use the `oaOut` command to convert the design to OpenAccess and save it in the design library that was created in the previous step:
   a. `oaOut myDesignLib myRoutedDesign layout -refLibs RapidPDK`
      (A cell view by the name of `myRoutedDesign` is created in the design library `myDesignLib` referencing the technology information in the `RapidPDK`.)

Note: Ensure that the UNITS in `BasePDK` and `TECH.lef` are the same. If the UNITS in the LEF file are 2000 and `BasePDK` has UNITS 1000, there might be errors in the migration to OA data. Some wire widths, begin and end extensions might be having a decimal precision that might not be supported on a lower UNITS value. The following error will be seen for such wires while saving the design to OpenAccess:

```
**ERROR: (IMPOAX-448): Could not save OA database: OA Exception: The width specified
for segStyle must be an even number.
```

A sample script to find such wires/swires is provided below. You can modify the same as per your design.

Sample Script for swires

```
set ww [dbGet top.nets.swires]
foreach w ${ww} {
set box_sizey [dbGet ${w}.width]
set beginExt [dbGet ${w}.beginExt]
set endExt [dbGet ${w}.endExt]
set layer [dbGet ${w}.layer.name]
set netname [dbGet ${w}.net.name]
# Check
set check_width [expr int(1000.0*${box_sizey}/2.0) != (1000.0*${box_sizey}/2.0)]
set check_begExt [expr int(1000.0*${beginExt}) != (1000.0*${beginExt})]
set check_endExt [expr int(1000.0*${endExt}) != (1000.0*${endExt})]
set check [expr ${check_width} + ${check_begExt} + ${check_endExt}]
if {${check} > 0} {
puts "FAIL: ($check_width $check_begExt $check_endExt) ${netname} ${layer} ${box_sizey}
[dbGet ${w}.box]"
}
}
```

This will flag the swires that will cause errors in migration to OpenAccess due to the difference in UNITS.

Note: You should run the same script for wires as well.

Sample result of above script

```
FAIL: (1 0 0) VDD M4 0.075 {336.7 63.035 341.1 63.11}
FAIL: (1 0 0) VDD M3 0.605 {308.85 117.42 309.455 121.82}
FAIL: (1 0 0) VSS M4 0.075 {343.0 63.035 347.4 63.11}
FAIL: (1 0 0) VSS M3 0.605 {308.85 123.72 309.455 128.12}
```

4.4 Migrating a LEF and Floorplan File-based Flow to OpenAccess

The main difference between the LEF-based flow and the OpenAccess flow is the use of OpenAccess libraries and cell views instead of the text-based LEF and floorplan files.

Following are the steps to migrate a LEF and Floorplan file-based flow to OpenAccess:

1. Create OpenAccess technology and cell libraries that would be compatible for use in Innovus and Virtuoso. You can do so by creating an Incremental Technology Data Base (ITDB) structure for the interoperable OpenAccess libraries. You will need to have Virtuoso in your path settings.
    a. Create an OpenAccess technology library for the technology LEF file using `lef2oa`.
    ```
    lef2oa -lef tech.lef -lib techLib -libPath refLibs
    ```
    This creates the `LEFDefaultRouteSpec` constraint group in the technology library.
    The `LEFDefaultRouteSpec` constraint group has all the technology information needed by Innovus.
    This also creates the display.drf file with the color and stipple patterns for the layers to be used in the dfII environment.
    b. Import the MACRO LEF files in a new ITDB with the `techLib` as the reference for the technology.
    ```
    lef2oa -lef stdcells.lef -lib cellsLib -techRefs techLib -libPath refLibs
    ```
    This creates the cell library with the OpenAccess abstract views for the cells while referring the technology information from the `techLib`.
    c. As `lef2oa` does not create bus information in the abstracts, you need to create the bus information in these abstracts using `verilogAnnotate`:
    ```
    verilogAnnotate -verilog stdcells.v -refLibs cellsLib -refViews abstract
    ```
    This creates the bus and the bus term order, ascending or descending, in the OpenAccess cell views.
    You will need the verilog stubs for the cells to be used for `verilogAnnotate`. You can use `oa2verilog` to create the verilog stubs in case you do not have them readily available.
2. Next, create an OpenAccess view for the text floorplan file. There can be two approaches for the above:

- Use `defOut` from the LEF-based Innovus database after the .fp file has been loaded and then use `def2oa` to create an OpenAccess cellview with the floorplan information.

  ```
  defOut -routing from_innovus.def
  def2oa -def from_innovus.def -lib designLib -cell top -view floorplan -
  techrefs techLib
  ```

  This creates an OpenAccess design library with the top cell and the initial seed floorplan view.
- Save OpenAccess directly from Innovus after loading the `.fp` file:

  ```
  saveDesign -cellview {designLib top floorplan}
  ```

  The OpenAccess floorplan view cannot be saved from data created using LEF-based libraries. The LEF libraries must be converted to OpenAccess form and the design reloaded using those OpenAccess libraries.

3. Finally, if you are using the Innovus Foundation Flow, you need to modify some variables in the `setup.tcl` file:
   a. Add the following variables in `setup.tcl` so that the OpenAccess data is read in for the technology, cells, and the floorplan:
      - `set vars(dbs_format) oa`
      - `set vars(oa_abstract_name) abstract`
      - `set vars(oa_layout_name) layout`
      - `set vars(oa_design_lib) designLib`
      - `set vars(oa_ref_lib) macroLib`
      - `set vars(oa_fp) "designLib top floorplan"`
   b. Unset the following variables or remove them from `setup.tcl` to do away with the text LEF and floorplan files:
      - `set vars (fp_file) top.fp`
      - `set vars (lef_files) “”`

   After the `setup.tcl` file has been modified as mentioned above, the rest of the scripts and supporting files in the foundation flow can be used as it is for all the flow steps.

4.5 Running Innovus in the OpenAccess Mode

In this mode, Innovus needs to use the common PDK described in the previous sections of this document. No LEFs are needed for Innovus to operate in the OpenAccess mode, because both the technology data as well as the standard cell data can be represented in OpenAccess. The design connectivity needed for Innovus in the mixed signal flow could be read from either a Verilog netlist or an OpenAccess database generated in Virtuoso XL. In Virtuoso XL, the *Generate From Source* command creates a layout view which has the connectivity information in the original schematics.

In typical mixed signal designs, there may be one or more digital blocks that need to get implemented. If the top level of the design is represented as schematics in Virtuoso, the digital blocks in the design are represented as soft blocks and typically come with Verilog Netlist created from Synthesis. In these cases, Innovus needs to be initialized using the Verilog netlist of the particular block, and later in the flow, the floorplan of the block created in Virtuoso could be read into Innovus using the command `oaIn`. This flow is referred to as **Analog on Top.**

In another commonly practiced flow, the entire design (captured as an OpenAccess cellview in Virtuoso) is brought into Innovus for either co-design, analysis, or ECO. In that flow, Innovus can be initialized with the OpenAccess database created by Virtuoso XL, and there is no need to have a Verilog netlist for the top level of the design. This flow is referred to as **Mixed Signal on Top**.

4.5.1 Initializing Innovus for Implementing a Digital Block in OpenAccess

The following setting can be used to initialize Innovus for implementing a digital block in OpenAccess. Note that these settings must be made once Innovus is invoked.

```
set init_abstract_view abstract

set init_layout_view layout

set init_oa_ref_lib {lib1 lib2 lib3}
```

#These are the OpenAccess reference libraries used in the design. *lib1* could be the OpenAccess library containing the technology information, *lib2* could be the OpenAccess standard cell library, and *lib3* could be the OpenAccess reference library that is referencing the technology information in *lib1*, but has additional Vias not present in *lib1*. Note that the first library (in this example, *lib1)* is the source of the technology information for Innovus. It is also acceptable for the first library to either attach or reference the library that contains the actual technology information. For example, if the first library is a standard cell library that either references or attaches the library containing the technology information, the standard cell library is allowed to be the first library listed in the `set init_oa_ref_lib` setting. Users typically include the standard cell library and the pad cell library in `set init_oa_ref_lib`, but not the IP block libraries.

`set init_oa_search_lib {lib1 lib2}` #The libraries listed here are loaded only if the design has cell references that do not exist in the libraries specified by `set init_oa_ref_lib`. You may have several macro libraries, but some designs may have macros instantiated from only one of these libraries. In this case, you do not want Innovus to load many libraries that are not actually needed by it. Instead, the tool loads only the libraries it needs based on whether the macro is instantiated. Note that this setting is meant for use only when Innovus is being initialized with a Verilog Netlist.

```
set init_verilog <name of the Verilog netlist representing the digital block being implemented in Innovus>
```

In addition, if the PDK you have been given is a Rapid MSOA PDK with multiple `foundry_innovus` constraint groups, you need to specify the following settings so as to select the correct `foundry_innovus` constraint group, along with the correct LEFDefaultRouteSpec and LEFSpecialRouteSpec:

```
set init_oa_foundry_rule <name of the foundry_innovus constraint Group>

set init_oa_default_rule <name of the LEFDefaultRouteSpec>

set init_oa_special_rule <name of the special Route Spec>
```

Some of the other common settings to start Innovus (not directly related to OpenAccess) are:

```
set init_mmmc_file "name of the Multi Mode Multi Corner file used for running analysis in the Innovus implementation flow"

set init_pwr_net {name of the power net in the design}

set init_gnd_net {name of the power net in the design}

set init_design_uniquify {true}
```

4.5.2 Initializing Innovus with the OpenAccess Database of a Design Created in Virtuoso XL

The following setting can be used to initialize Innovus when the OpenAccess database of a design created in Virtuoso XL is brought into Innovus for the first time. Note that these settings must be made once Innovus is invoked.

`set init_design_netlisttype {OA}` #This tells Innovus to get the needed connectivity from the OpenAccess database, instead of a Verilog netlist.

`set init_oa_design_lib {design_lib}` #This tells Innovus the OpenAccess design library that contains the cellview representing the top level of the design.

`set init_oa_design_cell {cellview}` #This tells Innovus the name of the OpenAccess

cellview representing the top level of the design.

`set init_oa_design_view {layout}` #This needs to be set to the name of the view containing the full layout of the cell.

`set init_oa_ref_lib {lib1 lib2 lib3}` #These are the OpenAccess reference libraries used in the design. *lib1* could be the OpenAccess standard cell library, *lib2* could be the OpenAccess library containing the technology information, and *lib3* could be the OpenAccess reference library that is referencing the technology information in *lib2*, but has additional vias not present in `lib2`.

In addition, if the PDK you have been given is a Rapid MSOA PDK with multiple `foundry_innovus` constraint groups, you need to specify the following settings so as to select the correct `foundry_innovus` constraint group, along with the correct LEFDefaultRouteSpec and LEFSpecialRouteSpec:

```
set init_oa_foundry_rule <name of the foundry_innovus constraint Group>

set init_oa_default_rule <name of the LEFDefaultRouteSpec>

set init_oa_special_rule <name of the special Route Spec>
```

Some of the other common settings to start Innovus and not directly related to OA are:

```
set init_mmmc_file "name of the Multi Mode Multi Corner file used for running
analysis in the Innovus implementation flow"

set init_pwr_net {name of the power net in the design}

set init_gnd_net {name of the power net in the design}

set init_design_uniquify {true}
```

4.6 Interoperability of Constraint Groups and Non-Default Rules between Virtuoso and Innovus

Innovus uses the following guidelines while reading constraint groups (the physical DRC rules) from the OpenAccess database for using them as non-default rules within the same Innovus session:

- The OpenAccess constraint group should not be a technology-specific constraint group.
- A constraint group should not have another nested or hierarchical constraint group within it.
- A constraint group should have all valid routing layers defined in it with minimum routing width defined for each layer. All the routing layers should have material type as `metal`. You can also use polysilicon layer for routing by attaching the following LEF property:
  PROPERTY LEF58_TYPE "TYPE POLYROUTING;";
- A constraint group should have valid routing vias defined for each pair of routing layer in the `validVia` list.

You can obtain a list of non-default rules using the `dbGet head.rules` command in Innovus.

Note: If any constraint inside a constraint group has a name starting with `ms`, Innovus treats that group as a mixed-signal constraint group and that group is read as a mixed-signal routing rule.

Note: You can confirm the non-default rule by writing the DEF file from an Innovus session, which has the complete definition of non-default rules and the nets using these non-default rules.

4.6.0.1 *Example*

```
- CLK ( PIN CLK )
+ SHIELDNET VSS
+ NONDEFAULTRULE doublewidth ;
```

For more information about non-default rules in DEF, see the DEF Syntax chapter in the *LEF/DEF Language Reference.*

4.7 Generic Guidelines To Run the Netlist-Driven Mixed-Signal Flow

Consider the following guidelines before running the netlist-driven mixed-signal flow:

- Designs can have black boxes, full custom analog blocks, custom digital blocks, and all interoperable custom shapes physically located in the top cell instance. Logical hierarchy in the design netlist is also supported.
- While working in Virtuoso, do not copy a library, cell, or a view in Innovus nor save it by changing the cell name. This can cause problems while reading the changed cell view in Innovus.
- Always use `restoreDesign` to load a design or cellview saved previously from Innovus. This command will ensure that all the variables and side files required to load a database in Innovus are present and you will be able to run all design implementation steps smoothly.
- Use the `init_design` and `oaIn` commands only while creating a new design library or while importing a design for the first time in Innovus.
- The routing layer and its corresponding purpose, for example, `drawing`, `pin`, and so on, in the OpenAccess database can be interoperated using Innovus. If a pin or net geometry is found for any routing layer on any other purpose at the top level, it will be converted to drawing purpose while interoperating the data using Innovus.

5

Schematic-Driven Mixed Signal Design Flow

- Schematic-Driven Mixed Signal Flow
  - Overview
  - Top-Level Early Floorplanning
  - The Analog Block Implementation Sub-Flow
  - Custom Digital Implementation Flow
  - Digital Block Implementation Flow
  - OpenAccess Based Interoperable Flow between Innovus and Virtuoso
  - Top-Level Block/Chip Assembly flow
  - Virtuoso Top-Level Assembly
  - Virtuoso Chip-Level Layout Assembly
- Flow Example: Power PushDown Flow for Digital Blocks in AoT Designs
  - Performing Top-Level Floorplanning in Virtuoso
  - Pushing Power Shapes into the Digital Block
  - Implementing the Digital Block in Innovus
  - Bringing Back the Digital Block into Virtuoso for Design Assembly

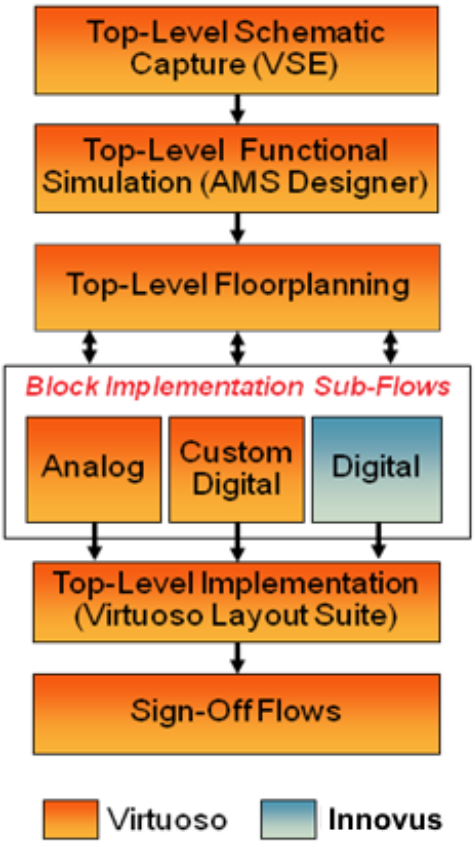
5.1 Schematic-Driven Mixed Signal Flow

5.1.1 Overview

In this chapter, we will begin by discussing a high level overview of the schematic-on-top flow. In addition to covering the core flow, we will include some of the peripheral tasks both at the beginning and at the end of the flow in order to provide some needed perspective.

The terms top-down versus bottom-up design methodologies refer to the order in which a hierarchical design is implemented. Ideally, the design should be done top-down, but in reality it is more common to construct the design simultaneously using a combination of top-down and bottom-up methodology.

Ideally, the schematic capture will begin in a top-down fashion with a top-level schematic. It is this top level connectivity source that makes it possible to do early floorplanning within the Virtuoso layout environment, and differentiates this schematic-on-top implementation from the netlist-driven or netlist-driven mixed signal approach. Schematic-on-top does not mean that the schematic must exist for all the blocks in the design throughout the entire hierarchy. It only refers to the top-level. In addition, the schematics will be used to drive the implementation of any analog portions of the design. However, netlists will and should be used for most of the digital-design content.



5.1.1.1 Top-Level Schematic Capture

At the initial stages of the design, only functional and performance specifications are available. The designer has the task of deciding the optimal-design partitioning with the major sub-systems identified as sub-blocks. Normally, the design partitioning is determined by the major functional blocks, performance, and design disciplines required for the overall implementation. As a result, the design is decomposed into analog and digital functional blocks.

5.1.1.2 Functional Verification

It is important to note that both the verification and implementation flows are driven from the same schematic database, thereby ensuring consistency between the two sub flows.

5.1.1.3 Floorplanning

In the schematic-driven mixed signal flow, most of the floorplanning, in particular the top-level floorplanning is done in the Virtuoso environment, using the top down floorplanning methodology. In this flow, some or all of the sub-blocks will be created as soft blocks, as they have not been fully implemented yet.

5.1.1.4 Block Implementation Sub Flows

When it comes to physically implementing the blocks, different sub flows will be used depending on the type of block being implemented. Designs are typically partitioned into analog and digital functions. For a digital design, the block Verilog netlist, physical abstracts of standard cells and other associated files such as timing constraints are passed to Innovus. After the digital block has been implemented in Innovus, the layout view of the block is brought back into Virtuoso to replace the autoAbstract which was earlier created for the block. It is important to note that a schematic is not required for the digital blocks in the design.

5.1.1.5 Top-Level Implementation

This task involves making any final adjustments to the floorplan (in Virtuoso) now that blocks are fully implemented, routing the design and verifying all aspects of the design, which may include timing. An important note is that power connections in the design are managed using the inherited connections in Virtuoso, or using explicit power terminals across the hierarchy.

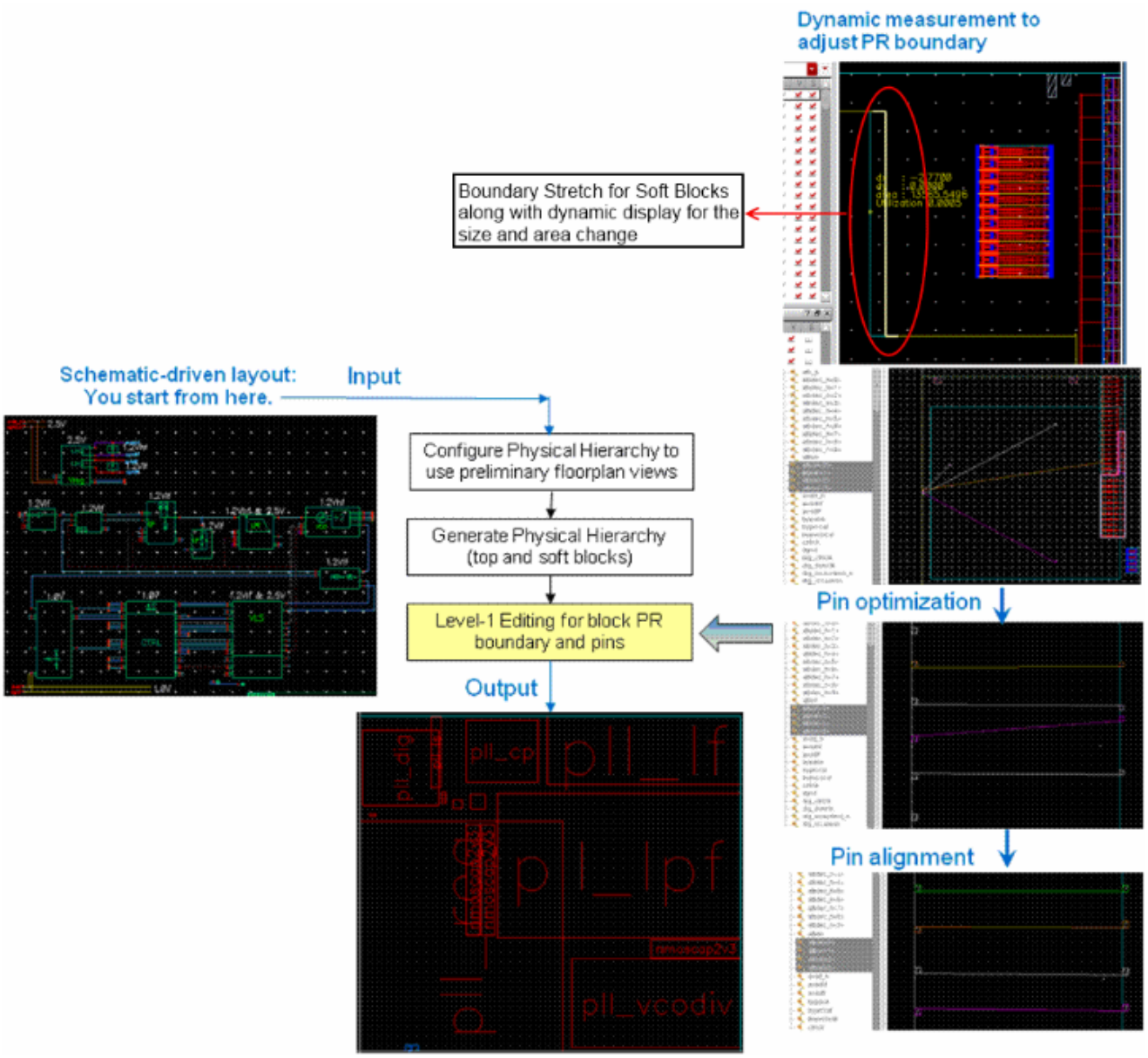
5.1.2 Top-Level Early Floorplanning

The common floorplanning tasks are:

- Configuring the physical hierarchy: This step creates what is called the physical configuration view (or `physConfig`). This identifies the mapping between schematic components and layout components. It is possible for the layout views to be located in other libraries. For blocks for which user does not have a layout view, the `prBoundary` can be entered using an estimate for the layout area and the pin information could be entered using the information found on the schematic symbol.
- Placing and arranging all the blocks - adjusting their location and shape. Adjust pin locations

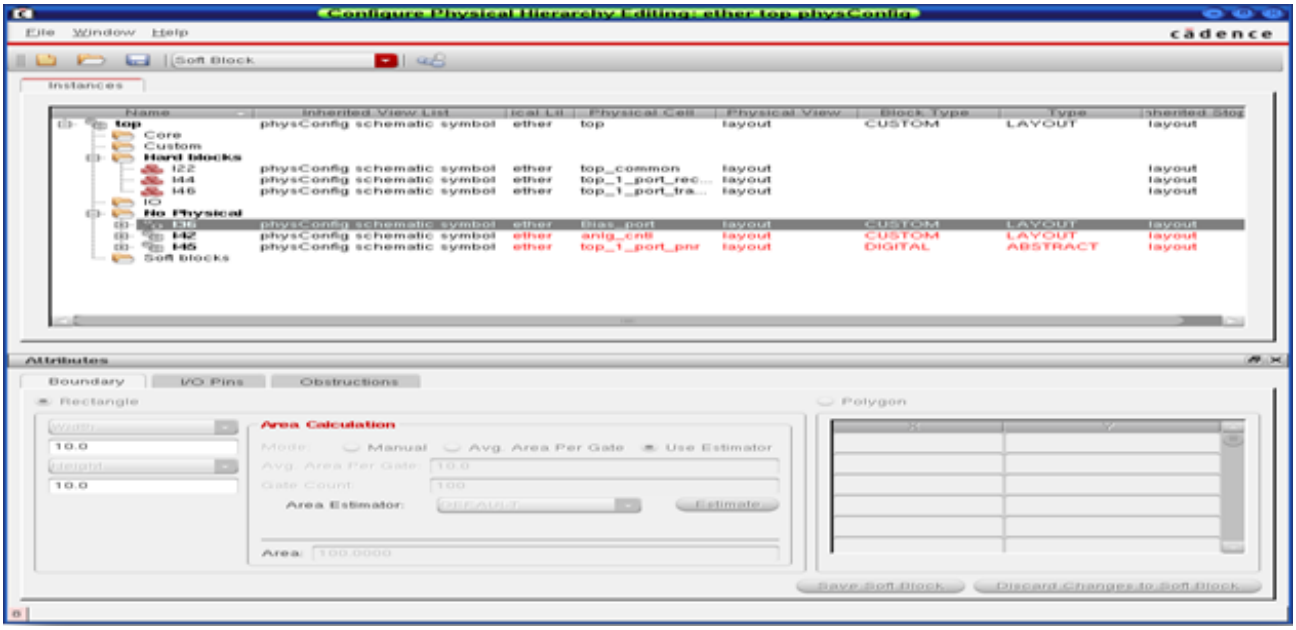
and sizes - taking advantage of the various pin alignment and optimization features available in the Virtuoso Floorplanner. You can also edit the block shape by using level-1 editing commands, which allow you to modify the block prBoundary from the top level.

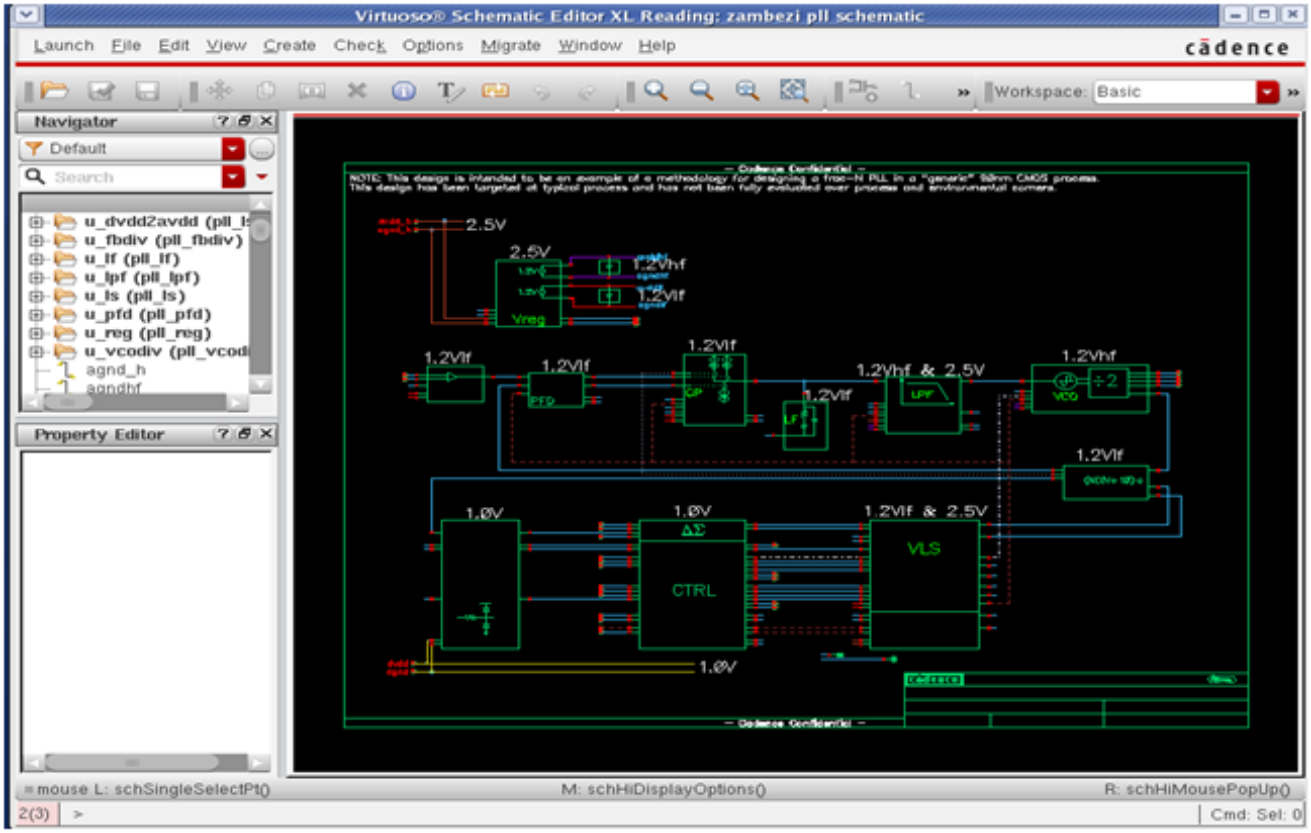
- Iterative floorplan refinement - Using the automated routing tools, perform trial top-level routing to identify congestion and other routing challenges.



5.1.2.1 Configure Physical Hierarchy (CPH)

The CPH is used to create mapping between the schematic and the layout instances. You can create softBlocks for cells that do not have an existing layout or schematic. You can drive the pin and cell boundary snapping to grids by setting the *Block Type* as `DIGITAL` or `CUSTOM`.

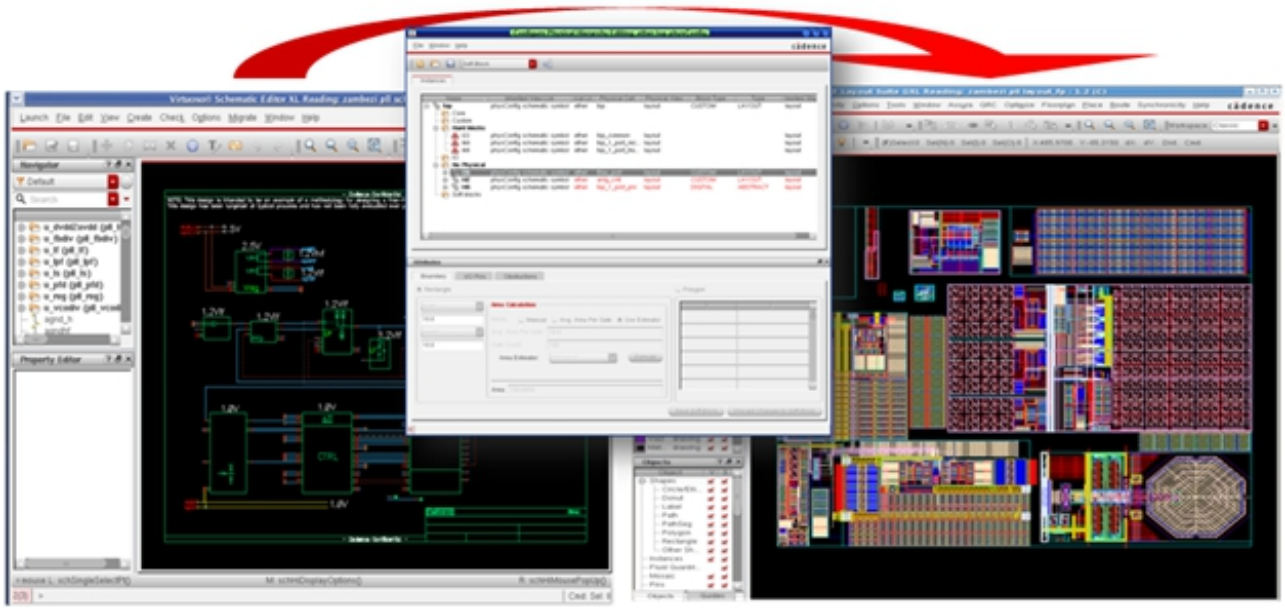




The above figures show how CPH works after launching Virtuoso Layout Editor-XL for a top-level schematic for the first time. CPH gives you control over the automated schematic-to-layout generation process without requiring the layout designer to modify the schematic.

5.1.2.2 Generate Physical Hierarchy (GPH)

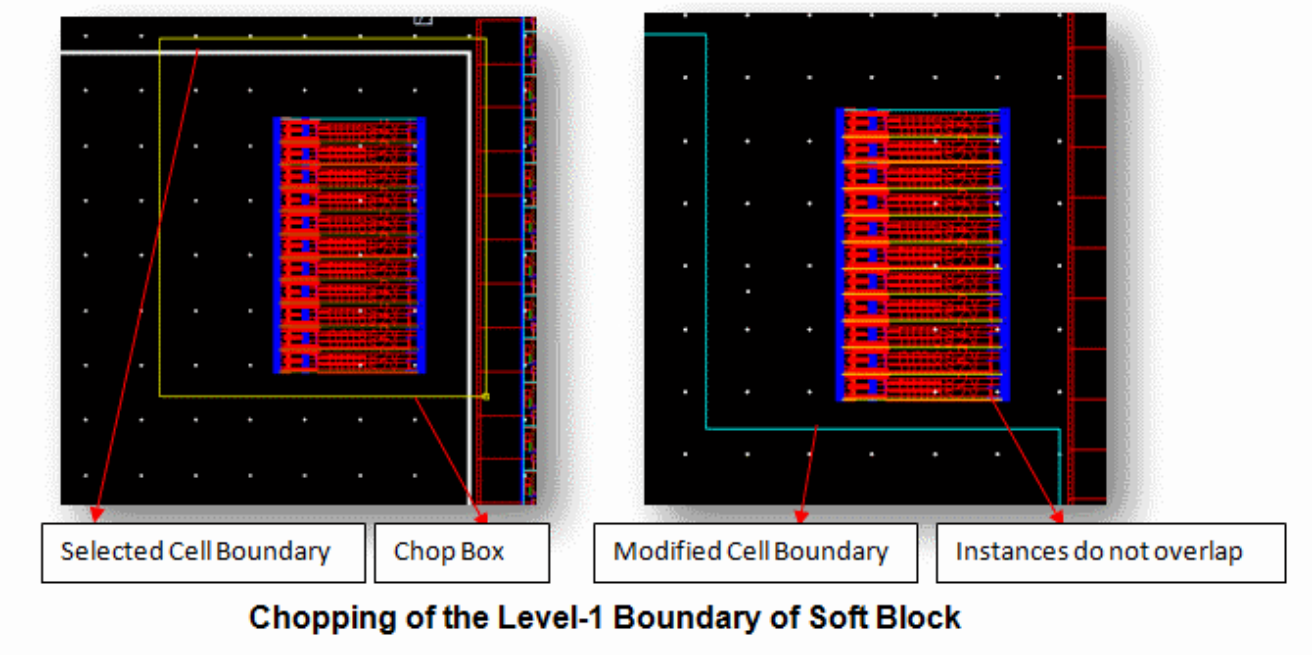
This step in the flow is used to perform bottom-up hierarchical generation of layout in order to have an accurate estimation of area. The following figure illustrates the use of CPH to drive the generation of layout using GPH. GPH generates the layout for the soft blocks and instantiates the hard blocks in the design based on the bindings specified in CPH UI. This enables more accurate estimation of the area at the top level. You can access GPH using the Floorplan menu in Virtuoso Layout Editor -GXL.



5.1.2.3 Level-1 Editing and Pin Optimization

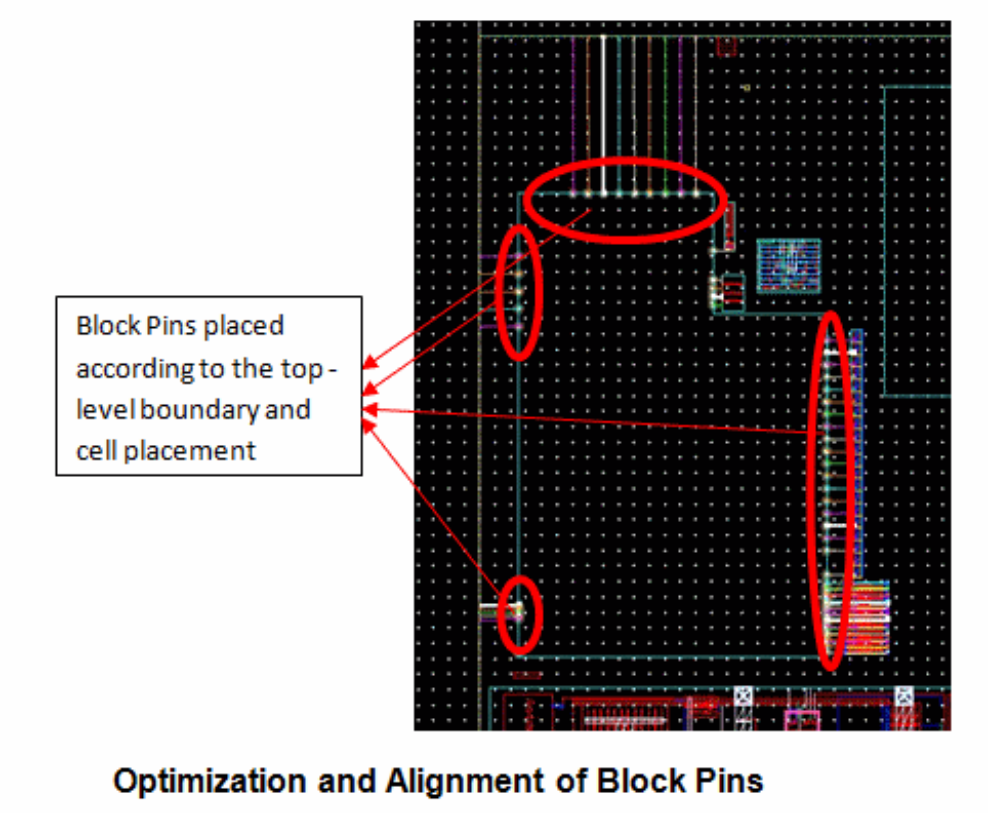
The floorplan for the top level can be adjusted to meet the aspect ratio and the utilization factor using a host of commands from the Virtuoso Floorplanner suite.

The user can adjust and even create rectilinear boundaries for the unimplemented soft blocks by editing the prBoundary for the blocks from the top level to obtain a better packing for the floorplan. The boundaries can be stretched and chopped to achieve the desired area.



Chopping of the Level-1 Boundary of Soft Block

You can use the pin optimizer and the pin alignment commands to increase the routability of the design by adjusting the pin locations for these soft blocks and the top-level design. This helps in minimizing the net length.

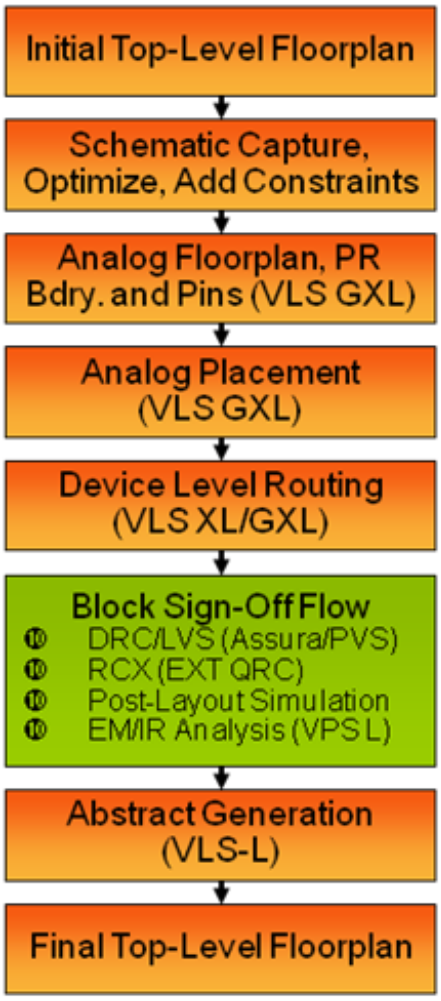


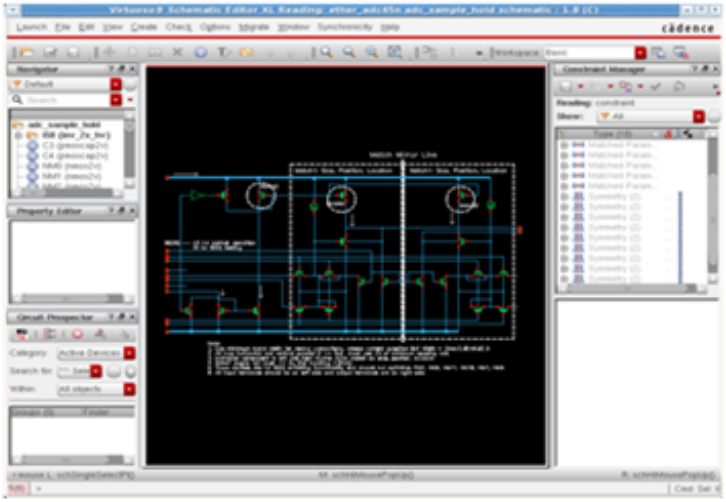
Optimization and Alignment of Block Pins

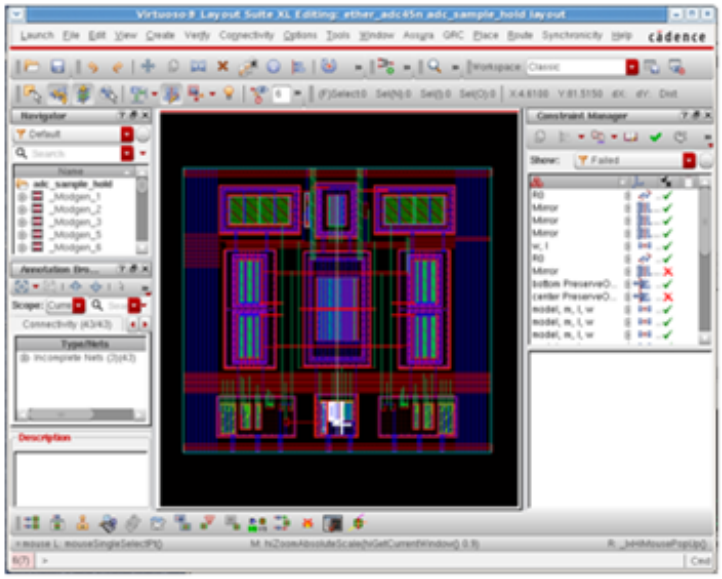
5.1.3 The Analog Block Implementation Sub-Flow

The analog block implementation flow is the traditional flow that most Virtuoso designers use. This section provides an overview of the steps involved in this sub-flow. For more detailed description of each flow step, refer to the *Virtuoso® Analog Placer User Guide*.









5.1.3.1 Initial Top-Level Floorplan

The assumption in this step of the flow is that the block being floorplanned is the top-level design for this sub-flow. The information derived from the top-level floorplanning (initial area estimate, width, height and the I/O pin locations) step is used in this step of the design. Although this data is just an estimate, it serves as a good start point for the flow. The real data is only known after circuit design is completed.

5.1.3.2 Schematic Capture

At this step in the flow, the circuit design is done based on the block-level specification. This is the task of capturing the schematic and making sure it is optimized to meet the specification. During this process, you can add some electrical constraints that would later get used for circuit optimization, such as matching parameter constraints, and so on. Before the design is handed off to the layout design team, layout constraints, such as placement and routing constraints, can also be added to the design.

5.1.3.3 Analog Floorplan (Also Referred to as the Cell Plan)

In this step, the initial floorplan is used to begin the layout implementation flow. Generate From Source (GFS) is run and the `prBoundary` and pin locations are loaded from the floorplan. After running GFS, the number and the size of the components to be placed are known, and floorplan refinement can also be done.

5.1.3.4 Analog Placement

This is the process of placing components in a design so they respect previously specified constraints. If the constraint-aware editing mode is enabled (default) during component placement, whether the devices are placed interactively or semi-manually, the placement constraints such as modgens, symmetry, alignments and orientation are enforced. For instance, if you place one component from a symmetrical pair, the other will be placed at the same time and the same distance from the shared axis of symmetry.

The Virtuoso Analog Placer (VAP - part of VLS-GXL) can be used to fully automate the component placement using the constraints previously specified by the user. For best results using VAP, there needs to be a sufficient number of constraints in a design.

5.1.3.5 Device-Level Routing

The Virtuoso Space-Based Router (VSR - part of VLS-GXL) can be used for automatic routing of the design. Alternatively, the Wire Editor (available in VLS-XL) which uses the same routing technology as VSR can be used to perform constraint-aware interactive routing.

5.1.3.6 Block-Level Sign-Off

Block-level sign-off is an entire sub flow within itself and consists of the steps necessary to make sure the design is logically correct, can be manufactured, and still meets its performance requirements. This flow is fairly standard and highly dependent both on the design-specific challenges and the foundry requirements. Refer to the Virtuoso documentation for more details on this sub-flow.

5.1.3.7 Abstract Generation

Abstract generation is required for the design assembly part of the flow, which can be considered as the layout finishing step of the flow. The Virtuoso integration of the abstract generator could be used to accomplish it. For more information, refer to the *Virtuoso® Abstract Generator User Guide*.

5.1.3.8 Constraint Verification

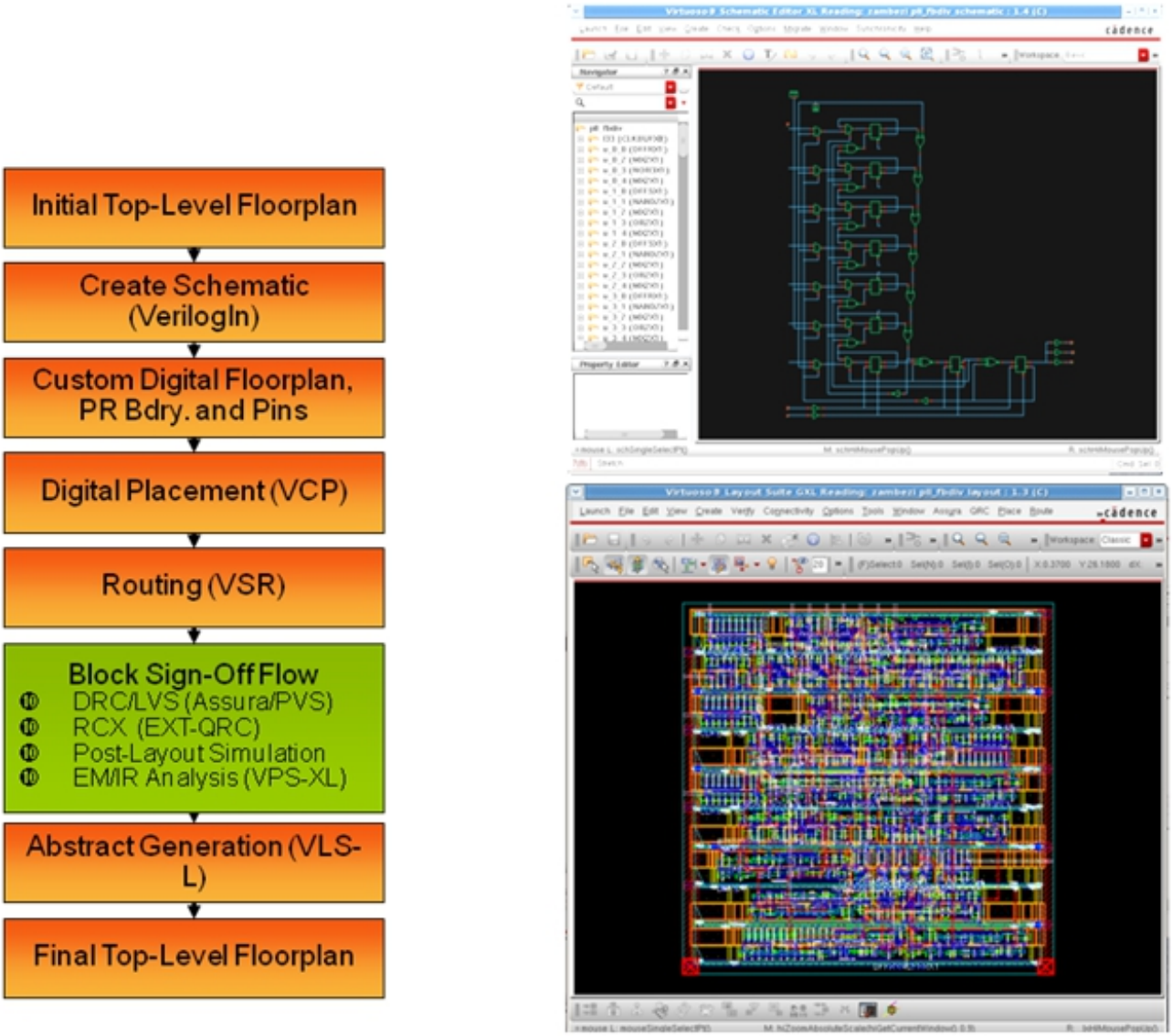
Finally, constraint compliance can be verified using the constraint checking features of the Virtuoso Constraints Manager.

The combination of the automated/assisted and constraint-driven layout features forms what is often referred to as the Rapid Analog Prototyping (RAP) flow. This flow makes it possible to quickly transform a schematic into a layout for the purpose of providing more accurate information for the floorplanning step of the flow and layout parasitics for design optimization/refinement.

5.1.4 Custom Digital Implementation Flow

This flow is very similar to the analog block implementation flow described previously. This flow assumes that the design contains some small, non-timing constrained digital blocks (typically a few hundreds of standard cells) that need to be added to the design. Although the overall function of a particular block may be analog, there is often a need to provide some type of digital interface for control related functions, such as the calibration function, so on.

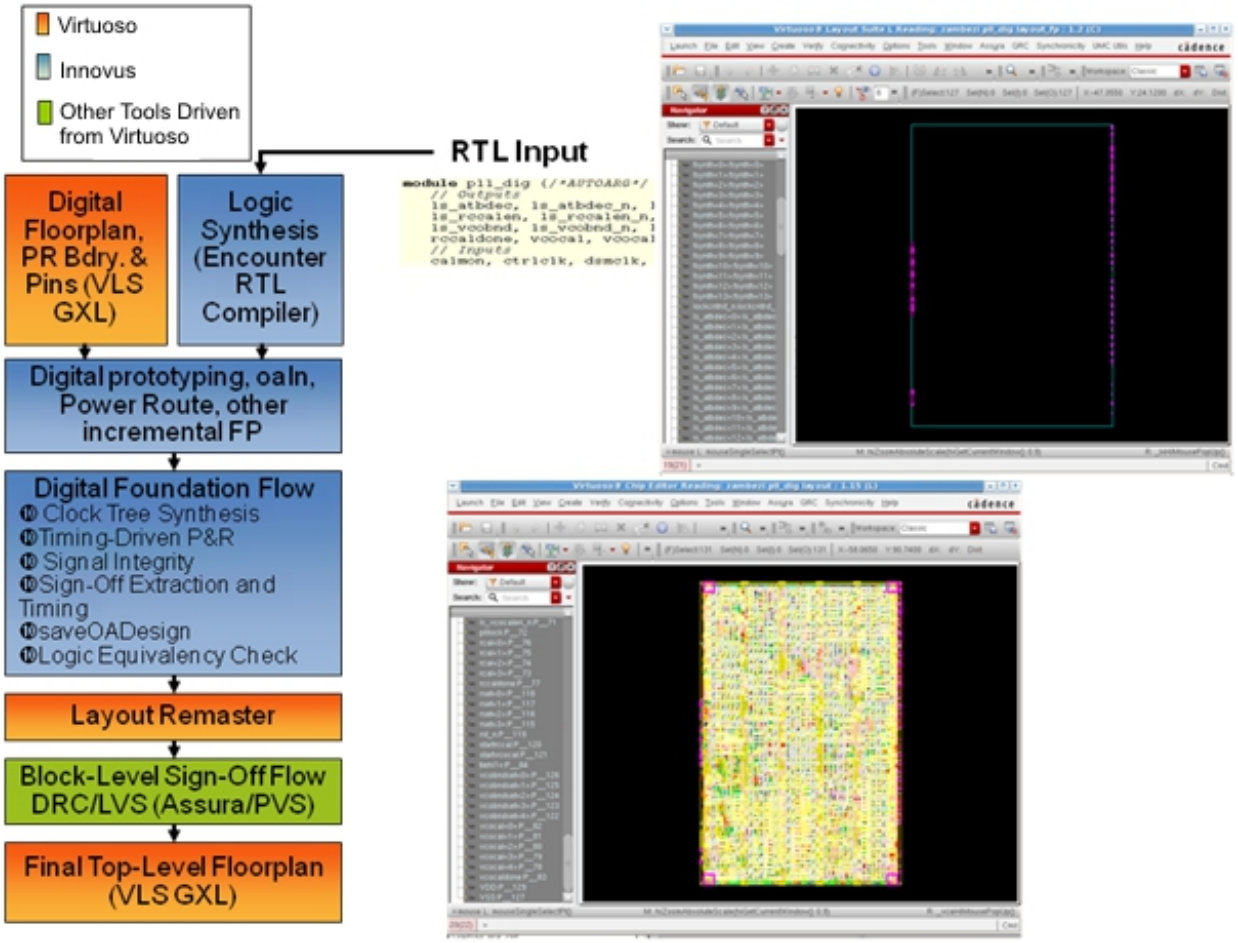
Either a schematic or netlist-driven flow could be used for the implementation of the custom digital function. For the schematic driven approach, user can create a schematic manually. However, if a gate-level netlist is available, the netlist can be synthesized using the digital block implementation flow.



The tasks and tools of this flow are the same as the analog flow except that a different placer, the Virtuoso Custom Placer (VCP - part of VLS-GXL), is used to automate the placement of the digital cells. Automatic VSR routing is done after the completion of placement. This highly automated flow, is similar to the standard digital flow. One of the main differences between this flow and the standard digital implementation flow in Innovus is the lack of a timing-driven implementation approach. This flow demonstrate that it is possible to implement a non-timing constrained digital block in a few minutes without leaving Virtuoso.

5.1.5 Digital Block Implementation Flow

This part of the flow takes advantage of the enhanced interoperability between the Virtuoso and Innovus platforms, and is made possible through the use of the OpenAccess database. Most of these tasks are typically executed by the digital designer. However, the goal is to ensure that the analog (Virtuoso) and digital (Innovus) designers are working from the same database at all times in order to maximize collaboration and reduce the chances of miscommunication. This flow has other advantages such as cross-platform ECOs, design-constraint interoperability, and a host of other advantages for the design teams.



5.1.5.1 Logic Synthesis and Design Initialization

The digital designer will synthesize the gate-level netlist and load the final verilog netlist into the Innovus platform along with the reference OA libraries for the cells. The design will be initialized with a default boundary and core rows.



Default Floorplan Created by the init_design Command

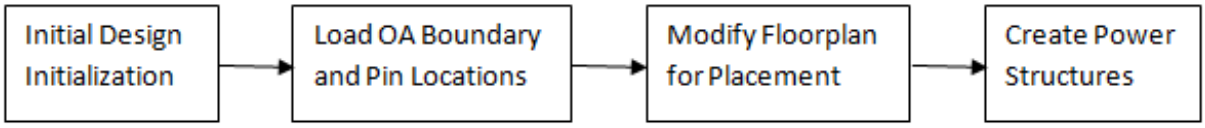
5.1.5.2 Digital Prototyping and Power Planning

While traditionally not considered part of the implementation flow, this portion of the flow allows you to perform incremental floorplanning for the digital portion of the design.

You have a choice to entirely implement the design in Innovus, or import the floorplan for the block (prBoundary and pin assignment) from the work that was earlier done in Virtuoso. Once the floorplan is imported from Virtuoso, it can be further modified in Innovus. For instance, the creation of power structures may require alteration of IO to core spacing. You can also alter the default row style created by the init_design command.

It is strongly recommended that you create the rows in Innovus if you are going to use the standard cell placer in Innovus for placing the cells in the design.

The following flow diagram shows the steps to prepare the OA cell for place and route in Innovus.



Controlling Floorplanning Information Read In from the OpenAccess Cellview

When reading in floorplanning information from the OpenAccess cellview to Innovus, you need not read in the entire cellview. You can use any of the following mechanisms to control the floorplanning information read in from the cellview:

- `oaIn` command in Innovus (without `-filter`): Use the `oaIn` command without `-filter` if you want to read updated placement and routing information with no changes to the netlist. For example, you can use this method if you want to send a placed database to another tool to perform routing and then read back the result. Error checking is performed to make sure that the you are reading from the correct database.
- `oaIn` command with the `-filter` option: Use `oaIn -filter` if you want to read floorplanning information from a cellview whose connectivity does not match the current Innovus in-memory connectivity. The `-filter` option allows you to load some data selectively, while not loading others. When `-filter` is used, error checking for database consistency (nets, instances, terminals, etc.) is disabled. While similar to Virtuoso Load Physical View (LPV), the `-filter` option is only a subset and not intended to be a complete replacement.
  From the Innovus 15.1 release, `oaIn -filter` has been extended to support additional floorplanning objects. The `-filter` option now accepts the following values:
  - `block_insts` - Indicates that any CLASS PAD instances should be updated. Physical-only block instances may be added to the database if the placement status is `fixed` or `cover`.
  - `blockages` - Indicates that blockages should be processed. If a blockage is attached to an instance that does not exist in the in-memory database, then it is ignored.
  - `boundary` - Indicates that the design boundary information, including rows and tracks, should be updated.
  - `fixed_core_insts` - Indicates that any CLASS CORE instances that have placement status `fixed` or `cover` should be updated. Physical-only core instances may be added to the database if the placement status is `fixed` or `cover`.
  - `floorplan` - Is equivalent to specifying "`block_insts blockages boundary fixed_core_insts pad_insts pin_shapes regions special_routing`"
  - `pad_insts` - Indicates that any CLASS PAD instances should be updated. Physical-only

pad instances may be added to the database if the placement status is `fixed` or `cover`.

- `pin_shapes` - Indicates that pin shapes should be read.
- `regions` - Indicates regions should be updated.
- `regular_routing` - Indicates that nets should be processed and regular routing (and associated shield net wiring) and net constraints updated.
- `special_routing` - Indicates that nets should be processed and special routing (except shield nets wiring) should be read.

When `-filter` is not specified, all information is read from the OpenAccess cellview. When `-filter` is specified, only the object types that are chosen is read from the OpenAccess cellview. Additionally, when reading pin_shapes or instances, information in the cellview that refers to terminals or instances that do not exist in memory is ignored.

- `ecoOaDesign`: Use `ecoOaDesign` if you want to read a new netlist and as much of the original cellview's information as possible. This method applies to a timing eco or post-mask eco flow. In these cases, the amount of change to the netlist is typically small and the goal is to disturb as little of the original information as possible for timing closure or "frozen-metal" eco purposes.

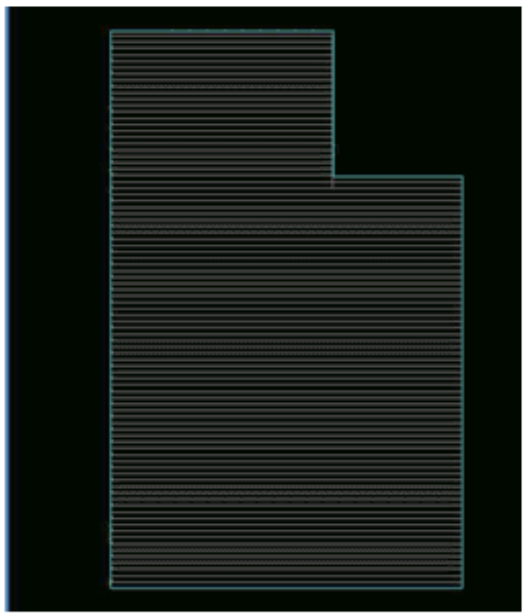
Incremental Floorplanning

As the prBoundary brought over from Virtuoso includes the IO to core spacing, you will need to update the IO to core spacing so that structures such as core rings can be created. This is shown in later steps in this document.

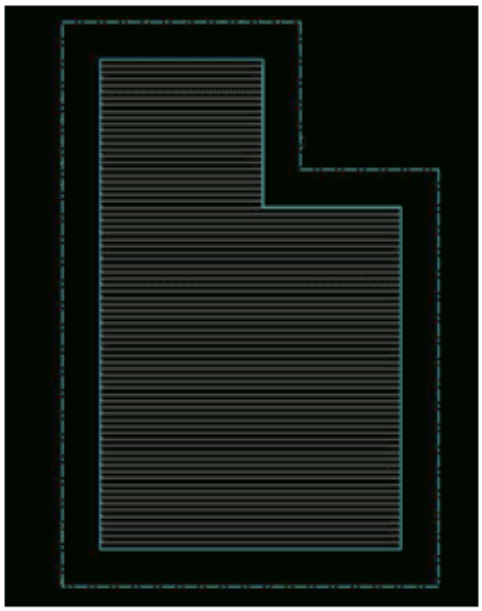
Note: Do not use the *Specify Floorplan* command because it is not an incremental command and will reset the boundary shape to rectangular in case the prBoundary is rectilinear. It will also reinitialize the pin and pad placements. Instead, use the following command to update the IO to core spacing:

```
changeFloorplan
[-noSnapToGrid]
[[-coreToLeft value][-coreToBottom value][-coreToRight value][-coreToTop value] |
[-coreToEdge {left bottom right top}]]
```

This will preserve the pin and pad placements done on the rectilinear edges of the boundary while keeping the prBoundary structure intact.



Boundary Size and Pin Locations from oaIn



Core to IO Spacings with Pin Locations Intact

To change the row style and size, you can use the commands `cutRow` and `createRow`.

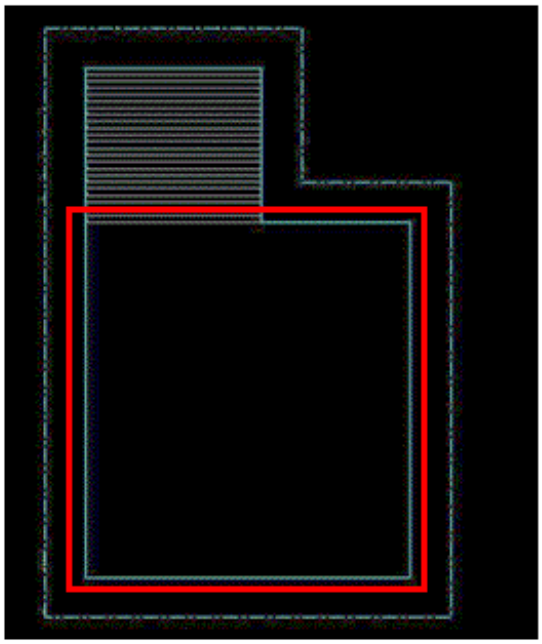
The `cutRow` command allows you to delete the site rows that intersect with the specified area.

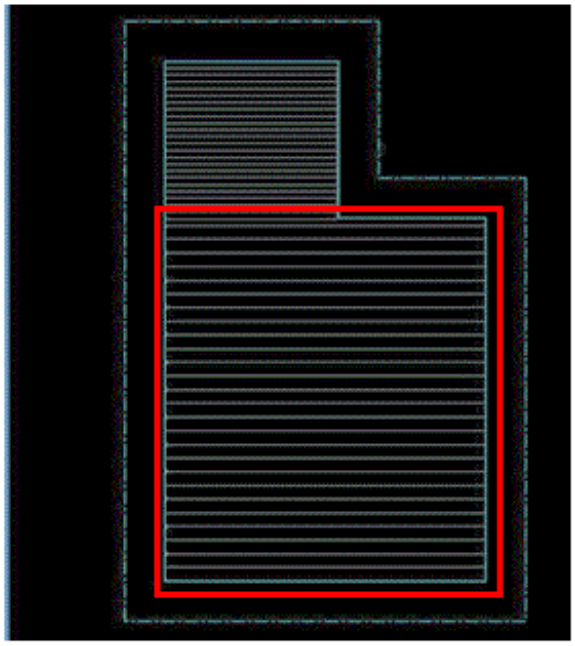
```
cutRow
[-area box | -object string | -selected]
[-halo float | {-leftGap float | -rightGap float | -topGap float | -bottomGap float}]
[-site siteName]
[-keepCell]
```

The `createRow` command then allows you to specify the core rows as per your requirement.

```
createRow
-site siteName
[-area {x1 y1 x2 y2}]
[-spacing distance]
[-noAbut | -noAbut1st]
[-flip1st]
[-noFlip]
```

`cutRow` to remove unwanted rows

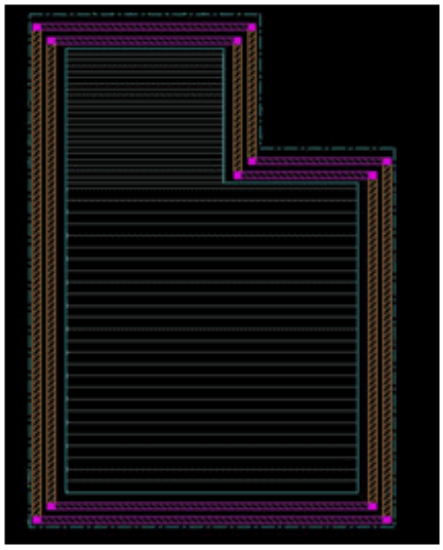




New rows created by `createRow`

Power Routing

After the floorplan for the block has been fixed, the power routing for the block can be performed. The space between the IO and the core region created can be used to create a power ring structure or you might choose to create stripes or custom power topology as per the design specifications.



Ring Power structure Created Around the Block

5.1.5.3 Digital Foundation Flow for Block Implementation

The digital foundation flows are a collection of scripted flows are delivered as a part of Innovus. There are different flows to address different types of design challenges such as hierarchical design, low power and so on. There is also a basic full flat implementation flow.

This Innovus based flow consists of all of the tasks listed below:

Clock tree synthesis, timing-driven logic optimization, timing-driven placement and routing, signal integrity analysis and fixing (crosstalk), sign-off extraction and timing analysis, logic equivalency checks using Conformal, and saving the design into OpenAccess. Timing analysis refers to the process of running Static Timing Analysis (STA) to validate the overall timing of the design, based on the timing constraints created for the digital blocks. For instance, STA checks that signals are available at the input to flops for a pre-defined amount of time before the clock arrives (setup check) or that signals are available long enough at the input to be properly captured (hold check).

The OpenAccess design saved in Innovus can then be opened in Virtuoso.

5.1.5.4 Layout Remaster

The remastering for the digital instance at the top level involves replacing the unimplemented digital block view with the fully implemented view saved in Innovus. However, when the implemented block layout is opened in Virtuoso, you will see only the abstract views instead of seeing the layout masters for each of the standard cell instance. Therefore, you must perform another step in the flow to replace the abstract references with their corresponding layout references (remastering) before the design can be verified in Virtuoso. A menu pick is provided in Virtuoso to make this step quick and simple. It can be accessed in the layout window under *Tools -> Remaster Instances*.

5.1.5.5 Sign-Off Flow

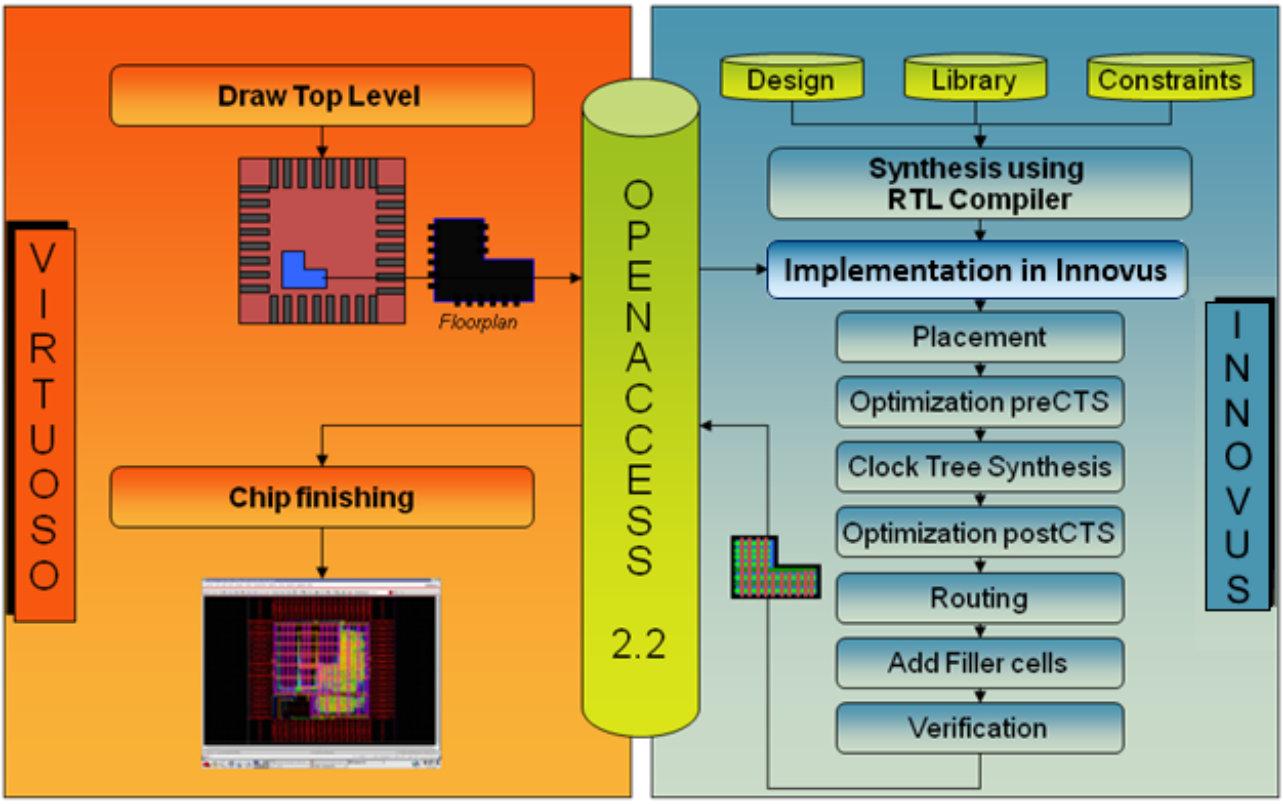
This step in the flow is simpler than for the other sub-block flows because the post-layout electrical verification has already been done in Innovus.

5.1.5.6 Top-Level Floorplan

We now have the final top-level floorplan with the digital block fully implemented.

5.1.6 OpenAccess Based Interoperable Flow between Innovus and Virtuoso

The figure below illustrates the interaction between Virtuoso and Innovus for fully implementing the contents of digital block in a design.



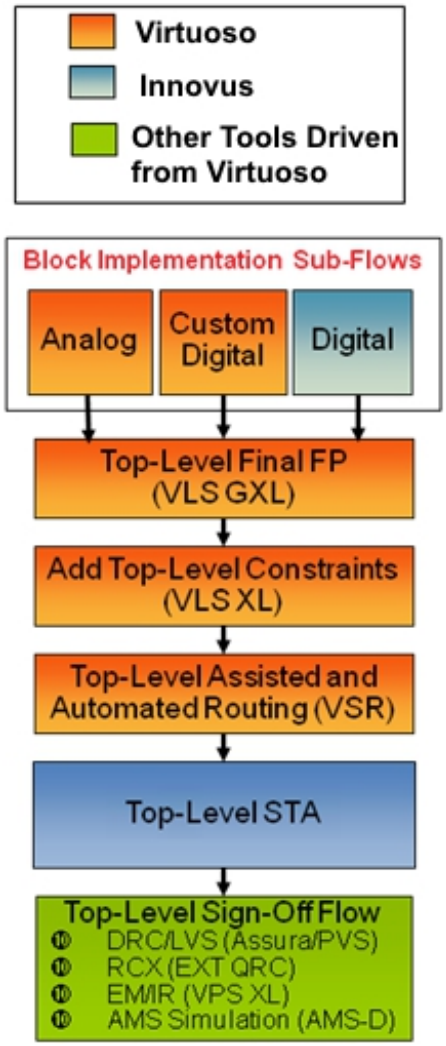
As can be seen from the illustration above, the chip finishing and the final assembly of the design will be done in the Virtuoso cockpit. This step in the flow is another differentiating factor between the schematic-driven mixed signal and netlist-driven mixed signal flows.

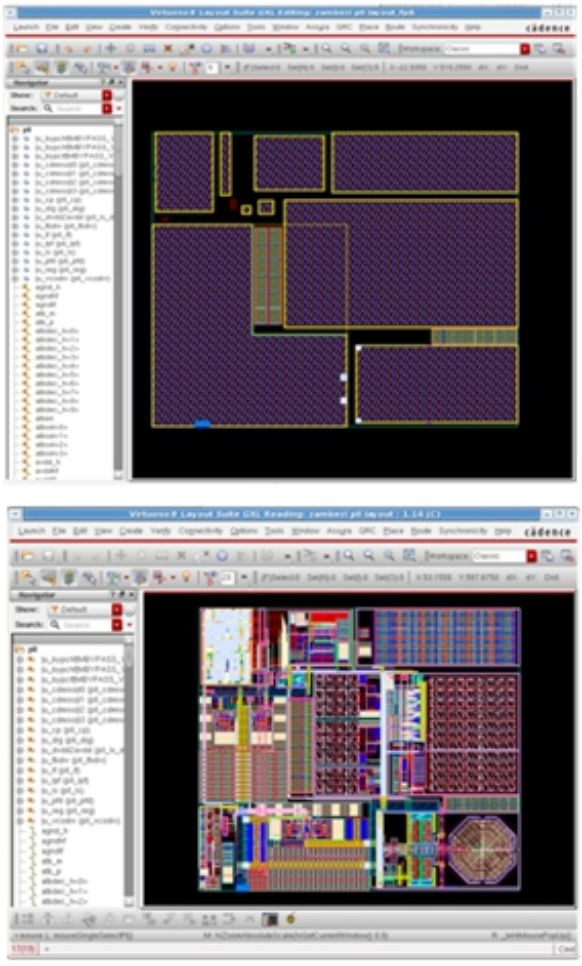
5.1.7 Top-Level Block/Chip Assembly flow

The top-level mixed signal block/chip assembly flow is a continuation of the bottom-up implementation flow. Planning for the top-level assembly typically starts at the beginning of the overall flow, before any of the sub-blocks have been implemented. You might have already performed some trial top-level routing in order to determine routing feasibility and any performance impact to critical signals.

At this stage of the flow, we assume that all of the sub-blocks are in their final form as implemented by their various sub-flows described previously in this document. In other words, the OpenAccess database contains the complete layout and abstracts, any constraints relevant to the top-level

assembly, and so on.





5.1.7.1 Top-Level Final Floorplanning

At this stage, any final adjustments can be made to the top-level floorplan to account for any changes to final block sizes and IOs. This will result in the final design placement.

5.1.7.2 Add Top-Level Constraints

The top-level layout may contain additional constraints. These are typically various routing type constraints. These constraints may be driven from the schematic, added directly to the top-level layout, or automatically or manually propagated upward from lower level blocks using the new hierarchical constraint management features available from the Virtuoso IC 6.1.5 release.

5.1.7.3 VSR Routing

The Virtuoso Space-Based Router (VSR) supports a wide variety of routing constraints and is typically used in a more automated flow when routing the top-level design. However, the interactive constraint-aware wire-editing features in Virtuoso can be used for selective nets.

5.1.7.4 Top-Level STA Task (Optional)

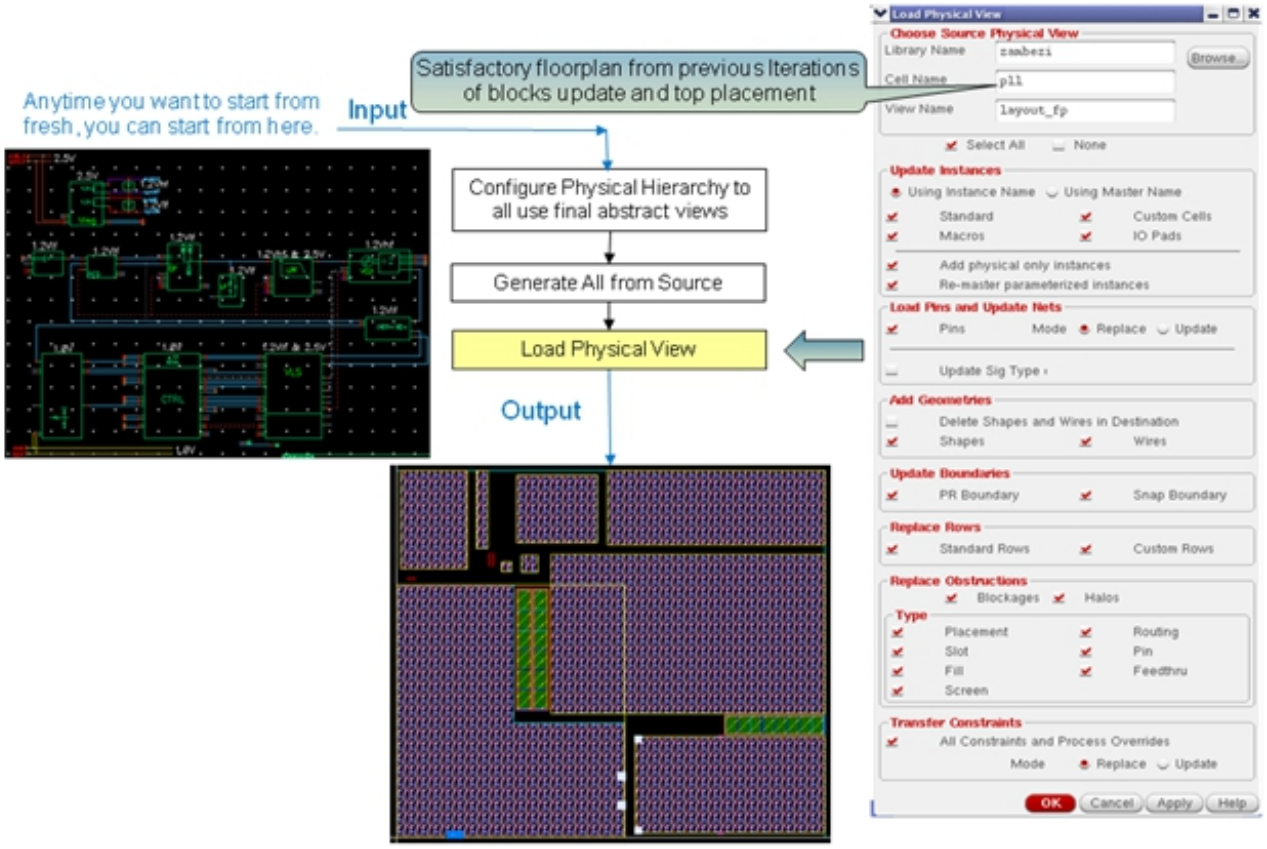
Performing top-level STA requires the existence of a Verilog netlist at the top level which represents the interconnection between the various blocks at the top level. If a Verilog netlist is available, or one can be created by the design team, it is possible to perform STA at the top level assuming that certain blocks have their timing characterized in the `.lib` format used by the STA tool.

5.1.7.5 Top-Level Sign-off

This is a complex and multistep flow that varies depending on the design-specific challenges, target foundry requirements, and so on. It involves a combination of physical verification tasks (DRC, LVS) and electrical verification tasks (MSPS, EM/IR, functional verification). A typical list of tasks are given below. All of these tasks can be driven directly from the Virtuoso environment. The AMS-D simulator is typically used here just as it was used earlier in the design flow. However, the simulations will now be based on extracted layout netlists with parasitics. This simulation can make use of the advanced features of ADE, such Virtuoso Parasitics Aware design (VPAD) to rapidly switch between pre-layout schematic and post-layout extracted representations of the design and debug parasitic-related problems.

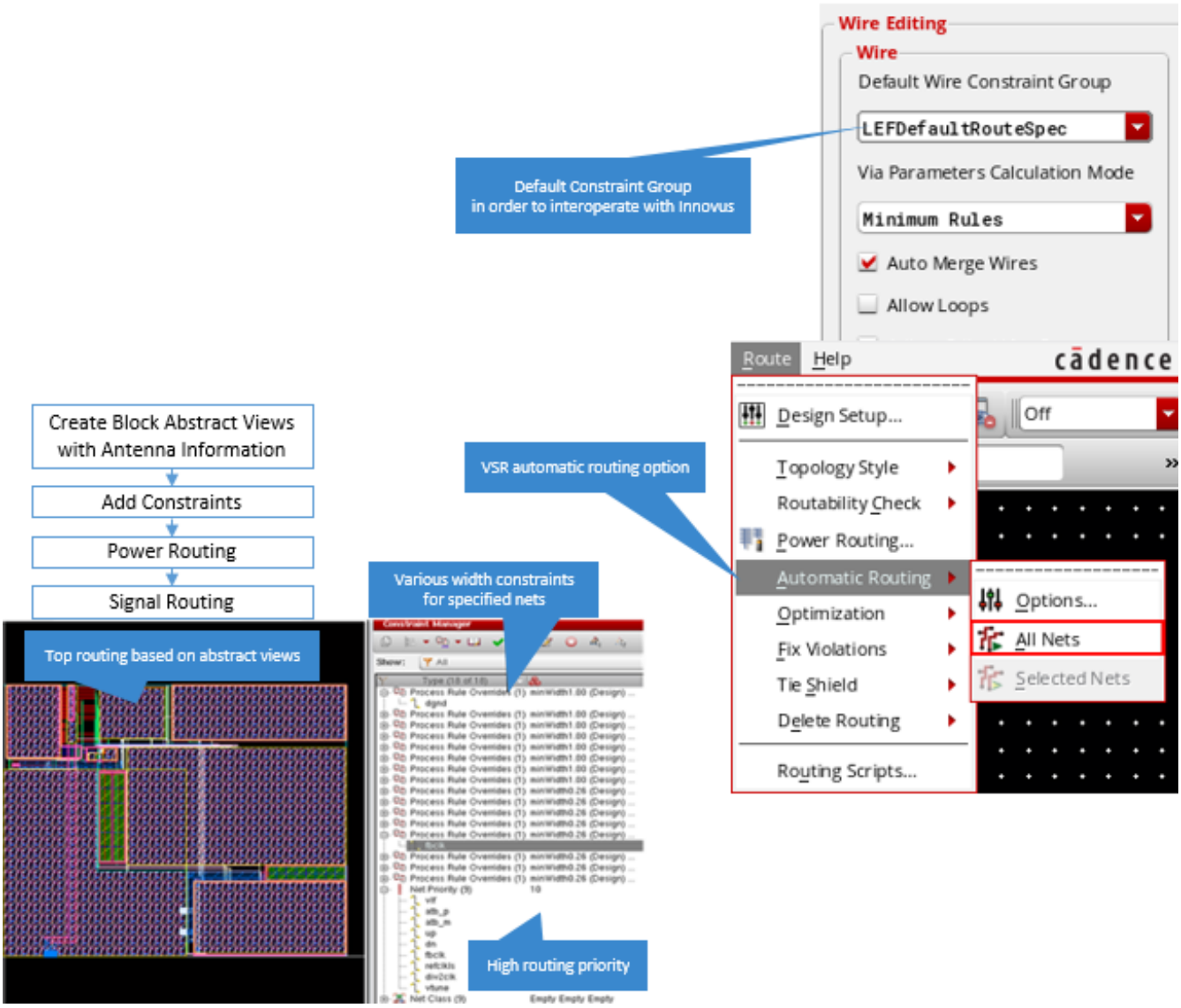
5.1.8 Virtuoso Top-Level Assembly

This figure below illustrates how connectivity and constraint information captured in the schematics representing the top-level of the design is used to create an initial layout using the placement information provided by the floorplan view. At this step, the floorplan is loaded using the Load Physical View (LPV - part of VLS -XL) utility in the Virtuoso layout environment, which will place all of the sub-blocks according to the floorplan. Note that as an intermediate step, you should have configured the physical hierarchy (CPH) to use the abstract masters.



5.1.9 Virtuoso Chip-Level Layout Assembly

The figure below shows a more detailed description of the VSR-based top-level block assembly flow. The routing on the top-level layout is being performed by the VSR engine, using abstracts for each of the sub-blocks, and using the supplied constraints. You can choose to perform VSR routing interactively or by using a more automated approach.


Wire Editing
Wire
Default Wire Constraint Group
LEFDefaultRouteSpec
Via Parameters Calculation Mode
Minimum Rules
Auto Merge Wires
Allow Loops
Default Constraint Group
in order to interoperate with Innovus
Route
Help
cadence
Design Setup...
Topology Style
Routability Check
Power Routing...
Automatic Routing
Optimization
Fix Violations
Tie Shield
Delete Routing
Routing Scripts...
Off
Options...
All Nets
Selected Nets
VSR automatic routing option
Create Block Abstract Views
with Antenna Information
Add Constraints
Power Routing
Signal Routing
Various width constraints
for specified nets
Top routing based on abstract views
High routing priority

5.2 Flow Example: Power PushDown Flow for Digital Blocks in AoT Designs

In the schematic-driven mixed signal flow or the Analog-on-Top (AoT) methodology, the top-level floorplanning is done in the Virtuoso environment. Digital sub-blocks are created as soft blocks because they have not been fully implemented yet. The power shapes created during top-level floorplanning in Virtuoso are pushed down into the digital blocks. The Verilog netlist, physical abstract, and other associated files, such as timing constraints, are then passed to Innovus™ Implementation System for complete physical implementation of the digital blocks. After a digital block has been fully implemented in Innovus, the layout view of the block is brought back into Virtuoso for design assembly. The fully implemented block now replaces the abstract that was earlier created for the block in Virtuoso.

This chapter covers the procedure for implementing a digital block in Innovus with power shapes created during floorplanning in Virtuoso. In the example used in this chapter, the block `pll_dig` is implemented as a digital block. The example illustrates the typical steps for running this flow. You can modify the floorplanning and implementation steps according to your needs.

To implement a digital block in Innovus using the power shapes created during top-level floorplanning in Virtuoso, you need to perform the following basic steps:

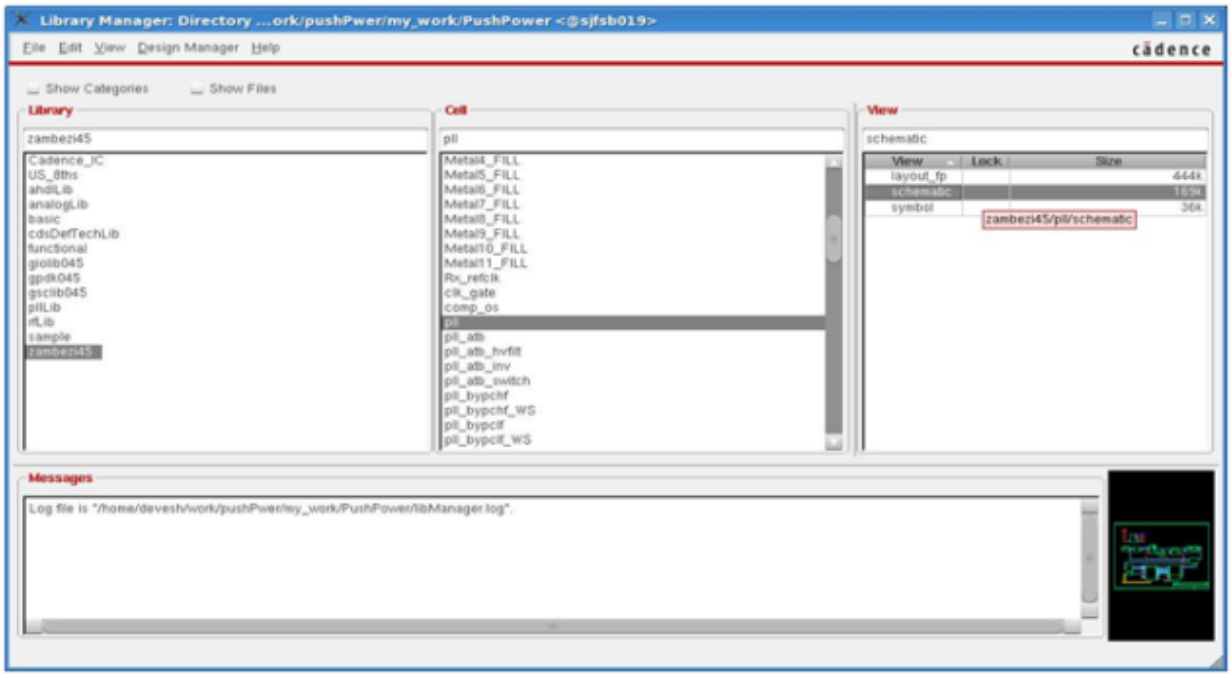
1. Perform top-level floorplanning in Virtuoso.
2. Push power shapes into the digital block.
3. Implement the digital block in Innovus.
4. Bring back the digital block into Virtuoso for design assembly.

Note - The snapshots used in this example are from IC 6.1.6 ISR3 and Innovus 15.10.

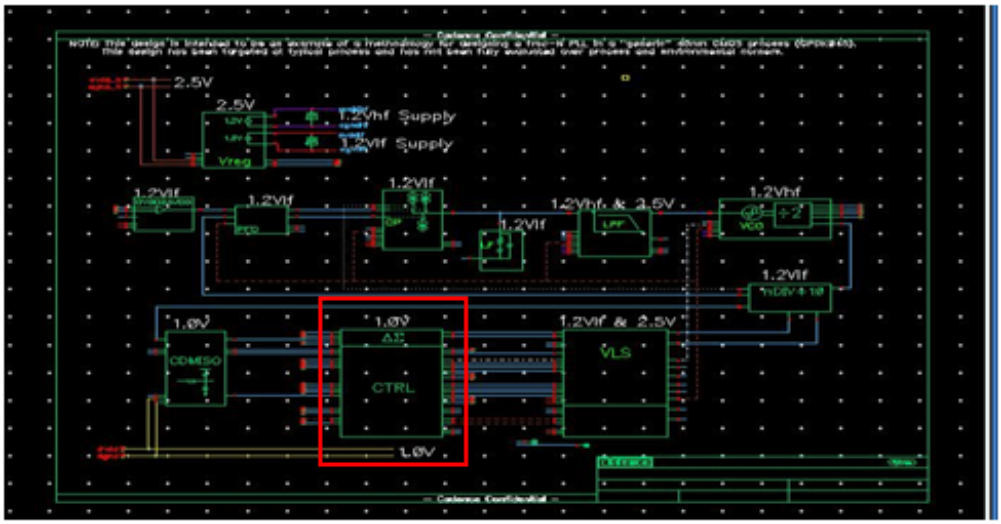
5.2.1 Performing Top-Level Floorplanning in Virtuoso

To perform top-level floorplanning in Virtuoso, perform the following steps:

1. Open the top-level schematic of the design in Virtuoso.
    a. In CIW, select *Tools - Library Manager.*
    b. In the Library Manager form, select the required library, cell, and view (`schematic`) to open the top-level schematic.



The top-level schematic contains a symbol for the block that needs to be implemented as digital block . The block does not have any power or ground pins as shown below. It needs to be supplied power by the top-level digital power and ground nets. The sample top-level schematic below contains a symbol for the `pll_dig` block.



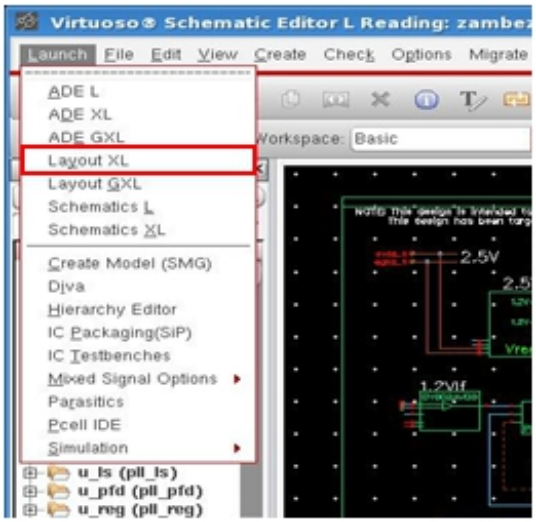
2. View the block-level Common Power Format (CPF) file to get the power and ground names for the digital block.
    a. Open the block-level CPF file in any editor.
    b. Note the power supply net name specified in the file for the digital block. This helps you

determine the block's power pin that will be connected to the top-level digital power net.

c. Note the ground net name specified in the block CPF file. This specifies the ground pin that will be connected to the top-level ground net.

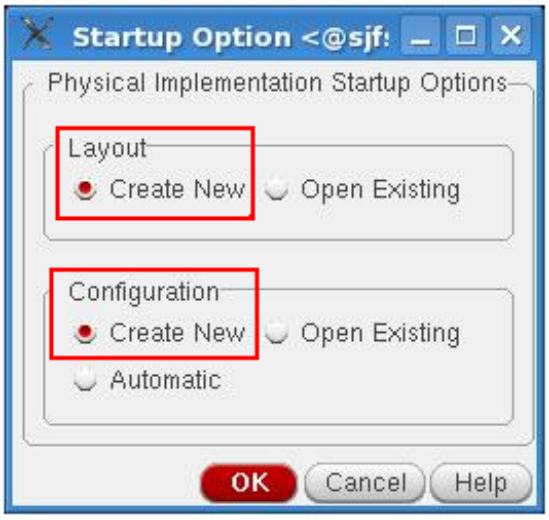
3. Launch VXL to start the creation of the top-level floorplan.

a. In Virtuoso Schematic Editor, select *Launch - Layout XL* from the menu bar.

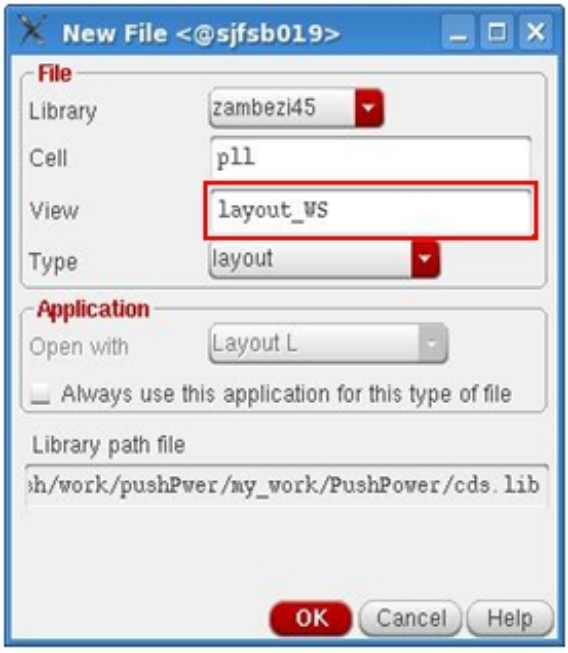


The Startup Options form is displayed.

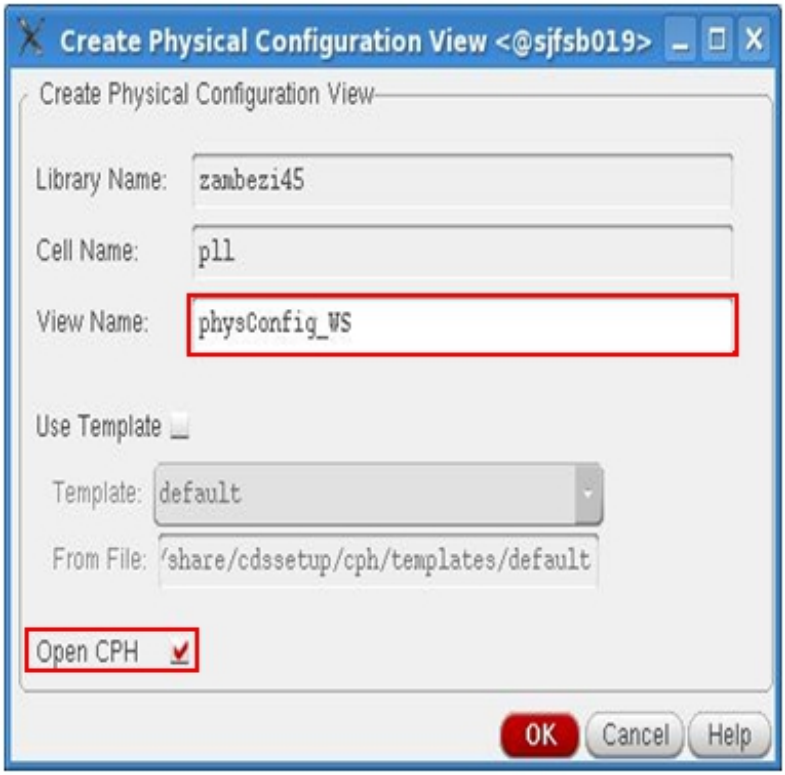
b. In the Startup Options form, select the options for creating new layout and configuration views and click *OK*.



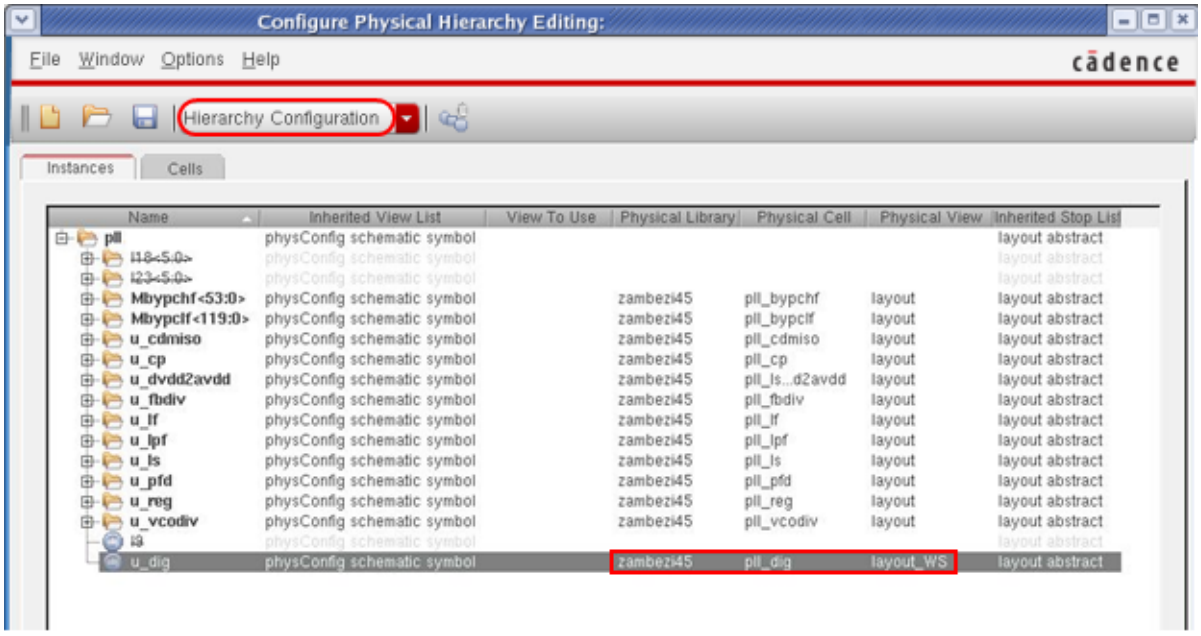
c. In the New File form that is displayed, specify a new name for the layout view, such as `layout_WS`, in the *View* text box and click *OK*.



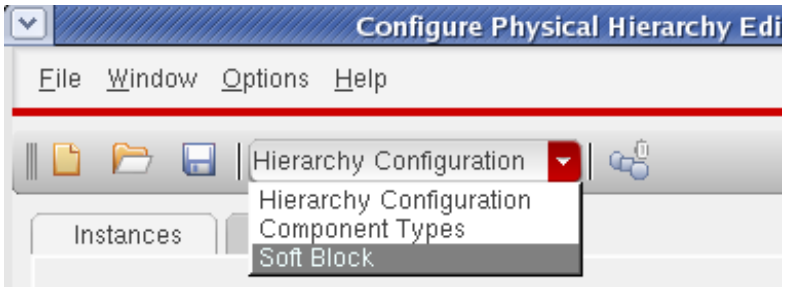
d. In the Create Physical Configuration View form that is displayed next, specify a name for the configuration view in the *View Name* text box and select the *Open CPH* check box to open the Configure Physical Hierarchy form in which you will configure the top-level floorplan. Click *OK* when done.



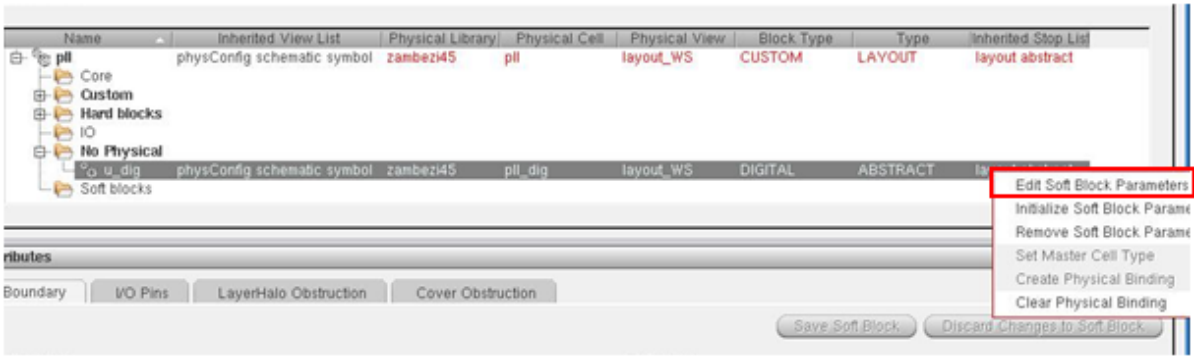
4. Create a top-level floorplan using the Configure Physical Hierarchy (CPH) form. CPH is used to specify the hierarchical bindings between schematic and physical views. It also serves to specify soft block parameters to be generated by Generate Physical Hierarchy (GPH). As the new layout view you specified in the New File form does not exist yet, it will be generated later by GPH.
   As you have only the pin information for the digital block from the symbol at this stage and do not have a schematic available for any hierarchy details, you can only create a blockBlackBox for the digital block. After creating the blockBlackBox for the digital block, you can add power and ground pins as per the net names specified in the CPF file for the digital block.
   a. To create a blockBlackBox for the digital block with an estimated boundary, perform the following steps:
      a. In the default *Hierarchy Configuration* mode, enter the library, cell, and, view name for the digital block in the *Physical Library, Physical Cell,* and *Physical View* columns, respectively. The view name should be the new layout view name you specified in the New File form in the previous step.



      b. Switch to the *Soft Block* mode in the CPH form. The *Soft Block* mode lets you define the soft blocks that will be created by the *Floorplan – Generate Physical Hierarchy command.*

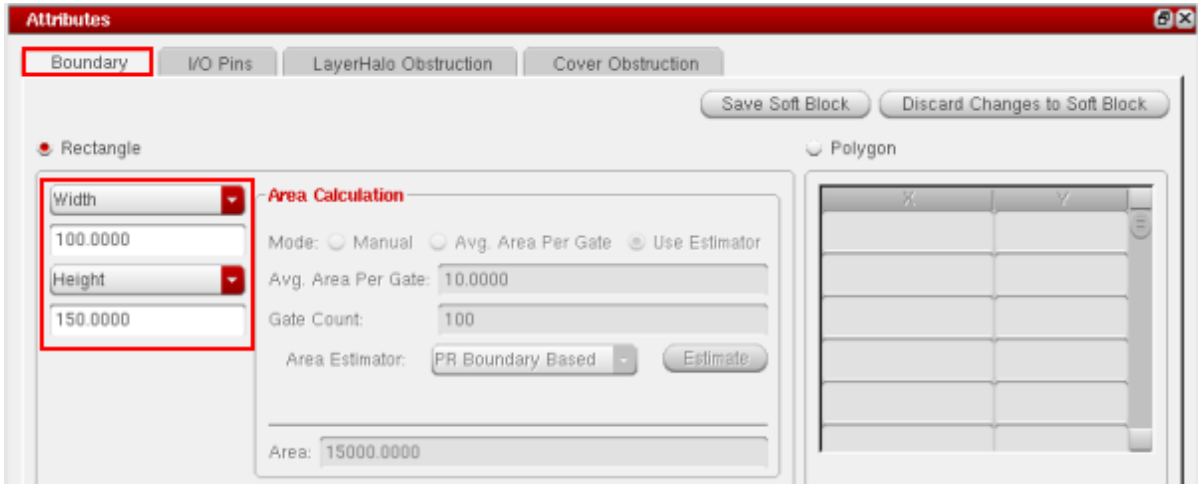


c. In the *Soft Block* mode, expand the *No Physical* list and select the digital block. Next, right-click the block and select *Edit Soft Block Parameters* from the context menu.



This makes the *Attributes* section editable for the selected block.

d. In the *Attributes* section of the CPH form, you can set up the block's boundary as per your estimate. Specify the block's width and height in the *Width* and *Height* text boxes, respectively, on the *Boundary* tab.

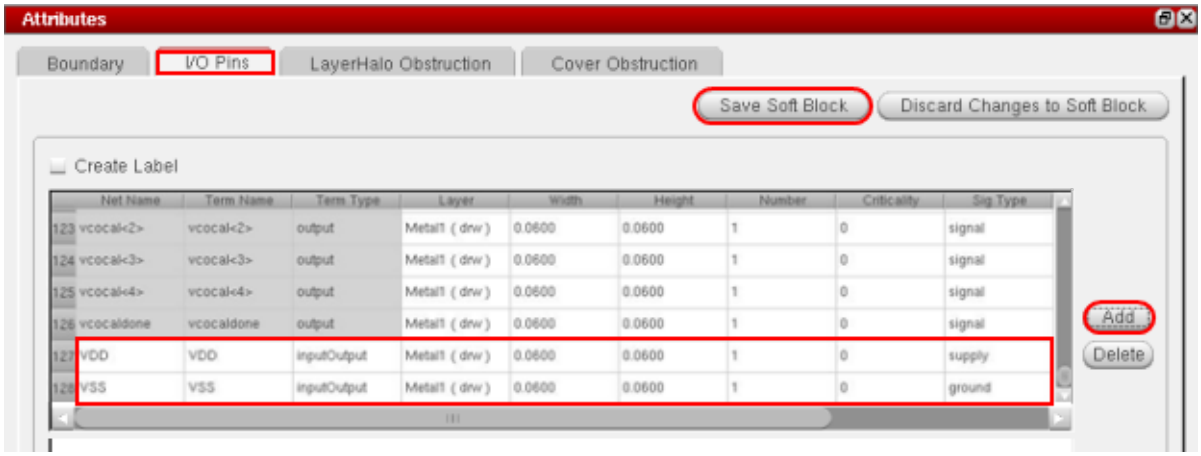


b. To add power and ground pins to the digital block, perform the following steps:

a. On the *I/O Pins* tab in the *Attributes* section of the CPH form, add two pins with the

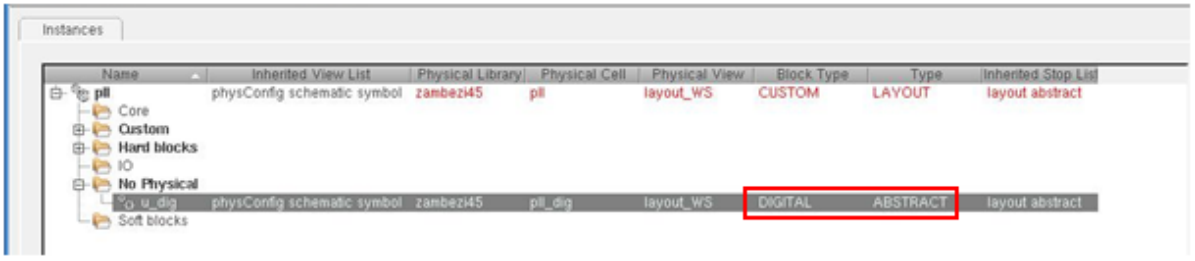
following specifications:

- *Net Name: power_net_name Term Name*: `power_net_name` *Term Type:* `inputOutput` *Sig Type*: `supply`
- *Net Name:* `ground_net_name` *Term Name*: `ground_net_name` *Term Type:* `inputOutput` *Sig Type*: `ground`
  Here, `power_net_name` and `ground_net_name` refer to the power and ground net names identified from the block CPF file in Step 2. In the sample screenshot below, `power_net_name` is `VDD` and `ground_net_name` is `VSS`.

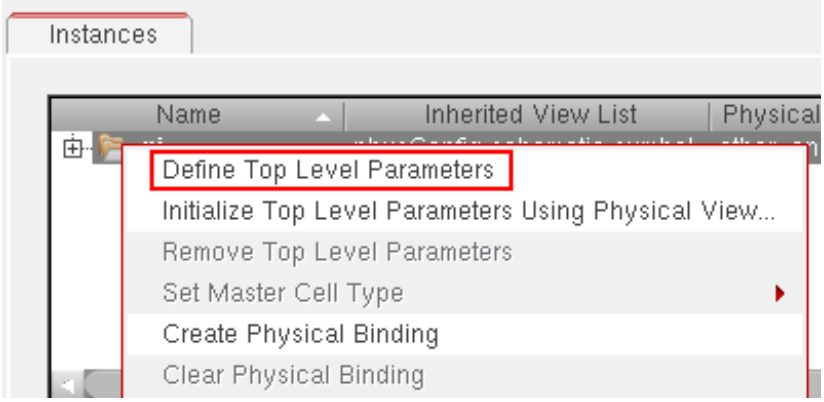


After adding the pin details, click the *Save Soft Block* button.

b. In the *No Physical* list, change the *Block Type* from `Custom` to `Digital`. This changes the snap grid for the pins and boundary to the place & route grid instead of the manufacturing grid. In addition, set *Type* as `Abstract`.



c. Right-click the top-level cell and select *Define Top Level Parameters* from the context menu.



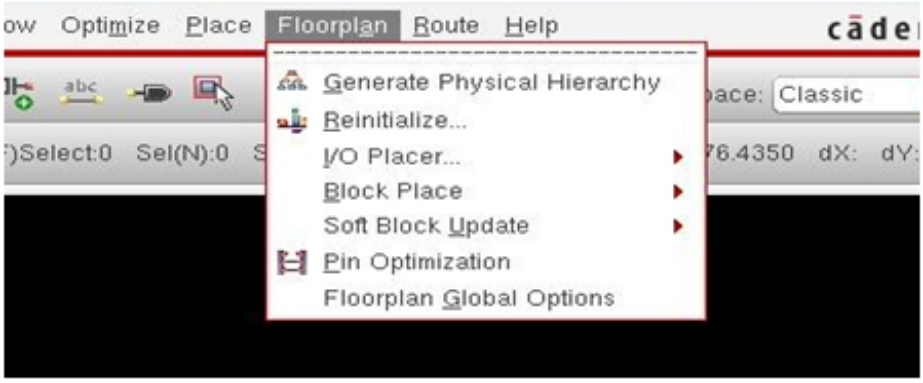
d. Save the top level parameters as is with no change.

Note: As is the case with the Innovus timing analysis flow for mixed signal designs, the top level is flattened using the `assembleDesign` command in Innovus and then the flattened view is saved using the `saveDesign` command. Ensure that the top-level power and ground nets are marked as global in Virtuoso. If not, it can lead to problems while saving the flattened view with `saveDesign` in Innovus. Innovus will report the following error:

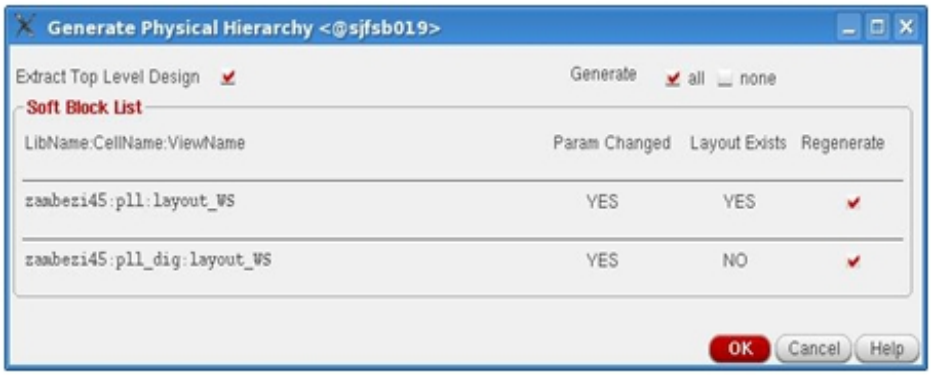
```
**ERROR: (IMPOAX-1085): Cannot override the connection of instance terminal 'SN'
of instance 'I0/preset_int_reg\[5\]' connected to net 'tie1' to global net
'VDD_1P2' as A physical-only instTerm cannot connect to a net located in a
different occurrence. Check if the terminal is connected to a supply or tieHi/Lo
net in the Verilog.
```

The reason for this error is that the block-level power and ground pins are not in the original block symbol (verilog netlist) and are added manually, which causes them to be considered as physical-only pins. The top-level power and ground pins, however, are part of the original logical netlist (schematic) and are hence not marked as physical-only pins. Therefore, after `assembleDesign` and the subsequent `saveDesign`, Innovus tries to connect a physical-only instance terminal with a net that exists in the logical domain, and reports the above error.

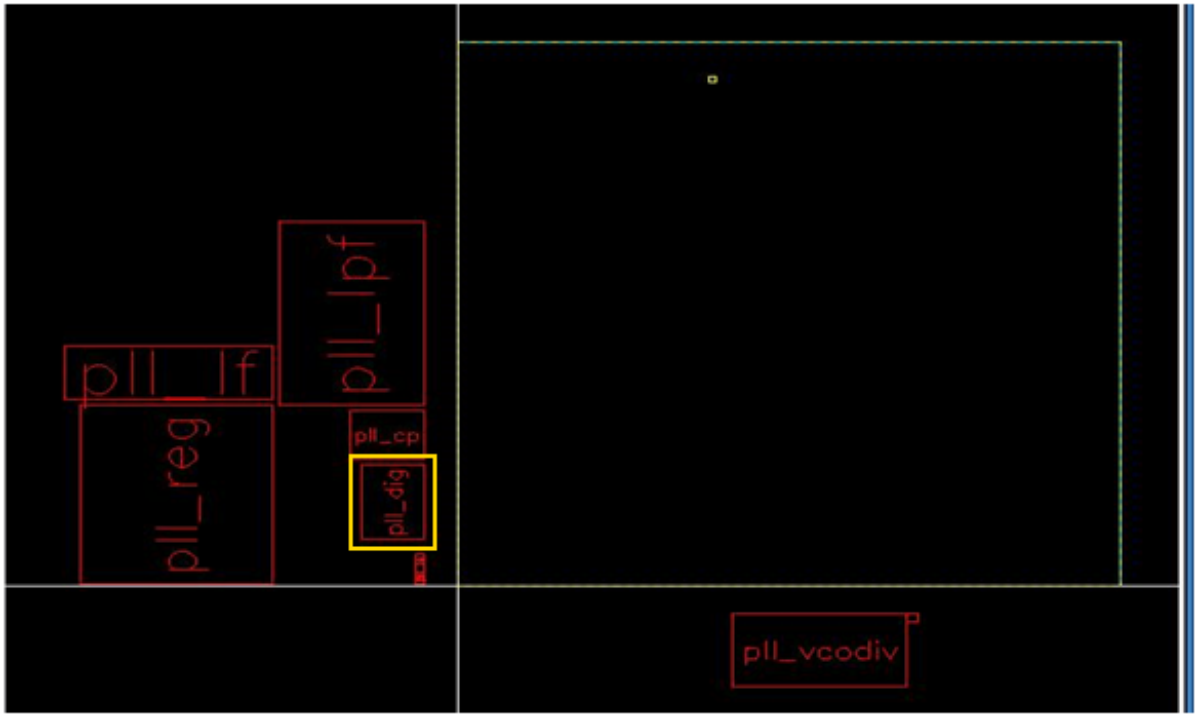
5. Generate the full physical hierarchy.
   a. From the layout, select *Floorplan - Generate Physical Hierarchy*.



b. In the Generate Physical Hierarchy form, click *OK*.



The following layout is created initially and you can see the digital block created:



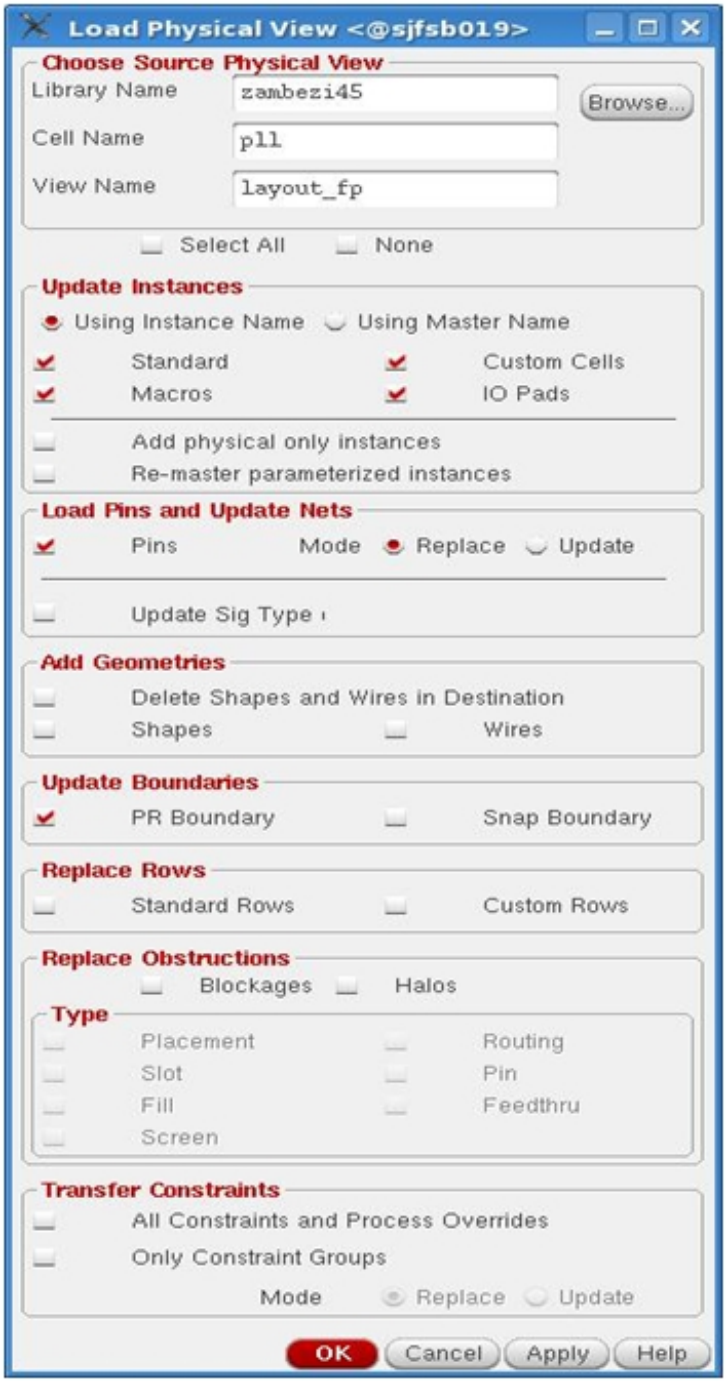
6. Load the placement floorplan. To get the boundary size and the instance and pin locations for

the top-level, use an existing view:

a. From the layout window menu bar, choose *File - Load Physical View*.

b. In the Load Physical View form, specify details of an existing cell view that you want to open.

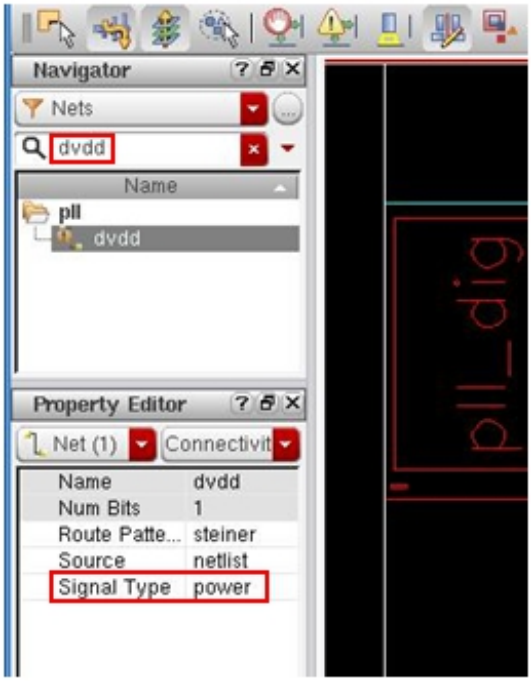
c. The *Select All* check box is selected by default. First, select the *None* check box to deselect all instances. Then, select the instance check boxes *(Standard, Custom Cells,* Macros, and *IO Pads*), *Pins,* and *PR Boundary* check boxes as shown below:



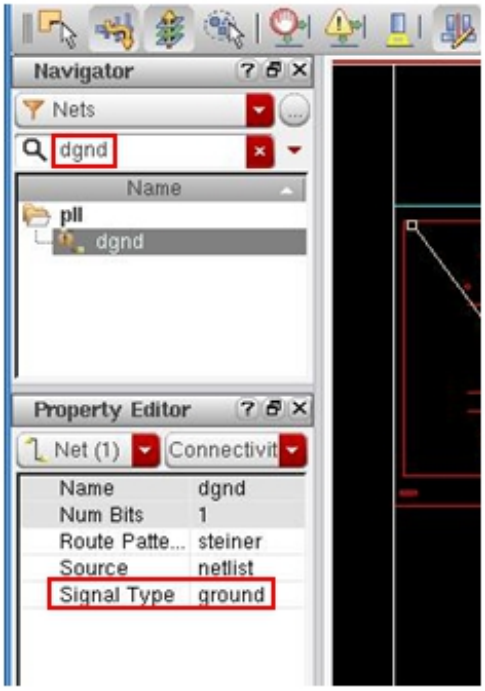
d. Click *OK*.

7. Specify top nets as power and ground:
   a. Invoke the Navigator window by selecting *Window - Assistants - Navigator*.
   b. Invoke Properties Editor by selecting *Window - Assistants - Property Editor*.

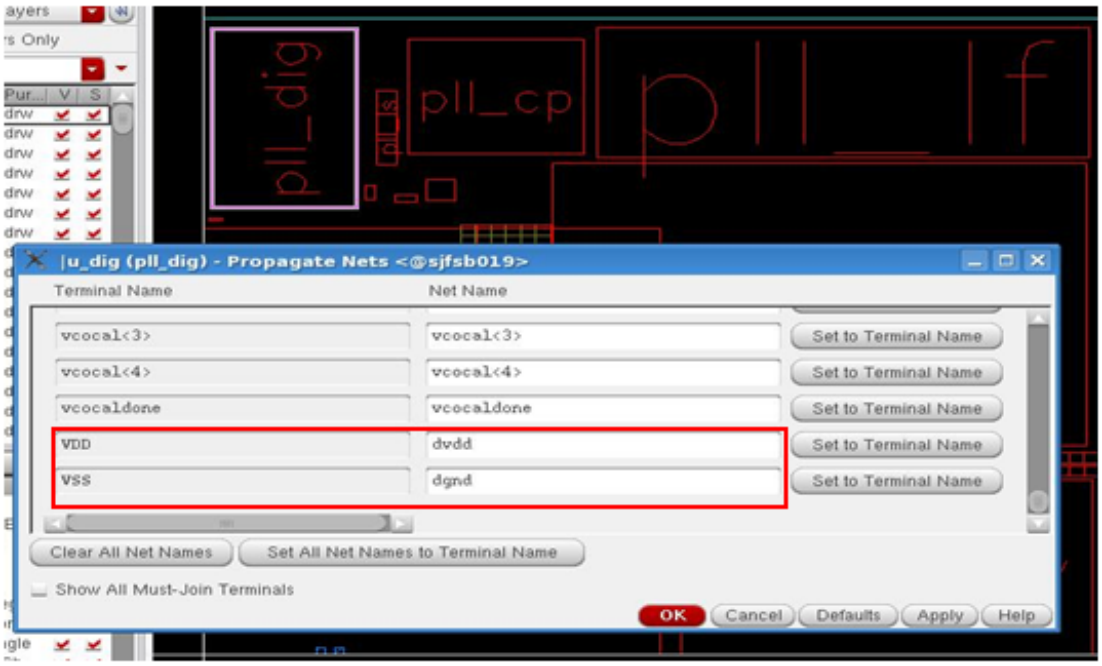
c. In the Navigator window, search for the top-level net that is to be specified as a power net and change S*ignal Type* to `power` for that net in Property Editor. In the example below, top net `dvdd` is specified as a power net.



d. In the Navigator window, search for the top-level net that is to be specified as a ground net and change S*ignal Type* to `ground` for that net in Property Editor. In the example below, top net `dgnd` is specified as a ground net.

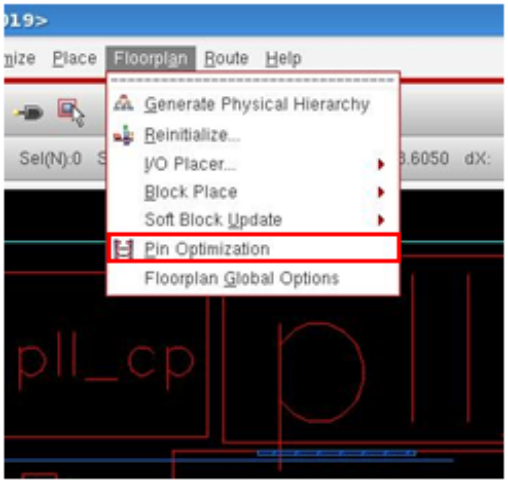


8. Propagate the power and ground nets on the digital cell instTerms as per the top-level nets. The top-level net is connected with the instTerms so that the push command can identify which net to create at the lower level for the top-level power or ground nets.
    a. Select the digital block that you will be implementing later in Innovus.
    b. Select *Connectivity - Nets - Propagate* and scroll to the end of the Propagate Nets form.
    c. Assign the block power terminal to the top-level power net. In the sample screenshot below, the block terminal `VDD` is assigned to top power net, `dvdd`.
    d. Assign the block ground terminal to the top-level ground net. In the sample screenshot below, the block terminal `VSS` is assigned to top ground net, `dgnd`.



    e. Click *OK*.

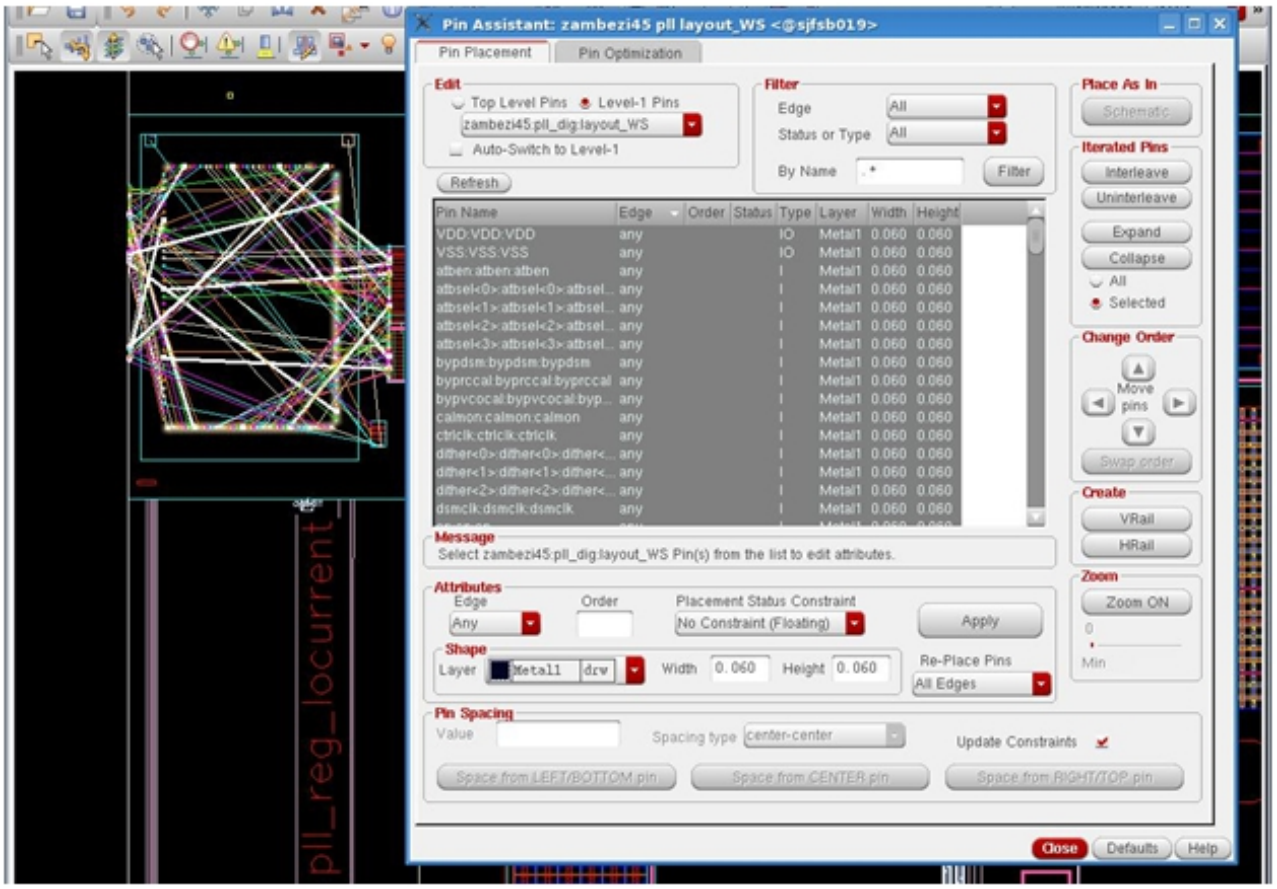
9. Place and optimize the pins for the digital blockBlackBox as per the top-level floorplan:
    a. Select the digital block.
    b. Select *Floorplan - Pin Optimization*.



This displays the Pin Assistant form.

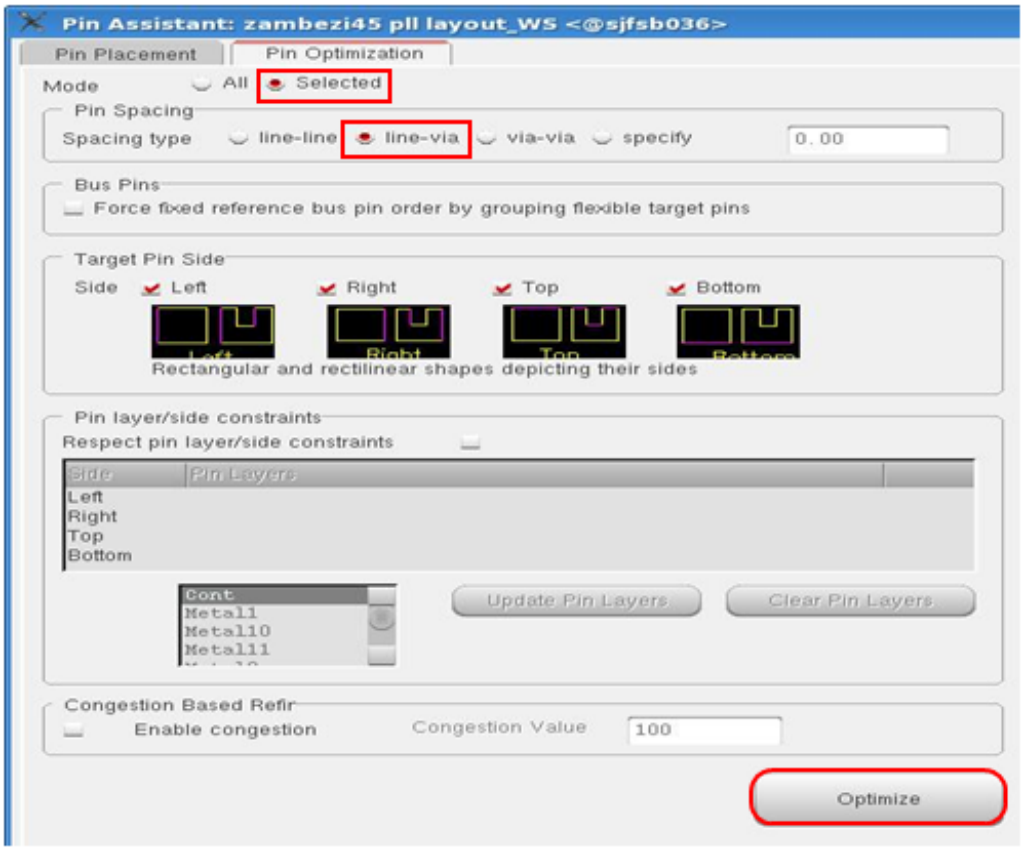
c. On the Pin Placement tab of the Pin Assistant form, and select *Level -1 Pins*.

d. Notice that the digital block is highlighted. Select all the pins for the block in the list box on the *Pin Placement* tab and view the flightlines in the main window.



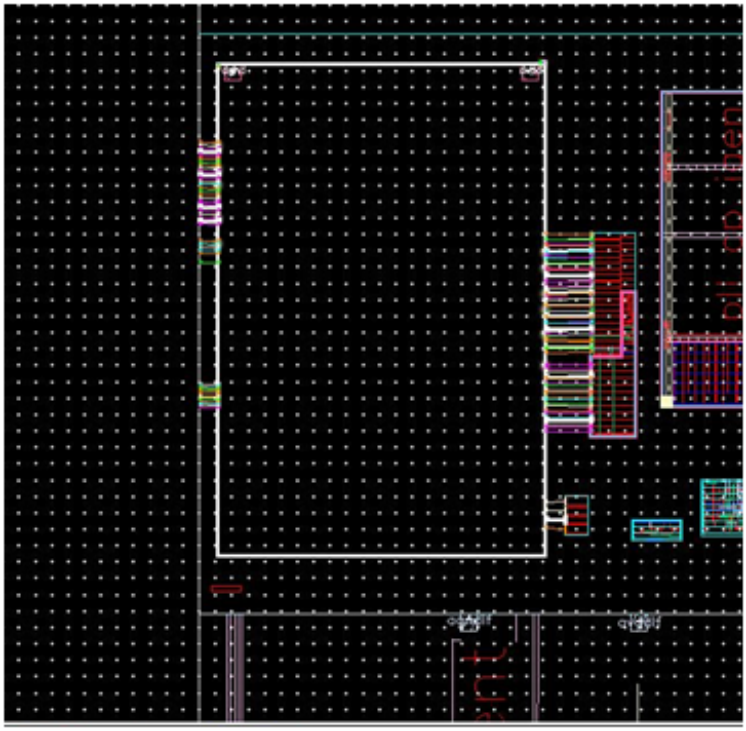
e. Click the *Pin Optimization* tab.

f. Click the *Selected* mode and specify *line-via* as *Spacing Type*.



g. Click the *Optimize* button.

h. After pin optimization of the block, the flightlines look as follows:

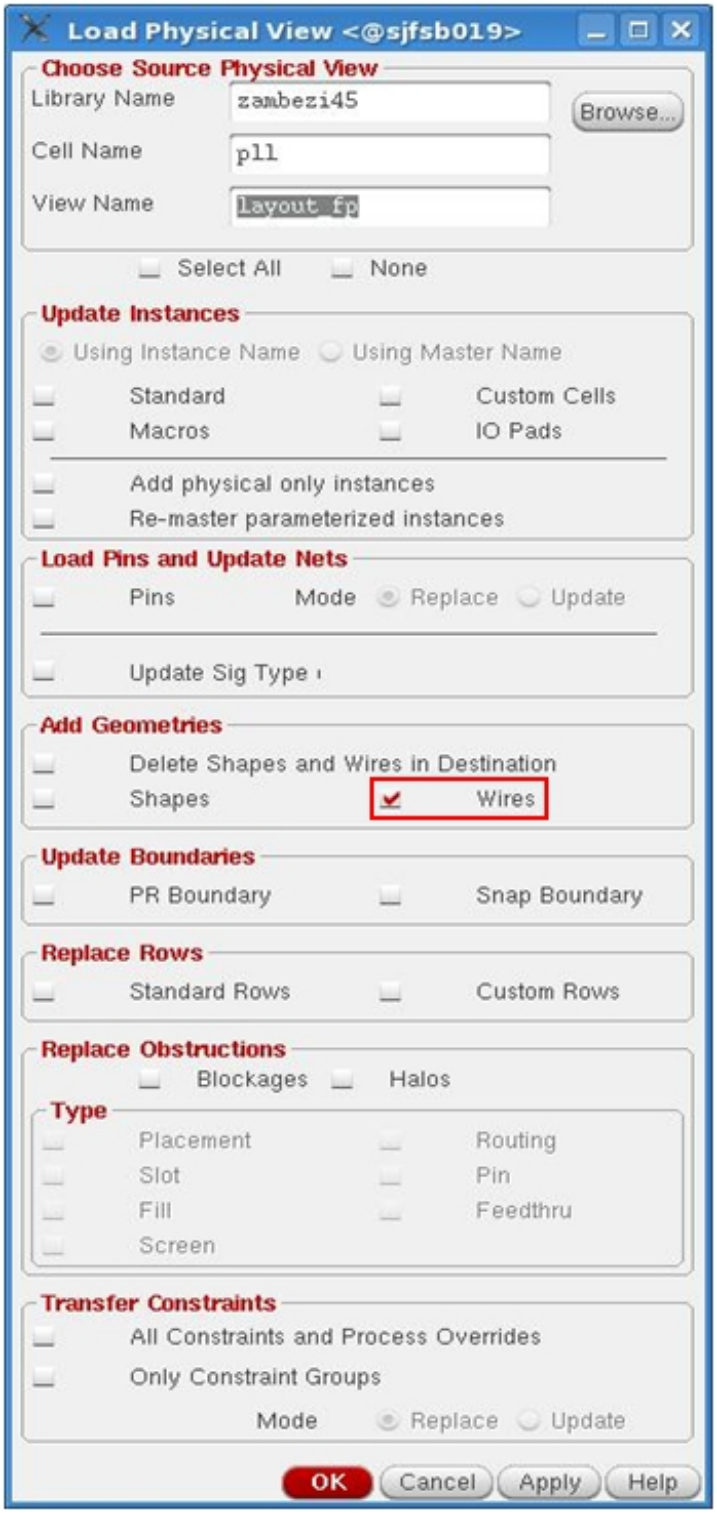


Next, you need to create the power plan at the top level and then push the power shapes into the digital block.

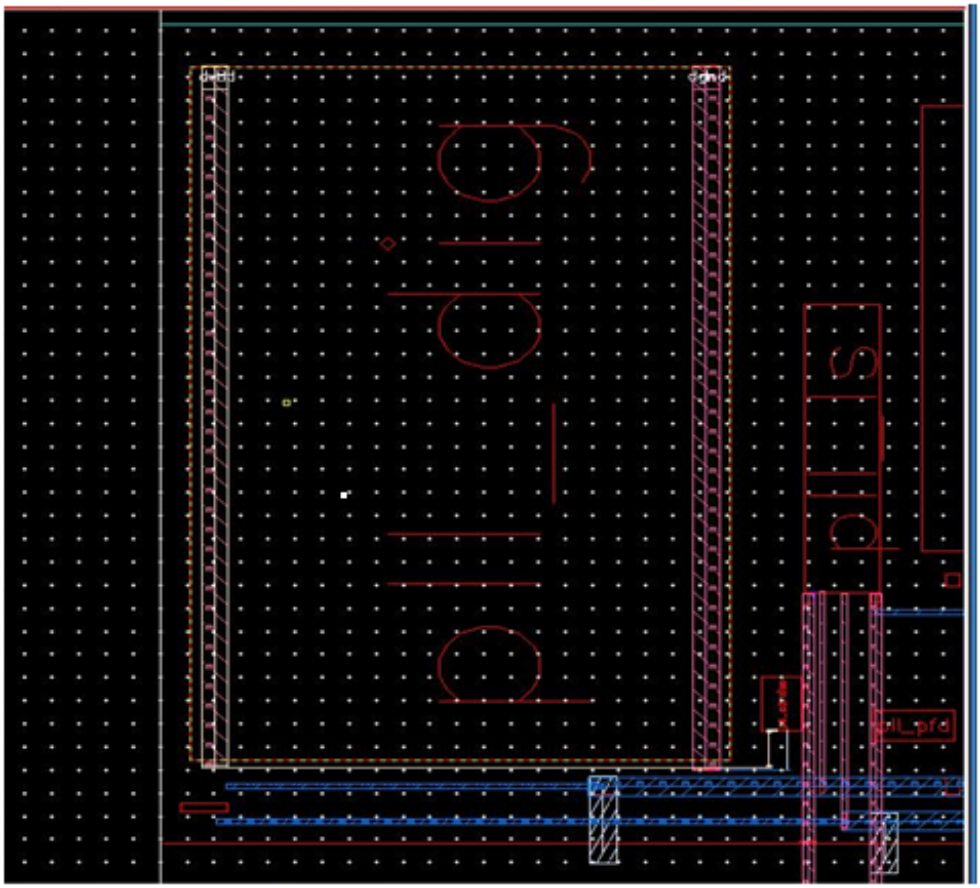
5.2.2 Pushing Power Shapes into the Digital Block

Before you push power shapes into the the digital block, you need to create the power plan at the top level and ensure that the topology is set correctly on the power shapes to be pushed down. To do so:

1. Create the power plan at the top level. To load the power stripes:
    a. Select *File - Load Physical View*.
    b. Specify the required library, cell, and view name.
    c. Select only the *Wires* option and click *OK.*



The canvas appears as follows after the power routes have been loaded:



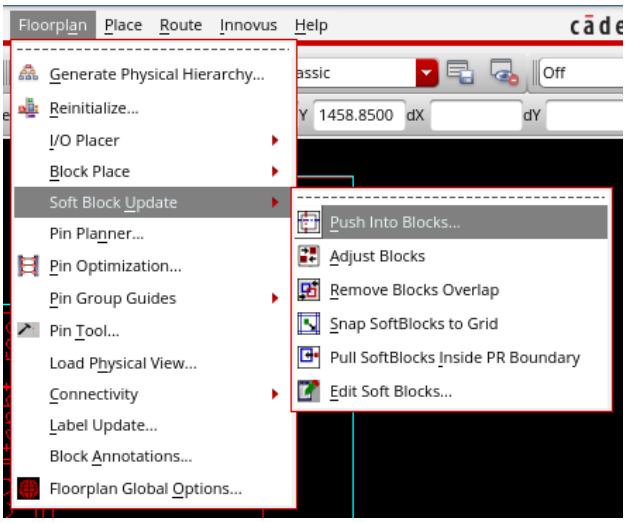
Note: Use Virtuoso Space-based Router (VSR) power planner for power planning so that the correct topology (for example, stripes in this case) is set on the power shapes to be pushed down. The *Power* menu is available in IC 6.1.6. For IC 6.1.5, you need to launch RIDE from the layout canvas for power planning.

You can select the power stripes on top of the digital block and set it in the CIW as follows:

```
geGetSelSet()~> topology = "stripe"
```

This needs to be done only if you are not using IC 6.1.6.

2. Push down power shapes:
    a. Select the digital block into which you want to push power shapes.
    b. From the layout window menu bar, select *Floorplan - Soft Block Update - Push Into Blocks*.



This opens the Push Into Blocks form.

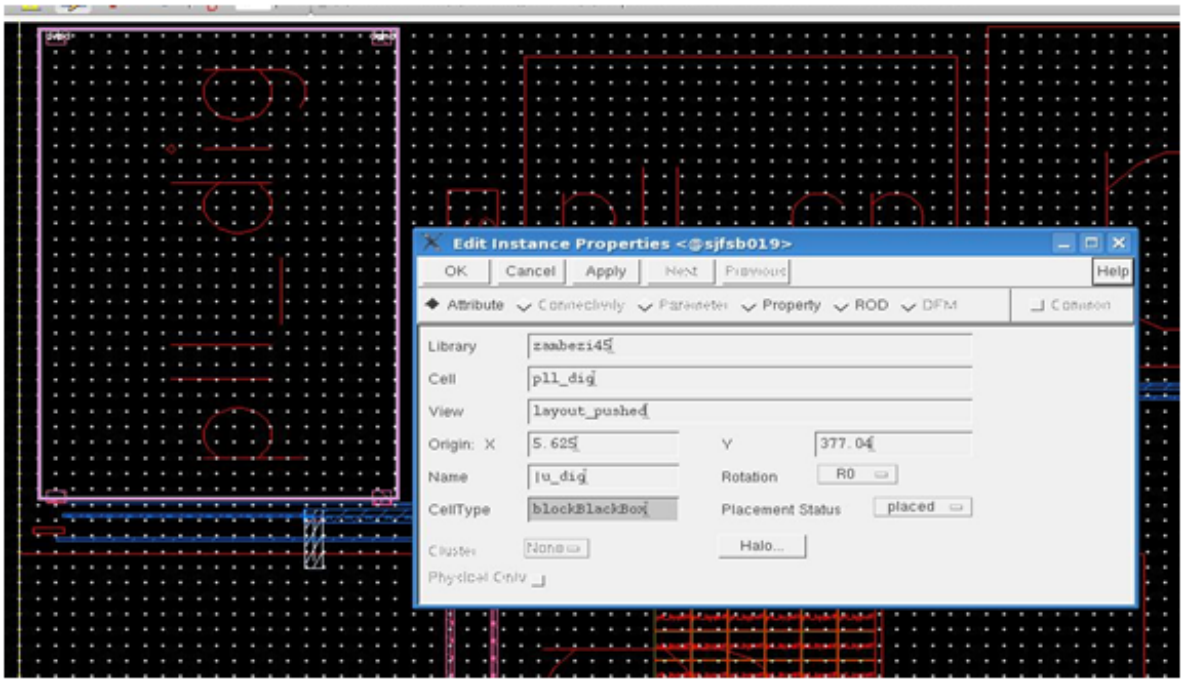
c. The library and cell view details are filled by default in the Push Into Blocks form based on the selected block. Click *OK* to push the power shapes into the new `layout_pushed` view that is created by default.



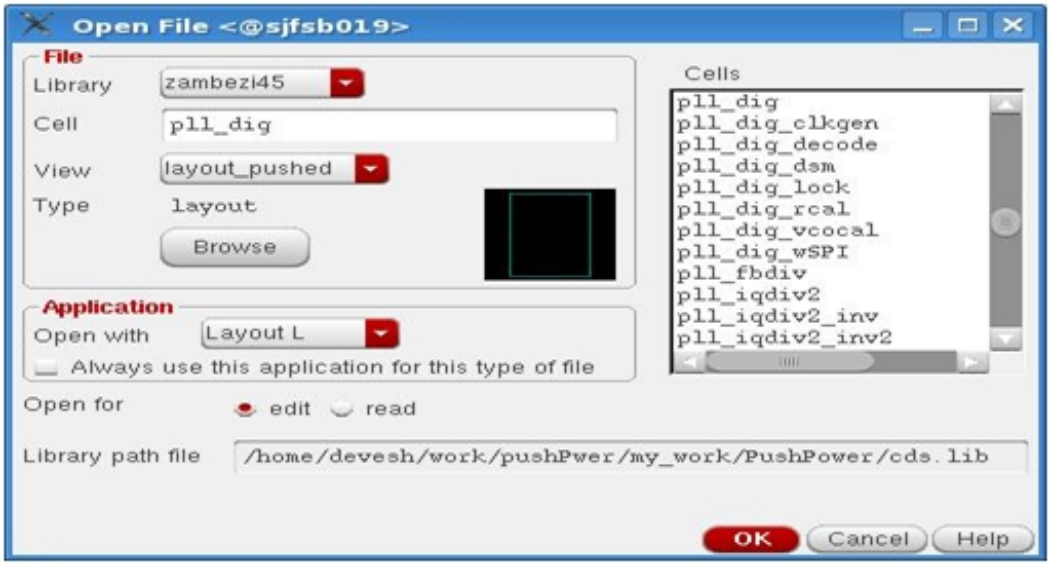
The top-level power net `(dvdd)` shapes are pushed as block power terminal `(VDD)` shapes and the top-level ground net `(dgnd)` shapes are pushed as block ground terminal `(VSS)` shapes into the digital block. If the top-level nets had not been marked as power and ground earlier, the top-level shapes would have been pushed as

blockages as only the *Power - Ground Net(s)* check box is selected in the Push Into Blocks form.

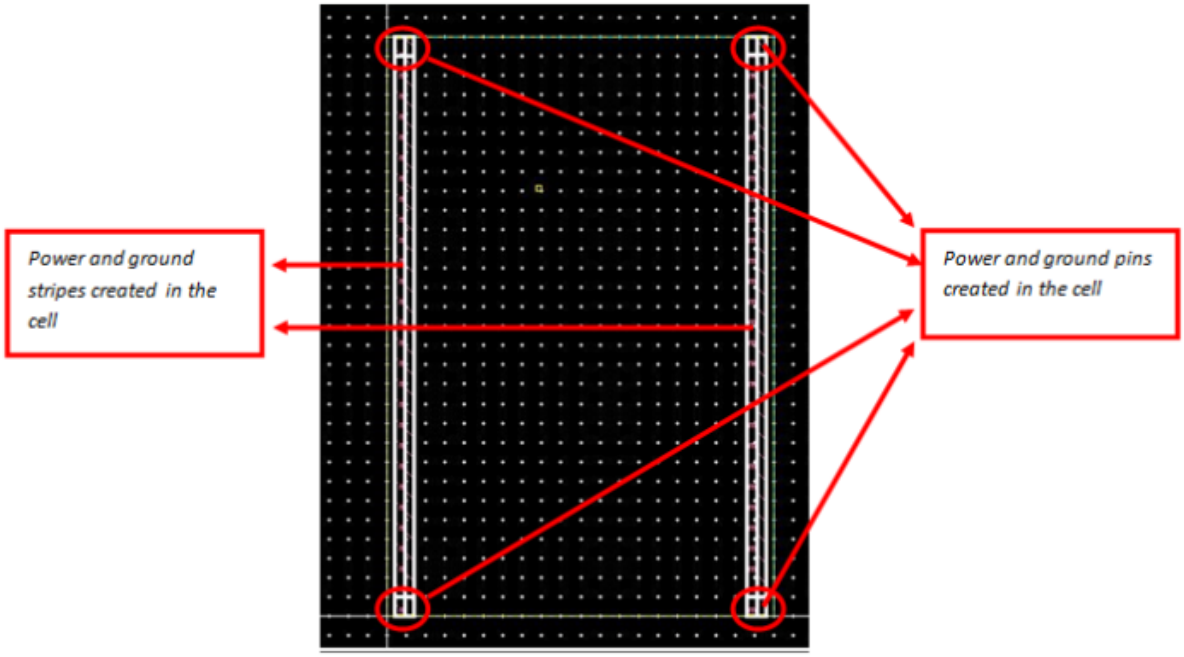
The power shapes are pushed down in the new view `layout_pushed` and removed from the top level. The new view is replaced at the top level and is used in Innovus to implement the block.



3. Review the block power structures:
    a. Open the newly created view by selecting *File - Open* and specifying the required cell view details.



b. Click *OK*.



Select the power and ground stripes and check that the net name is now VDD and VSS for the pushed shapes. You can also view the pins VDD and VSS created at both the ends for the stripes that have been pushed down.

Note: If you are not using IC 6.1.6, you need to re-establish the topology for the power stripes using the following steps. Otherwise, this is automatically handled in the 6.1.6 release:

a. Select both the power routes and check that the topology is set to none by using

the following skill command:

```
geGetSelSet() ~> topology
```

This will return "`none`".

b. Set the topology back to stripe by selecting the shapes and typing the following skill command:

```
geGetSelSet() ~> topology  = "stripe"
```

Setting the correct topology is important to fully utilize the capabilities of the Special Router in Innovus.

4. Save the pushed down view as an OpenAccess cell and exit Virtuoso:
   a. Save all the cell views.



   b. Exit Virtuoso.

5.2.3 Implementing the Digital Block in Innovus

To implement the block in Innovus and save the resulting view in OpenAccess, perform the following steps:

1. Launch Innovus.
2. Load and initialize the design by typing the following set of commands on the Innovus shell:
   a. Start the design implementation in Innovus with the Verilog file for the digital block while using the OpenAccess reference libraries for the blocks.

```
set init_verilog pll_dig.vg
```

```
set init_top_cell pll_dig
set init_oa_ref_lib {gsclib045 gpdk045 giolib045}
```

From where is the tech information read?

If an OpenAccess design is read, the tech graph is analyzed from the design_lib that contains the cellview.

If a Verilog netlist is read with OpenAccess reference libraries, then the first library in the OpenAccess ref lib list is used for the tech. This means that the order of the ref libs is important when a Verilog netlist is used.

Note: Additionally, sites and vias can be read from other libraries even though they are mostly considered to be tech information.

b. Specify the power net and the ground net.

```
set init_pwr_net {VDD}
set init_gnd_net {VSS}
```

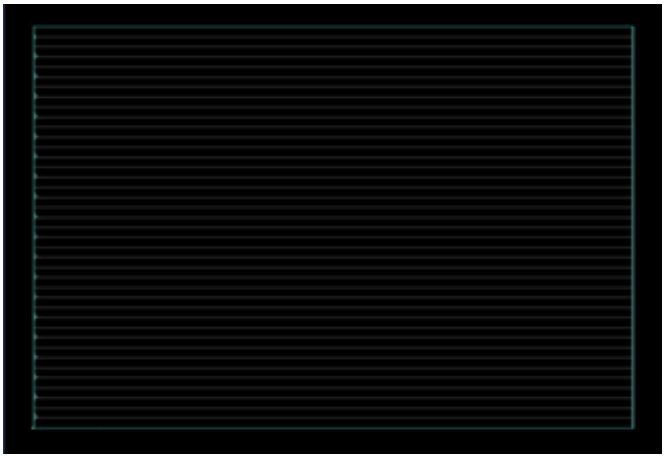
c. Initialize the design.

```
init_design
```

d. Launch the GUI.

```
win
```

The design appears as follows after initialization.



3. Load OpenAcess data, modify floorplan, and load the block CPF file:

a. Load the OpenAccess data for the block to get the boundary, pin locations and the power shapes from the OpenAccess cellview.

```
oaIn zambezi45 pll_dig layout_pushed
```

b. As the prBoundary height and width has the space for both the core and IO area, you

need to modify the floorplan so that the core area rows are not extended to the boundary.

```
changeFloorplan -coreToEdge {8 2 8 2}
```

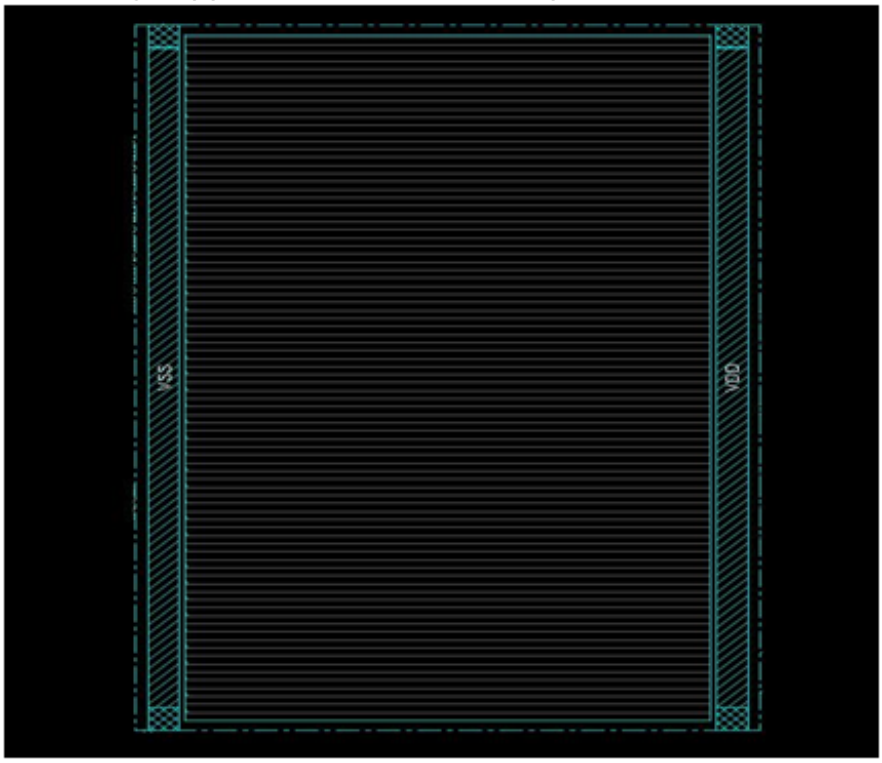
c. Load the block CPF file to create the global net connections for the block in Innovus.

```
read_power_intent -cpf pll_dig.cpf
commit_power_intent
```

d. Select *Power - Connect Global Nets* from the Innovus menu bar to open the Global Net Connections form where you can check the power and ground connections for the block.



The design appears as follows after OpenAccess load.



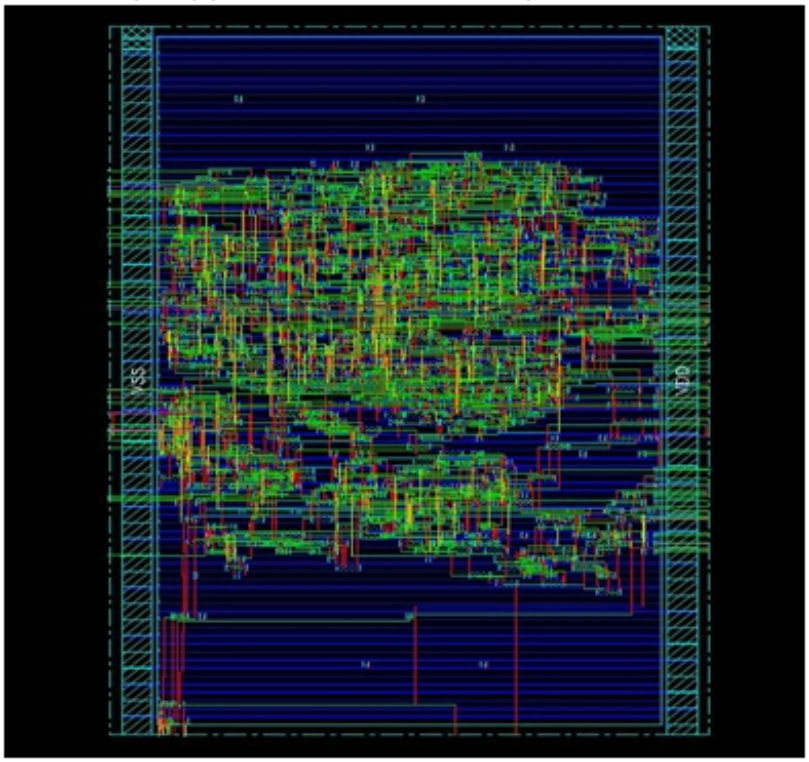
4. Place and route the design to complete the block implementation:

a. Type the following command at the Innovus shell to start placement:
```
place_design
```

b. Type the following set of commands to start routing:
```
dbDeleteTrialRoute
sroute -connect { corePin } -nets { VDD VSS }
generateVias
routeDesign
```

c. Verify connections after routing is complete by typing the following command:
```
verifyConnectivity -noAntenna
```
There should be no errors after `verifyConnectivity`. The `-noAntenna` option allows for the dangling wires at the end of the rows.

The design appears as follows after placement and routing.



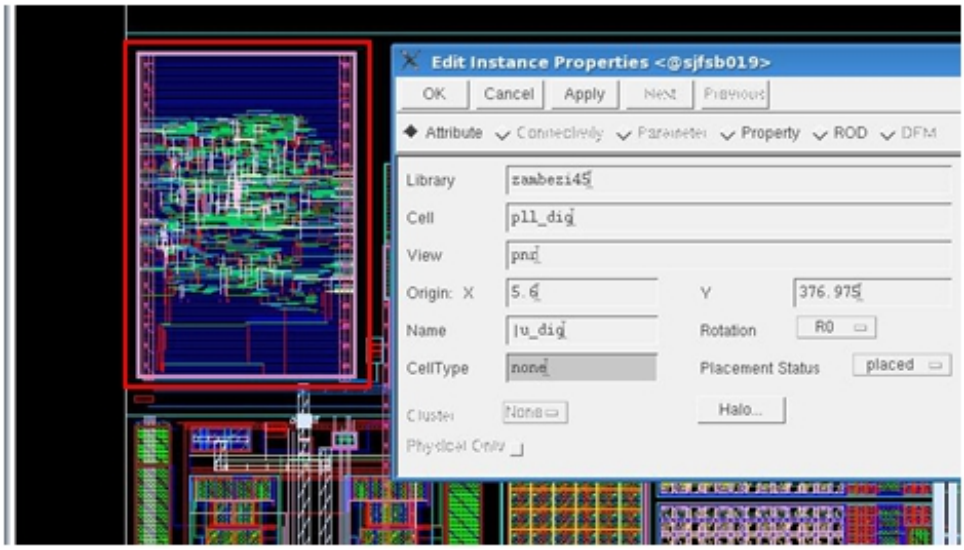
5. Save the design using the `saveDesign` command. Specify a new view name, such as `pnr`, while saving the design
```
saveDesign -cellview {zambezi45 pll_dig pnr}
```

6. Exit Innovus.

5.2.4 Bringing Back the Digital Block into Virtuoso for Design Assembly

Remaster the digital block at the top level in Virtuoso to use the implemented view. To do so:

1. Invoke Virtuoso and open the required cell.
2. Select the digital block and invoke the property editor.
3. Change the view name for the digital block from `layout_pushed` to the new view name you specified while saving the design in Innovus. In the example below, the new view name is `pnr`.



6

Virtuoso Digital Implementation

- Overview
- Invoking the VDI Environment
  - Starting Innovus Using a Verilog Netlist
  - Starting Innovus Using an OpenAccess Database from Virtuoso
  - Starting Innovus Using an Existing Script
  - Loading Existing Form Configuration
- Configuring Power Planning and Power Routing
- Creating the View Definition File
  - Implementing a Digital Block in the Single Timing Analysis Mode
  - Implementing a Digital Block in the BcWc or Min/Max Analysis Mode
  - Implementing a Digital Block in the MMMC Analysis Mode
- Specifying Physical Cells
- Specifying Optional Plugin Scripts
- Generating the Innovus Script
  - With Customization
  - Without Customization
- Completing the Implementation
- Sample Files
  - Sample View Definition File
  - Sample SDC File
  - Generated Innovus Script Sample
- Helpful Hints

6.1 Overview

In the Cadence mixed signal implementation flow, Innovus Implementation System is used for complete implementation of digital content of the design. Very often the digital content is captured in one or more digital blocks contained in an Analog-on-Top (AoT) design. As the AoT design style is most often used by Virtuoso-based mixed signal design teams, the team may be unfamiliar with Innovus and the flow used for implementing the digital portion of mixed signal designs. As a result, many mixed signal design teams rely on digital designers to help implement the required digital functionality in a mixed signal design.

The Virtuoso Digital Implementation (VDI) environment in Virtuoso enables users who do not have familiarity with digital design methodology to implement the required digital functionality easily. The VDI environment invokes Innovus from within Virtuoso and

creates the necessary script to implement the needed digital functionality completely.

To use the VDI environment, your unix path needs to have both an Innovus and a Virtuoso executable specified. If you cannot see the pull down menu for Innovus from within Virtuoso, chances are that you either do not have the Innovus installation specified in your UNIX path settings or there is a problem with that installation.

Please note that this interface only works with Virtuoso 6.1.8 or later versions. Customers using Virtuoso 6.1.7 will not be able to take advantage of this capability.

Note: The VDI interface requires the following product option license in addition to the Innovus startup license specified by the user:

- Mixed Signal Option (INVS30).

This license is checked-out automatically when the VDI interface is launched.

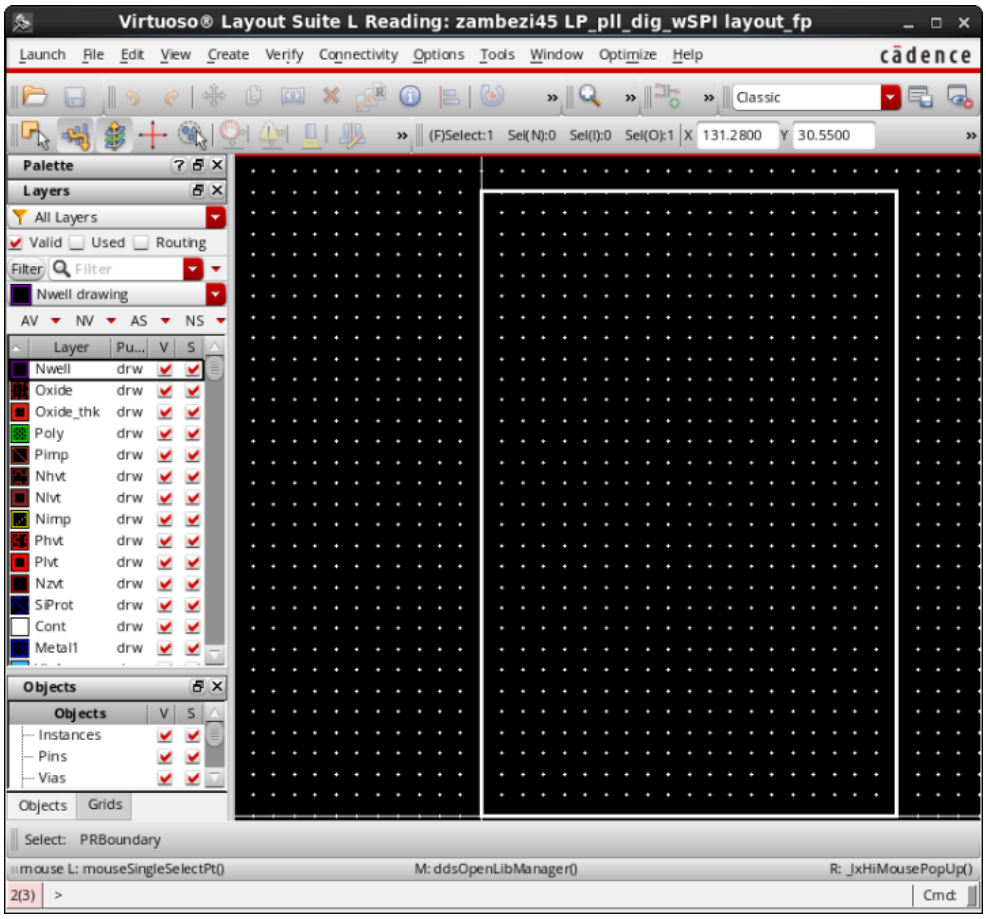
6.2 Invoking the VDI Environment

You can use the VDI environment to create the necessary scripts required to fully implement a digital block. In the AoT flow, the top-level floorplanning is typically done in Virtuoso and the boundary and the pins for the digital block are fixed during this step.

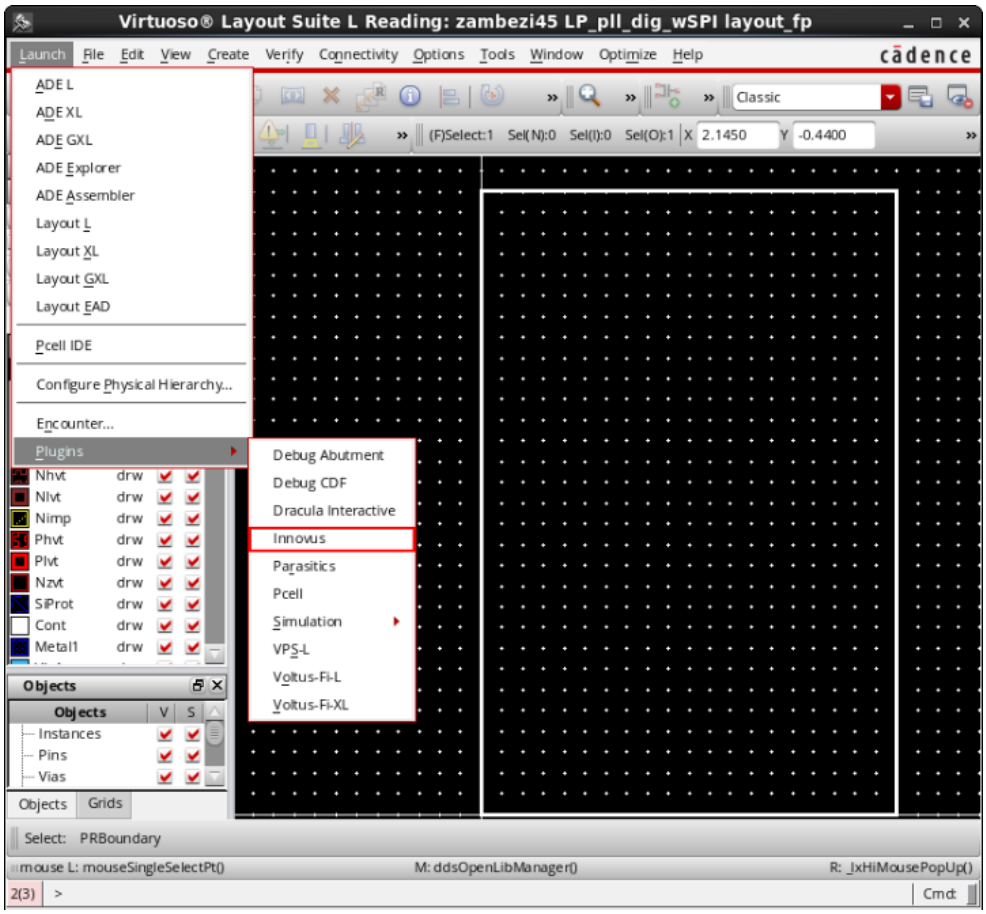
> Set the `blockType` property as `digital` for the digital block so that the pins are snapped during the Move command or the Virtuoso Pin Optimizer on the routing grid. This will help during the routing stage in Innovus.

Use the following steps to launch the VDI interface:

1. Open the soft block representing the digital block as created during top-level floorplanning. At this point, the layout view of the digital soft block is just the boundary with the pins placed.

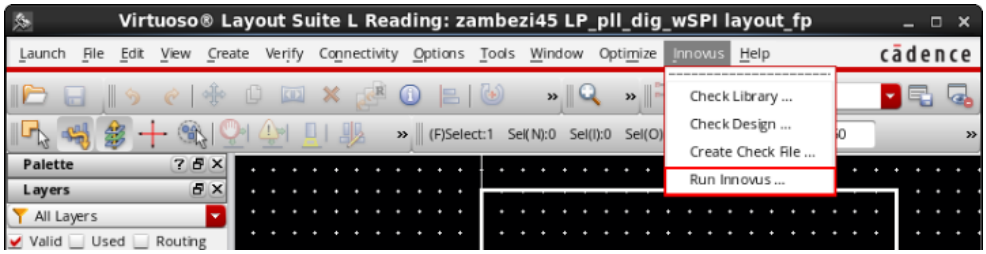


2. Next, launch the Innovus plugin by selecting *Plugins ->Innovus* from the *Launch* menu of Virtuoso Layout Suite.



This adds the *Innovus* pull-down menu to the menu bar in the Virtuoso window.

3. The *Innovus* pull-down menu includes controls for the OpenAccess Database Interoperability Checker (oaDBChecker) as well as the VDI environment. To invoke the VDI environment, select *Run Innovus* from the pull-down menu.



This opens the Innovus launch GUI form, which provides various options for launching Innovus in the VDI environment.



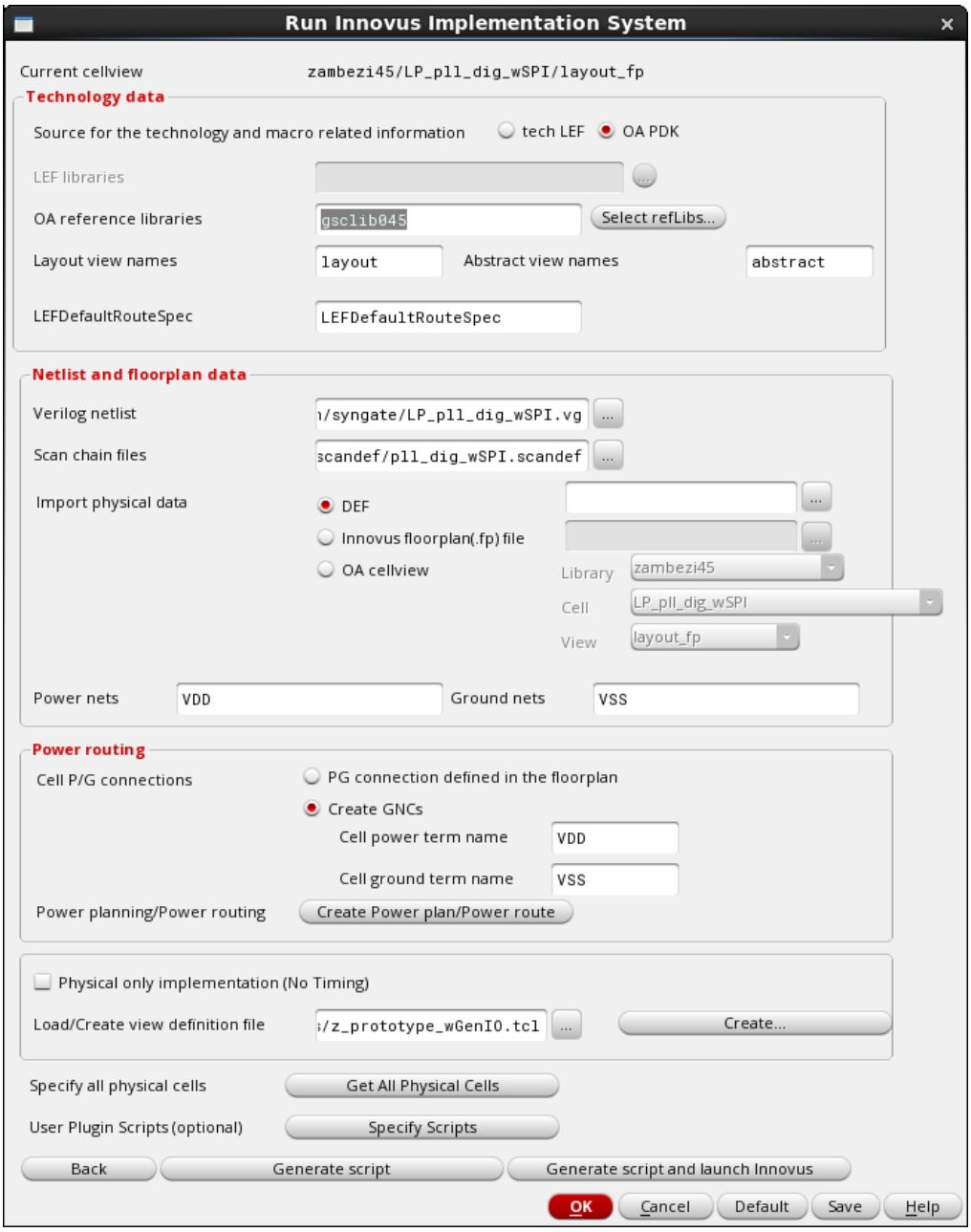
The table below describes these fields and options:

Innovus mode	Select the mode for creating the script and launching Innovus, *Default* or *Stylus CUI*.	
Process setup	Select the process node, foundry, and the corresponding standard cell library from the drop-down list. The VDI environment will then populate the run scripts with the settings that are needed for that particular process setup. If the process setup used for your design is not listed, select *Default* from the list.	
Start Innovus using	Select the method to be used for starting Innovus through the VDI interface. The possible options are by using:	
	A Verilog netlist	Select this option if you have been given a Verilog netlist that contains the connectivity for the digital block. See the **Starting Innovus Using a Verilog Netlist** section for subsequent steps.
	An OA database containing connectivity for the design	Select this option if the connectivity for the digital block has been captured in a schematic, and a layout view containing the schematic connectivity has been created using Generate from Source (GFS) in Virtuoso. The VDI interface enables you to use the layout view containing the schematic connectivity to invoke Innovus. See the **Starting Innovus Using an OpenAccess Database from Virtuoso** section for subsequent steps. Note that Innovus is unable to read Virtuoso schematics directly. However, a layout view created in Virtuoso using GFS would contain the necessary connectivity for Innovus.
	An existing script	Select this option if you have previously run the interface and saved the script. Specify the script name in the associated field to pass the script to the interface so that Innovus uses that script for the entire run. This option is useful in cases where you have made a manual modification to the script to either correct or improve the Innovus run, and do not need to go through the entire GUI to reconstruct the script. See the **Starting Innovus Using an Existing Script** section to view an example.

Innovus run directory	Specify the directory in which the Innovus run files are to be created. If you leave this field blank (default), the run files are created in the current directory.
License to use	Specify the license to be used. The *VDI* license is selected by default.
Launch method	Specify the xterm launch method. The available options are: • *Local-Host*: This is the default method in which the xterm for invoking Innovus is launched on the machine from which Virtuoso is launched. • *Launch Command*: Select this option if you want to launch the xterm on a separate machine. On selecting this option, the *Command options* text box is enabled, where you can specify the command for launching the xterm. For example, if you want to launch an xterm by using the command `bsub -q interactive -W 500:00 -R "select[(OSREL==EE60) && (CPUS>=4)] rusage[mem=8000] " xterm`, you should specify the following in the *Command options* text box: `bsub -q interactive -W 500:00 -R "select[(OSREL==EE60) && (CPUS>=4)] rusage[mem=8000] "` **Note**: You do not need to specify `xterm` at the end of the command.
Load existing form configuration	Select this option to load the form from a previously saved configuration file. The VDI flow includes a series of forms that help you capture the necessary information for the construction of the script. If you have previously run the interface, you can simply save the contents of the forms to a file and then load this file to fill in the forms with the data in the next run. See the **Loading Existing Form Configuration** section to view an example.

6.2.1 Starting Innovus Using a Verilog Netlist

If you select the *A Verilog netlist* option for starting Innovus from the Innovus Launch GUI form, the following form is displayed. You can specify the inputs required to run Innovus and implement the digital block for your Verilog netlist.

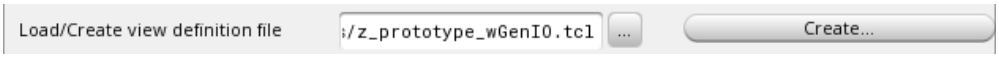


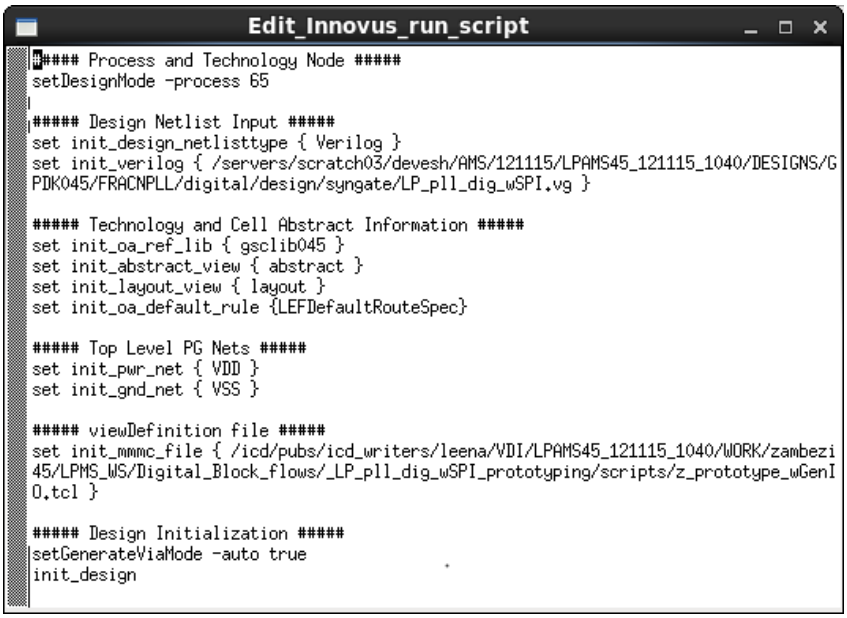
The table below describes the options in this form and how to complete them:

Current cellview	Shows the cell view in Virtuoso from which the VDI interface was invoked.
Technology data section	

	Source for the technology and macro related information	Specify the source for the technology information. Choose one of the following options: • *tech LEF* - Runs the VDI interface by reading the technology and macro information from the LEF files provided. • *OA PDK* - Runs the VDI interface by reading the technology and macro information from the OpenAccess libraries. The OpenAccess PDK must be an interoperable mixed signal OpenAccess PDK (MSOA PDK).
	LEF libraries	Specify the LEF files to be used in this field if you are running the VDI interface in the LEF mode. Note that the first LEF file name should be of the technology LEF file.
	OA reference libraries	Specify the mixed signal OpenAccess reference libraries to be used for implementation in this field if you are running the VDI interface in the OpenAccess mode. Note that the first library in the list is the technology library for the particular process being used. This field is grayed out if you have selected the *tech LEF* mode for running the VDI interface.
	Layout view names	Specify the layout view names in this field if you are running the VDI interface in the OpenAccess mode. This field is grayed out if you have selected the *tech LEF* mode for running the VDI interface.
	Abstract view names	Specify the abstract view names in this field if you are running the VDI interface in the OpenAccess mode. This field is grayed out if you have selected the *tech LEF* mode for running the VDI interface.
	LEFDefaultRouteSpec	Specify the name of the Constraint Group in the MSOA PDK containing the technology information for Innovus. Typically, this specificaion is called *LEFDefaultRouteSpec*. However, if your MSOA PDK has a different name, specify that here. This field is grayed out if you have selected the *tech LEF* mode for running the VDI interface.
Netlist and floorplan data section		
	Verilog netlist	Specify the path to the Verilog netlist containing the connectivity information for the digital block being implemented.
	Scan chain files	Specify the path to the scan file, if available, for scan chain insertion.
	Import physical data	Specify the file containing the floorplan for the digital block. If you invoked the VDI interface from a layout view in Virtuoso, the *OA cellview* field is automatically set to that lib/cell/view. Alternatively, you can use a DEF file or a Innovus floorplan file as the block floorplan. The *DEF* and *Innovus floorplan (.fp) file* options are intended for use by LEF/DEF users of the VDI interface, but they can also be used by OpenAccess users, if needed.
	Power nets	Specify all the power nets in the design here.
	Ground nets	Specify all the ground nets in the design here.
Power routing section		

	Cell P/G connections	Specify the power/ground (P/G) connections here. Choose one of the following options:
		• *PG connection defined in the floorplan* - Select this option if you already have P/G connections defined in the layout view of the digital block.
		• *Create GNCs* - Select this option if you do not have predefined P/G connections. Specify the cell P/G terms in the *Cell power term name* and *Cell ground term name* fields. The VDI interface creates the proper Global Net Connect commands (GNCs) to connect the cell P/G terms to the global power and ground. The commands that would be generated in the Innovus script for the digital implementation of the block would look similar to the following:
		`globalNetConnect VDD -pin VDD`
		`globalNetConnect VSS -pin VSS`

	Power planning/Power routing → Create Power plan/Power route	Click this button to create power stripes and routes through the GUI for the digital block being implemented. This is required if the floorplan you loaded for the digital block does not already have power stripes and power routing configured. See the **Power Planning/Power Routing** section for more details on creating power stripes and power routing through the GUI.
Physical only implementation (No Timing)		Select this check box if you want a physical-only implementation. In the physical-only implementation mode, timing analysis or timing optimization is not done. This mode is ideal for small digital blocks that do not have critical timing paths. This mode is default and is also invoked if you do not specify a view definition file in the *Load/Create view definition file* field.
Load/Create view definition file		Specifies the view definition file required for running Innovus. The view definition file describes the modes and corner the analysis engine in Innovus should use when evaluating timing. The VDI interface supports the following implementation modes: • A single timing analysis mode and implementation • Best case/worst case (BcWc or Min/Max) analysis mode and implementation • Full multi-mode multi-corner (MMMC) analysis mode and implementation For all three implementation modes, you must specify a view definition file.  If you already have the view definition file, you can specify its path in the *Load/Create view definition file* field to load it. Alternatively, click the *Create* button to create a new view definition file. See the **Creating the View Definition File** section for more details. **Note** - If you want a physical-only implementation for the block, select the *Physical only implementation (No Timing)* check box and leave the *Load/Create view definition file* field blank.
Specify all physical cells → Get All Physical Cells		Click the *Get All Physical Cells* button to extract the list of all physical cells (TIE, PAD FILLER, WELLTAP, ENDCAP, and CORE FILLER cells) from the specified OpenAccess reference libraries or LEF files. See the **Specifying Physical Cells** section for more details.
User Plugin Scripts (optional) → Specify Scripts		Click the *Specify Scripts* button if you want to plug in your own code in the Innovus script. This opens the Plugin form, which provides options for adding plugin files at various stages in the Innovus script. See the **Specifying Optional Plugin Scripts** section for more details.

Generate script	Click this button after completing all settings in the form to invoke the script generation part of the interface. The generated script is displayed in a vi text editor window, allowing you to customize it, if required. The script contains all the necessary steps required to implement a design using Innovus, based on the inputs you have specified.  See the **Generating the Innovus Script** section for more details.
Generate script and launch Innovus	Click this button to generate the script and launch Innovus in a single step. Use this button if you do not need to customize the script generated by the VDI interface in any way. See the **Generating the Innovus Script** section for more details.

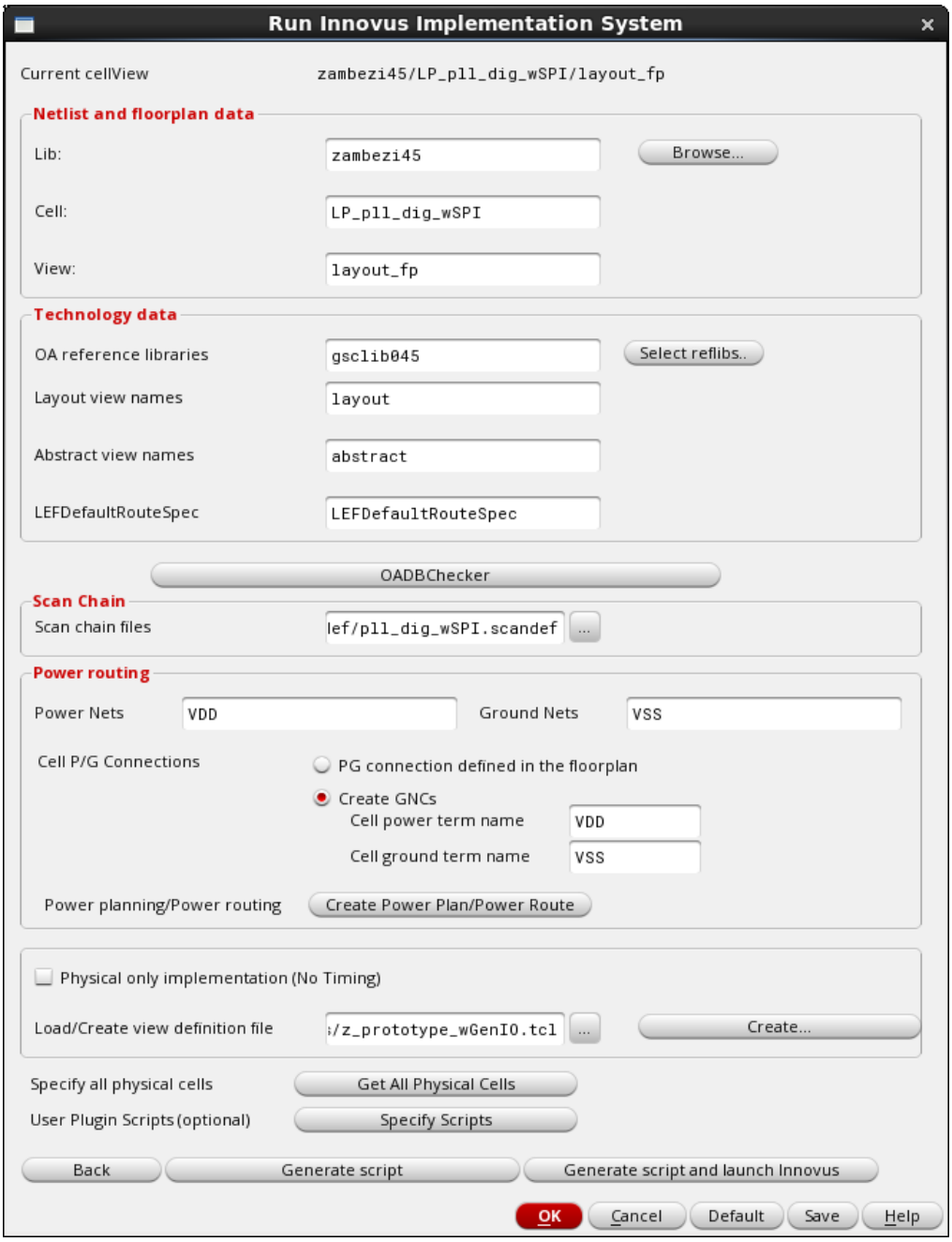
6.2.2 Starting Innovus Using an OpenAccess Database from Virtuoso

Instead of using a Verilog netlist, the VDI interface could also get the connectivity information for the digital block from an OpenAccess database that has been properly created, using Virtuoso XL connectivity driven flow. The logical connectivity of the design should have been captured using a schematic in Virtuoso-XL, and a layout database should have been created using Generate From Source (GFS). In such a case, you should select the *An OA database containing connectivity for the design* option from the Innovus Launch GUI form:



Note - This is the only way Innovus can implement a digital block that does not have connectivity captured in a Verilog netlist. A workaround is to generate a Verilog netlist from a schematic in Virtuoso, and then use the VDI interface with the generated Verilog netlist. However, the generation of the Verilog netlist in Virtuoso has to be controlled using special settings to make the generated Verilog netlist consumable by Innovus.

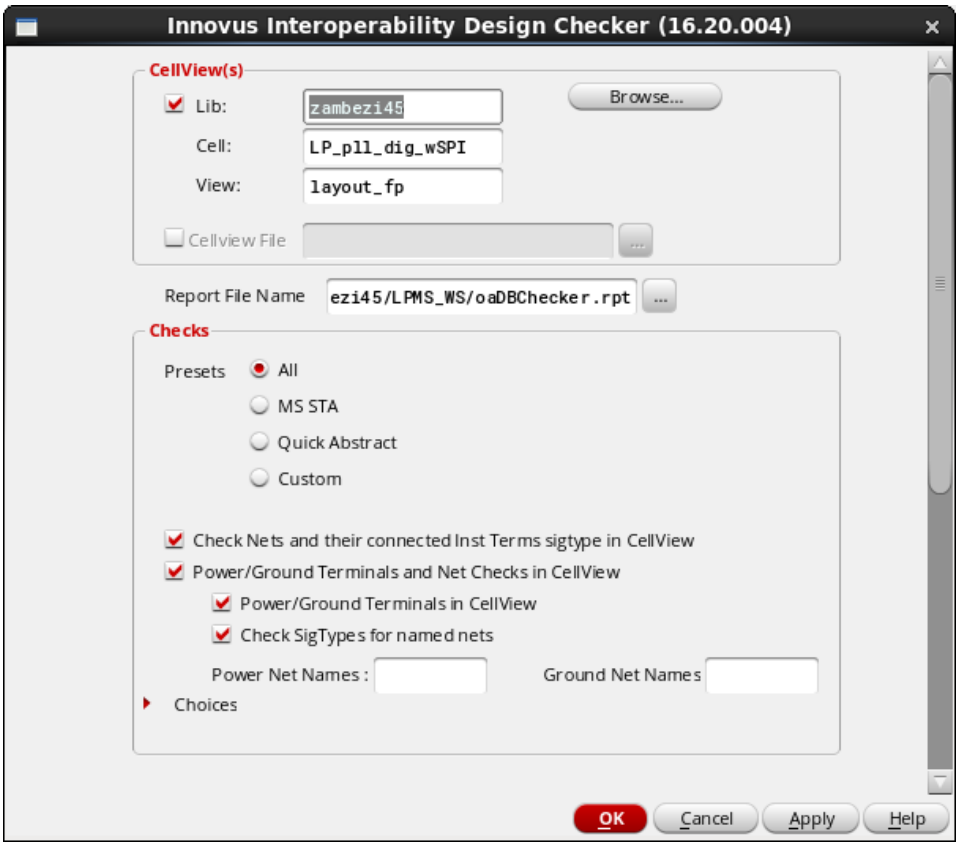
When you click *OK* after selecting *An OA database containing connectivity for the design* option, the following form is displayed:



The Run Innovus Implementation form guides you through the process of starting Innovus using an OpenAccess database for connectivity. Notice that the the netlist is populated with the cell view in Virtuoso from which the VDI interface was invoked.

Most options in this form are the same as described in the **Starting Innovus Using a Verilog Netlist** section. Refer to this section for details of setting these options.

An addition to the Run Innovus Implementation form when you start Innovus using an OpenAccess database is the *OADBChecker* button. As you are using an OpenAccess database of the cell view to drive Innovus, you should run the interoperability checker to determine the readiness of the design and the technology library for use by Innovus. Clicking the *OADBChecker* button launches the Innovus Interoperability Design Checker form. For details on how to use this form, see the **OpenAccess Database Interoperability Checker** chapter.



6.2.3 Starting Innovus Using an Existing Script

If you have previously run the VDI interface and saved the script, you can start Innovus by choosing the *An existing script* option from the Innovus Launch GUI form:

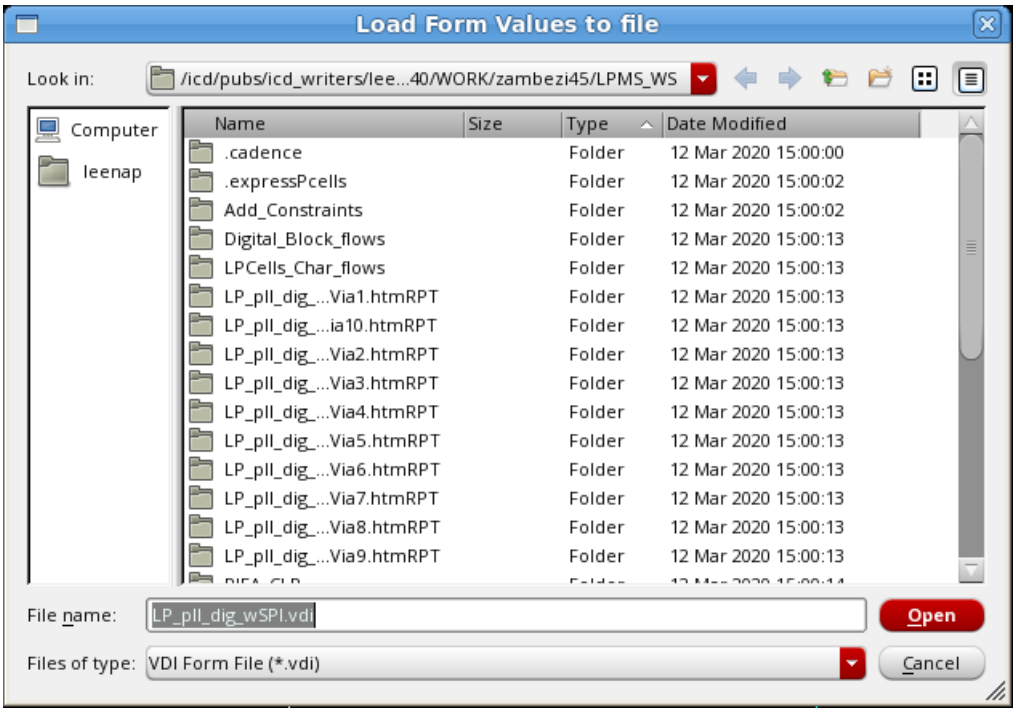


Specify the script name in the associated field to pass the script to the interface so that Innovus uses that script for the entire run. The VDI interface will skip the entire GUI sequence for capturing your inputs and will instead launch Innovus and implement the digital block based on the settings in the specified script. This option is useful in cases where you have made a manual modification to the script to either correct or improve the Innovus run.

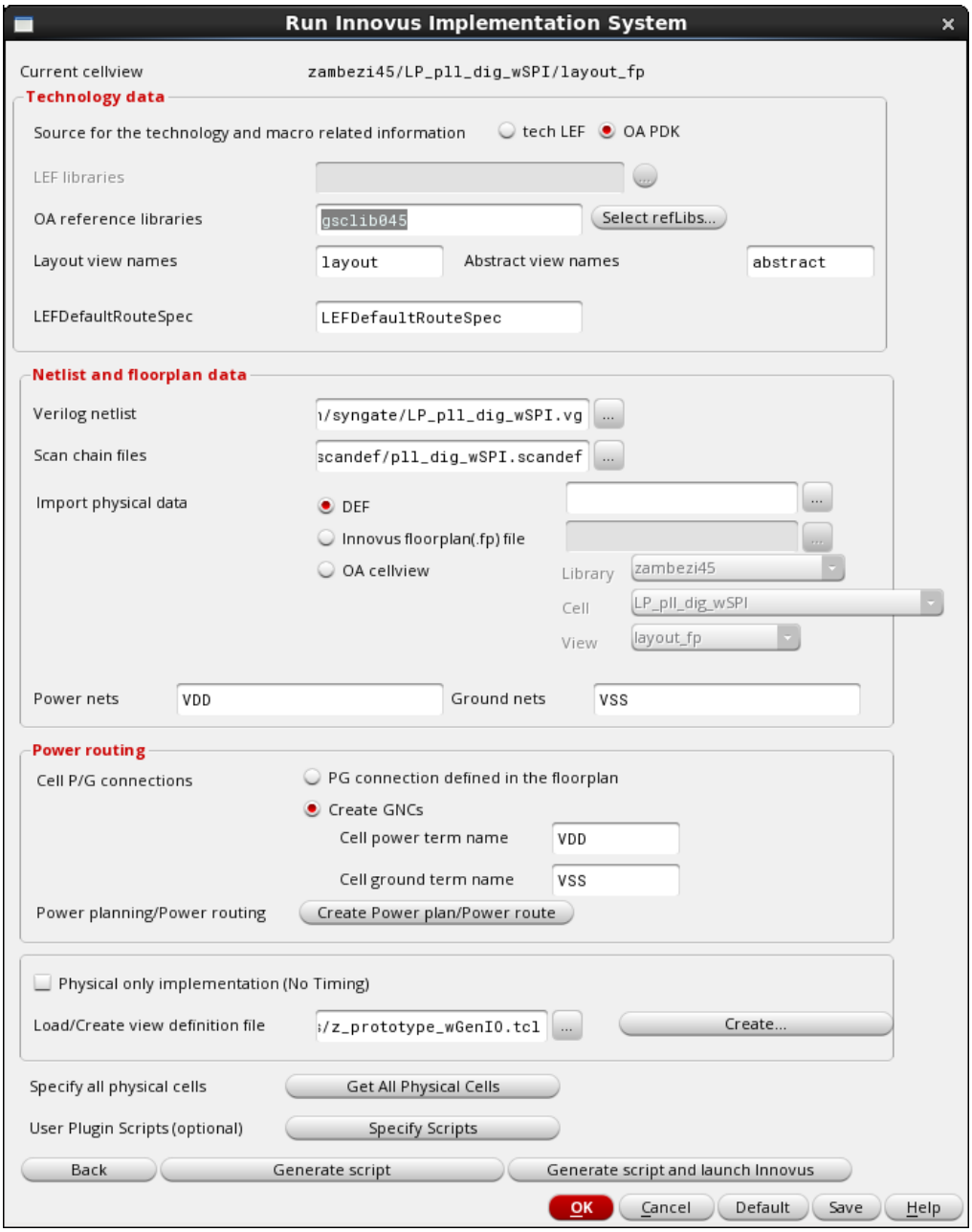


6.2.4 Loading Existing Form Configuration

The VDI flow includes a series of forms that help you save the necessary information for the construction of the script. If you have previously run the interface, you could simply recall the data entered in the forms when you next run the VDI flow. Click the *Load existing form configuration* button in the Innovus Launch GUI form, and select the file in which you have saved the form configuration in the Load Form Values to file form.

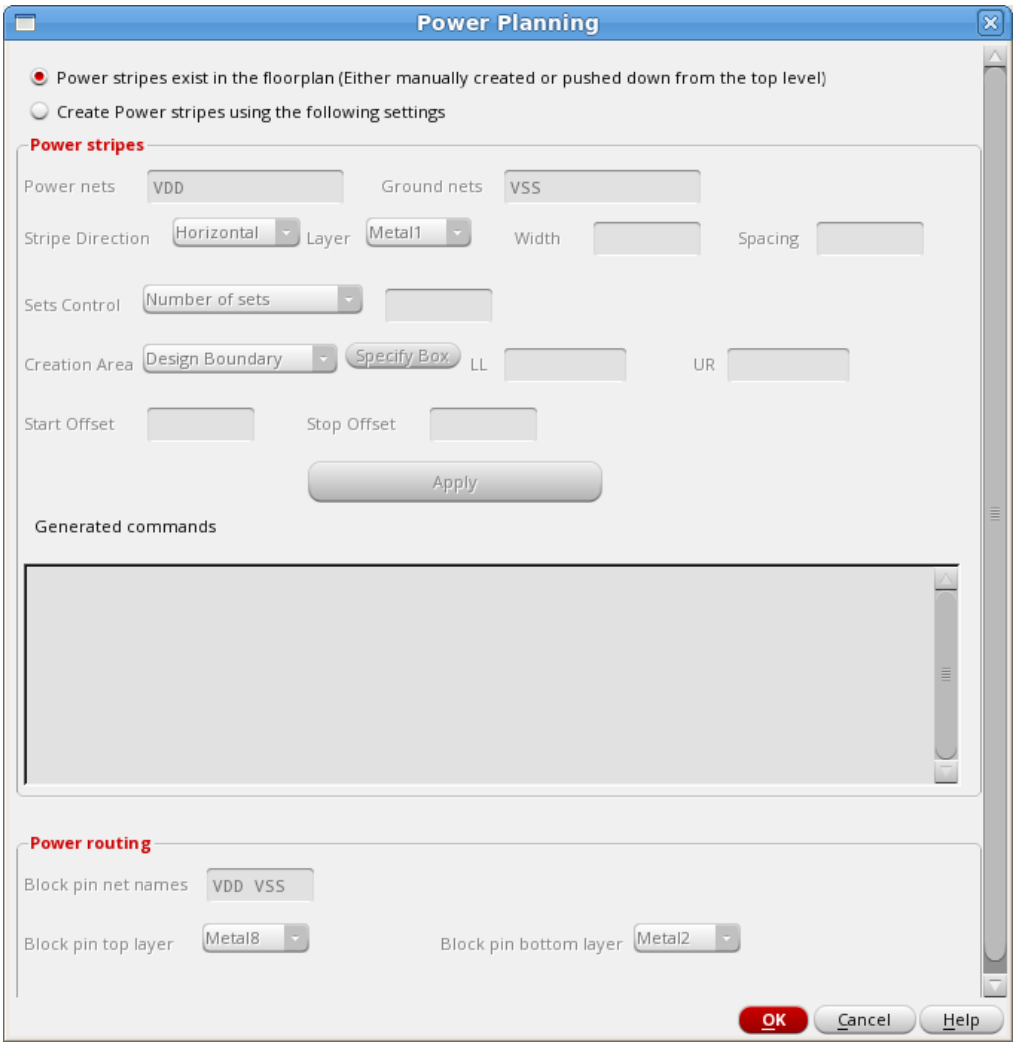


The VDI interface prefills all the forms and subforms in the sequence with the settings captured in the form value file that you loaded. For example, the Run Innovus Implementation System form below has been prefilled with the values captured from a previous run.



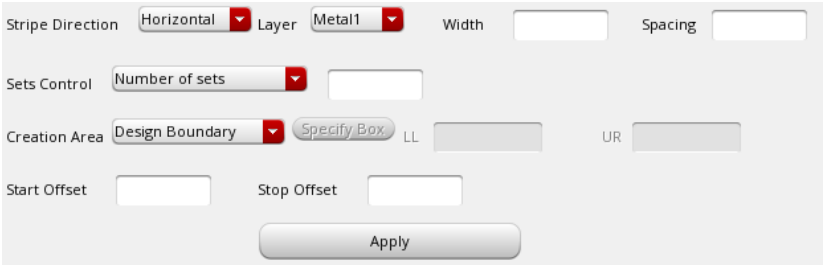
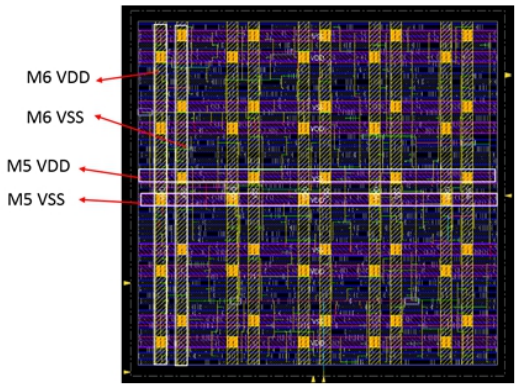
6.3 Configuring Power Planning and Power Routing

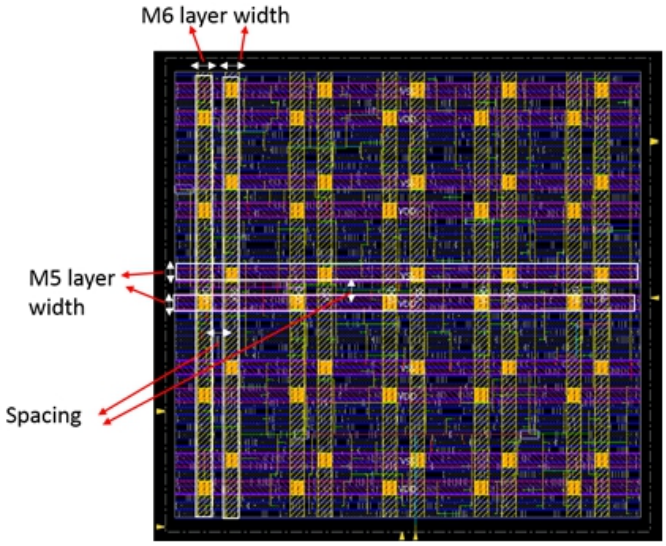
You can use the Power Planning form to configure how power routing is done for the digital block being implemented. To access the Power Planning form, click the *Create Power plan/Power route* button in the *Power routing* section of the Run Innovus Implementation System form.

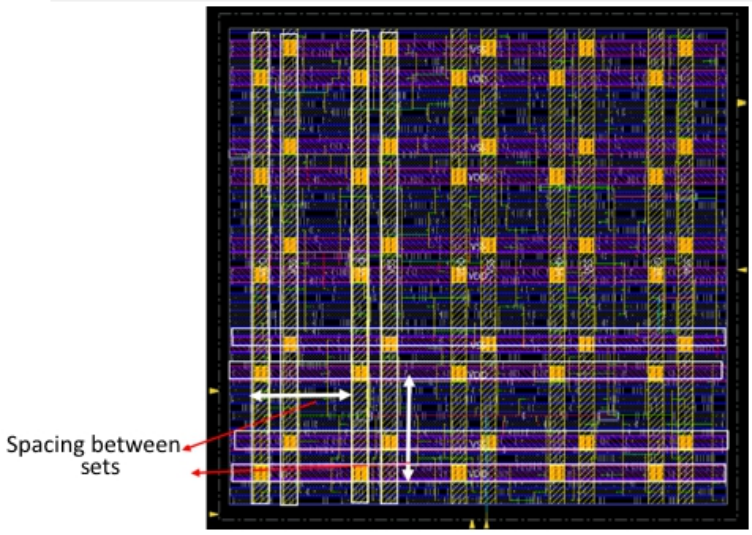


The table below describes the options in this form and how to configure them:

Power stripes exist in the floorplan	Select this option if power stripes and power routing already exist in the floorplan for the digital block. The power stripes could have been either manually created or pushed down from the top-level floorplan. If the *Power stripes exist in the floorplan* option is selected, the rest of the form is grayed out. However, if the floorplan has only the power stripes but the connections to corePins and blockPins need to be created, uncomment the `sroute` command dumped in the script before launching Innovus.
Create Power stripes using the following settings	Create power stripes and power routing based on the settings you specify in the *Power stripes* and *Power routing* sections of the form. The VDI interface generates the necessary entries in the Innovus script based on the values you specify.
Power stripes	Specify the settings for the power stripes in this section as described below.

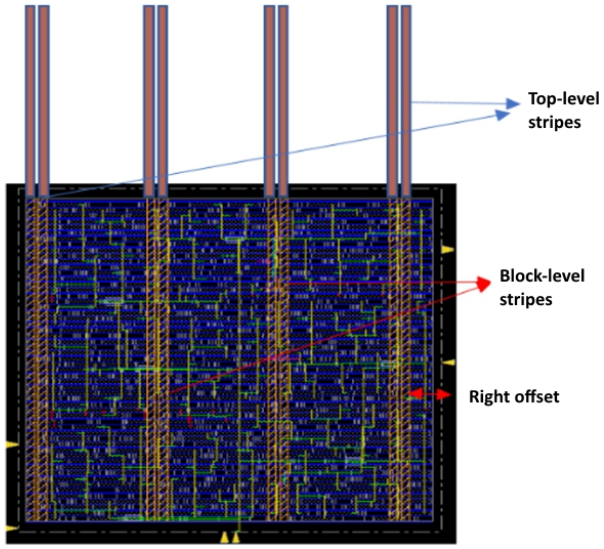
	Power nets *Ground nets*	Specify the power and ground nets in these fields.  The interface auto-fills the power and ground net names that have been specified in the main GUI.
	Stripe Direction	Select the stripe direction here and then specify the layer, width, spacing, and other settings for the stripe set. After clicking *Apply*, you can select another stripe direction and specify settings for that direction. 
	Layer	Select the layer for the power and ground nets from the drop down field. The layer names are automatically filtered in the drop-down list, based on the direction set in the technology data. 
	Width	Specify the width of the stripes for each direction. You can specify a different width for each net within a set. You can specify different widths for different layers. Make sure that you specify reasonable spacing and width values in the GUI. If, for example, you use a width value that is smaller than the minimum width value for that particular layer, Innovus will throw Warning or Error messages when the script is run. If this happens, you can modify the values in the Innovus scripts based on the recommendation in the log file or the LEFDefaultRouteSpec Constraint Group values.

	Spacing	Specifies the spacing between stripes in each set. You can specify different spacing between each pair of stripes. 

	Sets Control	Use this drop-down to control how many stripe sets are created: • *Number of sets* - Select this option if you want to create a specific number of stripe sets. Specify the required number in the adjacent text box. The spacing between sets is then automatically derived by diving the total area by this number. • *Spacing between sets* - Select this option if you want to have a specific distance (pitch) from the reference stripe of one set to the reference stripe of the next set. Specify the required distance in the adjacent text box in micrometers. The number of sets is then automatically derived based on this value.  
	Creation Area	Specify the area in which the stripes are to be created. By default, the stripe sets are created within the entire design boundary. If you want to create stripes within a specific area, choose *Select Layout Area* from the *Creation Area* drop-down list, click the *Specify Box* button, and draw a box in the layout cellview for the required area. The lower-left and upper-right coordinates of the box are populated in the *LL* and *UR* fields, respectively. 
	Start Offset	Specify the starting offset value from the appropriate boundary of the stripe generation area to the nearest edge of the first stripe. In some designs, the top-level stripe may break at the block boundary. In this case, make sure that the stripe generated in the block aligns with the top-level stripe by using the *Start Offset* option, which is based on the `-start_offset` option of the `addStripe` command. If the P/G nets are on the same layer inside the block as they are at the top level, you can use

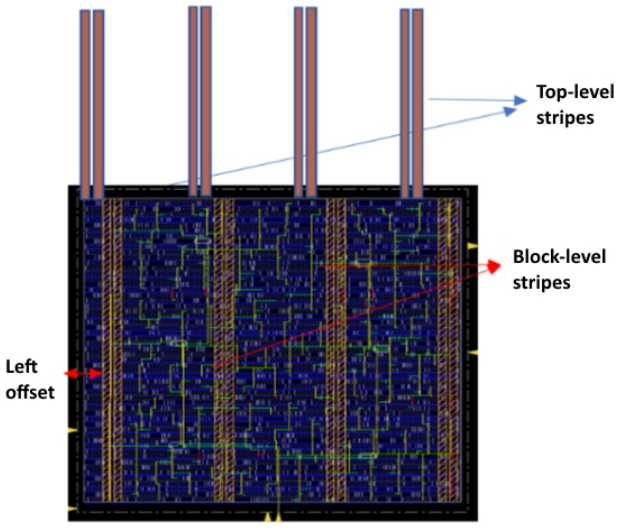
the *Start Offset* option to ensure that the stripes are continuous. Also, make sure that the global connectivity of the top-level and the block level nets match.

Top and block-level nets and stripe layers are the same



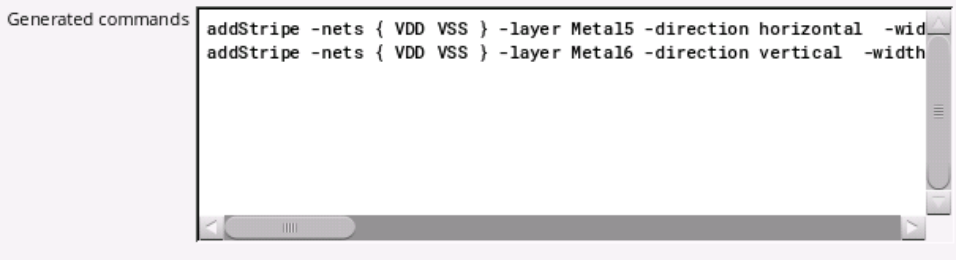
If the net names of the top and bottom are different, use the `-start_offset` option to make sure that the block stripes are generated away from the top-level stripes to avoid any potential DRCs.

Top and block-level nets and stripe layers are different



Note that for vertical stripes, the offset is measured from the left side of the stripe generation area to the left edge of the first stripe in the `addStripe` command created by the VDI interface (`-start_from left`). Similarly, for horizontal stripes, the offset is measured from the bottom of the stripe generation area to the bottom edge of the first stripe (`-start_from bottom`). You can edit the `-start_from` option in the `addStripe` commands in the *Generated Commands* section, if required.

Refer to latest *Innovus Text Command Reference* for details on using the `-*_offset` and `-start_from` options of the `addStripe` command.

	Stop Offset	Specify the stopping offset value off the appropriate boundary of the stripe generation area. This option is based on the `-stop_offset` option of the `addStripe` command.
	Apply	Click this button to create the `addStripe` command based on the settings you have specified in the *Power stripes* section. You can create multiple `addStripe` commands by changing settings in the *Power stripes* section and then clicking *Apply.*
	Generated commands	This section displays all the `addStripe` commands you have created.  You can review, edit, and delete commands from this section. The final commands in this section are added to the Innovus script generated by the VDI interface.
Power routing	Specify the settings for the power routes in this section as described below.	
	Block pin net names	Specify the net names of the block pins to be connected by the power routing engine, Special Route (Sroute), in this field. You can specify multiple net names separated by a space as shown below.  The interface auto-fills the power and ground net names that have been specified in the main GUI.
	Block pin top layer *Block pin bottom layer*	Specify the top and bottom layer to be used in routing. If you leave these fields blank, Sroute will try to route all power/ground pin layers of block pins to the closest power/ground stripe. 
	Note: If the digital block has only standard cells and no blocks in the floorplan, the power routing section can be ignored. Innovus, by default, will only route the standard cell pins (followpins). Based on the values you specify in the *Power routing* section of the form, the VDI interface generates commands, such as the following, in the Innovus script: `sroute -connect { corePin } -nets { VDD VSS }` #this command routes only followpins `sroute -connect { corePin blockPin} -nets { VDD VSS } -blockPinLayerRange { Metal5(5) Metal8(8) }` #this command routes both block pin and followpins	

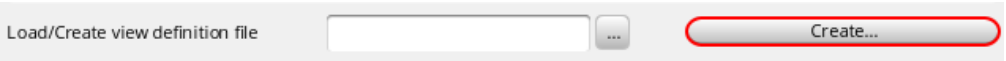
6.4 Creating the View Definition File

The view definition file is required by Innovus to complete many of the steps in physical implementation of a particular block. It describes the modes and corner the analysis engine in Innovus should use when evaluating timing. In its simplest form, the view definition file could just define the clock in the design. In a more complex form, it could define multiple mode and corners for the physical implementation flow. To see an example of a view definition file, click **here**.

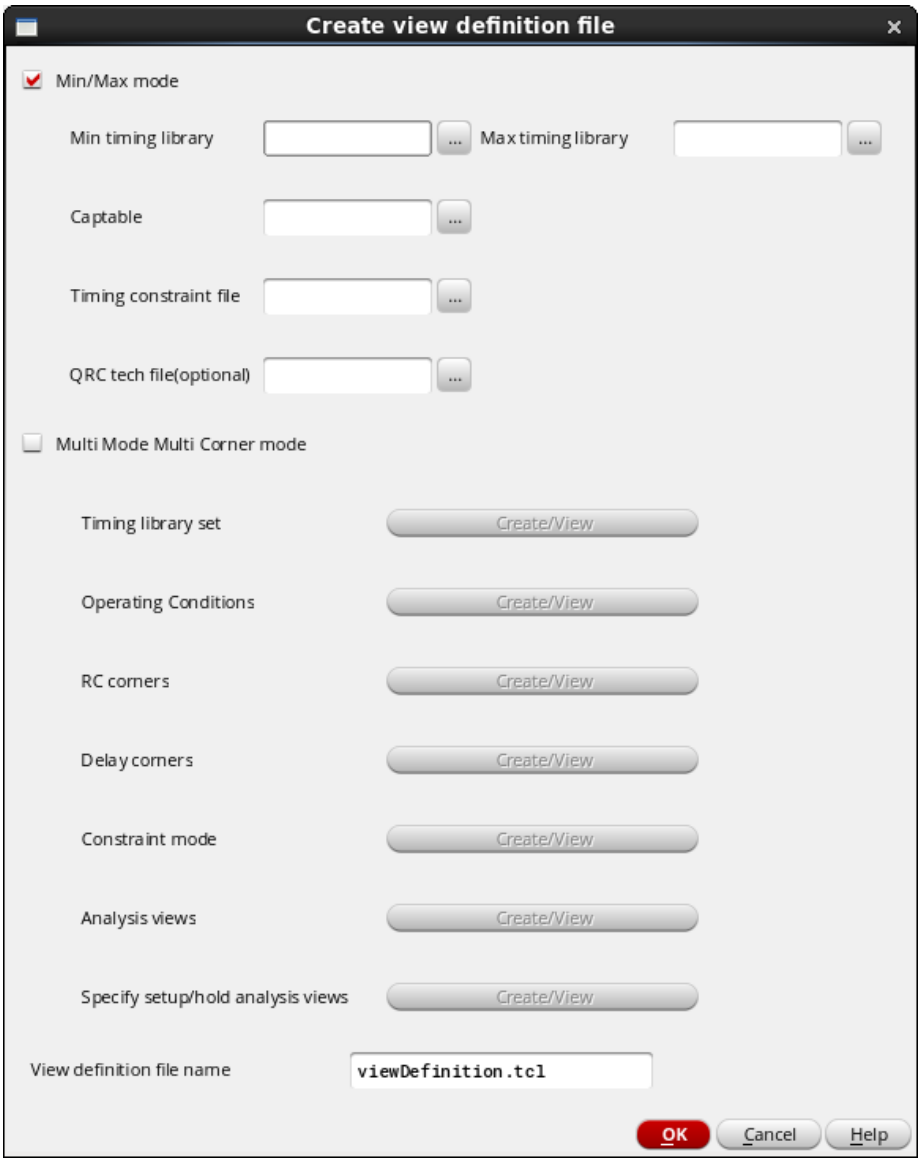
The VDI interface supports the following modes for the physical implementation of a digital block:

- Single timing analysis mode implementation
- Best case/Worst case (BcWc or Min/Max) analysis mode implementation
- Full multi-mode multi-corner (MMMC) analysis mode implementation

The mode you choose for implementing the digital block is controlled through a view definition file. To create a view definition file from the VDI interface, click the *Create* button next to the *Load/Create view definition file* field in the Run Innovus Implementation System form:

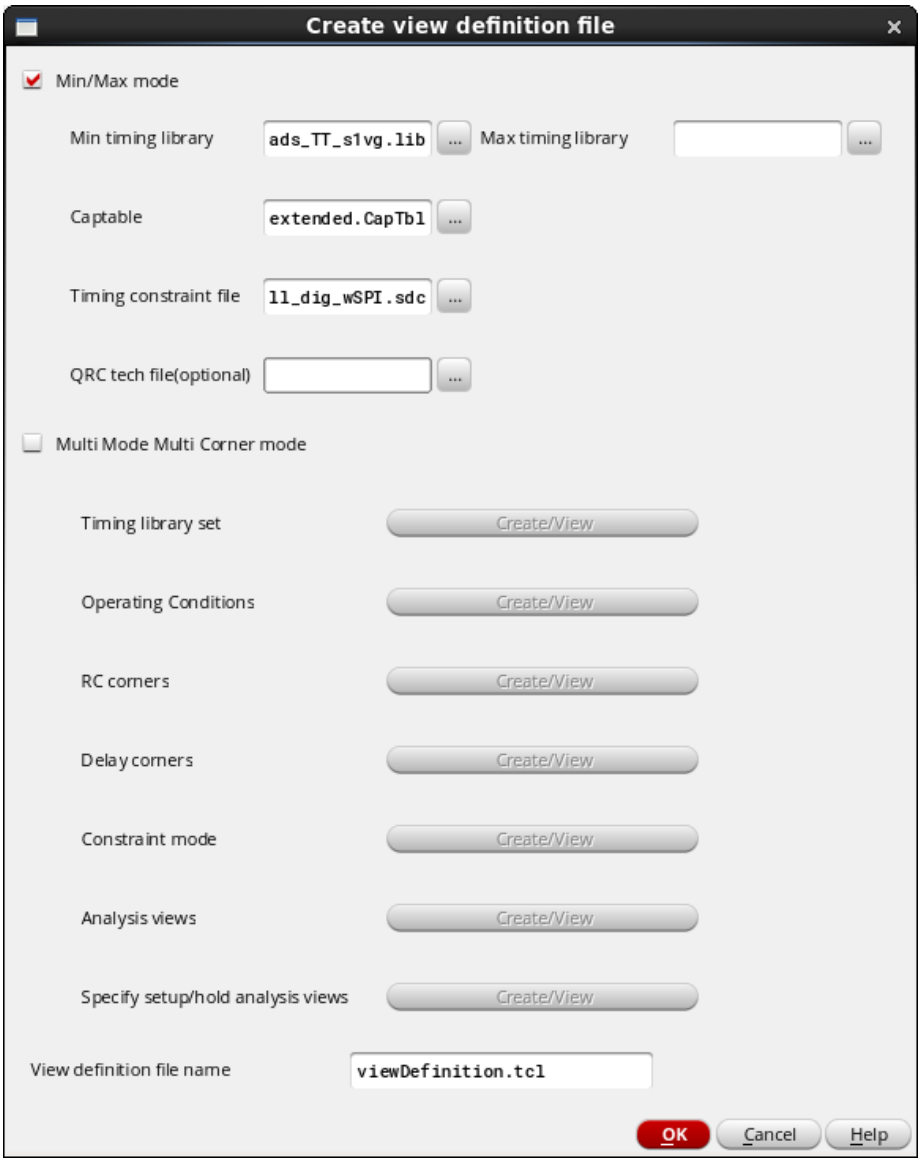


This opens the Create view definition file form:



6.4.1 Implementing a Digital Block in the Single Timing Analysis Mode

In the single timing analysis mode, Innovus uses a single set of delays using one library group. To invoke this mode, choose the *Min/Max mode* in the Create view definition file form of the VDI interface, but specify only one timing library (either *Min timing library* or *Max timing library*) instead of two libraries as shown below:



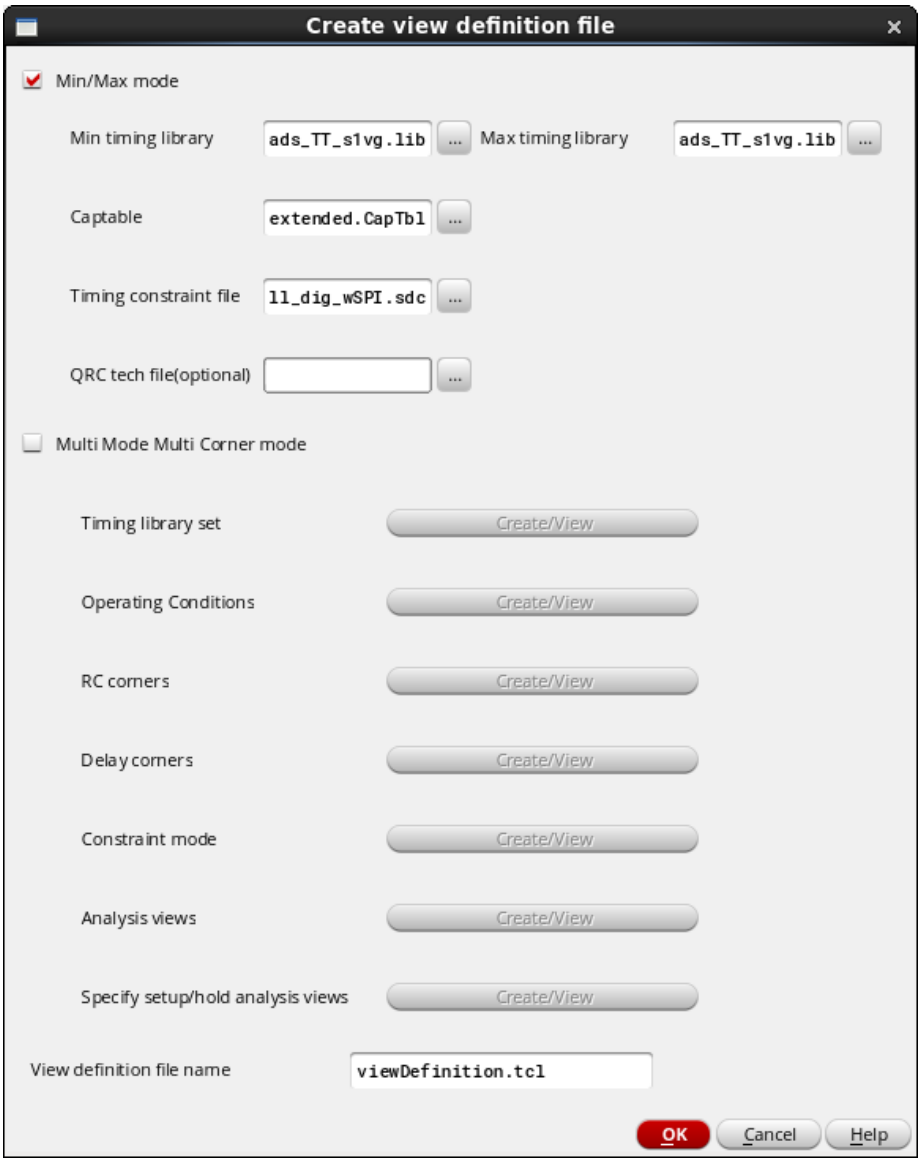
In addition to the path to a *Min timing library* (or a *Max timing library*), you can specify the following options in the Create view definition file form while implementing a digital block in the single timing analysis mode:

Min/Max mode		
	Captable	Specify the capacitance table to be used in the implementation in this field. The capacitance table and the QRC tech file are used in the parasitic extraction portion of the implementation flow. If only one of the files is available, you could specify just a single file. If only the capacitance table file is provided, the parasitic extraction step in the flow will run in effort level `Low`.
	Timing constraint file	Specify the required timing constraint file here. The timing constraint file (.sdc) is created during the logic synthesis part of the implementing the digital block.

	QRC tech file (optional)	Specify the name of the Quantus QRC Technology file here. If the QRC tech file is provided, the parasitic extraction will use effort level `High` for a more accurate extraction.
View definition file name	Specify a name for the view definition file you are creating here. The default name is `viewDefinition.tcl`.	

6.4.2 Implementing a Digital Block in the BcWc or Min/Max Analysis Mode

In the Best case/Worst case (BcWc or Min/Max) analysis mode, Innovus uses two set of delays, one from the maximum library group and the other from the minimum library group, in a timing analysis run. To invoke this mode, choose the *Min/Max mode* in the Create view definition file form of the VDI interface, and specify both minimum and maximum timing libraries as shown below:

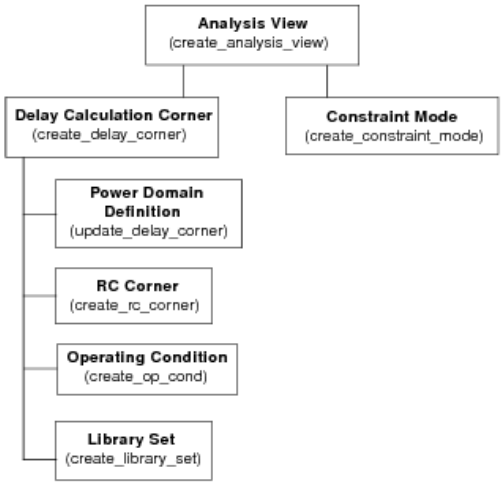


In addition to specifying paths to both *Min timing library* and *Max timing library*, you can specify the following options in the Create view definition file form while implementing a digital block in the BcWc or Min/Max analysis mode:

Min/Max mode		
	Captable	Specify the capacitance table to be used in the implementation in this field. The capacitance table and the QRC tech file are used in the parasitic extraction portion of the implementation flow. If only one of the files is available, you could specify just a single file. If only the capacitance table file is provided, the parasitic extraction step in the flow will run in effort level `Low`.
	Timing constraint file	Specify the required timing constraint file here. The timing constraint file (.sdc) is created during the logic synthesis part of the implementing the digital block.
	QRC tech file (optional)	Specify the name of the Quantus QRC Technology file here. If the QRC tech file is provided, the parasitic extraction will use effort level `High` for a more accurate extraction.
View definition file name	Specify a name for the view definition file you are creating here. The default name is `viewDefinition.tcl`.	

6.4.3 Implementing a Digital Block in the MMMC Analysis Mode

In the full multi-mode multi-corner (MMMC) analysis mode, Innovus uses a tiered approach to assemble the information necessary for timing analysis and optimization. Each top-level definition is called an analysis view. The diagram below depicts the various elements required for creation of analysis views. Each analysis view is composed of a delay calculation corner and a constraint mode:

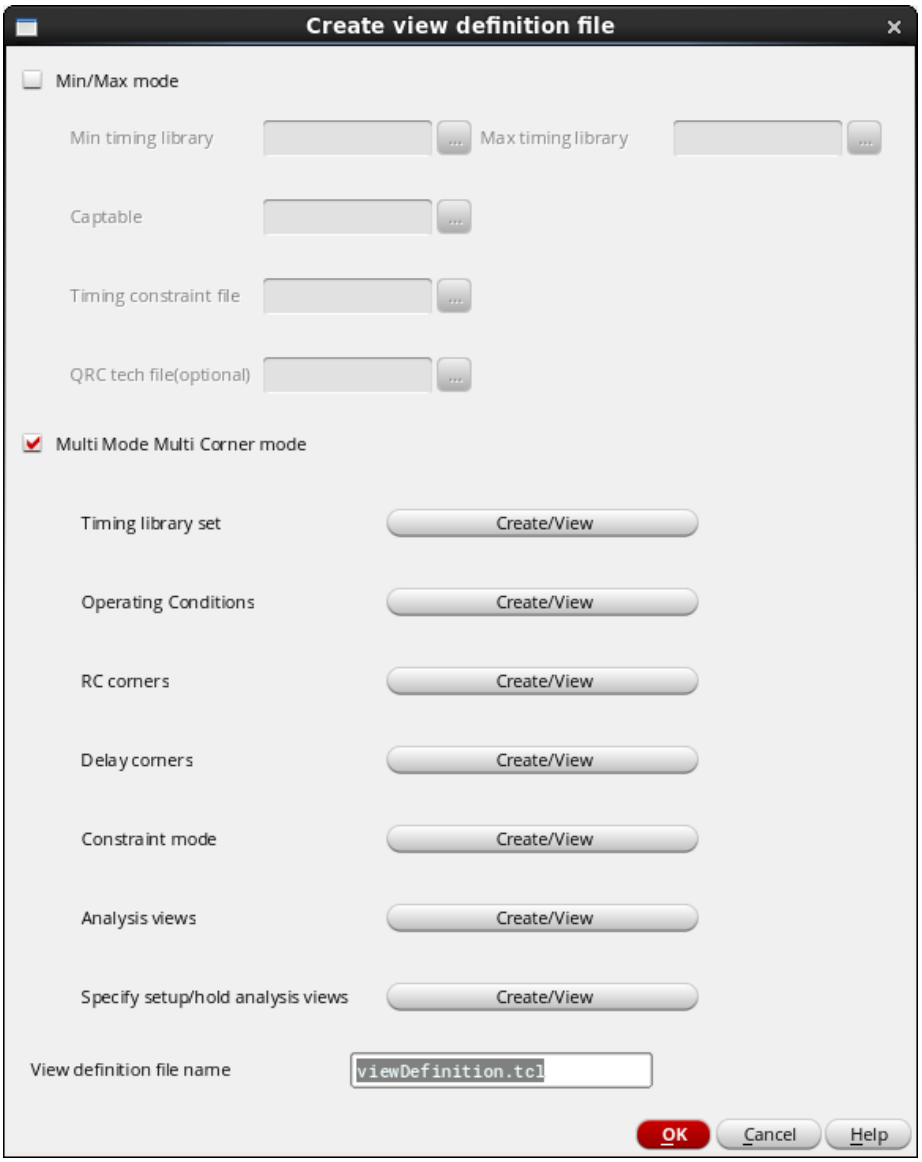


The VDI interface guides you through the creation of all required entries in the view definition file for an MMMC implementation. Once all analysis views are created using the interface, you can select the active analysis views to be used for representing different variations that will be analyzed during the implementation of the digital block.

Note - In the first version of the VDI interface, the script generation supported one power domain. As such, power domain definitions are not needed for creating the view definition file, as shown in the diagram.

To invoke the multi-mode multi-corner (MMMC) analysis mode, select the *Multi Mode Multi Corner mode* option in the Create view definition file form of the VDI interface. When you do so, the options for creating or viewing timing library sets, operating

conditions, RC corners, delay corners, constraint mode, and analysis views are enabled as shown below:



The following sections describe the procedure for creating or viewing timing libary sets, operating conditions, RC corners, delay corners, constraint modes, and analysis views:

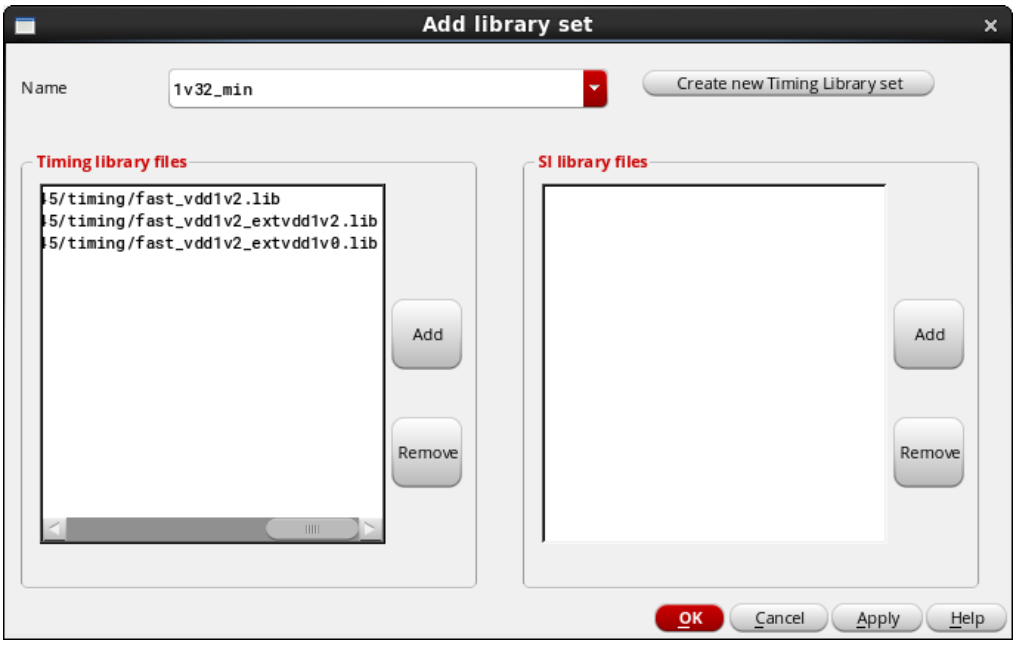
- Creating/viewing timing library sets
- Creating/viewing operating conditions
- Creating/viewing RC corners
- Creating/viewing delay corners
- Creating/viewing constraint modes
- Creating/viewing analysis views
- Specifying setup/hold analysis views

You can specify a name for the view definition file you are creating in the *View definition file name* field. The default name

is `viewDefinition.tcl`.

6.4.3.1 Creating/viewing timing library sets

Timing library sets are the basic elements of a view definition file. Typical designs contain standard cells, memories, IPs, pads, and so on. Library sets allow a group of library files to be treated as a single entity so that higher-level descriptions, such as delay corners, can simply refer to the library configuration by name. A library set may contain only timing libraries or may also include `.cdb` libraries for running signal Integrity. To create a timing library set, click the *Create/View* button next to *Timing library set* in the Create view definition file form of the VDI interface. This launches the Add library set form.



To complete creating timing library sets:

1. Specify a name for the set in the *Name* field.
2. Add the timing and SI library files provided to you for implementing the digital block.
3. Click *Apply*.
4. Click *Create new Timing Library set* to reset the fields and repeat steps 1 to 3 to add another library set.
5. When you have completed defining library sets, click *OK*.

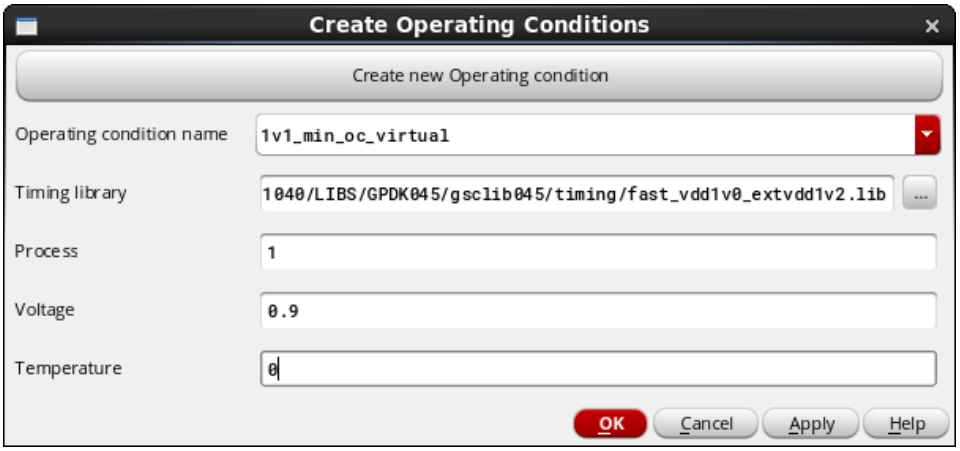
Whenever required, you can view any of the library set you have created by selecting its name from the *Name* drop-down list in the Add library set form.

Note - The order in which you define timing libraries is important. The software considers the first library you specify in the list as the master library, with each successive library having a lower priority.

6.4.3.2 Creating/viewing operating conditions

In digital designs, there are situations when there are no predefined operating conditions in the timing libraries being used for implementing the design, or the pre-existing operating conditions in the timing libraries are not consistent with the operating environment of the design. In these cases, it is important to specify the desired operating conditions, so Innovus could use the data to derate the data in the timing libraries as well as perform other aspects of analysis. 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 process, voltage, and temperature (PVT) operating point. As far as Innovus is concerned, it would see these 'virtual' operating conditions, as if they actually existed in the timing library.

To create operating conditions, click the *Create/View* button next to *Operating conditions* in the Create view definition file form of the VDI interface. This launches the Create Operating Conditions form.



To complete creating operating conditions:

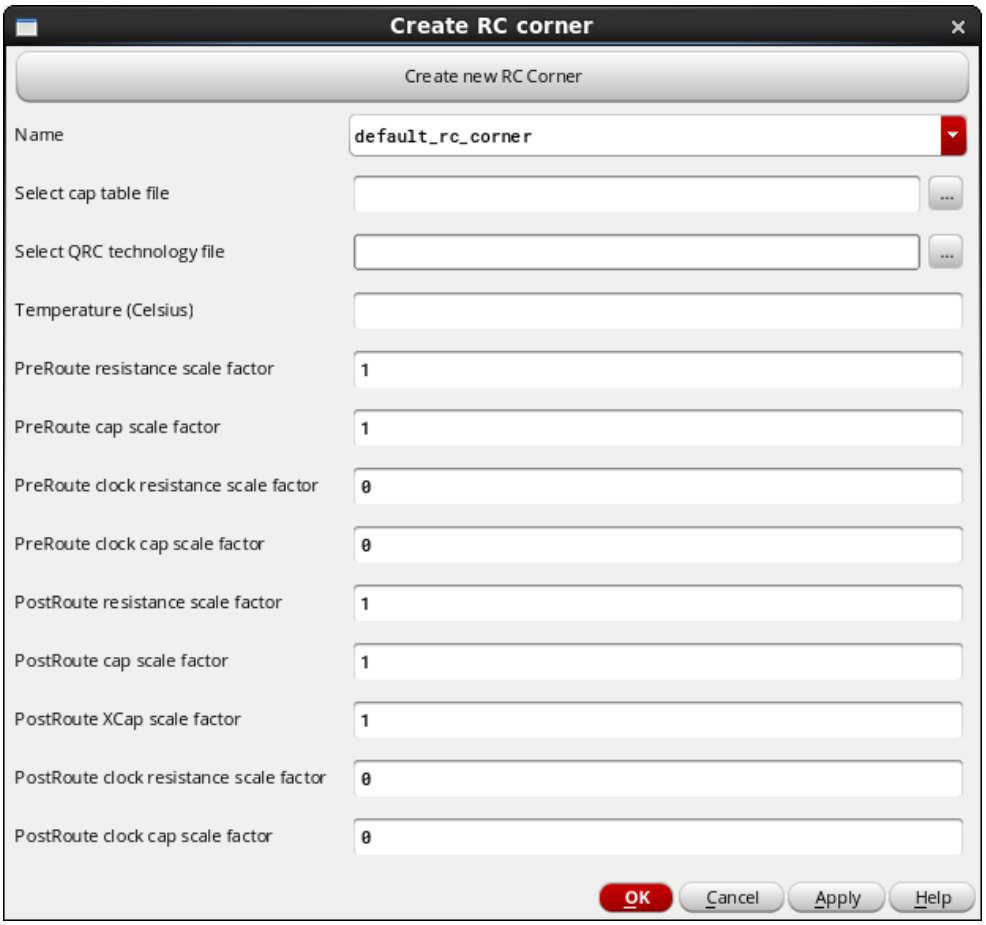
1. Specify a name for the set in the *Operating condition name* field.
2. Select the timing library file to be used in the operating condition. Note that this timing library must be one of the timing libraries specified while creating library sets in the previous step.
3. Specify the desired PVT values.
4. Click *Apply*.
5. Click *Create new Operating condition* to reset the fields and repeat steps 1 to 4 to create a new operating condition.
6. When you have completed defining operating conditions, click *OK*.

Whenever required, you can view any of the operating conditions you have created by selecting its name from the *Operating condition name* drop-down list in the Create Operating Conditions form.

6.4.3.3 Creating/viewing RC corners

An RC corner provides Innovus all the information it needs to properly extract the required parasitic information for implementing the digital design. RC corners are also used for sign-off extraction after the design is fully implemented. Innovus extracts and stores a unique set of parasitics for each RC corner that you identify as an active corner. Note that you can choose which of the RC corners you create are active. The RC corners that are not identified as active are not used in analysis.

To create RC corners, click the *Create/View* button next to *RC corners* in the Create view definition file form of the VDI interface. This launches the Create RC corner form:

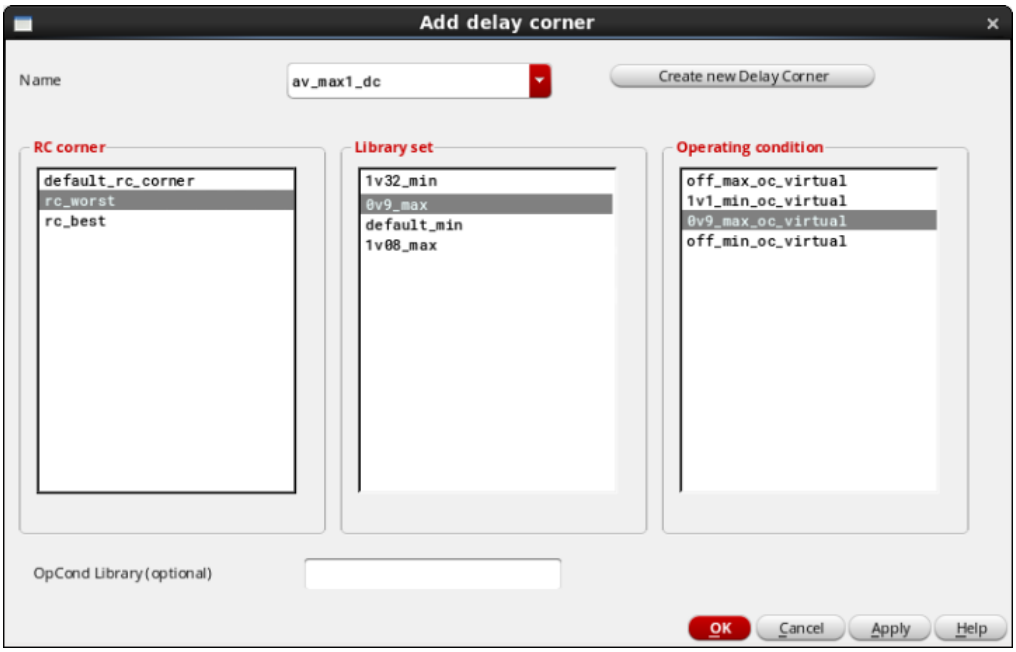


As can be seen in the form, different derating factors can be used in the creation of the RC corner. If no derating of the RC data is desired, you can simply provide the capacitance table and the Quantus QRC technology file, and leave all other fields to default.

6.4.3.4 Creating/viewing delay corners

You can create the delay calculation corner to be used by Innovus after you have completed creating the timing library sets, operating conditions, and RC corners. In addition to library sets, operating conditions, and RC corners, a delay calculation corner can also include power domain definitions. However, as mentioned earlier, this version of the VDI interface creates Innovus scripts only for designs with a single power domain. Therefore, you do not need to create power domain definition at present for creating a delay corner.

To create delay corners, click the *Create/View* button next to *Delay corners* in the Create view definition file form of the VDI interface. This launches the Add delay corner form:



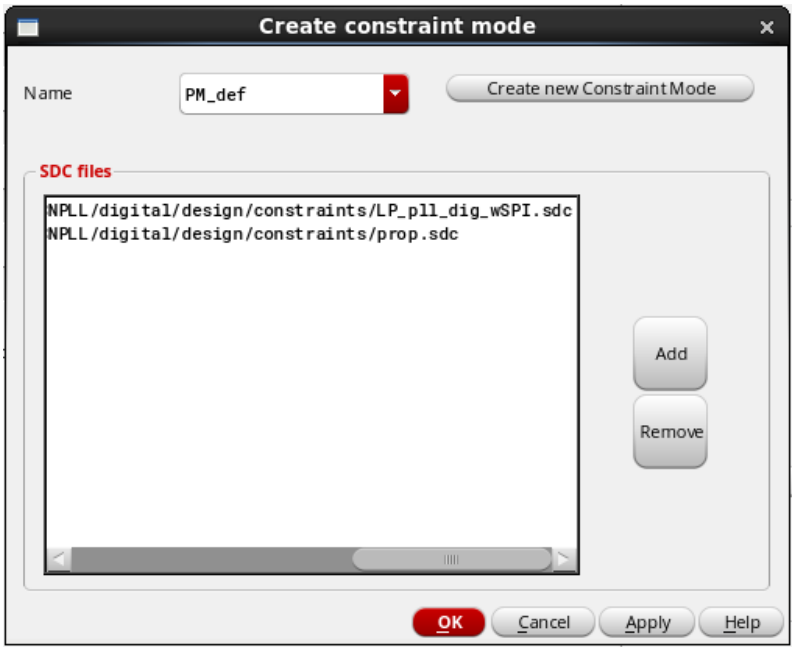
To create delay corners:

1. Select the desired RC corner, Library set, and Operating condition.
2. Specify the name of the delay calculation corner in the *Name* field.
3. Click *Apply*.
4. Click *Create new Delay Corner* to reset the fields and repeat steps 1 to 3 to create another delay corner.
5. When you have completed defining delay corners, click *OK*.

6.4.3.5 Creating/viewing constraint modes

A constraint mode defines one of the many possible modes of operation of a design. For example, when implementing the digital block using the VDI interface, you may be interested in analyzing the timing of the block in the functional and test modes. For designs with multiple power domains, another possible mode of operation is Dynamic Voltage and Frequency Scaling (DVFS). To get the most accurate analysis results, different timing constraint (SDC) files are used for the different modes in which the digital block will function. A constraint mode consists of a list of SDC 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. As an example, the test mode of the digital block will have `set_false_path` for the paths that are not executed in this mode. Click **here** to view an example of a simple SDC file for a digital block.

To create constraint modes, click the *Create/View* button next to *Constraint mode* in the Create view definition file form of the VDI interface. This launches the Create constraint mode form:



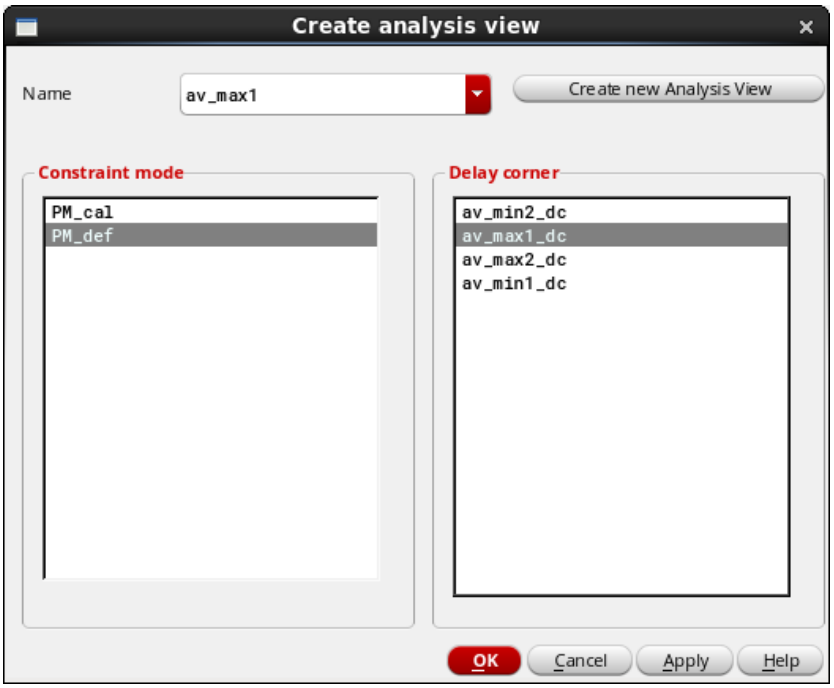
To create constraint modes:

1. Specify the name of the new constraint mode in the *Name* field.
2. Add the required SDC files that you want to include in this mode.
3. Click *Apply*.
4. Click *Create new Constraint Mode* to reset the fields and repeat steps 1 to 3 to create another constraint mode.
5. When you have completed creating constraint modes, click *OK*.

6.4.3.6 Creating/viewing analysis views

An analysis view is a combination of a delay calculation mode and a constraint mode. The analysis view is the object that actually controls the multi-mode multi-corner (MMMC) analysis that takes place in Innovus.

To create analysis views, click the *Create/View* button next to *Analysis views* in the Create view definition file form of the VDI interface. This launches the Create analysis view form:

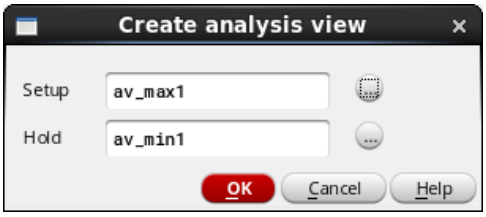


Now, simply select the constraint mode/delay corner combination, specify a name, and click on the *Apply* button.

6.4.3.7 Specifying setup/hold analysis views

In this stage of creating an MMMC view definition file, the analysis views created in the previous step can be used to specify which of the analysis views created in the previous step can be used by the analysis engine during setup and hold analysis.

To specify setup/hold analysis views, click the *Create/View* button next to *Specify setup/hold analysis views* in the Create view definition file form of the VDI interface. This launches the Create analysis view form:

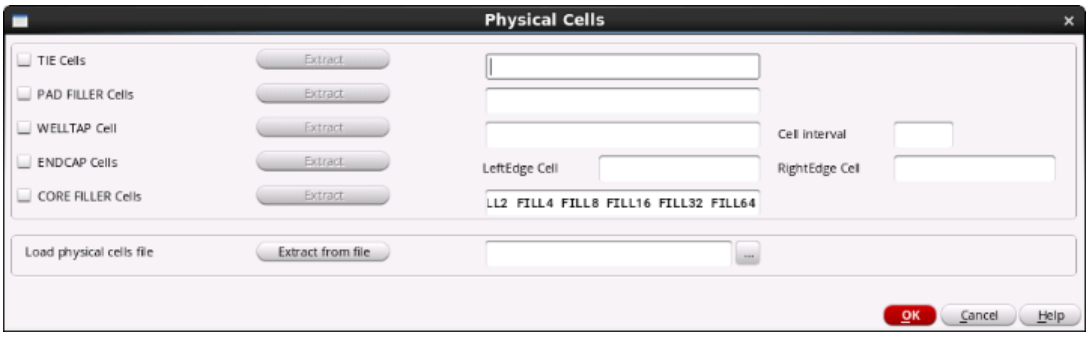


Click the ... button next to the *Setup* and *Hold* fields to select the desired views from the list of available analysis views.

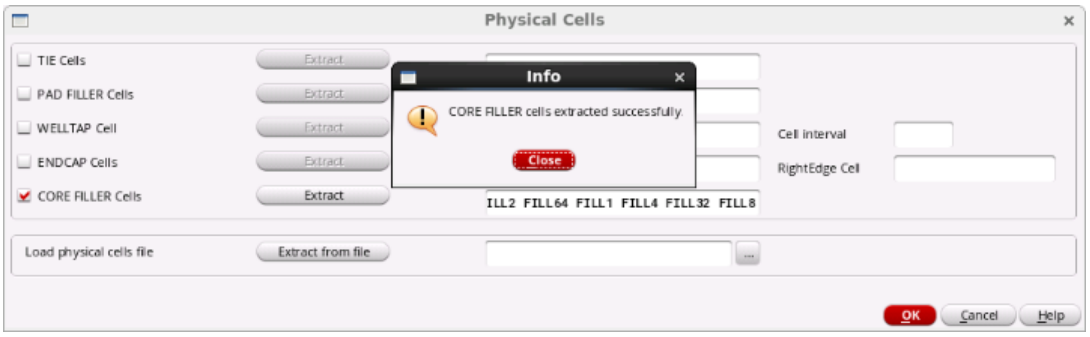
6.5 Specifying Physical Cells

To extract physical cells from the specified OpenAccess reference libraries or LEF files, click the *Get All Physical Cells* button on the Run Innovus Implementation System form. This opens the Physical Cells form, which you can use to extract the required physical cells (TIE, PAD FILLER, WELLTAP, ENDCAP, and CORE FILLER cells) from the specified OpenAccess reference libraries or LEF files.

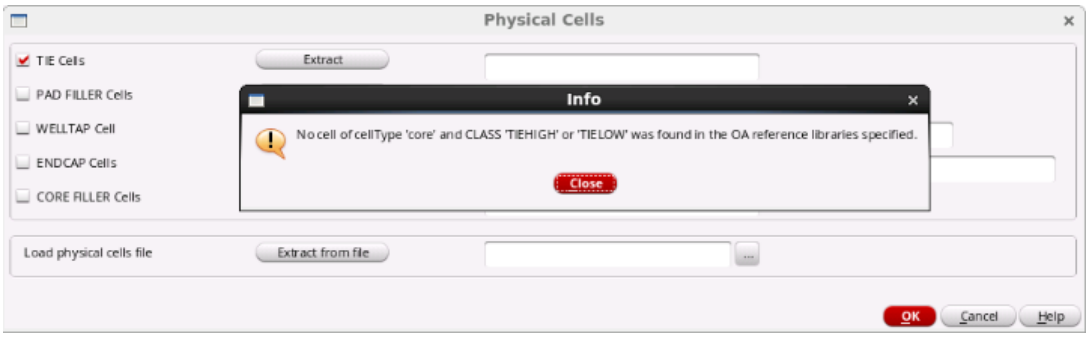
Note - The extraction of the physical cells depends on the cellType being set properly in the LEF files or the OpenAccess reference libraries.



The Physical Cells form provides an *Extract* button for each physical cell type. To extract a specific type of physical cell, select the check box next to it and click the corresponding *Extract* button. The tool extracts the available physical cells of that type and displays them in an editable text box. You can update the list of cells, as required.



If physical cells of the selected type are not available in the specified OpenAccess reference libraries or LEF files, the VDI interface displays a message as shown below:



Instead of extracting each cell type individually, you can choose to load all physical cells from a specified file. To do so, select the browse button (...) corresponding to the *Load physical cells file* option in the new Physical Cells to select the file containing the cell names and then click the *Extract from file* button.

A typical physical cells file will have the following format:

```
Tie Cells: cell_a cell_b cell_c ...

CORE Filler Cells:cell_d cell_e cell_f ...

PAD Filler Cells: cell_g cell_h cell_i ...

LeftEndCap Cells: cell_j cell_k ...

RightEndCap Cells: cell_l cell_m

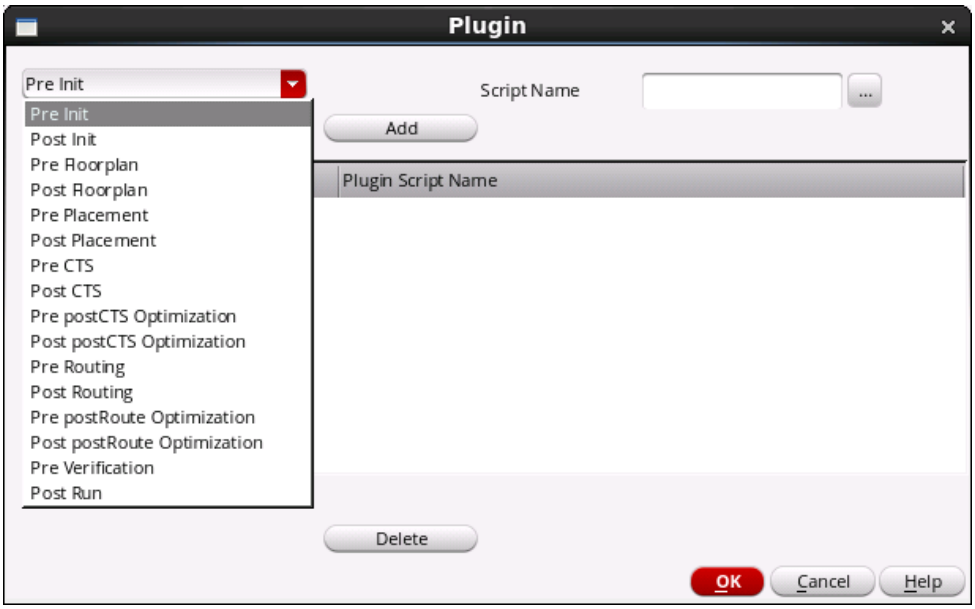
WellTap Cells: cell_n cell_o
```

Note - If you load the cells from a file, it will override all the cells extracted or specified previously. However, you can choose to

load a file and then extract a particular physical cell type.

6.6 Specifying Optional Plugin Scripts

If you want to plug in your own code in the Innovus script, click the *Specify Scripts* button on the Run Innovus Implementation System form. This opens the Plugin form, which provides options for adding plugin files at various stages in the Innovus script as shown below:



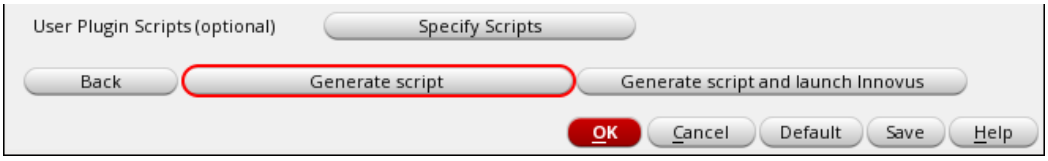
Select the required stage from the drop-down list, specify the path to the script in the *Script Name* field and click *Add*. Your script is plugged in at the specified stage into the Innovus script generated by the VDI interface.

6.7 Generating the Innovus Script

After you have made all the necessary settings in the Run Innovus Implementation System form, you need to invoke the script generation part of the VDI interface. You can choose to generate the script and customize it before launching Innovus or generate the script and launch Innovus in a single step.

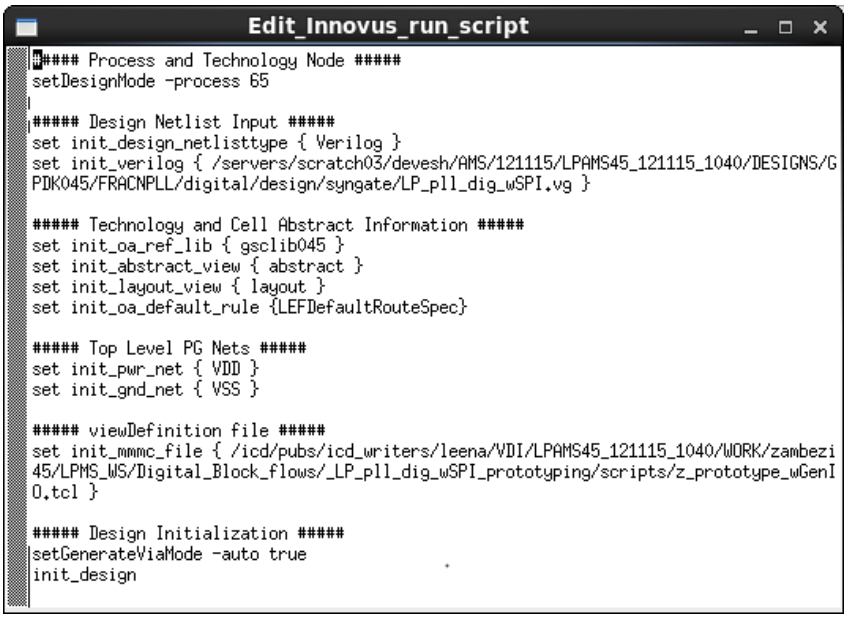
6.7.1 With Customization

If you need to customize the script before launching Innovus, click the *Generate script* button.



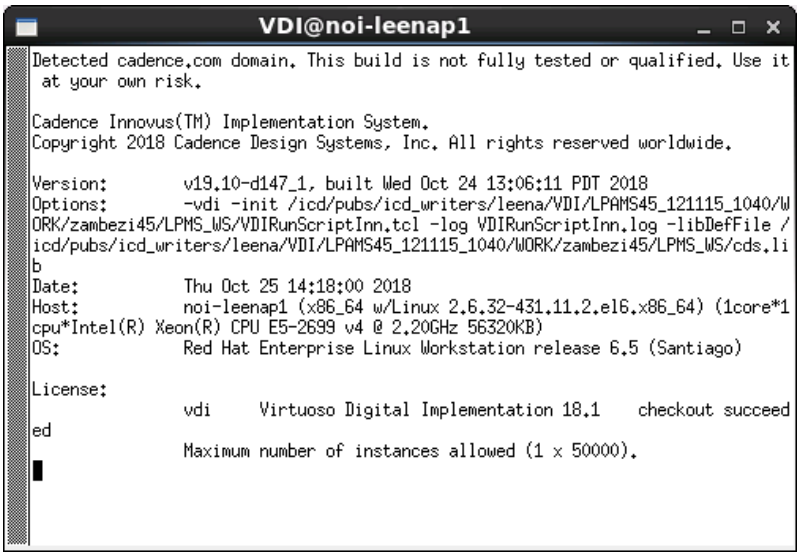
The generated script is displayed in a vi text editor window, allowing you to customize it, if required. The script contains all the necessary steps required to implement a design using Innovus, based on the inputs you have specified. See the **Generated**

Innovus Script Sample section to view an example of a generated Innovus script.



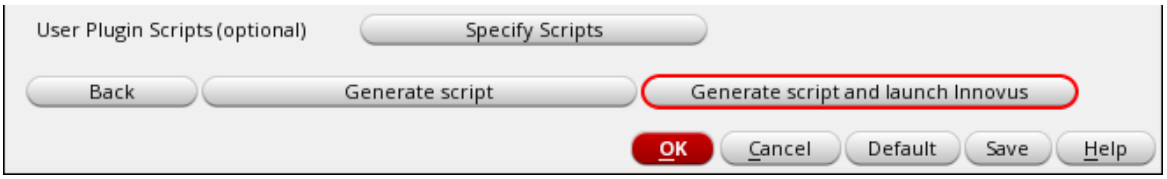
After making any necessary edits, save the script so that it can be reloaded, if required, for later runs.

Next, click the *OK* button in the Run Innovus Implementation System form. This launches the Innovus executable in a separate window.

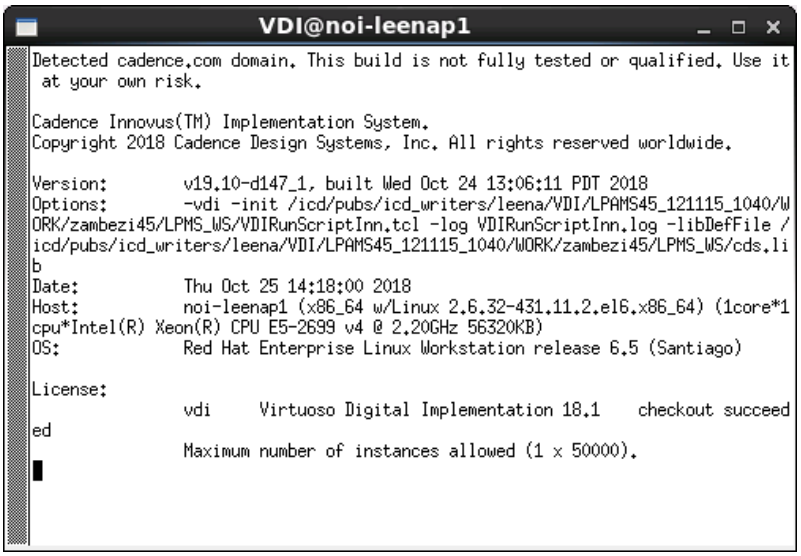


6.7.2 Without Customization

If you do not need to customize the script generated by the VDI interface in any way, you can generate the script and launch Innovus in a single step by clicking the *Generate script and launch Innovus* button instead.

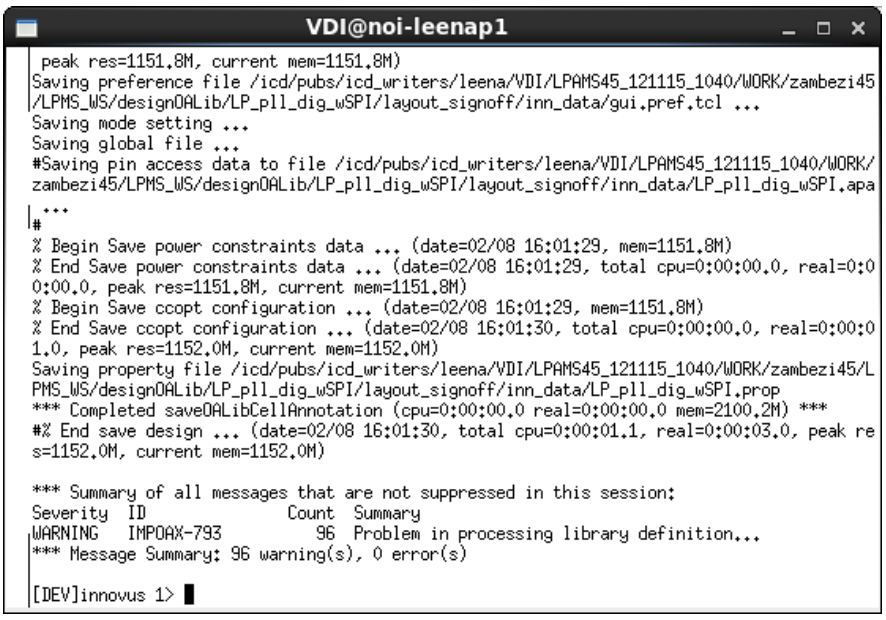


On clicking this button, the Run Innovus Implementation System form is closed automatically and the Innovus executable is launched in a separate window.

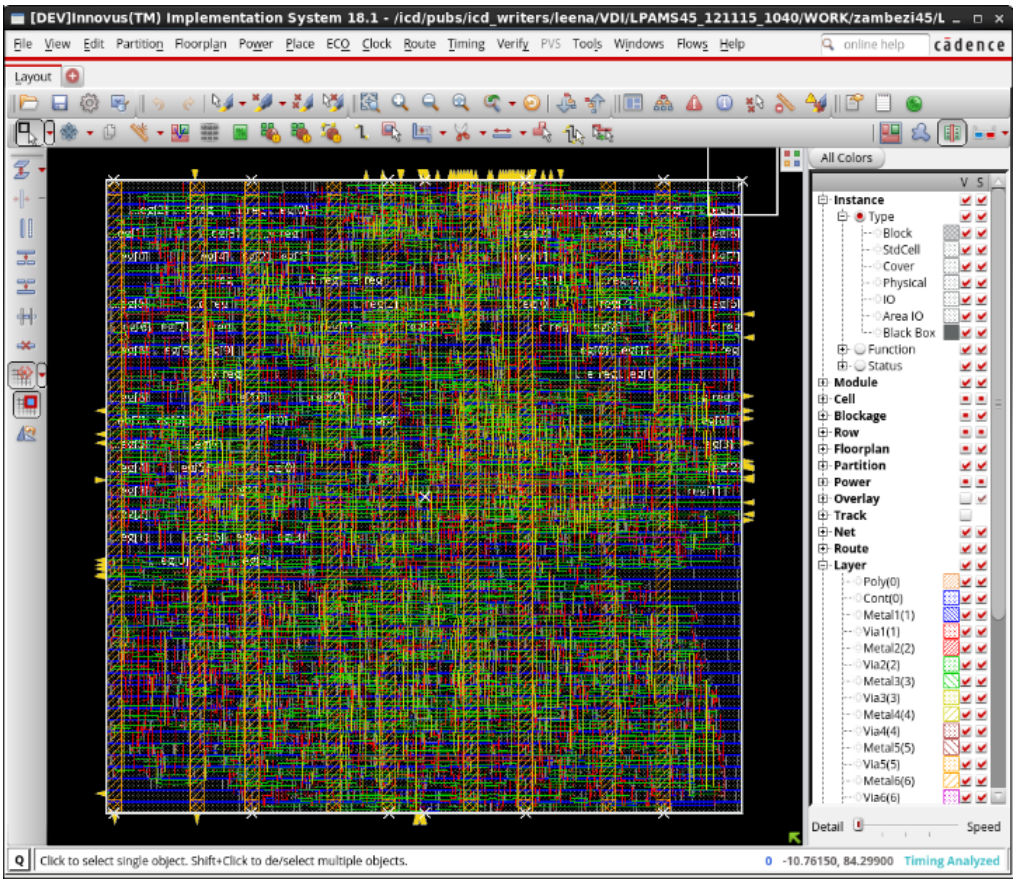


6.8 Completing the Implementation

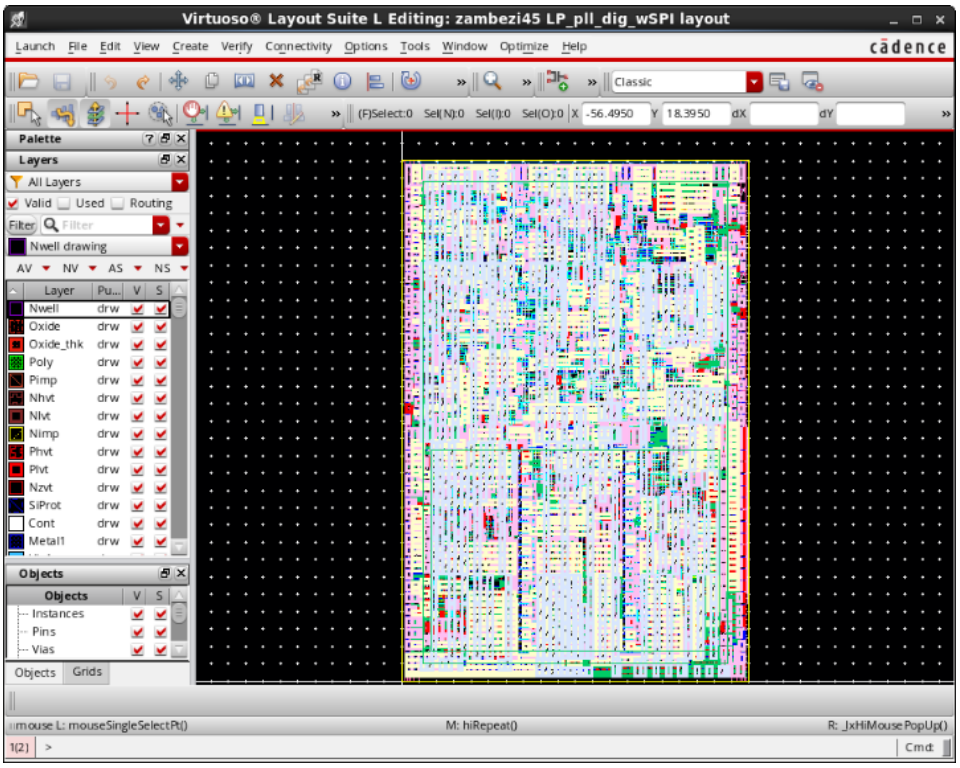
Innovus proceeds to run all the commands in the generated script. Once the complete script has been processed, the `innovus>` prompt is displayed.



Type `win` at the `innovus >` prompt to launch the Innovus GUI and view the fully implemented digital block:



The implemented digital block can also be viewed in Virtuoso:



6.9 Sample Files

6.9.1 Sample View Definition File

Here is an example of a view definition file:

```
create_library_set -name 1v32_min\
   -timing\
    [list //LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v2.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v2_extvdd1v2.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v2_extvdd1v0.lib]
create_library_set -name 1v1_min\
   -timing\
    [list /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v0_extvdd1v2.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v0_extvdd1v0.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v0.lib]\
   -si\
    [list /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/celtic/fast.cdb]
create_library_set -name 0v9_max\
```

```
   -timing\
    [list /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v0_extvdd1v2.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v0_extvdd1v0.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v0.lib]\
   -si\
    [list /icd/hierflow/devesh/VDIFlow/LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/celtic/slow.cdb]
create_library_set -name 1v08_max\
   -timing\
    [list /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v2.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v2_extvdd1v2.lib\
    /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v2_extvdd1v0.lib]
create_op_cond -name off_max_oc_virtual -library_file
//LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v0_extvdd1v2.lib -P 1 -V 0 -T 125
create_op_cond -name 1v1_min_oc_virtual -library_file
/LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v0_extvdd1v2.lib -P 1 -V 0.9 -T 0
create_op_cond -name 0v9_max_oc_virtual -library_file
/LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/slow_vdd1v0_extvdd1v2.lib -P 1 -V 0.9 -T 125
create_op_cond -name off_min_oc_virtual -library_file
/LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/timing/fast_vdd1v0_extvdd1v2.lib -P 1 -V 0 -T 0
create_rc_corner -name default_rc_corner\
   -preRoute_res 1\
   -postRoute_res 1\
   -preRoute_cap 1\
   -postRoute_cap 1\
   -postRoute_xcap 1\
   -preRoute_clkres 0\
   -preRoute_clkcap 0
create_rc_corner -name rc_best\
   -cap_table /LPAMS45_121115_1040/TECH/GPDK045/gpdk045/soce/gpdk045.extended.CapTbl\
   -preRoute_res 1\
   -postRoute_res {1 1 1}\
   -preRoute_cap 1\
   -postRoute_cap {1 1 1}\
   -postRoute_xcap {1 1 1}\
   -preRoute_clkres 0\
   -preRoute_clkcap 0\
   -T 0\
   -qx_tech_file /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/qrc/qx/gpdk045.tch
```

```
create_rc_corner -name rc_worst\
   -cap_table  /LPAMS45_121115_1040/TECH/GPDK045/gpdk045/soce/gpdk045.extended.CapTbl\
   -preRoute_res 1\
   -postRoute_res {1 1 1}\
   -preRoute_cap 1\
   -postRoute_cap {1 1 1}\
   -postRoute_xcap {1 1 1}\
   -preRoute_clkres 0\
   -preRoute_clkcap 0\
   -T 125\
   -qx_tech_file /LPAMS45_121115_1040/LIBS/GPDK045/gsclib045/qrc/qx/gpdk045.tch
create_delay_corner -name av_max1_dc\
   -library_set 0v9_max\
   -opcond_library slow_vdd1v0_extvdd1v2\
   -opcond 0v9_max_oc_virtual\
   -rc_corner rc_worst
update_delay_corner -name av_max1_dc -power_domain PD_def\
   -library_set 0v9_max\
   -opcond_library slow_vdd1v0_extvdd1v2\
   -opcond 0v9_max_oc_virtual
update_delay_corner -name av_max1_dc -power_domain PD_cal\
   -library_set 0v9_max\
   -opcond_library slow_vdd1v0_extvdd1v2\
   -opcond off_max_oc_virtual
create_delay_corner -name av_min2_dc\
   -library_set 1v1_min\
   -opcond_library fast_vdd1v0_extvdd1v2\
   -opcond 1v1_min_oc_virtual\
   -rc_corner rc_best
update_delay_corner -name av_min2_dc -power_domain PD_def\
   -library_set 1v1_min\
   -opcond_library fast_vdd1v0_extvdd1v2\
   -opcond 1v1_min_oc_virtual
update_delay_corner -name av_min2_dc -power_domain PD_cal\
   -library_set 1v1_min\
   -opcond_library fast_vdd1v0_extvdd1v2\
   -opcond 1v1_min_oc_virtual
```

```
create_delay_corner -name av_max2_dc\
   -library_set 0v9_max\
   -opcond_library slow_vdd1v0_extvdd1v2\
   -opcond 0v9_max_oc_virtual\
   -rc_corner rc_worst
update_delay_corner -name av_max2_dc -power_domain PD_def\
   -library_set 0v9_max\
   -opcond_library slow_vdd1v0_extvdd1v2\
   -opcond 0v9_max_oc_virtual
update_delay_corner -name av_max2_dc -power_domain PD_cal\
   -library_set 0v9_max\
   -opcond_library slow_vdd1v0_extvdd1v2\
   -opcond 0v9_max_oc_virtual
update_delay_corner -name av_max2_dc -power_domain PD_cal\
   -library_set 0v9_max\
   -opcond_library slow_vdd1v0_extvdd1v2\
   -opcond 0v9_max_oc_virtual
create_delay_corner -name av_min1_dc\
   -library_set 1v1_min\
   -opcond_library fast_vdd1v0_extvdd1v2\
   -opcond 1v1_min_oc_virtual\
   -rc_corner rc_best
update_delay_corner -name av_min1_dc -power_domain PD_def\
   -library_set 1v1_min\
   -opcond_library fast_vdd1v0_extvdd1v2\
   -opcond 1v1_min_oc_virtual
update_delay_corner -name av_min1_dc -power_domain PD_cal\
   -library_set 1v1_min\
   -opcond_library fast_vdd1v0_extvdd1v2\
   -opcond off_min_oc_virtual
create_constraint_mode -name PM_def\
   -sdc_files\
    [list /LPAMS45_121115_1040/DESIGNS/GPDK045/FRACNPLL/digital/design/constraints/LP_pll_dig_wSPI.sdc\
     /LPAMS45_121115_1040/DESIGNS/GPDK045/FRACNPLL/digital/design/constraints/prop.sdc]
create_constraint_mode -name PM_cal\
   -sdc_files\
```

```
    [list
/icd/hierflow/devesh/VDIFlow/LPAMS45_121115_1040/DESIGNS/GPDK045/FRACNPLL/digital/design/constraints/LP_pll_dig_wSPI.sdc\
     /LPAMS45_121115_1040/DESIGNS/GPDK045/FRACNPLL/digital/design/constraints/prop.sdc]
create_analysis_view -name av_max1 -constraint_mode PM_def -delay_corner av_max1_dc
create_analysis_view -name av_max2 -constraint_mode PM_cal -delay_corner av_max2_dc
create_analysis_view -name av_min1 -constraint_mode PM_def -delay_corner av_min1_dc
create_analysis_view -name av_min2 -constraint_mode PM_cal -delay_corner av_min2_dc
set_analysis_view -setup [list av_max1 av_max2] -hold [list av_min1 av_min2]
```

6.9.2 Sample SDC File

```
### Create clock for CTRLCLK domain#
create_clock -name ctrlclk -period 50 [get_ports ctrlclk]
set_input_delay  10 -clock ctrlclk [all_inputs]
set_output_delay 10 -clock ctrlclk [all_outputs]
### Create clock for DSMCLK domain#
create_clock -name dsmclk -period 50 [get_ports dsmclk]
set_output_delay 10 -clock dsmclk [get_ports {ls_ndiv*}]
### Create clock for SCLK domain#
create_clock -name SCLK -period 50 [get_ports SCLK]
set_input_delay  -clock_fall 10 -clock SCLK [get_ports {SI}]
set_output_delay 10 -clock SCLK [get_ports {SO}]
### Create clock for SV_n domain#
create_clock -name SV_n -period 200 [get_ports SV_n]
### Create false path(s) between clock domains#
set_false_path -from ctrlclk -to dsmclk
set_false_path -from dsmclk -to ctrlclk
set_false_path -from SCLK -to ctrlclk
set_false_path -from ctrlclk -to SCLK
set_false_path -from SCLK -to dsmclk
set_false_path -from dsmclk -to SCLK
### Create false path(s) on reset signals#
set_false_path -from [get_ports {rst_n}]
#set_false_path -through [get_pins {u_clkgen_rst_ref_n_reg/Q}]
#set_false_path -through [get_pins {u_clkgen_rst_dsm_n_reg/Q}]
### Set timing on input/output signals#
```

```
set_driving_cell -lib_cell BUFX2 [all_inputs]

set_load -pin_load 0.02 [all_outputs]
```

6.9.3 Generated Innovus Script Sample

An example of the generated Innovus script is shown below. As can be seen, the script contains all the necessary steps required to implement a design using Innovus.

```
##### Process and Technology Node #####

setDesignMode -process 45


##### Design Netlist Input #####

set init_design_netlisttype { Verilog }

set init_verilog {
/icd/hierflow/devesh/VDIFlow/LPAMS45_121115_1040/DESIGNS/GPDK045/FRACNPLL/digital/design/syngate/LP_pll_dig_wSPI.v
g }


##### Technology and Cell Abstract Information #####

set init_oa_ref_lib { gsclib045 }

set init_abstract_view { abstract }

set init_layout_view { layout }

set init_oa_default_rule {LEFDefaultRouteSpec}


##### Top Level PG Nets #####

set init_pwr_net { VDD }

set init_gnd_net { VSS }


##### viewDefinition file #####

set init_mmmc_file {
/icd/hierflow/devesh/VDIFlow/LPAMS45_121115_1040/WORK/zambezi45/LPMS_WS/Digital_Block_flows/_LP_pll_dig_wSPI_proto
typing/scripts/viewDefinition.tcl }


##### Design Initialization #####

setGenerateViaMode -auto true

init_design


##### Load Scan Chain and Floorplan Data #####

# The following commands can be used to read design floorplan information.

# defIn design.def
```

```
# loadFPlan design.fp

# oaIn <lib> <cell> <view>

# The following commands can also be added to create core margins for the floorplan.

# changeFloorplan -coreToLeft 2.0

# changeFloorplan -coreToBottom 3.0

# changeFloorplan -coreToRight 2.0

# changeFloorplan -coreToTop 3.0

defIn
/icd/hierflow/devesh/VDIFlow/LPAMS45_121115_1040/DESIGNS/GPDK045/FRACNPLL/digital/design/scandef/pll_dig_wSPI.scan
def


##### Global Net Connections #####

globalNetConnect VDD -pin VDD

globalNetConnect VSS -pin VSS


##### Pre-Placement Timing Check #####

timeDesign -preplace -prefix preplace -outDir RPT

checkDesign -all

check_timing

set topCell [dbget top.name]

catch { createLib designOALib -referenceTech gsclib045 }

saveDesign -cellView " designOALib $topCell layout_init "


##### Power Planning Commands #####

addStripe -nets { VDD VSS } -layer Metal6 -direction vertical  -width 2.00  -spacing 10.00  -set_to_set_distance
20.00  -start_from left -switch_layer_over_obs false -max_same_layer_jog_length 2 -use_wire_group 0 -
snap_wire_center_to_grid None -skip_via_on_pin {  standardcell } -skip_via_on_wire_shape {  followpin }

sroute -nets { VDD VSS }  -connect {blockPin corePin } -blockPinTarget { nearestTarget }  -corePinTarget {
firstAfterRowEnd} -allowJogging 1 -allowLayerChange 1 -blockPin useLef


##### Placement and Pre-CTS Optimization #####

# Optionally, endcap and welltap cells can be added. Sample commands are given below.

# addEndCap -prefix PwrCap

# addWellTap -cell TAP -cellInterval 28 -prefix WELLTAP

setPlaceMode -place_global_place_io_pins true -place_global_ignore_scan 1

# setOptMode -usefulSkew true

place_opt_design -out_dir RPT -prefix place
```

```
##### Tie-hi and Tie-lo Cells #####
# Specify Tie-hi and Tie-lo Cells
# setTieHiLoMode -cell "TIEHI TIELO"
# addTieHiLo
saveDesign -cellView " designOALib $topCell layout_placed "

#####  Clock Tree Synthesis (CTS) #####
setAnalysisMode -analysisType onChipVariation -cppr both
set_ccopt_mode  -integration "native" -ccopt_modify_clock_latency true
#setNanoRouteMode -route_with_litho_driven true
create_ccopt_clock_tree_spec
ccopt_design -outDir RPT -prefix cts
saveDesign -cellView " designOALib $topCell layout_cts "

#####  Post-CTS Hold Fixing #####
optDesign -postCTS -hold -outDir RPT -prefix postcts_hold
saveDesign -cellView " designOALib $topCell layout_postcts_hold "

##### Filler Insertion #####
# Specify the Commands to add Filler Cells.
# setFillerMode -core "FILL64 FILL32"\
#    -corePrefix FILL
# addFiller

##### Global and Detailed Routing #####
routeDesign
saveDesign -cellView " designOALib $topCell layout_routed "

##### Post Route Timing Optimization & Hold Fixing #####
setDelayCalMode -engine aae
optDesign -postRoute -outDir RPT -prefix postroute -setup -hold
saveDesign -cellView " designOALib $topCell layout_postRoute_hold "

##### RC Extraction #####
setExtractRCMode -engine postRoute -coupled true -effortLevel high
extractRC
```

```
##### Signoff Timing Check #####

timeDesign -prefix signoff -signoff -reportOnly -outDir RPT

timeDesign -prefix signoff -signoff -reportOnly -hold -outDir RPT

summaryReport -outDir RPT


##### Verify Checks #####

verifyConnectivity -noAntenna -report ${topCell}_verify_conn.rpt

verify_drc -report ${topCell}_verify_drc.rpt

verifyMetalDensity -report ${topCell}_verify_density.rpt

verifyProcessAntenna -report ${topCell}_verify_antenna.rpt

saveDesign -cellView " designOALib $topCell layout_signoff "
```

6.10 Helpful Hints

- The 18.10 release of the VDI environment does not have support for digital blocks with multiple power domains.
- You may encounter the following messages from Innovus while running the script generated by the VDI interface:

  ```
  **WARN: (IMPCCOPT-1183):   The library has no usable balanced buffers for power domain auto-default, while balancing clock_tree clk125. If this is not intended behavior, you can specify a list of lib_cells to use with the buffer_cells property.
  **WARN: (IMPCCOPT-1184):   The library has no usable balanced inverters for power domain auto-default, while balancing clock_tree clk125. If this is not intended behavior, you can specify a list of lib_cells to use with the inverter_cells property.
  **ERROR: (IMPCCOPT-1135):   CTS found neither inverters nor buffers while balancing clock_tree clk125. CTS cannot continue.
  ```

  The above warning/error could be reported during clock tree synthesis and optimization step of the script. These messages may indicate an issue with the technology library being used. The clock tree optimization engine is unable to find the right buffer/Inverters for clock tree optimization. The reason could be that Innovus is unable to find the properly balanced buffers to use in the clock tree. You can add the following two settings to the generated script to force Innovus to use certain buffers/inverters:

  ```
  set_ccopt_property buffer_cells {<list of cell names separated by a space>}
  set_ccopt_property inverter_cells {<list of cell names separated by a space>}
  ```

- If the top-level P/G net names are different from the block-level P/G pins, the following entry in the script should be modified to better match the design characteristic:

  ```
  ##### Global Net Connections #####

  globalNetConnect VDD -pin VDD
  globalNetConnect VSS -pin VSS
  ```

- The generated Innovus script has the tap cell insertion commented out, so libraries that do not have tap cells, would work without needing to modify the generated script. If tap cells are present in the library, please uncomment the following section of the script, and add the necessary tap cell names.
  # Optionally, endcap and welltap cells can be added. Sample commands are given below.

  ```
  # addEndCap -prefix PwrCap
  # addWellTap -cell TAP -cellInterval 28 -prefix WELLTAP
  ```

- No specific settings are included in the generated script for placement, routing, and timing analysis. Certain settings that might be of interest to the user, are included as comments in the script.

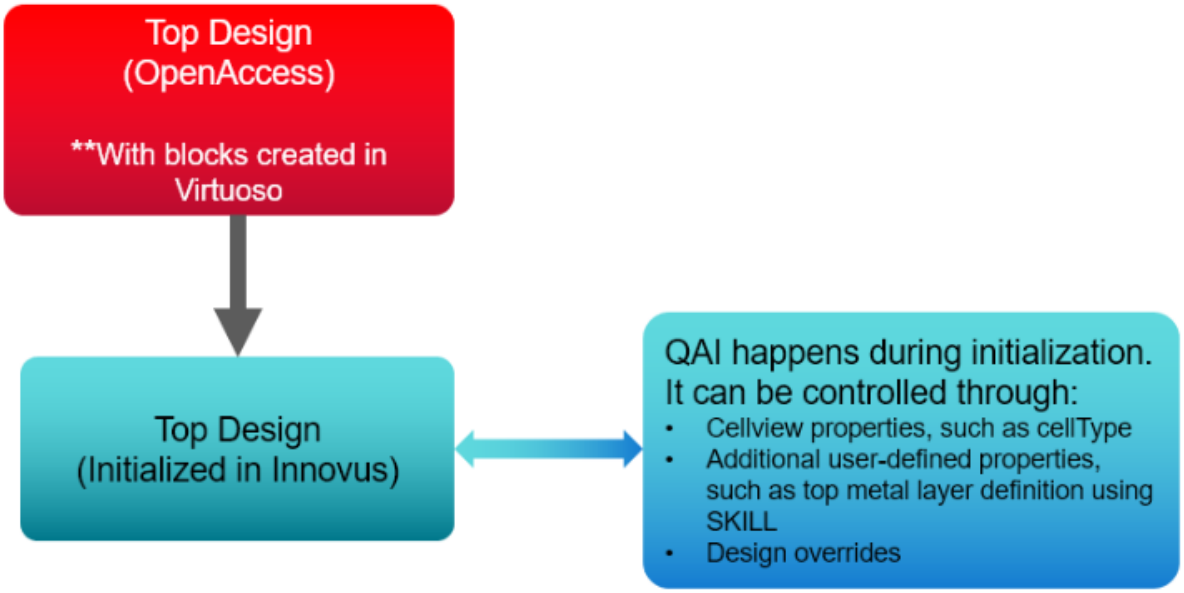
7

Quick Abstract Inference

- Overview of Quick Abstract Inference
- Why Abstracts Are Still Required for Standard Cells
- How QAI Works
  - Rules for Abstract Inference
  - Detailed Description of Data Objects
  - Blockage Modeling in QAI
    - Controlling QAI Through Cellview Properties
    - Controlling QAI Through User-defined Properties
    - Controlling QAI Using Design Overrides
  - QAI Pin Modeling
  - Antenna Annotation Utility for Creating Accurate Antenna Information for the Innovus Flow

7.1 Overview of Quick Abstract Inference

The Quick Abstract Inference (QAI) feature in Innovus is intended to support a specific use case in which a block-level designer in Virtuoso is working on a cellview and does not want to create a new abstract view after each update. The block designer will simply deliver the layout without abstract generation in Virtuoso. Innovus has the ability to create an on-the-fly abstract from a layout view if the abstract is not available in the OpenAccess database. The process of abstract generation happens during design initialization in Innovus, based on the physical data information in the cellview. Innovus can also control the abstract generation based on certain properties given either in the cellview or in the top OpenAccess design by the Virtuoso user.



7.2 Why Abstracts Are Still Required for Standard Cells

Many foundries create optimized standard cell abstracts, which are then included as part of the Virtuoso Process Design Kit (PDK) or added during the creation of an OpenAccess PDK (MSOA PDK). The standard cell abstracts are needed by tools in the Place-and-Route environment to optimize the performance of many steps, such as clock tree synthesis, placement, and routing. Other information in the standard cell abstracts is used by Innovus to perform various functions. For example, the properties of the standard cell abstracts help direct various implementation steps in Innovus. Here are some of the properties that are not generally present in the layout view of a standard cell, but are needed in the abstract to drive the Innovus implementation flow.

- The standard cell abstracts have `cellType == core`. If this property is not present in the abstracts, Innovus will treat the cells as `CLASS BLOCK` instead of core cells, which are `CLASS CORE`.
- The standard cell abstracts have a siteDef which is required for the placement step in Innovus.
- Often, the antenna information is not included in the layout view of a standard cell. The antenna information is used by NanoRoute to detect and fix antenna violations during routing.

For information on creation of an MSOA PDK, refer to the Technology Data Preparation chapter.

7.3 How QAI Works

During design initialization, Innovus reads the library cells (LEF equivalent) from the list of reference libraries provided by the `init_oa_ref_lib` variable.

As there can be many views of a cell, the name of the abstract view to look for is provided by the `init_abstract_view` variable. The OAX reader reads all the cells present in the reference libraries,

irrespective of whether or not these cells are used inside the design. This results in some performance penalty when there are many cells in the library that are not used in the design but are still read into Innovus. To avoid this performance issue, library cells are processed on demand. This means that if the library name of a particular cell is not defined in the list of reference libraries, Innovus reads that particular cell from its corresponding binding (lib/cell/view) in the OpenAccess design database. This on-demand processing helps improve performance by reading only those library cells that are actually required by the design. For example, IP block libraries are generally not provided in the reflib list and they are read using on-demand processing.

Another aspect of on-demand processing is determining the type of the view that is bound. The binding could point to an under-development IP (layout view) or its abstract equivalent. Innovus detects the view type automatically by applying heuristic methods. For the layout view, Innovus uses the Quick Abstract Inference (QAI) feature to infer internally the abstract equivalent from the layout information during design import. This process is called on-the-fly abstract inference.

A summary report is printed in the *design*`.oaread.rpt` file. This report contains the summary of all the blocks read by inferring the layout view of the block.

ⓘ For automatic abstract inference, pin shapes and `prBoundary` must be available in the layout view.

Innovus supports dual views. A dual view is a layout view that also contains abstract information, such as `prBoundary`, antenna, site, symmetry, and cell type, in the same cellview. A dual view can be used both as a layout in Virtuoso and as an abstract in Innovus through QAI.

Note: You need not generate abstracts each time you change the layout view of the block. To generate detailed abstracts after finalizing the block design, use the Virtuoso Abstract Generator.

7.3.1 Rules for Abstract Inference

The following table describes the behavior of auto-abstract inference when the following data objects are present in the layout view. To know more about these data objects, refer to Detailed Description of Data Objects.

Data objects in the cell layout view	**Behavior of Innovus if the data object is present in the layout view**	**Behavior of Innovus if the data object is not present in the layout view or its value is null**
Pin	All the top-level pin shapes are copied to the abstract view, along with shapes on the same layer that enclose the pin shapes. Check the QAI Pin Modeling section for details.	No pins appear in the abstract view. **Note:** Create pins as required at the top level.
Cell type	The specified type is considered. **Note:** If the cell type is `none`, then the `BLOCK` cell type is assigned by default.	The default type `BLOCK` is considered.

Cover Obstruction
Virtuoso commands for applying obstruction up to metal layer `Metalx`:
`cv=geGetEditCellView()`
`dbCreateCoverObstruction(cv~>prBoundary techGetLayerMaskNumber(techGetTechFile(cv) "Metalx"))`
`dbDeleteObject(cv~>prBoundary~>coverObstruction)`
	Obstructions are created only on the routing layers up to the `Metalx` routinglayer specified.
	Gets ID of currently open cellview.
	Specifies the top metal layer to be obstructed.
	Removes the cover blockage created by `dbCreateCoverObstruction`
		The tool automatically determines the top metal layer used in the design hierarchy.
		All layers up to the top layer are blocked.

prBoundary	Exact region as enclosed by `prBoundary` is taken.	`prBoundary` is required for the cell types `CORE` and `ENDCAP`. If you do not specify `prBoundary` for cells other than core, auto-abstract inference will set its value to zero. However, while reading this data, Innovus tries to auto compute the box for the cell from the pin shapes, if possible.
Symmetry	The specified symmetry option is taken. **Note:** If the symmetry value is `none`, then the default value `XYR90` is considered.	The default value `XYR90` is taken.
Site	The values specified in the layout are copied without any changes.	If the `SITE` value is not specified for cell types `CORE` and `ENDCAP`, an error message is displayed. For other cell types, it is left blank.

Antenna	The values specified in the layout are copied without any changes.	A warning message is displayed. This message describes the number of cells and missing antenna values for `SIGNAL`, `SCAN`, `CLOCK`, and `RESET` pins.

7.3.2 Detailed Description of Data Objects

The following table describes the data objects that are used for inferring the abstracts:

Data Object	Description
Pin	Specifies a physical shape through which one block connects to other blocks in a netlist.
Cell type	Specifies various cell types, such as `CORE`, `ENDCAP`, `COVER`, `RING`, `BLOCK`, and `PAD`.
`prBoundary`	Specifies the block boundary for place-and-route applications. Any shape or instance created for the block should be enclosed within the `prBoundary`.
Symmetry	Specifies the possible orientation of the block.
Site	Provides placement for the family of macros, such as I/O, core, block, analog, digital, short, tall, and so on, in a design.
Antenna	During deep sub-micron wafer fabrication, gate damage can occur when excessive static charges accumulate and discharge, passing high current through a gate. If the area of the layer connected directly to the gate or connected to the gate through lower layers is large relative to the area of the gate and the static charges are discharged through the gate, the discharge can damage the oxide that insulates the gate and cause the chip to fail. This phenomenon is called the process antenna effect (PAE). Run Virtuoso Abstract Generator to obtain the antenna value. For more information, see the LEF/DEF Language Reference.

7.3.3 Blockage Modeling in QAI

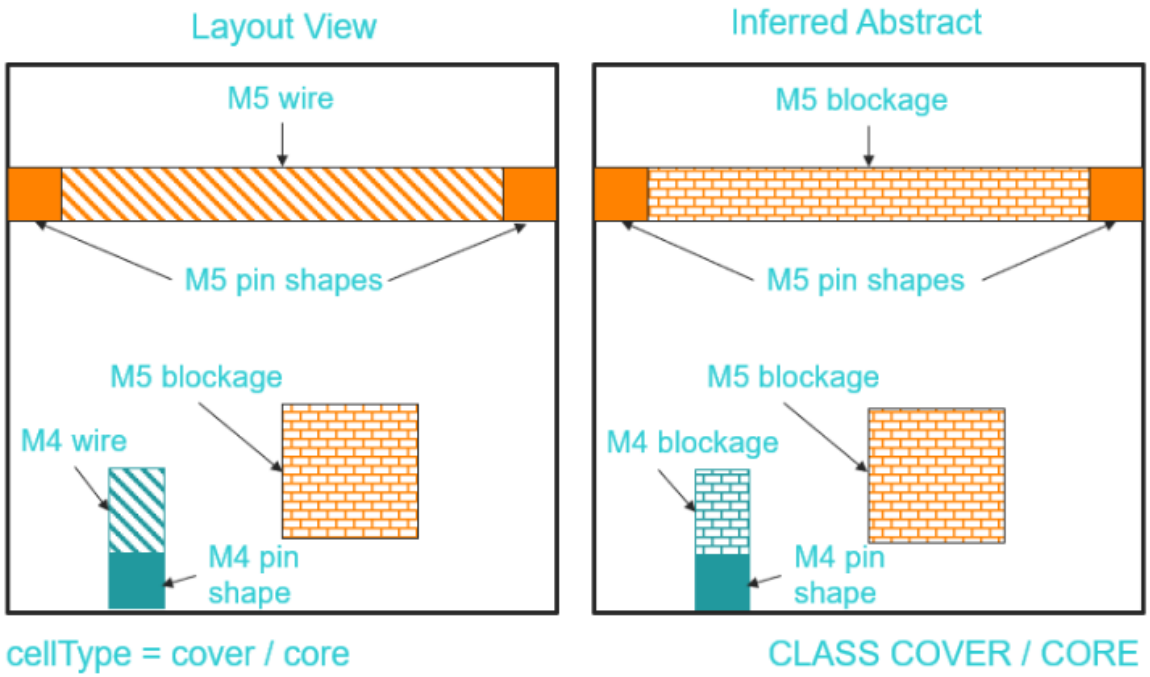
During the abstract generation process, Innovus models blockages in the following two ways:

- *Detailed modeling* - In this type of modeling, Innovus models each shape, such as a pin, wire, or blockage, exactly as it is in the generated abstract, and the *Obstruction* is limited to the bbox where shapes exist in the `prBoundary`.
- *Full obstruction modeling* - In this type of modeling, Innovus covers the whole block (excluding base layers) with an *Obstruction* over the `prBoundary` for any shapes that appear in the layout.

7.3.3.1 Controlling QAI Through Cellview Properties

You can control how abstracts are generated during QAI by using cellview properties, such as `cellType`.

- For `CORE` or `COVER` cell types, Innovus applies detailed modeling to blockages/shapes during QAI, as shown below.

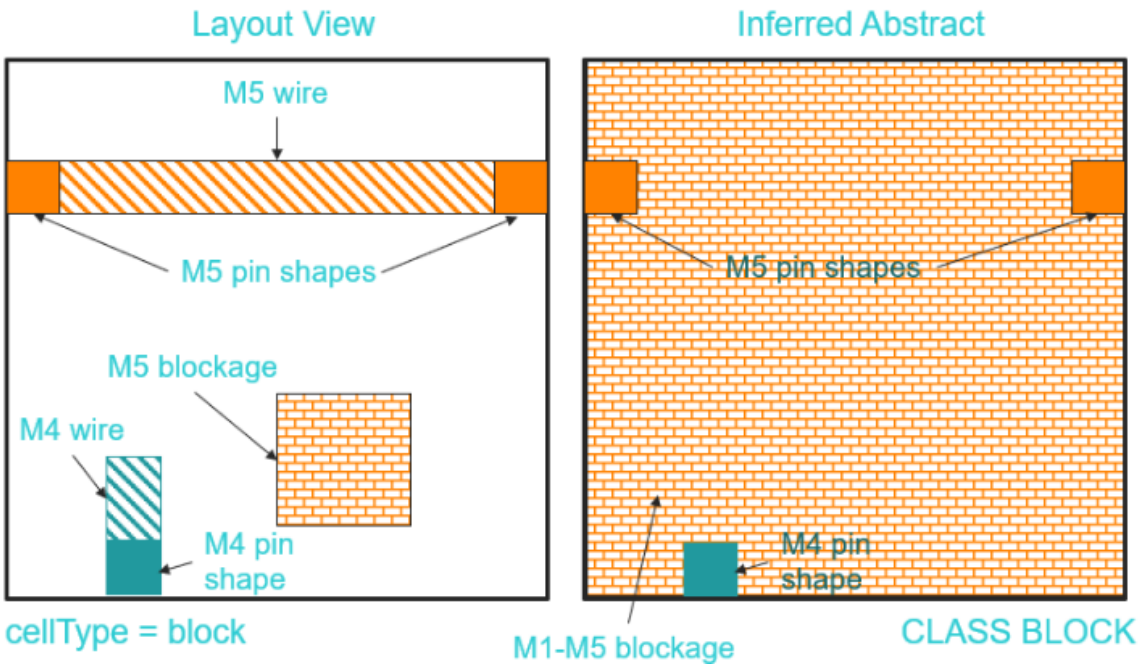


The modelling based on `cellType = core` can be used to model standard cells.

⚠

1. Base layers, such as OD, POLY, and NWELL, will not appear in the inferred objects.
2. **QAI may not work correctly with the self-aligned double patterning (SADP) process in advanced nodes. For example, although trim layers work for parameterized cells (PCells), they are not taken care of currently for standard cells and blocks.**

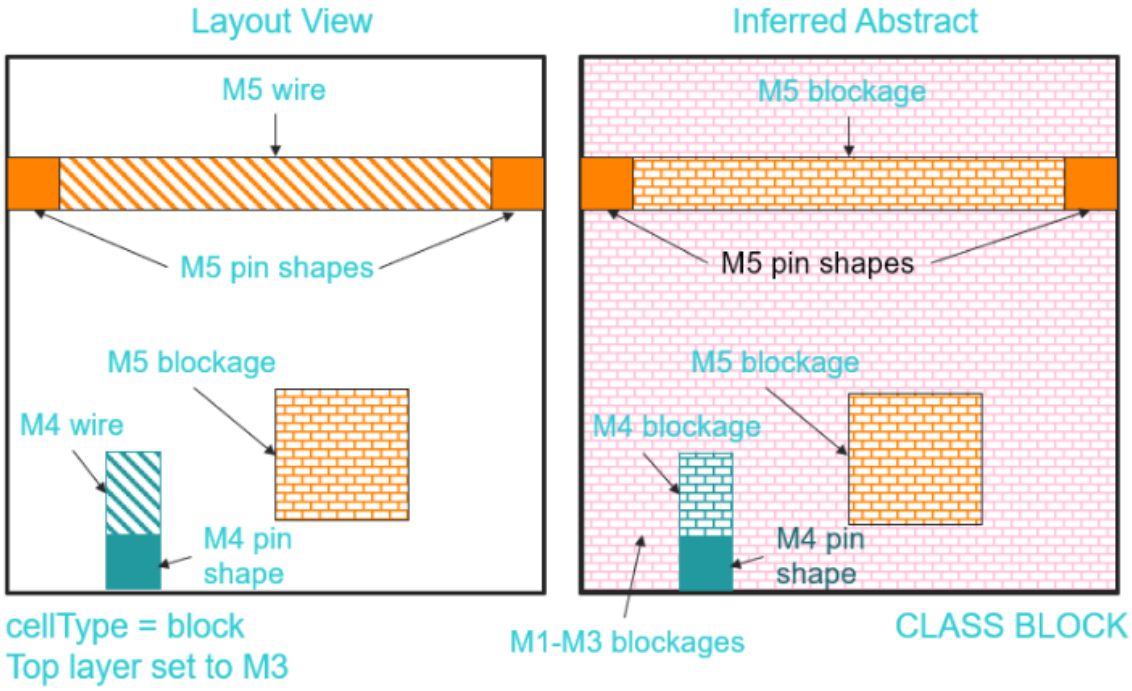
- For `BLOCK` or `NONE` cell types, Innovus applies full obstruction modeling to blockages/shapes during QAI, as shown below.



Blockages will be present up to the top layer (`M5` in the above diagram) in the inferred abstract.

7.3.3.2 Controlling QAI Through User-defined Properties

You can control how blockage modeling is done for `BLOCK` or `DEFAULT` cell type by specifying the top layer. When you do so, full obstruction modeling is limited to the specified top layer and detailed modeling is done on the layers above it. For example, consider the layout view below of a macro having cell type `BLOCK`. By default, the inferred abstract should have full obstruction up to `M5`. However, as the user-specified top layer is `M3`, the inferred abstract has full obstruction up to `M3` and detailed modeling above it.



The top layer can be specified and COVER obstruction be set on the cell using the following SKILL command:

```
cv = geGetEditCellView()
dbCreateCoverObstruction(cv~>prBoundary
techGetLayerMaskNumber(techGetTechFile(cv) "M3"))
```

Specifying the top layer explicitly is useful in cases where routing is sparse on the upper layers. Specifying the top layer implies that the wiring on the layers above it is not sensitive to cross-coupling to the digital wiring that may be routed nearby and therefore detailed modeling would be applied to these layers. Without the detailed modeling on these layers, the top-level router would have to detour around the block for all of the layers in the cellview.

Detailed modeling is not recommended for all layers of a block due to potential cross-coupling impact on internal nodes.

QAI Behavior in Specific Use Cases

Keep in mind the following notes on QAI behavior in the specific use cases listed below:

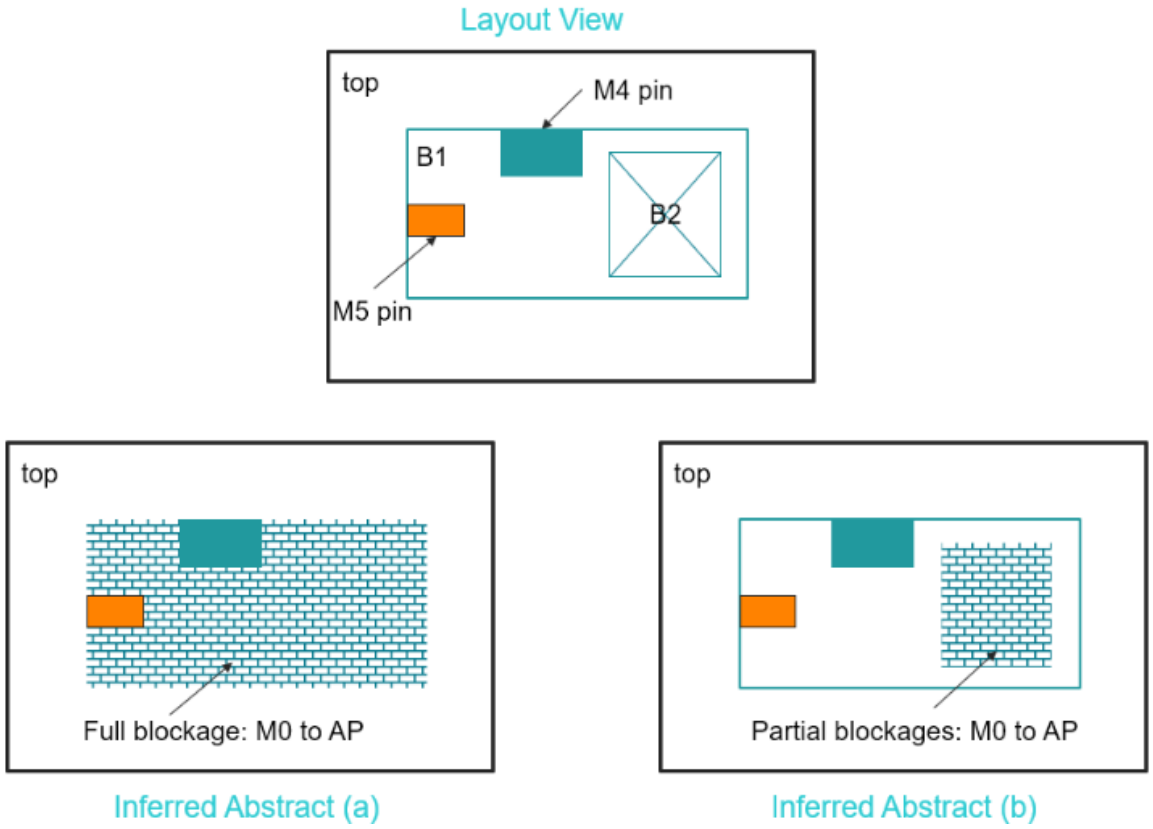
- If you specify a cover obstruction on a `cellType = cover` cellview, the tool will give the following warning message and ignore the cover obstruction:

  ```
  WARN: (IMPOAX-1597): Cover Obstruction is defined for cell 'cellName'. Cell is of type
  core/cover. Cover Obstruction will be ignored by the tool
  ```

- If you specify a cover obstruction above the used layers for a block, there will be obstruction from the first routing layer to the specified layer (no detailed obstruction modeling in this case).
- Any user-drawn layout blockages are ignored if they have the `+ PUSHDOWN` attribute as they apply to the

current level in the layout and not the level above. This is same as Innovus behavior and means that those blockages are not considered during QAI.

- Consider the case below of a block with a missing sub-block. In the layout view below, B1 is the block inside top for which QAI is needed. Block B1 has pins/shapes in M4 and M5. Inside B1, B2 is a sub-block with missing master. If you try to create a quick abstract for Block B1 and:
  - Block B1 is of cell type `BLOCK`, the tool will not find the master for B2 during QAI. The tool will assume B2 has all the base tech metal layers, although B1 only has layers up to M5. In the abstract, full blockages (M0 to AP) will be dumped, as shown in Inferred Abstract (a).
  - Block B1 is of cell type `COVER`, the tool does detailed modelling and assumes all the base tech metal layers in the instance area of the missing master, as shown in Inferred Abstract (b).



7.3.3.3 Controlling QAI Using Design Overrides

If you need to control the quick abstract generation but the block layout has been frozen, you may prefer to specify override information on the top design instead of editing the block or asking the block owner to update the block. Virtuoso/OpenAccess allows different overrides to be specified on different instances of the same cell. By storing the override information on the design cellview property, the same overrides can be applied to all instances of a given cell.

Use the `INVS_QAI_CELL_OVERRIDE` *`string`* property to specify all cell-based overrides. Apply this property on

the design cellview, so that all instances of the specified cell(s) will have the same setting. Its string value has the following format:

```
[CELL cellName [TYPE cellType] [TOP_LAYER topLayer] [SYMMETRY symmetry] [SITE siteName]] ...
```

Here:

- *cellName* is the name of the cell to which you need to apply the overrides.
- *cellType* specifies the type of the cell. To check valid values, see Cell type.
- *topLayer* is the top layer name used for dbCreateCoverObstruction.
- *symmetry* specifies orientation. See Symmetry.
- *siteName* specifies the name of the site. This applies only to core, pad, and endcap cell types.

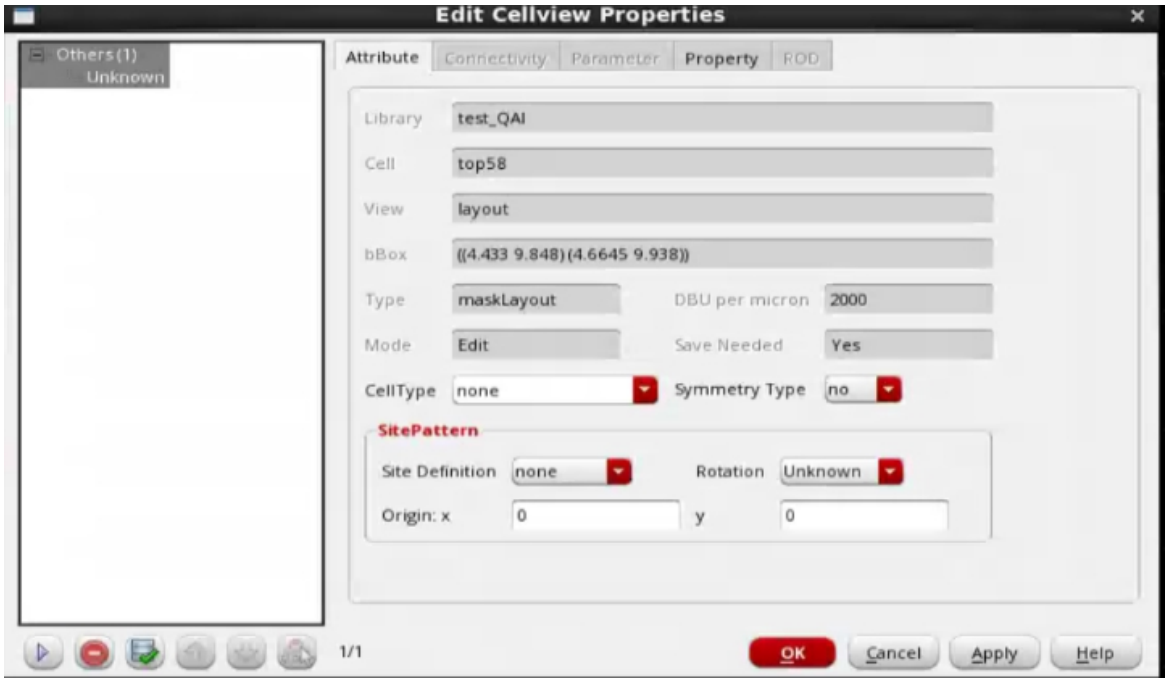
The string value can contain multiple cells, with each CELL keyword starting a new set. Each set can contain one or more of TYPE, TOP_LAYER, SYMMETRY, and SITE keyword-value pairs. For example, to set an override for two blocks, ring_osc and div_ckt, in a top design, you can write the override property as follows:

```
INVS_QAI_CELL_OVERRIDE CELL ring_osc TYPE block TOP_LAYER M5 CELL div_ckt TYPE block TOP_LAYER M3
```

How to Set the Override Property in a Design

Use the following steps to set the INVS_QAI_CELL_OVERRIDE property in a design:

1. Open the top-level design in *Virtuoso Layout Suite.*
2. Open the Edit CellView Properties form by pressing the Shift+Q keys.

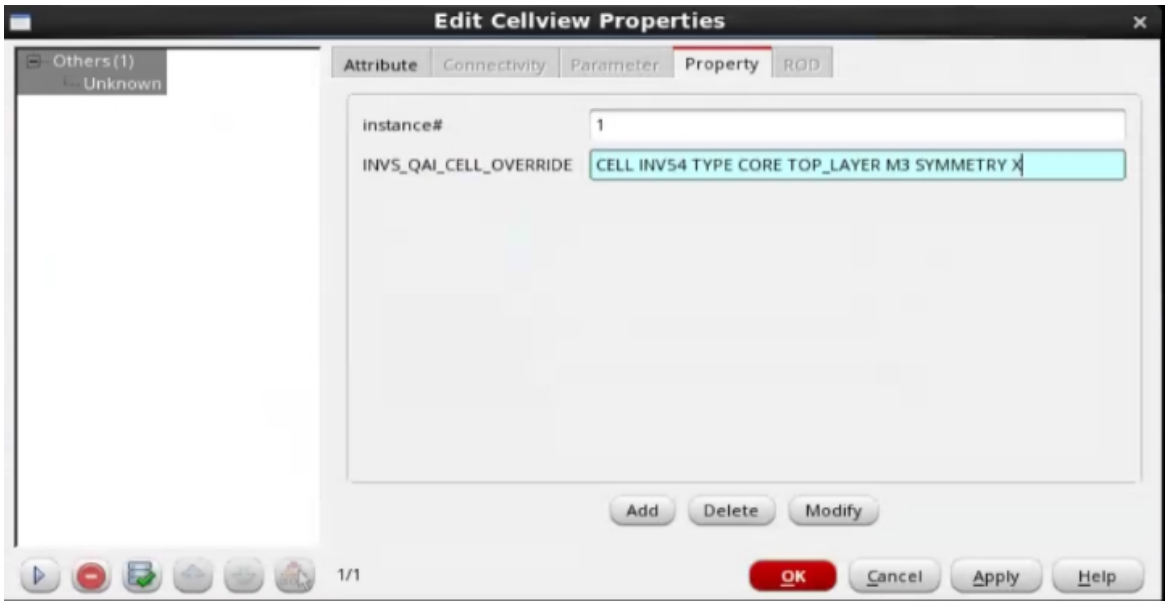


3. Open the *Property* page and then click the *Add* button.

4. In the Add Property form, specify `INVS_QAI_CELL_OVERRIDE` as the property name and click *OK*.



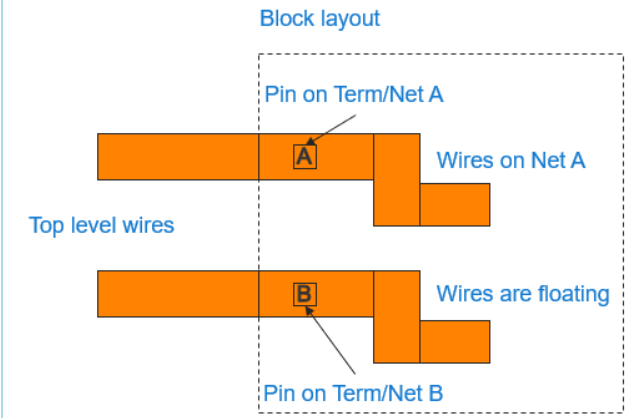
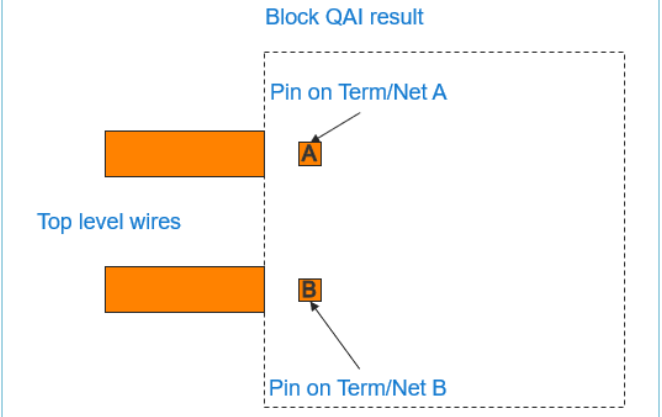
5. In the `INVS_QAI_CELL_OVERRIDE` property text box, specify the properties to override.
   In this example, the value is: `CELL INV54 TYPE CORE TOP_LAYER M3 SYMMETRY X`.



6. Click OK to apply the property setting and close the Edit Cellview Properties form.

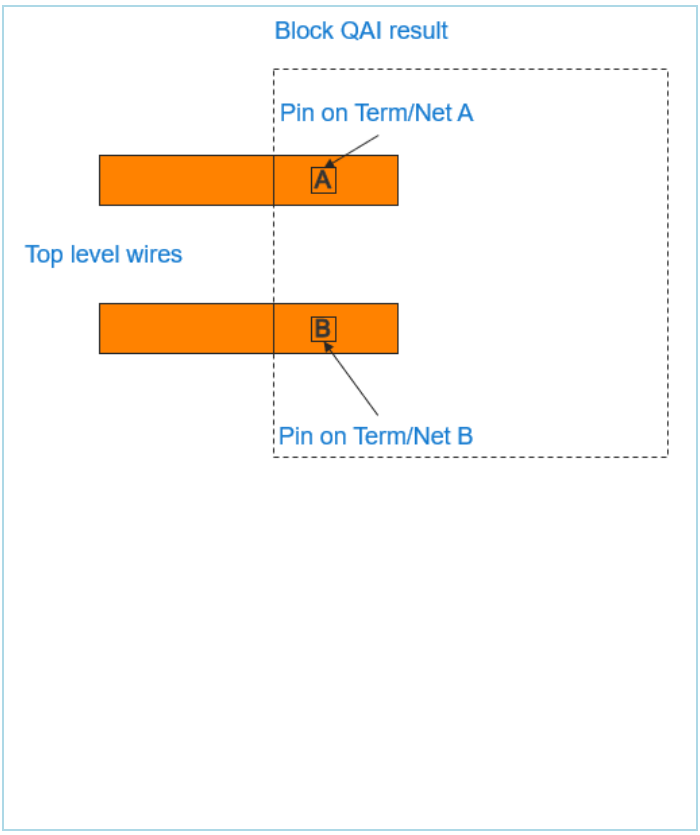
7.3.4 QAI Pin Modeling

The pin modeling methodology has improved in the latest Innovus release. Previously, QAI just copied the terminal~>pin shapes to the Innovus abstract representation, resulting in opens when the pins in the layout were derived from label/text locations. For example, consider an input layout cellview in which the top-level wiring abuts to shapes in the block. Here, the upper pin is VXL compliant, and the lower pin could be from GDSII with the *Create Pins from Text* feature in Virtuoso or Virtuoso Layout Editor. As QAI would just copy the terminal~>pins, resulting in opens on connections to A and B in the Innovus view of the block.

Input Layout Cellview	**Innovus view of top level and block, as processed by QAI, in previous releases**
	

From the 21.10 release, QAI attempts to find additional wire shapes that enclose the pin shape and are on the same layer as the term~>pin shape. For instance, in this example, QAI assumes that the pin shapes are enclosed by the shape that touches the boundary , resulting in the following Innovus view of the top level and block with no opens:

Innovus view of top level and block, as processed by QAI from this release

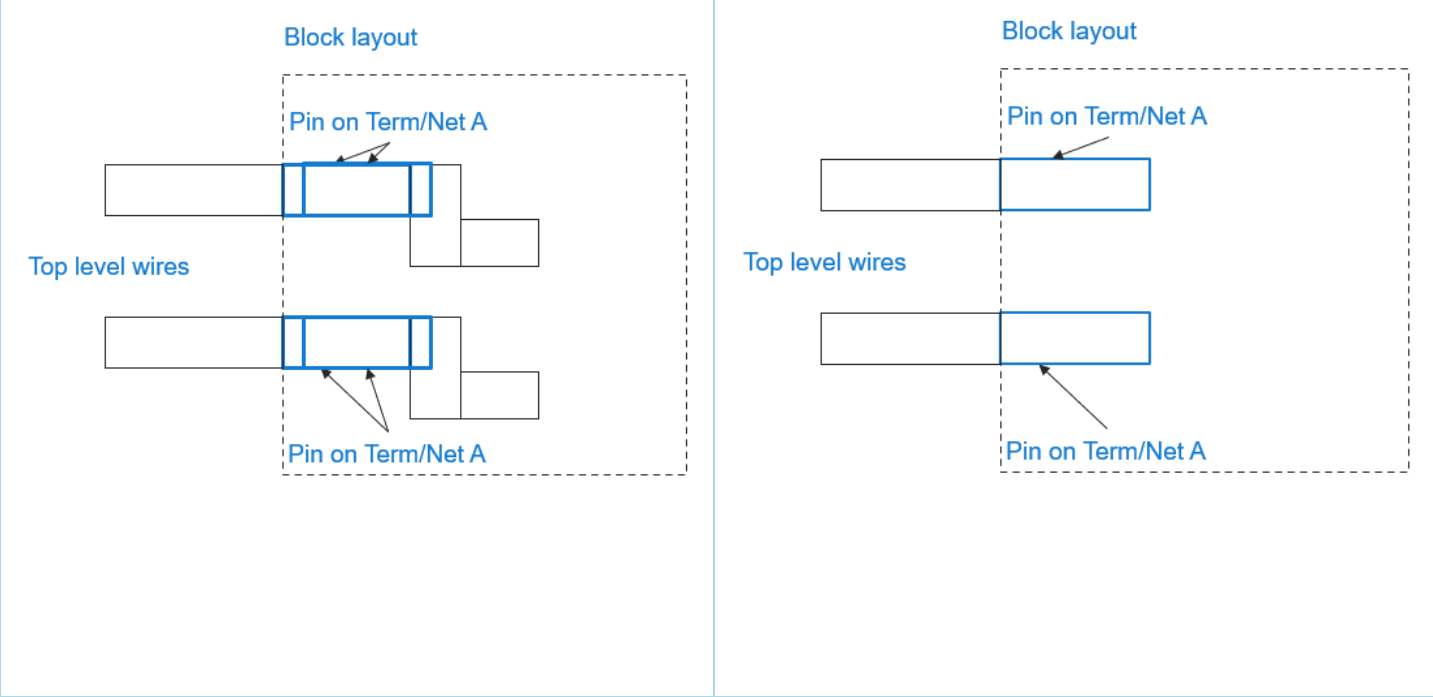


Note that the enhanced QAI pin modeling considers only level 0 shapes. There is no "shape tracing"; only the shapes that are on the same layer and that enclose the pin are added to the Innovus pin shapes.

This feature is on by default. If you do not want to include shapes that enclose the layout pin shapes as part of the pins during pin modeling, set the `setOaxMode -encloseQuickAbstractPins` parameter to `false`.

If there are completely overlapping pin port shapes on the same layer, QAI pin modeling merges the pin port shapes and creates one pin for NanoRoute. Consider the example below for processing terminals~>pins.

Input layout cellview	**Innovus view of top level and block, as processed by QAI from this release**



Merging of overlapping (including redundant) pin shapes occurs before the wiring enclosure step and is not impacted by the `setOaxMode -encloseQuickAbstractPins` parameter setting.

7.3.5 Antenna Annotation Utility for Creating Accurate Antenna Information for the Innovus Flow

As mentioned earlier in this section, when the Open Access flow is invoked in Innovus, if a macro block/cell does not have abstract views provided, Quick Abstract Inference (QAI) creates an abstract view on the fly, in addition to the already available layout view. This abstract has only the basic information for the block/cell, such as `prBoundary`, pin, and site information; it does not contain the process antenna information. The process antenna information is needed by routers, such as NanoRoute, to fix and repair the violations that are generated when using the process antenna information available on the pins. One way to process the antenna information accurately is to annotate the layout with the antenna. This can be done by specifying the layer, poly, gate, and oxide information through the antenna options file and then running the antenna annotation utility (`antenna_annotate.csh` script file) available in the directory below:

The antenna annotation utility (`antenna_annotate.csh` script file) is available in the directory below.

`<Innovus Installation directory>/lnx86/share/innovus/gift/MixedSignal/Utilities/skill`

You can execute the antenna annotation utility with the desired layout and the options file. This utility runs the Abstract Generator, which extracts the antenna information from the antenna options file and annotates it to the layout. When automatic abstract inference is done, the antenna information is included in the abstract created through abstract inference.

In addition to the `antenna_annotate.csh` script, the skill directory above has the readme

file, `annotate_antenna.readme.txt`, which is necessary for creation of the options file. The readme file also contains the command usage of the script.

A sample test case named `Antenna_annotation.tar.gz` is also available for the users to test the utility. Please see the readme file inside the test case directory for more information.

Note: Before invoking the script, make sure that the Abstract Generator path is set and the necessary license is available.

8

Netlist-Driven Mixed Signal Design Flow

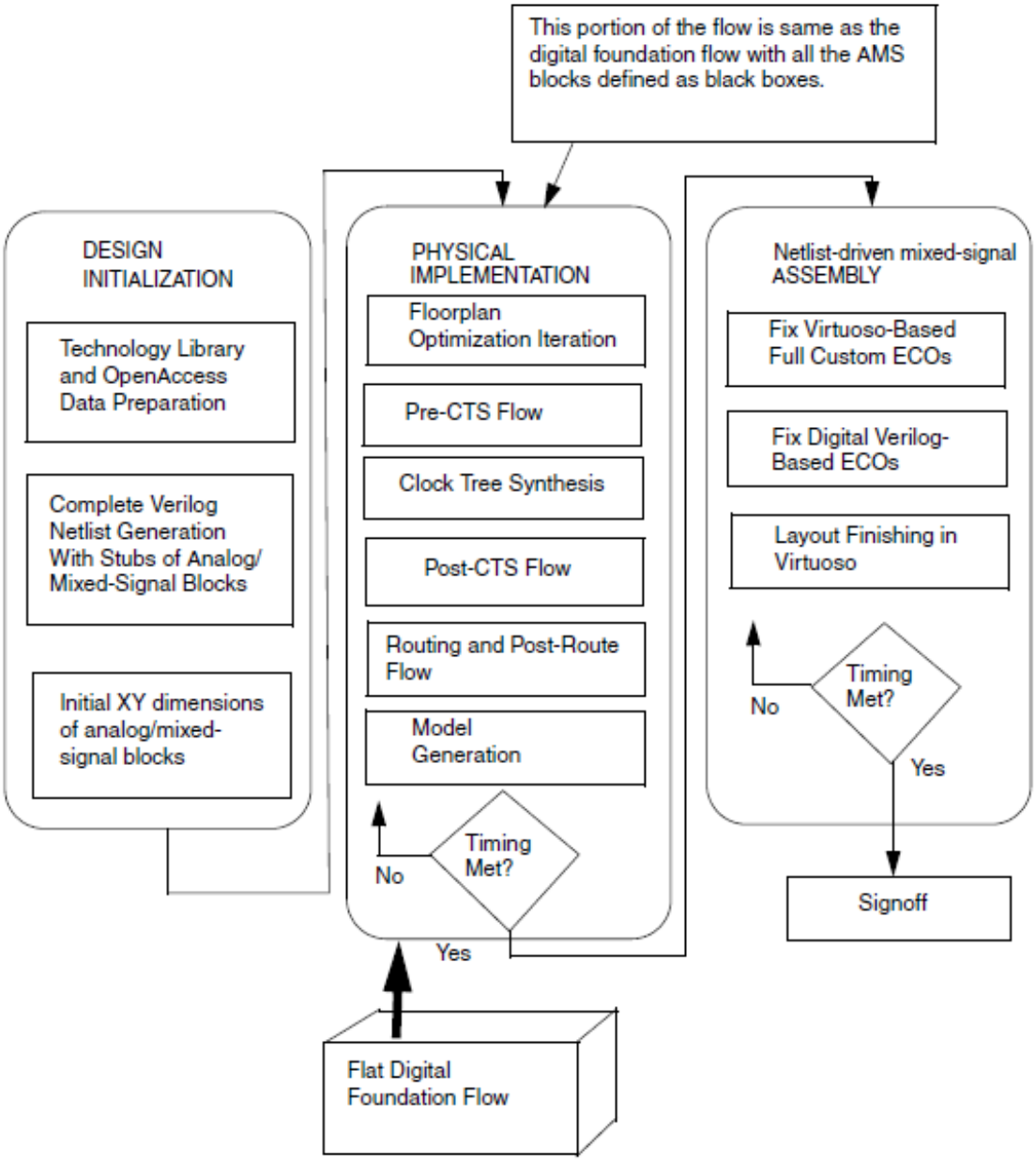
- Overview
- Technology and IP Library Preparation
- Verilog Netlist Creation
- General comments
- Floorplanning
    - Floorplanning of Verilog Netlist Using Blackboxes
    - Generate From Source for Soft Analog Block Layout Using Virtuoso
    - Load Physical View to Merge Optimized Pin Locations and Block Boundary
    - Physical Implementation of Soft Analog Blocks Using Virtuoso
    - Physical Implementation of Soft Digital Blocks Using Innovus
- Top-level Analog Net and Power Routing
- Top-level Design Implementation
- Final Chip Integration and Sign-Off
- ECO Flows

8.1 Overview

Most of the designs today are mixed signal in nature. A typical SoC may look like a pool of analog mixed signal IPs with a lot of embedded mixed signal logic into it. To enable mixed signal floorplanning, advanced interactive and automatic editing and pre-routing, chip assembly capabilities within the Innovus layout environment, Cadence has provided concurrent floorplanning flow using Virtuoso and Innovus. This allows you to plan, implement and connect blocks within a physical layout environment, eliminating the need for time-consuming and error-prone data abstractions and conversions.

A typical netlist-driven mixed signal flow may look similar to the digital implementation flow with

minor differences during floorplanning, top level specialty net routing, and ECO flows.



8.2 Technology and IP Library Preparation

- During this stage, you are expected to merge the technology LEF data into its base PDK using the details provided in the Technology Data Preparation chapter. The chapter also describes the dos and don'ts that you need to observe to achieve smooth interoperability of design data between Virtuoso and Innovus platform.
- Design library creation requires special care so that all the technology tree is visible to both the tools with proper reference and/or attach mechanism of design library preparation.
- All the cells used in the design are provided to Innovus in the form of reference OpenAccess libraries instead of the standard LEF files.
- Some environment settings need to be used before the start of Innovus and/or Virtuoso session. These are described in the Useful Tips chapter.

8.3 Verilog Netlist Creation

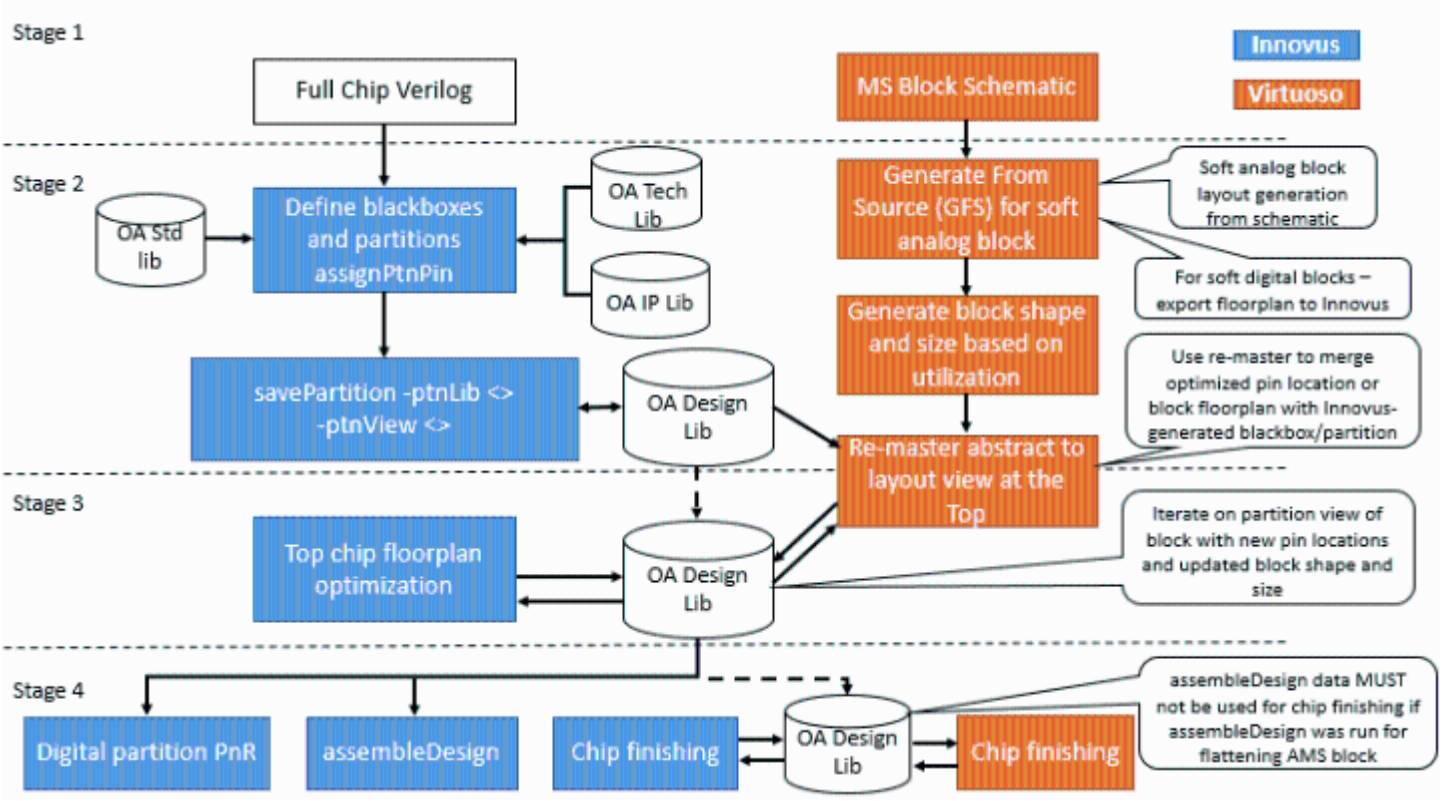
The Verilog netlist for the netlist-driven mixed signal flow is generally created through Synthesis. In this flow, the analog block is treated as a hard block during synthesis.

8.4 General comments

- Just as for any other design in Innovus, you need to prepare the timing and power intent definition files for more accurate and elaborate analysis within Innovus. Defining power intent through CPF is the recommended methodology.
- Quick abstract inference can be use for fast changing blocks containing digital and analog. This eliminates the need to generate abstract views again and again. Innovus requires an abstract view for all blocks in the design. As custom blocks may only have a layout view during the implementation phase, this capability is useful for creating the required abstract automatically during the floorplanning phase of the design.
- Static timing sign-off is fast becoming a requirement for mixed signal designs, which have timing paths between the digital and analog circuitry. Cadence offers an approach using a full timing model for mixed signal blocks, so exhaustive characterization of such blocks is not necessary.

8.5 Floorplanning

Netlist-driven mixed signal floorplanning flow can be divided into four stages depending on the maturity of the netlist.



Floorplanning starts very early in the design cycle, sometimes even before a formal Verilog netlist is available. During stage one, a very early chip area estimate is done with determination of a rough floorplan, power resource allocation and IP locations. General routing resource allocation can also be obtained.

In stage two, chip designers can start populating the details of some of the blackboxes for which little or no information was available. During this stage, lot of iterations happen between top-level chip designer and the block implementation team where they negotiate between their implementation challenges.

In stage three, designers move more towards obtaining a much more mature chip. Analog mixed signal blocks might also become available during this stage. Some of the challenges of not having correct abstract are also identified and closed during this stage. Block or IP designers start delivering their final block databases.

In stage four, chip designers start their final sign-off checks. Keeping track of the ECO changes in the block and the top is a big challenge during this stage. Both the block and the top level

implementation teams try to keep their blocks closed and clear all the sign-off checks.

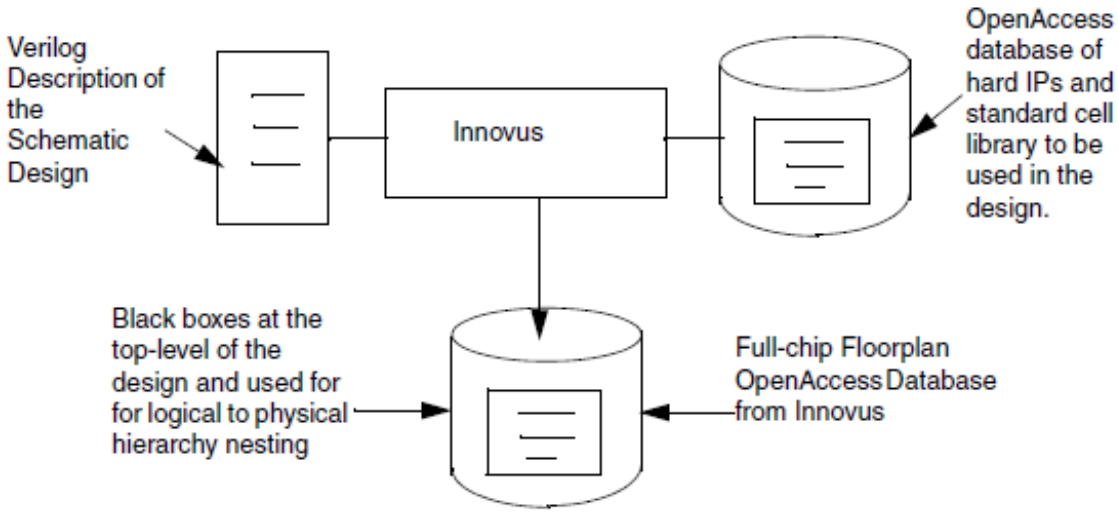
Innovus and Virtuoso provide a rich set of commands and GUI functions to floorplan your design interactively. There are also commands for creating an initial floorplan automatically, or, resizing a finished floorplan while keeping relative placement of objects.

- For information on floorplan commands, see the Floorplan Commands chapter, in the *Innovus Text Command Reference*.
- For information on floorplan GUI, see the Floorplan Menu chapter, in the *Innovus Menu Reference*.

8.5.1 Floorplanning of Verilog Netlist Using Blackboxes

In Innovus, you can use the `specifyBlackBox` and `definePartition` commands to create early shapes for the hierarchical blocks where physical hierarchy has to be maintained. Follow with the `partition` and `savePartition` command sequence to save hierarchical black-boxes. You can run congestion and timing aware block and standard cell placement in Innovus.

You can use `earlyGlobalRoute` results for block pin optimization. Manual editing might be required for pin optimization to address grid, side-constraints, and alignment. Power planning, power routing and early power analysis is possible using `sroute` and Voltus in Innovus. The full-chip floorplan OpenAccess database is saved from Innovus. Use the `savePartition -ptnLib <> -ptnView <>` command to save hierarchical nested black boxes and partitions in the OpenAccess database.



8.5.1.1 Steps to Perform the Floorplanning of Verilog Netlist

1. Set `setOaxMode -updateMode` to `true` to ensure complete interoperability. Create a configuration file within Innovus and use the top-chip Verilog netlist with OpenAccess reference libraries. Use the following variables while floorplanning a Verilog netlist: The following settings are for a design named `dtmf`. The Verilog netlist name is dtmf_chip.v, and the top level cell name is `dtmf_chip`.

   - `set lefDefOutVersion {5.8}`
   - `set init_abstract_view {abstract layout}`
   - `set init_layout_view {layout}`
   - `set init_pwr_net {vdd}`
   - `set init_gnd_net {vss}`
   - `set init_oa_ref_lib "techlib reflib"`
   - `set init_verilog "dtmf_chip.v"`
   - `set init_design_netlisttype {Verilog}`
   - `set init_top_cell {dtmf_chip}`

   Notes

   - No LEF files are required. The library is read by Innovus using the OpenAccess technology and the standard cell library.
   - The `init_abstract_view` {*`abstract layout`*} variable uses the layout view to infer abstract information directly from the layout view using on-the-fly quick abstract inference utility within Innovus. This capability is used for fast-changing layout views of macros and this avoids the need of creating abstracts again.

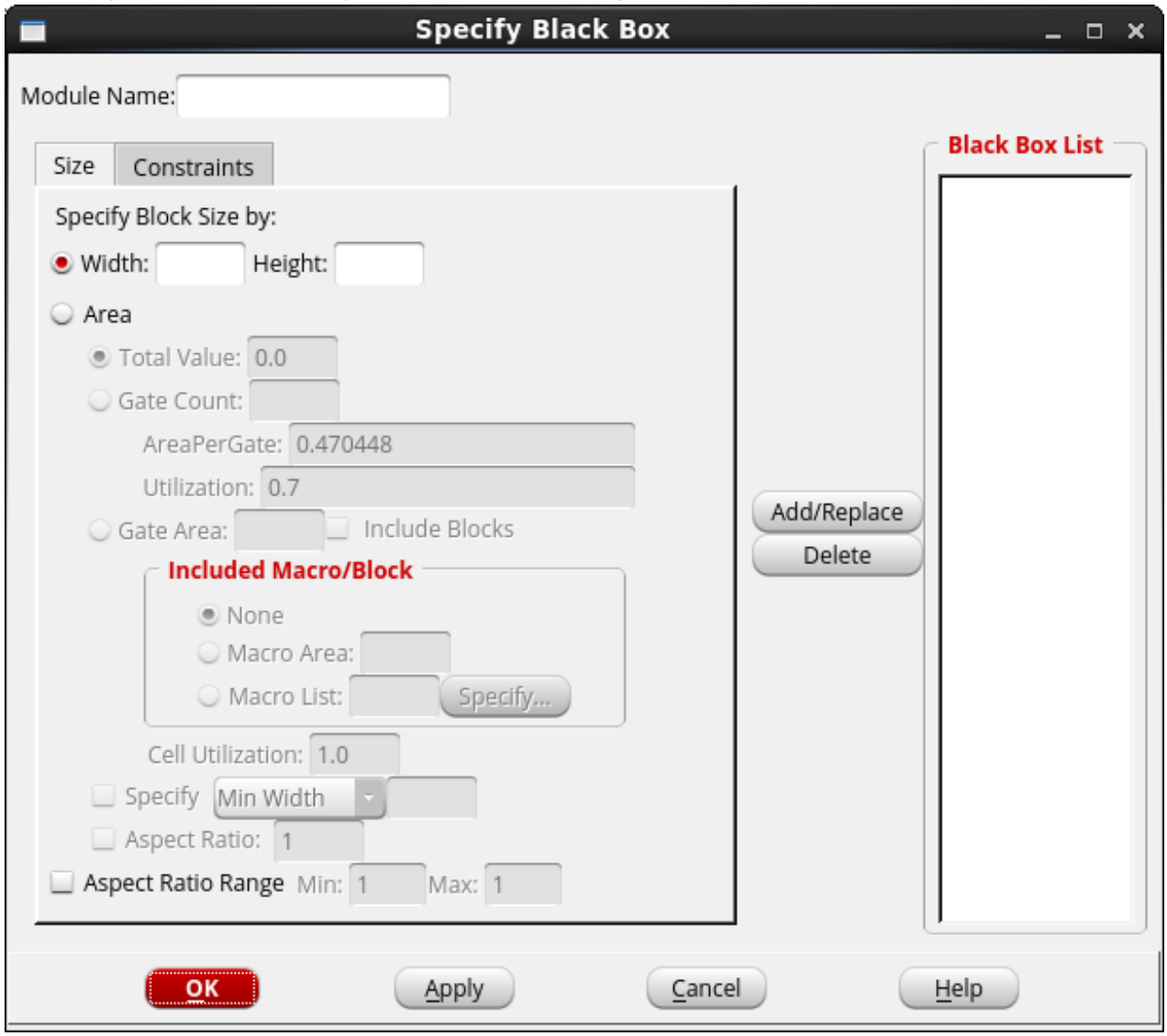
     From where is the tech information read?

     If an OpenAccess design is read, the tech graph is analyzed from the design_lib that contains the cellview.

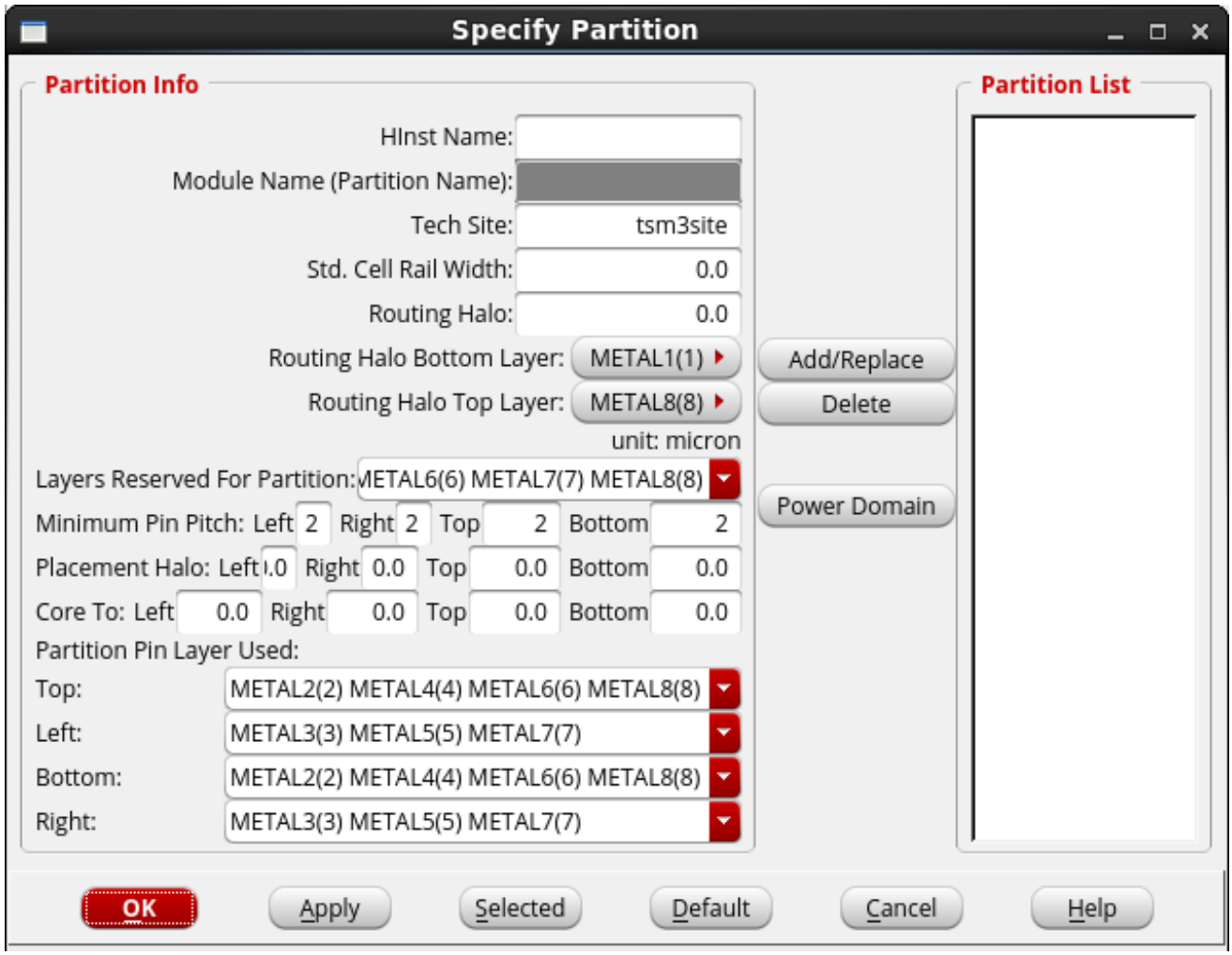
     If a Verilog netlist is read with OpenAccess reference libraries, then the first library in the OpenAccess ref lib list is used for the tech. This means that the order of the ref libs is important when a Verilog netlist is used.
   - Additionally, sites and vias can be read from other libraries even though they are mostly considered to be tech information.

2. After the design is imported in Innovus, use the *Partition-Specify Black Box* menu command to

define a rectangular shape within the logical hierarchy.
Use Hierarchy Up/Down browser to reach to the logical hierarchy where you want to specify a black box. After the dark green box is obtained, you can make the box rectilinear according to your requirements and adjust it within the floorplan.



3. Use the *Partition-Specify Partition* or *Partition-Specify Black Box* menu command to define a rectangular shape within the logical hierarchy which is required to have a separate physical hierarchy. This could be a soft analog block as well as a digital soft IP. You can use Hierarchy Up/Down browser to reach to the logical hierarchy which you want to specify as a partition. After the orange box is obtained, you can make the box rectilinear according to the requirement and fit it into the floorplan.



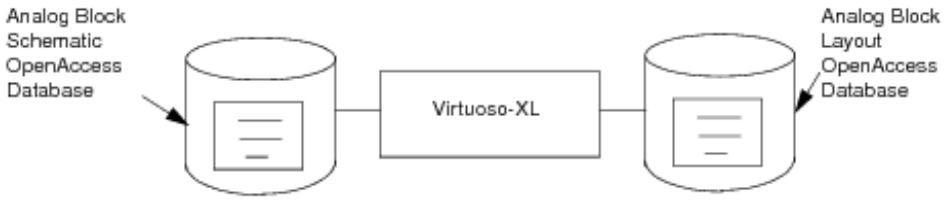
4. Use the standard cell and block placer in Innovus to run timing-driven detailed placement of hard blocks and standard cells. You can use the `place_design` command with its default setting (you can change the default by using the `setPlaceMode` cmd) to obtain automatic timing-driven and congestion-driven placement.
   Innovus provides the `setPlaceMode` cmd to control the local cell density, local utilization, in-place optimization, timing targets, and so on, to control and guide the placement optimizer. Iterate this step to get optimal placement of partitions and blackboxes specified in the design.
5. After completing detailed placement, you can use `earlyGlobalRoute` to get a quick estimate of the routing required. Using `earlyGlobalRoute` also helps in black box and partition pin assignment and pin layer and placement optimization.
6. Use the `savePartition -ptnLib` *`lib_name`* `-ptnView` *`layout_view_name`* command in the Innovus command prompt to save the partitions in OpenAccess database. This will create a

new library if the library name specified is not an existing name in the `lib.def` list. Two new views are created for each partition and black boxes. The view name specified is used for creating the layout view that should be used for detailed physical implementation of the block. The other view that is created is *`layout_view_name`* `_abstract`, which is used as an abstract view of the block or partition or blackbox.
Whenever block shape, size or pin position or pin layer is changed, you should update this `_abstract` view because at the top level, the abstract view is referenced and used for detailed implementation. The `savePartition -ptnLib` *`lib_name`* `-ptnView` *`layout_view_name`* command will also create a new top level view by the name *`layout_view_name`* provided as input. This top level view should be used for top-level implementation of the chip.
Note: Innovus allows blackboxes with a non`-R0` orientation to exist in the OpenAccess database. While reading a cellview that has a black box with non-`R0` orientation within Innovus, Innovus creates a geometric equivalent as the `R0` image of the blackbox within its virtual memory. While saving the design in the on-disk persistent OpenAccess database, Innovus will revert the black box instance to its original orientation.

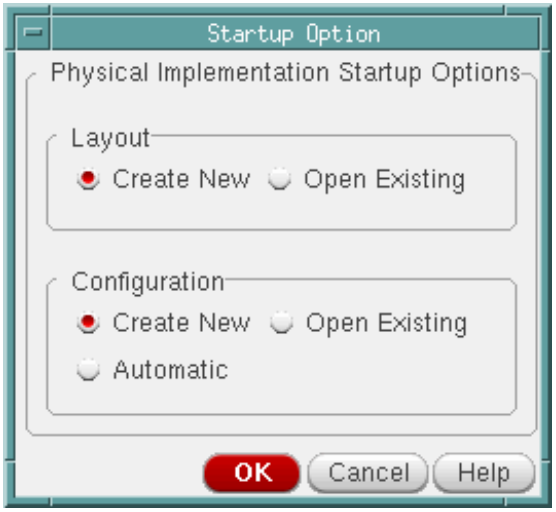
Do not change the black box orientation during the Innovus session.

8.5.2 Generate From Source for Soft Analog Block Layout Using Virtuoso

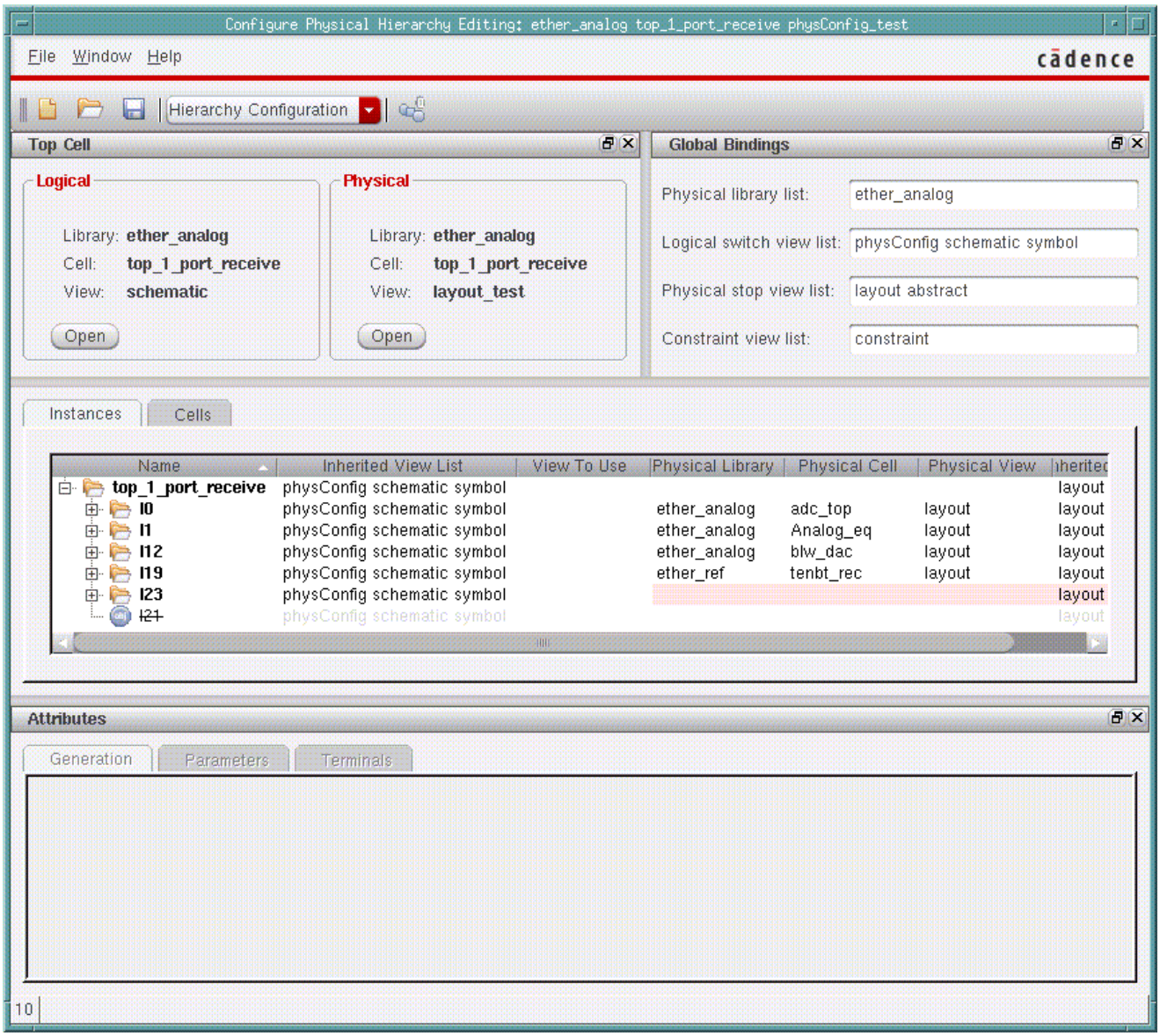
This part of the flow takes place in the Virtuoso environment. In this step, the analog/mixed-signal block in the design will start to get implemented based on the schematic that has been captured for this part of the block.



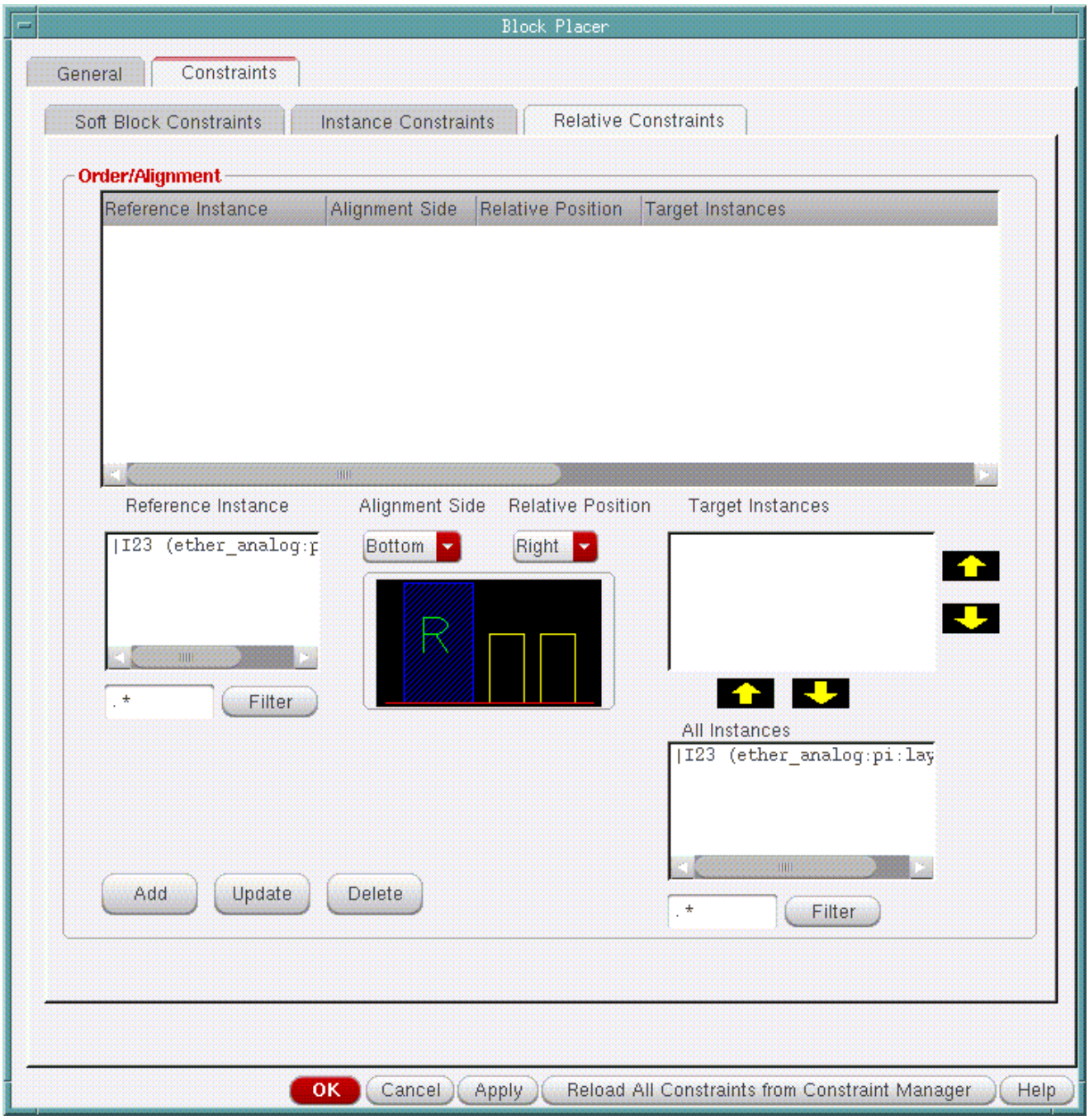
1. Run Virtuoso Layout-GXL/XL from the schematic window. The following GUI is displayed:



Configure Physical Hierarchy (CPH) is a shared interface between VLS-XL and Floorplanner available in Virtuoso. You can launch Virtuoso Layout-XL to generate layout view from the given schematic view. This layout view generated for the given schematic uses CPH information to process the generation of layout information according to the given constraints.



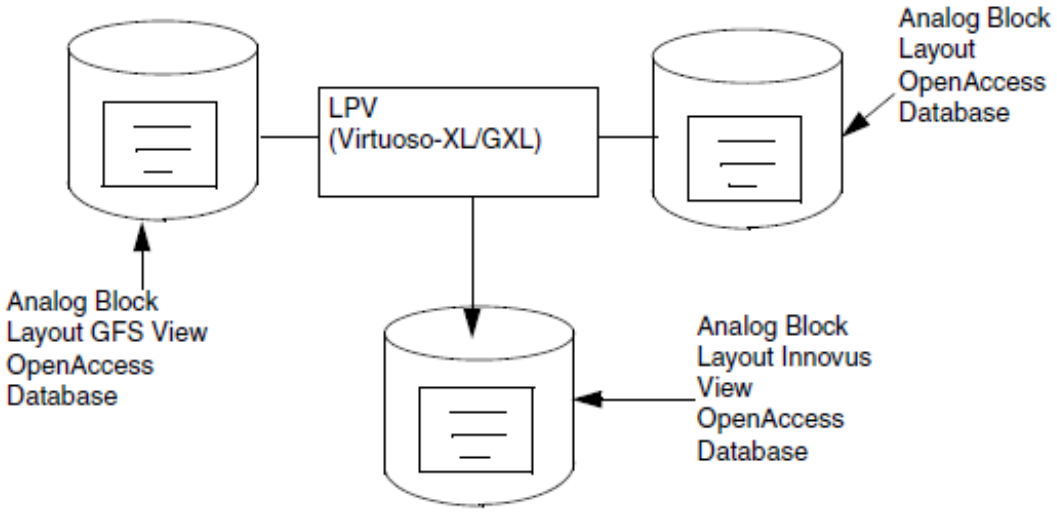
2. The soft analog block boundary is automatically generated and you can reshape it to match the design constraints. You can run Virtuoso Block Placer to place bigger blocks inside the analog design module. The feature of Block Placer and menu details are shown in the figure below.



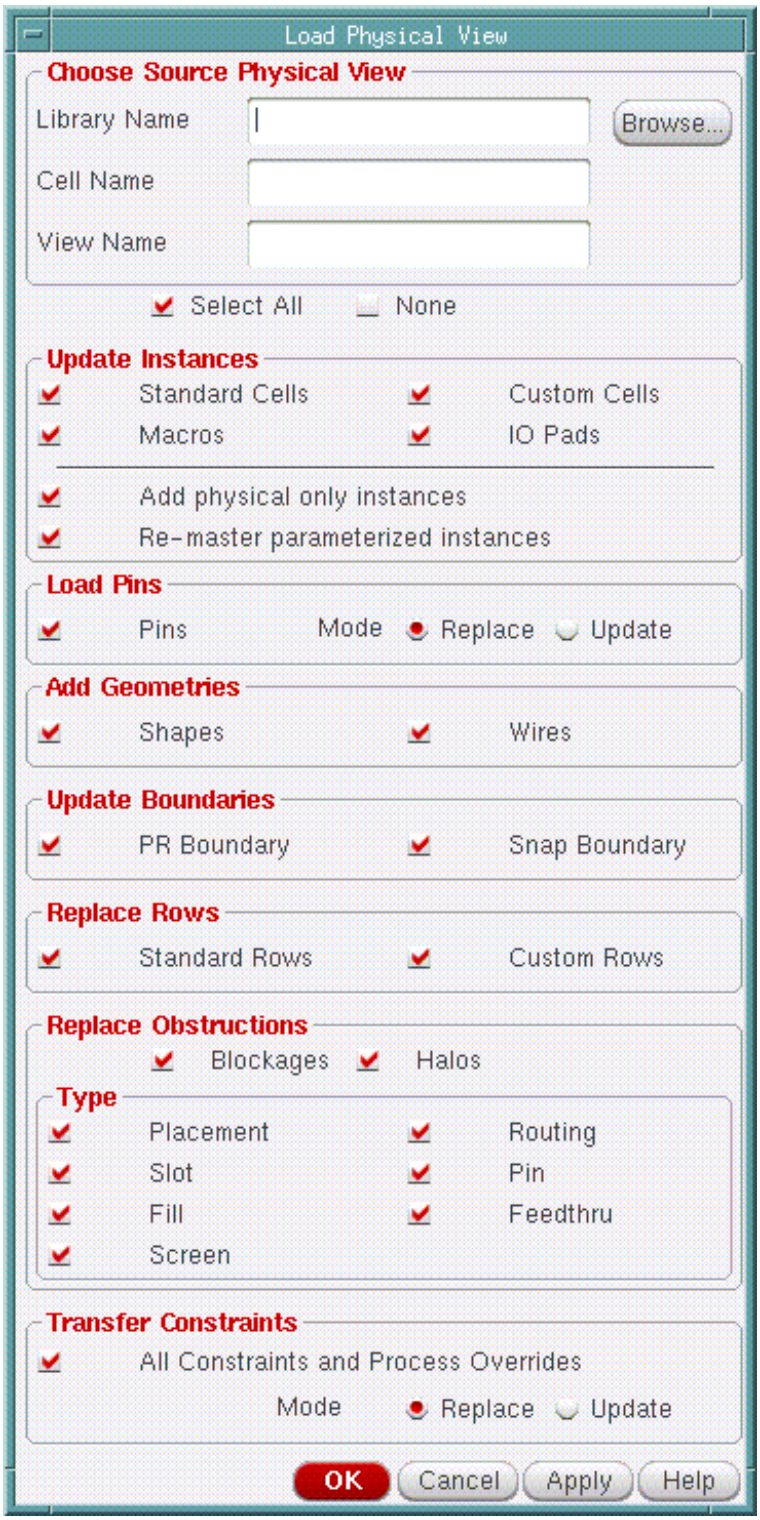
3. The layout generation of a given design is an iterative process. As soon as first cut layout view is ready, save the layout view in OpenAccess database.
   For more information, see *Virtuoso Layout-XL User Guide.*

8.5.3 Load Physical View to Merge Optimized Pin Locations and Block Boundary

This step of the flow also takes place in the Virtuoso environment. The Load Physical View command in Virtuoso is used to merge the optimized pin location done in Virtuoso with the floorplan created in Innovus for this block.



1. Open the layout view of the soft analog block generated during the floorplanning of Verilog netlist and then call Load Physical View (LPV) to merge the optimized pin locations, layer information, size and updated block boundary generated during Generate from Source. The LPV form is available in the Virtuoso Layout-XL/GXL window. In the *Choose Source Physical View* field, enter the view name generated during Generate From Source.



2. If Virtuoso driven Analog block is completed 50 percent or greater, then merge using the following steps:

a. Open analog block in Virtuoso.

b. Set the status of critical pins to fixed.

c. Do Load Physical View from the Innovus -generated soft block view.

d. Create abstract to update the abstract view.

If Virtuoso-driven analog block implementation is less than 50 percent done, then layout view can be merged using the following steps:

a. Open Innovus generated soft block abstract.

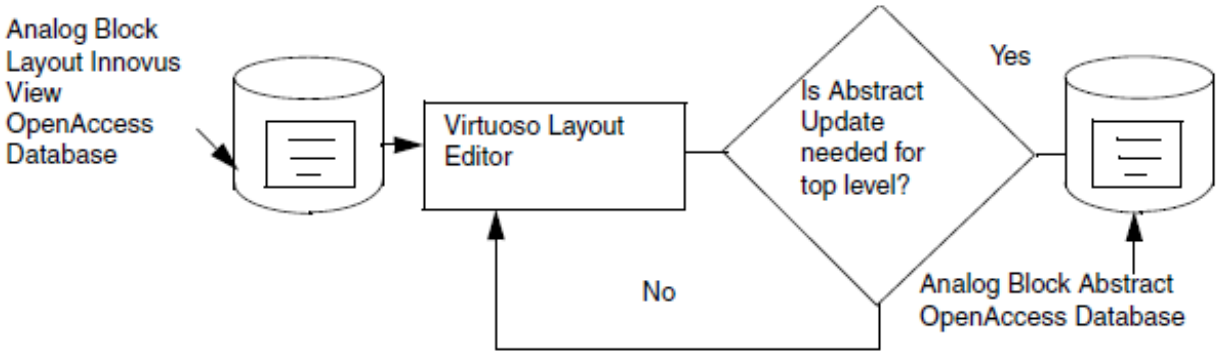
b. Set the status of critical pins to fixed (if needed).

c. Load the physical view from the Virtuoso analog block.

d. Create abstract to update the abstract view for top level usage.

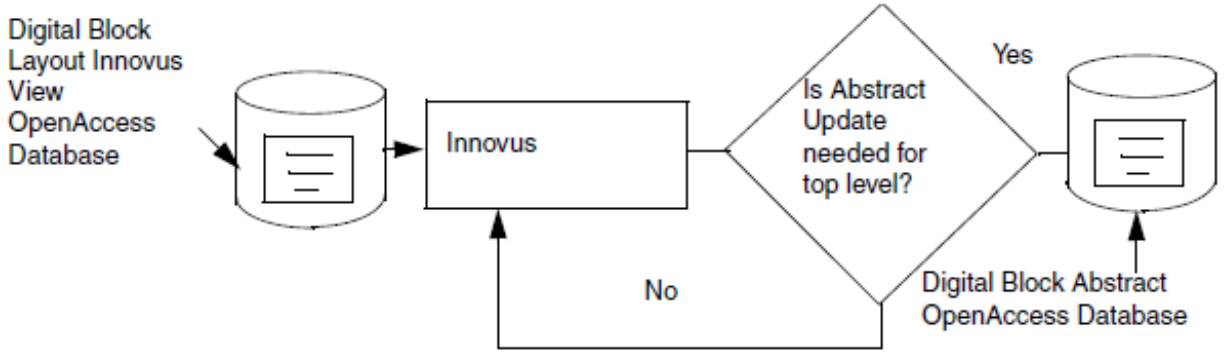
3. Update the abstract view of the soft analog block to avoid Innovus to read the full detailed layout view of the block.

8.5.4 Physical Implementation of Soft Analog Blocks Using Virtuoso



Since physical implementation of soft analog block is an iterative process, you can continue with further refinement of the analog block using Virtuoso Layout Editor. Keep the abstract view updated so that the top-level implementation place-and-route tool can get updated information about the analog blocks.

8.5.5 Physical Implementation of Soft Digital Blocks Using Innovus



Physical Implementation of soft digital IPs/blocks can continue in parallel with detailed standard cells and hard blocks within the soft digital block placement, pre-CTS timing closure, CTS, post-CTS timing closure, clock net routing, detailed signal routing, SI and voltage analysis, and so on. This is an iterative process and might take long time. The final delivery after this step is the DRC and LVS clean layout view with all sign-off checks completed.

If the top-level STA flow requires block SPEFs, then run extraction and save the parasitic data back into OpenAccess. Keep the abstract view of the soft digital block updated and in-sync with the layout view to avoid any kind of layout mismatches at the top level-layout integration. The abstract view is required for the top-level integration because reading the detailed layout view and using that for place-and-route of the top level results in more run time. So, to avoid long run times, use the abstract view with boundary pins and metal blockages of the block.

8.6 Top-level Analog Net and Power Routing

At this point, you will use the top-level view saved from Innovus by using the `savePartition` command which will have the abstract view referenced for the partitions and blackboxes at the top level of the design. To do this, follow these guidelines:

- Perform analog net routing. Mark the nets as analog so that applications like Innovus can understand such instances/nets as analog and avoid optimization of such nets and instances. To route these nets, use the Virtuoso Space-Based Router (VSR).
- You can perform power net routing as required by the design in Innovus. The `sroute` power router can be used for detailed power routing and follow-pin connections to the top-level power grid.
- VSR, a specialty available in Virtuoso platform, can be used for top-level analog net routing for 45nm and below technologies. It can also be called from within Innovus using the `run_vsr` command.
- With mixed signal routing constraints (DiffPair, MatchLength, NetClass and Shielding) interoperability possible, users are recommended to specify the IP and top-level routing constraints using recommended methods in either Virtuoso or Innovus platform. These constraints will be used throughout the implementation flow. For more information, refer to the Routing Constraint Interoperability chapter.
- PVS-CV tool can be used to validate the compliance of these constraints by net geometries. The tool can validate if the routing has been done manually or by automatic routers.
- The top-level critical and analog net routing can be done and pushed down as blockages into the partitions and blocks using the Virtuoso platform. Power nets can also be pushed inside the blocks for the blocks to use it for closing the connection with internal power network of the block.
  Similarly, power planning and power routing can be done in Innovus before running the `partition` command and then the `-pushDownSpecialNetAsObs` option can be used to push power routing and any signal routing or metal blockage as blockages inside the partitions created by Innovus.
  Nanoroute, a digital router, should not be used for connecting the partially routed nets manually. If the partially routed sections of the net are left in the design and Nanoroute is run in either ECO or complete mode, multiple hanging routed segments are deleted by Nanoroute. To keep such partially manually routed sections of the signal routes, mark such nets as fixed.

- The `saveDesign -cellview` command should be used with `setOaxMode -updateMode true` at the start of the session in Innovus to save design data in OpenAccess database.
  The `setOaxMode` command should also be used at the start of the Innovus session. You cannot change the update mode settings in the middle of the session.
  While saving from Virtuoso, do not change the top cell name used in Verilog netlist. Innovus always refers to the logical top cell name of the module used in the Verilog netlist for reading and writing the physical database.

8.7 Top-level Design Implementation

This portion of the flow is same as the digital foundation flow with all the mixed signal and analog blocks defined as blackboxes.

8.8 Final Chip Integration and Sign-Off

In this step:

- The `assembleDesign` command in Innovus has been enhanced to run multi-level assembly within one single command. This automation can be used to assemble final routed database of the AMS blocks for performing more accurate timing and SI analysis.
- Transition time violations can be checked on long top level nets by STA engine and automatic buffers can be inserted on the violating nets within Innovus.
- Timing and SI ECOs can be performed by doing the changes in the final verilog netlist. ECOs done using the `ecoOaDesign` command would retain the physical database sanctity during late stages of the design flow as well.

For more details, see the "Restoring the AMS Block Layout" section in the Static Timing Analysis for Mixed Signal Designs chapter.

8.9 ECO Flows

The Cadence platform based mixed signal solution offers a concurrent mixed signal floorplanning solution, which allows for efficient, frequent and smooth exchange of data between the top level designer and those designing the lower level blocks. The solution is implemented on the OpenAccess database, and allows for the exchange of floorplanning, placement and routing constraints. In addition, the environment offers the capability to embed mixed signal IP blocks with constraints that are used when instantiating such IPs within a design.

To address the challenges of performing ECOs on mixed signal designs, Cadence has implemented powerful ECO functionalities that perform ECOs on the actual OpenAccess database, containing both custom and place-and-route objects. For example, a design containing pcells can be ECO'ed in the Innovus environment without the need to separate out the custom objects in the design.

For information on pre- and post-mask ECO flows in Innovus on OpenAccess, refer to the "Innovus-Based ECO Flow" section in the Chip Finishing and ECO Flows chapter.

9

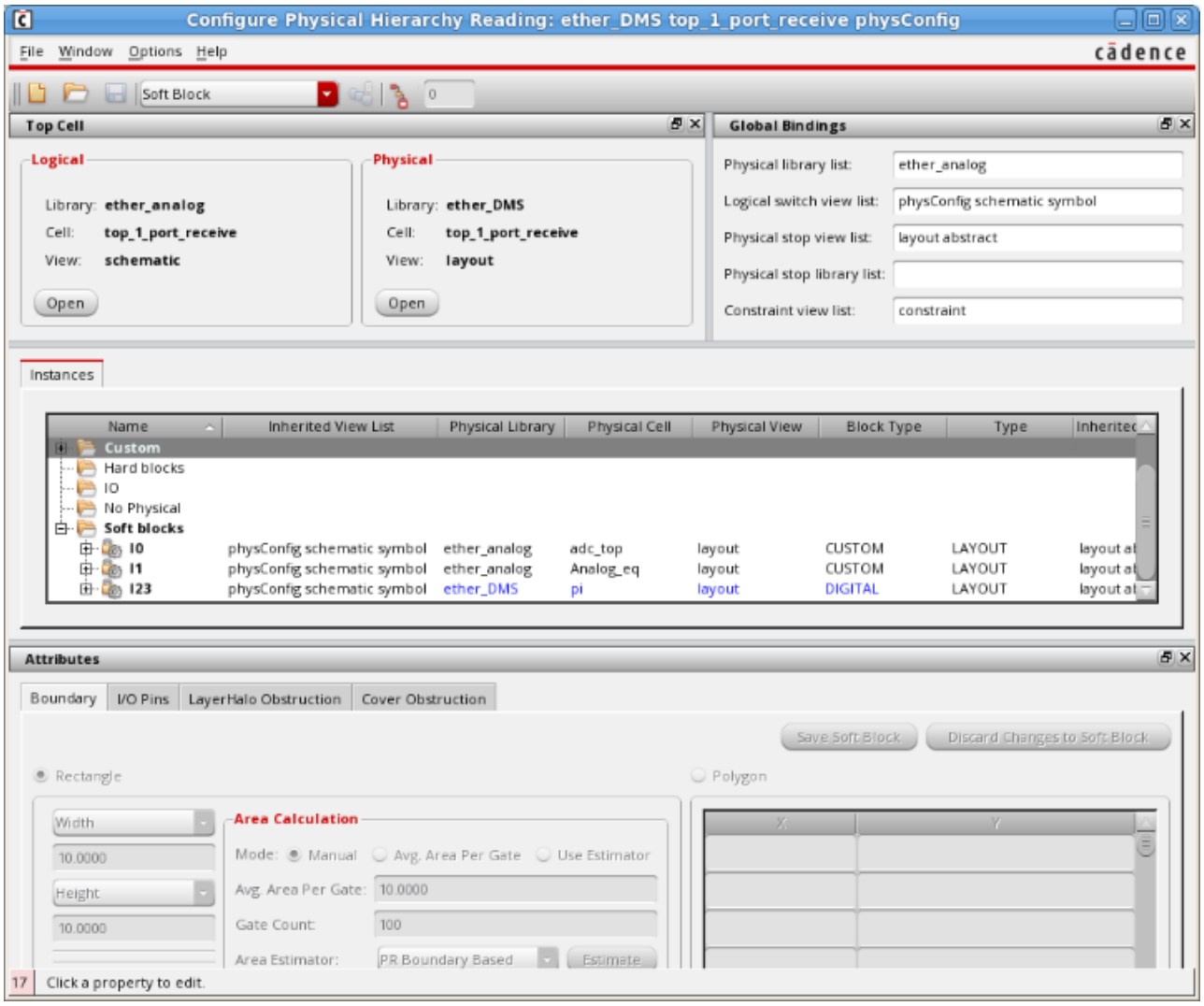
OpenAccess Wiring Terminology

- Creating an Interoperable Digital Block Floorplan in Virtuoso Layout Suite XL
- Understanding Symbolic and Geometric Routing
  - Differences Between Symbolic and Geometric Routing
  - Identifying Symbolic Routing and Geometric Routing
  - Identifying the Route Status for Symbolic and Geometric Routing
  - Ways of Creating Wires in Virtuoso and Innovus Environment
- Locking a Net Using Virtuoso Space-Based Router (VSR)
  - Locking a partially routed net
  - Locking a net that is completely unrouted
  - Locking a completely routed net
  - Unlocking Nets
  - Highlighting Locked Nets
- Editing Net Attributes in Virtuoso
- Wire/Net Mapping Between Virtuoso and Innovus
- Handling of Wires from Virtuoso in Innovus
  - Conversion of Wires to DEF SPECIALNETS in Innovus
  - Implications of the Conversion in Innovus
  - Addressing the Conversion Issue
  - How To Prevent the Conversion Issue
- Wiring Connectivity in Innovus
- Wiring Extraction in Innovus

9.1 Creating an Interoperable Digital Block Floorplan in Virtuoso Layout Suite XL

Virtuoso Layout Suite XL (VXL) provides a connectivity-aware environment. An interoperable digital block floorplan created in VXL can be later used in Innovus. To create an interoperable digital block floorplan in VXL:

1. Create a top-level floorplan in VXL.
    a. Use *Launch -> Configure Physical Hierarchy* (CPH) to create analog and digital softBlocks.
    b. Define the digital softBlocks as blockType `DIGITAL`.
    c. Create the top-level and softBlock layout views with *Floorplan -> Generate Physical Hierarchy* (GPH).



Refer to the Schematic-Driven Mixed Signal Design Flow chapter for more information on this step.

2. Optimize the top-level floorplan and reshape the softBlock boundaries (Level-1 editing) by using Virtuoso Floorplanner. See the Schematic-Driven Mixed Signal Design Flow chapter for more information on this step.
3. Optimize and align IO pins with Pin Optimizer.
    a. Use *Edit -> Advanced -> Align* to fully align these IO pins with reference objects.



b. Make sure that the Placement Status of the pins is *placed* or *fixed*.

4. Define the correct *Signal Type* for power/ground nets. In Innovus, the power and ground nets are SPECIAL NETS. If they are defined as `signal`, they will be treated as regular nets by NanoRoute.

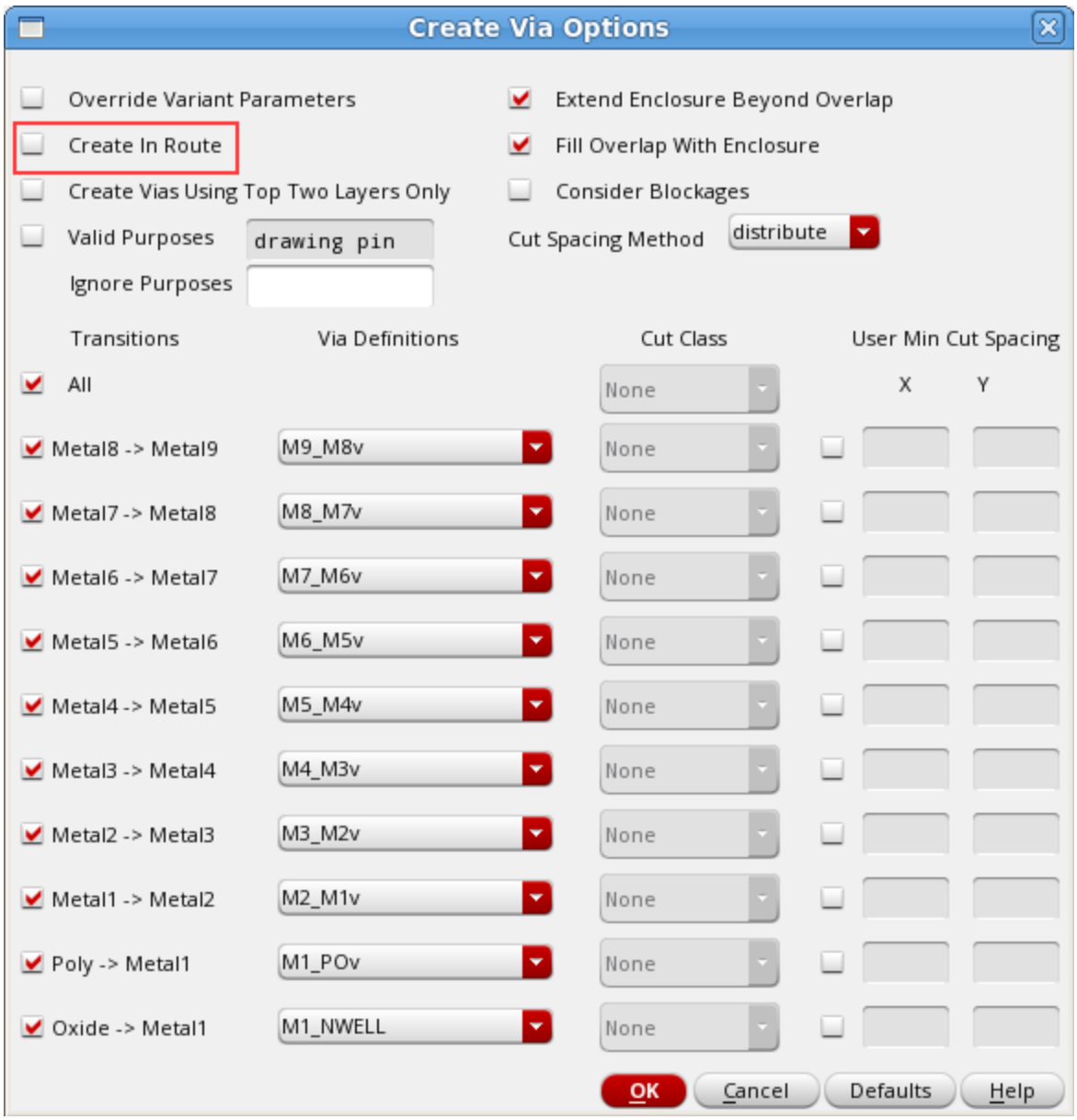
   a. Select the net in the VXL Navigator and use the Property Editor assistant to check or change the signal type. Ideally the signal type is correctly defined in schematic as this will be propagated automatically to layout.

5. Create top-level supply routing with geometric wires (geometric routes). Power/Ground pre-routing needs to be created as geometric wires ("Part of Route : FALSE", nets that would be in the DEF SPECIALNETS section).

   a. Use the *Create* -> *Shape* -> *Geometric Wire* after enabling GUI access for creating geometric wires by including the following in the `.cdsinit` file:

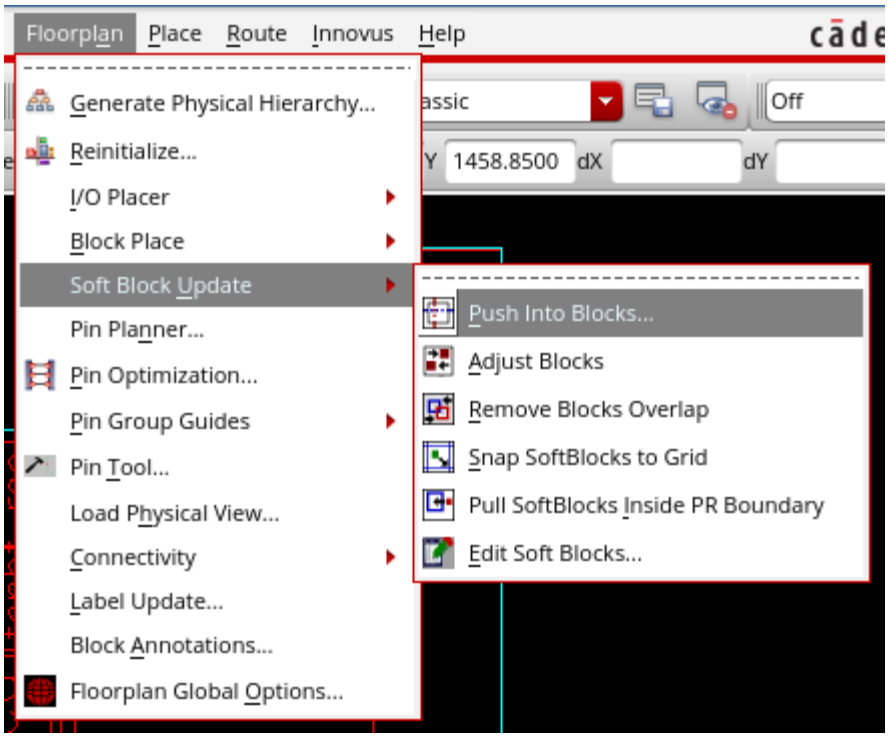
```
; Enable the Create -> Shape -> Geometric Wire functionality
envSetVal( "we" "mixedSignalWireEditingEnvironment" 'boolean t )
```

b. Make sure that the manually placed Vias (*Create -> Via)* are also geometric routes (Part of Route : FALSE). To achieve this, deselect the *Create In Route* on Create Via Options form.



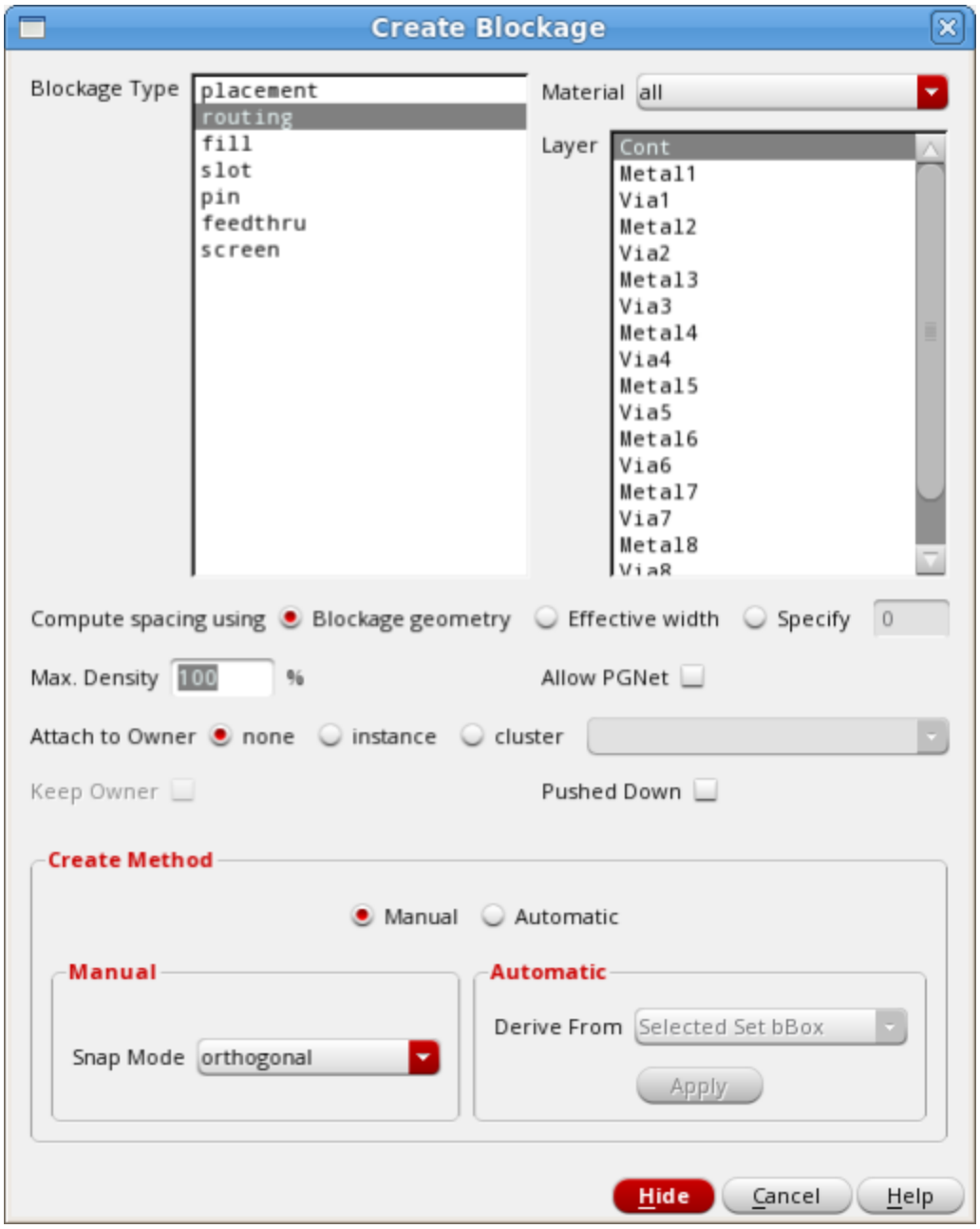
c. If you want to convert symbolic wires to geometric wires on legacy IP, use the *Edit -> Wiring -> Compose Trunks* command.
Normal signal pre-routing has to be done with symbolic wires either manually with *Create -> Wiring -> Wire* or with automatic routing (In Virtuoso Space-based Router (VSR), select *Route -> Automatic Routing*).

6. Push down power/ground routing. The top-level supply routing can be pushed down into the digital block-level floorplan.
    a. Select *Floorplan -> Soft Block Update -> Push Into Blocks.*



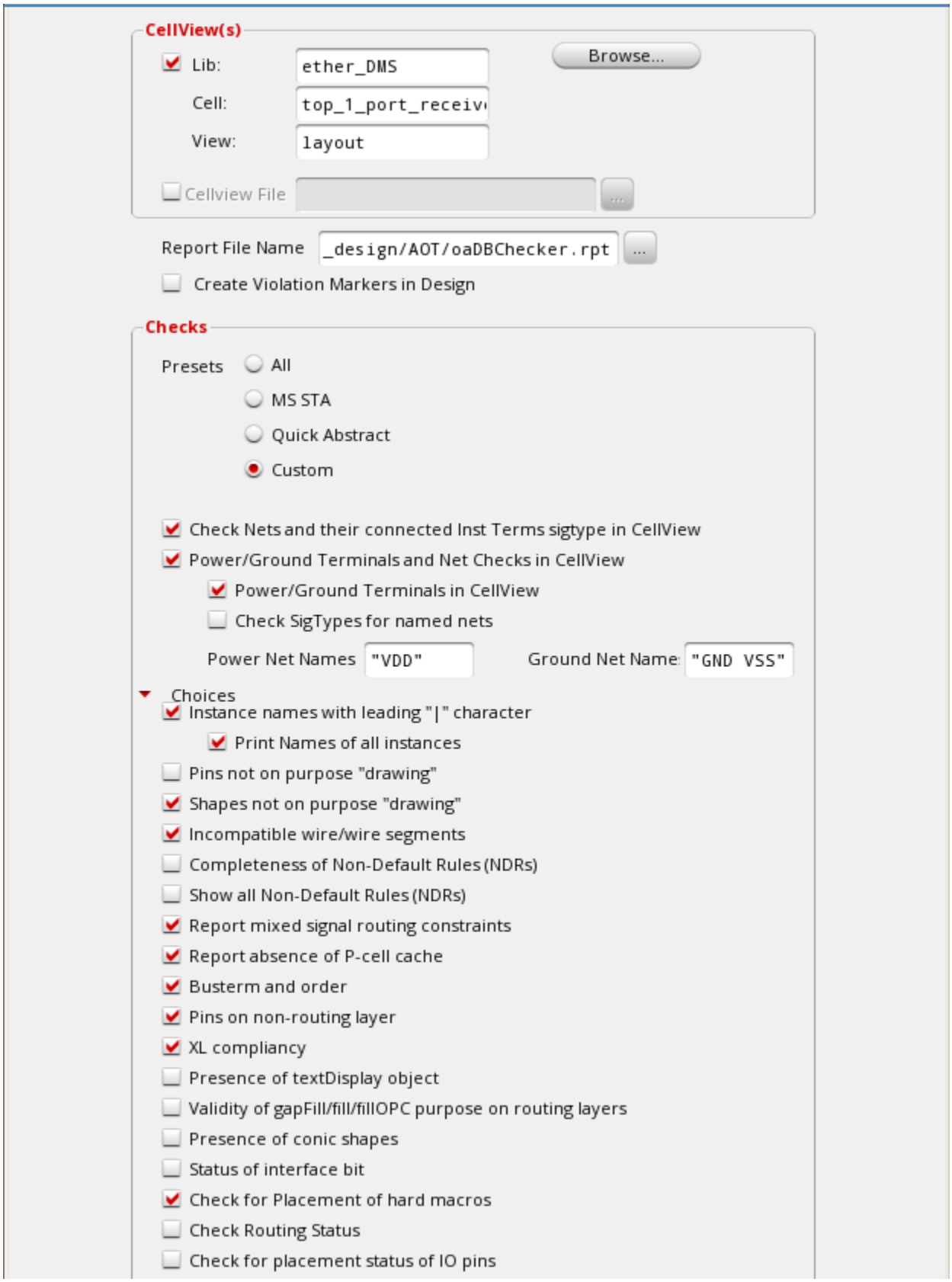
    Refer to the Schematic-Driven Mixed Signal Design Flow chapter for more information on this step.

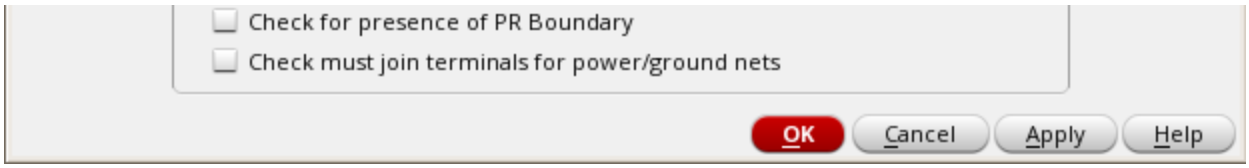
7. Create Placement and/or Routing (P&R) blockages in the digital block floorplan. The P&R blockages must be created as OpenAccess blockages:
    a. Select *Create -> P&R Objects -> Blockage*.



8. Define interoperable constraints. Create the required routing constraints (diffpair, shielding, non-default rules, bus constraints, and so on) with Virtuoso Constraint Manager. These constraints are saved in the OpenAccess view and are
interoperable with Innovus.
Check the Routing Constraint Interoperability chapter for more information on this step.
9. Run the OpenAccess Database Interoperability Checker (OA DB Checker). Check the OpenAccess Database Interoperability Checker chapter for more information on this step.
This is an example how to use custom settings to check for floorplan inconsistencies.




CellView(s)
Lib:
ether_DMS
Browse...
Cell:
top_1_port_receiv
View:
layout
Cellview File
Report File Name
_design/AOT/oaDBChecker.rpt
Create Violation Markers in Design
Checks
Presets
All
MS STA
Quick Abstract
Custom
Check Nets and their connected Inst Terms sigtype in CellView
Power/Ground Terminals and Net Checks in CellView
Power/Ground Terminals in CellView
Check SigTypes for named nets
Power Net Names
"VDD"
Ground Net Name
"GND VSS"
Choices
Instance names with leading "|" character
Print Names of all instances
Pins not on purpose "drawing"
Shapes not on purpose "drawing"
Incompatible wire/wire segments
Completeness of Non-Default Rules (NDRs)
Show all Non-Default Rules (NDRs)
Report mixed signal routing constraints
Report absence of P-cell cache
Busterm and order
Pins on non-routing layer
XL compliancy
Presence of textDisplay object
Validity of gapFill/fill/fillOPC purpose on routing layers
Presence of conic shapes
Status of interface bit
Check for Placement of hard macros
Check Routing Status
Check for placement status of IO pins




```
....
Performing Check for existence of PG terms in Netlist....
INFO (CHECKER_DESGN-27): Found terminal 'VSS' with ground sigType in the cellView.
INFO (CHECKER_DESGN-27): Found terminal 'VDD' with power sigType in the cellView.
PASSED.

Checking SigTypes of named power and ground nets....


Performing Check for existence of non-drawing shapes....PASSED.

Checking for pins on non-routing layers in the design....PASSED.


Checking the incompatible wires/wire segments....PASSED.


Checking for presence of pcell cache ......PASSED.

Checking for XL compliancy.....PASSED.

Checking placement status of hard macros...PASSED.


Final Summary
Type of checks PASSED FAILED
--------------------------------------------------------------------------------

Design Library Checks:
Pin shape check for terminals PASSED
'|' char in instance names FAILED
Bus Annotation check FAILED
Check Nets and their Inst terms' sig type FAILED
Power/Ground Checks PASSED
Check signal types of named nets PASSED
Shapes on drawing purpose PASSED
Pins on non-routing layer check PASSED
Unsupported routing shapes PASSED
MS constraints check PASSED
PCell cache check PASSED
XL Compliancy check PASSED
Placement status check PASSED
```

10. Import the floorplan, placement and routing data in Innovus.
    a. To enable mixed-signal interoperability flow, make the following settings at Innouvs startup:

    ```
    setOaxMode -updateMode auto

    setViaGenMode -symmetrical_via_only true
    ```

    For more information, see the Design Data Preparation chapter.
    b. Use `oaIn` with the `-filter` option to read only the object types that are chosen:
    `oaIn <lib cell view> -filter {boundary pin_shapes blockages special_routing}`

    For more information, see the Schematic-Driven Mixed Signal Design Flow chapter.

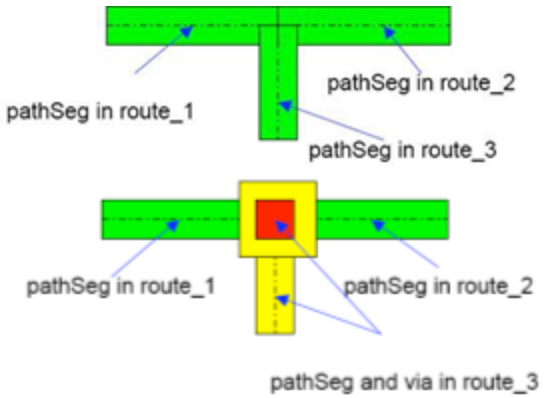
Now you should see all the floorplan objects created with Virtuoso XL in Innovus.

9.2 Understanding Symbolic and Geometric Routing

9.2.1 Differences Between Symbolic and Geometric Routing

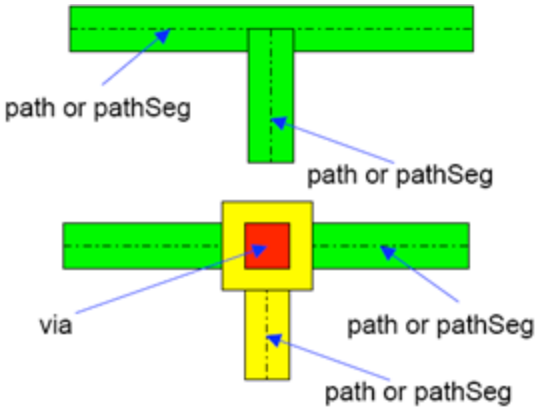
Symbolic routing:

- Represents the routing used by automatic routers.
- Corresponds to DEF NETS section which are signal (default) routing.
- Contains path segments with matching end-points.
- Contains via origin matching with end-points.
- is contained in oaRoutes.



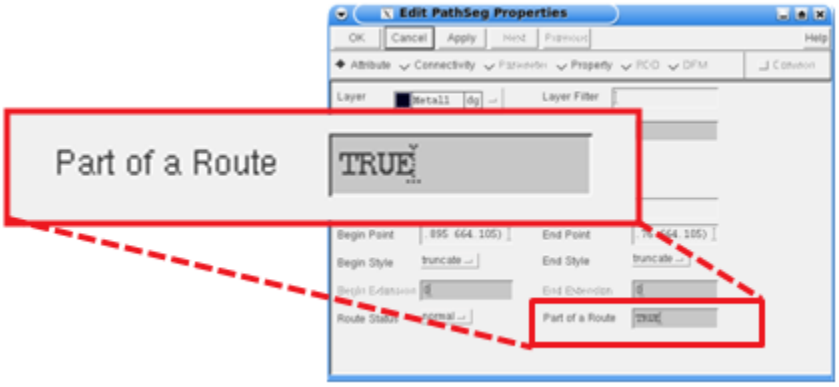
Geometric routing:

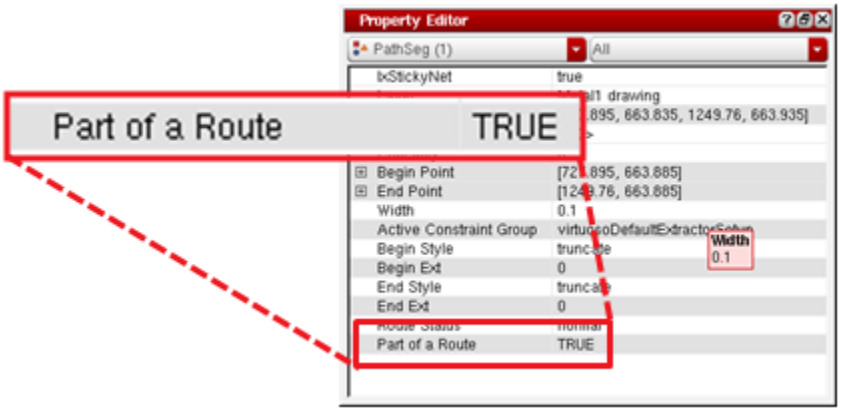
- is often used for power and ground and other hand-crafted nets.
- does not require paths, path segments and vias, and end points to match and be contained in oaRoutes.
- corresponds to DEF SPECIALNETS section which are special routing.



9.2.2 Identifying Symbolic Routing and Geometric Routing

- When you use the Property Editor GUI, the *Part of a route* field is true in case of symbolic routing and false in case of geometric routing.
- When you use Property Editor Assistant, invoked from the task assistant in Virtuoso, the *Part of a route* field is true in case of symbolic and false in case of geometric routing.

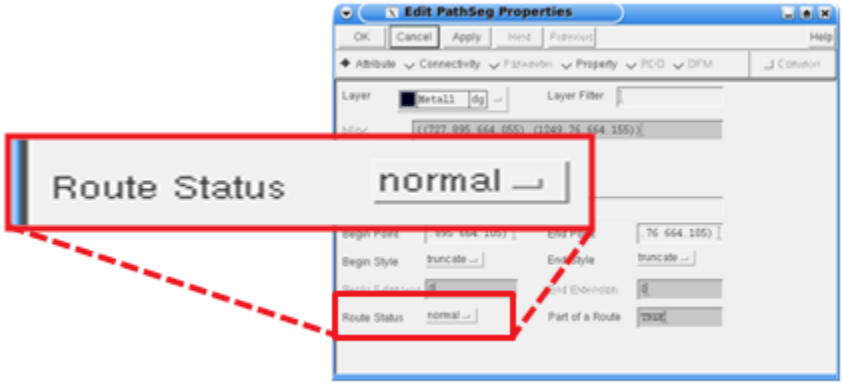


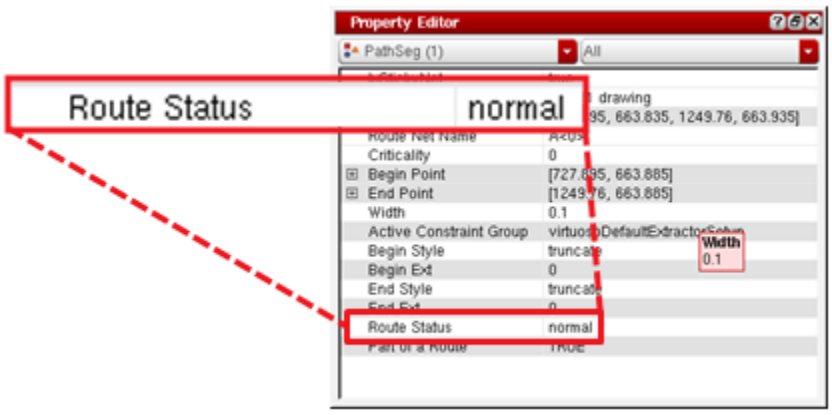


- When you use SKILL, it returns the ID number in case of symbolic routing and no value in case geometric routing.

9.2.3 Identifying the Route Status for Symbolic and Geometric Routing

- For Property Editor, the route status in case of symbolic and geometric routing is Normal, Fixed, and Lock.
- For Property Editor Assistant, the route status in case of symbolic and geometric routing is: Normal, Fixed, and Lock.





- In case of SKILL, the route status is returned as: nil, locked, fixed, and normal.

9.2.4 Ways of Creating Wires in Virtuoso and Innovus Environment

Virtuoso	Innovus

Geometric (SPECIALNETS)
• VLS-L ways of creating routing:
◦ *Create > Wiring + Compose Trunks* command (Recommended)
The *Compose Trunks* command converts the selected set of shapes into a trunk object, which can be used in Virtuoso with the Pin to Trunk router or the Power router.
◦ *Create > Shape > Path*
◦ *Create > Shape > Geometric Wire*
▪ Paths
▪ Segments
• RiDE/Chip Assembly Router
◦ Power Planning
	Geometric (SPECIALNETS)
	• Sroute and Wire Editor
	◦ PathSegs not belonging to oaRoutes
	• Power Planning
	• Flip Chip router
	Note: To create SPECIALNETS for Innovus (nets that would be in the DEF SPECIALNETS section), geometric routes need to be created in Virtuoso. To enable GUI access for creating geometric routes, include the following in the .cdsinit file.
	`; Enable the Create->Geometric Wire functionality`
	`envSetVal( "we" "mixedSignalWireEditingEnvironment" 'boolean t )`

Symbolic (NETS)
• VLS-L ways of creating routing:
  ◦ *Create > Wiring > Bus*
• VLS-XL/ GXL/CE ways of creating routing:
  ◦ Above 3
  ◦ *Create > Wiring > Wire*
  ◦ *Create > Wiring > Bus*
  ◦ *Create > Wiring > Point to Point*
  ◦ *Create > Wiring > Guided Routing*
  ◦ *Route > Automatic Routing*
  ◦ *Route > Routing Scripts*
• RiDE/Chip Assembly Router (VSR)
  ◦ Net/bus routing
	Symbolic (NETS)
	• Nanoroute, Wroute, and Wire Editor
	  ▪ Path segments belonging to oaRoutes

Note: In Innovus, the usage is for flip-chip routing and only geometric routing is allowed.

9.3 Locking a Net Using Virtuoso Space-Based Router (VSR)

9.3.1 Locking a partially routed net

To lock a partially routed net, ensure that:

- The existing wiring objects (pathSegs, vias, and paths) are locked.
- Rerouting the design can complete the net, but the pre-existing wiring objects do not change.
- The new objects that complete the route are not locked. You must lock them explicitly.

9.3.2 Locking a net that is completely unrouted

To lock a completely unrouted net, ensure that:

- There are no wiring objects to lock.
- Rerouting the design results in routing the net.
- The wiring objects are not locked. You must lock them explicitly.

9.3.3 Locking a completely routed net

To lock a completely routed net, ensure that:

- Every wiring object on the net is locked.
- Rerouting the design does not alter any wiring object on the net.

9.3.4 Unlocking Nets

To unlock the nets:

- Delete the constraints in Virtuoso Constraints Manager.
- Select the nets in the Navigator.

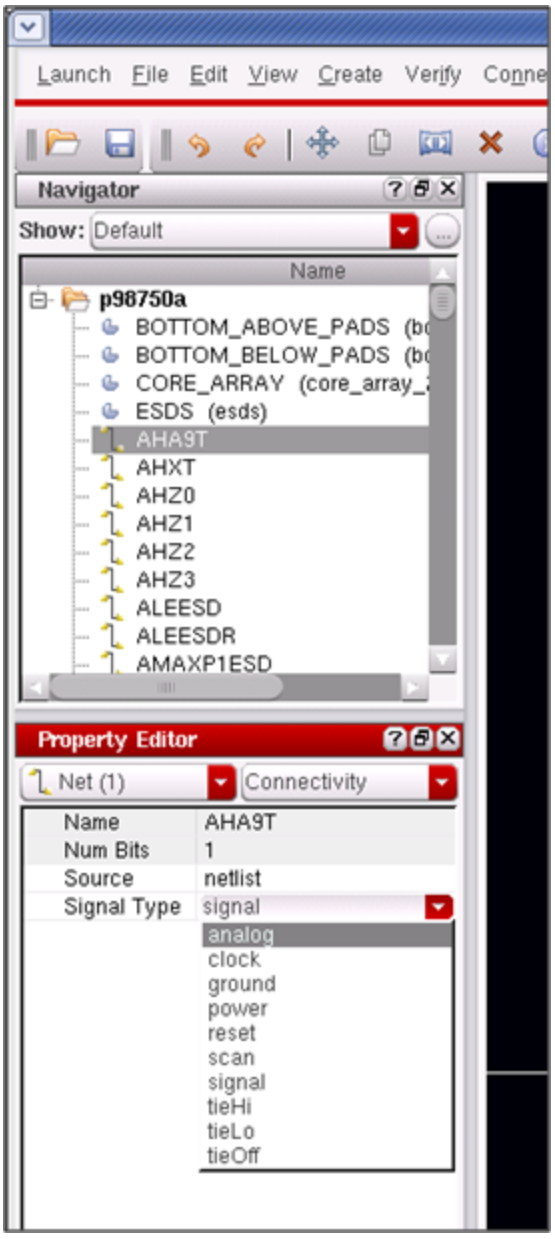
9.3.5 Highlighting Locked Nets

Highlight locked nets by clicking the highlight icon on the Virtuoso Space-Based Router tool bar. Clicking the icon a second time removes the highlighting.

9.4 Editing Net Attributes in Virtuoso

To edit net attributes,

1. Select the net in the Navigator.
2. Set the Signal Type using the pull-down menu in the Property Editor.



9.5 Wire/Net Mapping Between Virtuoso and Innovus

The following table summarizes the wire/net mapping between Virtuoso and Innovus:

Virtuoso			Encounter		
Wire Creation	**Net Signal Type**	**Route Status**	**DEF Section**	**Net USE Attribute**	**Wire Status Attribute**
Create→Wiring (pathSeg)	signal	normal	NETS		ROUTED
		fixed			FIXED
		locked			COVER
	analog	normal		ANALOG	ROUTED
		fixed			FIXED
		locked			COVER
Create→ Shape (path, pathSeg, rectangle, etc.) and the exception from Create→Wiring (non-default Width or non-default Begin/End Style)	signal	normal	SPECIALNETS		ROUTED
		fixed			FIXED
		locked			COVER
	analog	normal		ANALOG	ROUTED
		fixed			FIXED
		locked			COVER
	power (ground)	normal		POWER (GROUND)	ROUTED
		fixed			FIXED
		locked			COVER

9.6 Handling of Wires from Virtuoso in Innovus

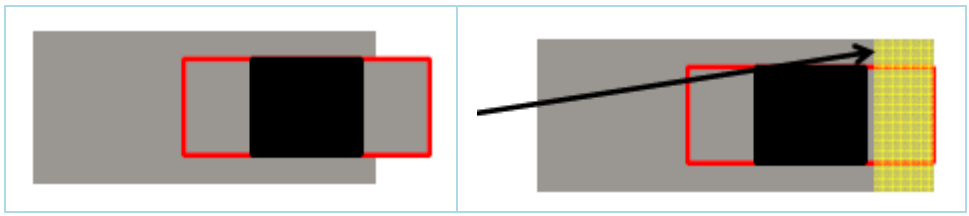
9.6.1 Conversion of Wires to DEF SPECIALNETS in Innovus

Mixed Signal users may face an issue where some wires or wire segments created in Virtuoso may change to DEF SPECIALNETS when brought into Innovus. For example, sometimes, wires created using wire edits, wire assistant auto route, and from Virtuoso Space-based Router may be converted to DEF SPECIALNETS in Innovus. This happens because Innovus does not support all the attributes associated with a wire when brought from Virtuoso (although there is no change in physical shape). Innovus looks for some specific attributes to retain properties on wires brought from Virtuoso. One of these attribute is related to the wire extension. If the wires follow the ½ width or 0 width extension, the wire attribute is retained. For other variable extensions, Innovus will convert the wires to DEF SPECIALNETS.

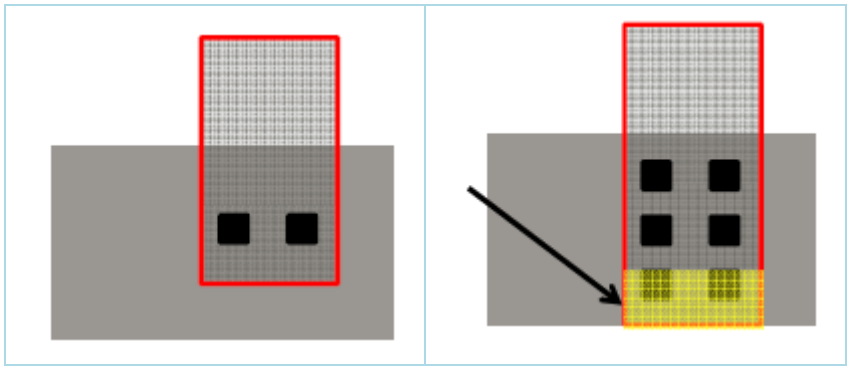
Examples of situations where Innovus would convert a wire to DEF SPECIALNETS

Wire editor and Wire Assistant auto route can create modified extents ,which will cause the conversion of wires to DEF SPECIALNETS. Modified extents are mainly used for:

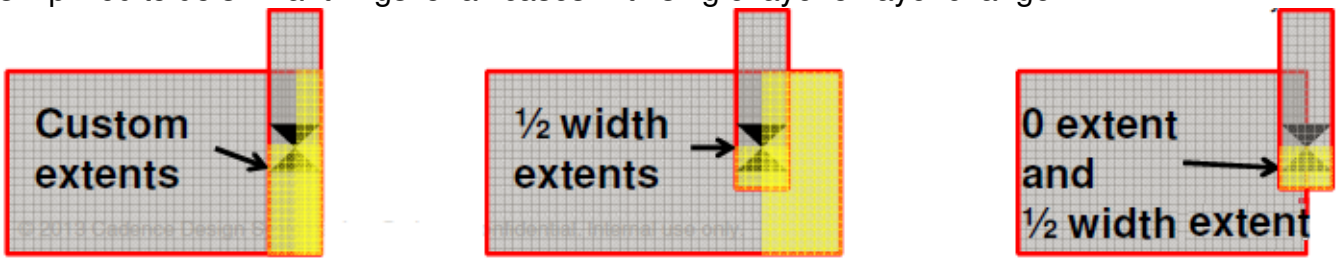
- DRC coverage
  In this example, the via is smaller than the wire width. The extents can be modified to adjust the corners. Note that this can occur from either the creation of vias or the use of custom via or via variants



- Via Maximization
  In this example, the code attempts to create an array of vias between the shapes. The minNumCut rules may or may not kick in and require additional vias but if there is sufficient room to add more, the router will try.



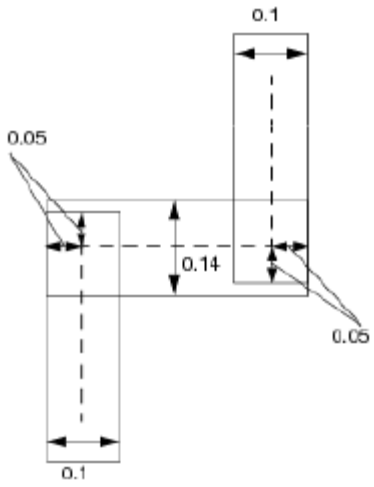
- Directional width and wrong-way routing
  In this example, either because of required directional width rules from the foundry or because of tapering and same layer wrong way routing, the router creates custom extents. There is no clean work around for this one in Innovus if we extend or truncate. The result is that the code has been simplified to do similar things for all cases with single layer or layer change.



  For routing layers that have different preferred direction and non-preferred direction widths, Innovus expects the following

  In the DEF, a vertical default route in the NETS section will have a width of 0.10 µm with extension of 0.05 µm, while a horizontal route will have a width of 0.14 µm with extension of 0.05 µm, as shown in the DEF NETS routing example below:

```
-net1 (…) + ROUTED metal2 (1 0) (1 2) (3 2) (3 3) ;
```



In more advanced nodes, wider widths are required for non-preferred direction. The following example shows route with wrong-way segment:

```
LAYER METAL2...WIDTH 0.1 ;
PROPERTY LEF58_WIDTH "WIDTH 0.14 WRONGDIRECTION ;
" ;...END METAL2
Or
LAYER METAL2...PROPERTY LEF58_WIDTHTABLE "WIDTHTABLE 0.1 ... ;
WIDTHTABLE 0.14 ... WRONGDIRECTION ; " ;...END METAL2
```

Notes

- Wires created through the `run_vsr` command in Innovus do not get affected.
- In Innovus:
  - DEF NETS is treated as symbolic wiring structure (centerline/endpoint connected)
  - DEF SPECIALNETS is geometrically connected (so any overlap or abutment is treated as connected)

9.6.2 Implications of the Conversion in Innovus

The conversion of wires to DEF SPECIALNETS has the following implications in Innovus:

- The conversion can be problematic because some parts of the flow in Innovus, such as buffer insertion on special nets, would not work well.
- Sometimes a special setting may be needed in Innovus. For example, parasitic extraction needs special settings to extract special nets.

- If the completed wires are not intended to be touched by Innovus tools, put a `skip_routing` attribute on those nets in Innovus. To do so, you can use one of the following commands:
  - `setAttribute -net net_name -skip_routing true`
    Or
  - `dbSet $net_ptrs.skipRouting true` (Use the `dbGet` command to pass nets with a specific attribute to the `dbSet` command.)

Note: To bring back the wires with the original attribute to Virtuoso after the round trip, for example after VXL > Innovus >VXL, set the following mode before opening an OpenAccess database in Innovus: `setOaxMode -updateMode true`

9.6.3 Addressing the Conversion Issue

You can use post route calls to clean up the modified/variable extents in Virtuoso. Use the following command in Virtuoso after all the wire edits are done:
`route_optimize -via_extent_optimization truncate -max_iterations 0`

The above command changes the wiring and verify DRC correctness internally.

The `run_vsr` interface automatically calls the `route_optimize` command to minimize the above issue.

To find out which wires will be affected in Virtuoso, you can use OpenAcess DataBase Interoperability Checker. Use the *Check for incompatible wire/wire segments* option on the *Design* page of the OA DB Checker to identify the wires that will be affected when taken into Innovus.

Note: In Innovus, you can use the `defOut` command to see if the corresponding net is in the SPECIALNETS or NETS section.

9.6.4 How To Prevent the Conversion Issue

In some situations, a pathSeg may have *Part of a Route* set to false and be represented as DEF SPECIALNETS but the corresponding via might have *Part of a Route* set to true and be represented as DEF NETS. In this case, when the design is taken to Innovus, there will be problems during the optimization step because NanoRoute might delete the vias.

To fix this problem within Virtuoso, you can convert the via from DEF NETS to DEF SPECIALNETS by using the Skill routine defined below as follows:

In CIW:

```
load("partOfRtFalseForNetsDefinedInFile.il")

partOfRtFalse("library" "cell" "view" "filewithPGnets.txt")
```

In the above call, replace the *library*, *cell*, and *view* as required. For example, if your library is `amsPLL`, cell is `AA` and view is `layout`, the call would be as follows:

```
partOfRtFalse ("amsPLL" "AA" "layout" "filewithPGnets.txt")
```

The `filewithPGnets.txt` lists the power and ground nets in the design, such as VDD, VSS , and so on.

Skill routine:

```
procedure(partOfRtFalse(library cell view fileName "t")let( (cv fp mylist mylist1 l_shape v_shape newmlist newvlist)
envSetVal("layout" "extractKeepShapeAssignment" 'boolean t); check if the file passed existsunless(isFile(fileName)  error("Please pass an existing file to the procedure") );unless

; open the file for read and get list of nets; in it into mylist variablefp = infile(fileName) ; In their flow hard code the location of the file and call it nets.txt.. check when(fp      while( gets( nextLine fp)      mylist = append(parseString(nextLine "\n ") mylist)       )   close(fp) );when

; make sure the elements in mylist are nets existing in the current design; provide warning for net names present in the input file but not in the currently open design.cv = dbOpenCellViewByType(library cell view);cv = geGetEditCellView()mylist1 = setof(elem mylist !member(elem cv~>nets~>name)            )

when(mylist1  warn("Nets %L listed in the input file %s are not present in the current cellview %s/%s/%s" mylist1 fileName cv~>libName cv~>cellName cv~>viewName) );when

mylist = setof(elem mylist              member(elem cv~>nets~>name)          )

when(mylist

 printf("List of nets present in the text file are: %L\n" mylist)

;Un-Comment this block section if need to work on pathSegs.; Here working on pathSegl_shape=setof(obj cv~>shapes obj~>objType == "pathSeg")              when(newmlist=setof(x l_shape member(x~>net~>name mylist))

              when( newmlist~>route!=nil && newmlist~>net~>name!=nil  foreach(targetshp newmlist ;targetshp~>lxStickyNet=t targetshp~>route~>objects=remove(targetshp targetshp~>route~>objects)



          );foreach       );when       );when

; Here working on stdVia defsv_shape=setof(vobj cv~>vias vobj~>objType == "stdVia")       when(newvlist=setof(x v_shape member(x~>net~>name mylist))            when(newvlist~>route!=nil && newvlist~>net~>name!=nil            foreach(targetvshp newvlist            ;targetvshp~>lxStickyNet=t            targetvshp~>route~>objects=remove(targetvshp targetvshp~>route~>objects) printf("Attribute Part Of Route set to FALSE for Vias with net name %L...completed.\n" targetvshp~>net~>name)            );foreach );when       );when

  );when mylist

;dbSave(cv);dbClose(cv)
```

);let);procedure

9.7 Wiring Connectivity in Innovus

- Nets with the `COVER` attribute:
  - Are not modified by Innovus optimization or routing.
  - Cannot be moved by either automatic layout or interactive commands.
- Nets with the `ANALOG` attribute:
  - Are not modified by optimization
  - This attribute can be imported from the OpenAccess database or specified within Innovus.
- NanoRoute routes analog nets unless you specify the following command before routing:
  ```
  setNanoRouteMode -route_skip_analog true
  ```
- Early Global Route routes analog nets by default.
- Connectivity extraction in verify connectivity based on shape overlap
  geomConnect
- Innovus wire editing will preserve the connectivity and DEF representation.

9.8 Wiring Extraction in Innovus

In case your pre-routes have been created with geometric wires (outside a route/ special nets), you need to make sure to use the following option to get you pre-routes correctly extracted:

Extraction within Innovus should enable special net handling in order to perform extraction on analog/special nets:

```
setExtractRCMode -specialNet true
```

You can specify this option in default and detail mode of extraction. Special nets are always extracted in sign-off extraction mode.

Innovus wire editing will preserve connectivity and DEF representation.

10

Trim Metal Interoperability

- Introduction to Trim Metal
  - The TRIMMETAL LAYER
  - General Requirements for N7, N5, and N3 Designs
  - What Is Trim Metal?
- Technology Requirements for Trim Metal Insertion
- Trim Metal Insertion Flow in Virtuoso
  - Trim Shape Rules
  - Trim Shape Extension Models
    - Extension from center
    - Extension from metal edge
    - Extension to adjacent track
- Supporting Tool Enhancements

10.1 Introduction to Trim Metal

10.1.1 The TRIMMETAL LAYER

The `TRIMMETAL` type masterslice layer is used to define trim metal layers. A trim metal layer is used along with metal layers manufactured with the self-aligned double patterning (SADP) technology, which is a form of double patterning technology (DPT). Several new LEF rules are applied on the trim metal layer. Place and route tools need to utilize this trim metal layer to achieve better QoR. `defIn`, `defOut`, and GDS `streamOut` need to support trim metal for RC extraction and physical verification.

The standard cell library may violate DRC rules, and you may require the router to patch on PIN and OBS to fix Minimum Area Rule (MAR) and/or enclosure violations as an example.

The first routing layer becomes `M0` and is used for power rail connection.

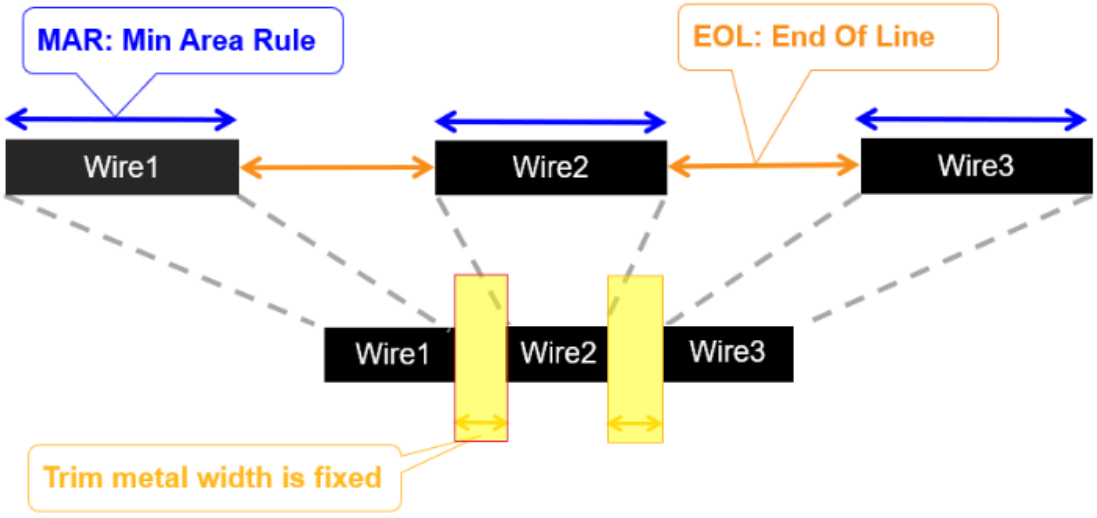
10.1.2 General Requirements for N7, N5, and N3 Designs

N7/N5/N3 designs have the following general requirements:

- All objects, such as macro pins, block/IO pins, and PG stripes, in N7/N5/N3 designs should be on a track or grid for SADP or DPT layers.
- All objects must be rectangles and not rectilinear shapes.
- The place and route tools must follow the track color to place and route.
- The router can touch fixed objects, such as standard cell pin/obs and PG nets and vias.
- Cell libraries should be fully colored.

10.1.3 What Is Trim Metal?

Trim (cut) metal is used to shrink the end-to-end spacing, enclosure and Minimum Area Rule (MAR). The trim metal should always be sandwiched between two wire line ends along the preferred direction on `TRIMMEDMETAL`.

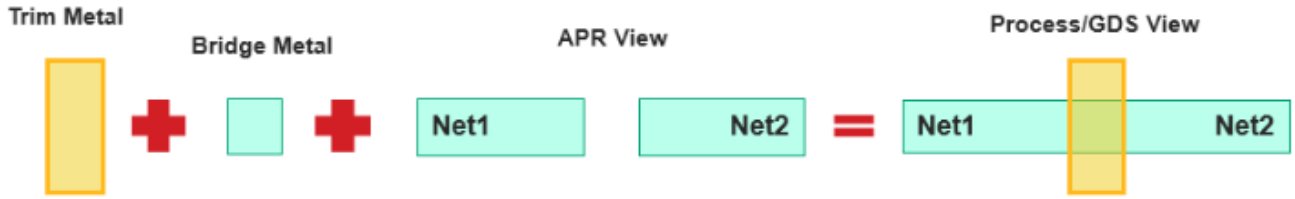


Consider the gap between wire line ends in the Automatic Place and Route (APR) view:



Metal needs to be generated to fill the gap between line ends in GDS, DEF, OpenAccess, and Full

Geometry Checker (FGC). Additional metal shape, known as bridge metal, is generated during GDS `streamOut`. Trim metal and bridge metal is added to the APR view to get the Process/GDS view.



Routing trim metal supports:

- Trim metal formation
- Cell object extension for recommended locations
- Trim metal spacing rule
- Trim metal to metal spacing rule
- Trim metal aware area rule
- Trim metal aware enclosure rule
- Trim metal for only one color on a metal
- Fill up metal gap between wire line ends for checking other rules

10.2 Technology Requirements for Trim Metal Insertion

To enable trim metal insertion, you must define the trim metal layers in the `functions` section of the ASCII tech file.

```
layerRules(
   functions(
   ;( layer      function      [maskNumber]      [attributes])
   ;( -----      --------      ------------      ------------)
    ( M1         "metal"       104                           )
    (CM1A        "trim"        105       'trims ((M1 m1ca_interconnect) ))
    (CM1B        "trim"        106       'trims ((M1 m1ca_interconnect) ))
   );functions
```

You must also add the OpenAccess layer map (`OALAYERMAP`) property in the `techParams` section of

the tech file. The `OALAYERMAP` property is used to define equivalence between LEF and OpenAccess layers and also between the geometries and shapes placed on the layers.

```
techParams(
   LEF58_OALAYERMAP "OALAYERMAP CM1A LAYER CM1 MASK 1 ; OALAYERMAP CM1B LAYER CM1 MASK
2;"
);techParams
```

Finally, add the `trimShape` constraint in the ASCII tech file:

```
spacings(
   ( trimShape tx_layer
       'exactWidth f_exactWidth
      ['maxLength f_maxLength]
      ['exceptSpacing f_spacing]
      ['exceptWidth f_width]
      ['extension f_extension ['metalEdge]]
      ['mask1 | 'mask2 | 'mask3 | 'mask4]
      ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)] |
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      ['adjacentWidthExtension (f_adjWidth f_adjExtension)]
      ['keepoutRegion (f_offset1 f_offset2)]
      {"adjacentTrack" | "midTrack" | "extension"}
   )
) ;spacings
```

The `exceptSpacing` and `exceptWidth` parameters trigger the interactive trim insertion.

A trim shape rule is created by using the `TRIMSHAPE` property in LEF:

```
PROPERTY LEF58_TRIMSHAPE
      "TRIMSHAPE EXTENSIONMODEL
         {ADJACENTTRACK | MIDTRACK | EXTENSION extension [METALEDGE]}
         EXACTWIDTH exactWidth
         [MAXLENGTH maxLength [MASK maskNum]]
         [EXCEPTSPACING spacing] [EXCEPTWIDTH width]
         [USEMETALMASK]
         ; " ;
```

The table below depicts the mapping of the `TRIMSHAPE` LEF property to ASCII:

Tech LEF	Virtuoso (ASCII)

PROPERTY LEF58_TRIMSHAPE	trimShape
EXTENSIONMODEL {ADJACENTTRACK | MIDTRACK | EXTENSION [METALEDGE]}	{"adjacentTrack" | "midTrack" | "extension"} ['extension f_extension ['metalEdge]]
[EXCEPTSPACING spacing]	'exceptSpacing
EXACTWIDTH	'exactWidth f_exactWidth
[MAXLENGTH [MASK maskNum]]	'maxLength f_maxLength] ['mask1 | 'mask2 | 'mask3 | 'mask4]
[EXCEPTSPACING spacing]	['exceptSpacing f_spacing]
[EXCEPTWIDTH width]	['exceptWidth f_width]

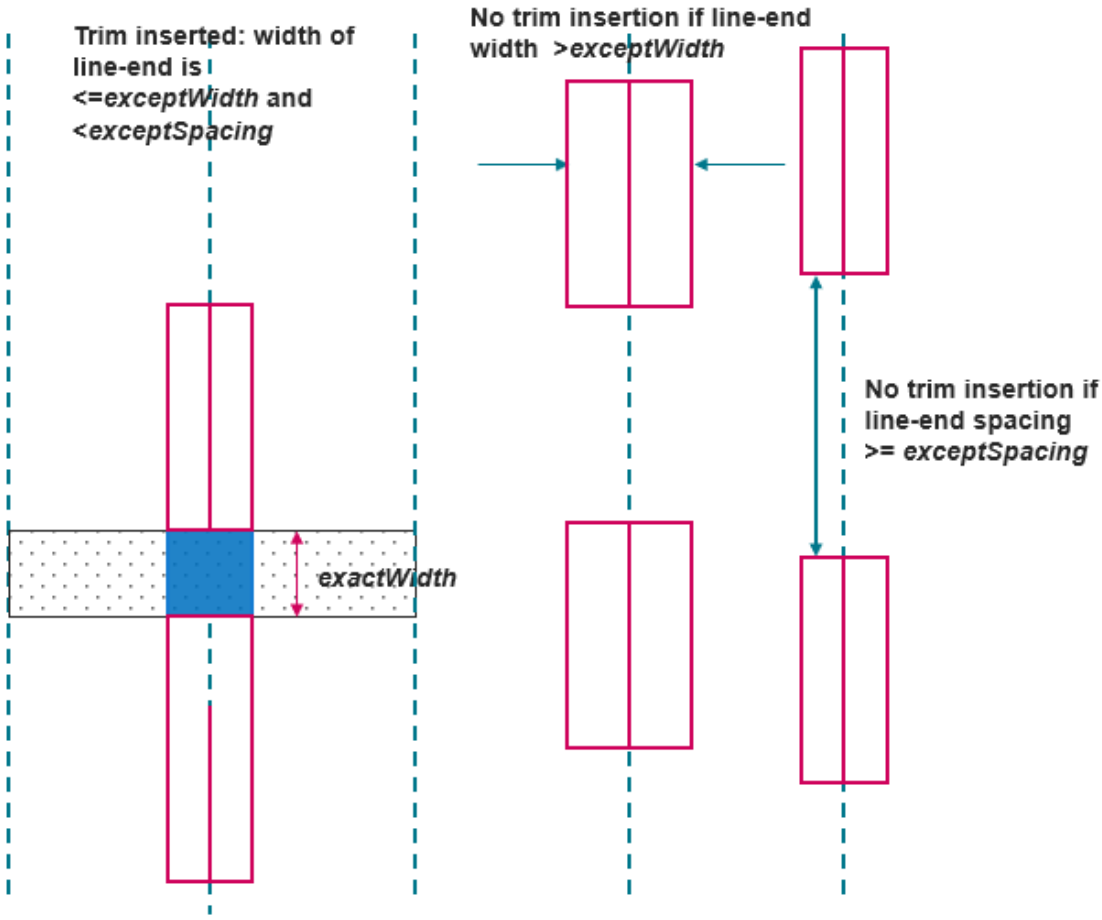
10.3 Trim Metal Insertion Flow in Virtuoso

10.3.1 Trim Shape Rules

Trim shape rules can be used to specify that shapes on the `TRIMMETAL` layer should be formed based on the line-end wires on the metal layer that is trimmed. The rules for trim metal shape insertion are as follows:

- The trim shape must be rectangular and not rectilinear.
- The width of the trim shape measured in the preferred routing direction of the trimmed metal must be exactly equal to the *`exactWidth`* value.

- The trim shapes must always be formed at the line-end of wires on the trimmed metal.
- When the width of the line-end is greater than `exceptWidth`, if specified, the trim shape is not formed.
- When the line-end to line-end spacing is greater than or equal to `exceptSpacing`, if specified, the shape is not formed.

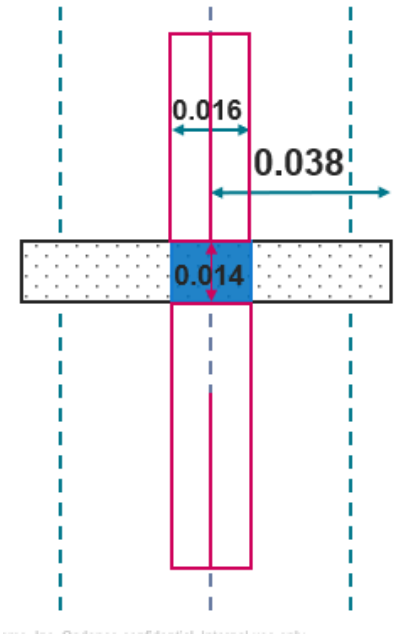


10.3.2 Trim Shape Extension Models

Trim shapes can be extended using the following extension models:

10.3.2.1 Extension from center

A trim shape can be extended from the center by the specified extension value in both directions. In the example below, the trim shape is extended from the center by `0.038` in both directions.



10.3.2.2 Extension from metal edge

If `metalEdge` is specified, the trim shape is extended from the metal edge in both the directions by the specified extension value. In the example below, the trim shape is extended from the metal edge by `0.022` in both directions.

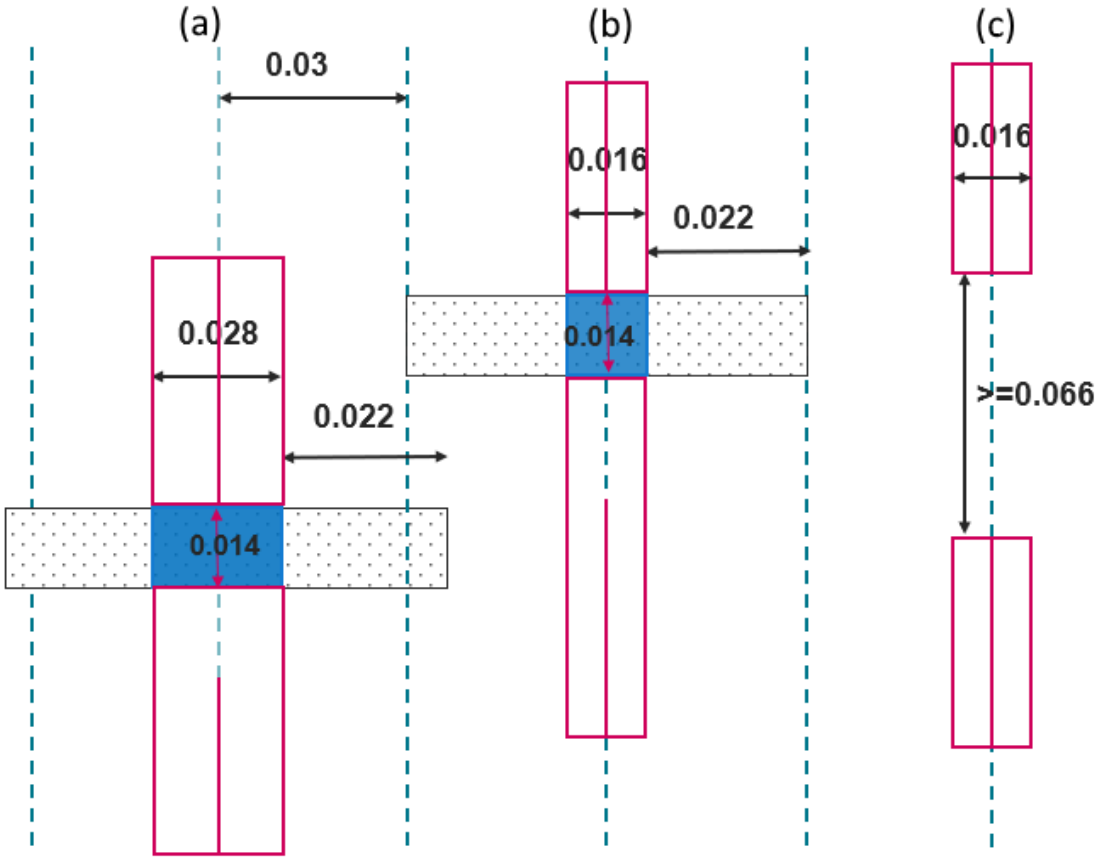
Consider the extension model specification below from the ASCII tech file:

```
spacings(

      trimShape "CM1A" 'exactWidth 0.014 'maxLength 3 'exceptSpacing 0.066 'exceptWidth
0.028 'metalEdge 'extension 0.022)

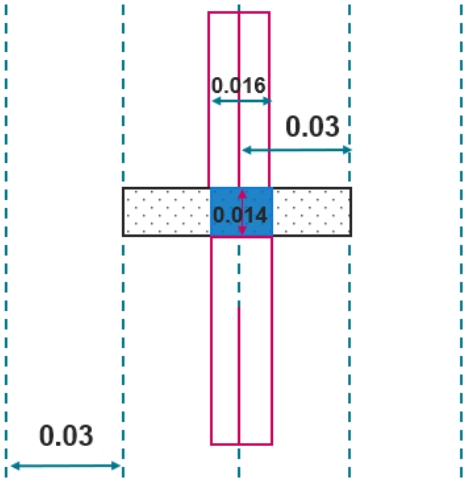
);spacings
```

Trim shapes are extended from the metal edge by `0.022` in both directions in (a) and (b) as `exceptSpacing` and `exceptWidth` conditions are met. In (c), the line-end to line-end spacing is greater than `exceptSpacing`, so the trim shape is not formed.



10.3.2.3 Extension to adjacent track

If adjacentTrack is specified, the trim shape extends to the center line of the adjacent tracks in the orthogonal direction.



10.4 Supporting Tool Enhancements

The OpenAccess Database Interoperability Checker (OA DB Checker) has been enhanced to detect the presence of trim metal in the design. This will alert you that you are using the interoperability flow.

In addition, the `leReportTrimmedShapesInCustomStyle` SKILL API can be used in Virtuoso to identify locations where metal and trim shapes were drawn in Virtuoso routing style.
The `leConvertTrimmedShapesToPRStyle` API can then be used to convert such shapes to P&R style for interoperability with Innovus.

11

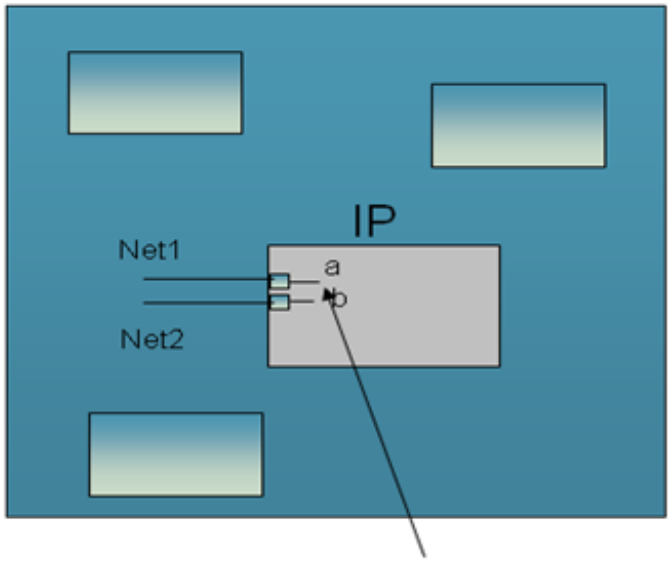
Routing Constraint Interoperability

- An Overview of Routing and Constraint Interoperability
- Steps for Creating an Interoperable NDR in Virtuoso
- Creating Interoperable Shielding Constraints in Virtuoso
- Routing Constraints Understood by NanoRoute (Digital Router in Innovus)
- Creating/Viewing Interoperable Routing Constraints in Virtuoso
    - Creating Interoperable Constraints Using Virtuoso Constraint Manager
    - Checking OpenAccess Database with Constraints Prior to Bringing a Design into OA DB Checker System
- Creating Interoperable Routing Constraints Using Innovus
- Hierarchical Propagation of Constraints
- Creating/Adding Mixed-Signal Routing Constraints on IP Blocks
- Routing Constraints Interoperability between NanoRoute and VSR
    - Getting a List of Nets with skip_routing Attribute
    - Checking Routing Results Against the Routing Constraints
    - Creating Interoperable Library between Innovus and Virtuoso
- Trying the Constraint Interoperability Environment and the run_vsr Command in Innovus for the First Time

11.1 An Overview of Routing and Constraint Interoperability

Given the full interoperability of the OpenAccess environment between various Cadence tools, Cadence has taken steps to create an environment in which design constraints can be shared across various platforms. For instance, in the Mixed Signal flow where top level of the design is captured as schematics in Virtuoso, routing constraints can be defined in Virtuoso for the digital portion of the design, which will later be implemented by Innovus. Conversely, in a flow where the top level of the design is captured as a Verilog netlist in Innovus, Innovus can define routing constraints that will be used by block designers when such blocks are later implemented in Virtuoso. In addition to these capabilities, mixed signal IPs can be embedded with the required integration constraints so that when the end users instantiate such IPs in their designs, the embedded constraints can be pulled to the top level of the design. This ensures that the integration requirements of the particular IP are met during the design process. Refer to the section titled “Creating/Adding Mixed Signal Routing Constraints on IP blocks” for a detailed description of this functionality.

The picture below depicts the instantiation of an IP that contains embedded routing constraints which dictate that top-level nets connecting to certain terminals of the block should be routed as Differential Pairs. The OpenAccess abstract of the IP contains such embedded constraints.

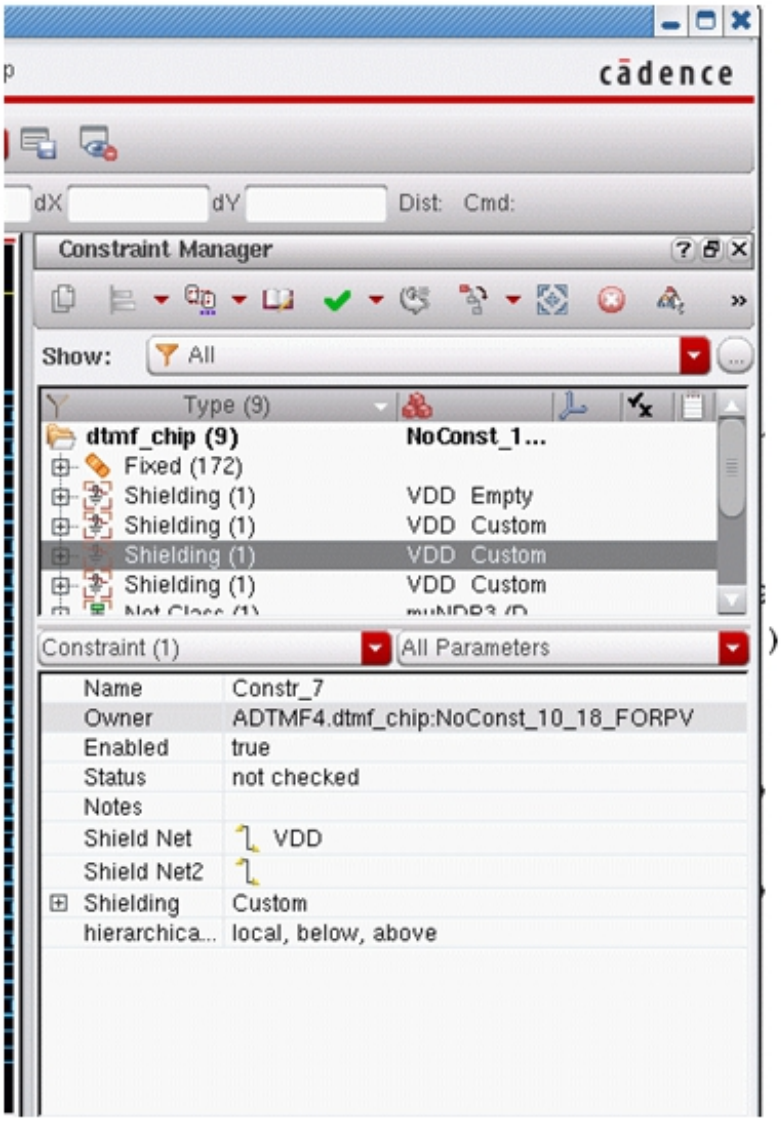


diffPairGroup (a b oacMinSpacing oacMinWidth)

If the IP is instantiated in either Virtuoso or Innovus, the Differential Pair constraint is propagated to the top-level nets (Net1 and Net2 in the above example) when you invoke commands that propagate these constraints to the top level. In Innovus, the command to pull constraints to the top

level is `pull_block_constraint`.

The hierarchical propagation of the constraints is controlled by the `hierarchicalScope` attribute, which is created at the time the constraints are initially created, either directly in a design or embedded in an IP. The `hierarchicalScope` property could be placed on any constraint, either in Virtuoso or in Innovus. The picture below shows how the hierarchical scope can be set on a shielding constraint. As the hierarchical scope value is `local`, `below`, `above`, it will instruct all applications to allow for Pushing (as defined by `below`) and Pulling (as defined by `above`) of this constraint.



After the implementation of the design is complete, PVS-cv can be used to check the final implementation of the design against embedded IP constraints that exist. Any implementation aspects of the design that do not conform to the intended design constraints are flagged as

violations by PVS-cv.

A design checker utility is included in Innovus and Virtuoso installations. It could be used to check Virtuoso generated constraints for interoperability with Innovus. For more information, see the OpenAccess Database Interoperability Checker chapter.

11.2 Steps for Creating an Interoperable NDR in Virtuoso

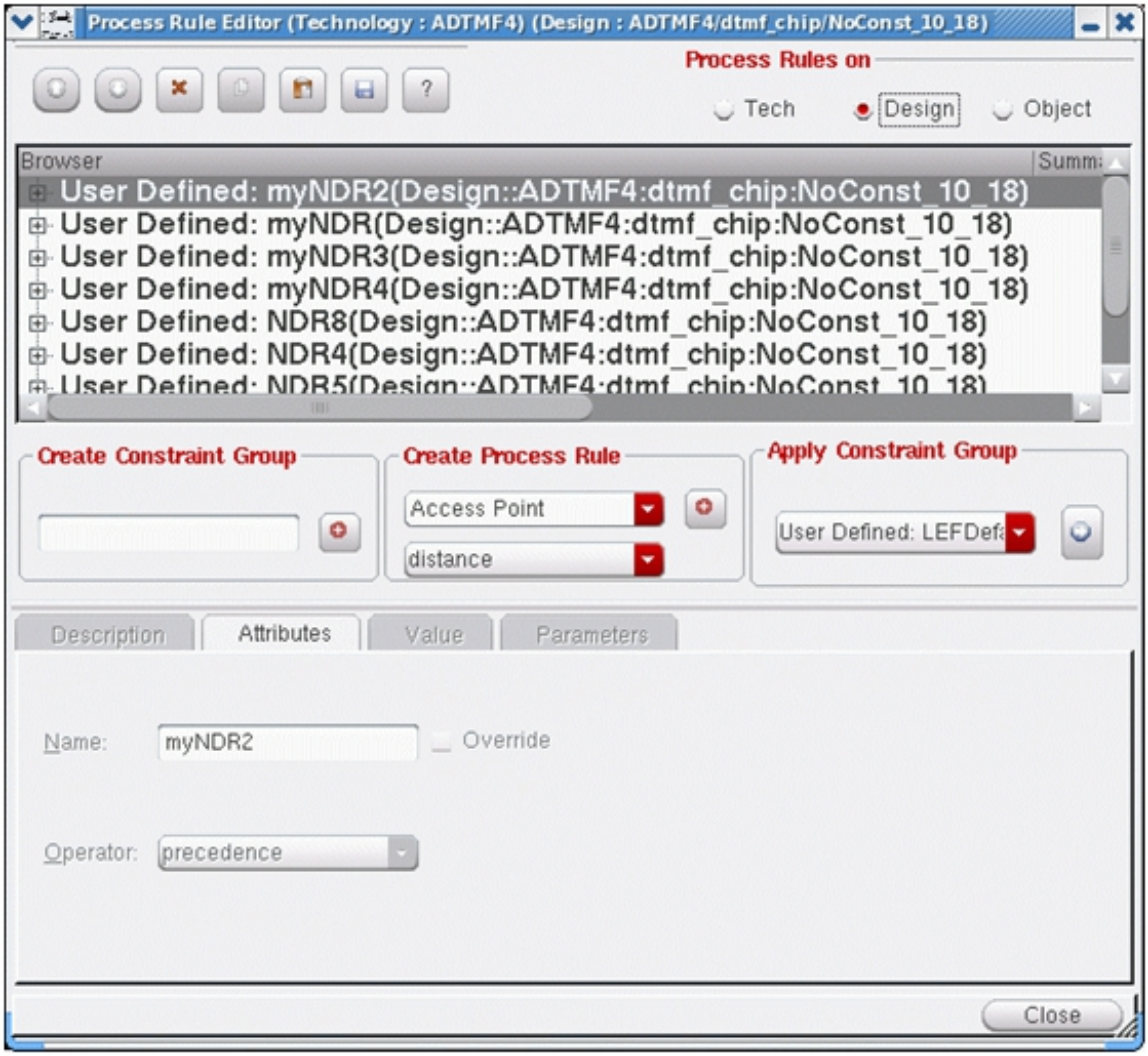
The Virtuoso Constraint Manager (VCM) enables you to create interoperable Non-Default Rules (NDRs). Virtuoso XL or GXL licenses are needed to access the Virtuoso Constraint Manager environment. There are several ways to create an NDR in Virtuoso.

Use the following steps for creating an interoperable NDR.

1. After loading your design in Virtuoso, select the *Process Rule Editor* option from the Virtuoso Constraint Manager pull-down menu. (Remember to involve Virtuoso XL or GXL.)



2. After the Process Rule Editor window is open, select the *Design* radio button. If there were existing NDRs in the design, you will see them displayed in the top part of the form.



3. In the Process Rule Editor window, enter a suitable name for the design NDR you are creating in the *Create Constraint Group* text box and then click the “+” sign next to it. You should now see the new NDR name in the upper portion of the form.
4. Scroll down the *Create Process Rule* drop-down list and select the *Valid Layers* option as shown in the picture below. Click the “+” sign next to the field.

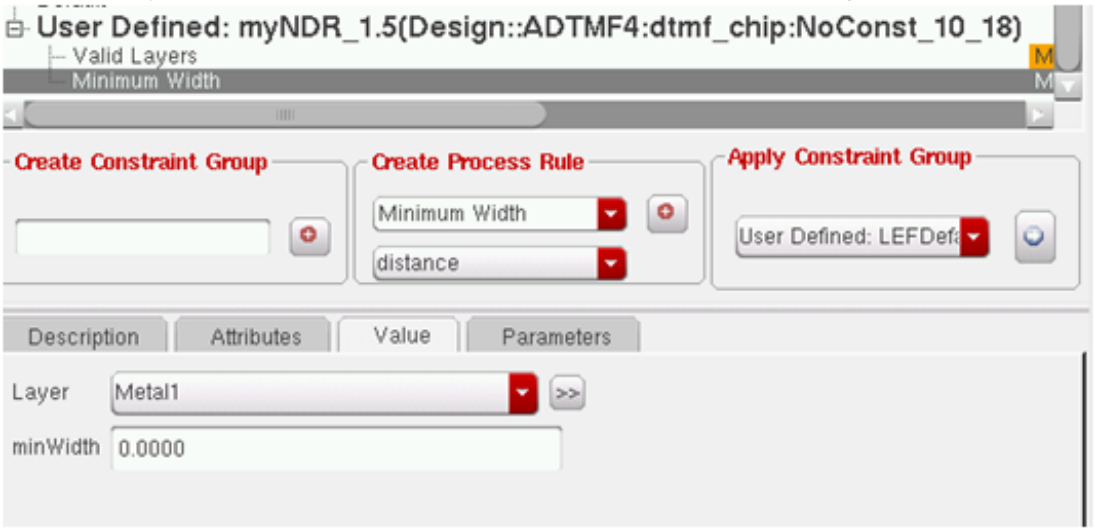


5. Select the *Value* tab as shown in the picture below and select the routing layers you wish to have for this NDR. Note that in order for Innovus to recognize the NDR as valid, the routing

layers for the NDR need to be selected from the routing layers defined in the LEFDefaultRouteSpec.



6. You have now completed the creation of a valid interoperable NDR/PRO between Innovus and Virtuoso. You can use the above method to create minimum width for all the layers, as well as any other attributes for this NDR/PRO. Refer to the picture below.



Note: Any PRO/NDR created using the above method is a design PRO/NDR. Ensure that you save the design after creation of PRO/NDRs. In addition, once the design is opened in Innovus, the `writeIntegRouteConstraint` command can also be used within Innovus to dump out the constraint in text format for viewing, validation, and further modification.

11.3 Creating Interoperable Shielding Constraints in Virtuoso

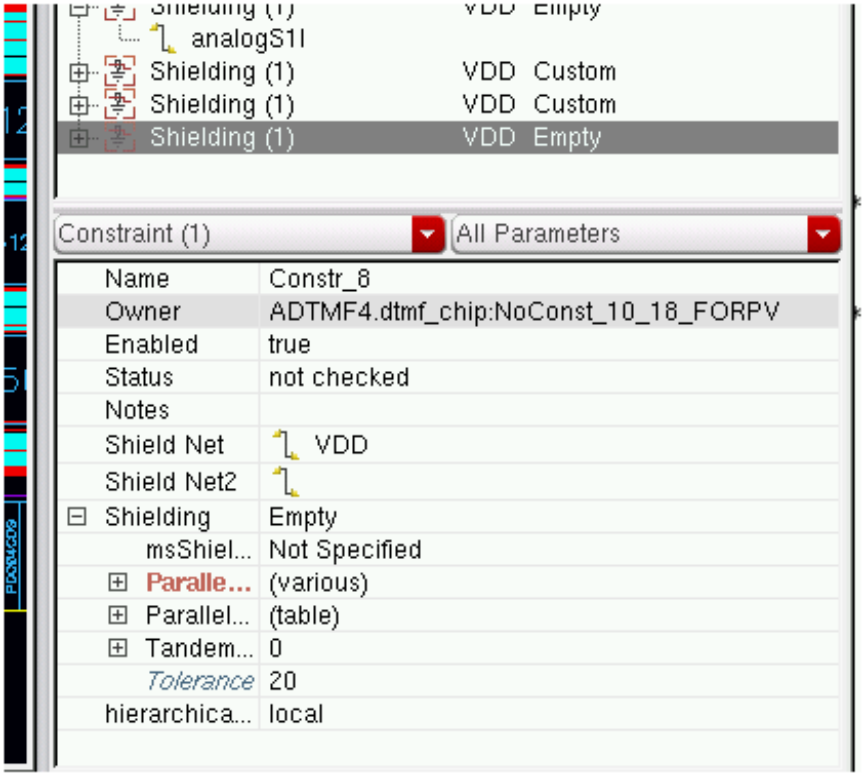
The interoperable environment allows multiple methods for creating shielding constraints in Virtuoso. In the mixed-signal flow, you can use the Virtuoso Space-based Router (VSR) to pre-route some mixed-signal nets and then use NanoRoute to complete the routing for the design. The definition of the shielding constraint in Virtuoso determines which router will be used to route the net that is to be shielded.

- If you would like to create a shielding constraint and use NanoRoute for creating the shielded net, you should create a shielding constraint in Virtuoso with `msShieldStyle = "Not Specified"` or `"parallel"`. This will translate into a *Simple Shielding* constraint in Innovus that can be recognized by NanoRoute.

  Note: Except `msShieldStyle`, no other parameter of the shielding constraint should be modified in Virtuoso. If any other parameters are modified, Innovus will not recognize the shield constraints as *Simple Shielding*.

- If you would like to create a shielding constraint and use VSR for creating the shielded net, you need to create a shielding constraint in Virtuoso with `msShieldStyle` set to the actual style you would like to use. The possible shield styles that you can use are `Tandem below`, `Tandem below + parallel`, `Tandem above`, `Tandem above + Parallel`, `Tandem`, and `Coaxial`. When you do so, all the nets with this type of shielding constraint will be recognized as a complex shield in Innovus and will be recognized by VSR as valid routing constraints.

The picture below shows a correctly formed interoperable shielding constraint, which will be interpreted by Innovus and Virtuoso as a *Simple Shielding* constraint. Note that any nets having a *Simple Shielding* constraint can be routed by NanorRoute in Innovus. The `msShieldStyle` could have also been set to `parallel`:



Note that VSR can also be used to route shielded nets where the `msShieldStyle = Not Specified` or `parallel`. If these nets are routed by VSR before NanoRoute is used to route the rest of the nets in the design, you must use the following command in Innovus to ensure that NanoRoute does not remove the previously routed nets.

```
setAttribute -net net_name -skip_routing true
```

11.4 Routing Constraints Understood by NanoRoute (Digital Router in Innovus)

NanoRoute supports all of the following:

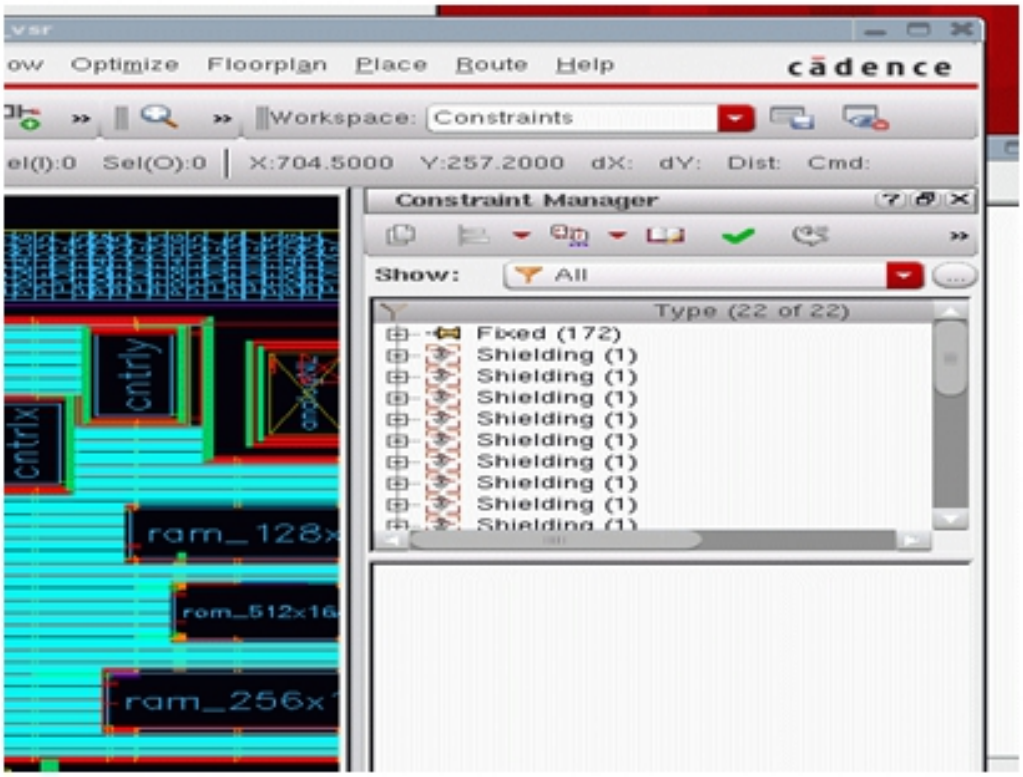
- Simple shielding
- Parallel shielding
- Nets with interoperable NDRs/PROs
- NetClass constraint (where a group of nets have the same constraints)

11.5 Creating/Viewing Interoperable Routing Constraints in Virtuoso

If you prefer to enter mixed-signal routing constraints using the Virtuoso environment, the Virtuoso *Constraint Manager* can be used to populate a design with the required routing constraints.

Virtuoso *Constraint Manager* is capable of creating some constraints that are used only by Virtuoso applications. Although these constraints are retained in the database as the design moves between Innovus and Virtuoso, none of the Innovus applications are able to interpret such constraints.

The Virtuoso *Constraint Manager* can be accessed by setting the workspace as constraints, using Virtuoso XL.



For more details on the user interface aspects of the tool, refer to the "Constraint Manager Assistant" chapter of the *Virtuoso Unified Custom Constraints User Guide*.

11.5.1 Creating Interoperable Constraints Using Virtuoso Constraint Manager

To create a differential pair or a match pair interoperable constraint using the Virtuoso Constraint Manager, follow these steps:

1. Using the Navigator, select the nets that should be a part of the differential pair or the match pair constraint group.
2. Using the Virtuoso *Constraint Manager*, select a differential pair constraint.
3. If a differential pair must have shielding constraints, create an additional shielding constraint on the net members of a differential pair or a matched pair.

11.5.2 Checking OpenAccess Database with Constraints Prior to Bringing a Design into OA DB Checker System

OpenAccess Database (OA DB) Interoperability Checker is a SKILL-based utility which can be run in the Virtuoso Environment to ensure that the OpenAccess database being brought into Innovus is compatible for the place-and-route flow, and all constraints entered are interoperable. The OpenAccess database should contain all the necessary database objects which are required by applications within Innovus. At the same time, it should not contain any database object which would stop the Innovus implementation flow.

For information on loading the OA DB checker, see OpenAccess Database Interoperability Checker.

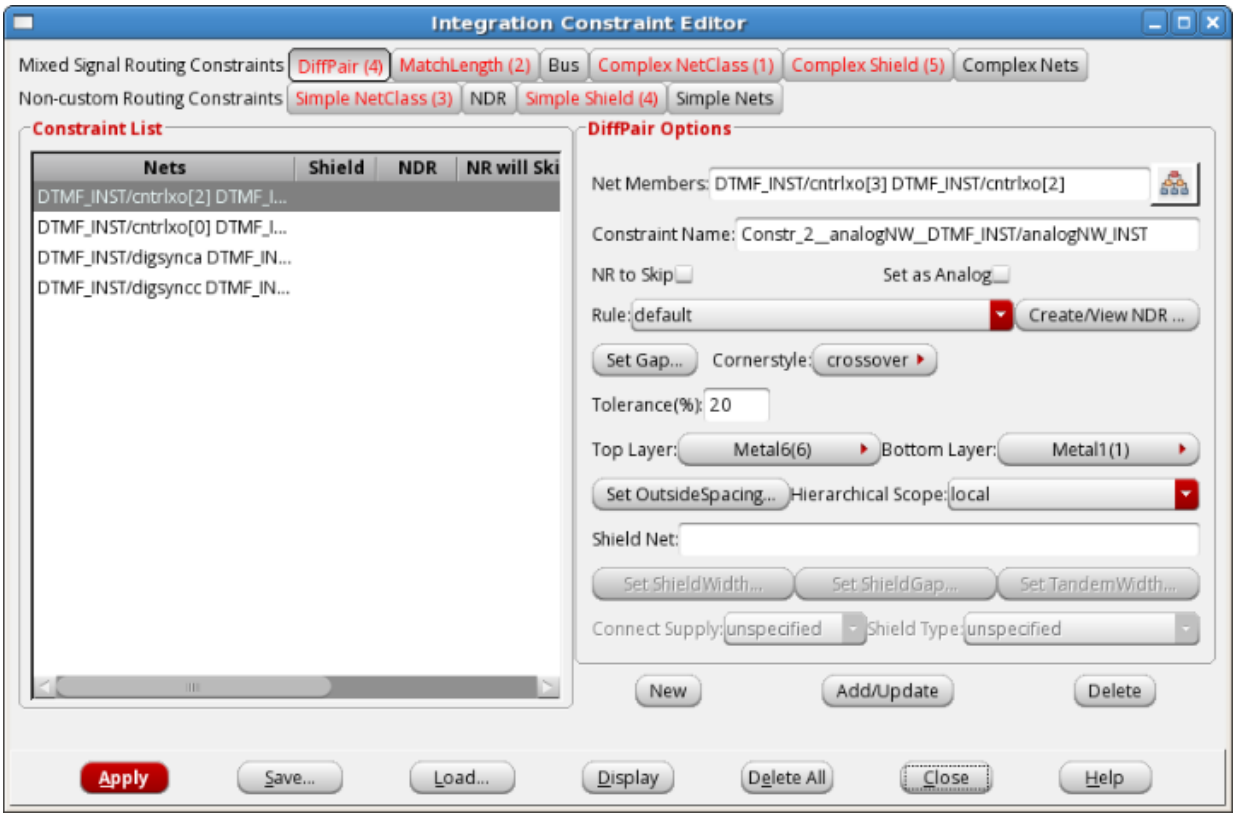
11.6 Creating Interoperable Routing Constraints Using Innovus

Use the Integration Constraint Editor in Innovus to create interoperable mixed-signal routing constraints.

Note: The routing constraint interoperability requires the OpenAccess database to be used while creating a design library. In addition, a mixed-signal ready OpenAccess PDK should be used as the technology library.

The Innovus constraint-entry mechanism can be accessed by choosing *Tools - Mixed Signal - Integration Constraint Editor*.

The active constraint tabs in the Integration Constraint Editor are labeled in red to make it easy for you to identify them. In addition, the Integration Constraint Editor also displays the constraint count on each active constraint tab.



In the above form, the tabs in the *Mixed Signal Routing Constraints* row display nets that are routed using custom routing techniques. These nets have a property identifying them as Mixed Signal nets, instructing all applications in the Innovus environment to ignore them. The tabs in this row are:

- *DiffPair*

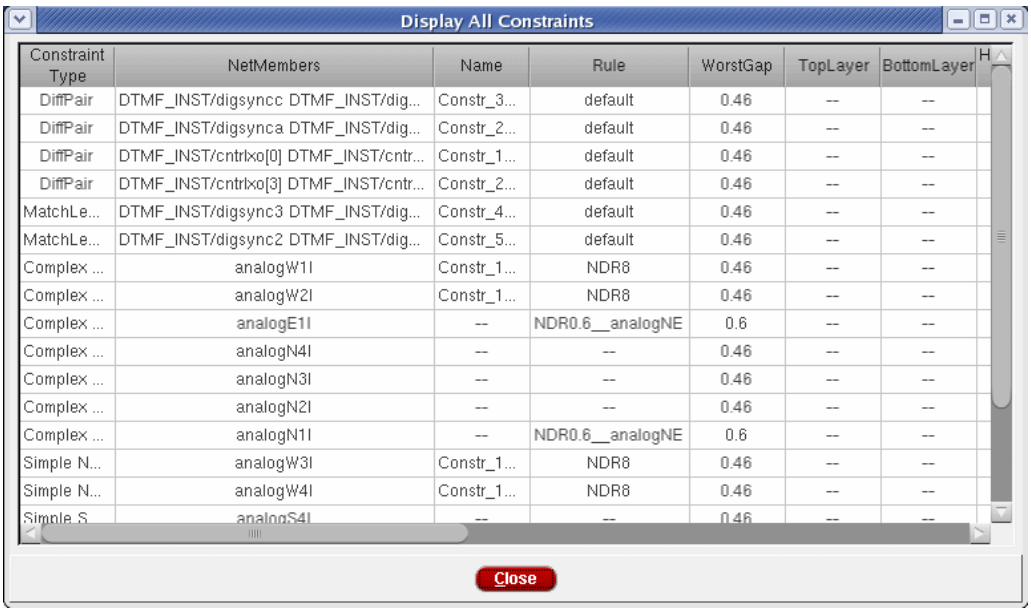
- *MatchLength*
- *Bus*
- *Complex NetClass*
- *Complex Shield*
- *Complex Net*

If a new constraint needs to be created, click the *New* button to clear the form so that nets can be added to the *Net Members* field. The net names can be entered either manually or by using the Design Browser button to the right of the *Net Members* field. This button opens Innovus's design browser within the Select Net Members form. After entering all required parameters of the constraint in the form, click the *Add/Update* button to add the constraint to the design. The net members are then shown under the *Constraint List*.

If an existing constraint needs to be modified, first select the constraint from the *Constraint List* and modify its parameters in the form. After the modification is complete, use the *Add/Update* button to save the constraint. If an existing constraint is to be deleted, first select it in the *Constraint List* and then click the *Delete* button. After the constraint is deleted, associated net names are no longer be shown in the *Constraint List*.

You can save all the constraints entered for the design into a file which can later be loaded and modified. This is not a necessary step, as all constraints are saved with the design in OpenAccess. If an existing constraint file is available, the *Load* button can be used to load the constraints and the *Apply* button to write such constraints into the database.

To see all constraints in the design in a spreadsheet format, click the *Display* button. The Display All Constraints form displays all design constraints.



For more information about the Integration Constraint Editor, see the Tools Menu chapter of the *Innovus Menu Reference*.

11.7 Hierarchical Propagation of Constraints

Virtuoso has the capability to push and pull routing constraints across hierarchies in a design. For instance, a shielding constraint could exist on the net in a sub-block of a design, which connects to a net at the top level. Virtuoso and Innovus can pull a shielding constraint from a sub-block and place it on the corresponding net at the top level. After the constraint is pulled, if NanoRoute or VSR are used to route the top-level nets in a design, the constraint is understood and honored by both the tools.

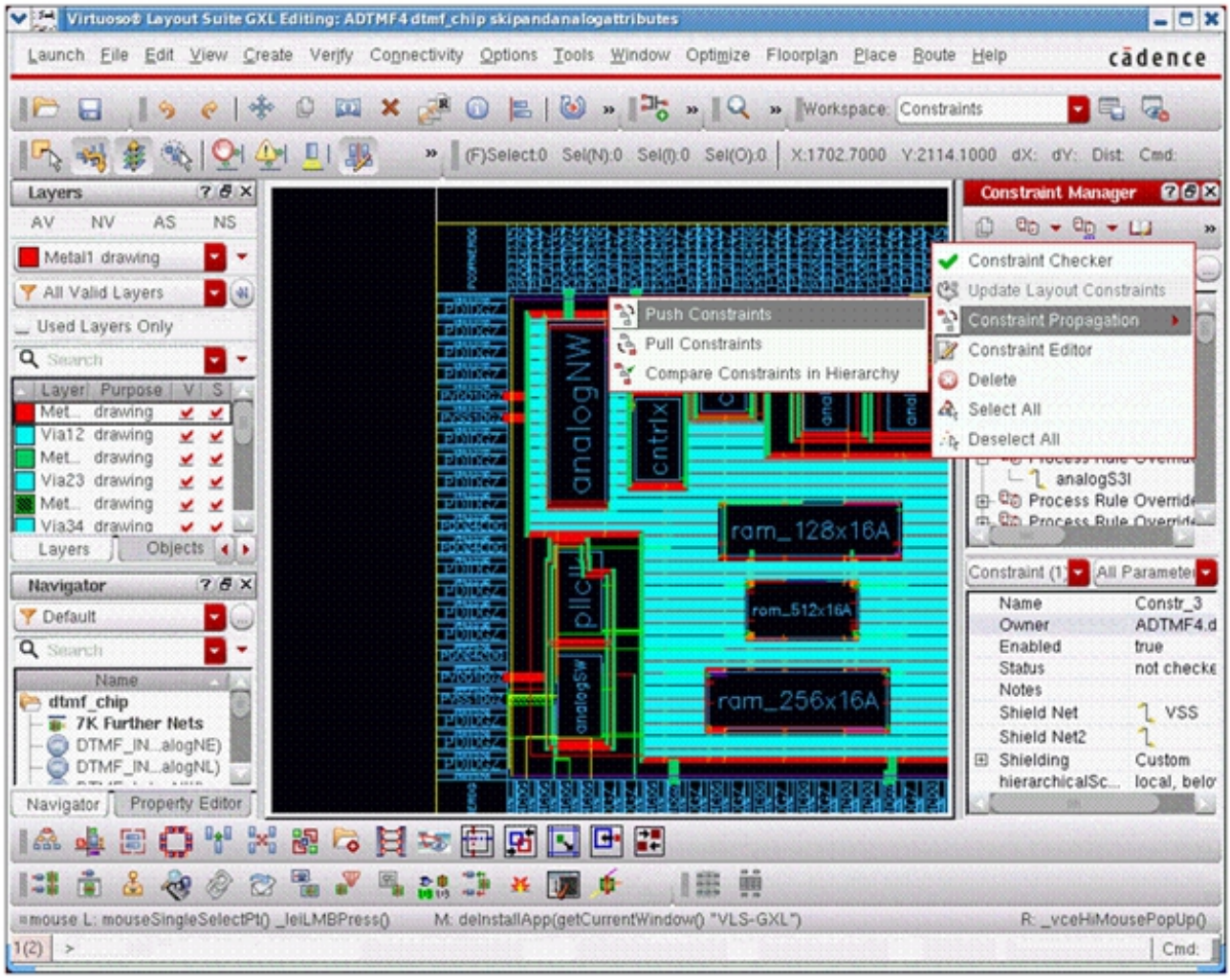
The Innovus command for pulling routing constraint is `pull_block_constraint`.

Note: The hierarchical scope constraint defines whether a particular constraint can be pulled or pushed across the physical hierarchy. The hierarchical scope can either be set in the Virtuoso *Constraint Manager*, or by using the following command in Innovus.

```
setIntegRouteConstraint
[-hierarchicalScope {local local_below local_above local_above_below}]
```

The `local_below` option indicates that the constraint can only be pushed down in a hierarchy, whereas the `local_above_below` option indicates that the particular constraint can be pulled and pushed across the physical hierarchy.

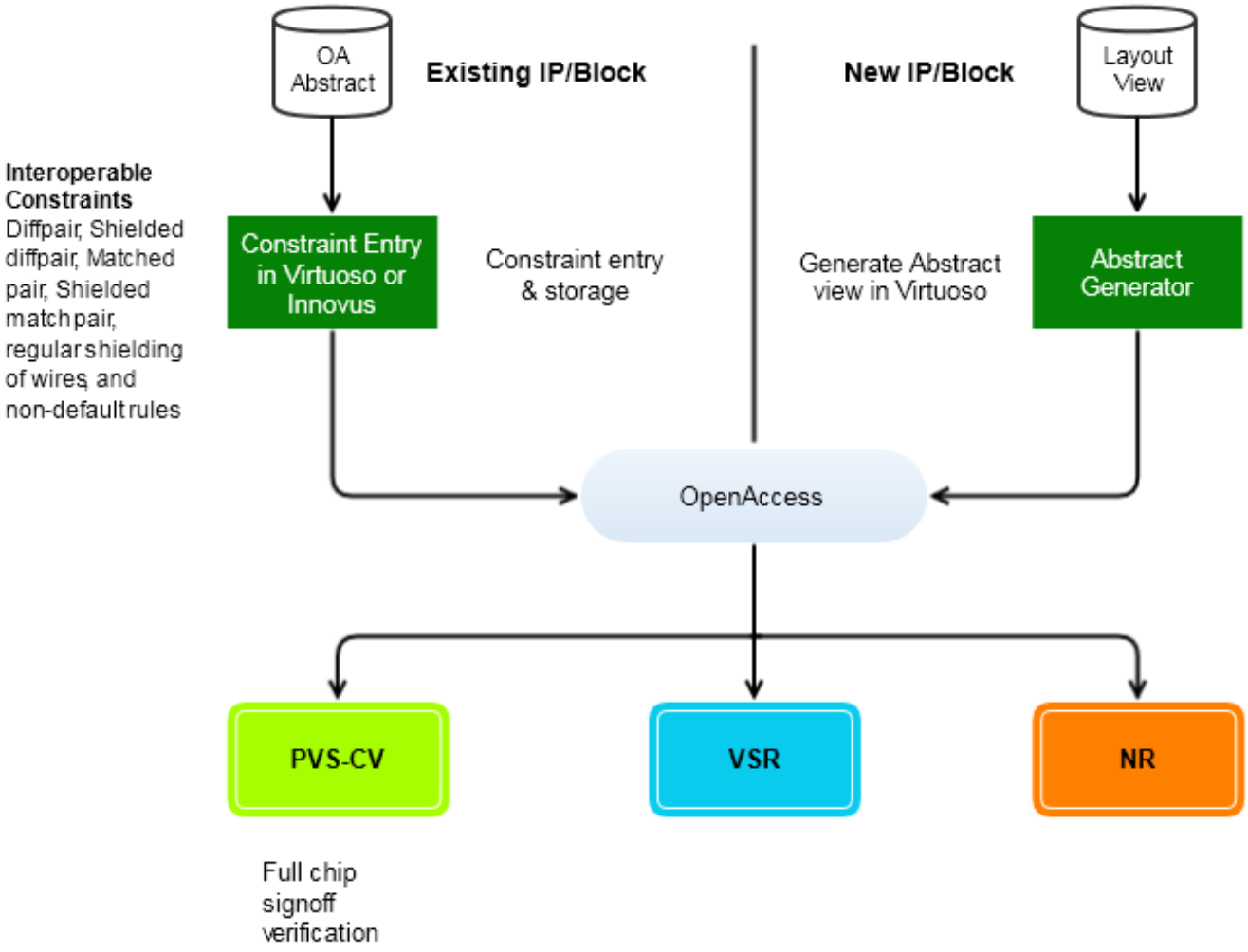
The pull/push of constraints in Virtuoso is done using the Virtuoso *Constraint Manager* .


Virtuoso® Layout Suite GXL Editing: ADTMF4 dtmf_chip skipandanalogattributes
Launch File Edit View Create Verify Connectivity Options Tools Window Optimize Floorplan Place Route Help
cadence
Workspace: Constraints
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:1702.7000 Y:2114.1000 dX: dY: Dist: Cmd:
Layers
AV NV AS NS
Metal1 drawing
All Valid Layers
Used Layers Only
Search
Layer Purpose V S
Met... drawing
Via12 drawing
Met... drawing
Via23 drawing
Met... drawing
Via34 drawing
Layers Objects
Navigator
Default
Search
Name
dtmf_chip
7K Further Nets
DTMF_IN...alogNE)
DTMF_IN...alogNL)
Navigator Property Editor
Push Constraints
Pull Constraints
Compare Constraints in Hierarchy
Constraint Manager
Constraint Checker
Update Layout Constraints
Constraint Propagation
Constraint Editor
Delete
Select All
Deselect All
analogS3I
Process Rule Override
Process Rule Override
Constraint (1)
All Parameter
Name Constr_3
Owner ADTMF4.d
Enabled true
Status not checke
Notes
Shield Net VSS
Shield Net2
Shielding Custom
hierarchicalSc... local, belo
analogNW
cntrlx
pllcir
ram_128x16A
rom_512x16A
ram_256x16A
mouse L: mouseSingleSelectPt() _leiLMBPress()
M: deInstallApp(getCurrentWindow() "VLS-GXL")
R: _vceHiMousePopUp()
1(2)
Cmd:

11.8 Creating/Adding Mixed-Signal Routing Constraints on IP Blocks

As seen in the previous section of this document, design-level constraints can be created using the Virtuoso Constraint Manager, or Innovus's constraint entry environment. In the previous section, we also discussed the fact that constraints can be created hierarchically, thereby enabling users to pull imbedded constraints from sub-blocks or IP's, or push constraints from higher levels in the design down to the lower sub-blocks. As can be seen from the diagram below, enhancement have been made to the abstract generator to create and store constraints in the OpenAccess abstract of an IP/block, which can later be pulled /pushed across the design hierarchy.

Routing Constraints Use Model and Flow



If a particular layout view has specific routing constraints defined, the abstract generator will be able

to store such constraints in the resulting abstract.

To enable this functionality, the following setting needs to be used with the abstract generator:

```
absSetOption("CopyMSRoutingConstraints" true")
```

In order to enable the pushing or pulling of routing constraints in the flow, the addition of the `hierarchicalScope` parameter to the routing constraint is needed. For instance, if the designer of the block/IP wants to allow the intended user of the block/IP to have the ability to push or pull a particular constraint across the hierarchy, then the the following hierarchical scope should be used:

```
hierarchicalScope=local&below&above
```

- `local` is always required.
- `below` tells the design environment that the particular constraint can be pushed down in the hierarchy.
- `above` tells the design environment that the particular constraint can be pulled up in the hierarchy.

For more information, refer to the *Virtuoso® Abstract Generator User Guide*.

11.9 Routing Constraints Interoperability between NanoRoute and VSR

The Virtuoso Space-Based Router (VSR) is used to route nets with differential pairs, match pairs, and complex-shielding constraints. Complex shielding is defined as any shield style other than tandem shields (the default), or those shielding constraints that do not use the default minimum spacing or width.

NanoRoute has been enhanced to recognize mixed-signal routing constraints, so if differential pair, match pair or complex shields are pre-routed by VSR, then NanoRoute will be able to route rest of the nets in a design without having the need to mark pre-routed nets as special routes/fixed routes, or with `skip_routing` attribute. In addition, if NanoRoute sees the simple shielding constraints, `netClass` constraint, or process rule overrides (NDRs), it routes such nets honoring the routing constraints stored in the database.

Note: If VSR is used to route NDRs or simple shields, add the skip_routing attribute on those nets and mark them as `fixed`, when the design is brought into Innovus. The `skip_routing` attribute could be set in Innovus as:

```
setAttribute -net net_name -skip_routing true
```

Failing to assign the `skip_routing` attribute to such pre-routed nets will result in the nets getting deleted when the design goes through placement, Early Global Route, CTS, and NanoRoute stages of the design flow

11.9.1 Getting a List of Nets with skip_routing Attribute

Innovus's `dbGet` command is a very powerful tool that can be used to extract valuable information from the database. In the context of interoperable routing constraints, this command can be used to get all nets having the `skip_routing` or the analog attributes.

```
dbGet [dbGet -p top.nets.skipRouting 1].name (will report all nets with skipRouting
attribute)

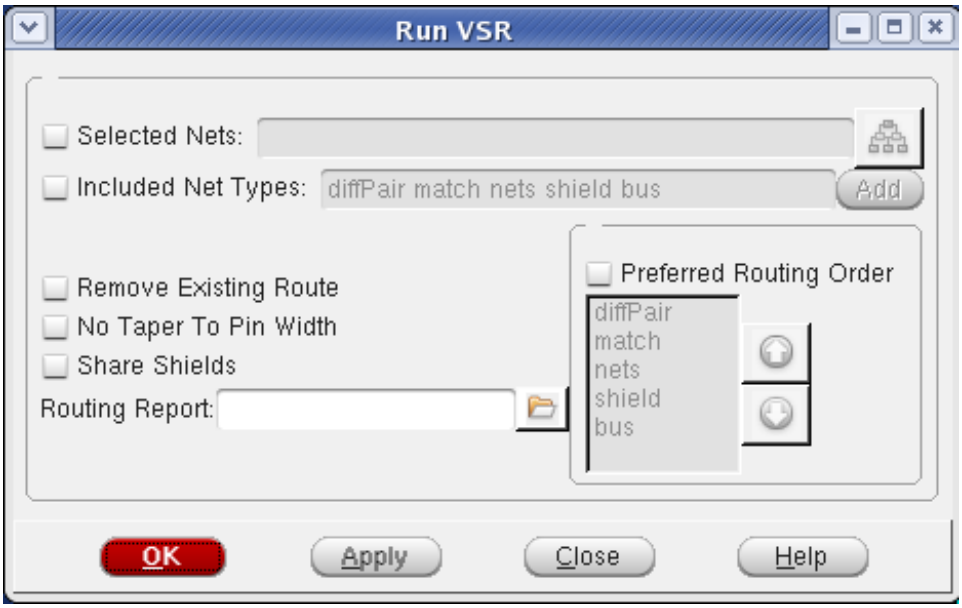
dbGet [dbGet -p top.nets.isAnalog 1].name (will report all nets with analog attribute)
```

To get the list of attributes of a net that one could search for with the `dbGet` command, use:

```
dbGet top.nets.?
```

Note: Virtuoso installation is needed in order to run VSR from Innovus.

The VSR interface can be accessed from Tools menu in Innovus by choosing *Tools - Mixed Signal - Run VSR*.

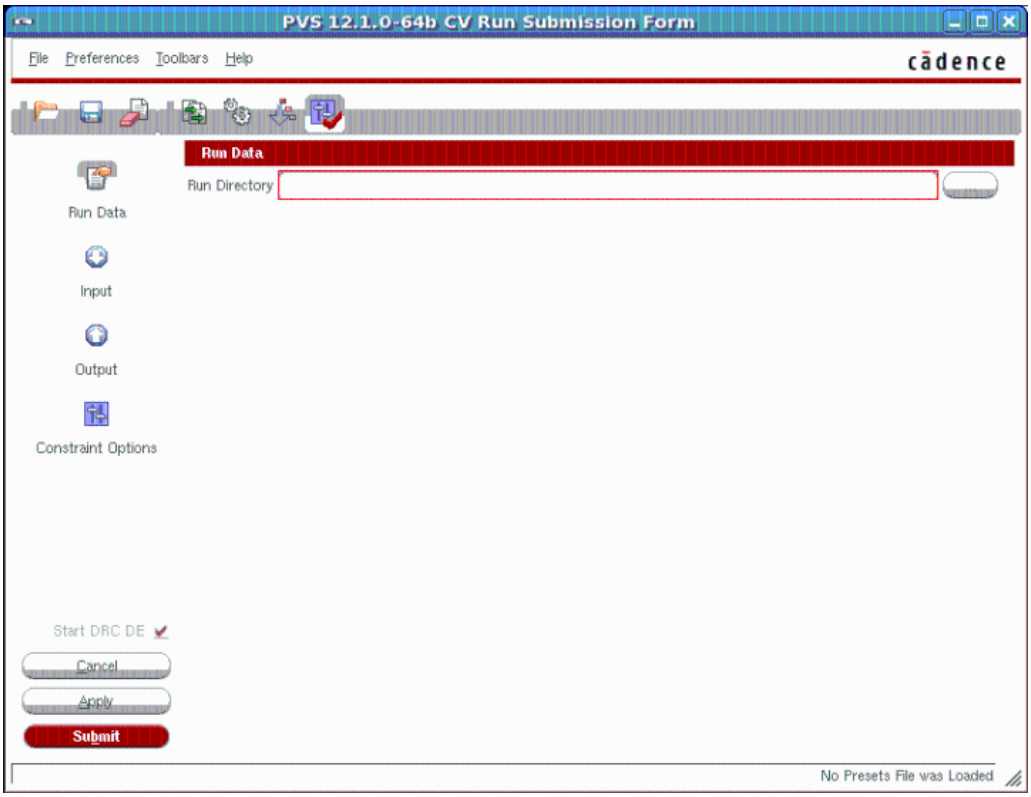


The Design Browser in Innovus can be used to select the nets to be routed. You can also instruct the router to route all nets with certain constraints.

11.9.2 Checking Routing Results Against the Routing Constraints

The PVS Constraints Validation (PVS-CV) tool can be used with Innovus to check and validate mixed-signal constraints in the Innovus design flow. PVS-CV can be used to check all interoperable MS routing constraints entered in Innovus or Virtuoso, against the actual routes created by VSR or NanoRoute. The error markers generated by PVS-CV can then be viewed using tools such as Violation Browser in Innovus.

To enable PVS-CV, select *Run CV* from the *PVS* menu in Innovus.



Note: PVS-CV is enabled only when a 64-bit version of Virtuoso installation exists on your system.

For information on PVS-CV, refer to the following chapters in the *Cadence Physical Verification User Guide:*

- Running PVS-CV
- Appendix H - PVS CV Functionality

11.9.3 Creating Interoperable Library between Innovus and Virtuoso

The compression of OpenAccess database is supported in Innovus and is done by default. However, the current Virtuoso 6.1.5 version does not support a compressed OpenAccess database as the input. So, in order to create an OpenAccess library that can interoperate with Virtuoso 6.1.5, the database compression in Innovus needs to be turned off using the following Innovus command:

```
setOaxMode -compressLevel false
```

Cellviews written to a library will follow the compression level of the library and not the `setOaxMode -compressLevel` setting at the time when the cellview is being created.

11.10 Trying the Constraint Interoperability Environment and the run_vsr Command in Innovus for the First Time

Constraint interoperability between Virtuoso and Innovus is possible only through the OpenAccess database. The creation of an OpenAccess database is divided in two steps, the reference library creation step and the design library creation step. The term “reference library” refers to the interoperable PDK that contains all the technology information needed for Innovus and Virtuoso to work properly.

If the OpenAccess library or the reference library is already available, you could reference it in Innovus’s global variable (`*.globals`) file to load the technology and IP-related information. **Note** - Global variables are the Tcl globals needed to initialize a design. In Innovus, all global variables are stored in the *.globals file when the design is saved using the `saveDesign` command.

If the reference library is not readily available, refer to the Cadence OpenAccess documentation on how to create an OpenAccess PDK. The library creation steps mentioned below are only used to create a quick OpenAccess library from LEF to try the constraint interoperability environment and the run_vsr interface. **PLEASE NOTE THE REFERENCE LIBRARY CREATED BY THIS METHOD SHOULD NOT BE USED IN A PRODUCTION SETTING.**

Method 1: Reference library does not exist, but a non-OA Innovus design is available.

The following flow steps are meant for users who have neither a previous OA-based Innovus database to start with, nor a reference library. This flow can be used for all nodes 16nm and higher.

- Creating the reference library
  Before you run the library creation step, ensure that you have set up the path for Virtuoso correctly. (The compatible Virtuoso release is IC 6.1.6ISR2 or higher and Innovus release is 13.2 or higher.)

  Example

  1. **write_oa_techfile tech.tf**
     After loading the Innovus database in Innovus with lef files, execute the above command. This command will output the Virtuoso tech file. This file would not contain fixed vias and LDRS (LEF Default Route Spec). These will be added in Step 3 below.
  2. **techLoadDump -l -createLib tech_base tech.tf**
     Where:
     `tech_base` specifies the output reference library name.

`tech.tf` specifies the technology file created in Step 1.

After you create a Virtuoso tech file, exit Innovus and execute the above command on a Unix prompt. This command will create an OA-based library using the Virtuoso tech file created in the previous step.

3. **lef2oa -lib tech_pnr -techRefs tech_base -lef tech.lef -pnrLibDataOnly**
   Where:
   `-lib` *`library`* specifies the output reference library name.
   `-techRefs` *`libList`* specifies the list of libraries that contain the master technology databases.
   `-lef` specifies the list of technology and other LEF libraries.
   `-pnrLibDataOnly` indicates that only library-specific place and route information and macro definitions will be imported. All foundry rules for existing layers in the referenced tech are assumed to be correct. All foundry rules defined in the LEF file will be ignored.

   Now execute the above command to create a `tech_pnr` library using the `tech_base` created in the above step and using the technology lefs and other lefs. The reference library `tech_pnr` can now be used for all interoperability purposes between Innovus and Virtuoso.
4. **verilogAnnotate -refLibs tech_pnr -verilog stub.v**
   Where:
   `-refLibs` specifies the OA reference library created in Step 3.
   `-verilog` specifies the stub verilog file that has bus information.
   Execute the above step to annotate bus information to abstract view through stub verilog. As we are just using the flow to try constraint interoperability and the `run_vsr` interface, this step is an optional step.

- Loading the design in Innovus
  1. Set the following variables in the Innovus shell:
     - `set init_pwr_net {`*`power_nets`*`}`
       #(Sets the power nets)
     - `set init_gnd_net {`*`ground_nets`*`}`
       #(Sets the ground nets)
     - `set init_oa_ref_lib {`*`tech_pnr`*`}`
       #(Sets the reference library created in Step 4)

- `set init_verilog {verilog_netlist}`
  #(Sets the verilog netlist)
- `set init_top_cell {top_cell_name}`
  #(Sets the top cell name)
- `init_design`
  #(Updates the above variables into the Innovus DB)

2. Load the design information from floorplan or DEF file using command `loadFPlan filename` or `defIn def_file.`
3. Run `saveDesign -cellview {designLib topCellName viewName}`
   The above command saves the design in OpenAccess mode and can be restored for any future sessions in Innovus or in Virtuoso.

   To restore a previously saved OpenAccess design, run the following command at the Innovus prompt:
   `restoreDesign -cellview {designLib topCellName viewName}`

Method 2: Flow to Load a Previously Implemented Design in Innovus if the Reference Library Already Exists

If the reference library already exists, ignore the library creation step in Method 1 and continue with the rest of the steps:

1. Initialize the variables in Innovus (Reference library `tech_pnr` already available)
   - `set init_pwr_net {power_nets}`
     #(Sets the power nets)
   - `set init_gnd_net {ground_nets}`
     #(Sets the ground nets)
   - `set init_oa_ref_lib {tech_pnr}`
     #(Sets the reference library)
   - `set init_verilog {verilog_netlist}`
     #(Sets the verilog netlist)
   - `set init_top_cell {top_cell_name}`
     #(Sets the top cell name)
   - `init_design`
     #(Updates the above variables into Innovus DB)

2. Load the design information from floorplan or DEF file using command `loadFPlan` *`filename`* or `defIn` *`def_file`*.

12

Working with Buses in OpenAccess

- Settings for Automatic Annotation of Bus Bits During Abstract Generation using Virtuoso Abstract Generator
- The Interface Bit Setting of a Terminal
  - Checking for the Presence of the Interface Bit
  - Running verilogAnnotate

12.1 Settings for Automatic Annotation of Bus Bits During Abstract Generation using Virtuoso Abstract Generator

While using the Abstractor Generator tool for bus terminals, use the following options before running the tool:

```
( absSetOption( "AnnotateBusInAbstract" "true"))
```

You need to run `verilogAnnotate` after running Abstractor Generator (with the above settings) to set the interface bit for each valid logical term in the cells. If you fail to run, you will get the following error while running `saveDesign -cellview`.

```
*ERROR: (IMPOAX-683): Unable to find module terminal "D" for module "JACK". Can not
create Connection Data for this terminal.
```

- If you are running `assembleDesign` on AMS blocks created in Virtuoso, `verilogAnnotate` needs to be run on the layout views (or the view name being used for `assembleDesign`) besides abstract views.
- If you use VLS-XL to generate layouts, set the following and you can ignore bus annotation during abstract generation flow:

  ```
  envSetVal("layoutXL" "createImplicitBusTerminals" 'boolean t)
  ```

12.2 The Interface Bit Setting of a Terminal

To preserve the logical connectivity of a terminal to the corresponding net, the terminal of the block/cell must have the `isInterface` bit set to `true`.

The following examples depict what can happen when the interface bit is not properly set:

Example 1 - isInterface of both the bus and its individual scalar terminals is set to true

Consider a design in which the `isInterface` attribute of both the bus, such as `main_bus<1:0>`, and its individual scalar terminals, such as `main_bus <1>` and `main_bus <0>`, is set to `true`. In this case, you may encounter an error with the following message during the `assembleDesign` step in Innovus:

```
Duplicate Fterms are encountered
```

To resolve the problem, run `verilogAnnoate` to turn off the `isInterface` attribute of both scalar terminals.

Example 2 - Some terminals in the block/cell do not have isInterface set to true during partitioning

Consider a situation in which some of the terminals in the block/cell do not have the `isInterface` bit set to `true` during the partitioning of a design. In this case, these terminals will have issues when the block is being assembled with the top level.

The `isInterface` bit is meant to align with the Verilog modules. So, if an OpenAccess abstract terminal has `isInterface = nil`, it should not appear in the Verilog netlist when running `saveNetlist`, unless the `-includePowerGround` or `-phys` options are used. Terminals with `isInterface = nil` should be connected only to a `USE POWER/GROUND` net, not a regular signal net. Assuming that the Verilog module (and `.lib`) has port, running `verilogAnnotate` will set the bit on the abstract's terminal to `true` instead of `nil`.

To fix this problem, run the `verilogAnnotate` command on the layout view of the cell or recreate the reference libraries with the interface bit set for this terminal.

12.2.1 Checking for the Presence of the Interface Bit

Before moving to the Innovus flow, you may want to check that certain terminals carry these interface bits. To do so, run oaDBChecker and check the log file for INFO such as the following:

```
Checking for status of interface bit for all blocks in the current view...
INFO (CHECKER_DESGN-14): Interface bit on terminal 'cntrlyo<7>' of cell 'dtmf_chip' is
set to false.
```

Note: If the original Virtuoso DB was created using Virtuoso-L, the `isInterface` bit could be set to

`true` on the Bus terminal as well as the scalar terminals. This might be okay with Virtuoso but Verilog does not support this configuration in Innovus. Run `verilogAnnotate` to resolve this issue as mentioned in the previous section.

12.2.2 Running verilogAnnotate

Run `verilogAnnotate` as follows:

```
verilogAnnotate -refLibs libraryList -verilog fileList [Optional_Arguments]
```

Example

```
restoreDesign -cellview {lib cell view}
```

If you have only the top-level design netlist but not the specific cell's netlist, you first need to generate a leaf-only netlist as follows:

```
saveNetlist leafOnly.v -onlyLeafCell
```

Make sure you have write permission to update the OpenAccess file:

```
verilogAnnotate -refLibs library_1 -verilog leafOnly.v

exit
```

#Restart the tool

```
restoreDesign -cellview {lib cell view}

attachTerm ...

saveDesign ...
```

The saved design will have the changes saved.

13

High Frequency Router In Innovus

- Overview
- Invoking NRHF
- Controlling NRHF Routing
- Supported Routing Styles
  - Non-default Routing
  - Shielding
  - Differential Pair
  - Match Length
  - Resistance Match
  - Layer Match
  - Bus Routing
  - Length Control

13.1 Overview

NanoRoute High Frequency Router (NRHF) is a high frequency router introduced in the Innovus 16.1 release to address the custom and high-frequency routing needs of mixed-signal users. Built on the NanoRoute infrastructure, it provides full interoperability between various Cadence tools. NRHF supports a multiple-entry mechanism for routing constraints. Routing constraints can be entered through the Integration Constraint Editor (ICE) in Innovus or through the Constraint Manager in Virtuoso. For constraints entered in Virtuoso, the OpenAccess-based flow should be used.

NRHF supports the following specialty routing features:

- Simple shielding (Parallel shielding)
- Differential pair

- Match length, where a net's length is matched with another net or group of nets
- Layer match
- Resistance match routing, based on the resistance input provided
- Bus routing (also supports split bus)
- Non-default rule (NDR) with any combination of the above constraints
- Length match routing, where a net is routed to match the length specified in the constraint

13.2 Invoking NRHF

Before you invoke NRHF, make sure that the design has routing constraints. Constraints can be added to the design through the following methods:

- Using the Integration Constraint Editor (ICE) in Innovus - Refer to the "Creating Interoperable Routing Constraints Using Innovus" section in the Routing Constraint Interoperability chapter for details on creating interoperable constraints in Innovus.
- Using the Constraint Manager in Virtuoso - Refer to the "Creating/Viewing Interoperable Routing Constraints in Virtuoso" section in the Routing Constraint Interoperability chapter for details.
- Using the `setIntegRouteConstraint` command directly at the Innovus command prompt
- By sourcing `setIntegRouteConstraint` using a file

After ensuring that the design has routing constraints, you can invoke NRHF by using the following command:

```
> routeDesign -highFrequency
```

13.3 Controlling NRHF Routing

Most mixed-signal users do the pre-routes on the top level before or after the placement of standard cell and blocks. Therefore, NRHF can be invoked before or after the placement of standard cells and blocks. By default, NRHF will route all nets that have routing constraints. To route only specific nets, you can use the following set of commands:

```
> setNanoRouteMode -route_selected_net_only true

> routeDesign -highFrequency
```

Note: A net will not be routed by NRHF if no constraints are present and `setNanoRouteMode -route_selected_net_only true` is used. Partially routed nets are also not routed.

You can also control the routing order followed by NRHF by using the following commands:

```
> setNanoRouteMode -route_high_freq_constraint_groups {order bus pair match shield net}

> routeDesign -highFrequency
```

Here, `-route_high_freq_constraint_groups` is used to restrict routing to only the specified constraint groups. If some groups are not in the list, they are not routed. The legal values for constraint groups are `net`, `match`, `bus`, `pair`, and `shield`, with `match` referring to all types of match constraints, such as match length, layer match, and resistance match.

If `order` is specified before the routing style arguments, NRHF honors the order in which the routing style constraint groups are specified. If `order` is not specified, NRHF determines the order.

Regular NanoRoute does not disturb the routes created by NRHF for `bus`, `pair` (diff pair), and `match` constraints. For `net` and `shield` constraints, NRHF might disturb the existing routing. In general, most of the NanoRoute options are honored by NRHF.

By default, NRHF tries to avoid DRC violations during routing. However, it might leave a few DRC violations because of pin access problems. In such situations, you can use the following flow to repair the violations using NanoRoute. This set of commands invokes NanoRoute ECO and does not call the NRHF router:

```
setNanoRouteMode -route_high_freq_search_repair true

routeDesign -highFrequency
```

Note: `verify_drc` cannot be used to check NRHF routes. If you are using an OpenAccess-based flow, you can use the Physical Verification System Constraint Validation (PVS-CV) tool to check NRHF routes against constraints and flag any issues.

13.4 Supported Routing Styles

NRHF supports the following routing styles:

- Non-default Routing
- Shielding
- Differential Pair
- Match Length
- Resistance Match
- Layer Match
- Bus Routing
- Length Control

These routing styles are described below.

13.4.1 Non-default Routing

NRHF supports regular Non-default Routing (NDR) just like traditional NanoRoute. You can set a `netClass` constraint type with NDR through ICE. You can use an existing NDR or create a new design NDR and attach it to the net. You can also add layer control to the NDR rule. An NDR can also be added to other constraint types, such as differential pair, match length, bus, layer match and resistance match. If an NDR is not attached to any net, default rules are used by the router.

Example:

```
setIntegRouteConstraint -type netClass -net {DTMF_INST/digsynca} -rule NDR2 -topLayer
Metal5 -bottomLayer Metal1
```

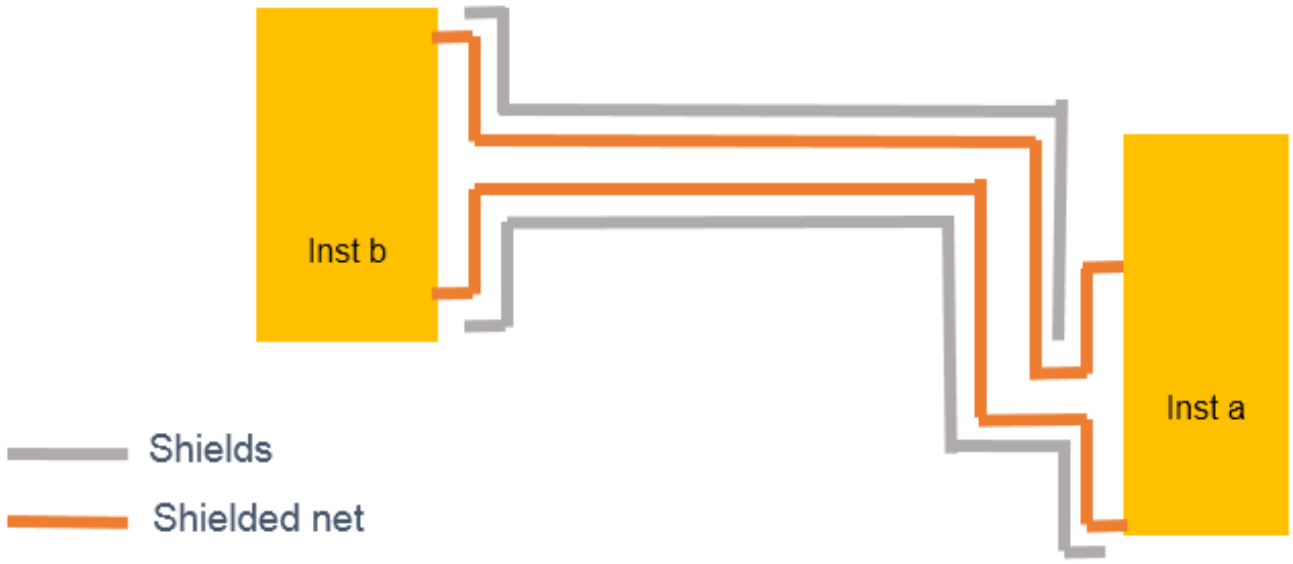
```
setIntegRouteConstraint -type netClass -net {DTMF_INST/digsync3 DTMF_INST/digsync4} -
rule DOUBLE_WIDTH -topLayer Metal5 -bottomLayer Metal1 -shieldNet VSS -connectSupply
anyPoint -shieldType side
```

13.4.2 Shielding

For shielding, NRHF calls the NanoRoute code. Shielding can be used along with other styles of routing, such as differential pair, length match, layer match, resistance match, bus routing and NDR. Currently, only side shield is supported in NRHF and shield width and shield gap are not supported. The default shield width is used, and the default gap is translated to tracks. NRHF snaps the routes to the nearest track. Shields can be floating or can be connected to any point of the power or ground. Shields are not guaranteed to cover the shielded net completely.

Example:

```
setIntegRouteConstraint -type diffPair -net {DTMF_INST/dout[15] DTMF_INST/dout[14]} -
rule wide -topLayer Metal6 -bottomLayer Metal1 -shieldNet VSS -connectSupply anyPoint -
shieldType side -tolerance 20.0
```



Note: Currently, the `shield` constraint type alone is not supported by NRHF through the `setIntegRouteConstraint` command. If you want NRHF to route just the `shield` constraint, add a `netClass` or `diffPair` constraint and specify the shield information as follows:

```
setIntegRouteConstraint -type netClass -net {neta netb} -shieldNet VSS
setIntegRouteConstraint -type diffPair -net {neta netb} -shieldNet VSS
```

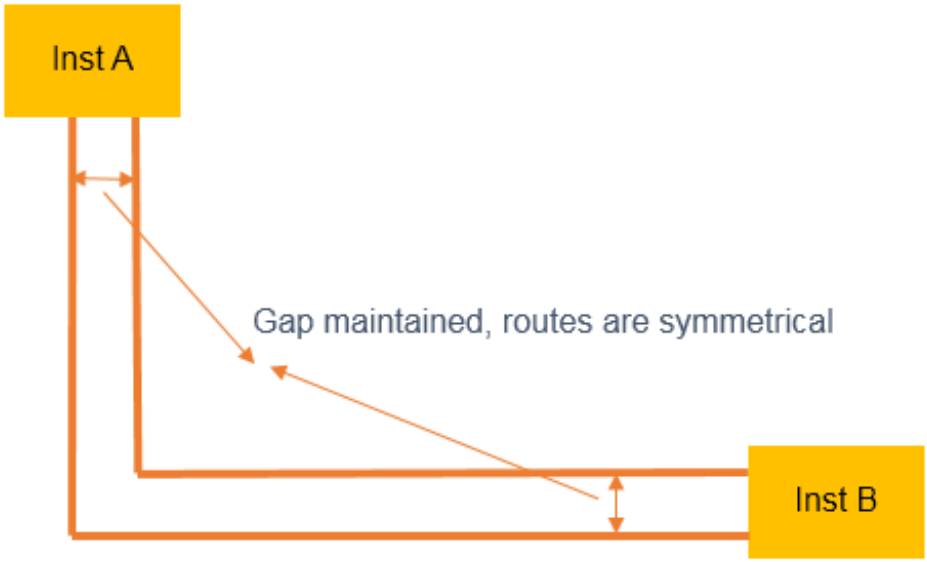
13.4.3 Differential Pair

Differential pair nets are symmetrical nets that are constrained by width and spacing. Width and spacing can be provided through an NDR. Additionally, NRHF also supports the use of a minimum cut, if specified through an NDR. If an NDR is not used, you can also explicitly specify the differential pair gap in the constraint. NRHF would also honor the top/bottom layer control if specified in the constraint.

If the differential pair needs to be shielded, you can specify the shield constraint along with the differential pair constraint. NRHF currently does not support shieldWidth and shieldGap. It tries to match the routes as close as possible.

Example

```
setIntegRouteConstraint -type diffPair -net {DTMF_INST/digsync1 DTMF_INST/digsync2} -
rule NDR2 -layerGap {Metal1 0.1 Metal2 0.1 Metal3 0.1 Metal4 0.1 Metal5 0.1 Metal6 0.1}
-shieldWidth {Metal1 1 Metal2 1 Metal3 1 Metal4 1 Metal5 1 Metal6 1} -shieldGap {Metal1
1 Metal2 1 Metal3 1 Metal4 1 Metal5 1 Metal6 1} -topLayer Metal5 -bottomLayer Metal1 -
shieldNet VSS -connectSupply anyPoint -shieldType side -tolerance 5
```

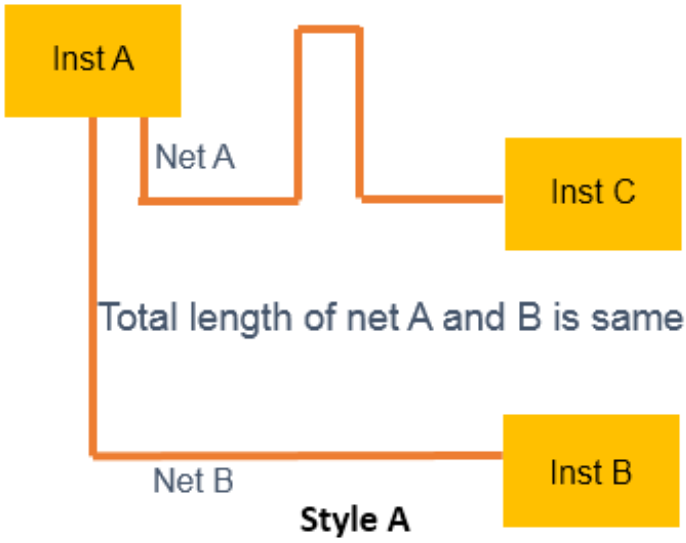


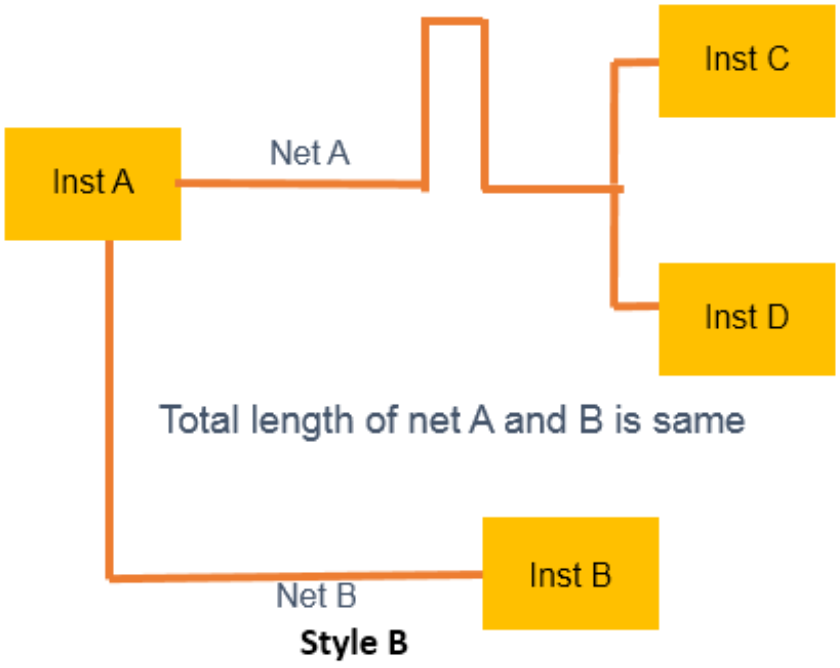
Differential Pair Routing

13.4.4 Match Length

If a match length constraint is specified, NRHF matches the output pin to input pin path length. NRHF also supports multi sink. Trombones or accordions are created to match lengths between the nets. NRHF matches to the requested value only if all the nets can be matched to that length. If not, it matches all nets to the length of the longest net.

Note: The router does not try to make the layers match.





13.4.5 Resistance Match

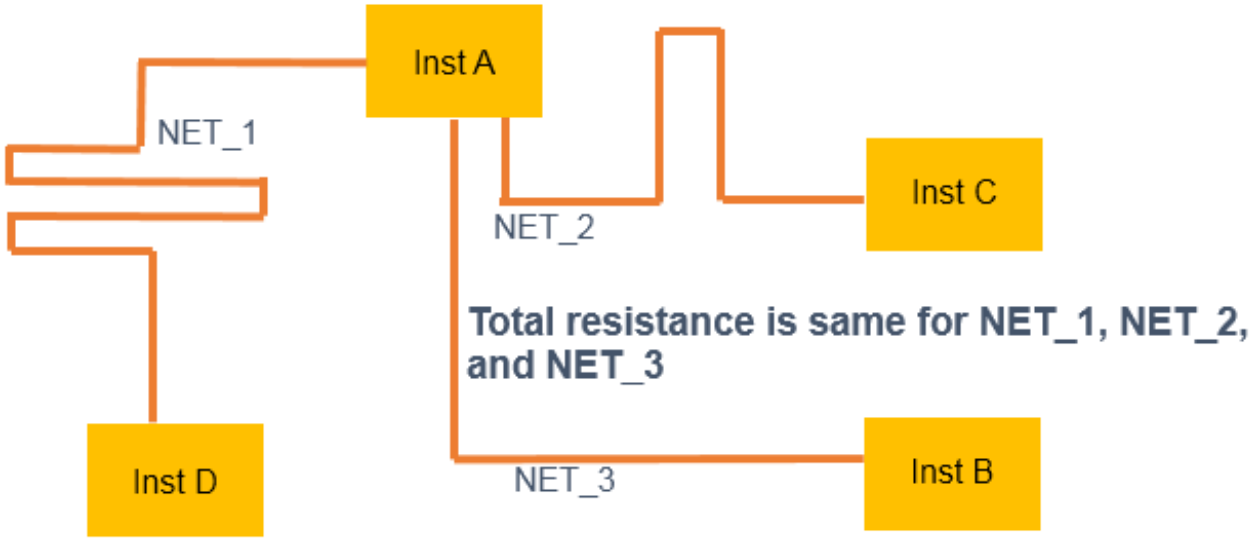
The router behavior for resistance match constraints is very similar to that of the length matching algorithm, except that instead of matching the length, the router uses the resistance value in the constraint to match the nets. The router calculates the resistance of each wire segment, including the vias, to calculate the resistance of a net. For multi-fanout nets, the effective resistance is calculated and used to match other nets. Basically, the router tries to match the output pin to input pin path resistance. Currently, the router does not alter the route width but tries to create the accordion or trombone topology to match resistance.

Notes:

- NRHF uses the layer/cut resistance values based on what Innovus database has either from LEF or the timing library. In the 16.1 release, you can specify the resistance values for each layer/cut with the constraint.
- The router does not try to make the layers match.
- Starting 17.1, NRHF has been enhanced to use the Quantus database for accurate resistance calculation.

Example

```
setIntegRouteConstraint -type matchLength -net {NET_1 NET_2} -rule default -
hierarchicalScope local -topLayer Metal7 -bottomLayer Metal5 -tolerance 20 -matchStyle
accordion -minRes 0.3 -maxRes 0.4
```



13.4.6 Layer Match

When Layer Match is enabled, the router will try to match the layers. In layer matching, all the nets are divided into multi-level from output pin to input pin. Each level contains the segment from pin/steiner to the next pin/steiner. For layer match routing, the router assumes that all output pins have the same number of levels to all input pins.

Example

```
setIntegRouteConstraint -type matchLength -name
msMatchedPair_3_DTMF_INST/digsyncbDTMF_INST/digsyncc -tolerance 5 -matchStyle accordion
-layerMatch -rule NDR2 -topLayer Metal5 -net {DTMF_INST/digsyncb DTMF_INST/digsyncc
DTMF_INST/digsyncd}
```

13.4.7 Bus Routing

NRHF has a very robust bus routing algorithm. The router supports a simple bus, with bus pins facing each other, as well as more complex bus requirements. For complex bus requirements, the router uses different types of patterns like z, inverted z and so on. By default, the router clusters all the pins and routes with the default gap (pin gap or given bus gap). You can modify this behavior by using `-accessStyle straight` option through the ICE commands. See **Example 4** below.

In addition to the NDR specification, you can control the routing using top/bottom layer control. You can also shield the bus routes. If the bus bits have enough spacing and will not cause any DRC violation, the router interleaves the shields. Otherwise, the bus will be only shielded on the sides. If you need a specific topology for each bus, you can specify bus guides and NRHF would honor them. If the pins are not the same width as the routes, the router will automatically taper to the pins.

By default, NRHF shields the bus only on the sides and does not interleave. To enable interleaving, use the `interleaved` option with the `-shieldType` option as shown in **Example 2** below.

For some designs that need the control of corner style, you can choose between crossover or river routing by using the `-cornerStyle crossover` or -cornerStyle river options for each bus specified in the ICE constraint. See **Example 3** below. The default style is `crossover`.

By default, NRHF chooses its own path for routing, based on the built-in heuristics. If you want to override this, you can create a bus guide prior to calling NRHF and the router will honor the guide. Buses are generally routed in the preferred direction; if you want to have planar bus routing, a bus guide is a must.

Example 1

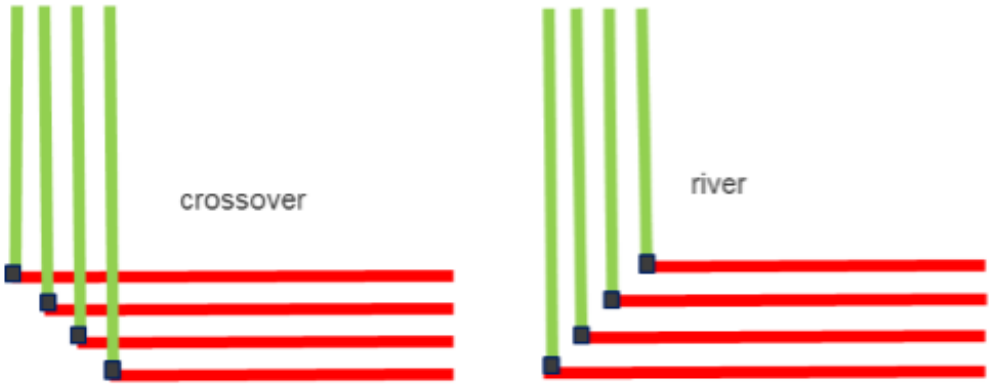
```
setIntegRouteConstraint -type bus -net {DTMF_INST/dxout[0] DTMF_INST/dxout[15]
DTMF_INST/dxout[8] DTMF_INST/dxout[2] DTMF_INST/dxout[5] DTMF_INST/dxout[6]
DTMF_INST/dxout[11] DTMF_INST/dxout[7] DTMF_INST/dxout[3] DTMF_INST/dxout[9]
DTMF_INST/dxout[14] DTMF_INST/dxout[12] DTMF_INST/dxout[1] DTMF_INST/dxout[13]
DTMF_INST/dxout[4] DTMF_INST/dxout[10]} -rule default -hierarchicalScope local -
topLayer Metal6 -bottomLayer Metal1 -tolerance 20.0
```

Example 2

```
setIntegRouteConstraint -type bus -net {DTMF_INST/dxout[0]
DTMF_INST/dxout[15]DTMF_INST/dxout[8]DTMF_INST/dxout[2]} -rule default -layerGap {
Metal3 1 Metal4 1 Metal5 1 } -hierarchicalScope local -topLayer Metal5 -bottomLayer
Metal3 -shieldNet avdd_h -shieldType interleaved -minBusBit 1
```

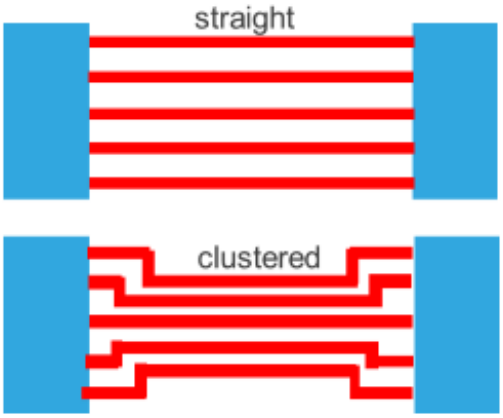
Example 3

```
setIntegRouteConstraint -type bus -net {DTMF_INST/dxout[0]
DTMF_INST/dxout[15]DTMF_INST/dxout[8]DTMF_INST/dxout[2]} -rule default -layerGap {
Metal2 2.5 Metal3 2.5 } -hierarchicalScope local -topLayer Metal3 -bottomLayer Metal2 -
cornerStyle river
```



Example 4

```
setIntegRouteConstraint -type bus -net {itc2\[0\] itc2\[1\] itc2\[3\]itc2\[4\]..} -rule
default -layerGap { Metal6 2.5 Metal7 2.5 } -hierarchicalScope local -topLayer Metal7 -
bottomLayer Metal6 -minBusBit 2 -accessStyle straight
```



13.4.8 Length Control

NRHF also enables you to restrict the length of nets within a specified range. You can specify the minimum and maximum lengths in um to be applied to the given nets through the `setIntegRouteConstraint` command. The router tries to restrict the routing within the specified minimum and maximum lengths. This applies to multi-fanout nets as well. In multi-fanout nets, NRHF honors the given length for each source to sink.

Example

```
setIntegRouteConstraint -type matchLength -net {NET_1 NET_2} -rule default -
hierarchicalScope local -topLayer Metal7 -bottomLayer Metal4 -tolerance 20 -matchStyle
accordion -minLength 78 -maxLength 100
```

14

Working with Vias in OpenAccess

- Overview
- Types of Via Definitions
  - Custom Vias or customViaDefs
  - VIARULE or standardViaDefs
  - standardViaVariants
- How Tools Find and Use Vias
  - generateVias
  - Summary Table
  - cdsFixedVias
- Examples
  - Setting the LDRS in Virtuoso Layout Suite
  - Checking Vias in the Tech File
- Directing the Router to Use Only the Vias in the ITDB in Innovus

14.1 Overview

The technology libraries of an OpenAccess database can comprise one or multiple files. In other words, the technology information (layers, layerRules, vias, constraintGroups) about a specific process and metal stack can be spread across multiple tech files. This approach is called an Incremental Technology Database (ITDB). The incremental approach is preferred to a single tech file because of the following advantages:

- The libraries can be made more specific. The baseTech file can contain the process information, design rules, and the data specific to Virtuoso users. Another tech file can contain the technology data specific to a standard cell library. This tech file is primarily used by Innovus™ Implementation System.

- In an Incremental Technology Database, it is possible to make changes in one library without having to modify the baseTech library. For example, a new via or constraint group can be added to a library for Innovus.
- Different tech files can be maintained by different teams. For example, one can be maintained by the digital team and another by the analog team.

This chapter focuses on different methods for defining vias in the OpenAccess libraries for use in interoperable flows by both Virtuoso and Innovus. These include customViaDefs, standardViaDefs, and standardViaVariants.

14.2 Types of Via Definitions

The vias found in a traditional LEF file are fixed or custom vias and used by NanoRoute, the router in Innovus, to implement signal nets. A LEF file can also contain VIARULE vias, which are primarily used by Innovus's special route (sroute) router to create the via arrays used in power routing. When a LEF file is converted into an OpenAccess tech file, the fixed or custom vias are converted to customViaDefs and VIARULE vias are converted to standardViaDefs in the tech file.

A standardViaVariant is an OpenAccess-specific method to create a fixed via from a standardViaDef based on user specifications.

14.2.1 Custom Vias or customViaDefs

In the LEF format, a fixed or custom via would read as follows:

```
VIA M11_M10_1x2_VH_S DEFAULT

        LAYER Metal10 ;

                RECT -0.105 -0.49 0.105 0.13 ;

        LAYER Via10 ;

                RECT -0.09 -0.45 0.09 -0.27 ;

                RECT -0.09 -0.09 0.09 0.09 ;

        LAYER Metal11 ;

                RECT -0.14 -0.48 0.14 0.12 ;

END M11_M10_1x2_VH_S
```

In the OpenAccess format, the via would have a cellview that is created during `lef2oa` translation of the LEF file containing the vias. Therefore, the OpenAccess tech file only points the software to the cellview. Notice there are no dimensions in the customViaDefs in the tech file.

```
customViaDefs(

 ;( viaDefName libName cellName viewName layer1 layer2 resistancePerCut)

 ;( ---------- ------- -------- -------- ------ ------ ----------------)

   ( M11_M10_1x2_VH_S  gsclib045 M11_M10_1x2_VH_S via Metal10 Metal11 0.0)

   ( M11_M10_1x2_VH_N  gsclib045 M11_M10_1x2_VH_N via Metal10 Metal11 0.0)

   ( M11_M10_2x1_VH_W  gsclib045 M11_M10_2x1_VH_W via Metal10 Metal11 0.0)
```

```
   ( M11_M10_2x1_VH_E  gsclib045 M11_M10_2x1_VH_E via Metal10 Metal11 0.0)
   ( M11_M10_M_SH  gsclib045 M11_M10_M_SH via Metal10 Metal11 0.0)
   ( M11_M10_M_NH  gsclib045 M11_M10_M_NH via Metal10 Metal11 0.0)
)
```

⚠ cdsFixedVias
cdsFixedVia is a new object supported in OpenAccess that is fully interoperable between Virtuoso and Innovus. cdsFixedVias provide a way of storing custom vias in the OpenAccess flow.

For more information, see cdsFixedVias.

14.2.2 VIARULE or standardViaDefs

The VIARULE vias from the LEF file are converted to standardViaDefs in the OpenAccess tech file. standardViaDefs are the foundation for parameterized vias used by routers such as Virtuoso Space-based Router (VSR). standardViaDefs are basically the rules or parameters that tell VSR how to construct a DRC-correct via. It could be a single-cut square via, rectangular via, or multiple flavors of double-cut vias.

The LEF format:

```
VIARULE M2_M1 GENERATE DEFAULT
        LAYER Metal1 ;
                ENCLOSURE 0.005 0.03 ;
        LAYER Metal2 ;
                ENCLOSURE 0.005 0.03 ;
        LAYER Via1 ;
                RECT -0.035 -0.035 0.035 0.035 ;
                SPACING 0.14 BY 0.14 ;
                RESISTANCE 5 ;
END M2_M1
```

The OpenAccess format:

```
 standardViaDefs(
 ;( viaDefName  layer1     layer2     (cutLayer cutWidth cutHeight [resistancePerCut])
```

```
 ;    (cutRows   cutCol     (cutSpace))
 ;    (layer1Enc) (layer2Enc)    (layer1Offset)  (layer2Offset)  (origOffset)
 ;    [implant1  (implant1Enc)  [implant2  (implant2Enc) [well/substrate]]])
 ;( ------------------------------------------------------------------------ )
   ( M2_M1            Metal1      Metal2         ("Via1" 0.07 0.07 5.0)
      (1 1 (0.07 0.07))
      (0.005 0.03)    (0.005 0.03)    (0.0 0.0)  (0.0 0.0)  (0.0 0.0)
   )
)
```

14.2.3 standardViaVariants

A standardViaVariant is a fixed via that is created from a standardViaDef based on user specifications. standardViaVariants are useful in situations where the software, whether in Virtuoso or Innovus, does not create by default the required type of via. A standardViaVariant is similar to a customViaDef in that both are based on user specifications. However, unlike a customViaDef, a standardViaVariant is not added to the technology library (no cellview) and therefore does not need to be maintained like a customViaDef. When a standardViaVariant is employed, its parameters are stored in the design library and not in the technology library.

```
standardViaVariants(
;( viaVariantName viaDefName (cutLayer cutWidth cutHeight)
;    (cutRows   cutCol  (cutSpace))
;    (layer1Enc) (layer2Enc)    (layer1Offset)  (layer2Offset)  (origOffset)
;    (implant1Enc) (implant2Enc) (cut_pattern) )
;( ------------------------------------------------------------------------ )
;this is a shifted 2-cut via to increase 2-cut ratio for a few specific std-cells pins
   ( M2_M1_shifted_DH_x        M2_M1       ("via" 0.26 0.26)
     (2 1 (0.26 0.26))
     (0.04 0.06) (0.11 0.11)     (0.0 0.0)       (0.0 0.0)       (0.17 0.0)
     (0.0 0.0)   (0.0 0.0)       ((1))
   )
```

```
  ( M2_M1_shifted_DH_x19         M2_M1    ("via" 0.26 0.26)
    (2 1 (0.26 0.26))
    (0.04 0.06) (0.11 0.11)      (0.0 0.0)        (0.0 0.0)        (0.17 0.12)
    (0.0 0.0)    (0.0 0.0)       ((1))
  )
  ( M2_M1_shifted_DH_x19ny         M2_M1        ("via" 0.26 0.26)
    (2 1 (0.26 0.26))
    (0.04 0.06) (0.11 0.11)      (0.0 0.0)        (0.0 0.0)        (0.17 -0.12)
    (0.0 0.0)    (0.0 0.0)       ((1))
  )
);standardViaVariants
```

Innovus provides the ability to create a set of vias based on standardViaDefs. This potentially eliminates the need for customViaDefs and the overhead they can incur. Some design teams do not like to maintain custom vias and would like the tools to employ only the vias that enhance routability. In Innovus, you can create a set of vias to be used in that session by using the following commands:

```
setGenerateViaMode -auto true

generateVias
```

`setGenerateViaMode -auto true` is recommended so that you do not have to issue the `genereteVias` command after the design is initialized for each session. If you are debugging or testing out standardViaDefs, you can turn off automatic via generation and execute the following command right after the design is initialized:

```
generateVias
```

14.3 How Tools Find and Use Vias

It is important to know how the Cadence software finds and utilizes some or all of the above vias. ConstraintGroups (CGs) in the tech file, such as the LEFDefaultRouteSpec (LDRS), guide the tools in determining which vias to use.

Note: You can use the following command to specify the specific constraint group you want to use in case of ITDB:

```
set init_oa_default_rule {LEFDefaultRouteSpec_gpdk045}
```

The following OpenAccess tech file example contains a `validVia` list. In this example, the library specifies a combination of standardViaDefs and customViaDefs to be available to Innovus. The list of standardViaDefs is optional.

```
constraintGroups(

;( group       [override] )

;( -----       ---------- )

 ( "LEFDefaultRouteSpec_gsclib045"     nil    "LEFDefaultRouteSpec"

interconnect(

     ( validLayers   (Metal1  Metal2  Metal3  Metal4  Metal5  Metal6  Metal7  Metal8
Metal9  Metal10  Metal11  ) )

     ( validVias     (M2_M1  M3_M2  M4_M3  M5_M4  M6_M5  M7_M6  M8_M7  M9_M8  M10_M9
M11_M10  M2_M1_HV  M2_M1_VV  M2_M1_VH  M2_M1_HH  M2_M1_2x1_HV_E  M2_M1_2x1_HV_W
M2_M1_1x2_HV_N  M2_M1_1x2_HV_S  M3_M2_VH  M3_M2_HH  M3_M2_HV  M3_M2_VV  M3_M2_M_NH
M3_M2_M_SH  M3_M2_2x1_VH_E  M3_M2_2x1_VH_W  M3_M2_1x2_VH_N  M3_M2_1x2_VH_S  M4_M3_HV
M4_M3_VV  M4_M3_VH  M4_M3_HH  M4_M3_M_EV  M4_M3_M_WV  M4_M3_2x1_HV_E  M4_M3_2x1_HV_W
M4_M3_1x2_HV_N  M4_M3_1x2_HV_S  M5_M4_VH  M5_M4_HH  M5_M4_HV  M5_M4_VV  M5_M4_M_NH
M5_M4_M_SH  M5_M4_2x1_VH_E  M5_M4_2x1_VH_W  M5_M4_1x2_VH_N  M5_M4_1x2_VH_S  M6_M5_HV
M6_M5_VV  M6_M5_VH  M6_M5_HH  M6_M5_M_EV  M6_M5_M_WV  M6_M5_2x1_HV_E  M6_M5_2x1_HV_W
M6_M5_1x2_HV_N  M6_M5_1x2_HV_S  M7_M6_VH  M7_M6_HH  M7_M6_HV  M7_M6_VV  M7_M6_M_NH
M7_M6_M_SH  M7_M6_2x1_VH_E  M7_M6_2x1_VH_W  M7_M6_1x2_VH_N  M7_M6_1x2_VH_S  M8_M7_HV
M8_M7_VV  M8_M7_VH  M8_M7_HH  M8_M7_M_EV  M8_M7_M_WV  M8_M7_2x1_HV_E  M8_M7_2x1_HV_W
M8_M7_1x2_HV_N  M8_M7_1x2_HV_S  M9_M8_VH  M9_M8_HH  M9_M8_HV  M9_M8_VV  M9_M8_M_NH
M9_M8_M_SH  M9_M8_2x1_VH_E  M9_M8_2x1_VH_W  M9_M8_1x2_VH_N  M9_M8_1x2_VH_S  M10_M9_HV
M10_M9_VV  M10_M9_VH  M10_M9_HH  M10_M9_2x1_HV_E  M10_M9_2x1_HV_W  M10_M9_1x2_HV_N
M10_M9_1x2_HV_S  M11_M10_VH  M11_M10_HH  M11_M10_HV  M11_M10_VV  M11_M10_M_NH
M11_M10_M_SH  M11_M10_2x1_VH_E  M11_M10_2x1_VH_W  M11_M10_1x2_VH_N  M11_M10_1x2_VH_S  )
)

    ) ;interconnect
```

```
) ;LEFDefaultRouteSpec_gpdk045
```

- If you want the software to use only standardViaDefs from the tech file, then the names of the standardViaDefs are optional. Do not add the customViaDefs to the LDRS. If no standardViaDefs are listed, the software will determine the correct design rules from the tech file and build vias accordingly.
    - If a standardViaDef name is specified in the list, any standardViaVariant that is based on the specific standardViaDef that exists in the tech file will be generated and marked as a DEFAULT via in Innovus, which allows them to be used by NanoRoute. This could lead to confusion, so always leave the list empty unless you have one or both of the following two scenarios.
        1. If the library uses customViaDefs, the names of all the customViaDefs you want the software to consider, must be listed in the validVia list.
        2. For standardViaVariants, you would also need to list the name of each variant you would like the software to consider during routing.
- You can verify the vias that Innovus will use by writing out a technology lef from Innovus using the `dumpOutVias` command after the design is initialized and via generation is on. All usable vias in the LEF file that is written out will have the DEFAULT keyword.

Notice that the LDRS has a validVia list. Here are a few guidelines for creating the validVia list:

- If the library only uses standardViaDefs, there is no need to add the names of the customViaDefs to the LDRS. The software will determine the correct standardViaDefs when the tech file is read.
- If the library uses customViaDefs, the names of all the customViaDefs you want the software to consider must be listed in the validVia list
- For standardViaVariants, you must list the standardViaDef name the variant is based on or the name of the variant itself for the via to be used by Innovus.

You can verify the vias that Innovus will use by writing out a technology lef from Innovus after the design is initialized and via generation is on. All usable vias will have the DEFAULT keyword.

Note: All vias in the tech file will be written out of Innovus into the LEF file. Only those with the DEFAULT keyword will be used by NanoRoute.

14.3.1 generateVias

The number of vias created by `generateVias` varies depending on the process technology used. The more metal layers in a process stack, the more vias you will get. The software will attempt to create as many vias as NanoRoute would use. These include single-cut, rotated single-cut, double-cut, and rotated double-cut vias and extensions in the horizontal and vertical directions. However, at this time the software does not look at the standard cell library for pin access. In some libraries, the pins on certain standard cells are off the routing grid and/or have metal shapes that are both on grid and off grid. This becomes important when trying to swap in double-cut vias because there might not be enough room to put dcut vias on adjacent pins or offgrid pins that come close to on grid routes. This is typically where a customViaDef comes in handy. A standardViaVariant can also be used for this purpose. Again, a standarViaVariant does not have the tech file overhead of a customVia.

The software will generate a set of vias based on standardViaDefs and standardViaVariants, and sometimes even customViaDefs. If a customViaDef in the tech file has the same geometry as a via that `generateVias` would normally create, the software simply maps the customViaDef into the database as opposed to creating a new via. Even if the customViaDef is not in the LDRS, the software will still map the customViaDef under the above circumstances.

If you define a standardViaVariant that has the same geometry as a via that `generateVias` would normally create, then the via added to the design library will have the name of the standardViaVariant and not the default name of TECH_RULE_via.

For the latest syntax of `generateVias` command and `setGenerateViaMode` commands, use the `-help` parameter of these commands at the Innovus console as follows:

```
innovus 28> generateVias -help

innovus 29> setGenerateViaMode -help
```

For more information on these commands, refer to the *Innovus Text Command Reference* or use the `man` command at the Innovus console:

```
innovus 30> man generateVias

innovus 31> man setGenerateViaMode
```

14.3.2 Summary Table

The following table summarizes what to include in the valid via list for different via definition combinations in LDRS:

ViaType in LDRS	Valid Via List	generateVias in Innovus	VSR Behavior
customViaDefs only	List all customViaDefs you want NanoRoute to consider	Not used	Only customVias will be available if you choose the same LDRS
standardViaDefs only	standardViaDef names are optional. Cadence recommends leaving the list empty	Turn on Auto Via Generation	When you choose the same LDRS as Innovus, similar parameterized via
standardViaDefs and standardViaVariants	List only the viaVariants you want NanoRoute to consider, `generateVias` will still create a full set of parameterized vias in addition to the variant	Turn on Auto Via Generation	VSR will work in same way. It will generate the ViaVariants and any additional standardOaVia it can based on the standardViaDefs it finds in the OpenAccess tech file
standardViaDefs and customViaDefs	standardViaDef names are optional. List all customViaDefs	Turn on Auto Via Generation	standardViaDef names are optional, list all customViaDefs
standardViaDefs, customViaDefs and sstandardViaVariants	List only customViaDefs and standardViaVariants	Turn on Auto Via Generation	VSR will work in a similar way. It will use the listed standardOaViaVariant, customViaDef and create standardOaVias based on the standardViaDefs found in the tech file

If you want VSR to use custom, user-specified vias, set the `validRoutingVias` constraint through a design-level Process Rule Overrides (PRO) constraint and then confirm the settings in the Via Configuration form within the Wire Assistant.

14.3.3 cdsFixedVias

In advanced process nodes with complex DRC rules, NanoRoute must generate new vias that cannot be expressed as standard vias. In the past, such vias were saved to the design library as custom vias, which required the design library to be created in reference tech mode and made it somewhat difficult for users to copy the design from one library to another. There was also a need to implement a more flexible model for interoperating between Virtuoso and Innovus for any vias that NanoRoute would create in future technology nodes.

The cdsFixedVia methodology has been implemented in Innovus 22.13 and ICADVM20.1ISR24 to make the mixed signal interoperable flow better prepared for the challenges of routing complex designs in more advanced process nodes. cdsFixedVia is a new object supported in OpenAccess that is fully interoperable between Virtuoso and Innovus. Take the following steps to enable this functionality in the flow:

1. Add cdsFixedViaDef to existing or new Rapid MSOA PDKs. The cdsFixedViaDef is the general definition of these vias, which will be used by the OpenAccess flow in Innovus to express any custom vias created during routing. The cdsFixedViaDef is added to the `viaDef` section of the technology file. A cdsFixedViaDef must be defined in either the ITDB section of the Rapid MSOA PDK or the base tech for each MetalN and MetalN+1 pair. This is similar to the existing requirement for oaStdVias. Below is an example of such entries in the Rapid MSOA PDK.

```
cdsFixedViaDefs(
; (t_viaDefName
;   (layers
;    ** Base Layers **
;     (layer1 tx_layer1)
;     (layer2 tx_layer2)
;     (cutLayer tx_cutLayer)
;   )
; )
; ( ------------------------------------------------------------------------ )
  (M1_M2_VIA12_cdsFixedViaDef
    (layers
      (layer1           M1      )
```

```
      (layer2            M2       )
      (cutLayer        VIA12     )
    )
  )
  (M2_M3_VIA23_cdsFixedViaDef
    (layers
      (layer1            M2       )
      (layer2            M3       )
      (cutLayer          VIA23     )
    )
  )
)
```

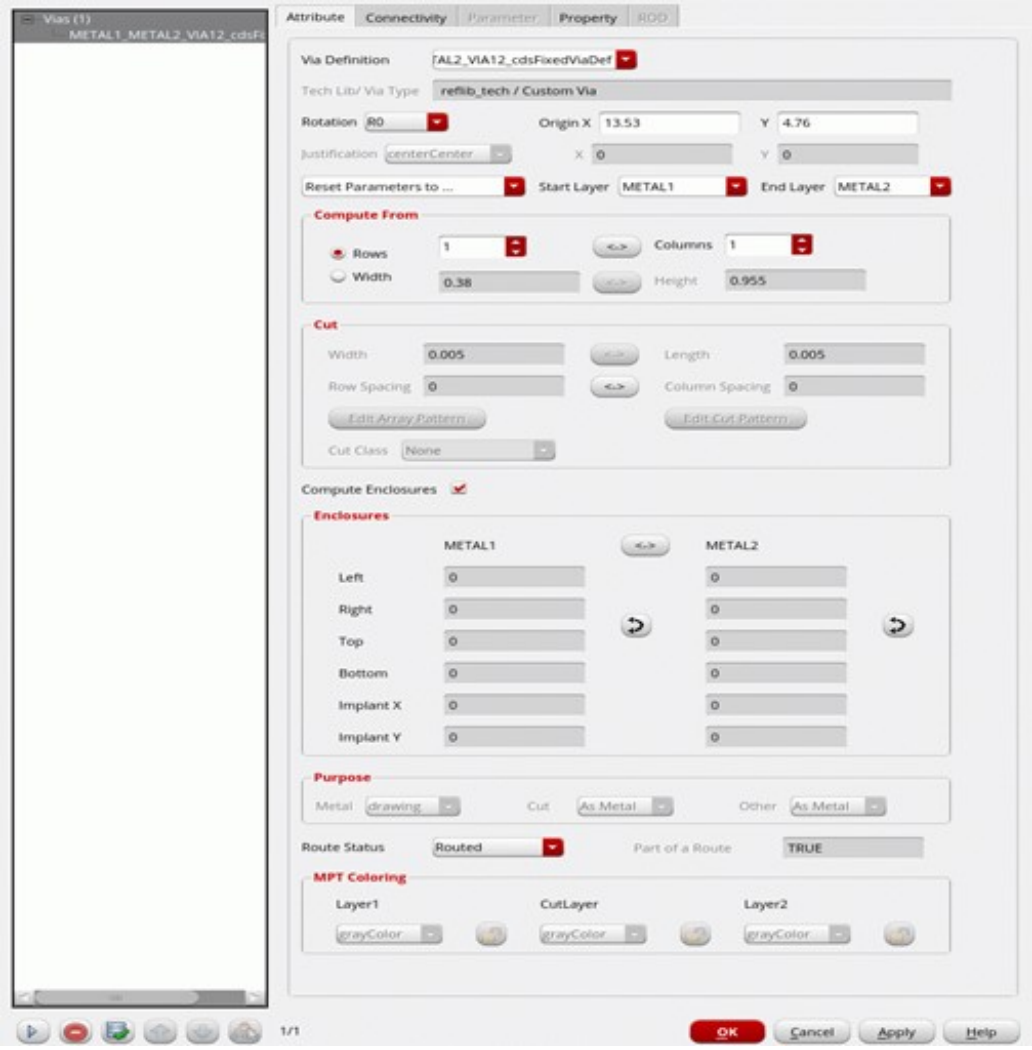
2. Create the cdsFixedViaDefs in the ITDB part of the Rapid MSOA PDK using one of the following methods:
   - Use the `lef2oa -createFixedViaDefs` option while creating the Rapid MSOA PDK. This command will automatically create the needed cdsFixedViaDefs, one for every MetalN and MetalN+1 pair, in the ITDB of the Rapid MSOA PDK.
     Or
   - Use techLoadDump to dump the technology data of the Rapid MSOA PDK and then manually add the cdsFixedViaDefs before loading the tech data back into the MSOA PDK.
3. To enable the creation of cdsFixedVias by Innovus, use the following command:
   `setOaxMode -saveCdsFixedVias { new_vias_only | off}`
   - `new_vias_only` - Any newly created custom via is saved as a fixed via while any pre-existing custom via in the Incremental Technology Database is reused when saving the design.
   - `off` - No cdsFixedVias are created. All fixed/custom vias in the design are saved as oaCustomVia. This is the default option.
4. Ensure availability of required license. The use of cdsFixedVias requires an INVS30 (Innovus Mixed Signal Option) license. With the cdsFixedVia methodology, the custom vias are now saved as parameters for the cdsFixedViaDef. As such, there is no need to save an actual

custom via in the design library. If you want to attach design libraries to a Rapid MSOA PDK, you can now use the attach method without Innovus reporting errors due to not being able to save custom vias in the design library.
Note that if the Rapid MSOA PDK is not enabled for the cdsFixedVia creation by Innovus (no cdsFixedViaDefs in the viaDef section) and the design library is attached to the Rapid MSOA PDK, Innovus will still report an error message when customVias have been created in the design by any of the tools in Innovus.

14.3.3.1 Examining cdsFixedVias in Virtuoso

To examine a cdsFixedVia in Virtuoso:

1. Select the cdsFixedVia in Virtuoso.
2. Open the Edit Via Properties form. The *Via Definition* field displays the via definition in the Rapid MSOA PDK that was used to create the via.
   Note: In the current implementation, all Virtuoso applications continue to show *Via Type* for a cdsFixedVia as *Custom Via*.



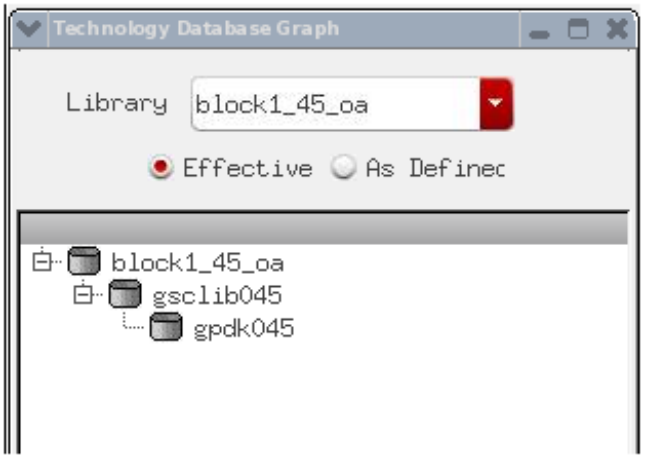
14.3.3.2 Need for cdsFixedViaVariant

Sometimes, you may need to create a constraint group (CG) containing the customVias in the CG's validViasList. For example, suppose a Non-Default Rule (NDR) is created containing a list of the custom vias to be used with that NDR. Given that currently cdsFixedVias cannot be in the validViasList of a CG such as a NDR, Innovus was enhanced to store these custom vias as cdsFixedViaVariants. The name of the equivalent cdsFixedViaVariant will be used by both Innovus and Virtuoso to refer to a particular custom via.

So in a typical design saved by Innovus, you may see cdsFixedVias (used to store custom vias contained in the design) as well as cdsFixedViaVariants (used to store custom vias that are in the validViasList of certain CGs such as NDRs). In the NDR example, if you query the NDR in Innovus, the name of the cdsFixedViaVariant will be displayed. Similarly, the name of the cdsFixedViaVariant will be displayed in Virtuoso as well.

14.4 Examples

When `lef2oa` is run, the software converts the input LEF file into an OpenAccess file. You can view the customViaDefs that were carried over from the LEF file in the gsclib045 library:



Sample customViaDefs in the OpenAccess tech file:

```
;*******************************

; VIADEFS

;*******************************

viaDefs(

 customViaDefs(

 ;( viaDefName libName cellName viewName layer1 layer2 resistancePerCut)

 ;( ---------- ------- -------- -------- ------ ------ ----------------)

   ( M11_M10_1x2_VH_S  gsclib045 M11_M10_1x2_VH_S via Metal10 Metal11 0.0)

   ( M11_M10_1x2_VH_N  gsclib045 M11_M10_1x2_VH_N via Metal10 Metal11 0.0)

   ( M11_M10_2x1_VH_W  gsclib045 M11_M10_2x1_VH_W via Metal10 Metal11 0.0)

   ( M11_M10_2x1_VH_E  gsclib045 M11_M10_2x1_VH_E via Metal10 Metal11 0.0)

   ( M11_M10_M_SH  gsclib045 M11_M10_M_SH via Metal10 Metal11 0.0)

   ( M11_M10_M_NH  gsclib045 M11_M10_M_NH via Metal10 Metal11 0.0)
```

By scrolling further down in the tech file, you can view the LDRS. The LDRS constraint group contains the validVia list:

```
interconnect(

     ( validLayers   (Metal1  Metal2  Metal3  Metal4  Metal5  Metal6  Metal7  Metal8
Metal9  Metal10  Metal11  ) )

     ( validVias     (M2_M1  M3_M2  M4_M3  M5_M4  M6_M5  M7_M6  M8_M7  M9_M8  M10_M9
M11_M10  M2_M1_HV  M2_M1_VV  M2_M1_VH  M2_M1_HH  M2_M1_2x1_HV_E  M2_M1_2x1_HV_W
M2_M1_1x2_HV_N  M2_M1_1x2_HV_S  M3_M2_VH  M3_M2_HH  M3_M2_HV  M3_M2_VV  M3_M2_M_NH
M3_M2_M_SH  M3_M2_2x1_VH_E  M3_M2_2x1_VH_W  M3_M2_1x2_VH_N  M3_M2_1x2_VH_S  M4_M3_HV
M4_M3_VV  M4_M3_VH  M4_M3_HH  M4_M3_M_EV  M4_M3_M_WV  M4_M3_2x1_HV_E  M4_M3_2x1_HV_W
M4_M3_1x2_HV_N  M4_M3_1x2_HV_S  M5_M4_VH  M5_M4_HH  M5_M4_HV  M5_M4_VV  M5_M4_M_NH
M5_M4_M_SH  M5_M4_2x1_VH_E  M5_M4_2x1_VH_W  M5_M4_1x2_VH_N  M5_M4_1x2_VH_S  M6_M5_HV
M6_M5_VV  M6_M5_VH  M6_M5_HH  M6_M5_M_EV  M6_M5_M_WV  M6_M5_2x1_HV_E  M6_M5_2x1_HV_W
M6_M5_1x2_HV_N  M6_M5_1x2_HV_S  M7_M6_VH  M7_M6_HH  M7_M6_HV  M7_M6_VV  M7_M6_M_NH
M7_M6_M_SH  M7_M6_2x1_VH_E  M7_M6_2x1_VH_W  M7_M6_1x2_VH_N  M7_M6_1x2_VH_S  M8_M7_HV
M8_M7_VV  M8_M7_VH  M8_M7_HH  M8_M7_M_EV  M8_M7_M_WV  M8_M7_2x1_HV_E  M8_M7_2x1_HV_W
M8_M7_1x2_HV_N  M8_M7_1x2_HV_S  M9_M8_VH  M9_M8_HH  M9_M8_HV  M9_M8_VV  M9_M8_M_NH
M9_M8_M_SH  M9_M8_2x1_VH_E  M9_M8_2x1_VH_W  M9_M8_1x2_VH_N  M9_M8_1x2_VH_S  M10_M9_HV
M10_M9_VV  M10_M9_VH  M10_M9_HH  M10_M9_2x1_HV_E  M10_M9_2x1_HV_W  M10_M9_1x2_HV_N
M10_M9_1x2_HV_S  M11_M10_VH  M11_M10_HH  M11_M10_HV  M11_M10_VV  M11_M10_M_NH
M11_M10_M_SH  M11_M10_2x1_VH_E  M11_M10_2x1_VH_W  M11_M10_1x2_VH_N  M11_M10_1x2_VH_S  )
)

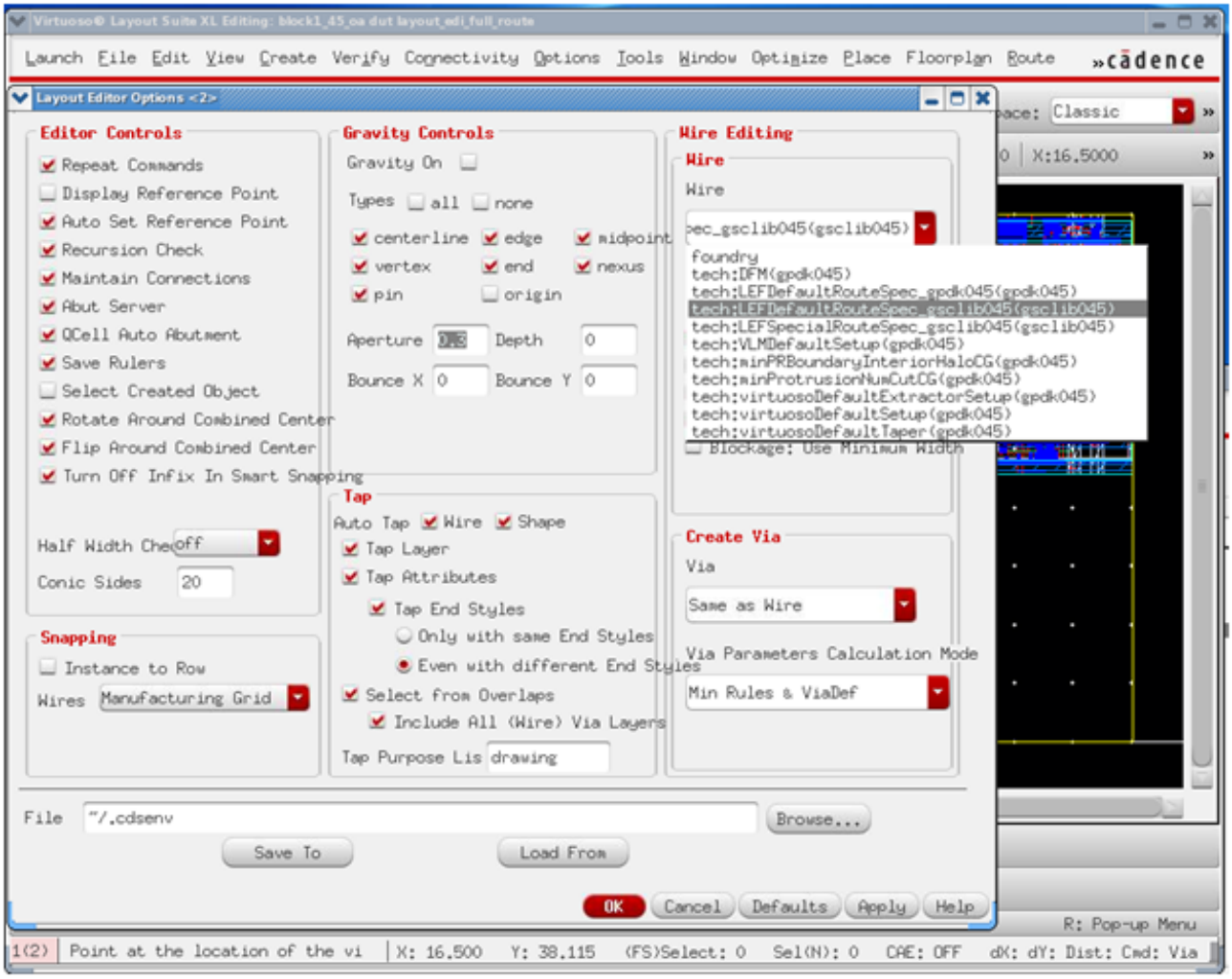
    ) ;interconnect

  ) ;LEFDefaultRouteSpec_gsclib045
```

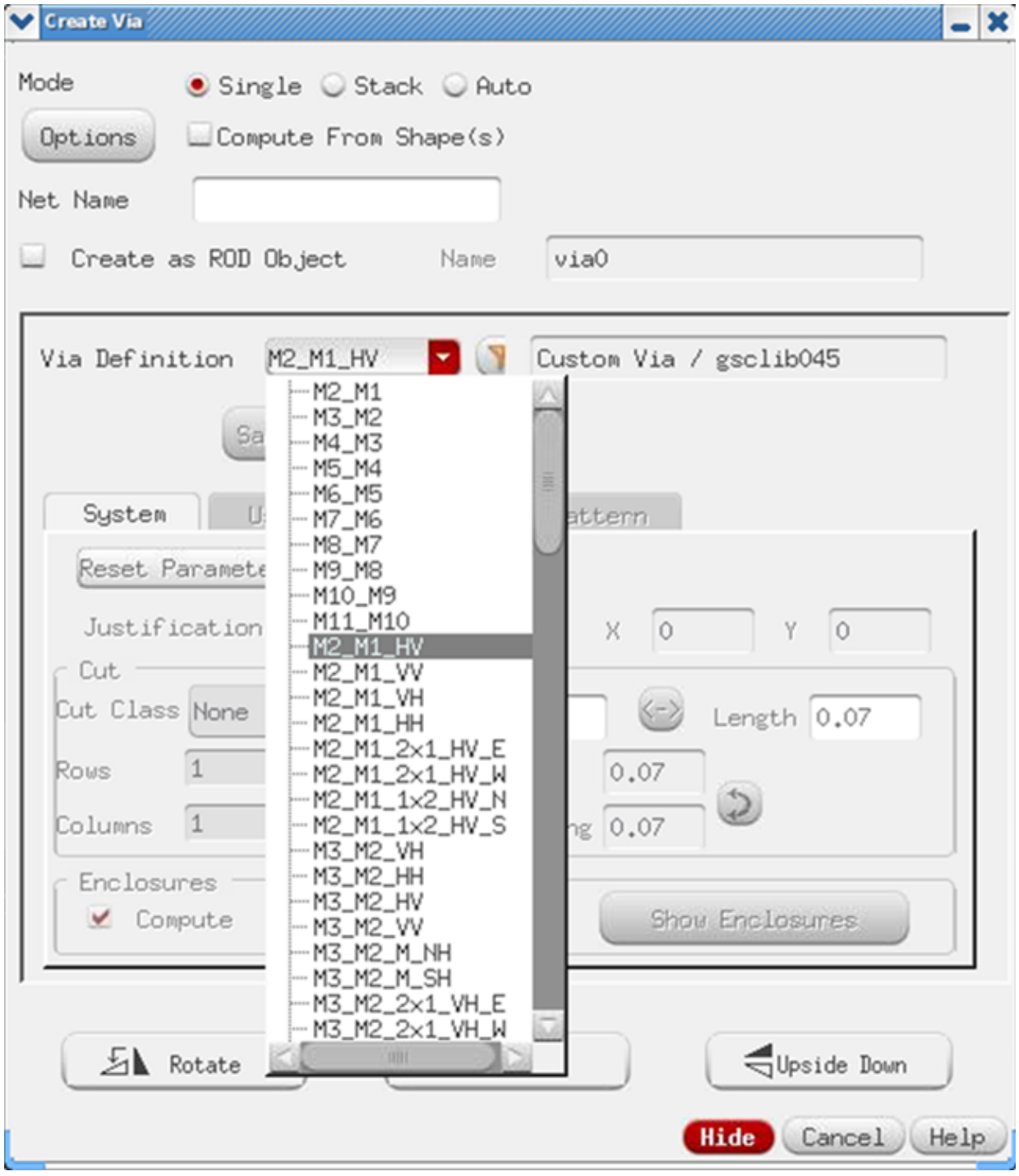
14.4.1 Setting the LDRS in Virtuoso Layout Suite

From Virtuoso Layout Suite XL:

1. Open the Layer Editor Options form by selecting *Options* - *Editor*.
2. On the Layout Editor Options form, specify the LDRS (constraint group) to be used.
3. Set *Create Via* to *Same as Wire* or to a specific LDRS. The is important because the Create Via form reflects the via spacing values in the constraint group selected here.



4. Click *OK*.
5. Open the Create Via form by selecting *Create* - *Via.*
   The Create Via form now displays all the vias in the ValidVia list of the LDRS from the gsclib045 tech file.

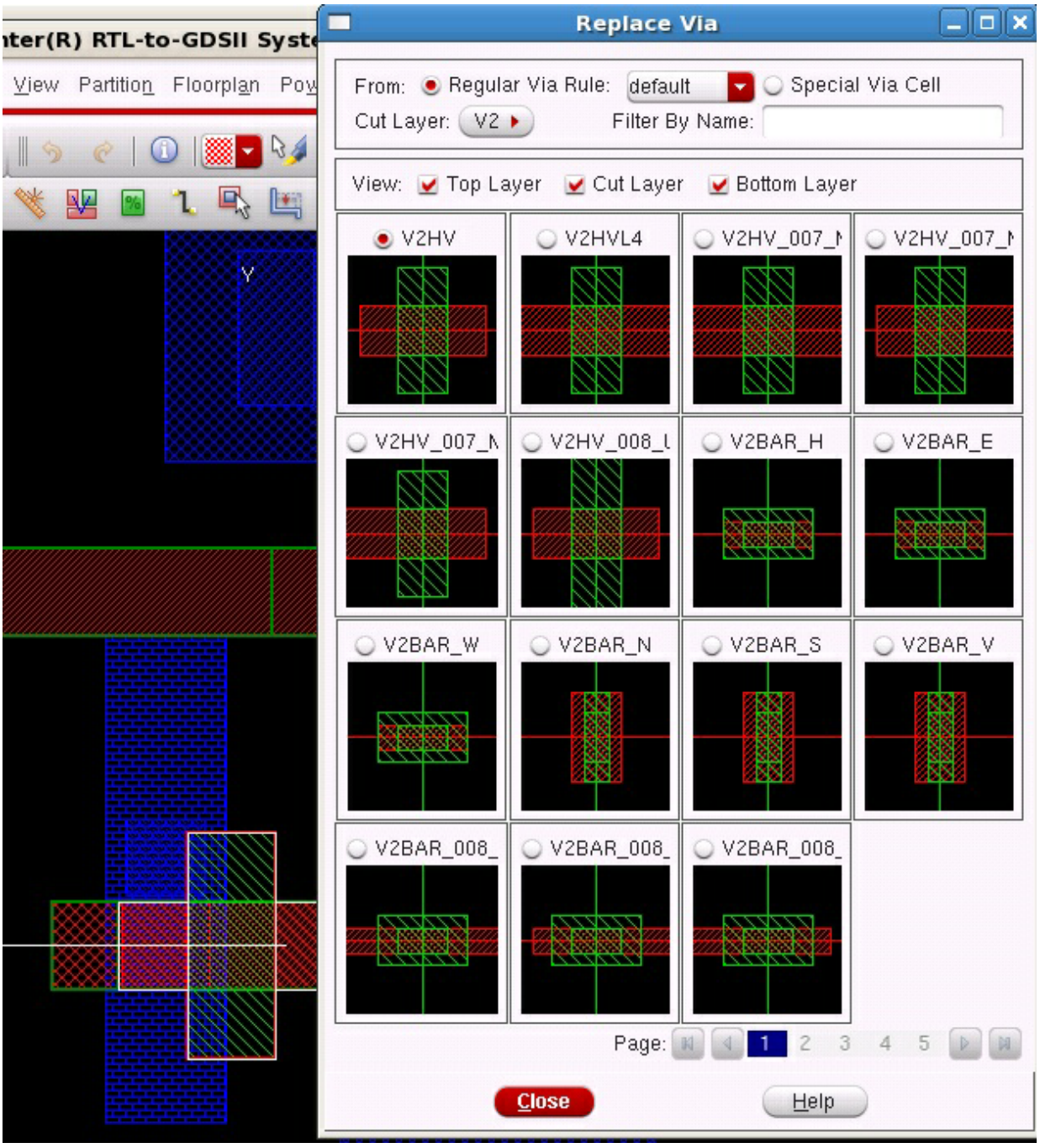

Create Via
Mode
Single
Stack
Auto
Options
Compute From Shape(s)
Net Name
Create as ROD Object
Name
via0
Via Definition
M2_M1_HV
Custom Via / gsclib045
M2_M1
M3_M2
M4_M3
M5_M4
M6_M5
M7_M6
M8_M7
M9_M8
M10_M9
M11_M10
M2_M1_HV
M2_M1_VV
M2_M1_VH
M2_M1_HH
M2_M1_2x1_HV_E
M2_M1_2x1_HV_W
M2_M1_1x2_HV_N
M2_M1_1x2_HV_S
M3_M2_VH
M3_M2_HH
M3_M2_HV
M3_M2_VV
M3_M2_M_NH
M3_M2_M_SH
M3_M2_2x1_VH_E
M3_M2_2x1_VH_W
System
Reset Parameter
Justification
X
0
Y
0
Cut
Cut Class
None
Rows
1
Columns
1
Length
0.07
0.07
0.07
Enclosures
Compute
Show Enclosures
Rotate
Upside Down
Hide
Cancel
Help

14.4.2 Checking Vias in the Tech File

Innovus provides a couple of ways to review the vias that are part of the tech file and/or are generated in Innovus:

14.4.2.1 Viewing Vias Interactively

In Innovus, you can view vias interactively. To do so, select a via and press the F4 function key. This displays all vias as seen below.



You can use appropriate filtering options as per your requirements to update the list of via cell thumbnails.

14.4.2.2 Using write_lef_library -tech_only

Use the following command to write technology data from the LEF file in Innovus to the specified file:

```
innovus 33> write_lef_library -tech_only generateVias.lef
```

The sample output file contains via information, such as follows:

```
VIARULE M2_M1 GENERATE DEFAULT
        LAYER Metal1 ;
                ENCLOSURE 0.005 0.03 ;
        LAYER Metal2 ;
                ENCLOSURE 0.005 0.03 ;
        LAYER Via1 ;
                RECT -0.035 -0.035 0.035 0.035 ;
                SPACING 0.14 BY 0.14 ;
                RESISTANCE 5 ;


END M2_M1


VIA M11_M10_1x2_VH_S DEFAULT
        LAYER Metal10 ;
                RECT -0.105 -0.49 0.105 0.13 ;
        LAYER Via10 ;
                RECT -0.09 -0.45 0.09 -0.27 ;
                RECT -0.09 -0.09 0.09 0.09 ;
        LAYER Metal11 ;
                RECT -0.14 -0.48 0.14 0.12 ;


END M11_M10_1x2_VH_S
```

14.4.2.3 Using dumpOutVias

Use the following command to dump out vias to a specified file:

```
innovus 34> dumpOutVias -file generatedVias.out
```

In the `dumpOutVias` output, each via is described using fixed via constructs. You can identify a custom or fixed via by the `LEF` keyword that appears after the via name. Similarly, the `AUTOGEN` keyword appears after the vias generated by the via generator software. The output file is still a valid LEF file. However, you should use it only for visualization purposes.

Sample output:

```
VIA M2_M1_VV DEFAULT  # LEF
  LAYER Metal1 ; RECT -0.0400 -0.0650 0.0400 0.0650 ;
  LAYER Metal2 ; RECT -0.0400 -0.0650 0.0400 0.0650 ;
  LAYER Via1 ;
    RECT -0.0350 -0.0350 0.0350 0.0350 ;
END M2_M1_VV
VIA M2_M1_HV DEFAULT  # LEF
  LAYER Metal1 ; RECT -0.0650 -0.0400 0.0650 0.0400 ;
  LAYER Metal2 ; RECT -0.0400 -0.0650 0.0400 0.0650 ;
  LAYER Via1 ;
    RECT -0.0350 -0.0350 0.0350 0.0350 ;
END M2_M1_HV
VIA M2_M1_2x1_VV_E DEFAULT  # AUTOGEN
  LAYER Metal1 ; RECT -0.0400 -0.0650 0.1800 0.0650 ;
  LAYER Metal2 ; RECT -0.0400 -0.0650 0.1800 0.0650 ;
  LAYER Via1 ;
    RECT -0.0350 -0.0350 0.0350 0.0350 ;
    RECT 0.1050 -0.0350 0.1750 0.0350 ;
END M2_M1_2x1_VV_E
VIA M2_M1_2x1_HH_E DEFAULT  # AUTOGEN
  LAYER Metal1 ; RECT -0.0650 -0.0400 0.2050 0.0400 ;
  LAYER Metal2 ; RECT -0.0650 -0.0400 0.2050 0.0400 ;
  LAYER Via1 ;
    RECT -0.0350 -0.0350 0.0350 0.0350 ;
```

```
    RECT 0.1050 -0.0350 0.1750 0.0350 ;

END M2_M1_2x1_HH_E
```

14.5 Directing the Router to Use Only the Vias in the ITDB in Innovus

By default, power routing (`sroute`) looks through all the via rules to identify the best-matched via. This leads to differences in behavior in the OA and LEF/DEF flows. This may also cause issues when you are using the NanoRoute engine for the routing signal flow.

To prevent such issues, use the following setting in the flow to force the power router to use only the vias available in the ITDB:

```
setViaGenMode -viarule_preference [dbGet head.viaRuleGenerates.name *xxxx*]
```

Note: In future releases, this setting will be enhanced to support multiple naming convention for vias.

15

Placing Standard-cell-based Designs from Virtuoso

- Overview
- Guidelines for Placing a Standard-cell-based Design from Virtuoso in Innovus
- What are Sites and Rows in Innovus and Why They Are Needed
- Site Property in the Cell View of Standard Cells
- Site Definition in the Technology Library
- Standard Row Creation by Innovus During init_design
- Switching the Orientation of the Standard Rows

15.1 Overview

You can use the Innovus placer to place standard cells on a cell view that was originally created in Virtuoso. The OpenAccess-based Mixed Signal flow supports the following flows:

- The floorplanning step, like I/O pin assignment and design size, is done on a cell view in Virtuoso. This cell view is then brought into Innovus for placement of standard cells. You may choose to place certain standard cells in Virtuoso manually, while leaving the rest to be placed by Innovus. These manually placed cells will have to be set to the `FIXED` placement attribute, either in Virtuoso or Innovus. Similarly, if the cell view contains macros/blocks, they can be pre-placed and fixed in the design.
- Making minor changes in Innovus to a design that is already placed or implemented in Virtuoso. For example, a certain cell is required to be upsized or a new cell is to be added into the design. The ECO flow in Innovus allows the new cells to be ECO-placed while refining the placement of the rest of the standard cells.
- Some cells are constrained with inclusive cluster boundary. When the cell view is read by

Innovus, the inclusive cluster boundary will be viewed as a floorplan object, named region, in Innovus. These cells are guided to be placed within the region. The rest of the standard cells will be as placed as normal.

In addition to offering flexibility in the flow, placing the design in Innovus offers some advantages:

- Innovus offers features such as timing driven placement if timing libraries and timing constraints are available.
- The ECO flow can also be exercised in which the tool can easily make changes to the design. Changes can be logical, such as cell changes, and physical, such as placement and/or routing.
- For lower nodes, the placer and router engines in Innovus are tuned to work together to achieve optimal quality of result.

After placement of standard cells is done, you can either save the resulting cell view to make it available to Virtuoso or proceed to route the design in Innovus using NanoRoute.

The OpenAccess-based Mixed Signal flow allows you to choose freely when to switch back to Virtuoso – either after placement or after routing.

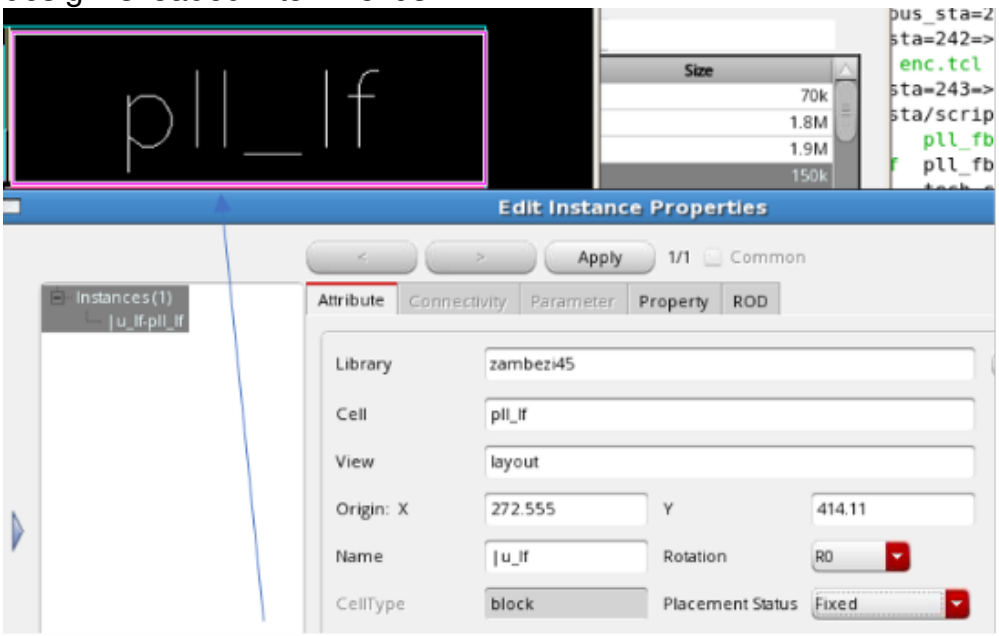
15.2 Guidelines for Placing a Standard-cell-based Design from Virtuoso in Innovus

Keep the following guidelines in mind while using Innovus to place a Standard-cell-based Design from Virtuoso:

- The placer in Innovus requires rows based on the site(s) to be created. Rows created in Virtuoso may not be interoperable in Innovus, especially if rows or row regions are created in which no site is associated. Thus, it is recommended to create standard cell rows in Innovus. For more details on row and site, refer to the What Are Sites and Rows in Innovus and Why They Are Needed section.
- The Core Boundary in Innovus affects the way it creates the row. Refer to the Design and Core Boundary in Innovus chapter for details on Core Boundary. If some cells are already pre-placed in Virtuoso and some other cells are to be placed in Innovus, the Core Boundary may have to be re-adjusted so that the rows created in Innovus are aligned with the standard cell rows.
- For a design that consists of blocks and standard cells, if the blocks are pre-placed in the

design by Virtuoso and not to be moved by Innovus in the flow later, specify their placement attributes to be `Fixed` (or `Locked` /`Cover`). If a block is not marked as `Fixed`, its location may be altered by Innovus while placing the standard cells. This may also lead to the tool not being able to perform placement refinement successfully due to the inclusion of the area occupied by `PLACED` or `UNPLACED` blocks in the utilization calculation.

If only routing is to be done in Innovus (and, for example, no ECO placement), the placement attributes of `None` or `Placed` are also fine as they will be translated to `PLACED` when the whole design is loaded into Innovus.



Query the block to open the GUI form to edit the placement status

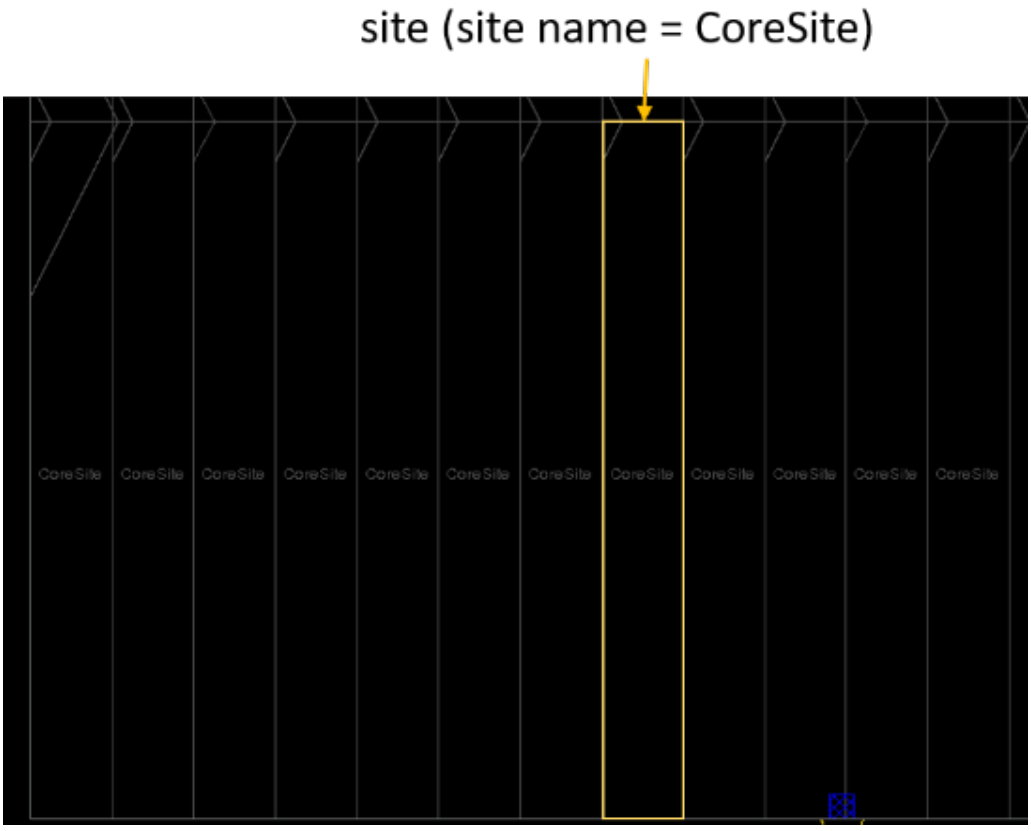
Note that in Innovus, it is a good practice to make blocks as `FIXED` once their locations are finalized. You can run the following command to set the placement status of all blocks (hard macros) to `FIXED`:

```
setInstancePlacementStatus -allHardMacros -status fixed
```

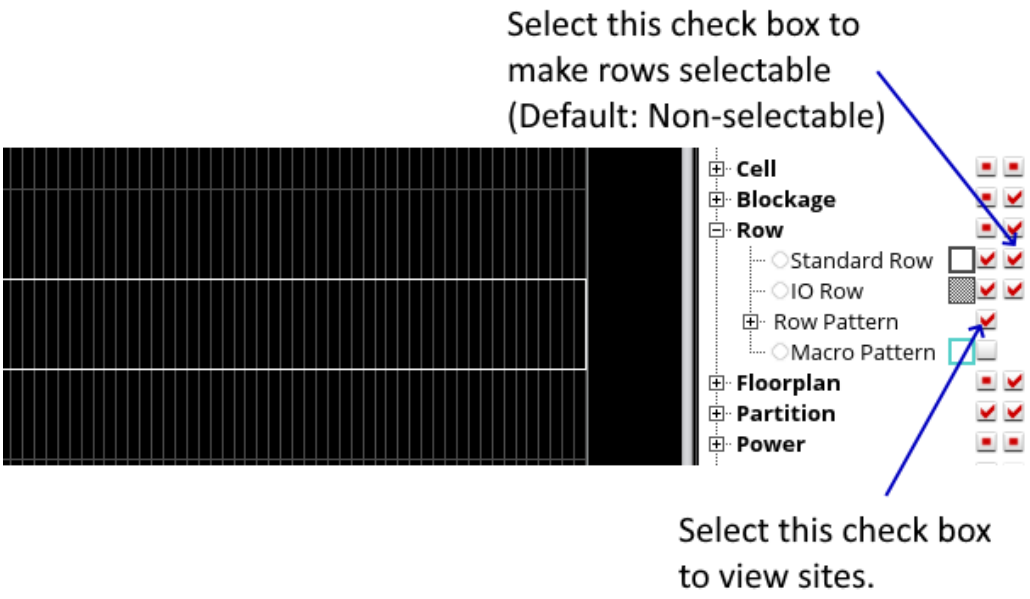
- If a group of standard cells is manually placed in the design, you may choose not to place filler cells in Virtuoso but to leave empty gaps between the standard cells and let Innovus auto-insert fillers in these gaps later in the flow. If you plan to do ECO placement in the cell view, inserting filler cells in Innovus allows you to remove them easily in Innovus during the ECO flow by running the `deleteFiller` command. Besides inserting filler cells, the Innovus placer also supports the insertion of tap cells and end cap cells.

15.3 What are Sites and Rows in Innovus and Why They Are Needed

Site and row are basic terms related to placement and routing. A site defines a placement site, and is a placement location that can be used by the cells that match the same site. Physically, it is a rectangle with a specific width and height. A repeating set of sites creates a row.

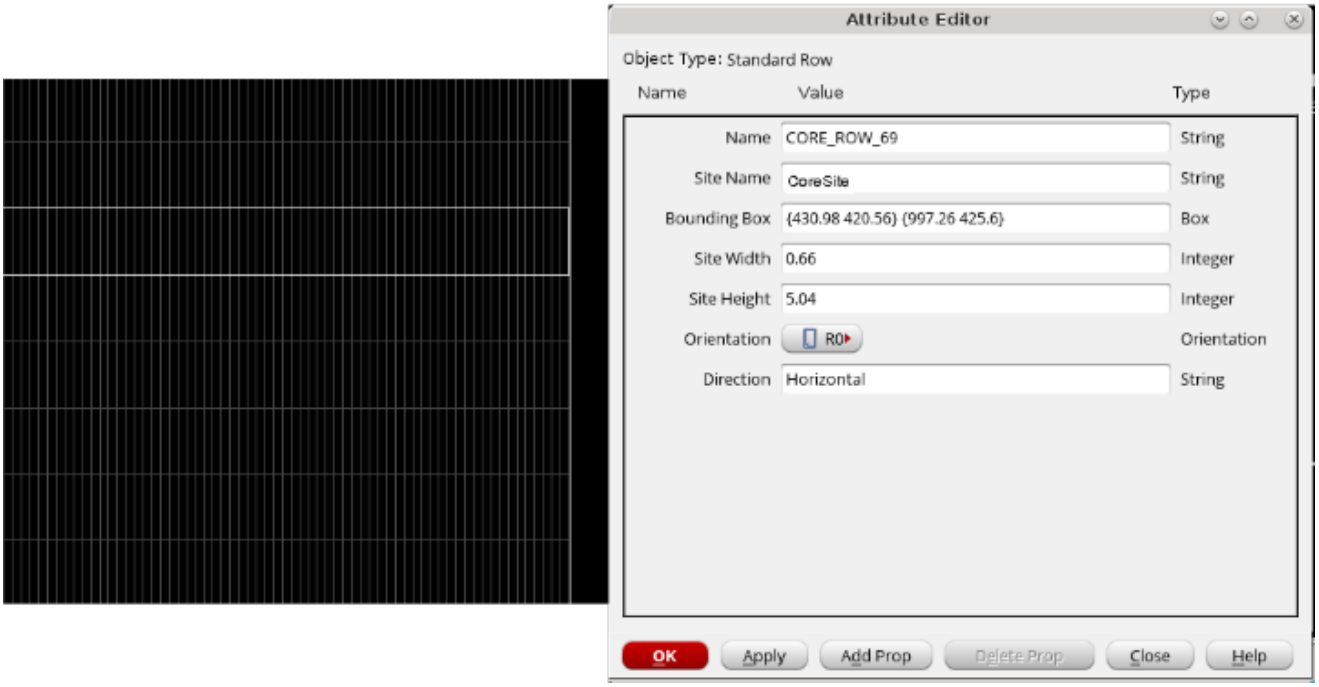


A site can either be an I/O pad site for the placement of I/O pads or a core site for the placement of standard cells. The above and below pictures show how the repeating sites form a horizontal standard cell row. Sites for standard cells are called core sites because the standard cells are placed in the core area of the design.



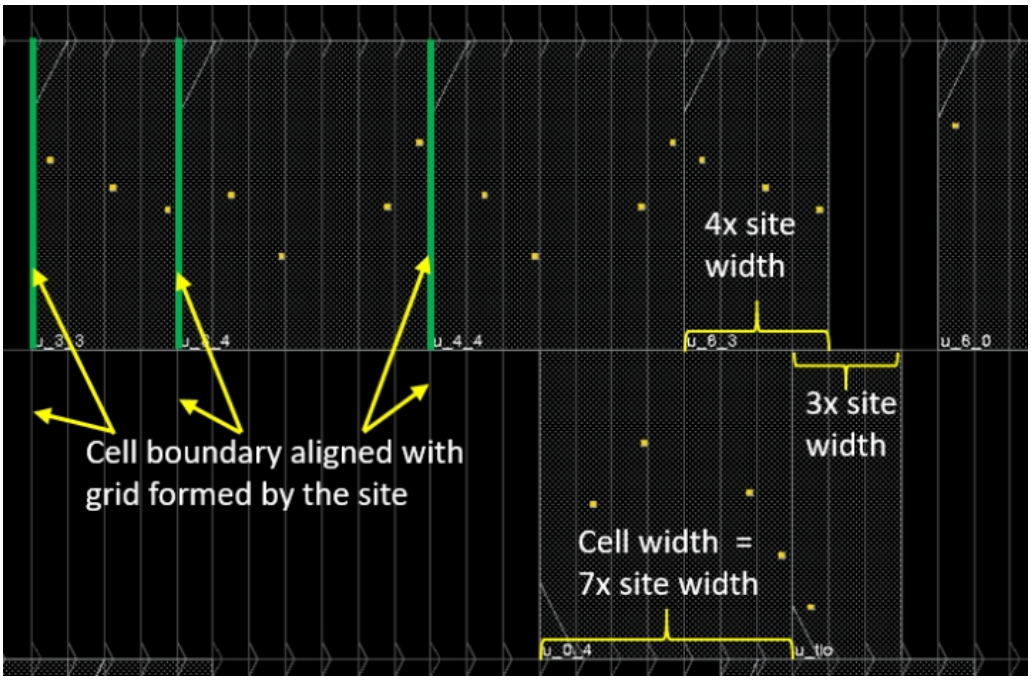
In Innovus, the standard cell rows in the design are not selectable by default. To make them selectable, check the *Selectability* check box for *Standard Row* in the Control Panel, which is located at the right of the main window. Similarly, the sites are not visible by default so you need to select the *Visibility* check box for *Row Pattern* to view the sites.

After selecting a standard cell row, double-click or press the q key to view the *Site Name*, *Site Width* and *Site Height*, and other attributes of the row.



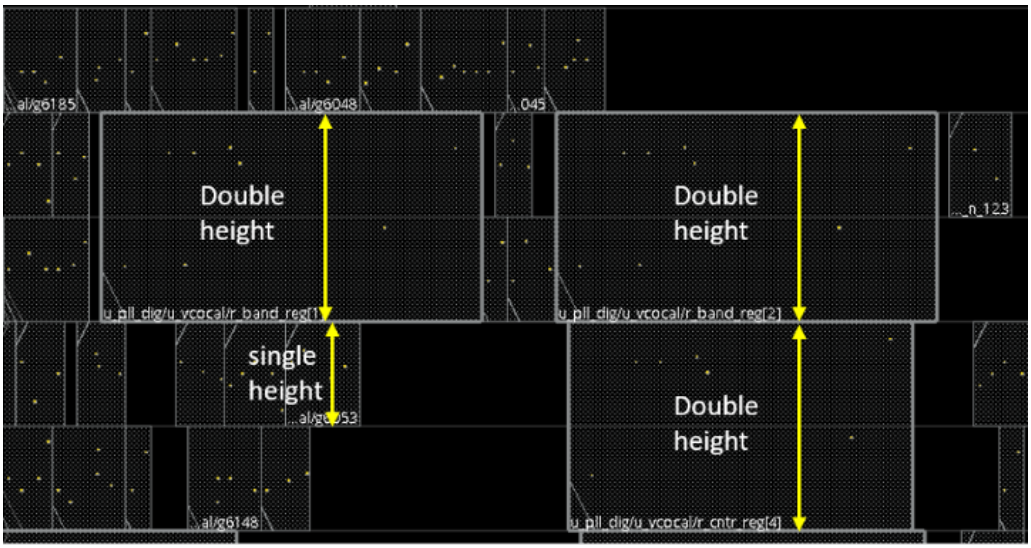
Typically, standard cells are structurally designed with heights that are integral multiples of the site

height, such as single-height cells or double-height cells. This is done for several reasons, such as optimal use of the area, enabling placement of cells with alignment, allowing power rails to be shared, and so on. Most importantly, standard cell rows must be present in the floorplan for Innovus to auto-place the standard cells.

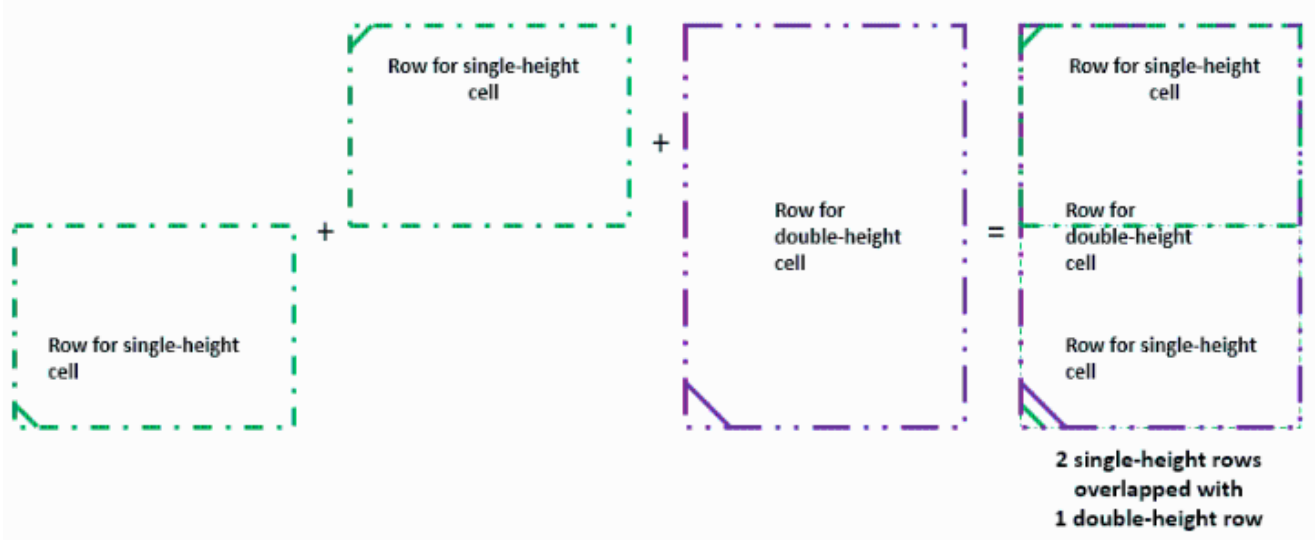


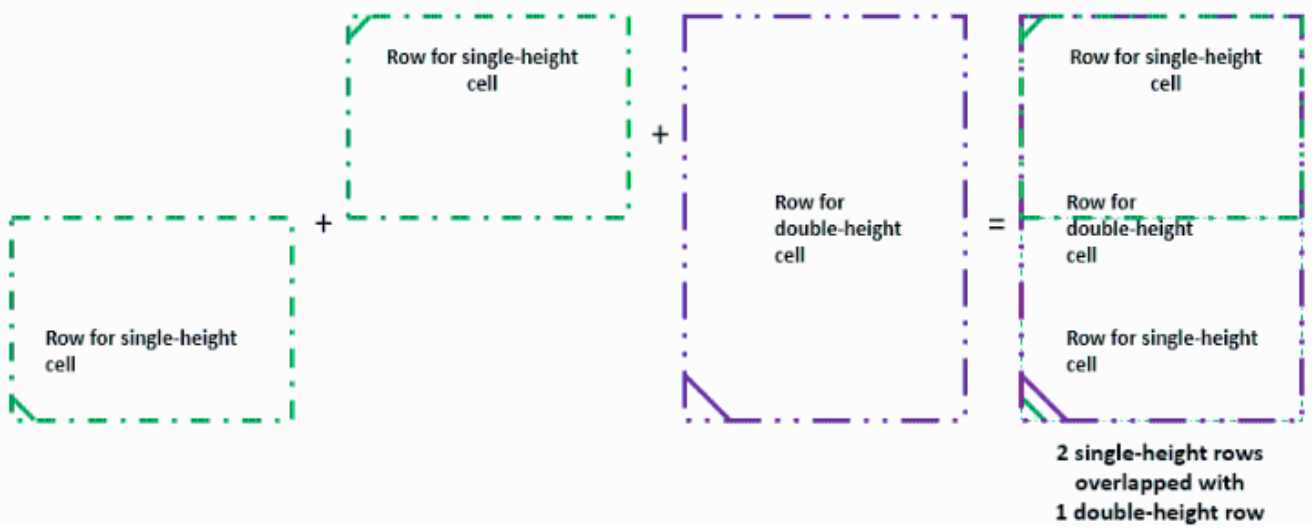
The diagram above shows how standard cells are placed along the row. These standard cells are single-height cells. Each row consists of abutted sites and the Placer in Innovus always snaps the cells to the site. With very few exceptions, standard cells are typically designed in a way that the cell widths are multiple integral value of the site width. For example, if the site width is 0.5um, the standard cells are designed with widths of 0.5um, 1um, 1.5um, and so on.

The diagram below shows three double-height cells placed along with many single-height cells.



To allow placement of both single- and double-height cells in the same area, both single-height rows and double-height rows are to be created such that they overlay each other. The following diagram shows how rows of different heights and orientations overlay each other.

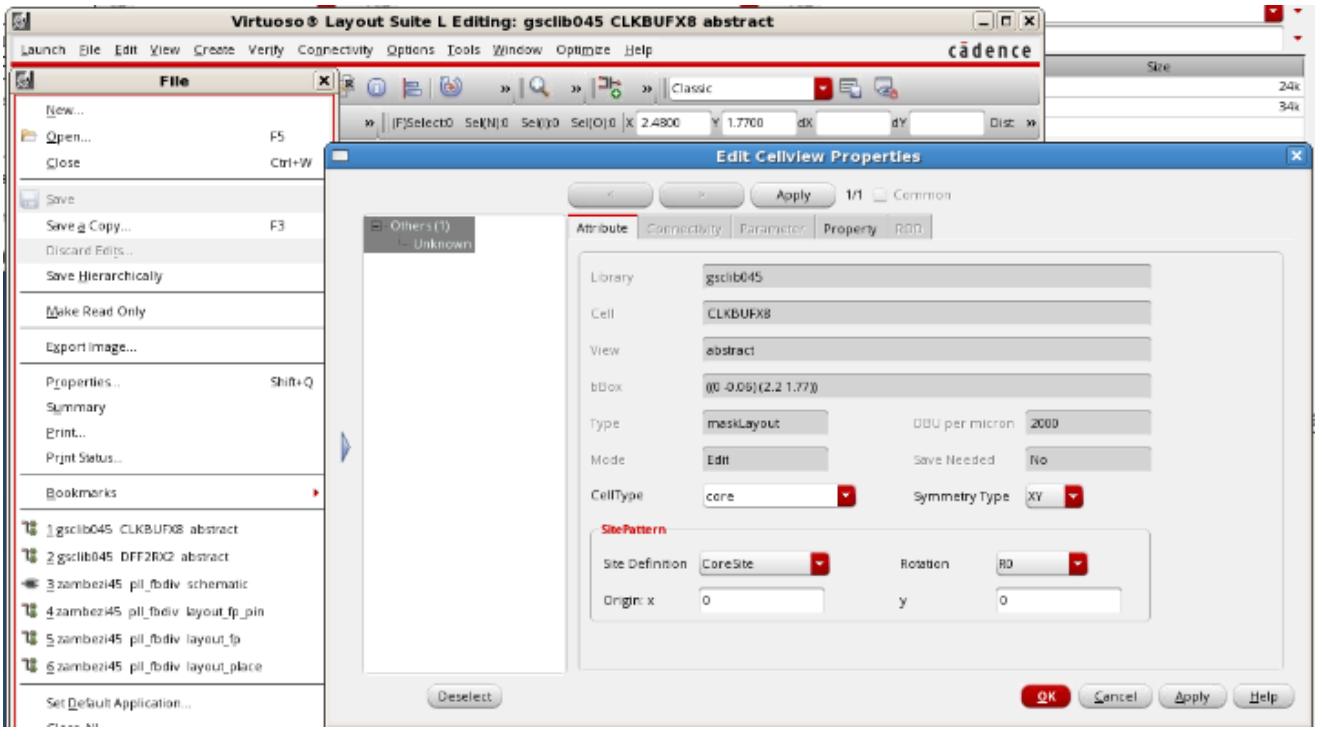




Note that `CoreSiteDouble`, for double-height standard cells, has the same site width as `CoreSite`, but its height is double that of `CoreSite` so that two single-height rows can overlay a double-height row completely.

15.4 Site Property in the Cell View of Standard Cells

Site definition is a property that should be defined in the abstract of the standard cells. To view the site definition of a cell view, open it in Virtuoso and select *File -> Properties* from the menu bar. This displays the Edit Cellview Properties form. The following example shows the properties of the `CLKBUFX8` standard cell. Its site name is `CoreSite`. Its *Symmetry Type* of `XY` indicates that it can also be placed with different orientations of `R0`, `MX`, `MY`, and `R180`. Note that this cell is of *cellType* `core`, which enables Innovus to identify it as a standard cell.



It is also possible to view the site property of a standard cell in Innovus. Simply run `write_lef_library` with the `-macro_only` option and search the generated output LEF file for the keyword `SITE` on that standard cell.

```
> write_lef_library -macro_only my_macro.lef
```

The following is the example for the `CLKBUFX8` cell from the output LEF file:

```
MACRO CLKBUFX8

        CLASS CORE ;

        FOREIGN CLKBUFX8 0 0  ;

        ORIGIN 0 0 ;

        SIZE 2.2 BY 1.71 ;

        SYMMETRY X Y ;

        SITE CoreSite ;

...
```

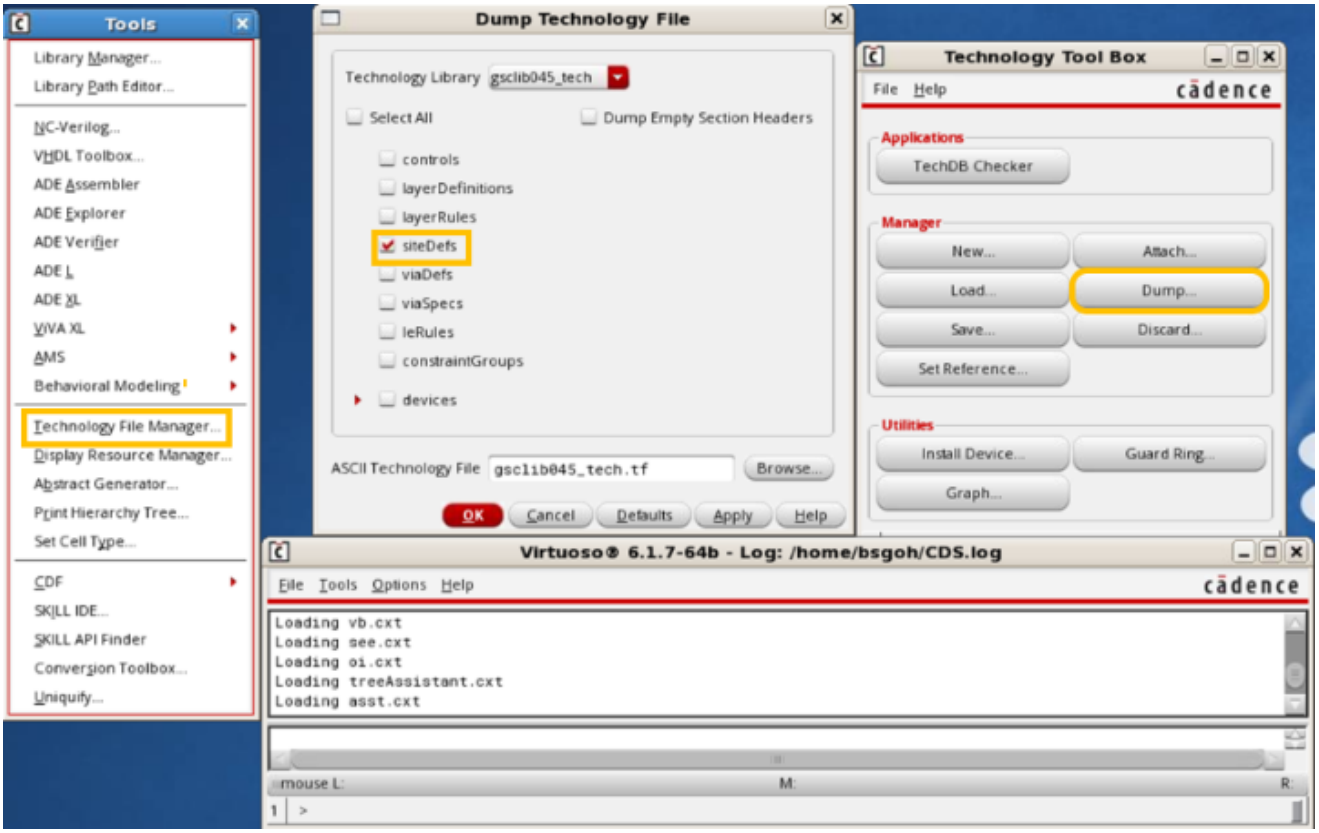
Note that the LEF statements for `CLKBUFX8` also show similar properties, such as symmetry type (`SYMMETRY X Y`), site name (`SITE CoreSite`), and cell type (`CLASS CORE`).

15.5 Site Definition in the Technology Library

The MSOA flow requires an MSOA PDK for the flow to be run successfully. A regular PDK (not MSOA PDK) meant for manual layout editing in Virtuoso will commonly not contain any site definition in the technology library. Refer to section "*3.2 Technology Data Preparation*" in the Technology Data Preparation chapter to see how the SITE definition contained in either the digital TLEF or the macro LEF are imported to the ITDB during the creation of an MSOA PDK.

To see if the technology library contains any SITE definition, dump out the text technology file and view the `siteDefs` section. The steps are as follows:

1. Invoke Virtuoso.
2. In the Command Interface Window, select *Tools -> Technology File Manager*.
3. In the resulting Technology Tool Box form, click the *Dump* button. This opens the Dump Technology File form.
4. Select the desired technology library, select the *siteDefs* check box, specify the file name in the *ASCII Technology File* text box and click *OK*.



The following example shows the SITE definition section in a technology file. Two sites are defined:

CoreSite and CoreSiteDouble.

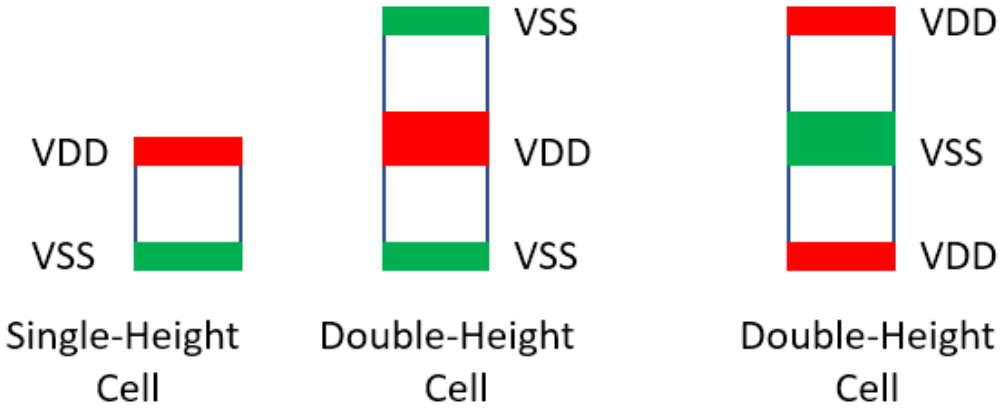
```
;*******************************
; SITEDEFS
;*******************************
siteDefs(
 scalarSiteDefs(
 ;( siteDefName          type width  height  symInX symInY symInR90)
 ;( -----------          ---- -----  ------  ------ ------ -------)
  ( CoreSite             core 0.2     1.71    nil    t     nil)
  ( CoreSiteDouble       core 0.2     3.42    t      t     nil)
) ;scalarSiteDefs
) ;siteDefs
```

Alternatively, in Innovus, run `write_lef_library -tech_only` to generate an output LEF file that contains the technology information. Open the LEF file and search for the keyword `SITE`:

```
> write_lef_library -tech_only my_tech.lef
SITE CoreSite
        CLASS CORE ;
        SYMMETRY Y ;
        SIZE 0.2 BY 1.71 ;
END CoreSite
SITE CoreSiteDouble
        CLASS CORE ;
        SYMMETRY X Y ;
        SIZE 0.2 BY 3.42 ;
END CoreSiteDouble
```

In the above example, for the `CoreSite` definition meant for single-height standard cells, the `t` Boolean value for the `SymInY` attribute or the `SYMMETRY Y` statement in TLEF indicates that the row allows the standard cell with the `SYMMETRY Y` or `SYMMETRY XY` property to be flipped horizontally along the row. This is typical for single-height standard cells as flipping the cells vertically will

cause a short on the power rails.



In double-height cells, the upper and lower cell edge typically contains the same type of power rail: either both are VDD or both are VSS. Therefore, double-height cells can be flipped vertically. So in the `CoreSiteDouble` definition meant for double-height standard cells, the `t` Boolean value for the `SymInX` attribute or the `SYMMETRY X Y` statement in TLEF indicates that the row allows the standard cell with the `SYMMETRY Y` or `SYMMETRY XY` property to be flipped horizontally along the row and standard cell with the `SYMMETRY X` or `SYMMETRY XY` property to be flipped vertically along the row.

Note that without having the site defined in the technology libraries, Innovus will issue a messages such as the following in the log file:

```
**ERROR: (IMPOAX-750):  Tech site 'CoreSite' used in macro 'CLKBUFX8' is not found.
Check if this macro in OpenAccess (OA) database has correct site information, and if
the site is defined in OA technology database.

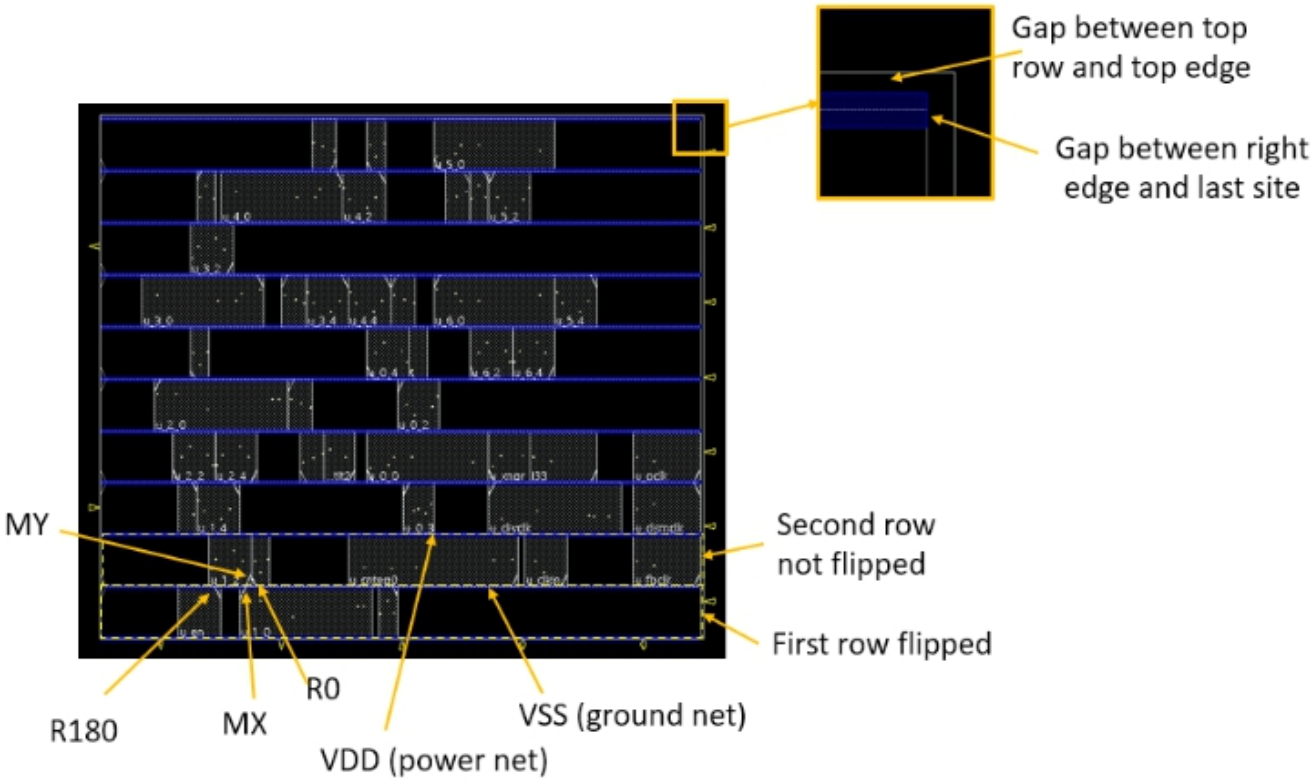
**WARN: (IMPOAX-755):   Macro 'CLKBUFX8' has no SITE statement and it is a class CORE
macro that requires a SITE statement. The SITE ENC_CORE_0 is chosen because it is a
core site with height 1.7100 that matches the macro SIZE height.
```

When Innovus observes that the site property of a standard cell is not defined in the technology, it creates a dummy site with its best guess for site height and width. In general, it does a good job in computing the site height and width. However, the dummy site may not provide the correct symmetry type. The Placer may be able to place that standard cell, but there is no guarantee that all placement operations will work properly with the dummy site. As the site definitions for standard cell sets are often defined in the macro LEF or technology LEF, these files should always be imported when creating the MSOA PDK.

15.6 Standard Row Creation by Innovus During init_design

Consider a cell view that has no standard cell rows and has its PRBoundary object created in Virtuoso. When this cell view is loaded into Innovus through the `init_design` TCL command or through the Design Import form, Innovus detects the absence of rows and populates the entire design with rows.

Innovus inserts the rows from the bottom to the top, with sites added from left to right. The bottom row is the first row added, and is flipped by default so that it can share the same power rail with the next row, which is not flipped. This implies that single-height standard cells with XY symmetry are placed either with the MX or R180 orientation in the first row. Similarly, for the other odd rows. All the even rows are not flipped, and therefore single-height standard cells in those rows are placed with either R0 or MY orientation.

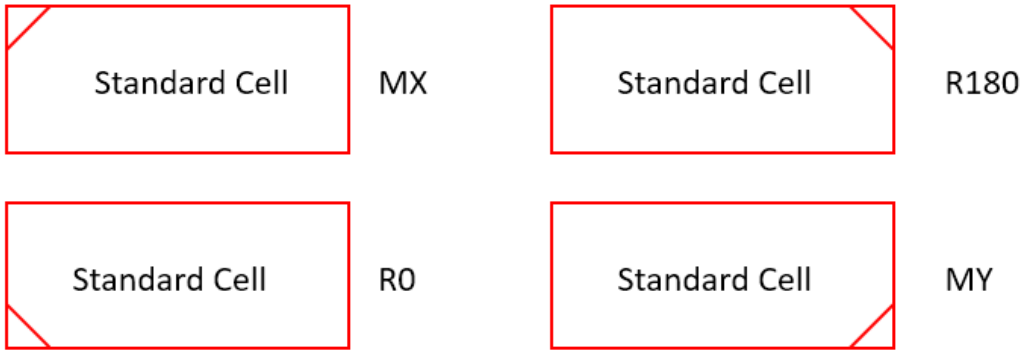


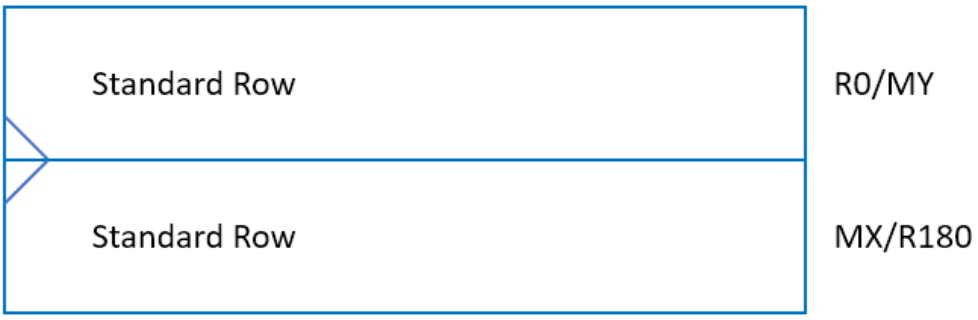
As shown in the picture above, the first row is inserted in such a way that the bottom edge of the row aligns with the bottom edge of the design boundary (determined by the specification of the PRBoundary object) and the left edge of the row aligns with the left edge of the design boundary.

As the tool fills up the design with only full sites and no partial site is created, some empty space is expected between the top edge of the top row and the top edge of the design boundary when the

height of the design is not a value multiple of the row height. Similarly, there is empty space between the right edge of the row and the right edge of the design boundary too when the width of the design boundary is not a value multiple of the site width.

To identify the orientation of a row or cell, check which corner of the row or cell has the cross line.



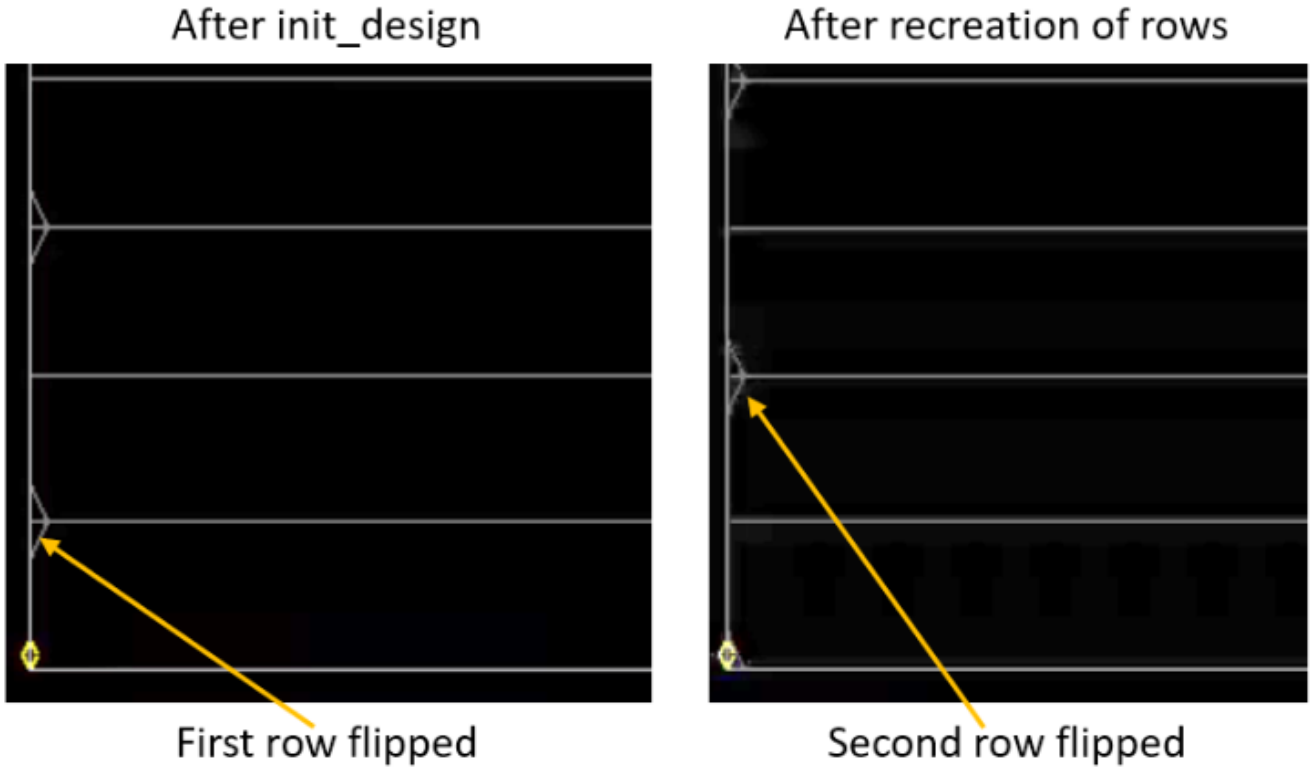


15.7 Switching the Orientation of the Standard Rows

As the bottom row (and other odd number rows) created by `init_design` is flipped by default, these rows are unusable if the standard cells are to be placed with a different orientation. To recreate rows with a different orientation, follow the steps below:

1. `deleteRow -all`
2. `setFlipping s`
3. `initCoreRow`

This set of steps basically deletes the current rows and recreates the rows with a different orientation. The `setFlipping` command specifies that the second and other even rows are to be flipped.



16

Using NanoRoute To Route a Design from Virtuoso

- Overview
- What Are Routing Tracks?
- Guidelines for Using Innovus To Route a Design from Virtuoso
- Interoperability of Routing Tracks
  - Supported Use Models for the Interoperability of Routing Tracks
  - Role of add_tracks in the Interoperability of Routing Tracks
  - Role of setOaxMode -updateMode in the Interoperability of Routing Tracks
- Track Alignment in Innovus
  - Relationship Between Wire Width, Routing Track Pitch, and Minimum Spacing Requirement
  - Alignment of I/O Pins with Routing Tracks
  - Alignment of I/O Pins with Non-Uniform-Pitch Tracks in Lower Technology Nodes
  - Access to I/O Pins with Double Patterning Layers in Lower Technology Nodes
  - Access to Standard-Cell I/O Pins in Lower Technology Nodes
  - Access to Standard-Cell I/O Pins with Double Patterning Layers
  - Track Patterns with Non-Uniform Pitch
  - Track Patterns with Non-Uniform Pitch and Non-Default Width

16.1 Overview

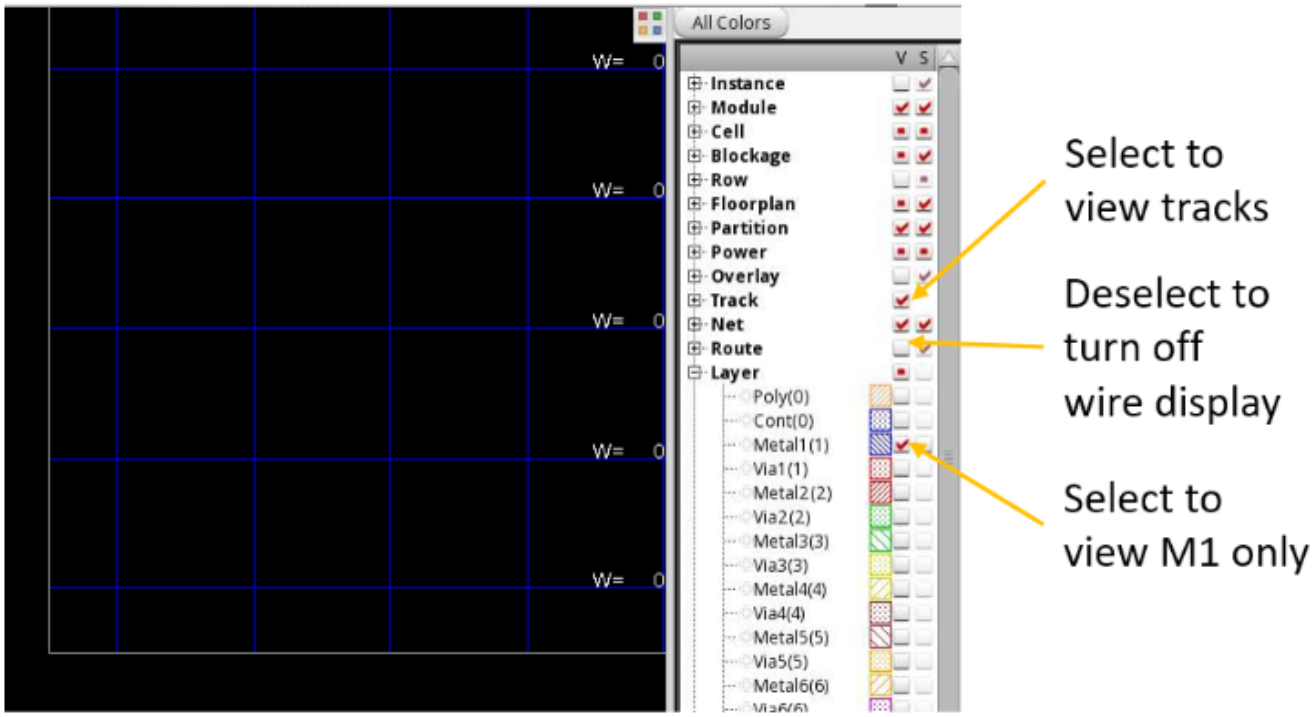
You can use the Innovus router, NanoRoute, to route a cell view that was originally created in Virtuoso. Two use models are available for this flow; you can use NanoRoute to route either a standard-cell-based block in the design or the top level of the design.

As discussed in the Routing Constraint Interoperability chapter, interoperable routing constraints can be created in Virtuoso and later be used by NanoRoute in Innovus. When routing a standard-cell-based block with NanoRoute, it is recommended that you do the standard cell placement as well in Innovus, especially at the lower technology nodes. The OpenAccess-based Mixed Signal flow is ideal for this capability because the entire implementation can be done in Innovus and the resulting cell view can be made available to Virtuoso with no translation.

If you plan to make ECO changes on a cell view using Innovus, then using NanoRoute to route this cell view will make the design friendly for running NanoRoute in the ECO mode. During ECO mode routing, NanoRoute will reroute any nets that were affected by the ECO changes, such as change of cell, insertion of new cell, or change of I/O pin location, through the `ecoRoute` command.

16.2 What Are Routing Tracks?

In Innovus, a routing track defines the routing grid for the signal router of regular nets. These routing tracks provide guide lines for routing and are used by the router, NanoRoute, to determine the preferred locations and directions for routes. Typically, the tracks patterns are uniformly-spaced, orthogonal patterns extending across the whole design. The routing tracks are, by default, invisible in the main layout window of Innovus. To view the routing tracks in design, select the *Visibility* check box next to the *Track* object type in the control panel. In the image below, the routing grid for M1 are displayed as blue lines:

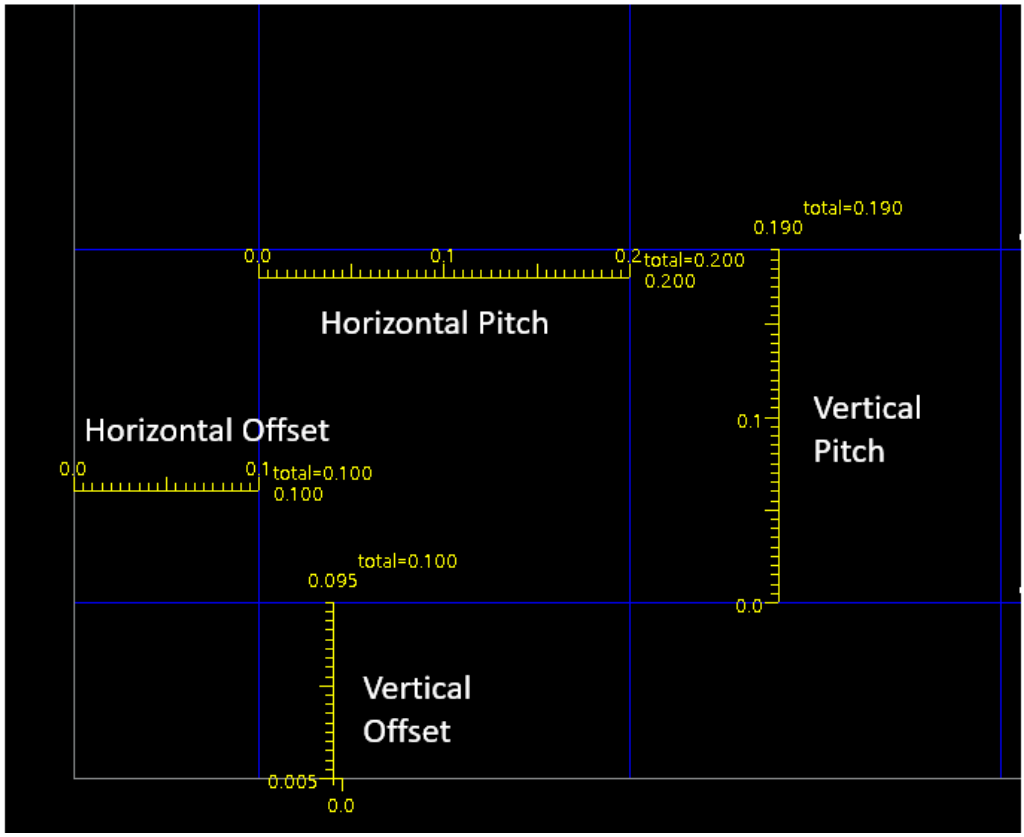


Note that the display of `W= 0` does not mean that the net is to be routed with a wire of 0 width. `W= 0` means that the default width is specified for the tracks, where default width is the width specified in the LEFDefaultRouteSpec in the technology library. If a non-default width is specified for the tracks, the display will show the exact width value.

To view the tracks better, you may deselect the *Visibility* check boxes of other object types, such as *Instance*, *Route*, and *Row*. You can also turn off other routing layers. For example, turn off any layers other than M1 if only M1 routing grid is to be viewed.

For a cell view that has its floorplan created in Virtuoso with floorplan but with no rows and no Width Spacing Patterns (WSPs), Innovus will insert the rows and tracks accordingly during the import process of the design.

Typically, the routing tracks for each metal routing layer have uniformed spacing, called `pitch`. The first horizontal track starts from the bottom and the distance between the first track and the bottom edge of the core area is called `offset`. These specifications are commonly found in the LEFDefaultRouteSpec constraint group in the technology library.



For the above example, the values are specified in the technology file as follows:

```
"LEFDefaultRouteSpec" nil "LEFDefaultRouteSpec"

...

routingGrids (

  ( horizontalPitch    "M1" 0.2 )

  ( verticalPitch      "M1" 0.19 )

  ( horizontalOffset   "M1" 0.1 )

  ( verticalOffset     "M1" 0.095 }
```

Alternatively, in the Technology LEF file, the equivalent statements are:

```
LAYER M1

...

PITCH 0.2 0.19 ;

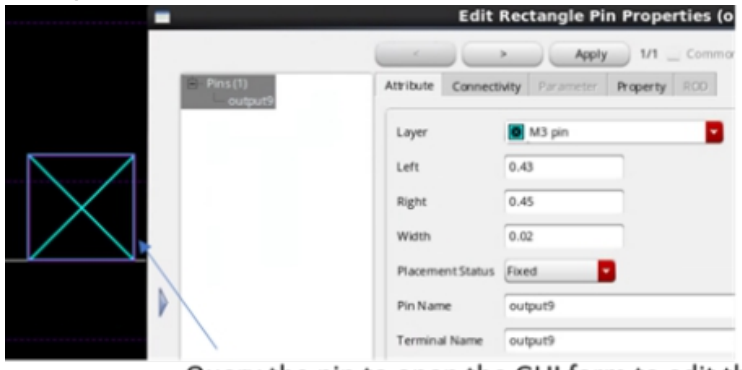
OFFSET 0.1 0.095;
```

16.3 Guidelines for Using Innovus To Route a Design from Virtuoso

- If I/O pins are created and placed in Virtuoso and their corresponding nets are to be routed in Innovus, do not specify the placement attribute to be `None` or `Unplaced` as then the router in Innovus will view them as not physically placed and will not able to route. The router will issue a message similar to the following:

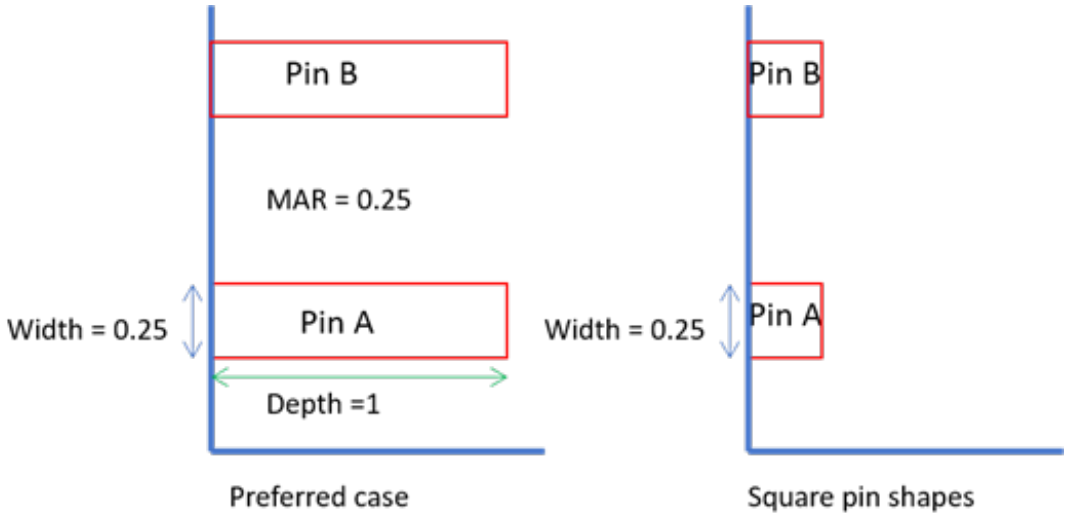
  ```
  #WARNING (NRIG-39) NanoRoute cannot route to pin PIN/output9 of net output9
  because it might be a logical pin and does not have physical shapes. To fix the
  problem, check the pin connection in the netlist and ensure the pin shapes exist
  in the floorplan.
  ```

  It is recommended to update the placement attribute of I/O pins to `Fixed` (`Locked`/`Cover` is fine too) if no adjustment of the placement location is to be done in Innovus or `Placed` if adjustment is expected.

  Query the pin to open the GUI form to edit the placement status

- For I/O pins created in Virtuoso, it is recommended that the depth of the pin shapes should be large enough to meet the minimum area rule (MAR), if such a rule exists. In most cases, the router in Innovus can add wires to increase the depth of pin shapes to meet the minimum area rule during the detailed routing of the whole design. However, ensuring that the pin shapes have appropriate depth upfront will ease the burden on the router and may prevent creation of DRC violations in corner cases.



- The detailed abstract of the standard cells to be routed in the design should be loaded into Innovus during design initialization (through the `init_oa_ref_lib` variable), especially for good quality of routing result in the lower technology nodes.
- The Core Boundary in Innovus affects the way routing tracks are created. Refer to the Design and Core Boundary in Innovus chapter for more details of core boundary and its relationship with track creation.
- Alignment of routing tracks with the I/O pins of the design or the standard cells are critical, especially in smaller nodes. Improper alignment may result in bad quality of result. Refer to the Track Alignment in Innovus section for more information on aligning tracks with other objects, such as the I/O pins of design and cells.

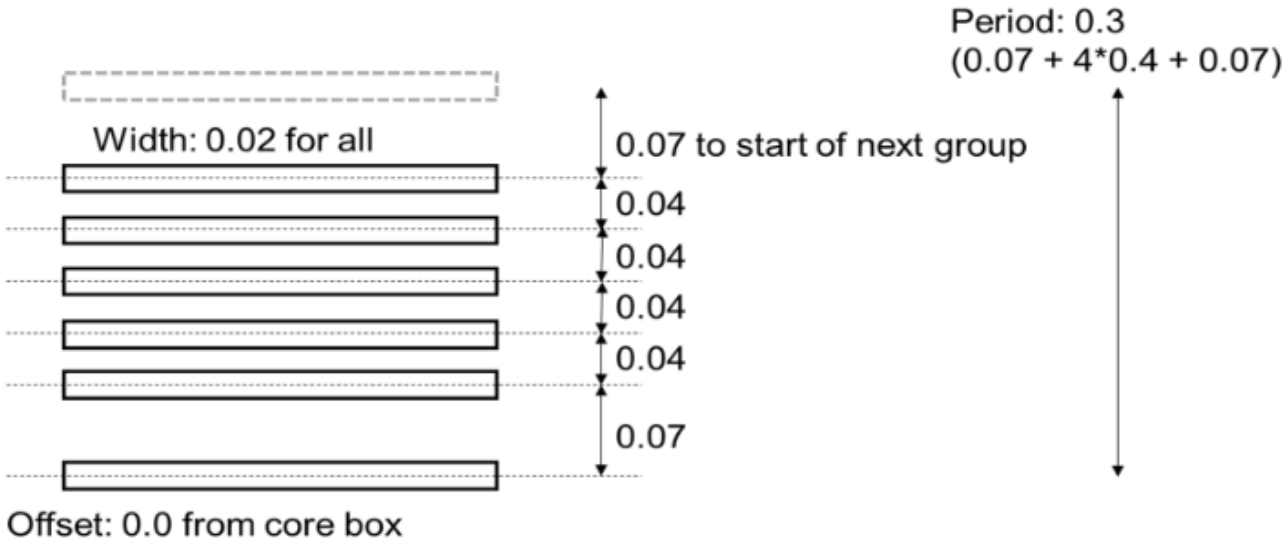
16.4 Interoperability of Routing Tracks

A higher percentage of mixed signal designs are being implemented at lower semiconductor process nodes. At these nodes, tools that are used to implement the custom portion of the mixed signal design have to adhere to certain structures for implementing the routing requirement of the design. Traditionally, place and route tools used for implementing the digital portion of mixed signal designs have had to create routes based on certain track definitions. For mixed signal designs using lower semiconductor process nodes, these two different methods of creating routes must be aligned. Otherwise, issues can crop up during the assembly of the overall design, which may require changes to the content that has already been implemented.

For lower process nodes, Virtuoso requires a Width Spacing Pattern (WSP) to be created and used when routes are created in the tool. An example of a WSP created for `METAL1` in Virtuoso is given below:

```
dbCreateWidthSpacingPattern(cv "oax_pattern1_cv"
        '(((0.020 0.070)) ((0.020 0.040)) ((0.020 0.040)) ((0.020 0.040)) ((0.020
0.040)) ((0.020 0.70)))
        0.0 ; Assume offset 0 from the axis and use the space in the pattern
)
dbCreateWidthSpacingSnapPatternDef(cv "oax_M1WSP_cv" list("METAL1"  "track")
        "vertical"           ; direction
        0.30                 ; period
        "oax_pattern1_cv"    ; default active pattern
        list(list("METAL1")) ; snapping layers
        0.0                  ; offset from PR Boundary or axis
        list("oax_pattern1_cv")  ; list of possible patterns
)
```

The graphical representation of the tracks defined by this WSP is shown below:



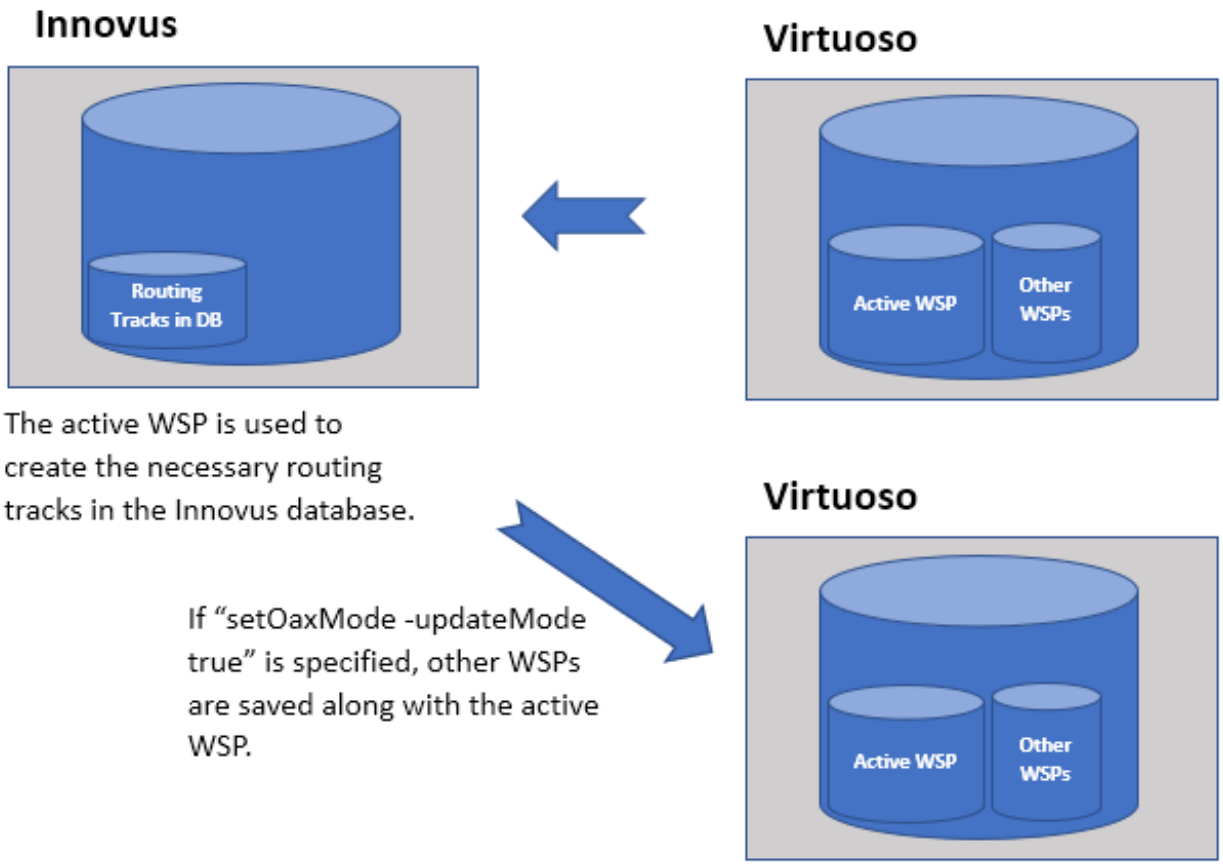
16.4.1 Supported Use Models for the Interoperability of Routing Tracks

16.4.1.1 Use Model A

In a mixed signal flow in which the custom content is created in Virtuoso and the digital content in Innovus, a user may create a WSP in Virtuoso and mark it as the active WSP to be used for implementing the digital content of the design. In this case, Innovus must read and convert the user-created, active WSP to track patterns that are used by the router in Innovus.

Note that a specific cell view may have many different WSPs. However, only the active WSP would

be converted to routing track patterns in Innovus. This use model is depicted in the diagram below.



Use Model A: The active WSP is automatically converted to routing tracks in Innovus.

In this flow, the active WSP is directly stored in the Innovus DB as routing tracks. However, if you later use database access routines to change the routing tracks in the database, Innovus will create an additional WSP with the following naming convention:

```
invs_<layer_name>_wsp // WSP Name
```

Once the design is saved in Innovus, you can see both the original WSP as well as the Innovus-generated one.

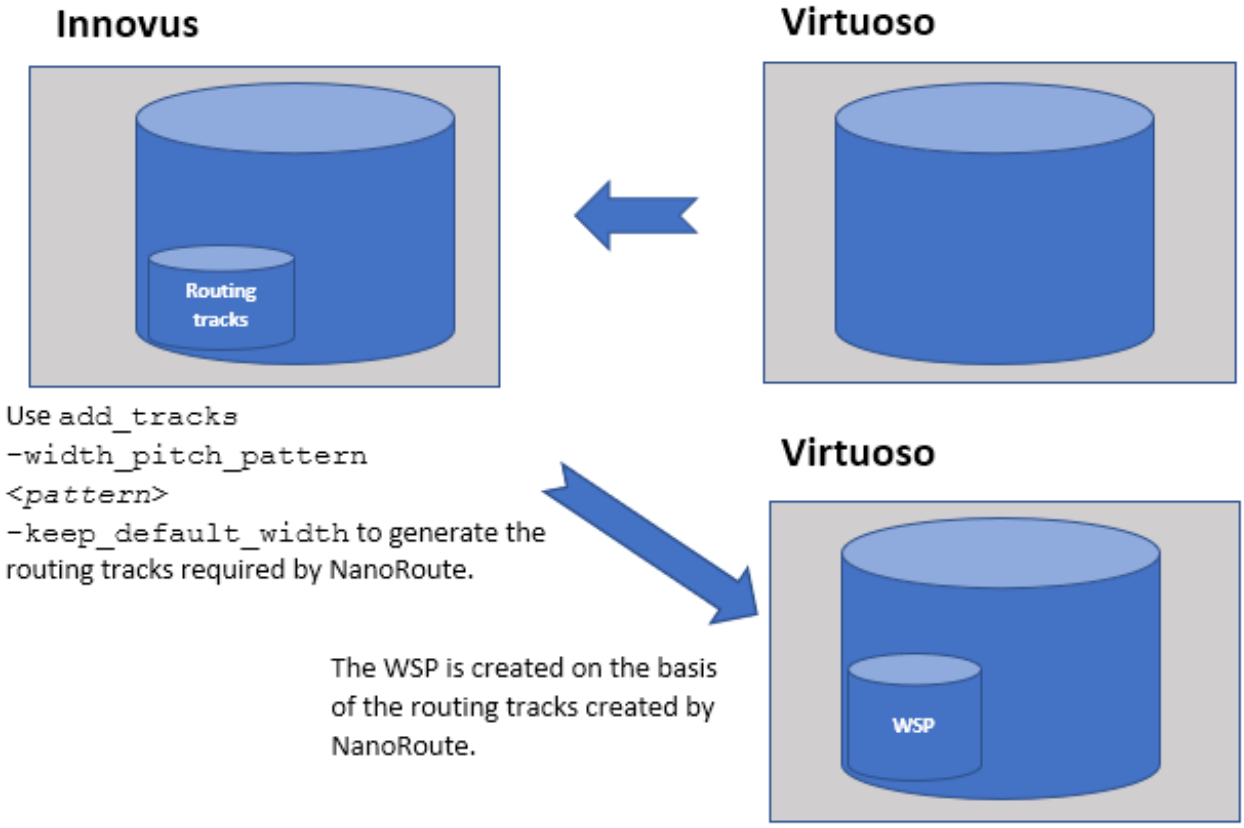
16.4.1.2 Use Model B

In this use model, you do not create any WSP in Virtuoso or do not mark any of the WSPs as active. This use model should be used if you wish to run NanoRoute in Innovus on either a standard-cell-based block in the design or the top-level design and take the results back to Virtuoso. NanoRoute has the ability to create the most efficient routing tracks for a specific design or cell view. You can take advantage of this capability to make Innovus generate a WSP automatically based on the routing tracks in Innovus. This Innovus-generated WSP can then be used for wire editing or other applications in Virtuoso. This process is done automatically in the OpenAccess flow when saving the design.

In this flow, the track patterns in Innovus can be created using the Innovus `add_tracks` command, with the following two options at a minimum: `-width_pitch_pattern` and `-keep_default_width`.

The WSP created during `saveDesign` by Innovus will have the following naming convention:

```
invs_<layer_name>_wsspd // Snap Pattern Def  name
```



Use Model B: No active WSP exists, and Innovus routing tracks are used to create WSPs.

16.4.2 Role of add_tracks in the Interoperability of Routing Tracks

The `add_tracks` command is used in Innovus for creating the complex routing track patterns required in lower technology nodes.

In some lower technology nodes, specific track pattern definitions are required by the foundry so that the tool can generate the most efficient routing results for the design. It is possible to make the Innovus-generated routing tracks compatible with Virtuoso so that a WSP can be created from the routing tracks used by Innovus. If `add_tracks` is used in Innovus, you need to make sure that the following two options are part of the syntax for the `add_tracks` command because the use of these options with the `add_tracks` command makes the generated tracks interoperable with the WSPs in Virtuoso. The two options are `-width_pitch_pattern` and `-keep_default_width`.

The `-keep_default_width` parameter has no argument. However, the syntax for the `-width_pitch_pattern` pattern is as follows:

`-width_pitch_pattern {{`*`layer1 offset offset width width pitch pitch`* `[`*`width width pitch pitch`*`]...} [{`*`layer2 ...`*`}]...}`

16.4.3 Role of setOaxMode -updateMode in the Interoperability of Routing Tracks

`setOaxMode` is an Innovus command that controls the behavior of the Innovus tool when run in the OpenAccess mode. One of the options for this command is `-updateMode false | true | auto`.

When `setOaxMode -updateMode` is set to `true`, Innovus retains and writes back all custom objects, such as Fig-groups, Modegen cells, and so on. Therefore, if a design containing such custom objects is brought into Innovus for modifications, it would be best to set `-updateMode` to `true`.

When `setOaxMode -updateMode` is set to `false`, Innovus ignores the custom objects in the design. This mode is typically used when working on a digital block of a mixed signal design, where the implementation is being done using Innovus. In this situation, there are no custom objects that Innovus needs to retain.

If there is more than one WSP in the design, it is important to specify `setOaxMode -updateMode true` if you wish to retain the non-active WSPs as part of the design. Similarly, if WSP regions or WSP families are to be preserved in the round trip to Innovus, `setOaxMode -updateMode true` is required. Innovus will not recognize either of these WSP types, but will retain the information and send it back to Virtuoso.

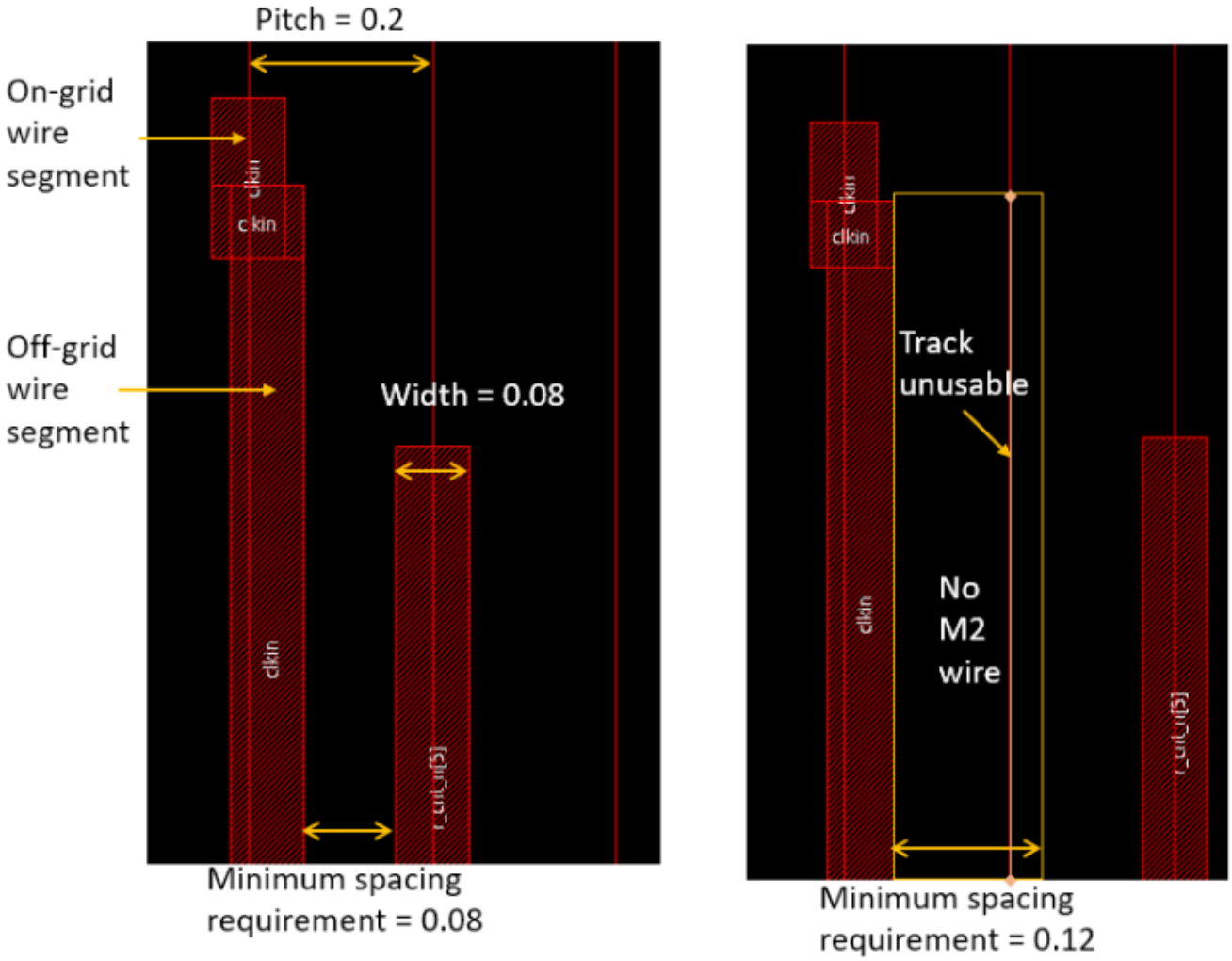
16.5 Track Alignment in Innovus

For aligning tracks with objects such as I/O pins, several points need to be considered. These are:

16.5.1 Relationship Between Wire Width, Routing Track Pitch, and Minimum Spacing Requirement

NanoRoute prefers to create routes along the routing grid with tracks being the center-line of wires. However, NanoRoute has the flexibility to create off-grid wires, where necessary. For example, an off-grid wire may be created to make a connection to an off-grid I/O pin. Nevertheless, that will mostly incur the loss of the adjacent track segment. The impact of the misalignment between certain I/O pins and routing tracks depends on the wire width, routing track pitch, and the minimum spacing requirements specified in the technology.

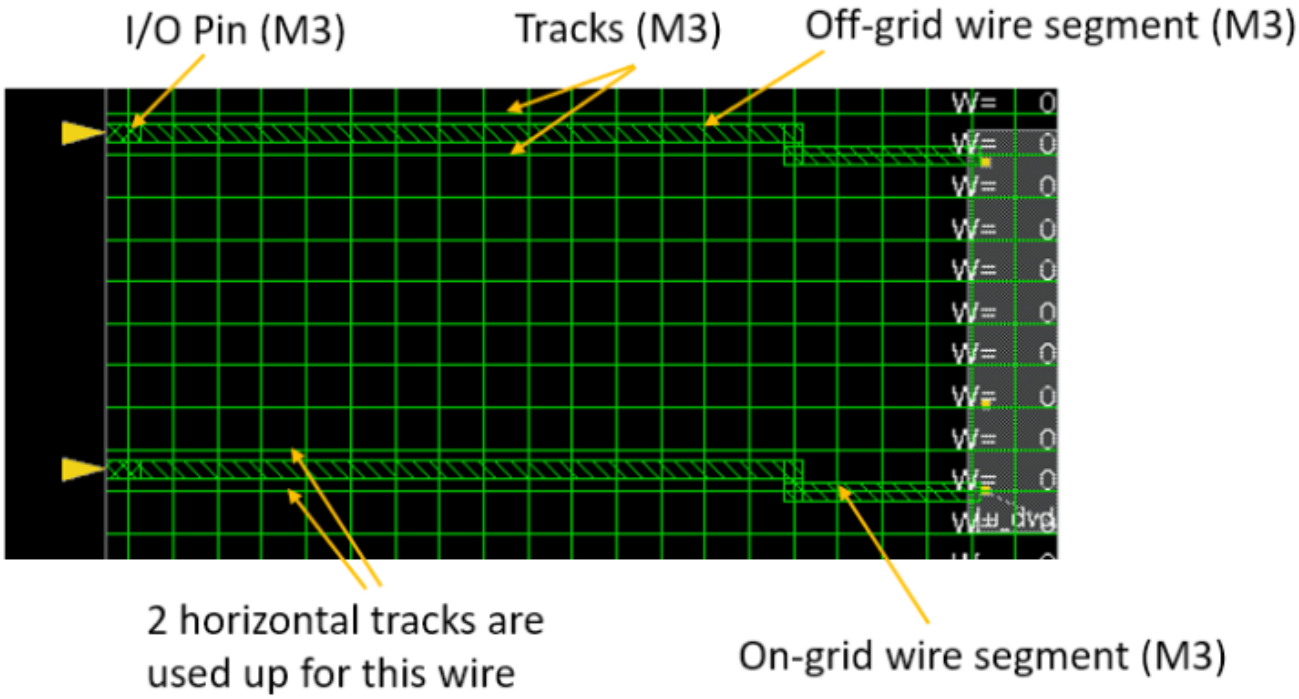
In the image on the left below, the routing pitch of M2 is 0.2um and the wire width of M2 is 0.08um. If the minimum spacing is 0.08, an off-grid wire with its center-line of not more than 0.04um deviation from the grid is still okay.



However, on certain technology nodes, it is possible that the minimum spacing requirement is the exact difference between the pitch and the wire width (0.12um in the case above). As a result, any off-grid M2 wire segment will make the area bounded by the orange box shown in the image on the right above unusable for the creation of an M2 wire. The router will not be able to use that portion of the M2 track for connections to other pins. Under such conditions, especially in a highly-dense design, the alignment of routing tracks with I/O pins and standard-cell pins becomes critical. This will be explained in detail in subsequent sections.

16.5.2 Alignment of I/O Pins with Routing Tracks

The I/O pins must be aligned with the routing tracks for optimal quality of result. The following diagram shows a sample routing result of I/O pins that are not aligned with routing tracks. In this example, the non-alignment of the pins with the tracks does not result in any DRC violation but incurs loss of adjacent routing track. However, in some other cases, non-aligned pins can create routing congestion and DRC violations due to non-optimal spacing between off-grid and on-grid wires.

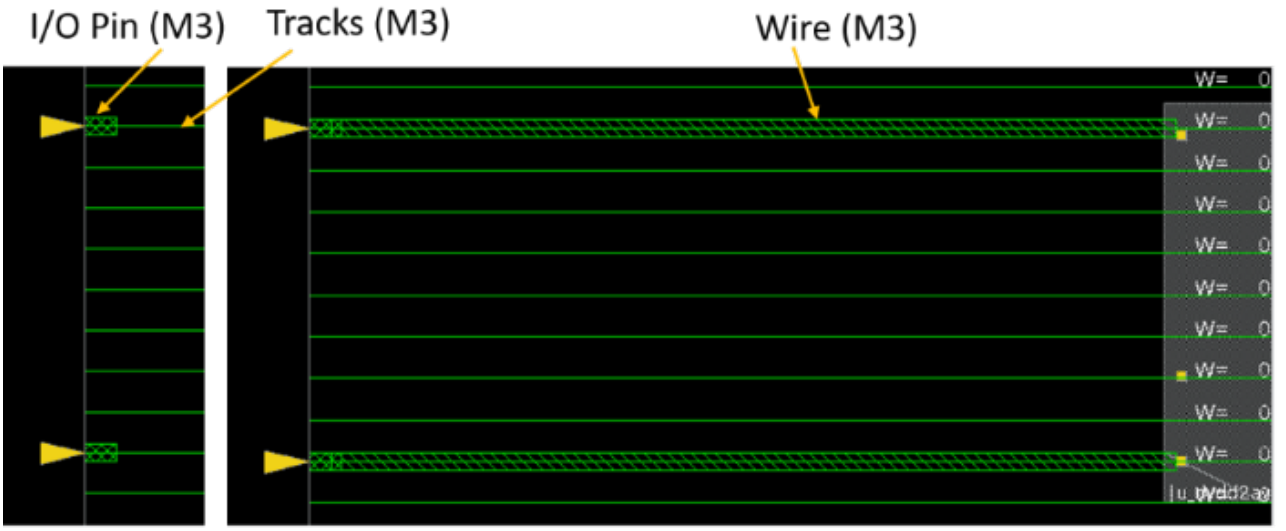


In addition, some lower-technology nodes require wires to be rectangular in shape. In such cases, the non-rectangular shape of the wire above will be considered as a DRC violation.

To avoid such issues in a design imported from Virtuoso into Innovus, the designer must have the desired routing track pattern or WSP in mind if the I/O pins are to be created in Virtuoso and the pins must be aligned with these tracks (WSPs). Note that if both the routing tracks and I/O pins are generated by Innovus, there won't be any issue because the I/O pins created by Innovus are always snapped to the routing track.

The following example shows the result of I/O pins aligning with the routing track.

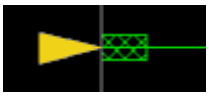
The preferred routing direction of M3 is horizontal in this example. By turning on the *Visibility* check box for only *Pref Track* on the control panel, you can view and analyze tracks more easily.





The yellow triangles in the design are symbols displayed in Innovus to help you locate the I/O pins and to determine their direction. A yellow triangle with its tip pointing in represents an input port, whereas a yellow triangle with its tip pointing out represents an output port. Bi-directional input/output ports are represented with yellow diamonds, instead of a triangles.

As the tip of the triangle is at the centerline of the rectangular pin shape, the alignment of the I/O pin with the track and can be determined easily by checking whether the tip touches the track.



Innovus also provides the `check_tracks` command to check for off-grid pins. Here's an example from a sample report showing off-tracks I/O pins in the design:

```
#Detail Summary of Off-Track Pins:

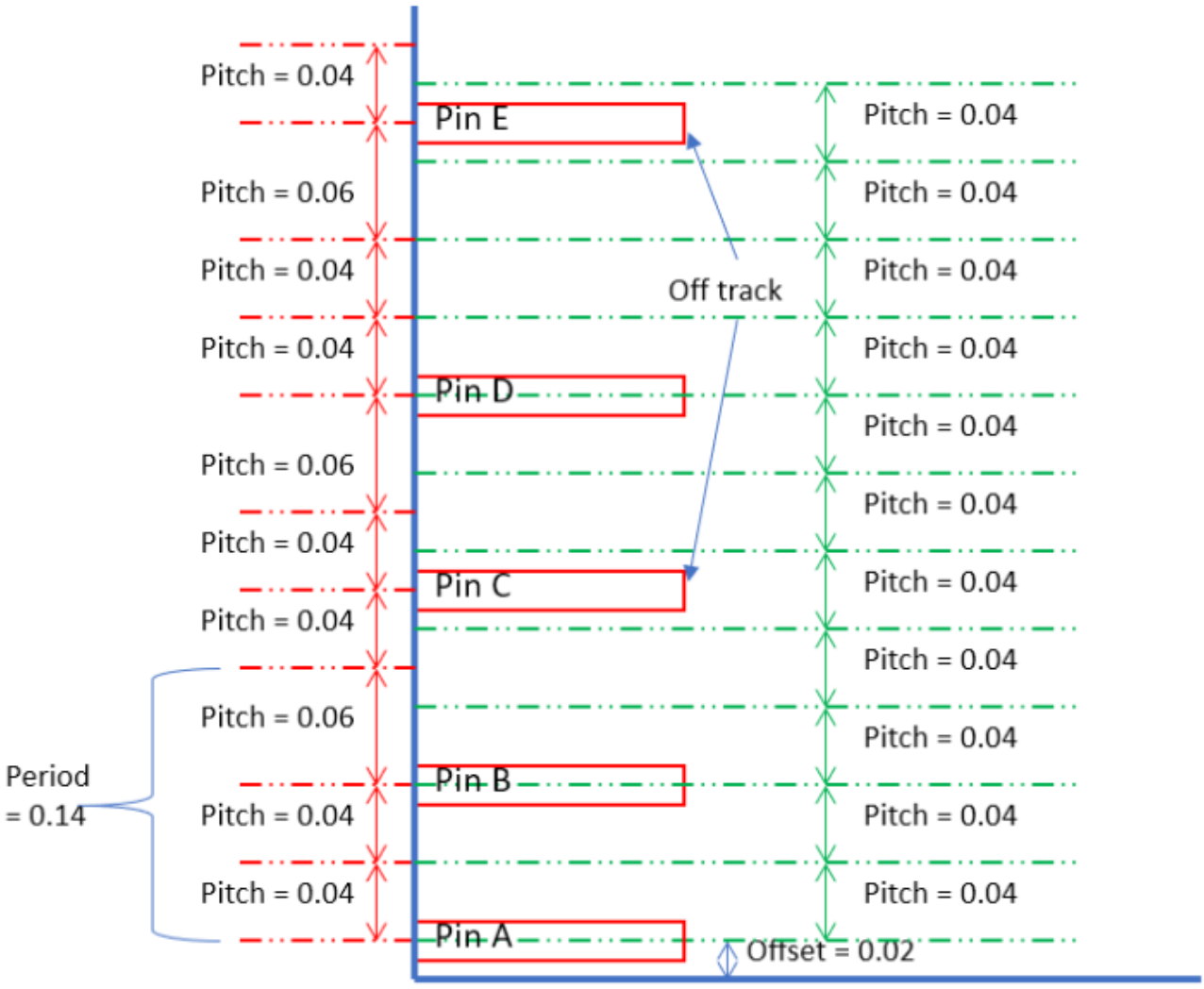
#Off Track Top Level Pin out3

#Off Track Top Level Pin out2

#Off Track Top Level Pin out16
```

16.5.3 Alignment of I/O Pins with Non-Uniform-Pitch Tracks in Lower Technology Nodes

In lower technology nodes, I/O pins may be placed so as to follow a specific track pattern with non-uniform spacing (or pitch). If a cell view does not contain a WSP that can be converted to routing tracks, then Innovus by default creates routing tracks with uniform pitch. This will result in misalignment between certain I/O pins and routing tracks. The image below illustrates the issue.

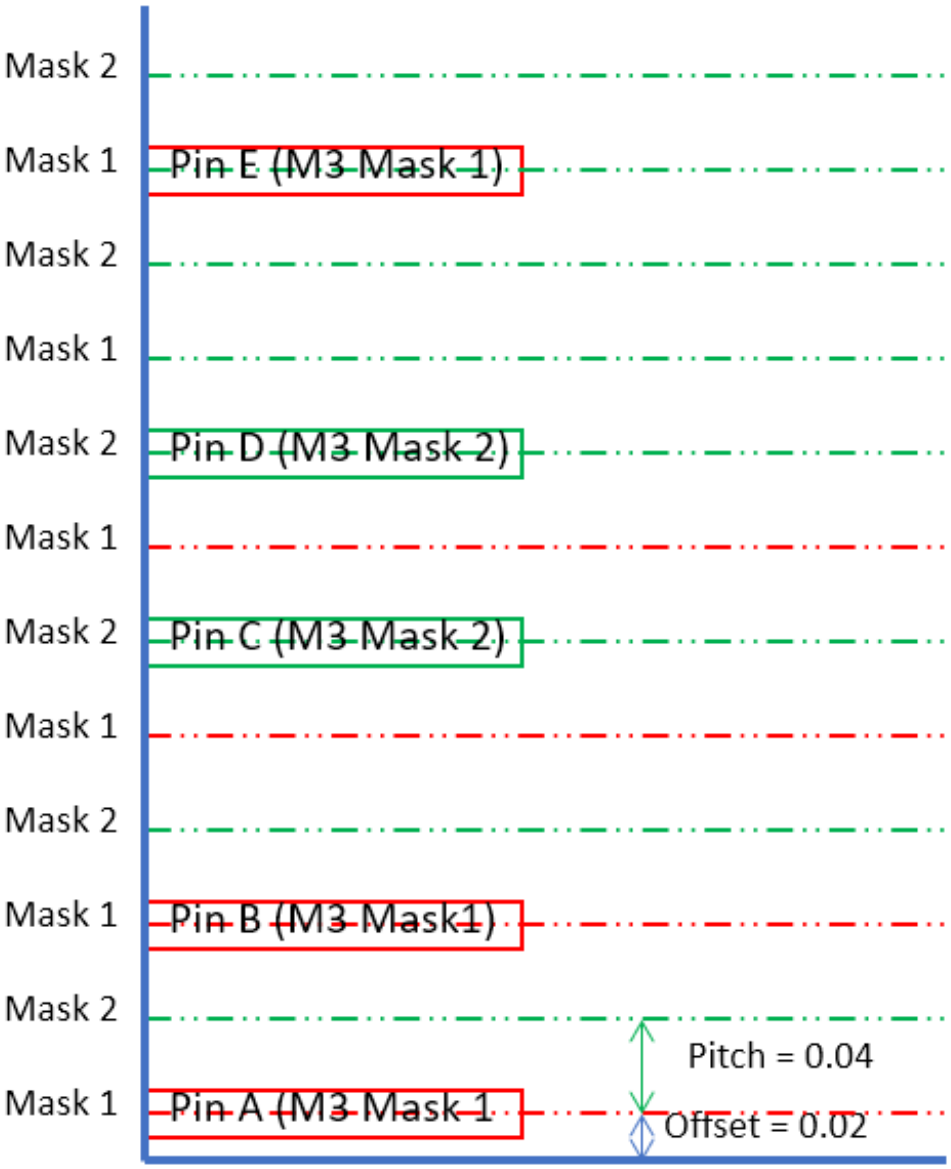


If the routing is not dense and pins are spread apart widely, the router may be able to route to the pins successfully with off-grid wires and no DRC violation. However, if the area is highly congested, routability issues may arise. In such a case, you can use the `add_tracks` command to specify a track pattern with non-uniform pitch. Refer to the Track pattern with non-uniform pitch subsection for more details and examples.

16.5.4 Access to I/O Pins with Double Patterning Layers in Lower Technology Nodes

For I/O pins that are assigned to metal routing layers that use double patterning, the router has to add wires of the right mask number to make a connection to the I/O pins. You can use the `-mask_pattern` option of the `add_tracks` command to specify the mask order of the routing track pattern. For example, for the following diagram, you can specify the mask order as `M3 1 2` to guide the router.

Note that specifying `-mask_pattern` is optional as the router is able to determine the mask order automatically in most cases. Nevertheless, it is a good practice to guide the router by specifying the mask order if this piece of design information is already known.

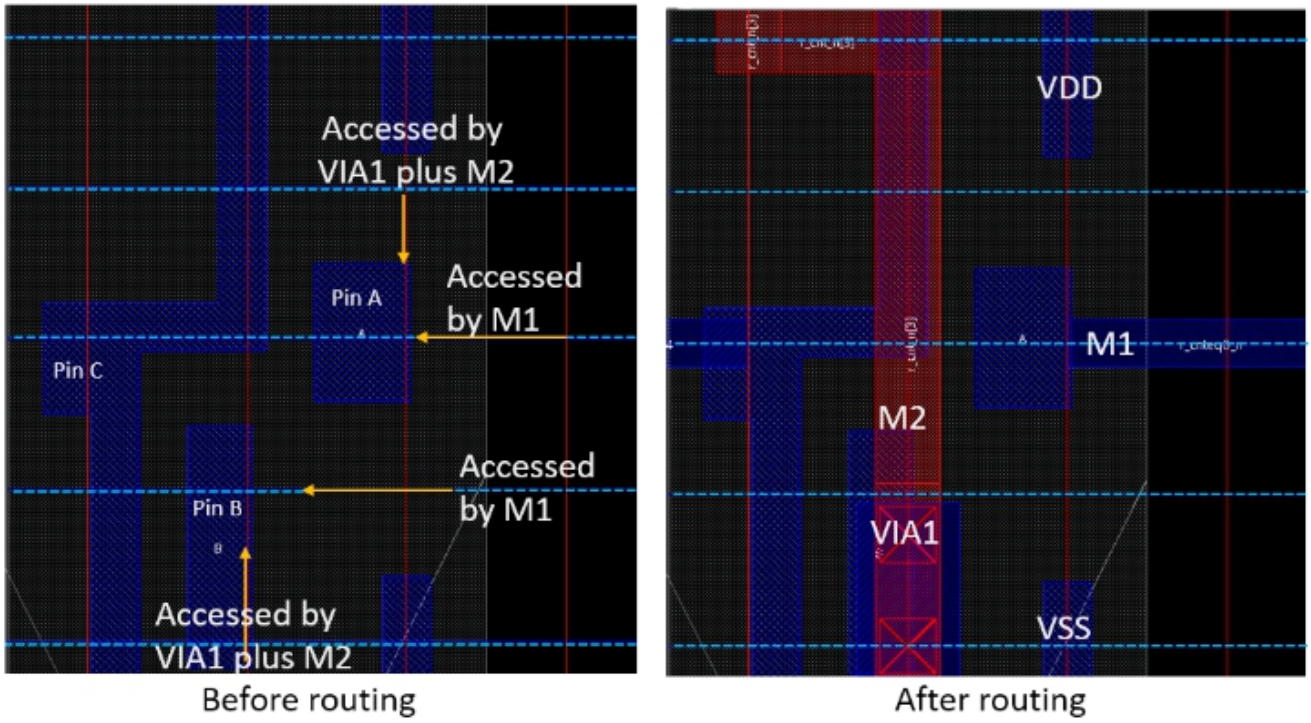


16.5.5 Access to Standard-Cell I/O Pins in Lower Technology Nodes

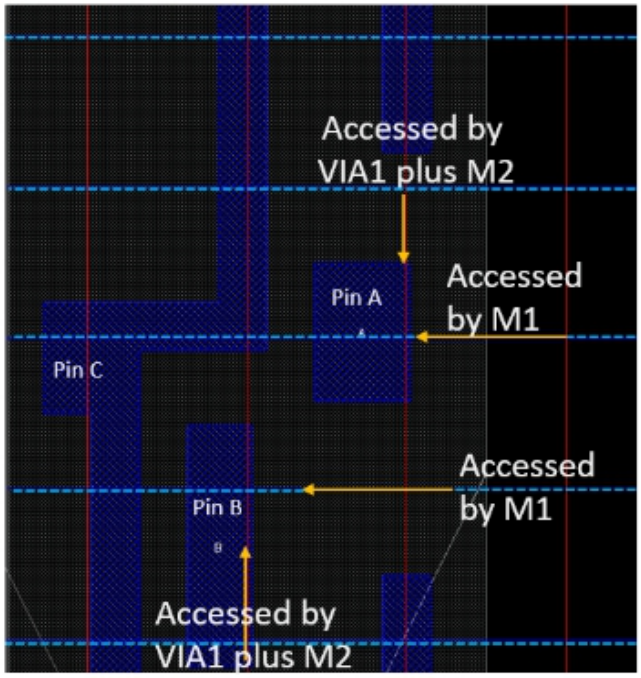
In addition to the I/O pins of the design, the routing track pattern also affects the quality of the routing result on the standard-cell I/O pin connections. While creating routes to make a connection to the I/O pins of a standard cell, a router can either add a wire to the same metal layer as the I/O pins or drop a via onto the I/O pin geometries. If a standard-cell I/O pin has no or few intersections with the routing track, it may create a DRC violation or block the routes to another pin, causing an open. This kind of problem is called a standard-cell pin access issue.

Consider the 'Before routing' image on the left hand side below. Pin B and Pin A can be accessed

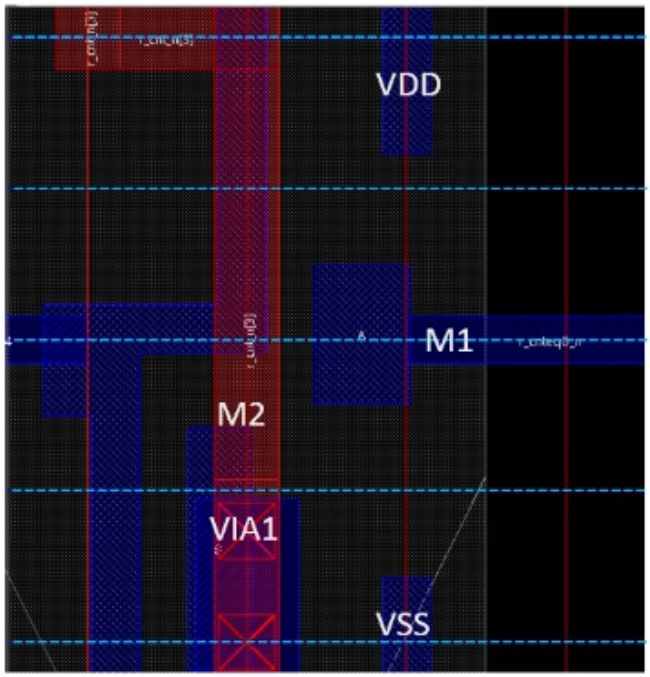
by the M1 wire or the VIA1 via with the routing tracks (blue dashed lines) generated at right locations. The image on the right shows one possible routing result. The router connects Pin B through VIA1 and the M2 wire and connects Pin A with the M1 wire.



To illustrate how the routing track pattern affects the routing result, the image on the left below shows the same standard cell with the routing tracks for M1 not created at the right locations. For Pin B, if the router adds M1 from the right by following either of the two horizontal tracks from the bottom, it will create a short to the other pins. As a result, Pin B will mostly be accessed only by the VIA1 via. NanoRoute can create an off-grid M1 wire to access Pin B, but this may mean the neighboring M1 track near the pin cannot be used for other nets to prevent DRC violations, such as a spacing violation. The image on the right below shows a sample routing result. The router chooses to connect Pin A with VIA1 because the router prefers to route on the grid. In this case, the routing congestion is low, so there is no DRC violation. However, if many standard cells with high-pin density are placed close to each other, poorly located routing tracks may lead to routing issues.

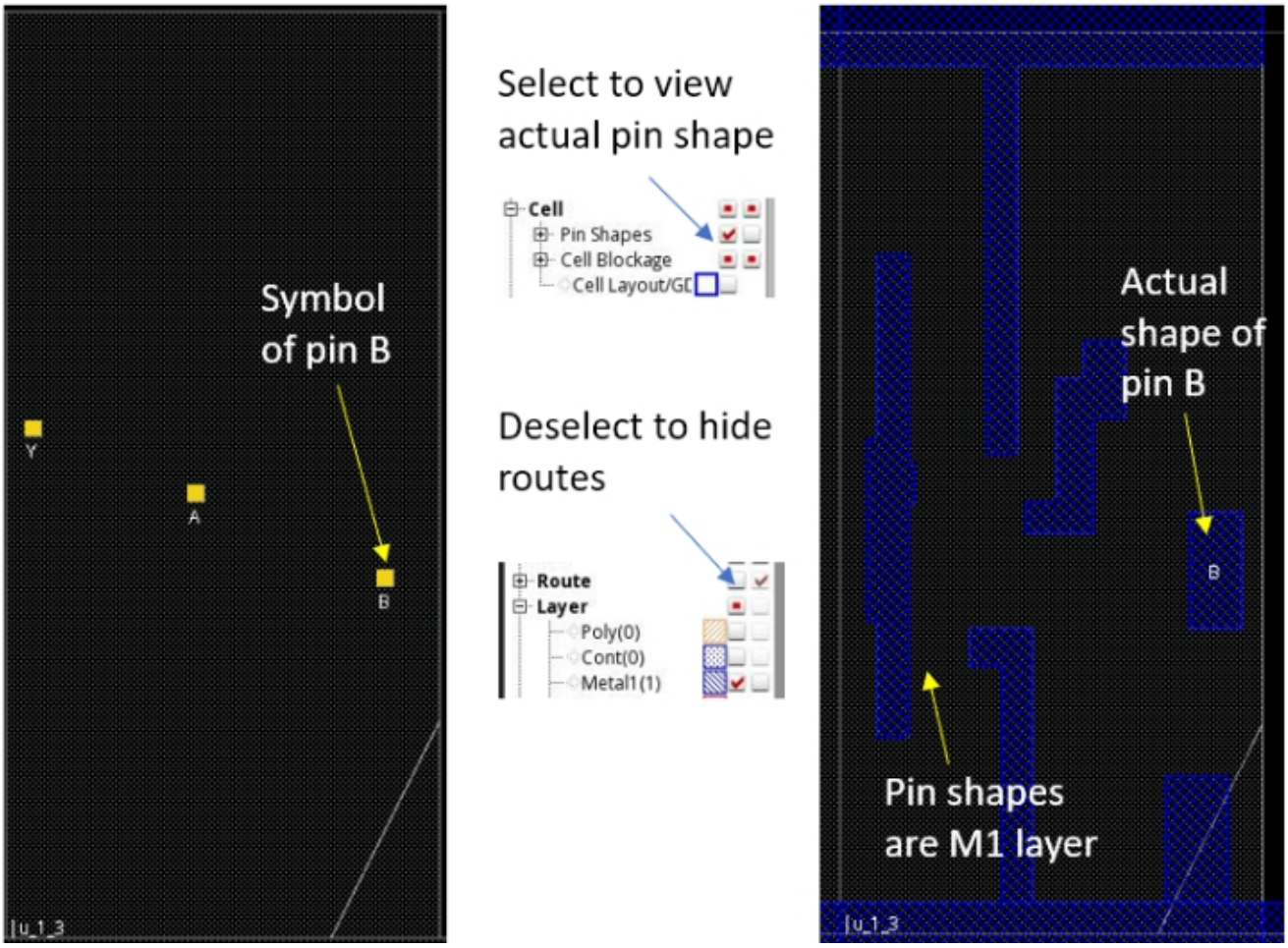


Before routing



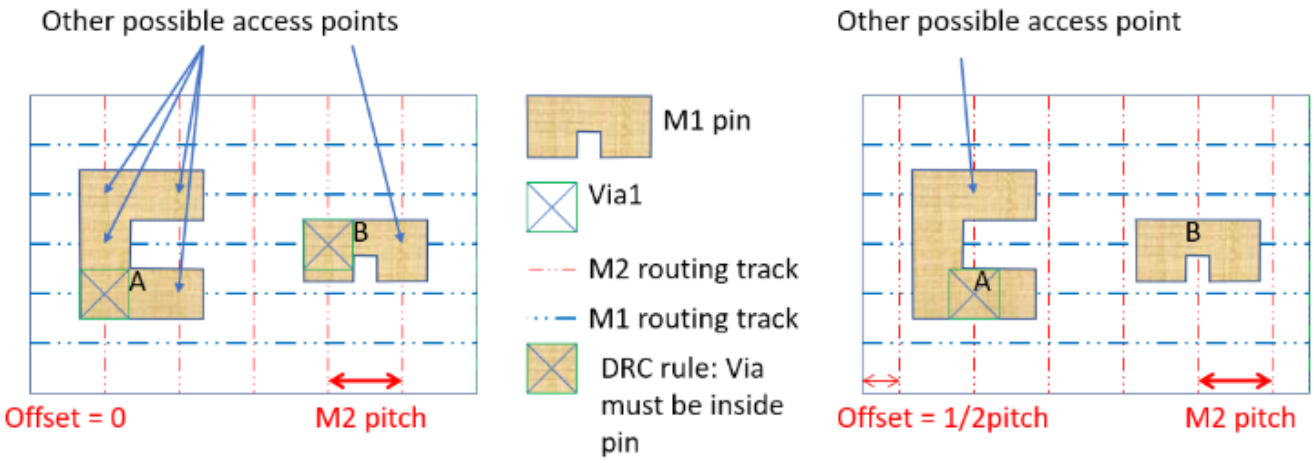
After routing

By default, Innovus does not display the pin shape of a standard cell. The yellow square(s) inside the standard cell is not a real physical object but just a symbol to indicate that there is an I/O pin. To view the actual pin shapes, select the *Visibility* check box of *Pin Shapes* on the control panel. Make sure the *Visibility* check box of the metal layer that the I/O pins use is also selected . For example, *Visibility* check box of *Metal1* should be selected if the pin is of layer Metal1. If the design is routed, deselect the *Visibility* check box of *Route* so that the wires are hidden and the pin shapes can be seen more clearly.



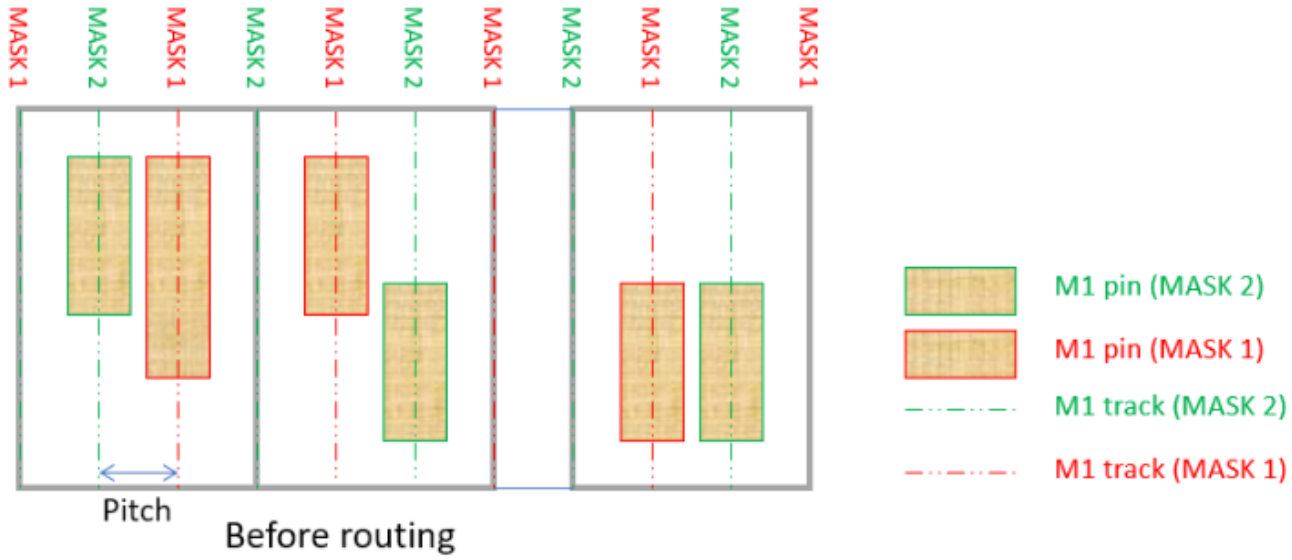
In the above case, any point where the routing tracks touch the pin shape geometries indicates a possible location for the router to insert a via to make a connection. In some nodes, there could be a design rule requirement that the vias inserted by the router must be fully inside the geometries of the I/O pin of the standard cell. In some cases, the I/O pins of these standard cells are designed in a way that the pins require routing tracks of a specific routing pitch and offset. If the routing tracks are not created with the right offset, it may become more difficult for the router to find the right solution for accessing the pins. As a result, the routing result may not optimal or the run time may increase significantly.

As an example, consider the image on the left below. If the right track pattern is created, there are five possible access points for the router to insert a via for Pin A and two possible access points for Pin B. However, if the offset is not specified correctly, resulting in the generation of poorly located routing tracks, the router would find less preferable locations to access the pins. Though it is possible to create an off-grid wire, the router either needs to spend more time to find a good solution or settle for a less optimal quality of result.

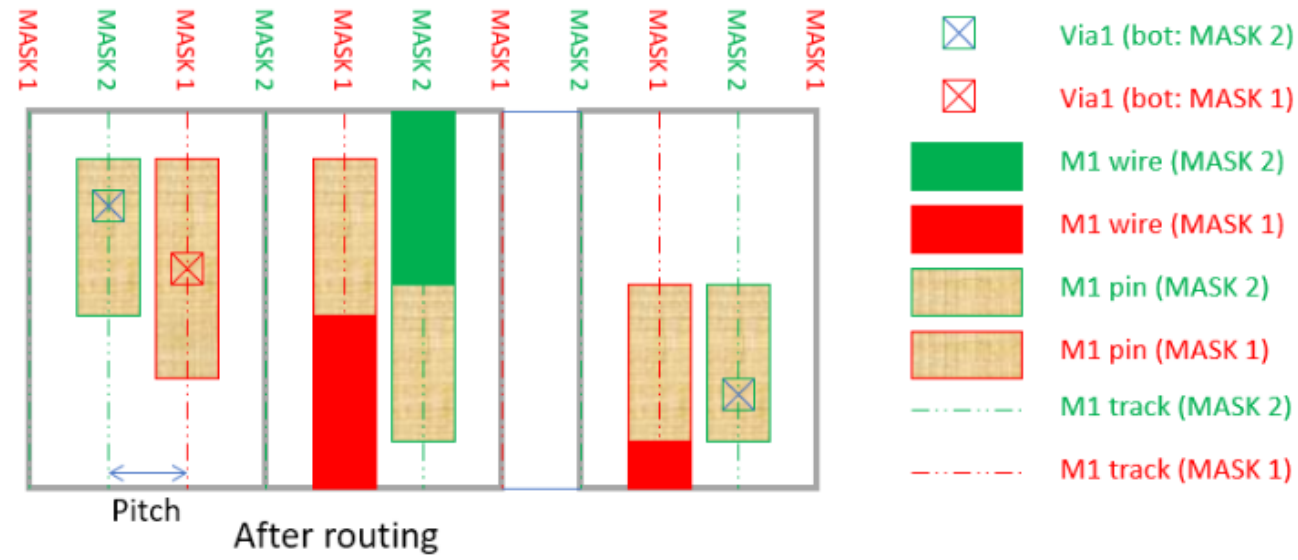


16.5.6 Access to Standard-Cell I/O Pins with Double Patterning Layers

In some lower technology nodes that use double patterning technology, the bottom few metal routing layers typically carry two masks. As most standard-cell I/O pins are implemented with one or two bottom metal routing layers, the routing tracks for these layers can also be differentiated with two different masks. When the mask pattern (or mask order) of the routing tracks are defined using the `add_tracks` command, ensure that the mask pattern is in the right order so that the mask number of M1 track matches the mask number of the standard cell pins.



The above diagram shows three standard cells and in each standard cell, each pin has its own mask number. Let us assume all the standard cells in this design are manually placed in Virtuoso and the cell view is loaded into Innovus. If the offset, pitch and mask pattern are specified correctly, the router in Innovus can add wires or vias with the right mask number and at the right location to access the pin easily as shown in the diagram below.

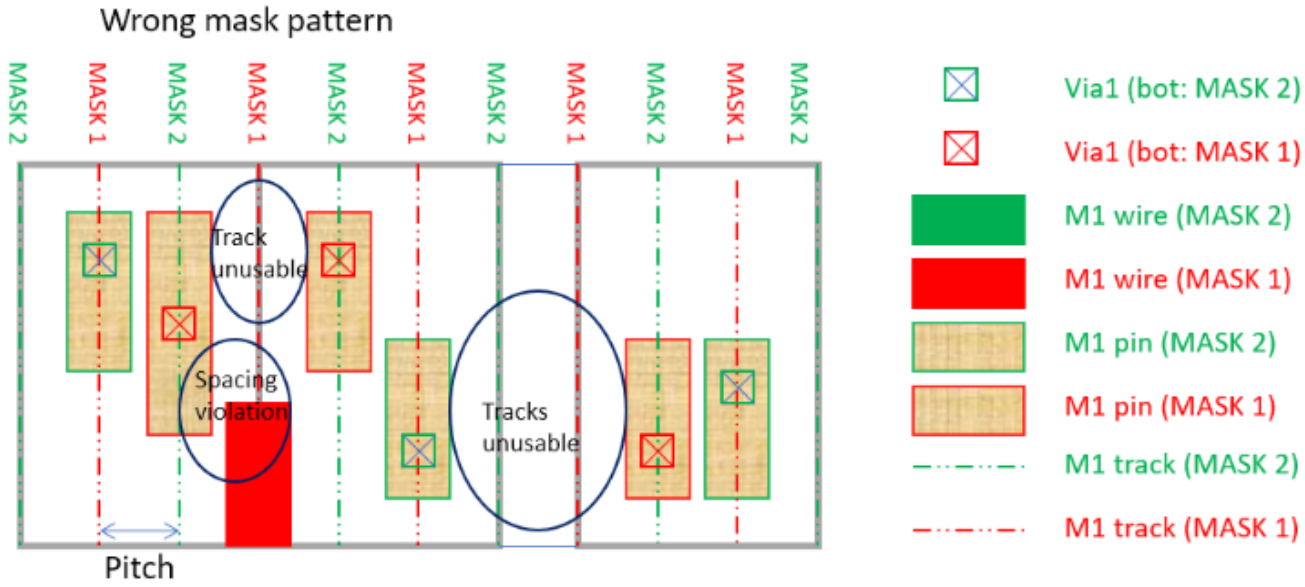


Let us assume the `add_tracks` command for the above case is specified correctly as follows:

```
add_tracks -mask_pattern {M1 1 2 ...} -offsets {M1 0 ...}
```

What would happen if the mask pattern is specified with the wrong order (reverse order) as follows?

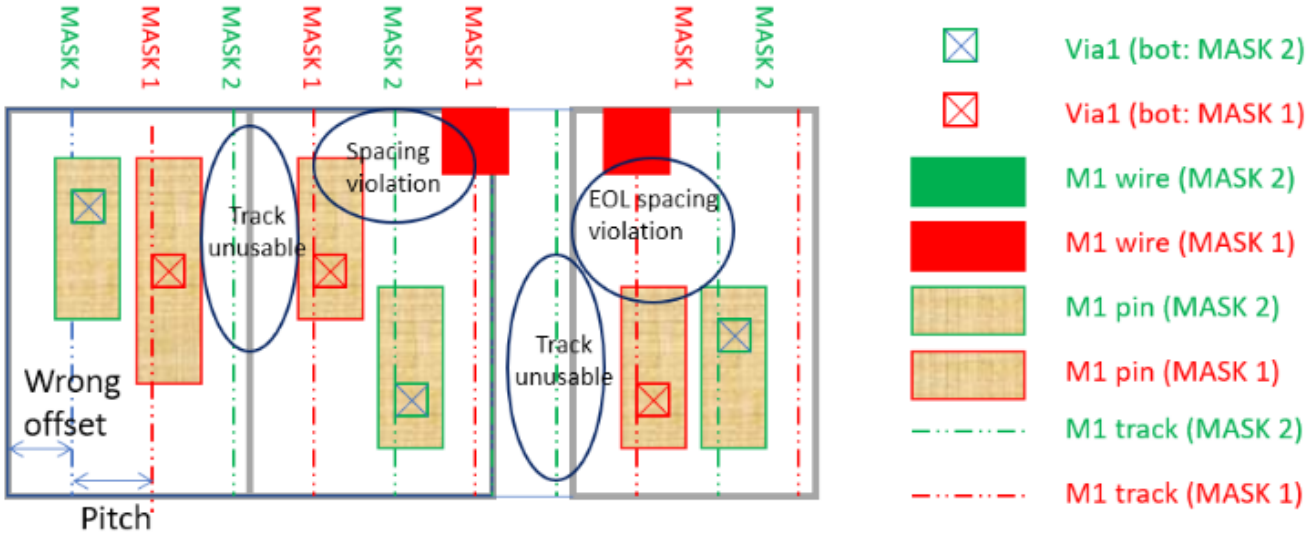
```
add_tracks -mask_pattern {M1 2 1 ...} -offsets {M1 0 ...}
```



Most likely, the incorrect mask pattern specification would result in the router being unable to add wires to make connection with the pin and would restrict the router's solution space to just via insertion for pin access. If the design rule requires the geometries of the same mask to be centered two track distance apart, it could also cause a spacing violation between the M1 pin and the M1 wire of the same mask number and make the track segments close to the pins unusable for routing.

What if the mask pattern is specified with the correct order but incorrect offset as follows?

```
add_tracks -mask_pattern {M1 1 2 ...} -offsets {M1 0.4 ...}
```



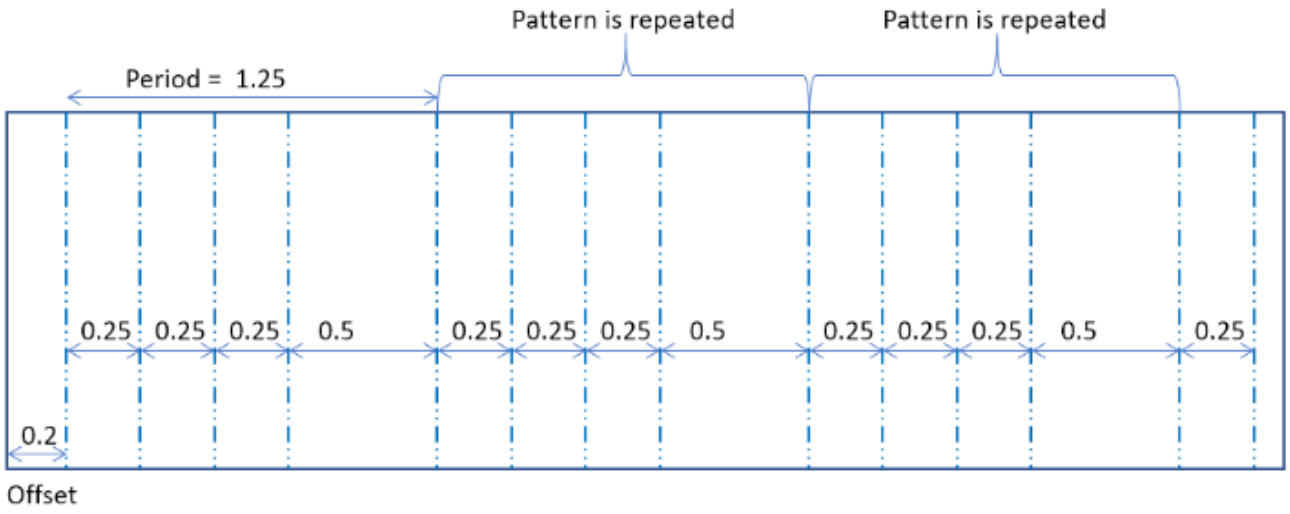
The incorrect offset specification may similarly make certain segments of tracks unusable for routing and may restrict the router's solution space to only via insertion for pin access. It may create a spacing and End-of-Line (EOL) spacing violations between geometries of the same mask number.

To summarize, in some lower technology nodes, the routing tracks are expected to be perfectly aligned with the pins with the right mask number. This is based on the concept of "correct by construction" in a way that as far as the routes are created on-grid, the chance of achieving a good routing result is high.

16.5.7 Track Patterns with Non-Uniform Pitch

In lower technology nodes, the designer may prefer a routing track pattern of non-uniform pitch. For example, suppose that the M4 metal routing layer has a vertical preferred routing direction and its routing track is desired to be a repeated pattern of four tracks with default wire width (specified in Tech LEF/LefDefaultRouteSpec) and non-uniform spacing values as shown in diagram below. The equivalent command in Innovus is:

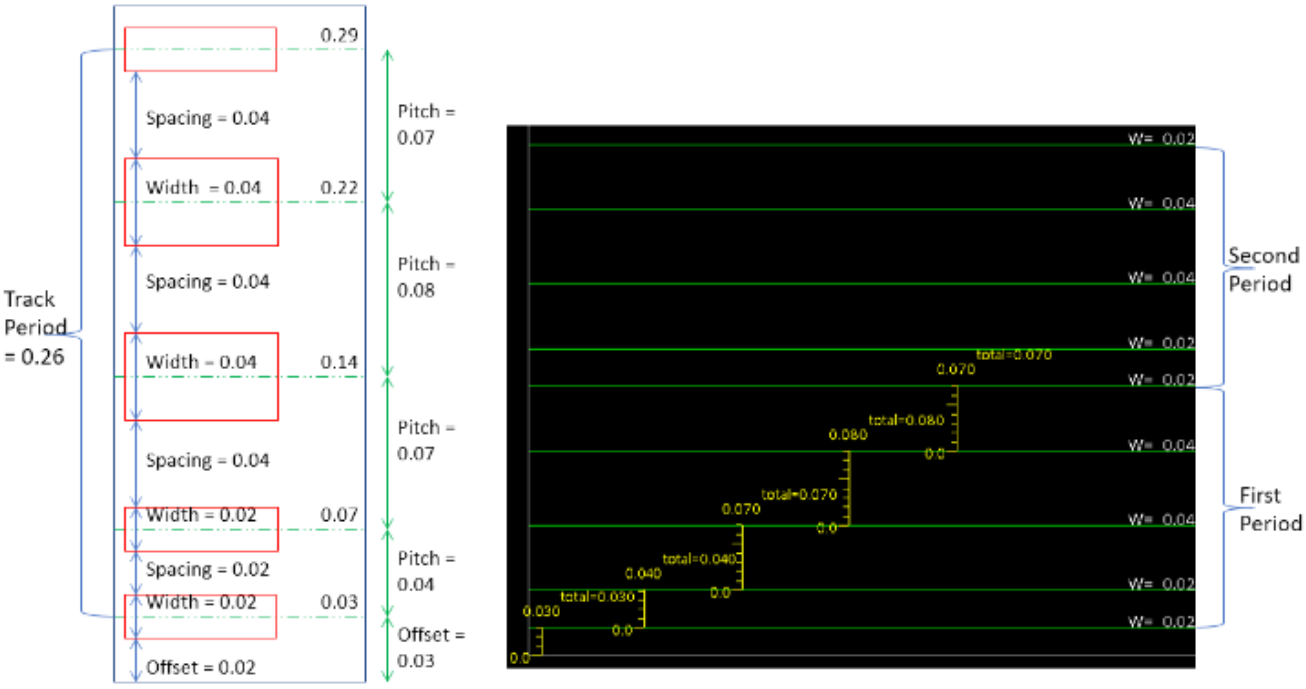
```
add_tracks -pitch_pattern {M4 offset 0.2 {pitch 0.25 repeat 3} pitch 0.5}
```



16.5.8 Track Patterns with Non-Uniform Pitch and Non-Default Width

The `add_tracks` command also allows you to specify routing tracks with width and pitch pattern. Consider a WSP for M3 with the offset value, wire widths and spacing requirements between wires as shown in the following diagram. The equivalent command in Innovus is:

```
add_tracks -width_pitch_pattern {{M3 offset 0.03 width 0.02 pitch 0.04 width 0.02 pitch 0.07 width 0.04 pitch 0.08 width 0.04 pitch 0.07}}
```



In the above example, note that:

- In the right-side image, the width values (`W = 0.02` or `W= 0.04`) is displayed for each track.
- The period offset in WSP is the distance between the bottom edges of the PRBoundary object and the first wire of the pattern. In Innovus, by default, the offset is the distance between the centerline of the first wire above the bottom edge of the Core Boundary and the bottom edge of the Core Boundary for horizontal tracks. Note that the size of the Core Boundary does not necessarily need to be the same as the size of PRBoundary object in Virtuoso. Refer to the Design and Core Boundary in Innovus chapter to understand how the Core Boundary can affect the creation of rows and routing tracks in Innovus.
- In WSP, if the spacing value is specified as the edge-to-edge mode, the spacing values at each interval should not be mistaken as the routing pitch in the `add_tracks` command. Note that the pitch is measured between the centerline of wires.

17

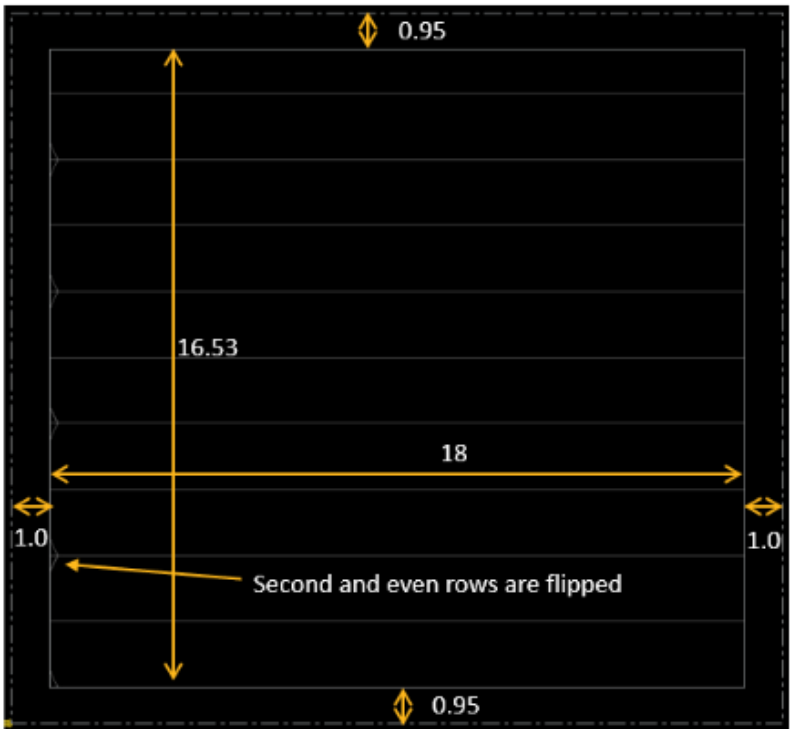
Design and Core Boundary in Innovus

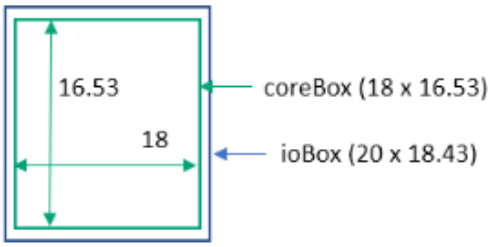
- Design and Core Boundary Definitions
- Matching the Orientation of Rows and Standard Cells
- Handling Gaps Between Standard Cells and the Design Boundary
  - Gap Between the Left Edge of the Leftmost Standard Cell and the Design Boundary
  - Gaps Between the Left and Bottom Edges of Standard Cells and the Design Boundary
- Creating Double-Height Rows
- Creating Routing Tracks with Respect to coreBox
- Resizing the PRBoundary Object in Virtuoso

17.1 Design and Core Boundary Definitions

In Innovus, the design boundary or the IO Box is the actual design boundary. In contrast, the core boundary or the Core Box is where core rows are created and standard cells normally placed. Typically, for a digital block that has its floorplan implemented in Innovus, the IO Box and Core Box will be different in size. For example, the following command in Innovus creates a Core Box of width `18` and height `16.53` with a margin of `1` from the left and right edges of the Core Box to the left and right, respectively of the IO Box and a margin of `0.95` from the bottom and top edges of the Core Box to the bottom and top of the IO Box.

```
floorPlan -site CoreSite -s {20.0 18.43 1.0 0.95 1.0 0.95} -flip s
```





Within the Innovus database, the design boundary and core boundary are floorplan properties – database objects named as `ioBox` and `coreBox`. You can either measure the size of `ioBox` and `coreBox` in the layout window or use the `dbGet` command to query the `x` and `y` coordinates of the `ioBox` and `coreBox`.

```
> dbGet top.fPlan.ioBox

{0.0 0.0 20.0 18.43}

> dbGet top.fPlan.coreBox

{1.0 0.95 19.0 17.48}
```

If the above cell view is saved and loaded into Virtuoso, SKILL code can be run in the Command Interface Window (CIW) to query the box properties.

For example, when the following is entered in CIW,

```
cv=geGetEditCellView()

cv~>prBoundary~>bBox
```

the system should return:

```
((0.0 0.0)

    (20.0 18.43)

)
```

Enter the following in CIW:

```
cv~>prBoundary~>ioBox
```

The system should return:

```
((0.0 0.0)

    (20.0 18.43)

)

cv~>prBoundary~>coreBoxSpec

 (((1.0 0.95)

      (19.0 17.48)

    ) nil 0.0 "evenRowSpacingType" "oddRowFlipType"

    db:0x2844419a

)
```

In Virtuoso, the PRBoundary object helps Innovus to determine the design boundary. For a design whose floorplan is created in Virtuoso (i.e. PRBoundary object is drawn in Virtuoso), the Core Box and IO Box have the same shape and size when queried in Innovus.

For example, assuming the same size of the PRBoundary object, when the following is entered in CIW,

```
cv=geGetEditCellView()

cv~>prBoundary~>bBox
```

the system returns

```
((0.0 0.0)

    (20.0 18.43)

)

cv~>prBoundary~>ioBox

nil

cv~>prBoundary~>coreBoxSpec

nil
```

When running the following queries in Innovus, the same result is obtained for `ioBox` and `coreBox` objects.

```
> dbGet top.fPlan.ioBox

{0.0 0.0 20.0 18.43}

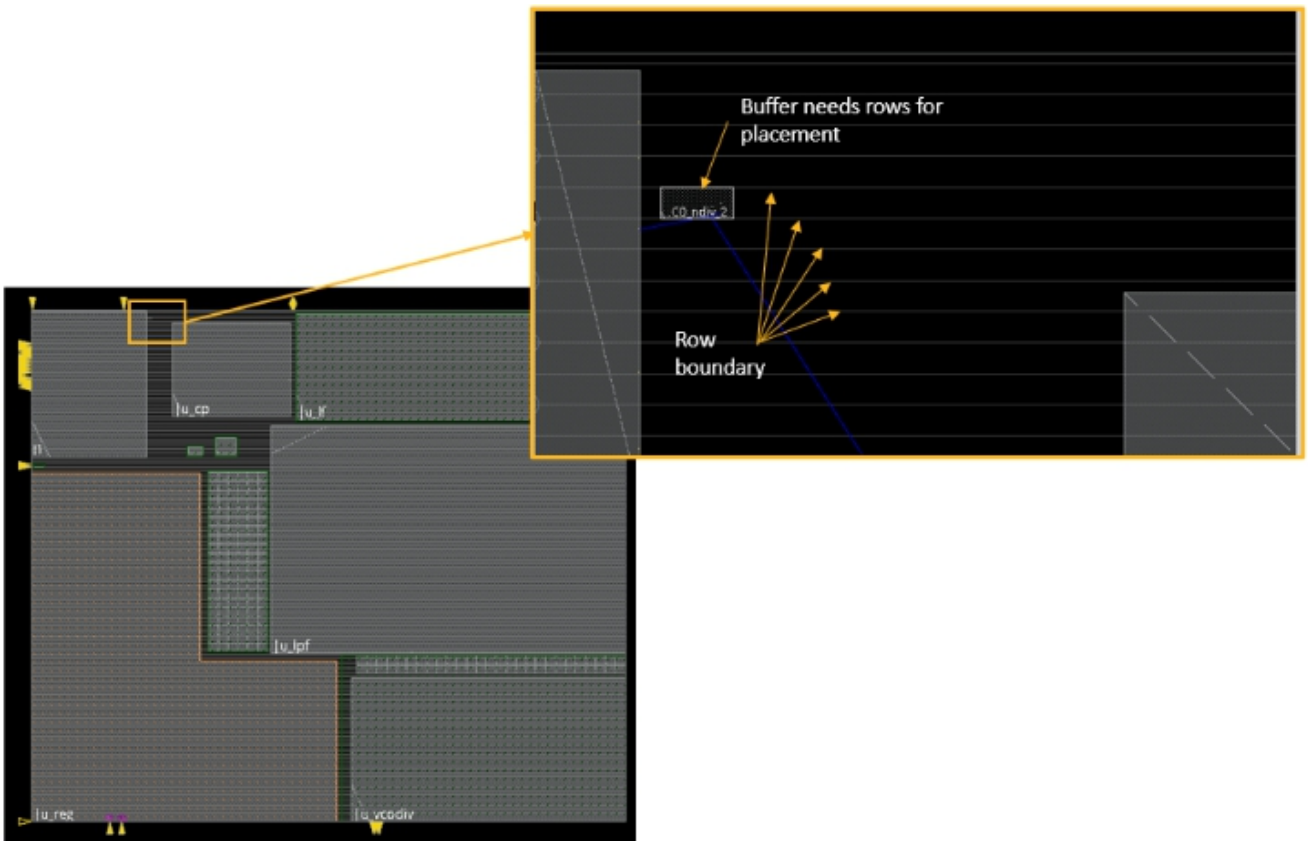
> dbGet top.fPlan.coreBox

{0.0 0.0 20.0 18.43}
```

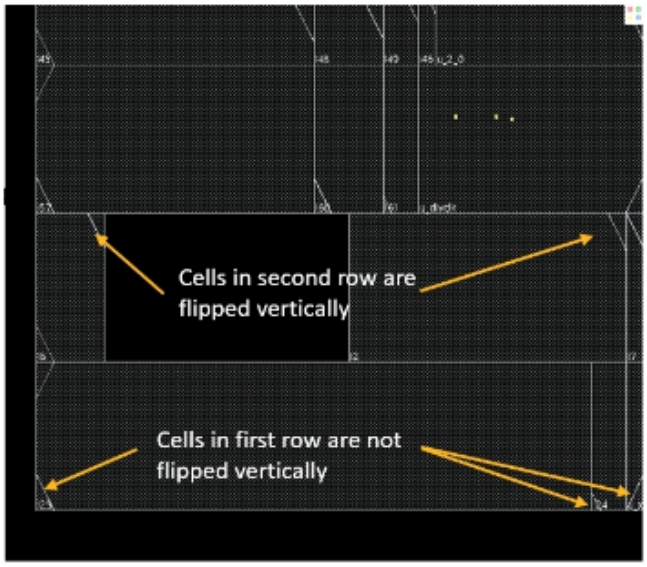
The Core Box affects how the generated rows and routing tracks are physically located in the design. The details of how Innovus creates the rows and tracks will be described in detail in the next few sections.

17.2 Matching the Orientation of Rows and Standard Cells

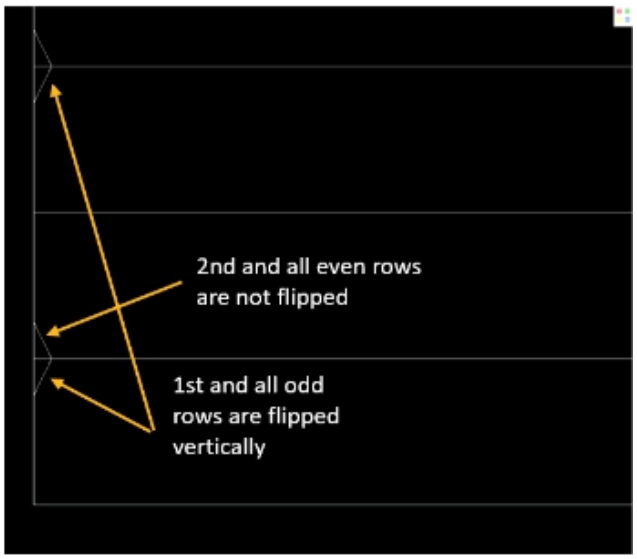
Consider a design that is initially implemented in Virtuoso and has no rows. When this cell view is loaded into Innovus, Innovus will automatically add rows to it. By default, the rows added by Innovus have the first and subsequent odd rows flipped vertically. Rows are created by Innovus for placement-type applications, such as buffer insertion for making an ECO change.



The above design is mainly a top-level design with only macros. In such designs, it does not matter whether the first or second row is flipped. However, for a design filled with only standard cells but no rows created in Virtuoso, some additional steps may be needed in Innovus to ensure that the orientation of the rows created by Innovus matches the orientations of the standard cells.



Display of instances and rows

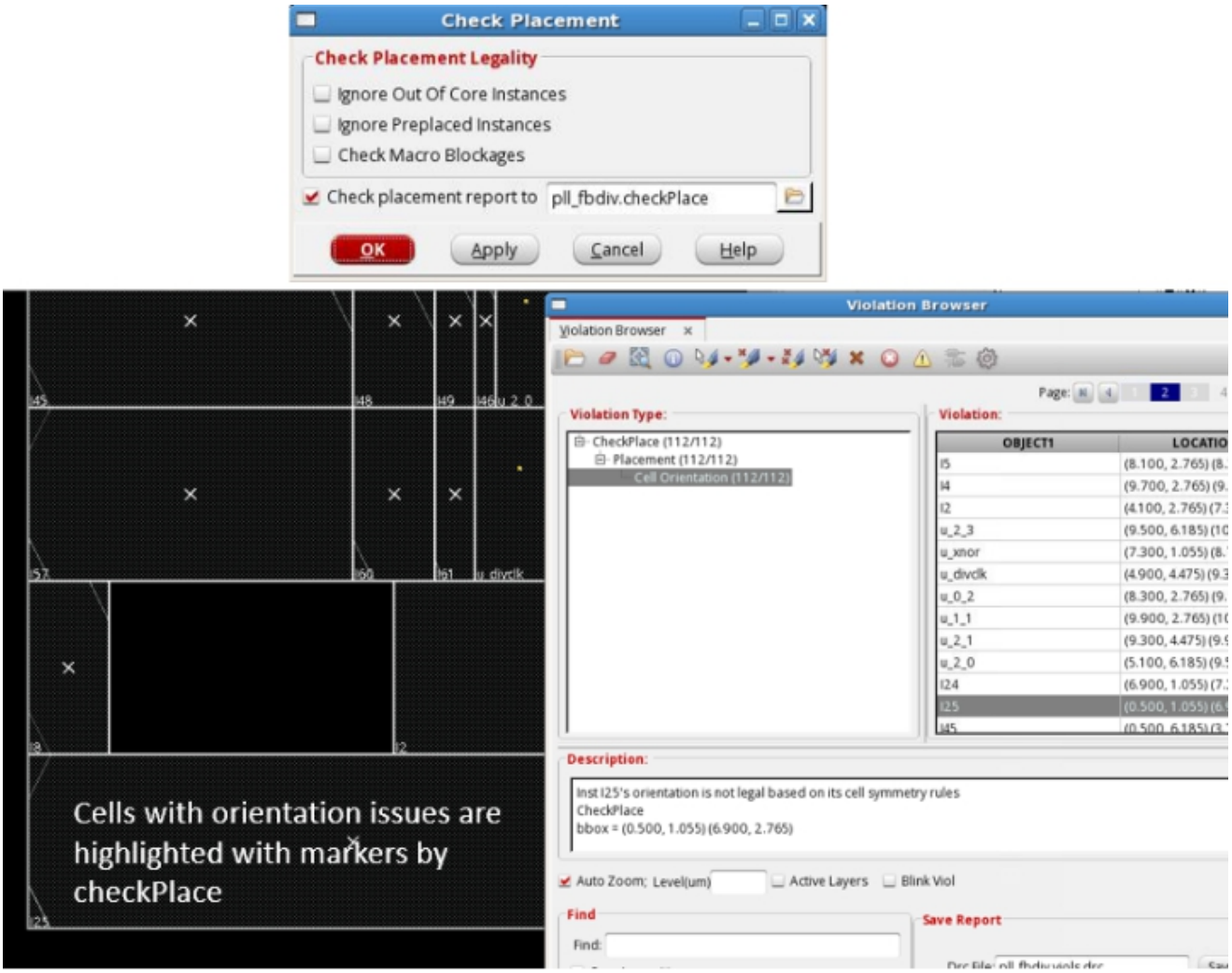


Display of instances is turned off to view row orientations

In the above case, the standard cells are manually placed in Virtuoso in a way that the cells of the first and subsequent odd rows are not vertically flipped while those in the second and subsequent even rows are vertically flipped. As by default, the first and other odd rows inserted by Innovus during `init_design` are vertically flipped, a row versus cell orientation issue arises. If at a later stage, a ECO change needs to be done to insert a buffer, this buffer will have a different orientation from the other cells in the same row.



Cell orientation issues can be detected by running the `checkPlace` command at the Innovus command prompt. Alternatively, you can use the Check Placement form, which can accessed from the *Place* menu. If `routeDesign` is run to perform routing, it will also call `checkPlace` before routing and generate violation markers where applicable.



When cell orientation issues are detected, you can rectify the issue by switching the orientation of the cells using the steps mentioned in the "*Switching the Orientation of the Standard Rows*" section in the Placing Standard-cell-based Designs from Virtuoso chapter.

Another way to remedy the issue is as follows:

1. Delete all rows. At the Innovus command prompt, enter:

   ```
   deleteRow -all
   ```

2. Find out the site property of the standard cell. In the layout window, select the standard cell and at the Innovus command prompt, enter:

   ```
   dbGet selected.cell.site.name
   ```

   Let us assume the systems returns:

   ```
   CoreSite
   ```

3. Get information on the Core Box coordinates. At the Innovus command prompt, enter:

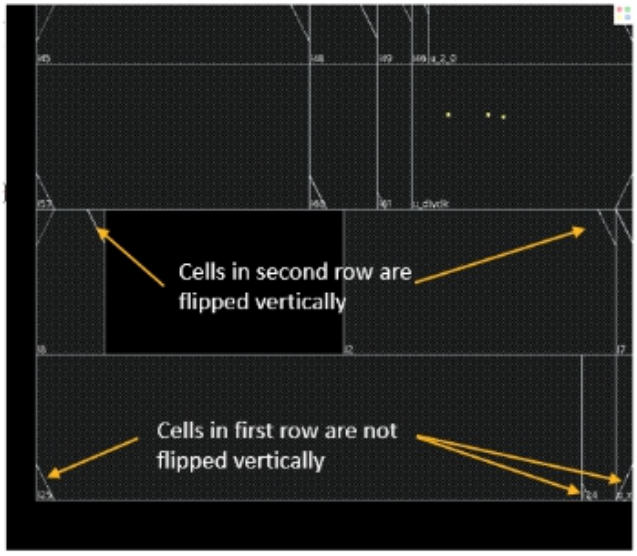
```
dbGet top.fplan.coreBox
```

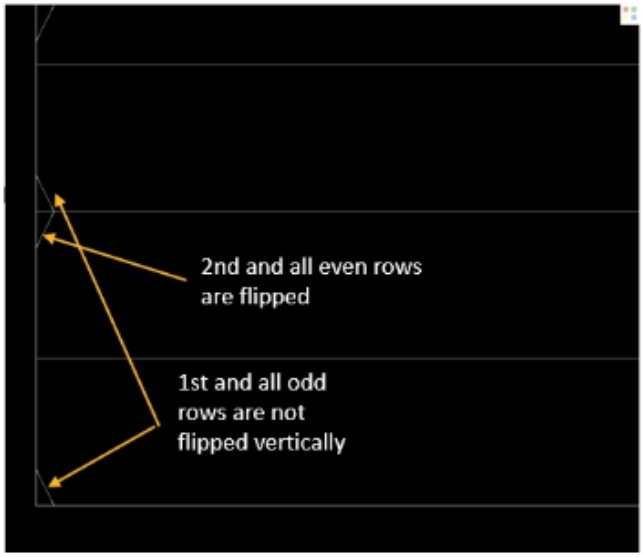
Let us assume the systems returns:

```
{0 0 9.2 9.2}
```

4. Create rows with the correct orientation in the design area. Assuming all the standard cells in the design have the same site name, enter the following command at the Innovus command prompt:

```
createRow -site CoreSite -area 0 0 9.2 9.2
```

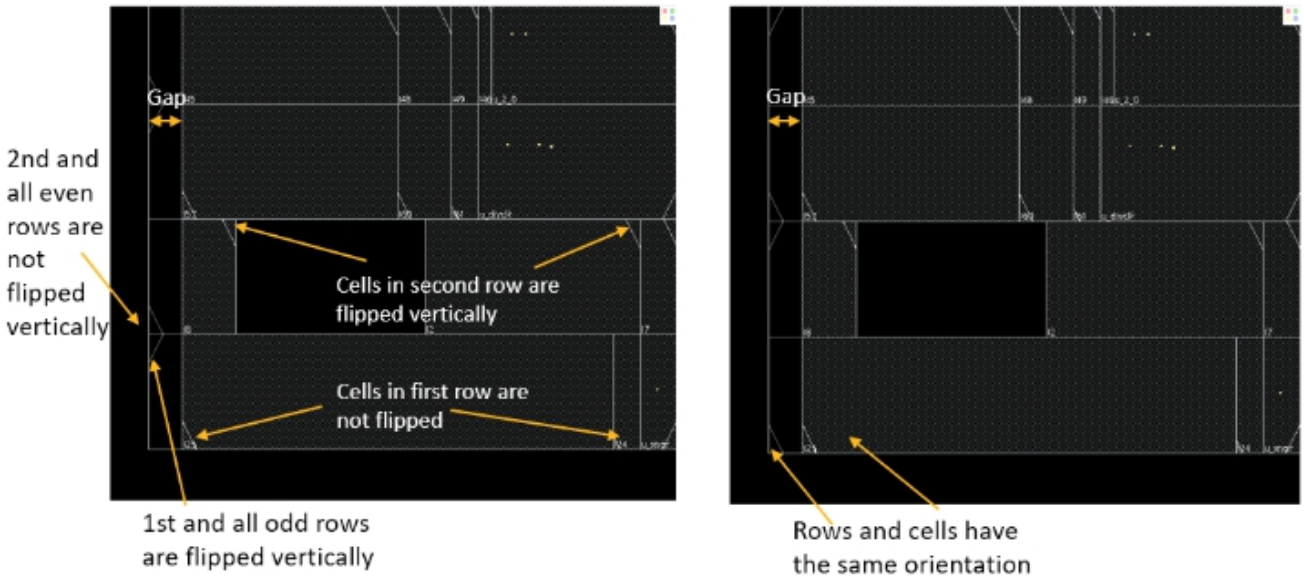




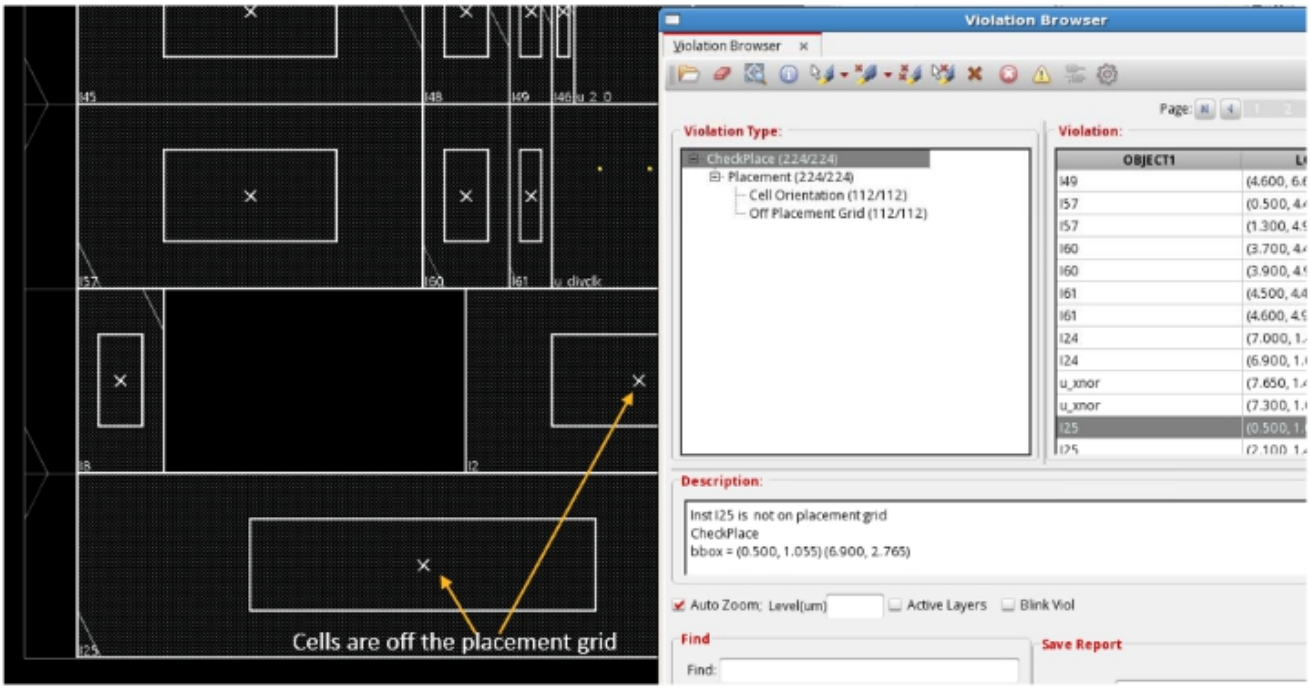
17.3 Handling Gaps Between Standard Cells and the Design Boundary

17.3.1 Gap Between the Left Edge of the Leftmost Standard Cell and the Design Boundary

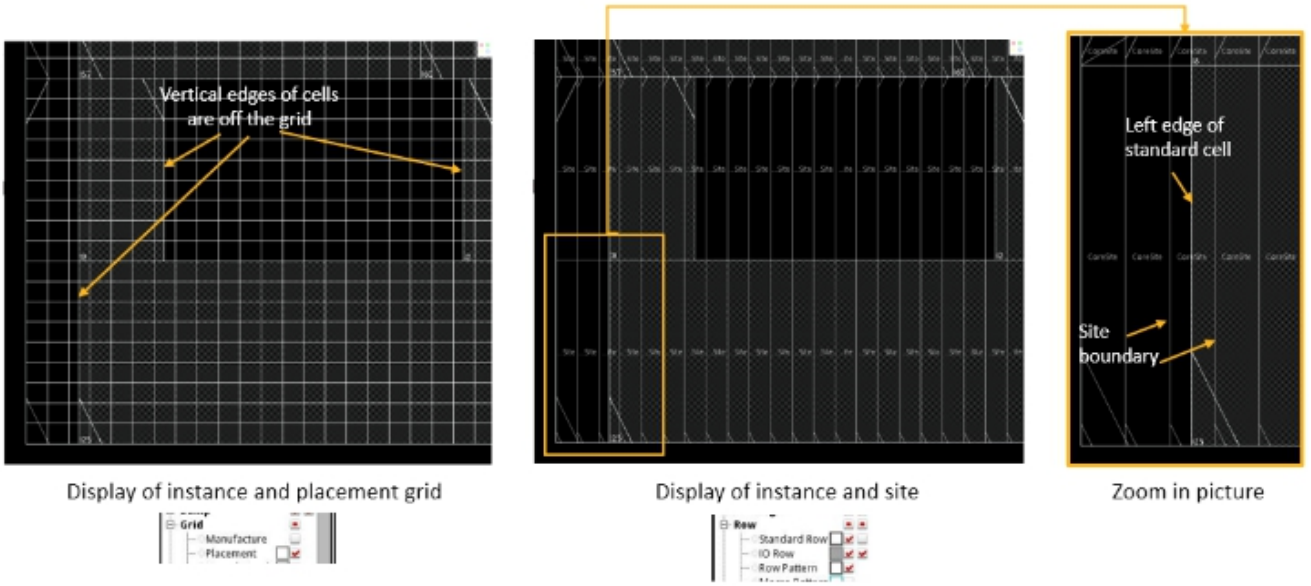
In the case shown below, not only do the manually placed standard cells have a different orientation from the rows created during `init_design`, there is also a gap between the left edge of the leftmost standard cell and the design boundary. However, having rows created with the correct orientation may not be enough if the cells are not snapping to the placement grid formed by the site width.



If `checkPlace` is run, the tool will report *Cell Orientation* and *Off Placement Grid* violations.



To understand what an *Off Placement Grid* violation means, select the *Visibility* check box of the *Placement* grid or *Row Pattern* option on the control panel in the main layout window. This will display the placement grid so that you can easily view whether a cell is placed on or off the grid.

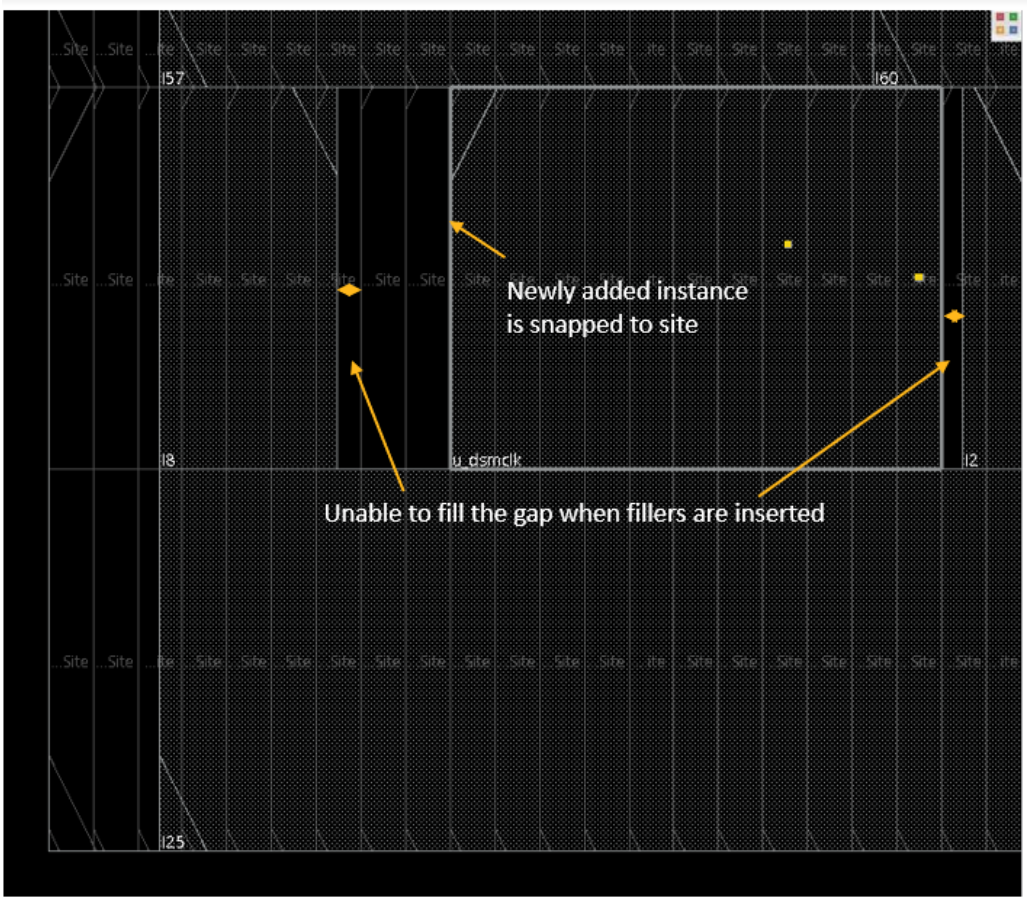


Display of instance and placement grid

Display of instance and site

Zoom in picture

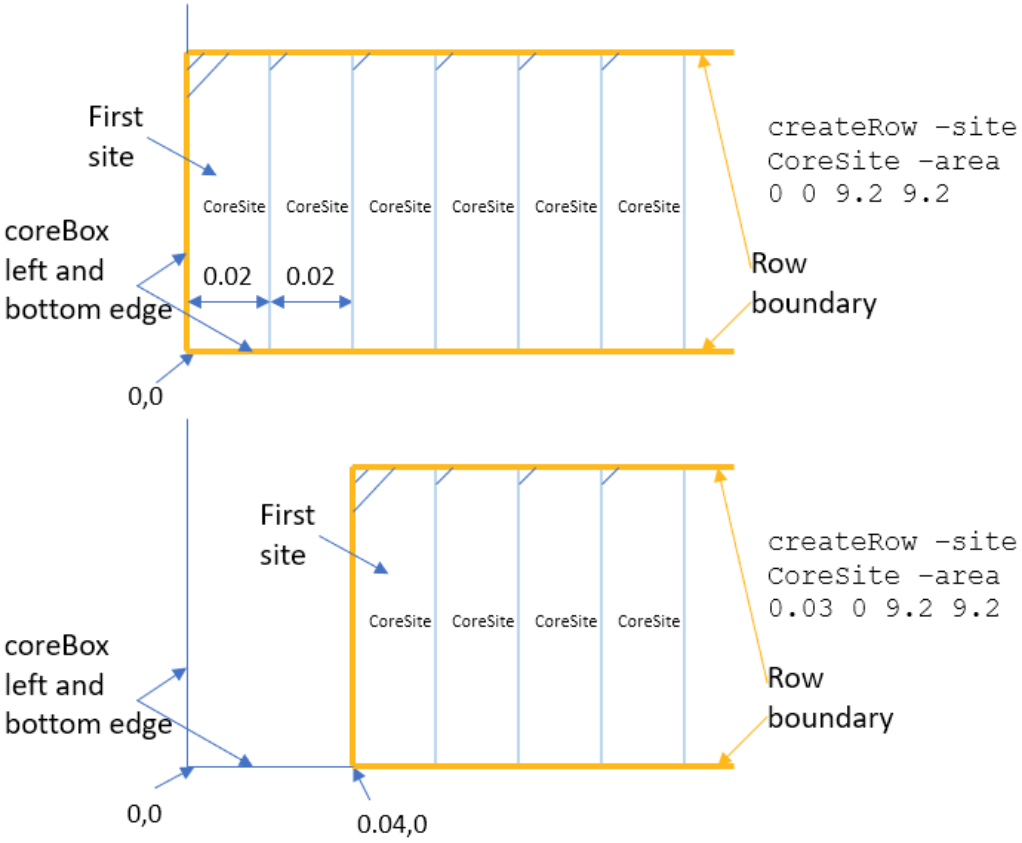
Such situations may cause issues at later stages of the implementation flow. For example, suppose a buffer is added by Innovus to make an ECO change. The inserted buffer will snap to the placement grid. In such a case, if you prompt Innovus to insert filler cells automatically to fill up the empty gaps in the design, Innovus will fail to fill up all the empty gaps because the gap between the new buffer and filler cells and other preexisting cells are not integral multiples of the site width.



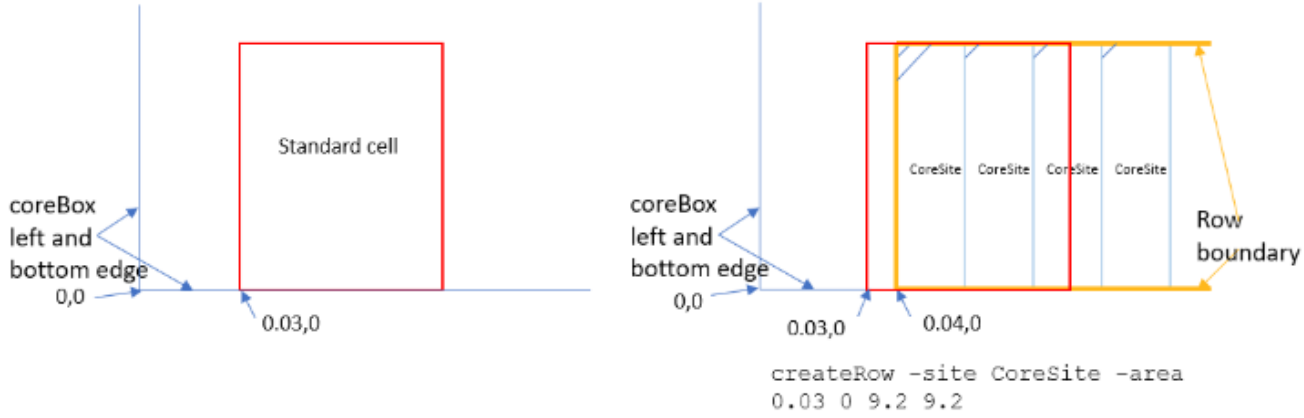
To understand how the problem can be resolved, you must understand the spatial relationship between standard rows and the Core Box. The following diagrams explain how the left edge of the rows are determined by the tool. The `-area` option of the `createRow` command is used to specify the `x` and `y` coordinates of the lower-left and upper-right corners of the area to be filled with rows.

Assume that the `x` and `y` coordinates of the lower-left corner of the `coreBox` (and `ioBox` object) is `0,0` and the site width is `0.02`. Now:

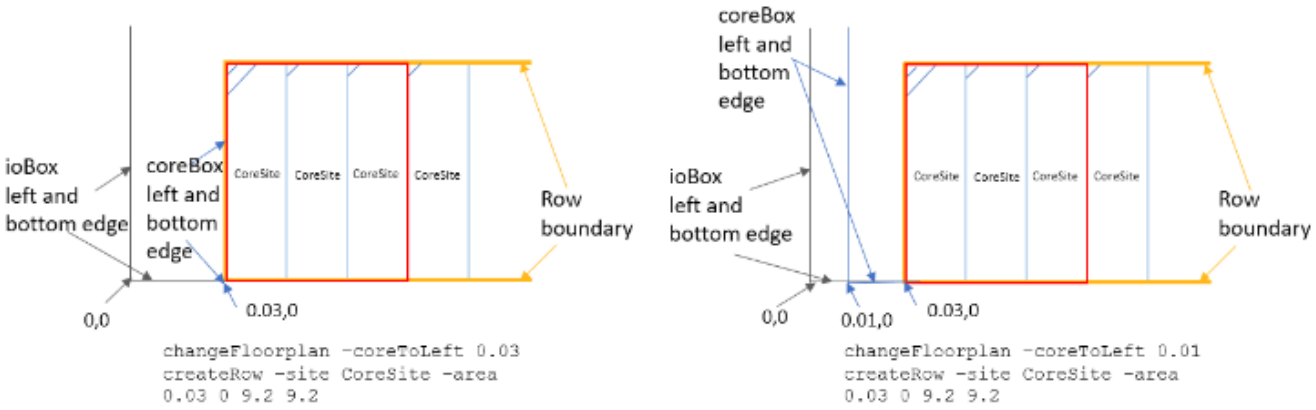
- If the `x` coordinate of the lower-left corner of the `-area` option is specified as `0`, the first site created will start its lower-left corner at `0,0`.
- If the lower-left corner of the `-area` option is specified as `0.03`, the first site created will have its lower-left corner located at `0.04,0`. Basically, the tool requires that the space between the Core Box and the left edge of the first site must be an integral multiple of the site width.



Thus, a standard cell placed with its lower-left corner at the point `0.03,0` will be off the placement grid.



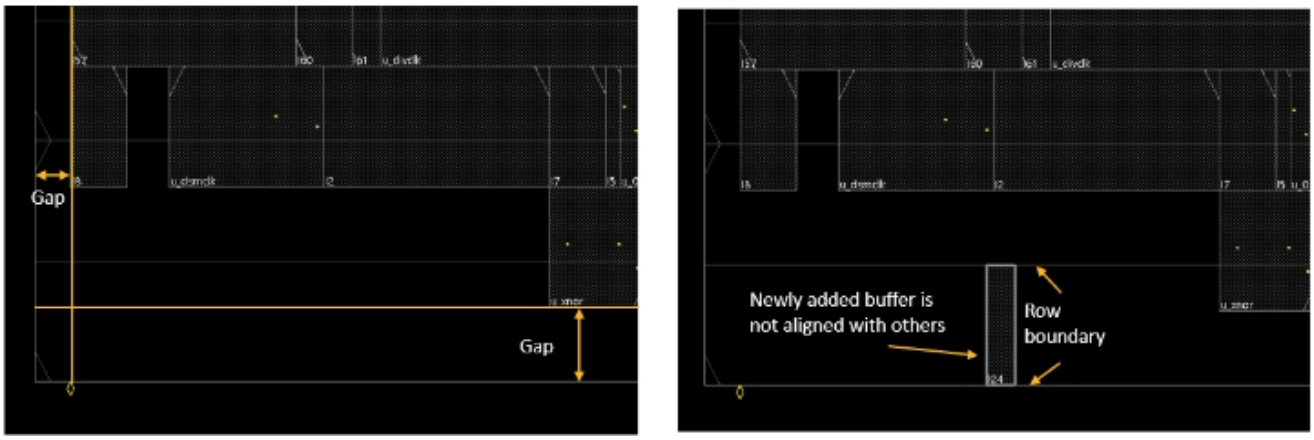
To fix the situation, one solution is to adjust the `coreBox` object using the `changeFloorplan` command. The diagrams below show that an adjustment of either `0.01` or `0.03` to move the left boundary edge of the `coreBox` will resolve the off-grid issue.



You may have to specify the `-noSnapToGrid` option with `changeFloorplan` so that the tool will honor the actual value specified in the `-coreToLeft` option.

17.3.2 Gaps Between the Left and Bottom Edges of Standard Cells and the Design Boundary

In the following case, gaps are present between the left and bottom edges of the standard cells and the design boundary. A new instance inserted by Innovus will not be aligned with other cells in the same row because Innovus will snap the new instance to the row.

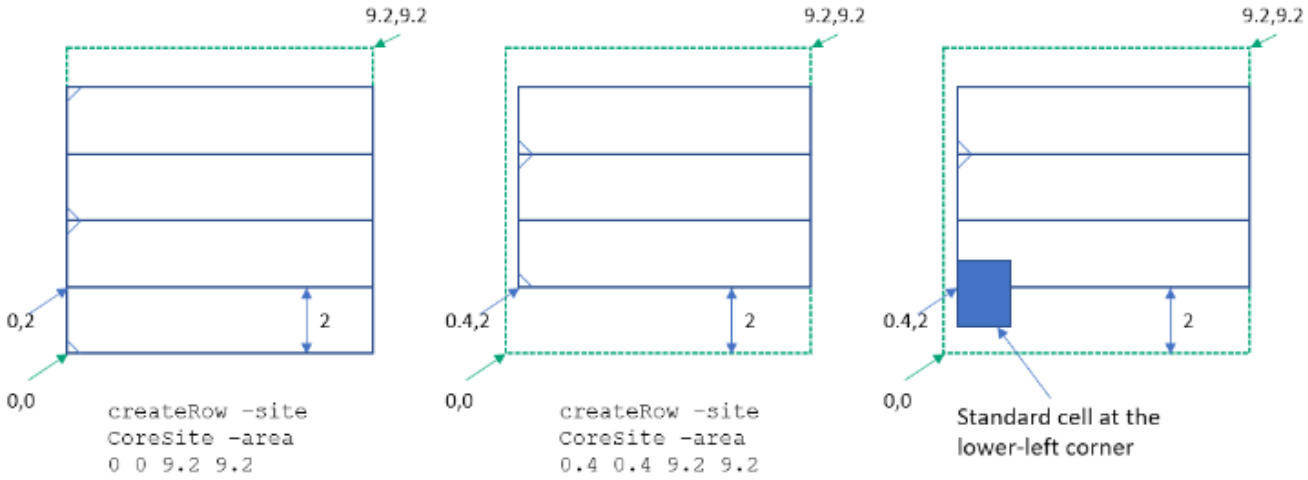


The following diagram explains how the bottom edge of the first row is determined by the `createRow` command. Let us assume the `x` and `y` coordinates of the lower-left corner of `coreBox` is `0,0`, the row height is `2`, and the site width is `0.02`:

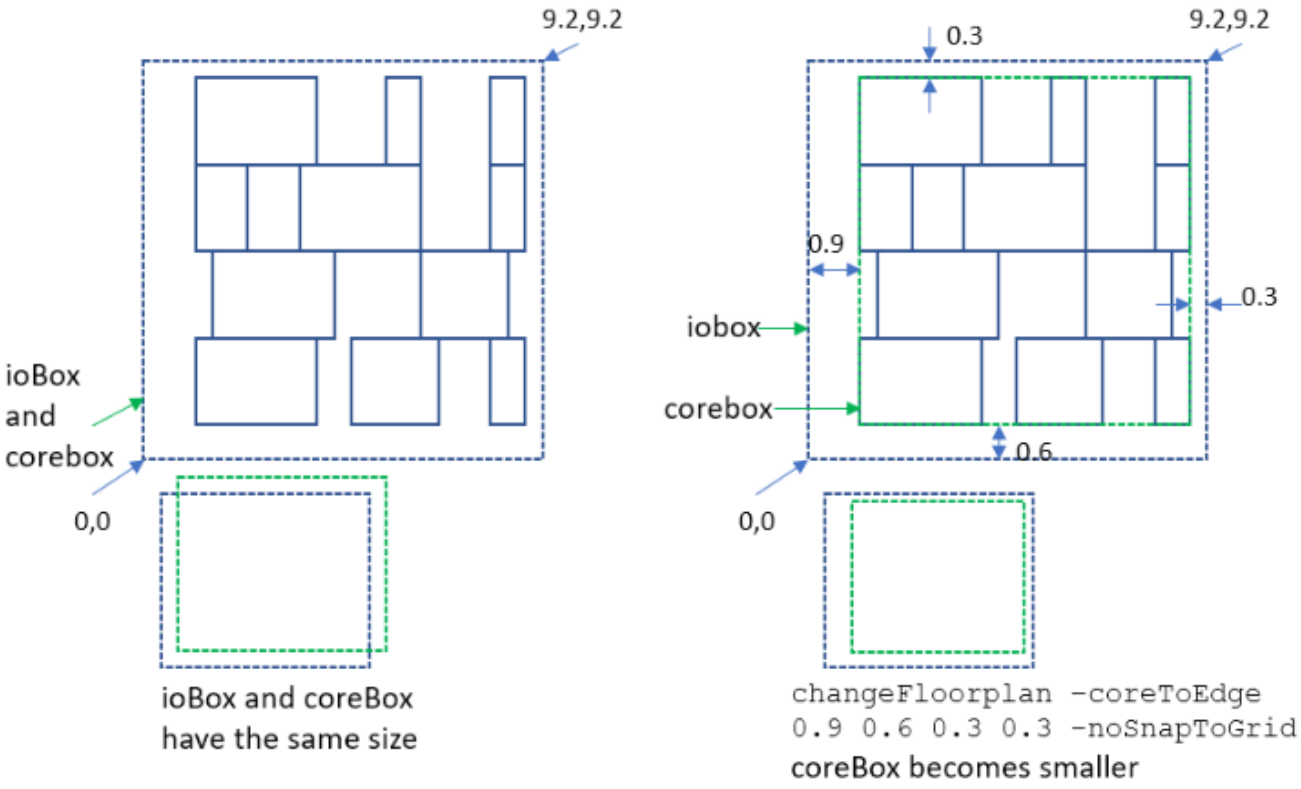
- If the `y` coordinate of the lower-left corner of the `-area` option is specified as `0`, the bottom edge of the first row will be located at 0.
- If the `y` coordinate of the lower-left corner of the `-area` option is specified as `0.4`, the bottom

edge of the first row will be located at `2`. The bottom edge of the first row will always snap to the location such that the distance between it and the bottom edge of the core box is an integral multiple of the row height.

Thus, a standard cell placed with its lower-left corner at the point `0.03,0` will be off the placement grid.



If the vertical gap between the bottom edge of the standard cells and the bottom edge of `coreBox` is not an integral multiple of the row height, the rows created will not fit with all the standard cells, irrespective of the value specified for the `-area` option. To make the rows fit, use the `changeFloorplan` command to readjust the `coreBox` before recreating the rows.



In the above case, a cell view with a PRBoundary object of `9.2x9.2` is loaded into Innovus.

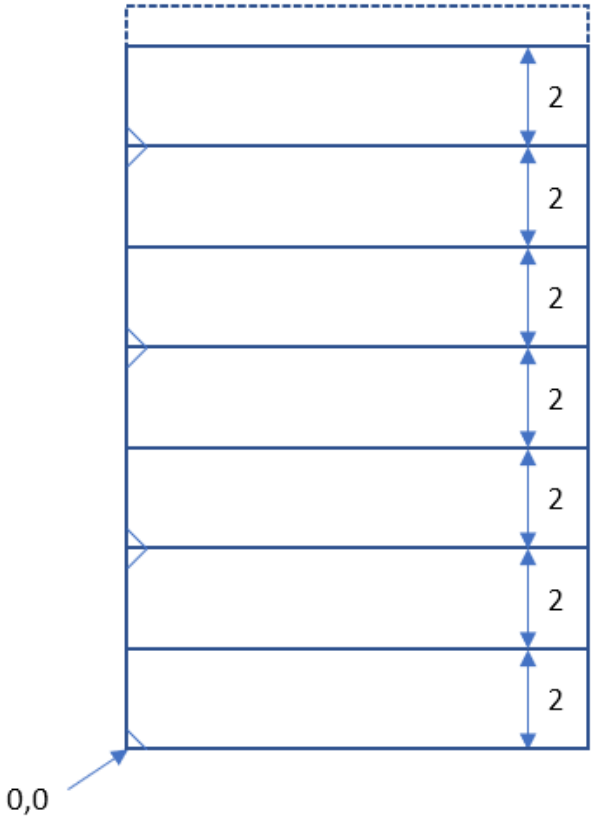
By running the `changeFloorplan` command with the `-coreToEdge` option in Innovus, the size of the Core Box is adjusted such that the boundary of the Core Box aligns with the boundary of the box that virtually bounds all the standard cells.

The `-coreToEdge` option lets you specify the adjustment for the distances between the core box and io box on each side – left, bottom, right, and top.

The `-noSnapToGrid` option for `changeFloorplan` is needed so that the tool will honor the actual value specified in the `-coreToEdge` option. Without `-noSnapToGrid`, the tool may make slight adjustments to the specified values so that the Core Box is aligned with the existing tracks. However, as the existing tracks are not likely to be valid and need to be recreated in the next step, the alignment to tracks is not needed.

17.4 Creating Double-Height Rows

Suppose your design has double-height cells. Once the single-height rows created in Innovus are aligned with the cells, creation of double-height rows will be straightforward. The following diagram shows single-height rows with CoreSite as the site name and a site height of `2` populated in the design.

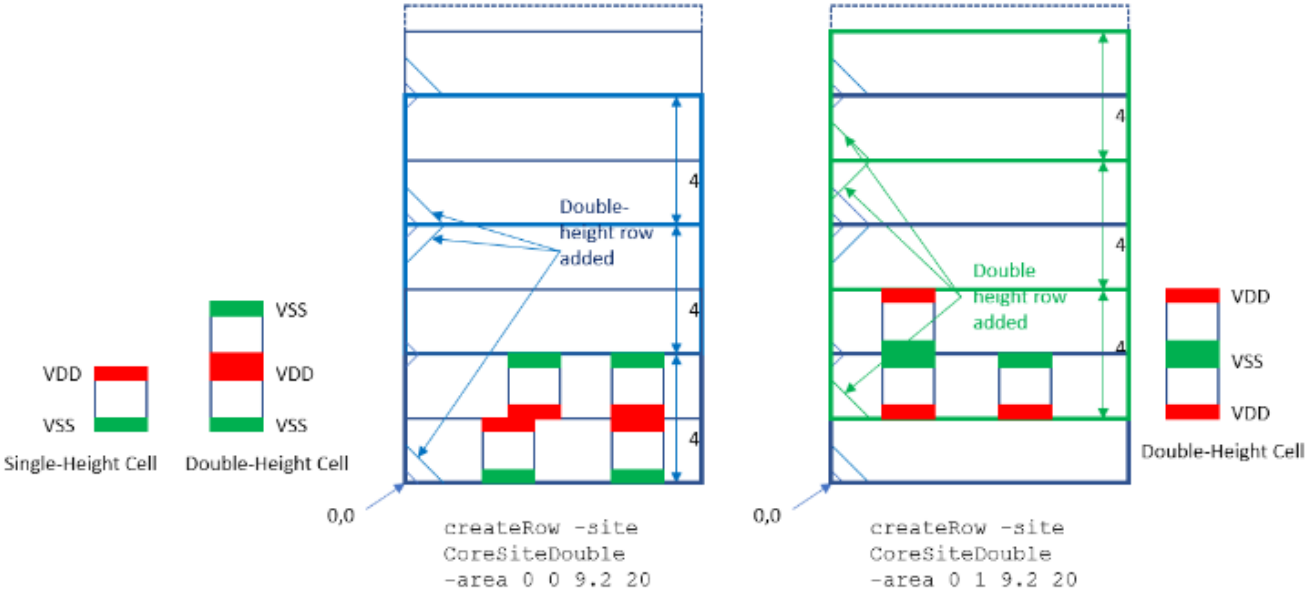



```
createRow -site CoreSite -area 0 0 9.2 20
```

Assume these single height rows are for cells with a ground pin (e.g. VSS) at the bottom and a power pin (e.g. VDD) at the top. If double-height cells with *VSS-VDD-VSS* combination of P/G pins are used in the design, run the following command to create the corresponding standard cell rows.

```
createRow -site CoreSiteDouble -area 0 0 9.2 20
```

The first `y` coordinate for the `-area` option will be `0` so that the first double-height row starts from the origin.



If double-height cells with *VDD-VSS-VDD* combination of P/G pins are used in the design, run the following command to create the corresponding standard cell rows.

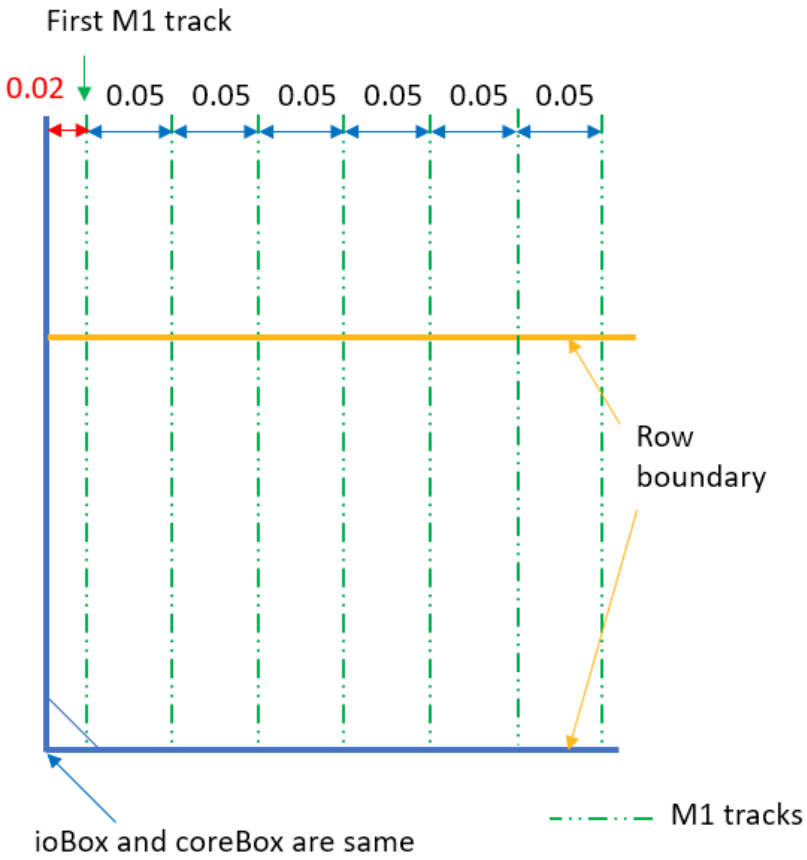
```
createRow -site CoreSiteDouble -area 0 1 9.2 20
```

The first `y` coordinate for the `-area` option can be any value greater than `0` and less or equal to `2` so that the first double-height row starts from the bottom edge of the second single-height row.

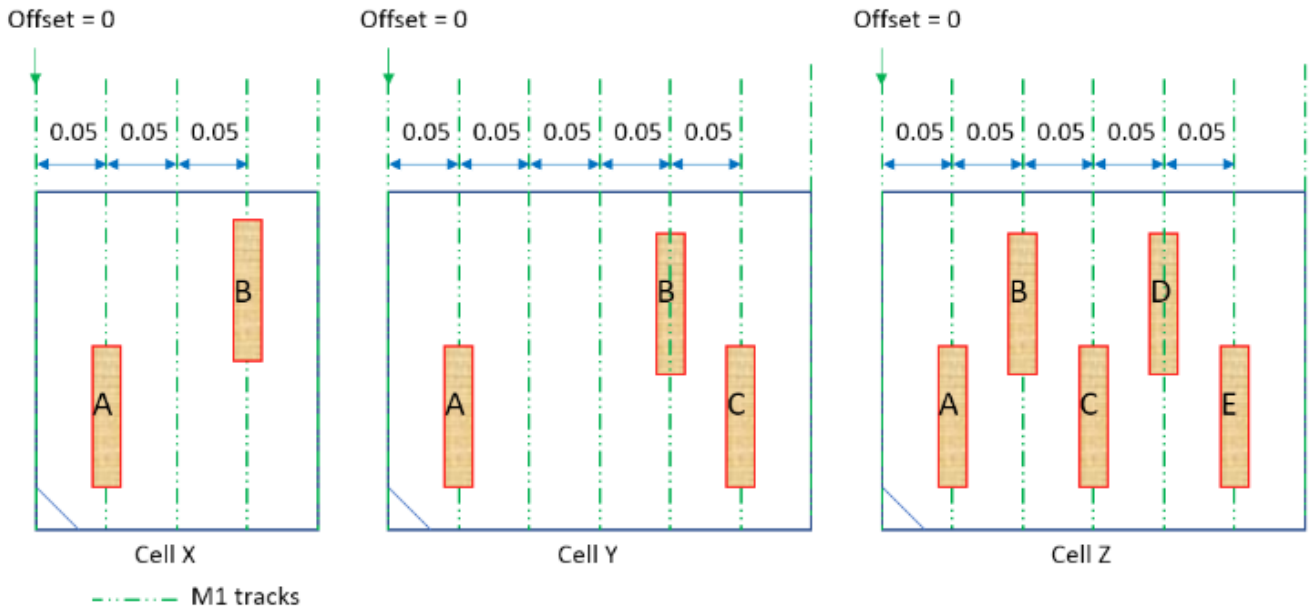
17.5 Creating Routing Tracks with Respect to coreBox

The location of the lower-left corner of the Core Box affects not only row creation but also track creation.

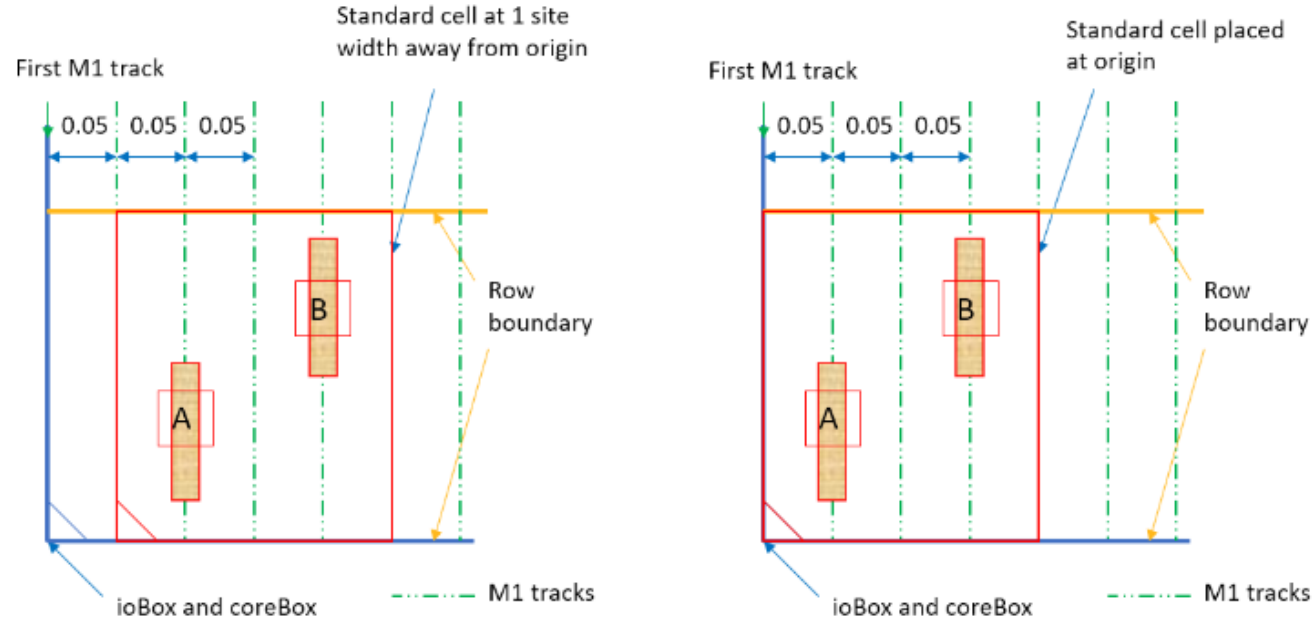
In the following diagram, the routing tracks are created with an offset of `0.02` um and a pitch of `0.05` um. The offset is measured from the left, vertical edge of the `coreBox` object to the first track. The initial rows and routing tracks are created by Innovus when the cell view is loaded (for example, with `init_design`).



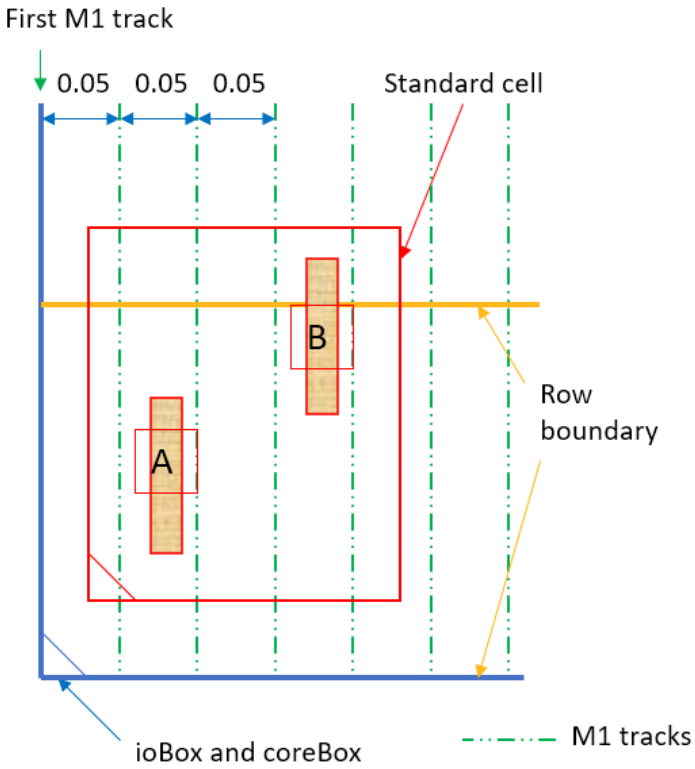
By default, the tool automatically calculates the offset and pitch values for best pin access results. It does the computation by examining the pin locations of all standard cells (abstracts) stored in the Innovus database. However, the offset value it determines is measured from the boundary of the standard cell. Note that typically, the technology TLEF or the MSOA PDK have recommended values for the offset and pitch values of the routing tracks for each metal routing layers. These offset values are also typically measured from the boundary of the standard cell.



Assume that the standard cell being placed has a site width of `0.05` and the calculated offset and pitch are `0` and `0.05` um, respectively. If the cell is placed in a way that it is aligned with the placement grid (formed by the site width), then the calculated offset and pitch value will result in the tracks being aligned with the I/O pins of the standard cell. The two examples below show standard cells being placed on the grid, resulting in their I/O pins aligning with the tracks.



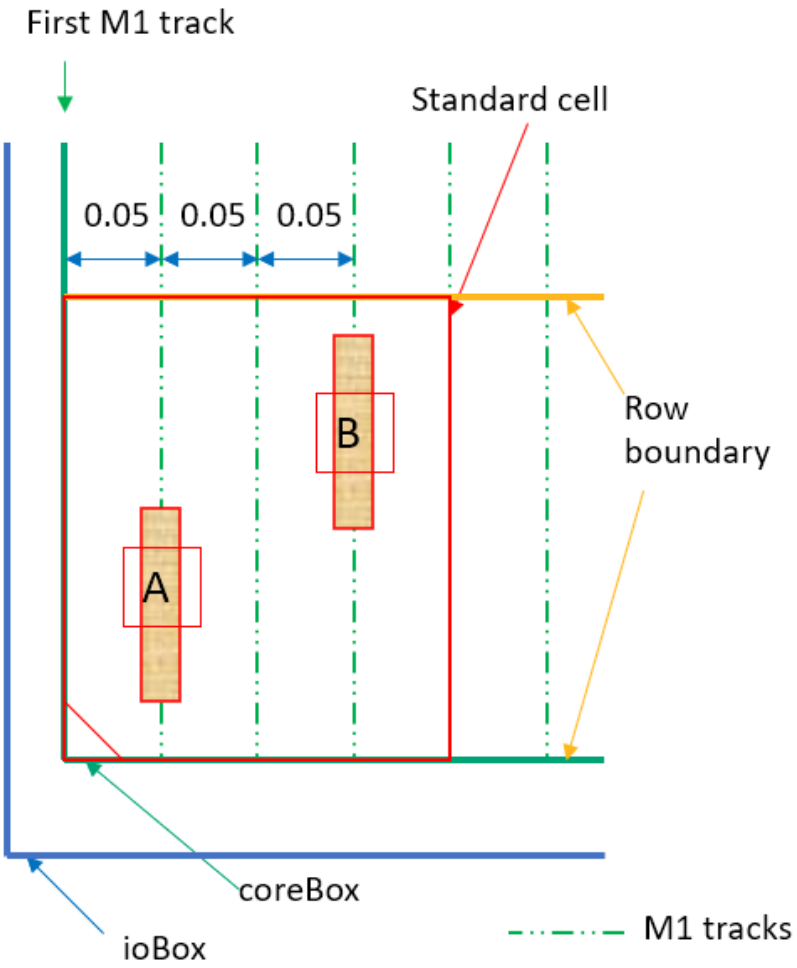
However, it is possible that the cells manually placed in Virtuoso are not aligned with the rows initially created by Innovus as shown in the beginning of this section. As a result, the tracks are not aligned with the centerline of the pins, which becomes a potential problem for routing.



To address such problems, follow the steps below to readjust the `coreBox` and recreate the tracks:

```
changeFloorplan -coreToEdge 0.03 0.06 0 0 -noSnapToGrid

add_tracks
```

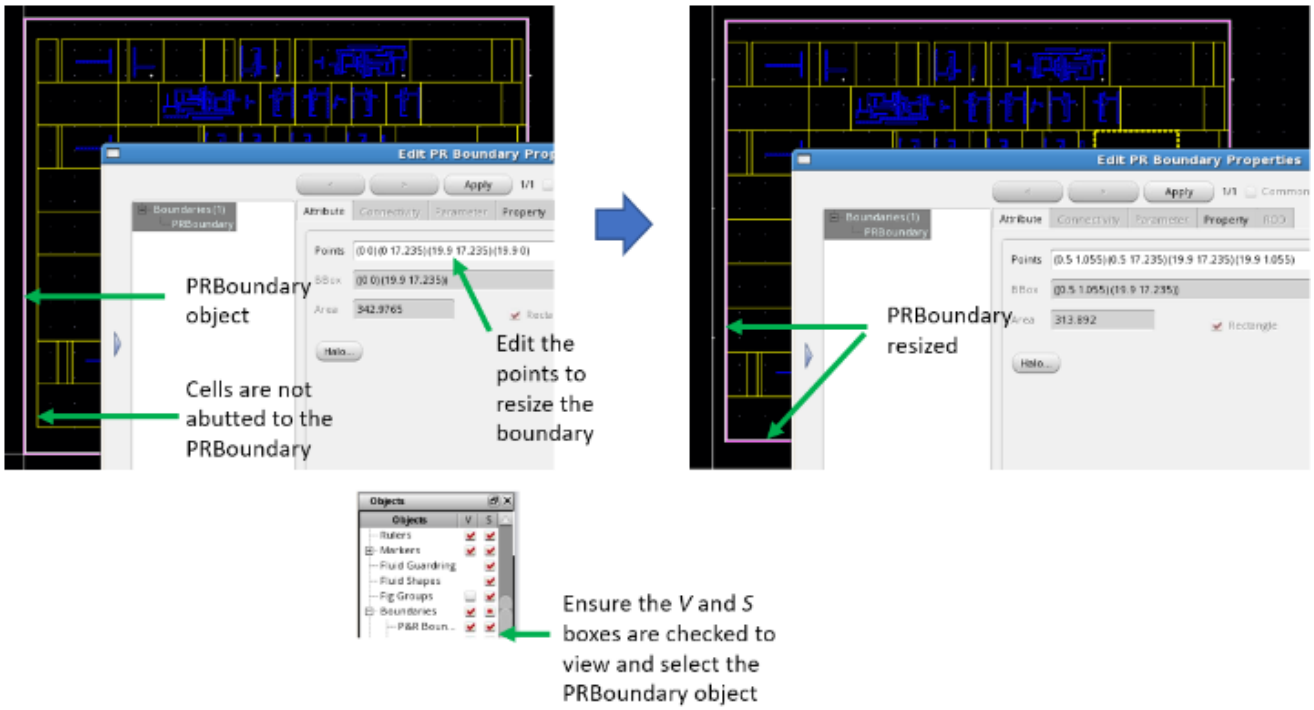


Note that in some complex cases using multiple sets of standard cell libraries, you may have to specify the offset, pitch value, and the mask pattern. For example:

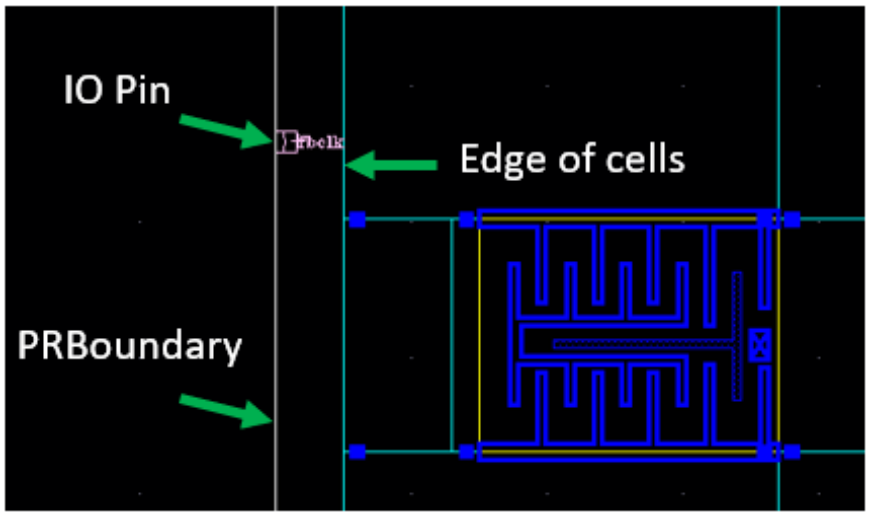
```
add_tracks -offsets {M1 vert 0 M2 horiz 0.04 M2 0 ...} -mask_pattern {M1 1 2 M2 1 2
...}
```

17.6 Resizing the PRBoundary Object in Virtuoso

The problem mentioned in the previous section may also be resolved by opening the cell view in Virtuoso and resizing the PRBoundary object so that the left and bottom edges of the PRboundary object are aligned with the left and bottom edges of the bounding box for all the standard cells.



However, note that the above correction is not allowed in cases where gaps are preferred to enable the I/O pins and the standard cell edges to be spaced apart.



18

Static Timing Analysis for Mixed Signal Designs

- Overview
- OpenAccess Compatibility for Mixed-Signal STA Flow
  - Basic Design Requirements
  - Requirements for Correct Connectivity Propagation
  - Requirements for OpenAccess Compatibility
  - Useful Utilities and Information
- Handling Schematic-Driven Designs Implemented Using a Non-Interoperable PDK
  - Checking the Technology Database Graph of a Specific Design Library
  - Opening a Design Attached to a Non-Interoperable PDK in Innovus
  - Ways To Switch a Design Library from a Non-Interoperable to an Interoperable PDK
- Running STA by Flattening the Design
  - Steps for Running STA Using the Flatten Approach
- Running STA by Using the FTM
  - Generating the FTM for Each Block
  - Running STA on Top-level Design with FTM of Blocks
- Difference between Flat and FTM Approaches
- Guidelines for Running STA Flow on Mixed-Signal Design
- Creating a Quick Timing Model for Analog IPs in Innovus
- Different Ways of Running assembleDesign
  - Running assembleDesign in the Batch Mode
  - Running assembleDesign in the Incremental Mode

    - Differences between the Batch and Incremental Modes of assembleDesign
    - Running assembleDesign with the -allTimingBlocks Option
    - Tips on Running assembleDesign -allTimingBlocks
    - Caveats for and Limitations of assembleDesign -allTimingBlocks
    - Running assembleDesign in the Batch Mode and Subsequently in the Incremental Mode
    - Limitations of assembleDesign
- Parasitic RC Extraction for Running MS-STA
    - Running Quantus Extraction with postRoute Engine for a Routed Design
    - Running Quantus Extraction with Signoff Effort Level
    - Using the Quantus Layer Map File
    - Auto-creation of the Quantus Layer Map File from Innovus
    - Running Signoff Quantus Extraction in the OpenAccess Mode
    - Auto-creation of Input Files from Innovus To Run Standalone Quantus QRC
    - Running Standalone Quantus QRC and Loading the Resulting SPEF Files Back to Innovus
- Tips for Debugging the No Constrained Timing Path Issue Generated by report_timing
    - Case Studies
    - Basic Conditions for Reporting the Timing on a Path
    - Common No Constraint Situations
    - Checking the Validity of a Timing Constraint
    - Checking the Existence of a Design Object in the Layout
    - Common Invalid Timing Path Situations

18.1 Overview

In analog mixed-signal designs, the digital logic is placed inside the analog mixed signal hierarchy and there is a need to accurately analyze the entire digital logic path during full-chip Static Timing Analysis (STA).

Generating dotlib Liberty models of the complete analog mixed-signal (AMS) block is a challenge because the dotlib information needs to be manually generated from extraction and Spice simulation results.

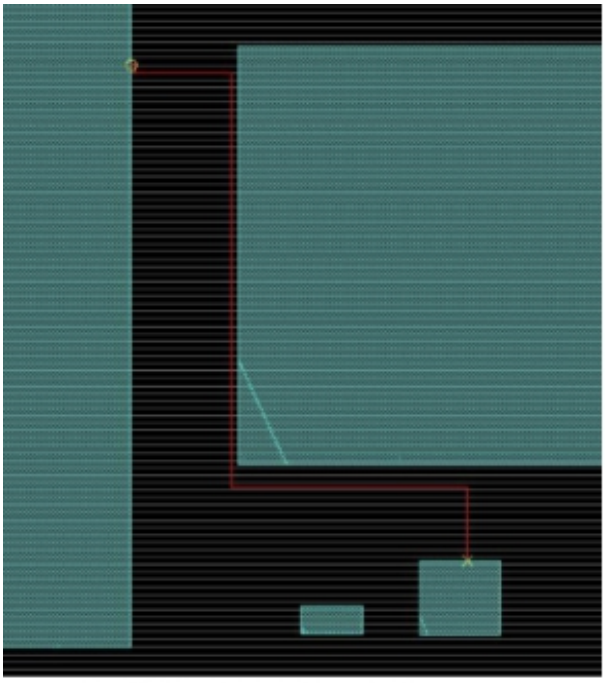
Ignoring the digital paths or stopping the timing analysis at the boundary of an AMS block and leaving large design margins has a direct impact on design performance. A solution is required to make it easy for mixed-signal design teams to execute comprehensive top-level STA that includes the entire digital logic in the chip.

Another challenge is that design methodologies of AMS blocks follow full-custom design flows using the Virtuoso platform, which poses a challenge in extracting a hierarchical netlist as well as parasitic information of all the nets. Therefore, it becomes difficult to analyze the embedded digital logic within the analog design hierarchy.

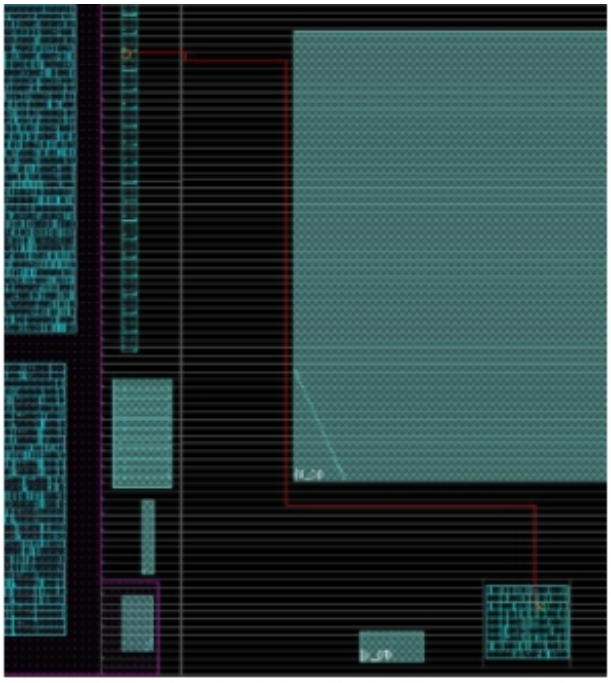
To address the above challenges, Cadence offers two ways of running STA on Innovus, through the interoperable Open Access database. The first way is to flatten the design to a certain physical level so that the timing paths to be analyzed are visible and extractable by Innovus. Another way is to generate full timing models (FTM) for mixed-signal blocks that contain the timing paths to be analyzed. The FTM contains full logical netlist information and RC parasitic information of the blocks that enables STA to be done in Innovus without providing the physical layout of the blocks. Both ways can be used either in a schematic-driven mixed signal flow or netlist-driven mixed signal flow. There is no need to generate dotlib Liberty models for the entire AMS block.

18.2 OpenAccess Compatibility for Mixed-Signal STA Flow

In many mixed-signal designs, the digital logic exists at various levels of the physical hierarchy. To perform an accurate analysis of such timing paths, the timer requires the digital logic in the lower levels of the hierarchy to be exposed for creation of the complete timing path. The digital tool should be able to trace the logical and the physical connectivity for the entire timing path.



Complete path not visible after initial design load



Complete path visible after assembleDesign

To meet this requirement, mixed-signal designs must be created using OpenAccess, which is the underlying database for both Virtuoso and Innovus.

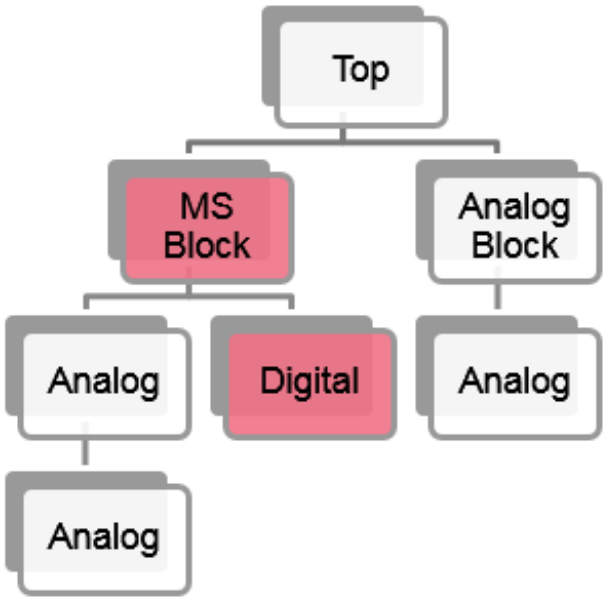
This section focuses on the requirements for interoperability of OpenAccess design objects between Virtuoso and Innovus for mixed-signal Static Timing Analysis (STA).

18.2.1 Basic Design Requirements

The basic guidelines for implementing mixed-signal blocks that will be involved in the STA flow are:

18.2.1.1 Design should be Virtuoso-XL compliant

The mixed-signal design should be Virtuoso-XL (VXL) compliant to the level in the physical hierarchy that fully contains the timing path in the design. For example, if there are eight levels of physical hierarchy in the design, but the timing path is from a register at the top level of the design to another register located at the third level of physical hierarchy, VXL compatibility is required only to the third level of the physical hierarchy.



Note: The focus here is only on the hierarchy from the top to the digital block. The hierarchy within the Analog Blocks is not important from the mixed-signal STA flow perspective.

The recommended method would be to create these mixed-signal blocks using Virtuoso-XL.

Use *Connectivity -> Check -> XL Compliance* in VXL to check for mismatches between instances, terminals, and nets. This check reports the following:

- Bound, unbound and un-generated instances
- Connectivity differences in the schematic and layout
- Parameter differences for instances
- Any unbound schematic terminal name which matches a label/text display in the layout to indicate missing layout terminals
- Mosaics in the layout
- Complex bindings for instances

The results for a VXL-compliant design should have everything bound properly:

```
INFO (LX-1501): Check finished:
                        bound
  terminals             14
  nets                  66
  instances             46
```

18.2.1.2 Design should have logical and physical connectivity

As Innovus is a connectivity-driven tool, it is essential to have logical and physical connectivity in the Virtuoso layout design that is being loaded in Innovus for analysis. Innovus cannot load the designs that are placed and routed by hand in Virtuoso and have no logical or physical connectivity. In addition, `assembleDesign`, which is required in mixed-signal STA flow, generates the following error:

```
% assembleDesign -block {designLib manualCell layout}

Reading EMH from OA ...

**ERROR: (IMPOAX-170): Top Module not found for specified design in OA. EMH data can
not be read from the OA design. Rerun command after correcting the OA database.

**ERROR: (IMPOAX-1149): Error in reading netlist from OA block Lib: designLib, Cell:
manualCell, View: layout. Refer to the errors above.

**ERROR: (IMPSYT-35001): Error in importing block netlist
```

For the mixed-signal STA flow, we therefore recommend that you create the mixed-signal blocks/IPs using VXL so that they are correct by construction.

If a design has not been implemented in VXL but is LVS clean, you can perform the following for restoring connectivity:

- Use *Connectivity -> Update -> Binding* in VXL to create the binding between the layout and the schematic instances. It is good to keep the layout and schematic hierarchy in sync.
- Use *Connectivity -> Define Device Correspondence* to create the mapping manually between the schematic and the layout instances.
- Use the *Connectivity -> Update -> Extract Layout* option to create the list of nets from scratch. With this option, all islands are extracted in complete mode and opens are computed for all nets. Nets are created on the top-level instance terminals based on the schematic connections or actual physical shapes overlap in the layout. It also creates nets on all physically connected top-level routing shapes, vias, and pin figures.

For the mixed-signal STA flow, VXL should not report any opens or shorts for top-level nets when the extraction level is set to 0. If there are opens, the correct connectivity propagation and RC

extraction will not happen in Innovus. The only exception is for the non-pin shapes on interconnect layers in the layout view that are a part of the pin shapes in the abstract view for the cell. In this case, even though the VXL extractor shows opens, but since in Innovus, the cell will be bound to the abstract, these will not be actual opens. For this case, set the extraction level to 1 and turn on the option to allow extraction to unassigned shapes in the Extract Layout options.

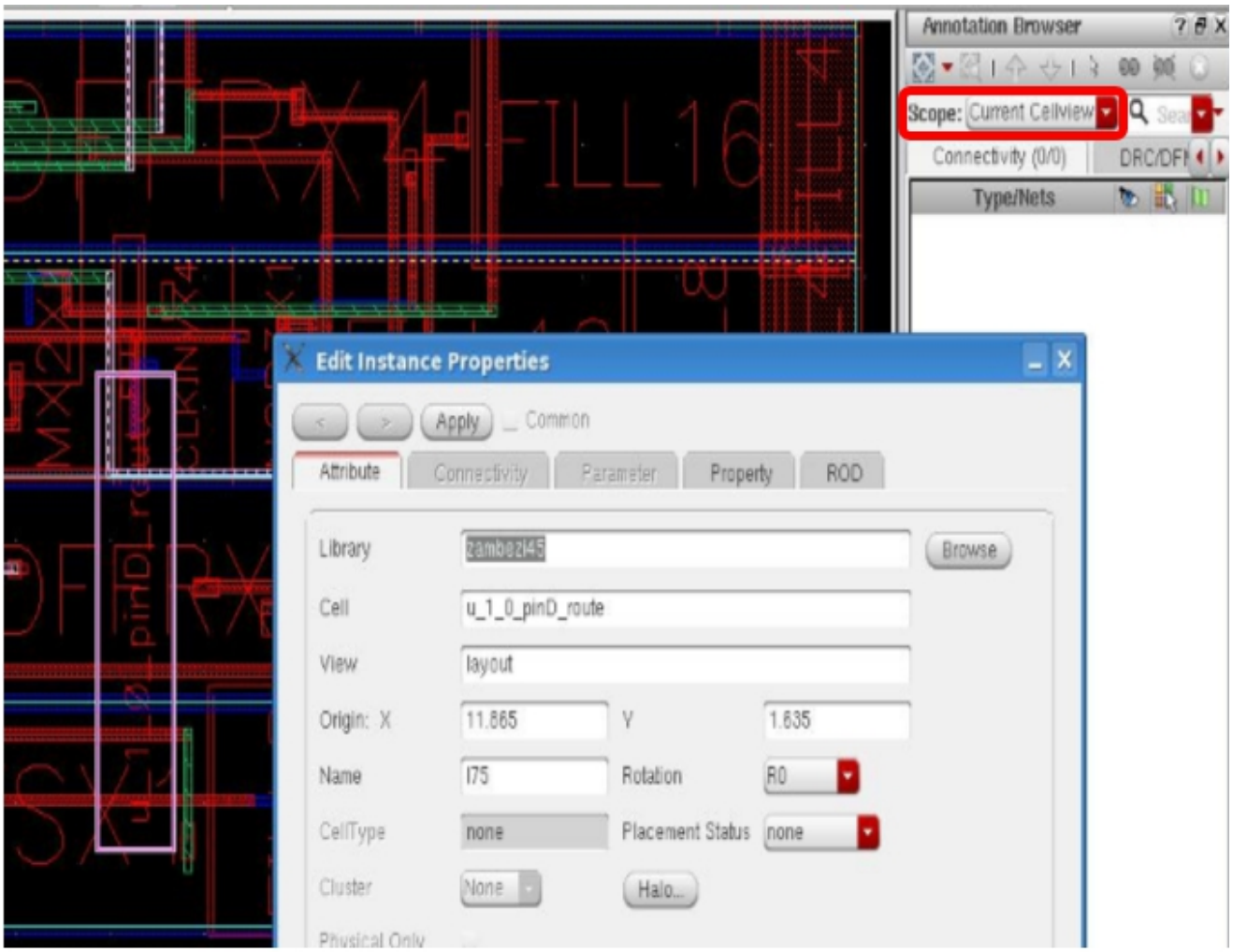
Window -> Assistants -> Annotation Browser should not show any Opens/Shorts:

```
INFO (LCE-1009): The cell verifier found no violations
```

In addition, the routing shapes for top-level nets should terminate on top of an instance pin figure for the entire net to be extracted properly.

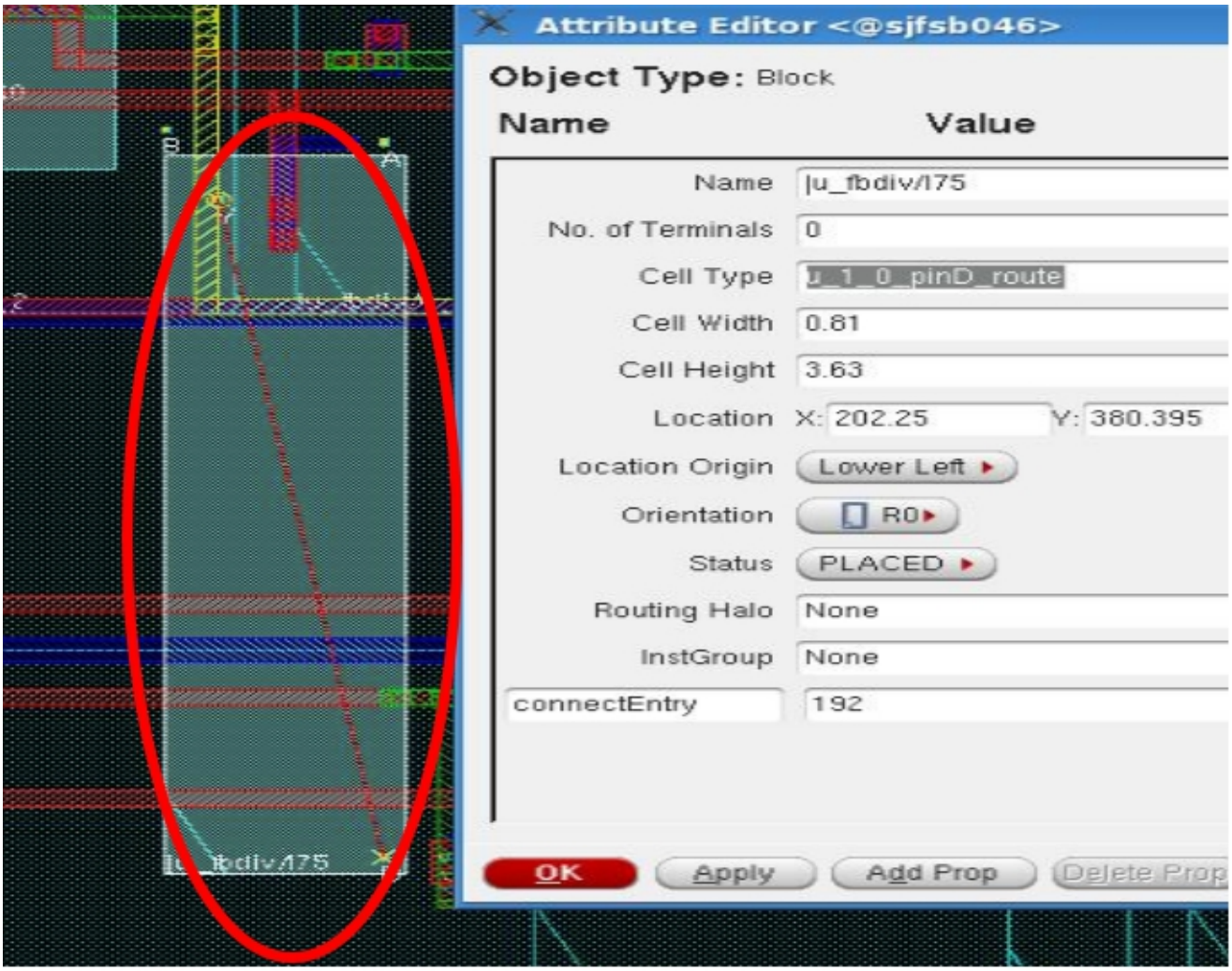
18.2.1.3 Routing shapes for a net should be at the same level

The routing shapes for a net should not be spread across the design hierarchy. If a net is routed in such a manner that the shapes/vias are at different levels, it will be difficult to figure out the levels to assemble in Innovus to expose the entire path at the top. In addition, it might not be possible to create the correct connectivity.



No opens with VXL Extract till level 1

In this design, the routing shapes at level 1 in Virtuoso-XL are fine and no opens are detected when extraction is run for level 1 in Virtuoso. However, Innovus cannot see the lower level shapes, as shown below, and reports opens.



Routing shapes missing in Innovus

18.2.1.4 Object names should match names in sdc file

The names of Layout objects should match with the names in the sdc file. If the sdc file is generated using the schematic as a reference, make sure that the layout and the schematic names also match. The layout should be created using the Generate From Source (GFS), Configure Physical Hierarchy (CPH) and the Generate Physical Hierarchy (GPH) commands. Keep the following points in mind to avoid naming mismatches:

- Avoid using the create instance and copy commands for adding instances because these are not schematic connectivity aware.
- Avoid Many-to-Many instance mappings or manually created instance name mappings because these can cause a mismatch in the names.

- The leading pipe (|) in instance names generated by Virtuoso-XL can cause issues of name mismatch when reading the sdc files. Specify the following variable in the Virtuoso-XL setup to create instances without the leading pipe (|) character:

```
envSetVal("layoutXL" "prefixLayoutInstNamesWithPipe" 'boolean nil)
```

18.2.2 Requirements for Correct Connectivity Propagation

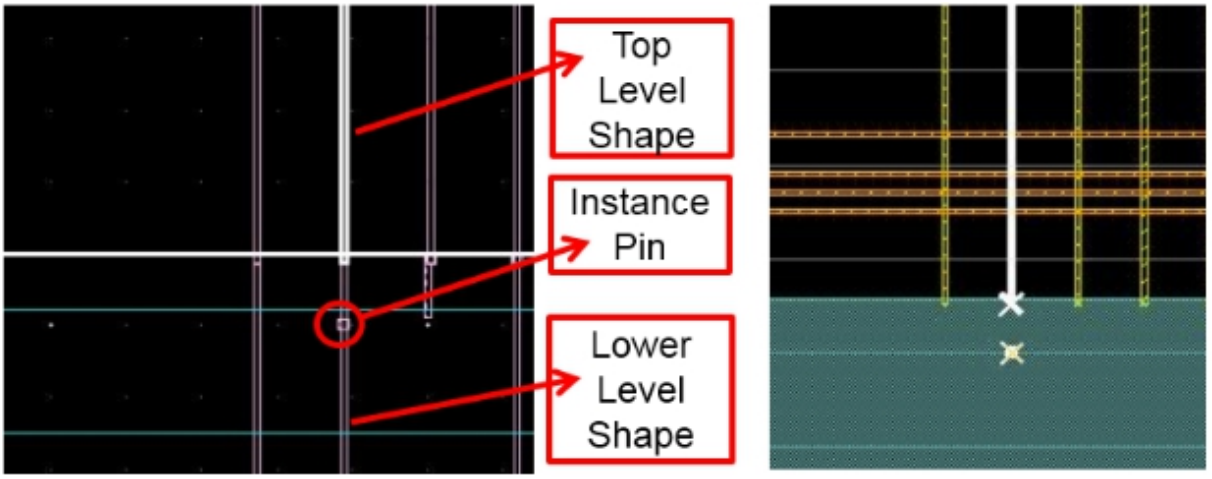
For correct connectivity propagation in mixed-signal designs, ensure the following:

18.2.2.1 Logical connectivity

The nets on the instance terminals should be properly connected for logical connectivity to be propagated. Mosaics should be avoided in the design because although Innovus creates the instance from the mosaic, `assembleDesign` does not map the top-level nets to the lower-level mosaic instances.

18.2.2.2 Pin shape creation

All cells should have shape pins on interconnect layers placed on the cell boundary.



The dot pins used in the design are not understood by Innovus and `verifyConnectivity` reports opens:

```
**ERROR: (IMPOAX-641): Found pin shape of type 'Dot' for terminal 'clkin'. Only shapes of type RECT or POLYGON or LINE and vias are supported as pin shapes. This shape will be ignored.
```

18.2.2.3 Via object creation

OpenAccess via objects should be used and not instances of cells from the library. The create via command should be used to create an OpenAccess via object with a valid viaDef from the technology. Do not use the create instance command or any cell from the library that acts as a via. If you do so, VXL and Innovus would not be able to trace through these instances.



Do not use instances instead of via objects

18.2.2.4 Symbolic type routing

Routing should preferably be of type symbolic. If all the routing is of type geometric, `extractRC` fails and the following error is reported:

```
% extractRC: Design must be routed before running extractRC
```

To prevent such errors, avoid creating paths/rectangles manually. Instead, use the *Create -> Wiring -> Wire/Bus* or the Auto Routing and point-to-point routing capabilities in Virtuoso to create paths.



Note that if you choose *truncate* or *extend* as begin and end styles, the nets will be read in as regular nets in Innovus. If you choose *variable* or *custom* as begin and end styles, the nets will be read in as special nets in Innovus.

18.2.2.5 No MultiPart Paths for bus routing and shielding nets

Avoid the use of MultiPart Paths (MPPs) for bus routing and shielding nets. Use the *Create -> Wiring -> Bu*s feature in Virtuoso for bus routing. Use the Auto Router for shielding the nets. If MPPs are used as buses or to shield nets, make sure that the correct net is set on each of the subpart. Note that MPPs can be used as guard rings because guard rings are not used for timing analysis.

18.2.3 Requirements for OpenAccess Compatibility

For OpenAccess compatibility, the following requirements must be met.

18.2.3.1 Interoperable PDK

A basic requirement for OpenAccess compatibility is an interoperable PDK created by the PDK factory team. To set up an interoperable PDK:

1. Import LEF rules into OA database

2. Review/compare LEF and OA technology files
3. Merge LEF rules with Virtuoso technology data. The additional layer definitions come from LEF import, whereas Virtuoso technology has super set rules.
4. Create incremental technology database to include additional data from LEF

For more information on setting up an interoperable PDK, refer to the Technology Data Preparation chapter.

18.2.3.2 OpenAccess Reference Libraries and Cell Setup

For the implementation of a top-level design, OpenAccess libraries must be available for all the standard cells and macros with an abstract view must be available for each cell. If the abstract is not available while reading an OpenAccess design, an abstract is created on the fly from the layout (provided it has the prBoundary object and the pins). The abstract created on the fly contains bare minimum data and allows the flow to continue. However, as such an abstract is not production quality, it should not to be used for production flow.

For the `assembleDesign` command in Innovus for STA, make sure that the pin count in the layout matches the pin count in the abstracts. In addition, make sure that the pin models are set up properly for the Macros:

- Strongly connected shapes (multiple shapes of one PORT in LEF) indicate that a signal router is allowed to connect to one shape of the PORT, and continue routing from another shape of the same PORT.
- Weakly connected shapes (separate PORTs of the same PIN in LEF) are assumed to be connected through resistive paths inside the MACRO that should not be used by routers. The signal router should connect to one or the other PORT, but not both.
- Mustjoin pins indicate that these are to be connected together by the router outside of the cellview

18.2.3.3 OpenAccess Objects instead of LPPs

OpenAccess interoperability works on database objects and not layer-purpose pairs (LPPs).

- Blockages should be created as an OpenAccess blockage object and not a shape on a layer/purpose pair:
    - Use *Create -> P&R Objects -> Blockage* in Virtuoso to create an OpenAccess blockage object.
- The cell Boundary should be a prBoundary type object and is needed by Innovus for all cells:

    - Use *Create -> P&R Objects -> P&R Boundary* in Virtuoso to create a prBoundary type object. Otherwise, you will get the following error:

      ```
      **ERROR: (IMPOAX-815): PRBoundary is not present in OA design. Cannot restore the design from given OA database. Correct the data and retry.
      ```

- Pins must be created as Pin Objects with shapes on interconnect layers and not as shapes with text on top:
    - Use *Create -> Pin* in Virtuoso and select metal layer as per LDRS to create Pin Objects.
- All metal shapes on any purpose are treated as routing shapes and shorted.

18.2.3.4 Correct sigType for Pins and Nets

Set the correct sigType for the Power and Ground pins and nets. If not, the following warning is generated:

```
**WARN: (IMPDB-1256):   Power pin VDD of instance DFF7 is connected to non-p/g net vdd_1p2v.  Mark the net as power net and create associated snet
```

The top-level power/ground net should be marked global in Virtuoso in case the assembled block has physical-only power/ground pins. Otherwise, the following error is generated:

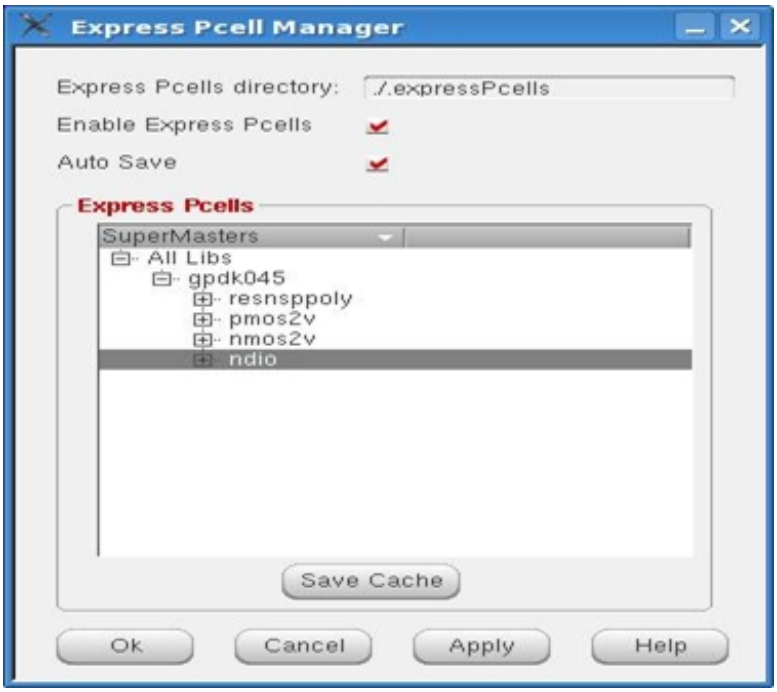
```
**ERROR: (IMPOAX-1087): Cannot create instance terminal 'VDD' of instance ‘DFF7' connecting to net ‘vdd_1P2' : A physical-only instTerm cannot connect to a net located in a different occurrence
```

To set the sigType, select the pins in Virtuoso, and then use the Navigator and the property editor widgets. The proper sigType is needed for marking nets as Analog as well. Nets marked as Analog are ignored by NanoRoute in Innovus.

18.2.3.5 PCell Cache Generation

Generate the parameterized cell (PCell) cache from Virtuoso before loading the OpenAccess DB into Innovus:

1. From the Layout in Virtuoso Window, invoke *Tools -> Express Pcell Manager*.
2. Fill in the directory name to save the PCell cache.
3. Select the *Enable Express Pcells* check box.
4. Select the *Auto Save* check box.
5. Click *Ok*.



This is equivalent to:

```
setenv CDS_ENABLE_EXP_PCELL true

setenv CDS_EXP_PCELL_DIR ./.expressPcells
```

18.2.3.6 Bus Ordering and Continuity

Run Verilog Annotate on the cells to set the bus bit information on the cells' pins before the data is loaded in Innovus. This is needed to set the bus terminal information and order on each cell that has bus pins so that the bus connections can be established properly by Innovus. If this not done, the following warning is generated:

```
**WARN: (IMPOAX-252): Found BusBit terminals of bus 'i_cnt[1]' of cell 'Design_Top'
without bus ordering information in OA library 'designLib'. This may lead to problems
during saveDesign. It is recommended to run verilogAnnotate on the library for
annotating bus ordering information to such terminals.
```

You also need to make sure that the bus indices are continuous. Otherwise, the following warning is generated:

```
**WARN: (IMPDB-2080):            Non-continuous bus index in cell TESTLV_HIGH_TOP.
Missing pin info between A[31] and A[64]

**WARN: (IMPDB-2080):            Non-continuous bus index in cell TESTLV_LOW_TOP.
Missing pin info between A[63] and A[96]
```

In addition, the OpenAccess flow at present does not support negative bus indices, such as the following, in the Verilog netlist:

```
wire [15:-16] CNTRL_O;

wire [15:-16] CNTRL_T;

wire [0:-15] CNTRL_S;
```

You should use logic synthesis to avoid generating such netlists.

18.2.3.7 Bus Annotation Set by Default

To make sure that the bus annotation is set up by default, enable the option to create implicit inherited terminals while creating the layout in VXL using Generate From Source and Generate Physical Hierarchy (preferred way to design). verilogAnnotate is not needed for blocks created with Virtuoso-XL and abstract Generator post the 615 ISR16 release because this option has been turned on by default from this release. For older cells/IPs, you can export a verilog stub from the symbol view of the cell and run the following command for setting the bus terms and bus order correctly:

```
verilogAnnotate -verilog block.v -refLibs cellsLib -libDefFile cds.lib -refViews
"layout abstract"
```

18.2.3.8 Fixed or Placed Placement Status

The placement status for instances and pins should be set as placed or fixed.

Sample skill to check/set the status:

- `% cv~> instances~> status` - Returns nil for all instances
- `% cv ~> instances~> status = "placed"` - Sets as placed for all instances
- `% cv ~> terminals~> pins~> status` - Returns nil for all terminals
- `% cv ~> terminals~> pins~> status = "placed"` - Sets placed for all terminals

You can even check or set the status via the property editor.

Innovus has the following option (on by default) to counter this issue:

`setOaxMode -instPlacedIfUnknown true` - Sets the status for instances from none/unplaced to `placed`

18.2.4 Useful Utilities and Information

18.2.4.1 Using OpenAccess Mapper for Migration

The OpenAccess Mapper assists to convert the LPP-based shapes to OpenAccess objects and change the layerName for the pins to use the interconnect layers. The OpenAccess Mapper is available at the following path in your installation:

your_Virtuoso_installation/tools/dfII/bin/mapper

The syntax for the mapper is as follows:

```
mapper –objectmap mapFileName | –map mapFileName –lib libName [–cells cellName] [–
views viewName] [–reportallerrors] [–namemap] [–copy]
```

To move the pins to interconnect layers, use LPP-to-LPP mapping (`–map` parameter). The sample layer map file syntax is as follows:

```
MET1PN drawing MET1 drawing
```

To migrate the boundary/blockage LPP shapes to OpenAccess objects, use object mapping (`–objectmap` parameter). The sample object map file syntax is as follows:

```
BOUNDARY drawing boundary pr

MET1 obs blockage route

MET1 dbk blockage fill
```

For more information about the OpenAccess Mapper utility, refer to the chapter "*Post-Processing of the Translated Data*" of the cdboa.pdf document at *your_Virtuoso_installation*/doc/cdboa/cdboa.pdf.

18.2.4.2 Using GDS flow for Setting Up Layer/Object Map Files

The following are samples of the Virtuoso stream out layer and object map files which can be useful in the OpenAccess flow:

- Layer mapping for pin shapes: Assume there are pin shapes and some regular shapes on the same LPP (M4 drawing):
  - If the layer map is "`M4 drawing 8 1`":
    All shapes on `M4 drawing` data will go on the stream layer purpose <`8:1`>.
  - If the layer map is "`M4 drawing 7 1 pin`":
    All pin shapes on `M4 drawing` will go on the stream layer purpose <`7:1`>.

- Object mapping for blockages or boundary: A sample object map file for stream out is as follows:

```
Boundary PR  1 1
layerBlockage routing MET1 2 2 (for "MET obs" lpp)
layerBlockage fill MET1 3 3 (for "MET dbk" lpp)
```

For more information, refer to the Virtuoso `streamOut` document available at `$CDSHOME/doc/transrefOA/transrefOA.pdf`.

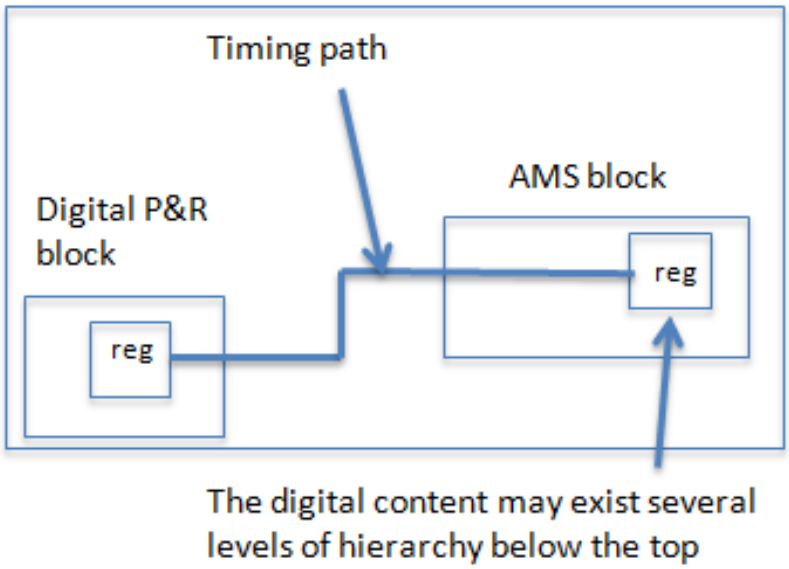
18.2.4.3 Using OpenAccess Database Checker for Verification

The OpenAccess Database Checker (OA DB Checker) allows you make various checks for the technology and design setup to achieve OpenAccess interoperability between Virtuoso and Innovus. Broadly, the OA DB Checker allows you to check for the following:

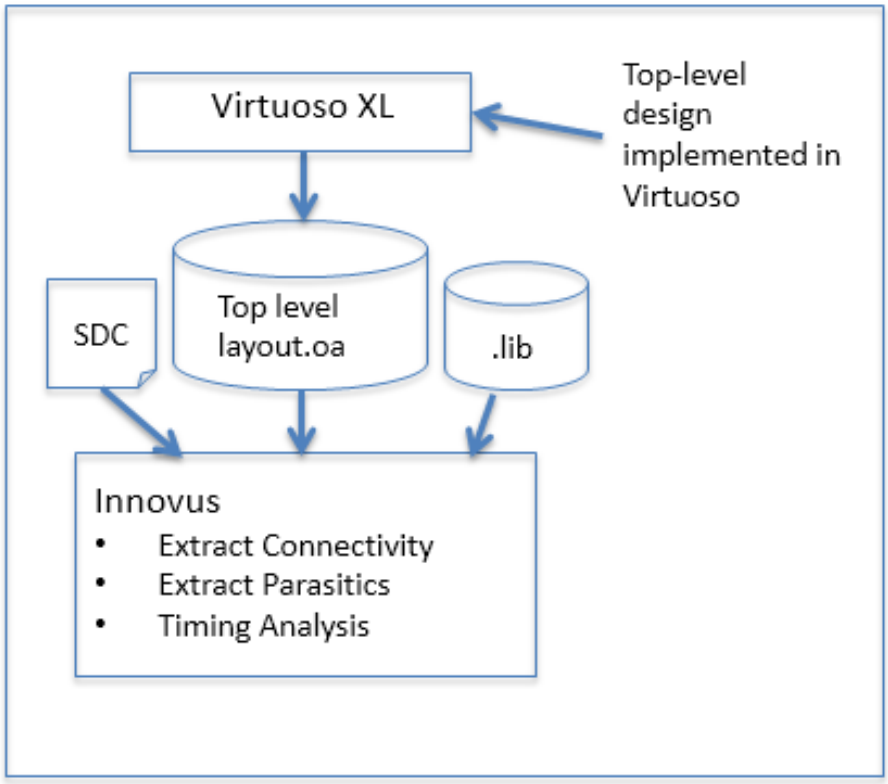
- Technology setup, such as constraint groups and so on
- Design objects and OpenAccess compliance, such as prBoundary
- Setup required for interoperability like PCell cache and bus annotation
- Design constraints

For more information about this utility, refer to the OpenAccess Database Interoperability Checker chapter.

Top-level design (schematic-driven)

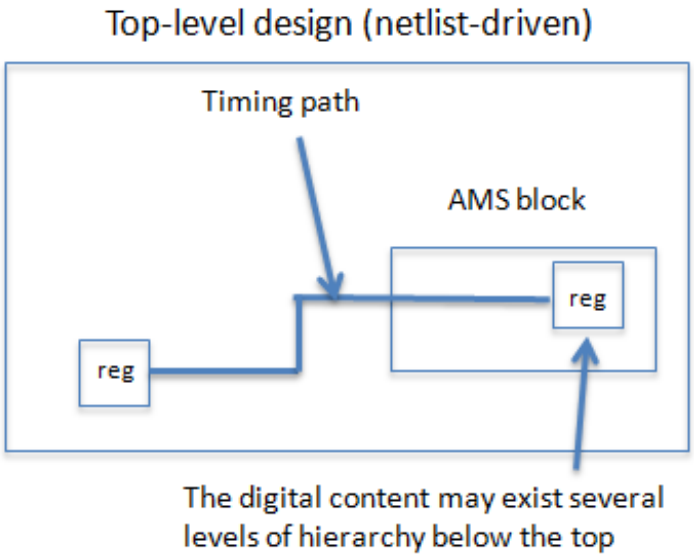


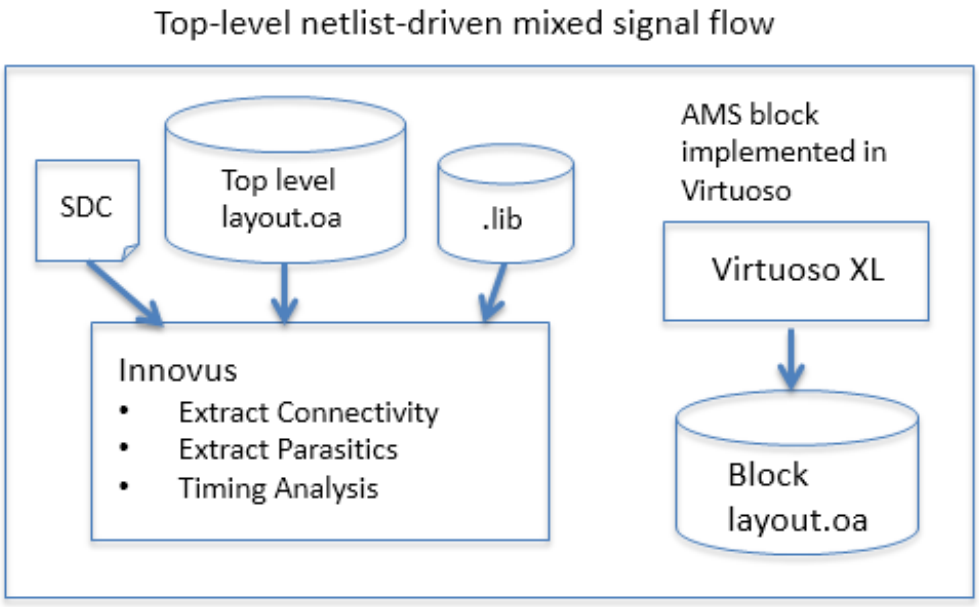
Top-level schematic-driven mixed signal flow



In the above example, the top-level design is implemented in Virtuoso using a schematic for top

level. It can be brought over to Innovus through the OpenAccess database to run timing analysis between digital logic of AMS block and digital P&R block.





The above example shows the case where the top level is implemented in Innovus through a Verilog netlist. After the physical layout is implemented, it is possible to perform timing analysis on the timing paths between the digital logic at the top physical level and the digital logic at the lower physical level of the AMS block.

18.3 Handling Schematic-Driven Designs Implemented Using a Non-Interoperable PDK

As mentioned earlier in the Requirements for OpenAccess Compatibility section, an Interoperable PDK is required to provide an interoperable technology library for the design. For schematic-driven designs, it is possible that the designs were already implemented based on a non-interoperable (or a non-MSOA) PDK long before the interoperable PDK was available. This section discusses what can be done to run static timing analysis in such a situation without making significant changes. Note that the interoperable PDK is still needed but some steps are needed to re-reference the top-level design, or a copy of the top-level design, to the interoperable PDK.

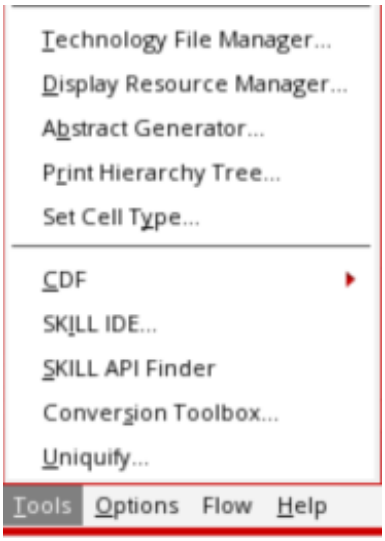
18.3.1 Checking the Technology Database Graph of a Specific Design Library

To check the technology library to which the design is referenced or attached, you can run either Virtuoso or Innovus:

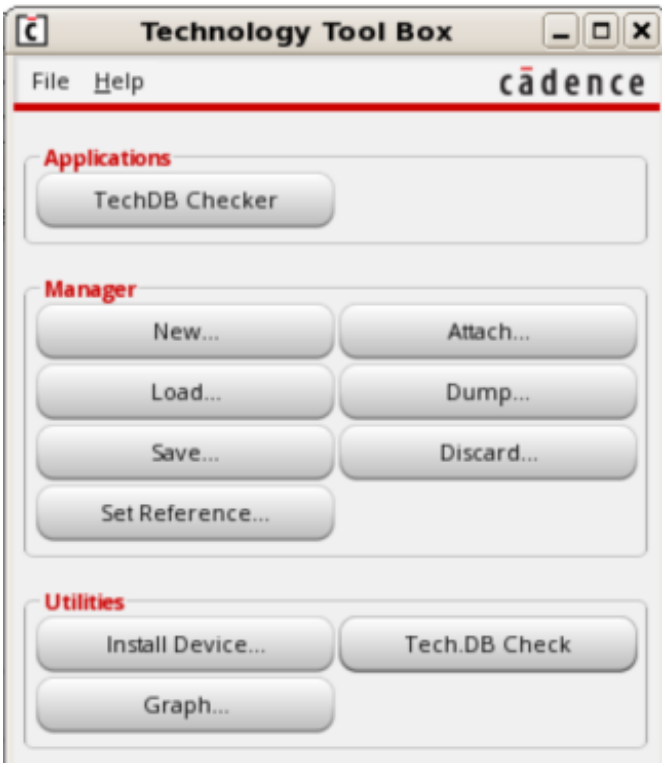
18.3.1.1 Checking in Virtuoso

For Virtuoso, use the Technology Tool Box that can be invoked from the Command Interface Window (CIW). To check the technology library of a specific design library, perform the following steps:

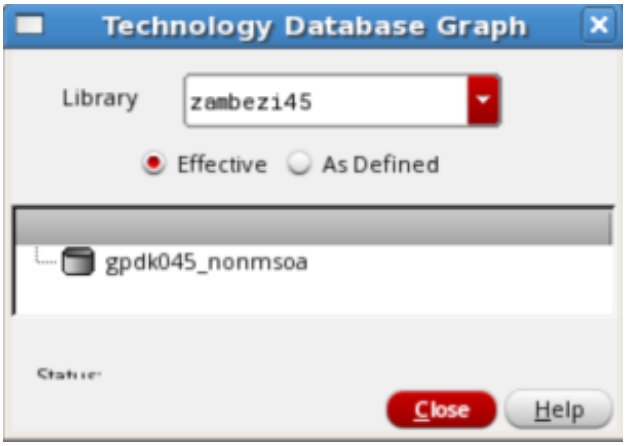
1. In the CIW, select *Tools -> Technology File Manager* to invoke the Technology Tool Box.



2. In the Technology Tool Box, click the *Graph* button to invoke the Technology Database Graph form.



3. In the Technology Database Graph form, click the *Library* drop-down list to select the design library to be verified.



In this example, `zambezi45` is selected. The Technology Database graph form shows that the design library, `zambezi45` is attached to the `gpdk045_nonmsoa` technology library. Note that the

`gpdk045_nonmsoa` technology library is not interoperable (that is,it is non-MSOA).

18.3.1.2 Checking in Innovus

Alternatively, you can determine the tehcnology library in Innovus by using the `report_oa_lib` command with the `-filter tech_graph` option.

For the same example as above, the command will be"

```
> report_oa_lib zambezi45 -filter tech_graph
```

Innouvs will report the following in the log file:

```
Technology Graph:

        Zambezi45

                gpdk045_nonmsoa (attached)
```

18.3.2 Opening a Design Attached to a Non-Interoperable PDK in Innovus

When a non-interoperable PDK-based design is loaded in Innovus, the tool may issue some error messages.

For example, assume that the top-level design is the layout view of cell `LP_pll` from the `zambezi45` library, and a script containing the following commands is run to load the design cellview into Innovus:

```
set init_design_netlisttype {OA}

set init_oa_design_lib {zambezi45}

set init_oa_design_cell {LP_pll}

set init_oa_design_view {layout}

set init_oa_ref_lib {gsclib045}

set init_mmmc_file {scripts/LP_pll.viewDefinition.tcl}

init_design
```

Here, assume that `gsclib045` is the MSOA technology library that is based on an interoperable PDK ITDB. Now, although `gsclib045` is specified with `init_oa_ref_lib`, but as the `zambezi45` design is based on `gpdk045_nonmsoa` library, Innovus will still read the technology information from `gpdk045_nonmsoa`.

Innovus may issue error messages, such as the following:

```
Reading tech data from OA library 'zambezi45' ...

...

**ERROR: (IMPOAX-1304): Base constraint group name 'LEFDefaultRouteSpec' for reading
all the rules is not found. Provide one or change the base constraint group using 'set
init_oa_default_rule <name>' and read the OA database again.

**ERROR: (IMPSYT-6692): Invalid return code while executing 'run.tcl' was returned and
script processing was stopped. Review the following error in 'f1.tcl' then restart.

**ERROR: (IMPSYT-6693): Error message: run.tcl: **ERROR: (IMPSYT-16006): Fail to read
LEF information from OA reference libraries. Check the OA database provided and your
inputs.
```

18.3.3 Ways To Switch a Design Library from a Non-Interoperable to an Interoperable PDK

The loading of the design requires the technology library to be based on an interoperable PDK. You can resolve the above issue, without having to re-implement the design, by following either of the two approaches described below:

18.3.3.1 Re-Referencing the Design Library to an Interoperable PDK

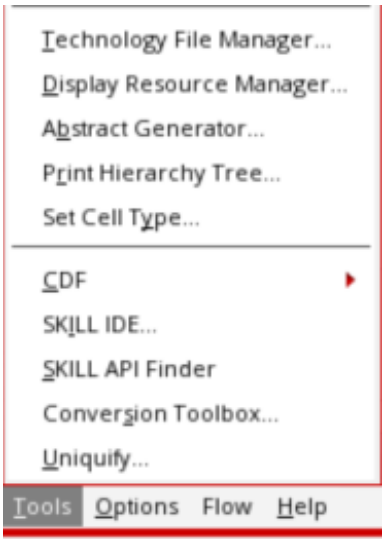
Assume that the interoperable PDK is an ITDB with the following hierarchy:

`gsclib045` (second level) => `gpdk045_msoa` (base level)

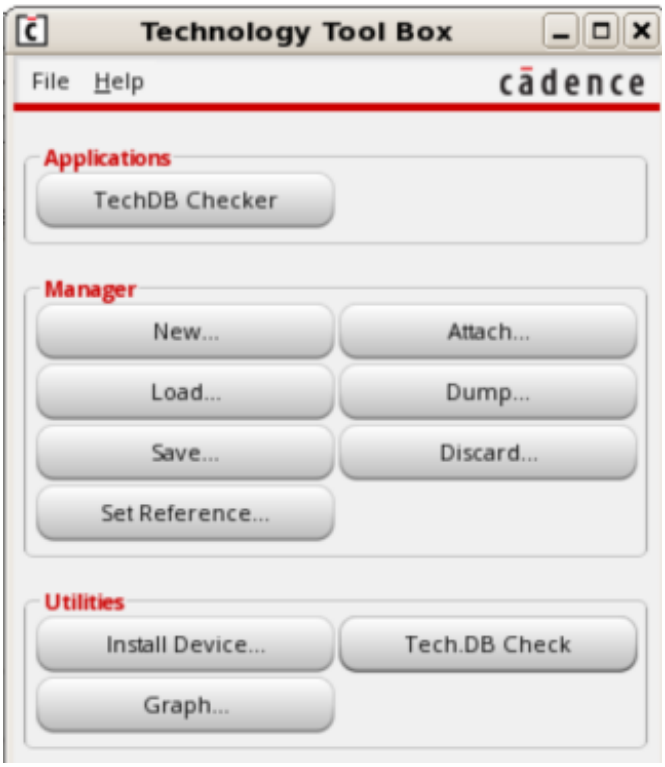
If the original design is attached to a non-MSOA PDK, the `update_oa_lib` command in Innovus can be used to switch the kind of relationship from “Attach” to “Reference”:

For example, to switch the `zambezi45` design library to reference to the `gsclib045` library, perform the following steps:

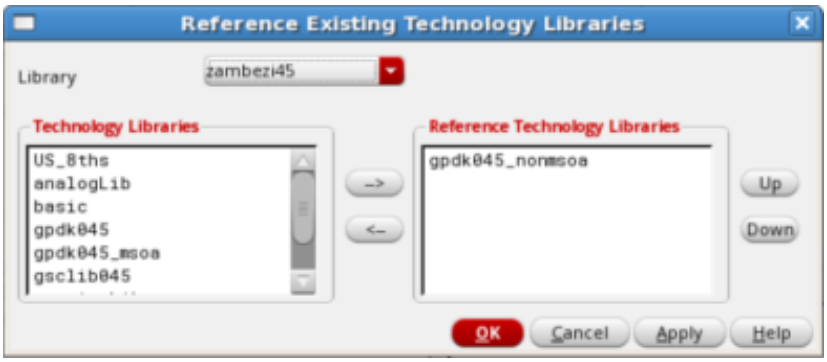
1. Run the following command in Innovus:
   ```
   update_oa_lib -tech_attach_to_reference zambezi45
   ```
   Upon completion, Innovus issues a message similar to the following:
   ```
   Successfully converted the library 'zambezi45' to use a tech reference from the
   attached tech.
   ```
2. In Virtuoso, in the CIW, select *Tools -> Technology File Manager* to invoke the Technology Tool Box.



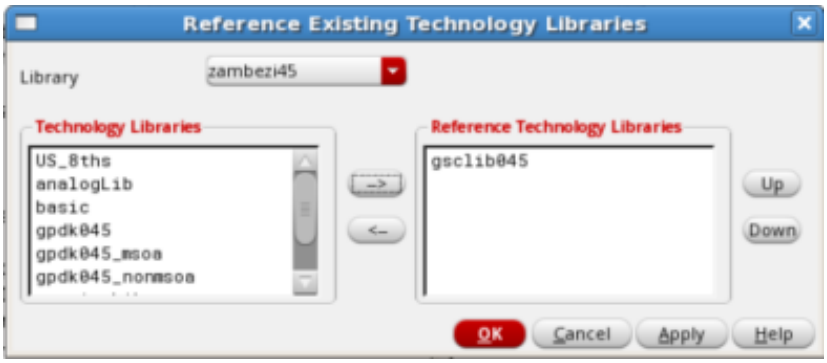
3. In the Technology Tool Box, click the *Set Reference* button to open the Referencing Existing Technology Libraries form.



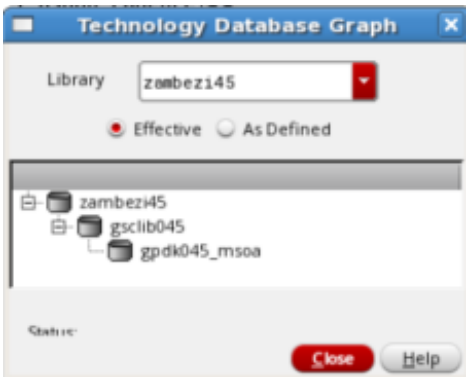
4. From the *Library* drop-down list in the Referencing Existing Technology Libraries form, select the zambezi45 library:



5. Select `gpdk045_nonmsoa` in the *Reference Technology Libraries* list and click the **<-** button so that it is no longer the reference library. Then, select `gsclib045` in the *Technology Libraries* list and click the **->** button so that it appears under the *Reference Technology Libraries* list as shown below:



6. Click the *OK* button to commit the changes.
7. Now, verify the technology graph of the design library by performing the steps described in the Checking the Technology Database Graph of a Specific Design Library section.

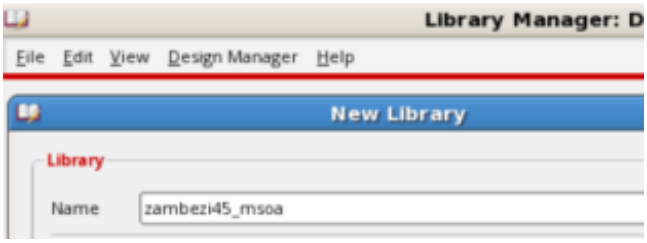


Note that the `zambezi45` design library now references the `gsclib045` library.

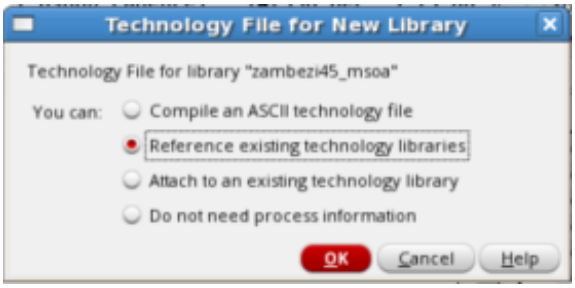
18.3.3.2 Copying the Top-Level Cellview to a New, Interoperable PDK-Based Design Library

The second approach is to create a new design library that is based on the interoperable PDK and copy the top-level cellview to this new design library. Note that only the top-level cellview is to be copied; copying of the cellviews of lower levels are not required. Then, load the top-level cell view from the new design library into Innovus for timing analysis:

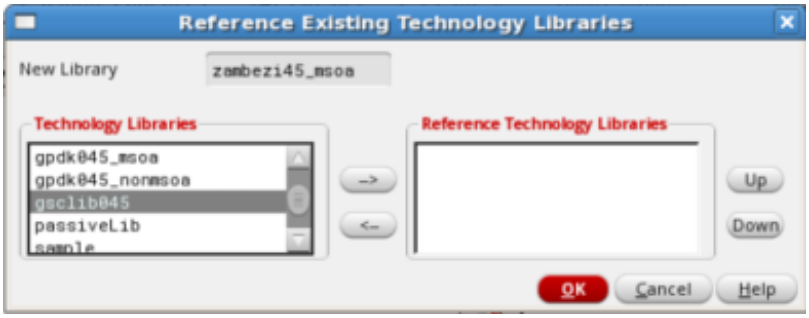
1. In the Library Manager, select *File -> New -> Library* to open the New Library form. Then, specify a new design library name, such as `zambezi45_msoa` and click the *OK* button.



2. In the Technology File for New Library form, select the *Reference existing technology libraries* option and click the *OK* button.



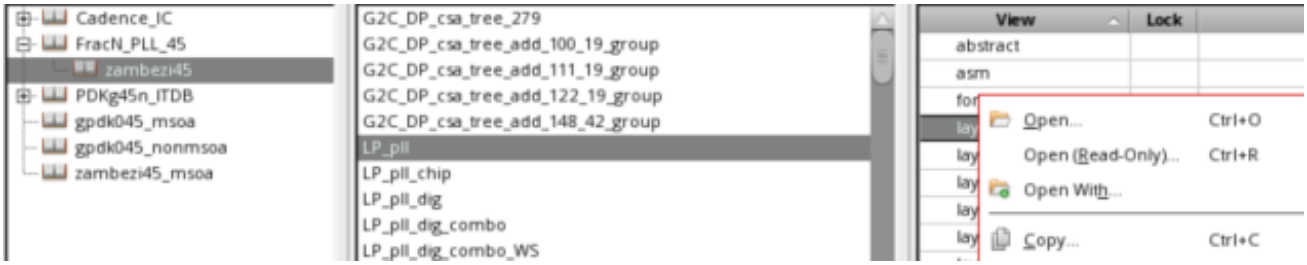
3. In the Reference Existing Technology Librariesform, scroll to the desired MSOA library (for e.g. `gsclib045`), select it and click the **->** button to move it to the *Reference Technology Libraries* list.



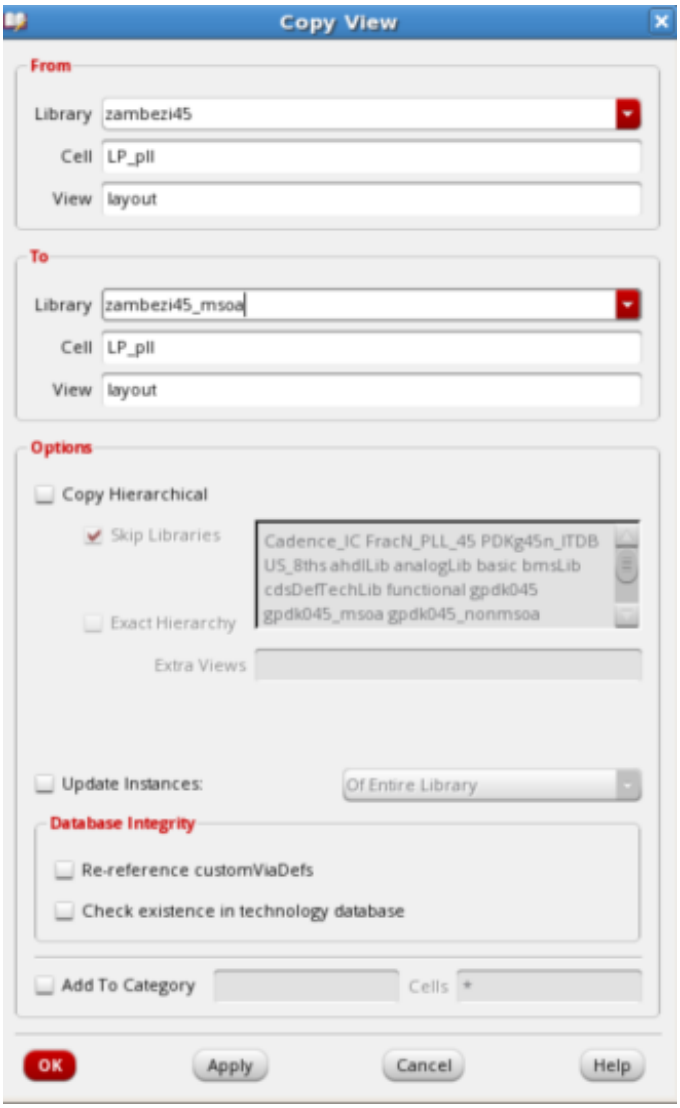
4. Click the *OK* button to commit the change.



5. In the Library Manager, select the top-level cellview of the original design library and choose *Copy.*



6. In the Copy View form, select *Zambezi45_msoa* from the *Library* drop-down list in the *To* section. Keep the *Cell* and *View* names in the *To* section the same as that in the From section. Click *OK* to commit the copy.



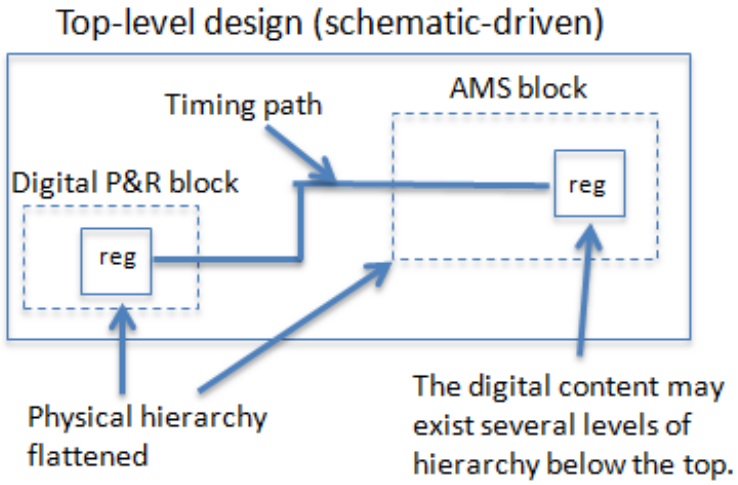
7. Load this cellview `zambezi45_msoa LP_pll layout` into Innovus to run the static timing analysis flow:

```
set init_design_netlisttype {OA}
set init_oa_design_lib {zambezi45_msoa}
set init_oa_design_cell {LP_pll}
set init_oa_design_view {layout}
set init_oa_ref_lib {gsclib045}
set init_mmmc_file {scripts/LP_pll.viewDefinition.tcl}
init_design
```

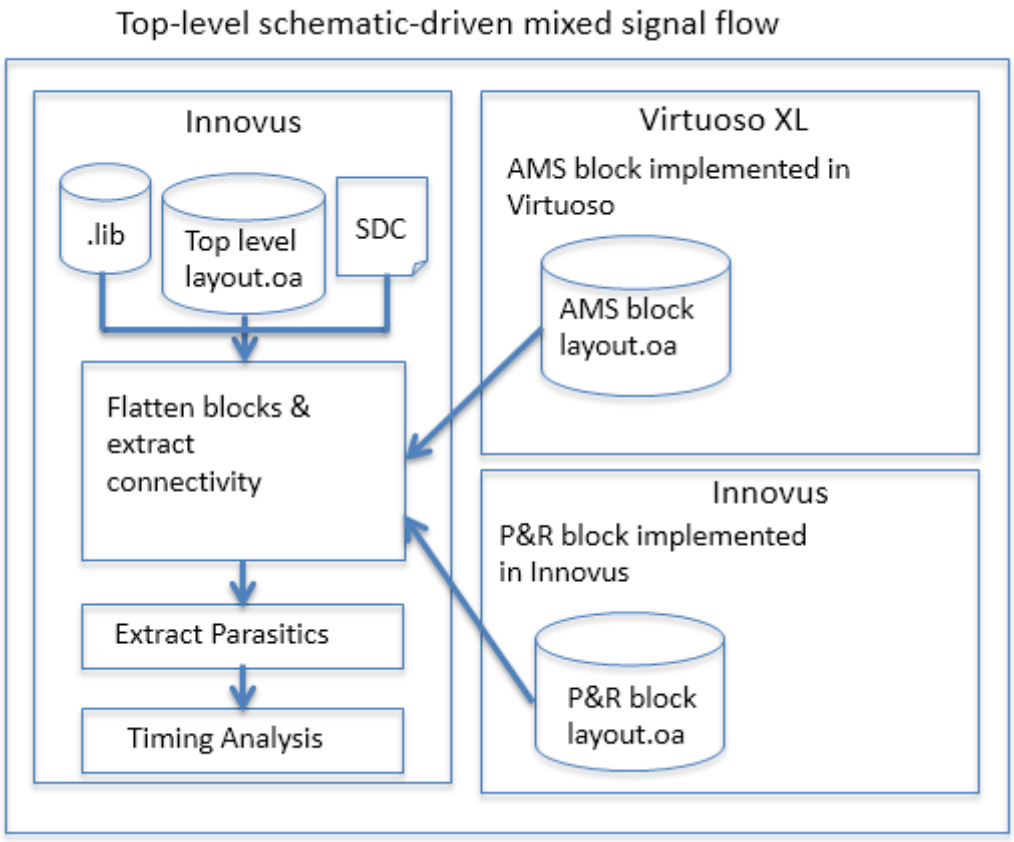
18.4 Running STA by Flattening the Design

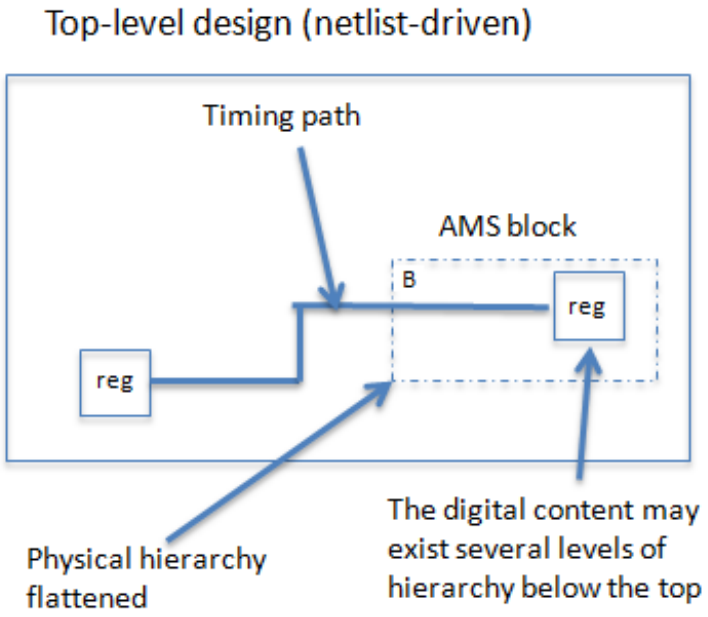
In this method, the Innovus command `assembleDesign` is used to flatten the physical hierarchy to bring the instances and wires at the lower physical level to the top for parasitic extraction and timing analysis.

Note that the physical hierarchy of the design is flattened only to enable static timing analysis. It does not alter the physical structure of the design in any way.

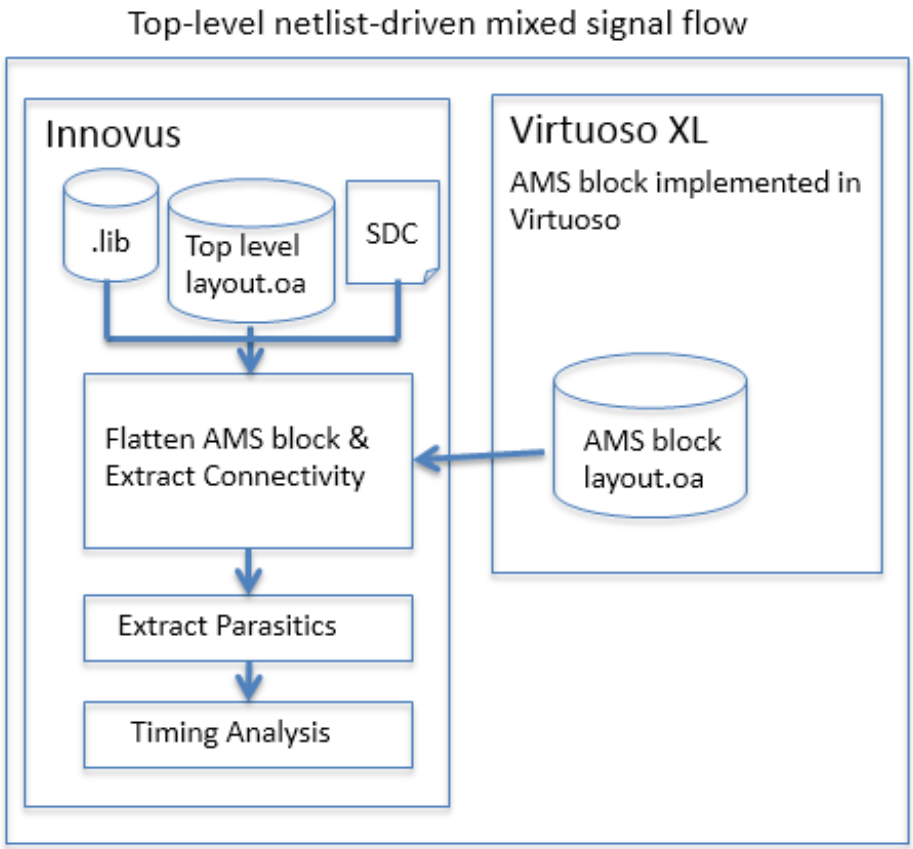


To time a path between registers inside a digital P&R block and an AMS block, the following shows the flow diagram of running STA using the flatten approach on a top-level, schematic-driven design.


Top-level schematic-driven mixed signal flow
Innovus
.lib
Top level layout.oa
SDC
Flatten blocks & extract connectivity
Extract Parasitics
Timing Analysis
Virtuoso XL
AMS block implemented in Virtuoso
AMS block layout.oa
Innovus
P&R block implemented in Innovus
P&R block layout.oa



For a netlist-driven, top-level design, the following shows the flow diagram of running STA on the timing paths between the register at the top and the register inside the AMS block using the flatten approach.



18.4.1 Steps for Running STA Using the Flatten Approach

You can run STA on your design using the following steps:

1. Load the top-level design
2. Flatten the blocks
3. Load and commit power intent, if necessary
4. Specify Parasitic Extraction options
5. Run static timing analysis

18.4.1.1 Load the top-level design

If the top-level design is originally a schematic-driven design implemented by Virtuoso, the `init_design` command should be used to load the OpenAccess design:

```
set init_design_netlisttype {OA}
set init_oa_design_lib {mylib}
set init_oa_design_cell {top}
set init_oa_design_view {layout}
set init_oa_ref_lib {gsclib045}
set init_pwr_net {VDD AVDD}
set init_gnd_net {GND AGND}
set init_mmmc_file {viewDefinition.tcl}
init_design
```

For a top-level design that is netlist-driven, `init_design` can be used to start the place and route flow. The following example shows how to specify the init global variables.

```
set init_design_netlisttype {Verilog}
set init_verilog {top.v}
set init_oa_ref_lib {gsclib045}
set init_pwr_net {VDD AVDD}
set init_gnd_net {GND AGND}
set init_mmmc_file {viewDefinition.tcl}
init_design
```

If the design is saved by Innovus into a cellview, `restoreDesign` can be used to retrieve the OpenAccess database.

Note: Innovus uses the following order of precedence to determine which technology library to use:

1. *lib* from `restoreDesign -cellview "lib cell view"`
2. Library specified with `init_oa_design_lib`
3. The first library specified in the `init_oa_ref_lib` list

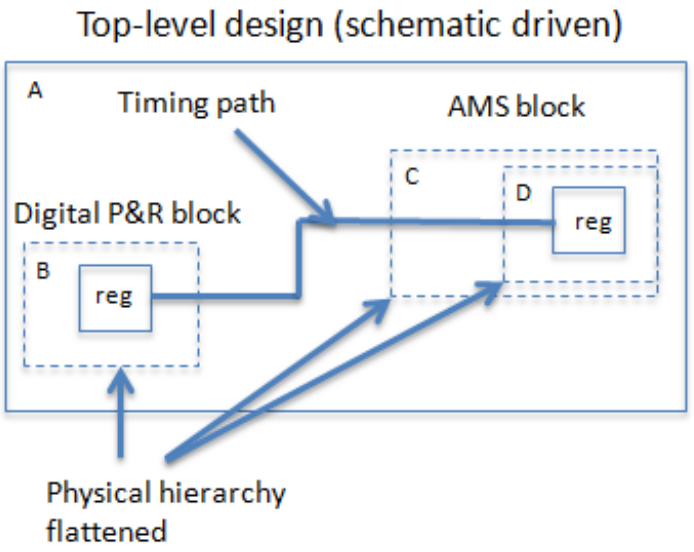
Therefore, when using an OpenAccess cellview as the design, the tech lib always comes from the design library that contains the cellview. In this case, the `init_oa_ref_lib` list is used only for loading cells and not for technology information. The first lib in the `init_oa_ref_lib` list is used for technology information only when the design is read from a Verilog netlist.

18.4.1.2 Flatten the blocks

For the top-level, schematic-driven design shown below, the two blocks can be flattened by running the following commands:

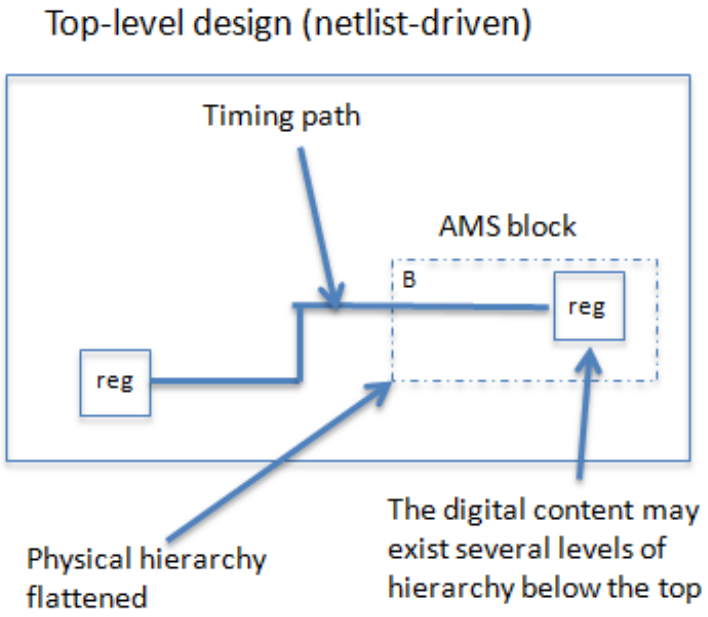
```
assembleDesign -block {mylib B layout} -block {mylib C layout}

assembleDesign -block {mylib D layout}
```



Similarly, to flatten an AMS block that has digital logic one level below the top, the following Tcl command can be run:

```
assembleDesign -block {mylib B layout}
```



Note: In the above example, `assembleDesign` is run in the incremental mode. You can also run `assembleDesign` in the batch mode. Refer to the section "*Different Ways of Running assembleDesig*n" for details.

18.4.1.3 Load and commit power intent if necessary

This step is required only when the digital logic to be timed belongs to more than one power domain. In such a situation, you have to load and commit the Common Power Format (CPF) file because the CPF file provides information about the instance's power domain. It is possible that two instances placed in different power domains are of the same cell type (same physical design). However, when the instances are placed in different power domains with connections to different power signals (for example, one is dvdd and the other is VDDsw), a CPF file is necessary to let Innovus know which timing library set should be used for these two instances.

An example of the Tcl script for this step is:

```
read_power_intent -cpf top_design_flat.cpf

commit_power_intent -keepRows
```

The main purpose of specifying `-keepRows` is to instruct Innovus to keep the rows of the digital P&R block (if present). It is recommended to use this option for `commit_power_intent` in hierarchical flow (after `assembleDesign` is used). If the `-keepRows` option is not specified, `commit_power_intent` removes the existing rows by default and recreates them. If the rows of the assembled block are not aligned with the rows at the top, the newly created rows will be misaligned with the instances that are brought to the top from the P&R block.

18.4.1.4 Specify parasitic extraction options

Typically, when the design is already routed, the extraction engine should be set to the `postRoute` mode to enable detailed extraction of the wires and to report coupling.

```
setExtractRCMode -engine postRoute
```

For accuracy, you can use the correct extraction engine setting to use standalone Quantus or Integrated Quantus (IQuantus) from Innovus. To invoke the IQuantus extraction engine,

```
setExtractRCMode -effortLevel high
```

To invoke the standalone Quantus extraction engine, use the signoff option and provide a layer-mapping file between the Quantus technology file and the technology library used in Innovus.

```
setExtractRCMode -effortLevel signoff

setExtractRCMode -lefTechFileMap Quantus_LEF.layermap_10lm.oxide
```

18.4.1.5 Run timing analysis

To run timing analysis, use:

```
report_timing
```

or

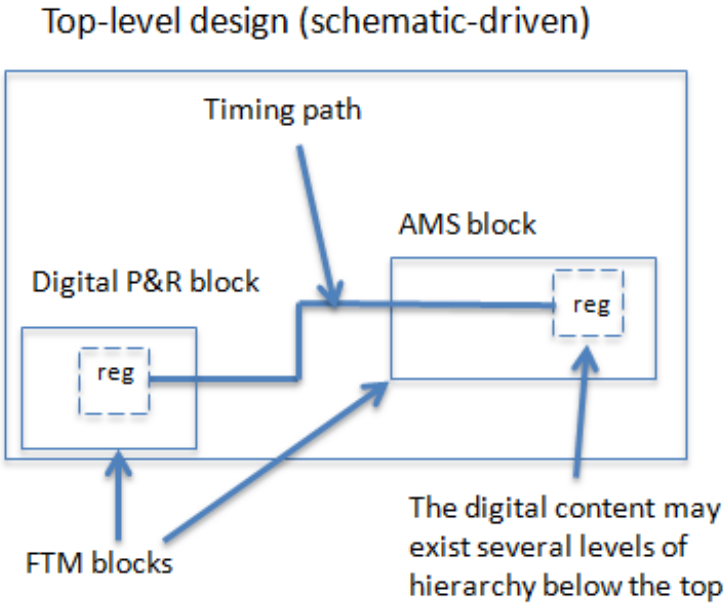
```
timeDesign -postRoute
```

The above commands will automatically run parasitic extraction before analyzing the timing path, if the tool has not performed parasitic extraction earlier.

18.5 Running STA by Using the FTM

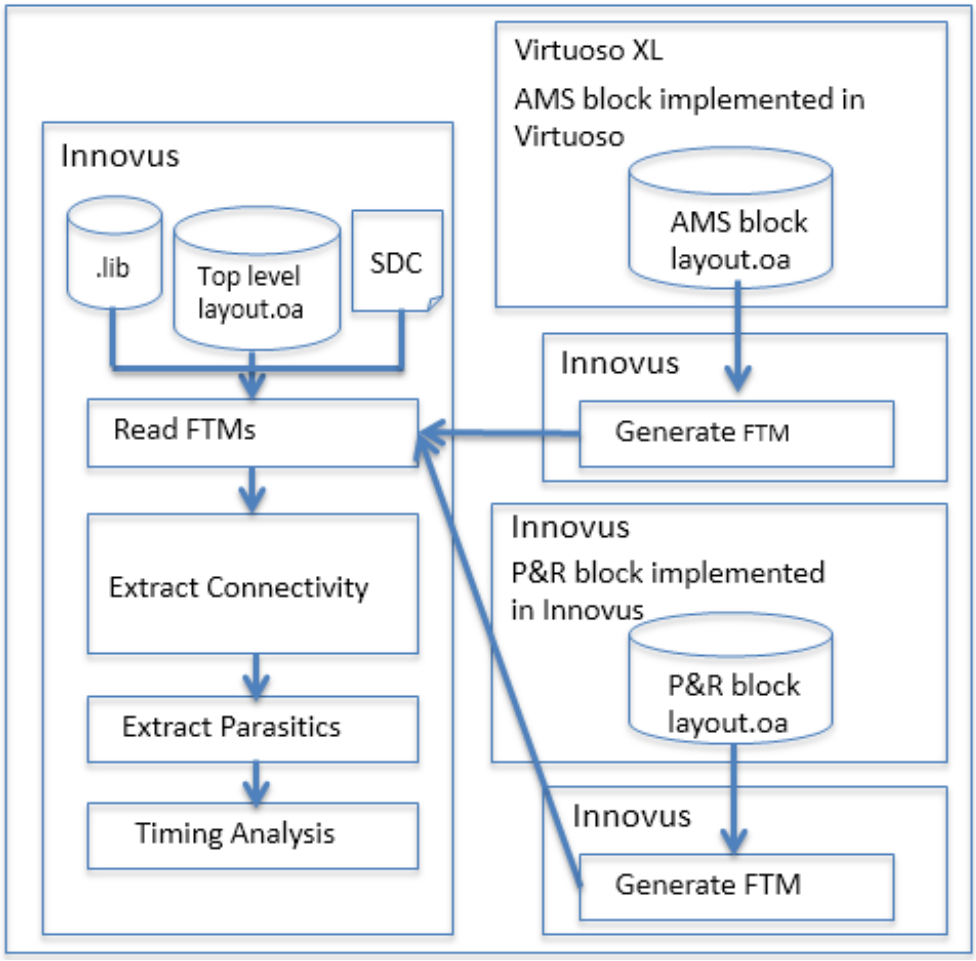
The process of running STA by using the FTM can be divided into two main tasks:

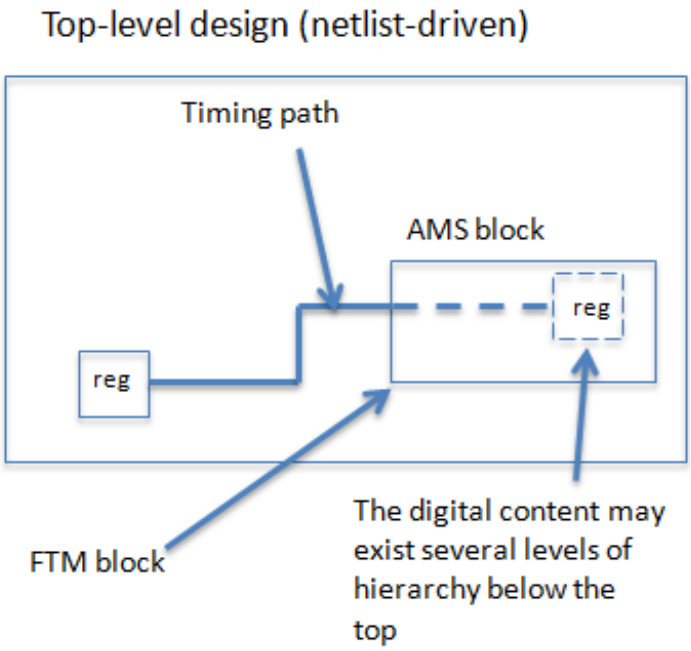
- Generating the FTM for each block
- Running STA on your top-level design with the FTM of blocks



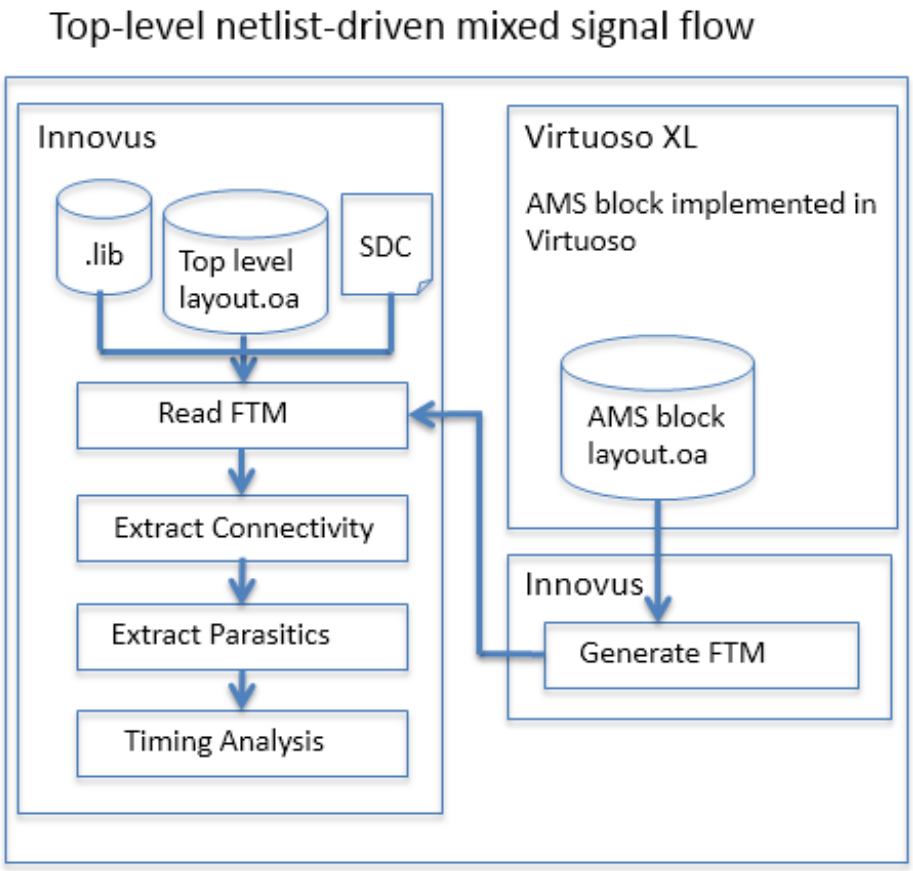
The following shows the flow diagram of running STA using the FTM approach on a top-level schematic-driven design.

Top level schematic driven mixed signal flow





The following diagram depicts the flow for running STA using the FTM approach on a top-level netlist-driven design.


Top-level netlist-driven mixed signal flow
Innovus
.lib
Top level layout.oa
SDC
Read FTM
Extract Connectivity
Extract Parasitics
Timing Analysis
Virtuoso XL
AMS block implemented in Virtuoso
AMS block layout.oa
Innovus
Generate FTM

18.5.1 Generating the FTM for Each Block

You can generate the FTM for your blocks using the following steps:

1. Load the block design.
2. Flatten sub-blocks, if necessary.
3. Load and commit power intent, if necessary.
4. Specify the parasitic extraction options.
5. Create and save the FTM.

18.5.1.1 Load the block design

If the block design is implemented using Innovus, `restoreDesign` can be used to load the design. If the block design is originally a schematic-driven design implemented by Virtuoso, the `init_design` command should be used to load the OpenAccess design. For example, to create the FTM block for block B below, run `init_design` with the following:

```
set init_design_netlisttype {OA}

set init_oa_design_lib {mylib}

set init_oa_design_cell {B}

set init_oa_design_view {layout}

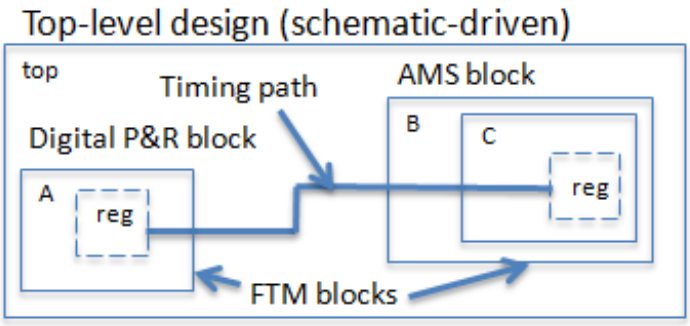
set init_oa_ref_lib {gsclib045}

set init_pwr_net {VDD AVDD}

set init_gnd_net {GND AGND}

set init_mmmc_file {viewDefinition.tcl}

init_design
```



18.5.1.2 Flatten sub-blocks if necessary

For the example shown above, because the logic instances to be timed are at a lower level (in the sub-block C), it is necessary to flatten the sub-block using `assembleDesign`.

```
assembleDesign -block {mylib C layout}
```

18.5.1.3 Load and commit power intent if necessary

If the block is a low power design, load and commit the CPF file. For example:

```
read_power_intent -cpf block_flat.cpf

commit_power_intent -keepRows
```

18.5.1.4 Specify the parasitic extraction options

Typically, when the design is already routed, the extraction engine should be set to the postRoute mode to enable detailed extraction of the wires and report coupling.

```
setExtractRCMode -engine postRoute
```

For accuracy, you can use the correct extraction engine setting to use standalone Quantus or IQuantus from Innovus.

To invoke the IQuantus extraction engine, use the following command:

```
setExtractRCMode -effortLevel high
```

18.5.1.5 Create and save the FTM

To create the FTM, use the `createInterface-removePowerGroundLogic` command with the `-keepAll` option to keep the full Verilog netlist. This will ensure that there is no trimming of any logic. This command will run RC extraction and save the RC parasitic information and netlist along with the cell view location specified by the `-cellview` option.

```
createInterface-removePowerGroundLogic -cellview {mylib B layout} -keepAll
```

If a design is originally created in Virtuoso, it is common to see power/ground ports and nets created in the netlist database of the FTM model. Tools like Genus do not accept the FTM model with power/ground ports or net. To create an FTM that has no power/ground ports and nets, specify the `-removePowerGround` option.

```
createInterface-removePowerGroundLogic -cellview {mylib B layout} -keepAll -
removePowerGround
```

Note: The saved FTM data will be used in top-level analysis and the library, cell, and view name will be used to help the tool determine the location of the FTM data.

18.5.2 Running STA on Top-level Design with FTM of Blocks

You can run STA on your design with the FTM of blocks using the following steps:

1. Load the top-level design
2. Specify FTM blocks
3. Switch to the ILM view
4. Load and commit power intent, if necessary
5. Specify parasitic extraction options
6. Run timing analysis

18.5.2.1 Load the top-level design

If the top-level design is originally a schematic-driven design implemented by Virtuoso, the `init_design` command should be used to load the OpenAccess design:

```
set init_design_netlisttype {OA}

set init_oa_design_lib {mylib}

set init_oa_design_cell {top}

set init_oa_design_view {layout}

set init_oa_ref_lib {gsclib045}

set init_pwr_net {VDD AVDD}

set init_gnd_net {GND AGND}

set init_mmmc_file {viewDefinition.tcl}

init_design
```

For a top-level design that is netlist-driven, `init_design` can be used to start the place and route flow. The following example shows how to specify the global `init` variables.

```
set init_design_netlisttype {verilog}

set init_design_netlisttype {top.v}

set init_oa_ref_lib {gsclib045}
```

```
set init_pwr_net {VDD AVDD}

set init_gnd_net {GND AGND}

set init_mmmc_file {viewDefinition.tcl}

init_design
```

If the design is saved by Innovus into a cellview, `restoreDesign` can be used to retrieve the OpenAccess database.

Notes:

1. The timing constraint file(s) used for FTM analysis should be a "flat" type constraint file that contains the constraints for the top design and instances inside the block. The top-design-only timing constraint file that constrains till to the boundary of the block should not be used for FTM analysis.
2. For timing analysis done with FTM blocks, the timing constraint file must be specified using the `-ilm_sdc_files` option, whether it is for the `create_constraint_mode` statement in `viewDefinition.tcl`, or for the `update_constraint_mode`, which is an interactive command.

For example, in the `viewDefinition.tcl` loaded into the top-level global file, the timing constraint file to be used for FTM analysis, `top_flat.sdc` is specified as follows:

```
create_constraint_mode -name mode1\

 -sdc_files [list ../DATA/top.sdc] -ilm_sdc_files {../DATA/top_flat.sdc}
```

If the `-ilm_sdc_files` option of `create_constraint_mode` in the `viewDefinition.tcl` is not specified, you can run `update_constraint_mode` after `init_design` as follows:

```
update_constraint_mode -name mode1 -ilm_sdc_files {../DATA/top_flat.sdc}
```

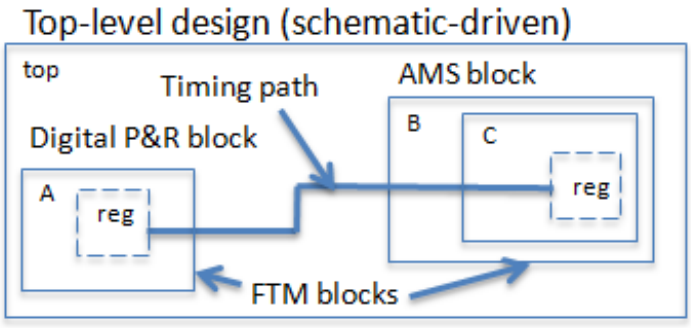
18.5.2.2 Specify FTM blocks

This step is to specify the FTM location for each block to be read as FTM.

```
specifyIlm -cellview {myLib A layout}

specifyIlm -cellview {myLib B layout}
```

By using the `-cellview` option of `specifyIlm`, the tool is able to find the OpenAccess path based on the cds.lib in the current working directory.



Note: If the FTM approach is used for the above example, only one `assembleDesign` is done to flatten sub-block C when generating the FTM for block B. In the flat approach, after the top-level design is loaded into Innovus, block A and block B have to be flattened using `assembleDesign`, while in the FTM approach, they are only specified as FTM blocks (no flattening is done.)

18.5.2.3 Switch to the ILM view

Switch to the ILM view by running the following command:

```
flattenIlm
```

While this command is being run, Innovus reads in the logical netlist and parasitic information of each FTM block. After the command completes running, the tool issues the following statement:

```
*** Switch to ILM view done
```

In the default mode (blackbox view), all FTMs blocks are perceived as blackboxes. The tool sees the connectivity only down to the boundary level (ports) of FTM. Switching to the ILM view enables the tool to see the instances and nets inside the FTM block(s). The physical boundary of the FTM block will be transparent to the tool in this ILM view. The tool now has the logical connectivity information for the instances inside the FTM block(s) to run timing analysis. For example, it is able to report timing on a path that starts from an instance outside the FTM block and ends at an instance inside the FTM block.

Note: Some commands that perform physical-only functions, such as `verifyConnectivity`, are not supported in this mode. Therefore, some of the top menus, such as *Floorplan* and *Verify*, appear grayed out in the main window in this mode. To run these commands, return to the default mode (blackbox view) by specifying the following command:

```
unflattenIlm
```

18.5.2.4 Load and commit power intent if necessary

Similar to the step done in flatten approach (see Steps to Run Static Timing Analysis using the flatten approach), this step is only required when the digital logic to be timed belongs to more than one power domain. An example of the TCL script for this step is as follows:

```
read_power_intent -cpf top_design_flat.cpf
commit_power_intent -keepRows
```

Note: The above two commands have to be specified after `flattenIlm`. The CPF file specified has to be a flat CPF (i.e. not a hierarchical CPF) that contains detailed specification for instances inside the FTM block(s). The macro model, which contains power domain information at the boundary port level, is not detailed enough for the FTM block(s). It is okay to use the macro model to represent non-FTM blocks.

18.5.2.5 Specify parasitic extraction options

Typically, when the design is already routed, the extraction engine should be set to postRoute mode to enable the detailed extraction of the wires and to report coupling.

```
setExtractRCMode -engine postRoute
```

To invoke the IQuantus extraction engine, use the following command:

```
setExtractRCMode -effortLevel high
```

18.5.2.6 Run timing analysis

To generate full and detailed timing reports, specify:

```
timeDesign -postRoute
```

The above command automatically runs parasitic extraction before analyzing the timing path if the tool has not performed parasitic extraction earlier.

Alternatively, you can run the following command to report information about the various paths in the design:

```
report_timing
```

Note: The `report_timing` command must be specified after `flattenIlm`.

Note: Make sure the timing constraint file is specified using `-ilm_sdc_files` (Refer to **Note** in "Load the top-level design" step.)

If the timing constraint files for FTM analysis are not specified using either `create_constraint_mode -ilm_sdc_files` or `update_constraint_mode -ilm_sdc_files`, Innovus issues the following

message:

```
No constrained timing paths found.

Paths may be unconstrained (try '-unconstrained' option)
```

18.6 Difference between Flat and FTM Approaches

The following table compares the FTM and flat approaches.

	FTM	Flat
Dealing with Physical hierarchy	No `assembleDesign` run at the top level	Runs one or more `assembleDesign` till instances are seen on top level
Representation of the block	The block is represented by a Verilog netlist and SPEF file(s) for the RC parasitic which make it portable.	Does not require writing out the logical connectivity in the Verilog netlist and RC parasitic in the form of SPEF for the block. Innovus extracts the logical connectivity and RC parasitic from the flattened block.
Usability and Debugging	• Useful when only the Verilog netlist and the SPEF file of a block (IP) is available (no layout view is given). • Instances and timing path inside the FTM block are not visible in the layout window • `specifyIlm` and `flattenIlm` commands are to be run prior to running `report_timing`.	• Require the layout view of the block. • Instances on the layout window are visible. • Does not require creation of the FTM block and understanding the ILM commands.

18.7 Guidelines for Running STA Flow on Mixed-Signal Design

Consider the following guidelines before running the design data flow:

- The mixed-signal block physical implementation must be done using the VLS-XL connectivity-driven physical implementation methodology using Configure Physical Hierarchy (CPH) and Generate Physical Hierarchy (GPH) methodology. Multiple logical hierarchies in the design netlist are fully supported.
- While in the Virtuoso platform, do not copy a library or cell or view and save it by changing the cell name. This will cause problems while reading the changed cell name view in Innovus.
- The physical database created in Innovus after running assembleDesign on AMS block should not be used for any physical implementation steps or while exporting GDS. The physical database is only for analysis.
- Names of objects (for example, instances) in timing constraint must match the layout design loaded into Innovus. This implies that:
    - If the instance in the layout has a pipeline character (i.e. "|"), the corresponding statement in the timing constraint that has constraint over this instance must contain the pipeline character as well.
    - The hierarchy of the instances must match between the layout and timing constraint.

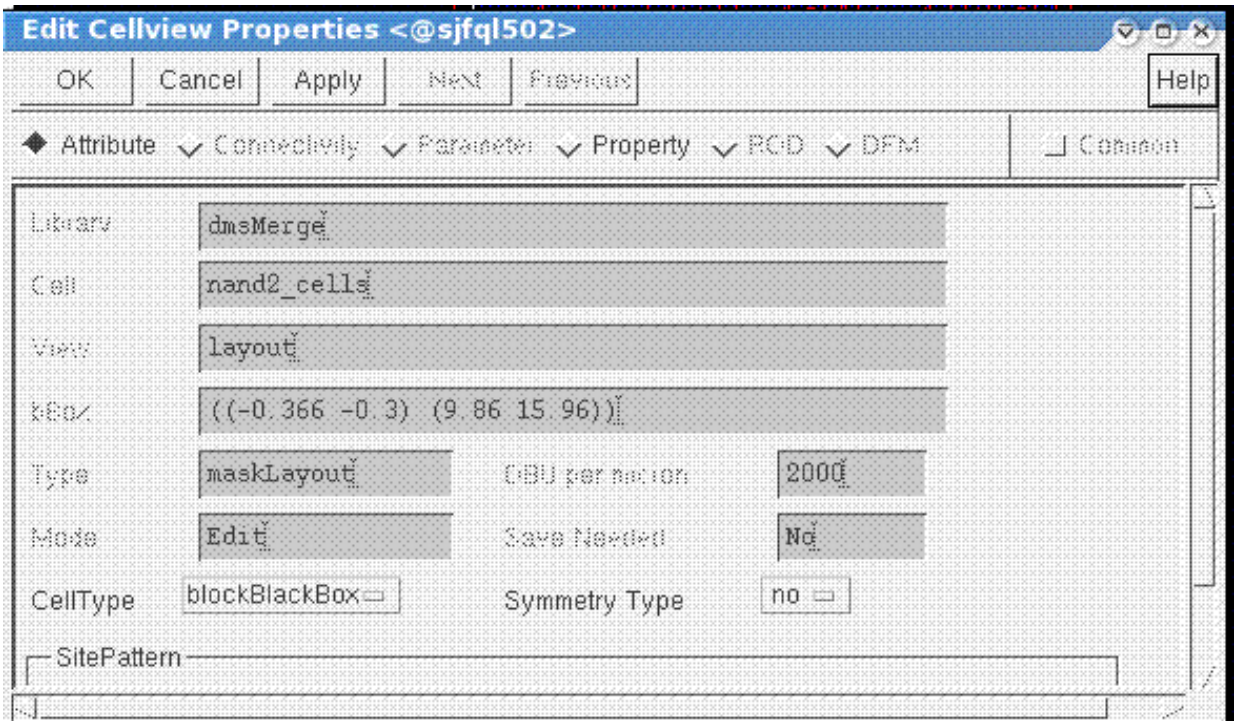
18.8 Creating a Quick Timing Model for Analog IPs in Innovus

In the netlist-on-top flow, the design could contain one or more analog blocks/IPs. The what-if timing analysis capability in Innovus allows for the creation of a quick timing model for such blocks, so that timing analysis can be performed at the top level. For more information on what-if timing model creation capability, refer to the "What-If Timing Analysis" chapter of the *Innovus User Guide*.

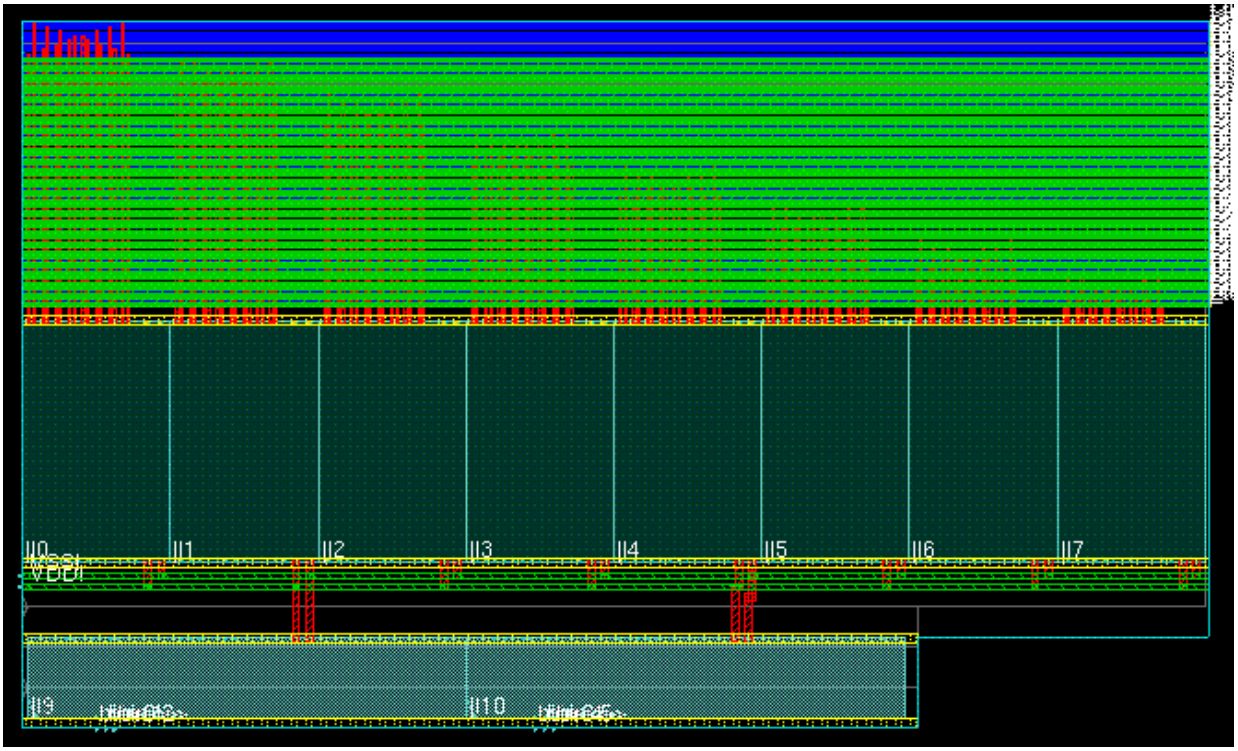
For the what-if timing model creation to work in Innovus, you need to set the blocks as blockBlackBox while using Virtuoso.

Following is the description of how a block is converted to a black box in Virtuoso.

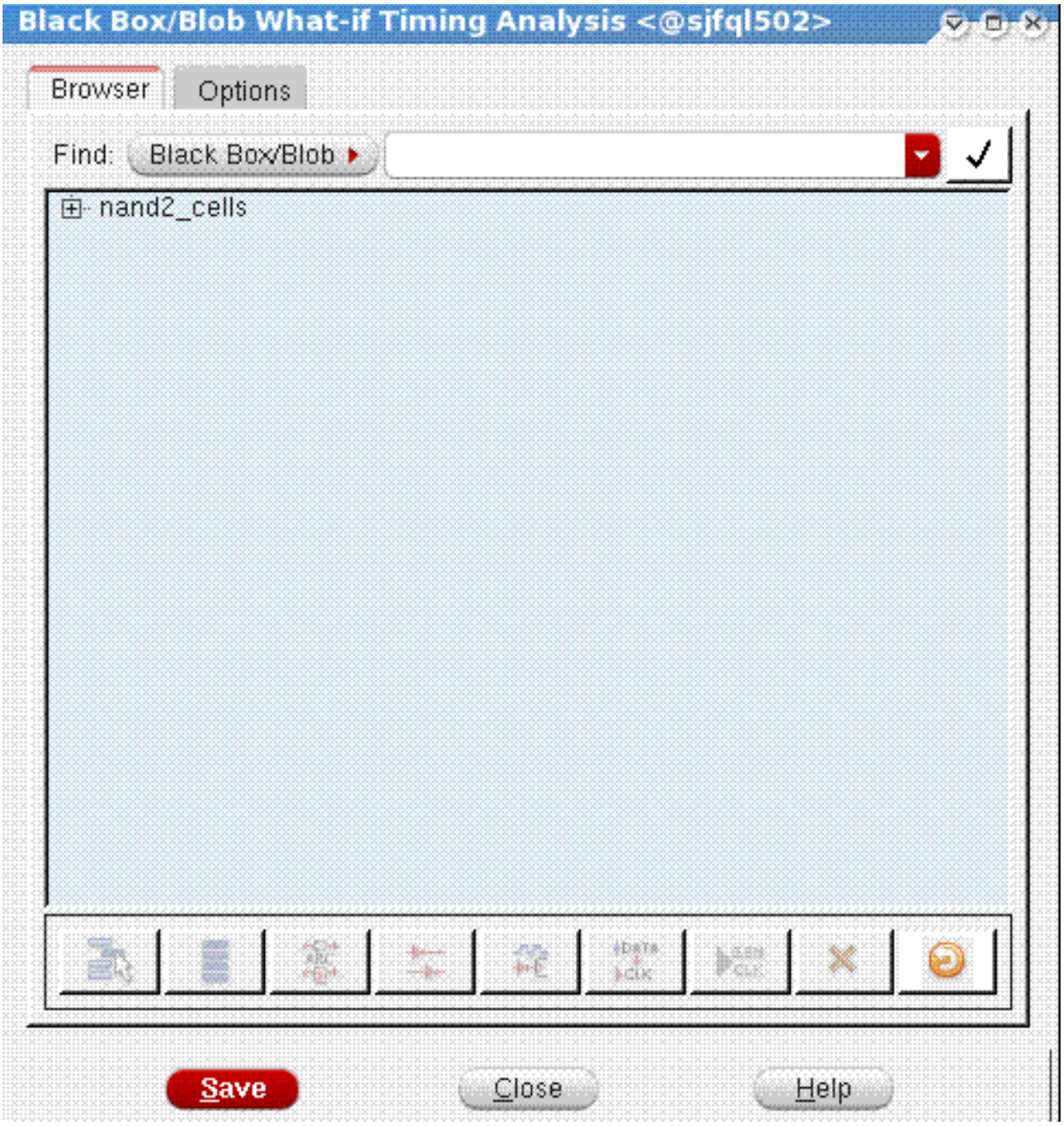
- Set the cellType to blockBlackBox.



- Now when you run `restoreDesign -cellview`, you see the cloned instances as black boxes. This was done with a quick abstract, but can also be done with an abstract cellview.



- The black box can now be modified for What-If Timing analysis.


Black Box/Blob What-if Timing Analysis <@sjfql502>
Browser
Options
Find:
Black Box/Blob
nand2_cells
Save
Close
Help

18.9 Different Ways of Running assembleDesign

The `assembleDesign` command supports different types of data. It can read an FE database, such as a DEF file and a Verilog netlist file, as well as an OpenAccess (OA) database. For the mixed-signal flow, the OA-based database should be used for interoperability. You can use the `-topDesign` and `-block` options of `assembleDesign` to specify the library, cell, and view information for the top-level design and the blocks, respectively. Options such as `-topDir`, `-blockData` and `-blockDir` are meant for non-OA type data and will not be discussed here.

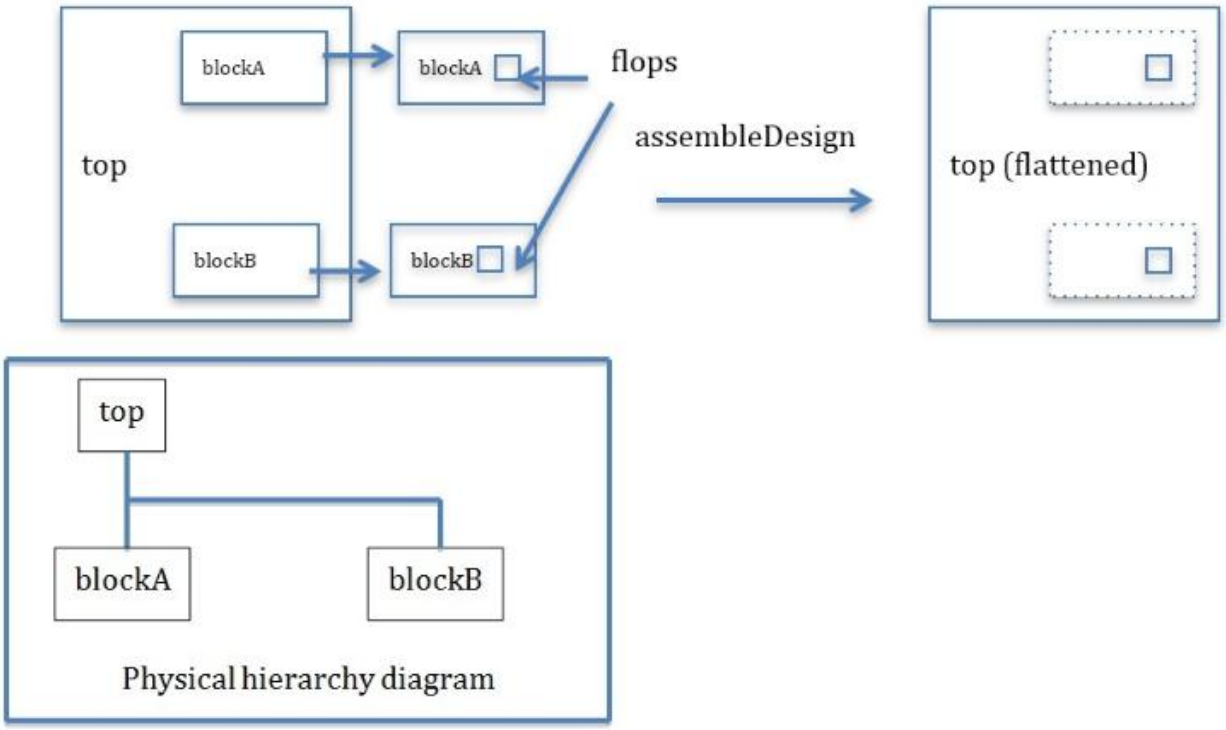
You can run the `assembleDesign` command in two modes, batch and incremental. It is also possible to start an Innovus session by running `assembleDesign` in the batch mode followed by running incremental `assembleDesign` for blocks that are at the next lower hierarchy.

When running in the incremental mode, you can also use the `-allTimingBlocks` option, without the need to specify the library, cell and view information, which allows the tool to perform automatic detection and assembly of blocks that has digital content. The `-allTimingBlocks` option can be used with the `-blockCell` option to force certain blocks to be assembled manually, the `-exceptBlocks` option to override the default list and exclude certain blocks from being assembled, or both.

18.9.1 Running assembleDesign in the Batch Mode

When running `assembleDesign` in the batch mode, a new Innovus session is initiated. After loading the top-level design and block(s), `assembleDesign` performs physical flattening and assembles the top-level design and block(s) together. In this mode, the library, cell, and view names of the top-level design and the block(s) are specified together. The following command shows how to run `assembleDesign` in batch mode to flatten two blocks, *blockA* and *blockB,* which are one physical level below the top-level design, *top*:

```
assembleDesign -topDesign myLib top layout_for_asm -block {myLib blockA layout} -block
{myLib blockB layout} -mmmcFile flat.viewDefinition.tcl
```



18.9.1.1 Loading a Design that Is Not Saved in Innovus

To run `assembleDesign` in batch mode, the top-level design must be such that it can be restored in Innovus.

For Innovus to read or restore an OA-based database, it requires the presence of a `.global` file, which contains the specification of various variables. The variables that have `init` as prefix are used by Innovus to initialize the design. If the top-level design has not been saved in Innovus, for example, if it is implemented through Virtuoso, use the `init_design` command to load it, and then use the `saveDesign` command to migrate it to an Innovus-restorable design. After saving it into a new design, the new database will contain the `.global` file with no change to the original physical layout database.

Here is a sample script to load the top-level design and migrate it to an Innovus-restorable design:

```
set init_design_netlisttype {OA}

set init_oa_design_lib {mylib}

set init_oa_design_cell {top}

set init_oa_design_view {layout}

set init_oa_ref_lib {gsclib045}
```

```
init_design

saveDesign -cellview {myLib top layout_for_asm}

exit
```

Alternatively, you can do the above through an interactive session using GUI as follows:

1. Invoke Innovus to start a new session.
2. From Innovus menu bar, select *File* > *Import Design*.
3. Fill in the variables and click *OK* when done.
4. After the design is loaded, select *File* > *Save Design* from the menu bar.
5. Specify the library name and a new view name. Click *OK* when done.
6. Exit Innovus.

18.9.1.2 MMMC Configuration File and Terminology

To run STA, Innovus requires the Multi-Mode Multi-Corner (MMMC) settings to be specified. This configuration file is typically named as `viewDefinition.tcl`. In this file, timing-related information, such as timing libraries, PVT (process, variation and temperature) operating conditions for the cells in the design, and the timing constraint file are specified. The MMMC configuration file also contains the information needed to run parasitic extraction for wire resistance and capacitance, such as technology file details.

Note: Though named MMMC, the configuration file supports single mode and single corner as well. The single mode and single corner settings are similar but simpler as compared to MMMC settings.

The term *top-level* timing constraint file is ambiguous in hierarchical flow and STA flow for mixed-signal design. To avoid confusion:

- The timing constraint file that contains constraints for only the top-level module (without covering the blocks) is called the top-level only timing constraint file
- The timing constraint file that contains constraints for the full chip (top module + block) is called the flat, full chip timing constraint file. To run STA in a flattened design or when FTM is used, the flat, full chip timing constraint file should be used.

18.9.1.3 Loading the MMMC Configuration File in the Batch Mode

In the batch mode, use the `-mmmcFile` option of `assembleDesign` to load an MMMC configuration file that contains the flat, full chip timing constraint and settings for the flattened design. The MMMC configuration file that might be available in the top-level design will be ignored by `assembleDesign` after the assembly.

Alternatively, the `-cpfFile` option of `assembleDesign` can be used instead of `-mmmcFile` if the CPF file has the MMMC settings.

Note: The `-cpfFile` option instructs `assembleDesign` to load the timing-related settings specified in the CPF file to initialize the MMMC settings for running STA. It does not load all the power intent related specification contained in the CPF file. After `assembleDesign`, the `read_power_intent -cpf` and `commit_power_intent` commands are still needed to be loaded to commit all the power intent specification contained in the CPF file.

18.9.2 Running assembleDesign in the Incremental Mode

To run `assembleDesign` in incremental mode, the top-level design must be already loaded. In the example shown below, the top-level design is not saved in Innovus. The `init_design` command initializes and loads the design with the `init` variables provided. After the top-level design is loaded, the specified block is then flattened using the `assembleDesign` command. An example is shown below:

```
set init_oa_design_lib {myLib}
set init_oa_design_cell {top}
set init_oa_design_view {layout}
set init_design_netlisttype {OA}
set init_mmmc_file {top.viewDefinition.tcl}

init_design
assembleDesign -block {myLib blockA layout} -block {myLib blockB layout}
```

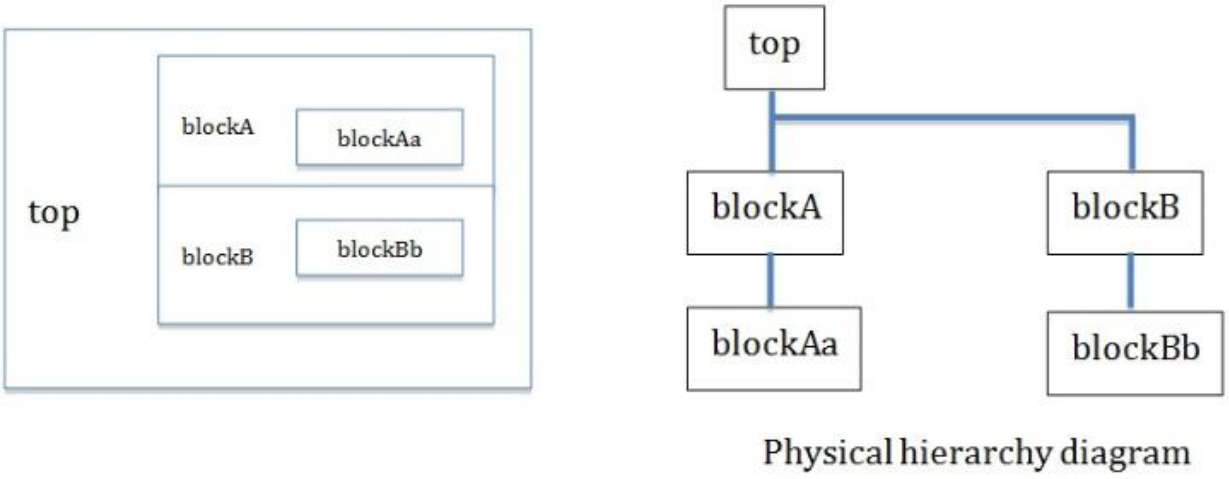
If the top-level design is previously saved in Innovus, the `restoreDesign` command can be used to load the design. Then, run `assembleDesign` with the specified blocks to perform incremental assembly.

```
restoreDesign -cellview {mylib top layout}

assembleDesign -block {myLib blockA layout} -block {myLib blockB layout}
```

The incremental mode allows the `assembleDesign` command to be run multiple times in sequence to flatten blocks of multiple levels of physical hierarchy (for example, blocks within blocks). For example, assume there are two blocks, `blockA` and `blockB`, in the top-level design to be flattened.

`blockA` has a sub-block, `blockAa`, while `blockB` has `blockBb` as its sub-block. Both `blockAa` and `blockBb` are to be flattened.



Physical hierarchy diagram

The following script can be used to assemble them incrementally. Each `assembleDesign` command flattens the design for one level of physical hierarchy.

```
assembleDesign -block {myLib blockA layout} -block {myLib blockB layout}

assembleDesign -block {myLib blockAa layout} -block {myLib blockBb layout}
```

Thus, the incremental capability of `assembleDesign` enables the physical flattening of a design with multiple levels of physical hierarchy through a single Innovus session.

18.9.2.1 Loading the MMMC Configuration File in the Incremental Mode

The top-level design must be loaded before `assembleDesign` is run in incremental mode. In addition, the top-level design must contain an MMMC configuration file. If not, Innovus will go into the physical-only mode. Typically, this MMMC configuration file contains the top-level only timing constraints. To reload a new flat, full chip timing constraint file, you must source it after `assembleDesign` as follows:

```
set init_oa_design_lib {myLib}
set init_oa_design_cell {top}
set init_oa_design_view {layout}
set init_design_netlisttype {OA}
set init_mmmc_file {top.viewDefinition.tcl}
init_design
assembleDesign -block {myLib blockA layout} -block {myLib blockA layout}
source flat.viewDefinition.tcl
```

You can specify an MMMC configuration file that has the flat, full chip timing constraint file when the

design is loaded using `init_design` as follows.

```
set init_mmmc_file {flat.viewDefinition.tcl}
```

In this case, you need not reload the MMMC configuration file after `assembleDesign`. The only drawback for this flow is that Innovus may issue warning and error messages while reading the MMMC configuration file during `init_design`. When the flat, full chip timing constraint file contains constraint statements on objects, such as instances, or paths that are inside the blocks to be assembled, Innovus issues messages that the specified path or objects could not be found. This is because those paths or objects will exist only after `assembleDesign`. Therefore, at this point in time, certain constraint statements in the timing constraint will be deemed to be invalid before `assembleDesign`.

Notes

- The `-mmmcFile` option of the `assembleDesign` command is meant only for batch mode.
- The incremental capability of the `assembleDesign` command does not currently support a design that has a physical netlist. For example, the netlist has `VDD` and `VSS` as input ports in the module. Run the command `saveNetlist design.v` to check whether the netlist has power and ground ports. Run batch mode `assembleDesign` to handle this type of design. One way of running batch mode `assembleDesign` to handle such a design with multiple levels of physical hierarchy can be as follows:

  ```
  #First Innovus session

  assembleDesign -topDesign {myLib top layout_for_asm} -block {myLib blockA
  layout} -block {myLib blockB layout}

  saveDesign -cellview {myLib top layout_stage1}

  exit

  #Second Innovus session

  assembleDesign -topDesign {myLib top layout_stage1} -block {myLib blockAa
  layout} -block {myLib blockBb layout} -mmmcFile flat.viewDefinition.tcl
  ```

18.9.3 Differences between the Batch and Incremental Modes of assembleDesign

	Batch Mode	Incremental Mode
Loading the top-level design	The top-level design and block(s) are loaded using `-topDesign` and `-block`, respectively. The `-topDesign` option is used only in the batch mode.	The top-level design must be loaded first using `init_design` or `restoreDesign`. The top-level design is never specified using `assembleDesign`.
Use of command/ Specification of the block(s)	Use only once. The top-level design and the block(s) must be specified together.	Can be used multiple times to specify different blocks or blocks at different levels of physical hierarchy.
Need to save the top-level design in Innovus	The top-level design must be saved by Innovus before. (`assembleDesign` uses similar function as `restoreDesign` to load the top-level design). For a design from Virtuoso, use `init_design` to load and save the design with a new view name. Then specify the new view name using the `-topDesign` option of `assembleDesign`.	The top-level design does not need to be saved by Innovus before. This means you can use `init_design` to load a design implemented using Virtuoso.
Loading `viewDefinition.tcl`	Use the `-mmmcFile` or the `-cpfFile` option of `assembleDesign` to load `viewDefinition.tcl`. These two options are meant for the batch mode only.	`viewDefinition.tcl` is loaded with `init_design` or `restoreDesign`.

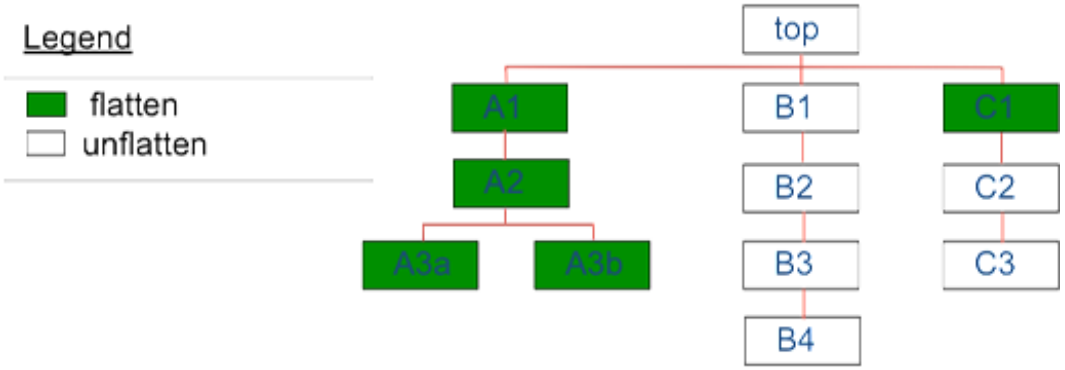
18.9.4 Running assembleDesign with the -allTimingBlocks Option

When the `-allTimingBlocks` option is specified, the `assembleDesign` command uses the following criterion to determine if a cell is to be assembled:

1. A standard cell (with CORE celltype) exists in the cellview.

2. The cellview has a cell that is bound with the timing library, thus requiring the `viewDefinition.tcl` and the timing library files to be loaded in the beginning of the flow.
3. Any of the cellview below the current level of the cell has met either of the above two conditions.

Note: If a cellview itself is bound with timing library, it will not get flattened.



For example, consider a design that has the physical hierarchy shown above. The cellviews at its lower hierarchy bear the following conditions:

- A3a has a standard cell.
- A3b has a cell bound with the timing library.
- None of the B* cells have standard cells or cells bound with the timing library.
- C2 itself is bound with the timing library.

The cellview will be assembled as follows due to the reasons stated below:

- A3a will be flattened because there is standard cell in it.
- A2 and A1 (the cellviews above A3a) will be flattened because A3a has to be flattened.
- A3b will be flattened because there is cell in this cellview bound with the timing library.
- A2 and A1 (the cellviews above A3b) will be flattened because A3b has to be flattened.
  Note: As long as either A3a or A3b is to be flattened, the A2 and A1 cellviews will be flattened as well.
- None of the B* cells are flattened because no digital content is found at any level.
- C1 will be flattened because C2 (i.e. a cell in C1) is bound to the timing library
- C2 and C3 will not be flattened because C2 has to be a leaf cell, which has timing library

bound.

18.9.5 Tips on Running assembleDesign -allTimingBlocks

1. View the log file to see which blocks are flattened.
   One example is shown below:

```
<CMD> assembleDesign -allTimingBlocks -exceptBlocks {pll_lf pll_lpf pll_vcodiv
pll_cp pll_pfd} -blockCell {LP_pll_dig_wSPI}
This command "assembleDesign -allTimingBlocks -exceptBlocks {pll_lpf ..." required
an extra checkout of license invs_ms.
Additional license(s) checked out: 1 'Innovus_MS_Opt' license(s)
Following 4 blocks (lib cell view) would be assembled:
block 1: zambezi45 LP_pll_dig_combo layout
block 2: zambezi45 LP_pll_dig_wSPI layout
block 3: zambezi45 pll_fbdiv layout
block 4: zambezi45 pll_ls_dvdd2avdd layout
```

2. View the verbose log file to see the reason why a block is flattened or not flattened.

   Some sample statements are shown below.

```
[07/09 14:38:44 13s] Block zambezi45/pll_lf/layout would NOT be assembled as it is
excluded by user.
[07/09 14:38:44 13s] Block zambezi45/pll_bypclf/layout would NOT be assembled as
it does not have any instance of leaf cell or timing library cell inside.
[07/09 14:38:44 13s] Block zambezi45/pll_pfd/layout would be assembled as it has
instance of standard cell.
[07/09 14:38:44 13s] Block zambezi45/LP_vco/layout would be assembled as it has
sub block (zambezi45/pll_vco_calsw/layout) which is detected to be assembled.
[07/09 14:38:44 13s] Block zambezi45/pll_ls_dvdd2avdd/layout would be assembled as
it has instance of timing library cell.
[07/09 14:38:44 13s] Block zambezi45/LP_pll_dig_combo/layout would be assembled as
it has instance of block zambezi45/LP_pll_dig_wSPI/layout which is specified by
user to assemble.
[07/09 14:38:44 13s] Block zambezi45/LP_pll_dig_wSPI/layout would be assembled as
it is specified by user to assemble.
```

18.9.6 Caveats for and Limitations of assembleDesign -allTimingBlocks

1. If `assembleDesign -allTimingBlocks` is used to explore and run on design that is not entirely XL compliant (i.e. some blocks are XL compliant while some are not), it may error out and exit if it attempts to flatten any of the blocks that are not XL compliant.
   Use `-exceptBlocks` to exclude those cellviews or blocks that are not XL compliant.
   Alternatively, run `assembleDesign` with the `-block` option.
2. The `-allTimingBlocks` option works for the incremental mode only. For example, it works in the following flow:

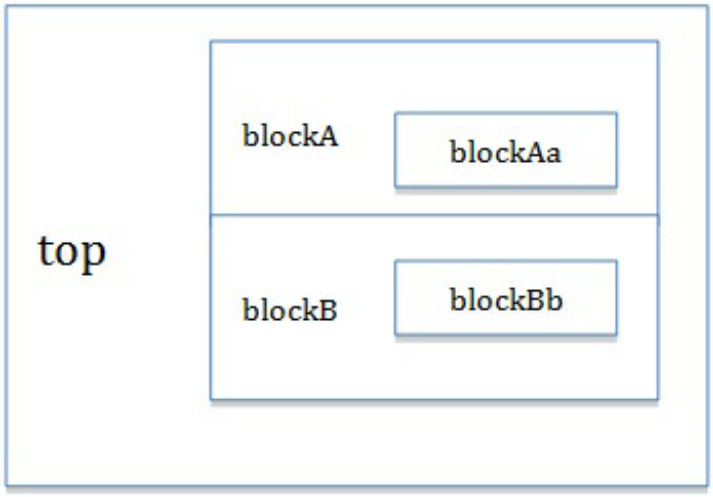
   ```
   source scripts/LP_pll.globals
   init_design
   assembleDesign -allTimingBlocks
   ```

   It is not supported in the batch mode. If `assembleDesign -allTimingBlocks` is run along with `-topDesign`, an error message will be shown.

   ```
   assembleDesign -topDesign {zambezi45 LP_pll layout_init} -allTimingBlocks
   **ERROR: (ENCSYT-35004): Cannot run assembleDesign with -allTimingBlocks option
   because design is not loaded. -allTimingBlocks option is supported in incremental
   assembleDesign only. Load the design and then run assembleDesign.
   ```

18.9.7 Running assembleDesign in the Batch Mode and Subsequently in the Incremental Mode

It is possible to start an Innovus session by running `assembleDesign` in the batch mode, follows by running incremental `assembleDesign` for blocks that are at the next lower hierarchy.



For the same example as described in the Running `assembleDesign` in Incremental Mode section, the following script should show similar results:

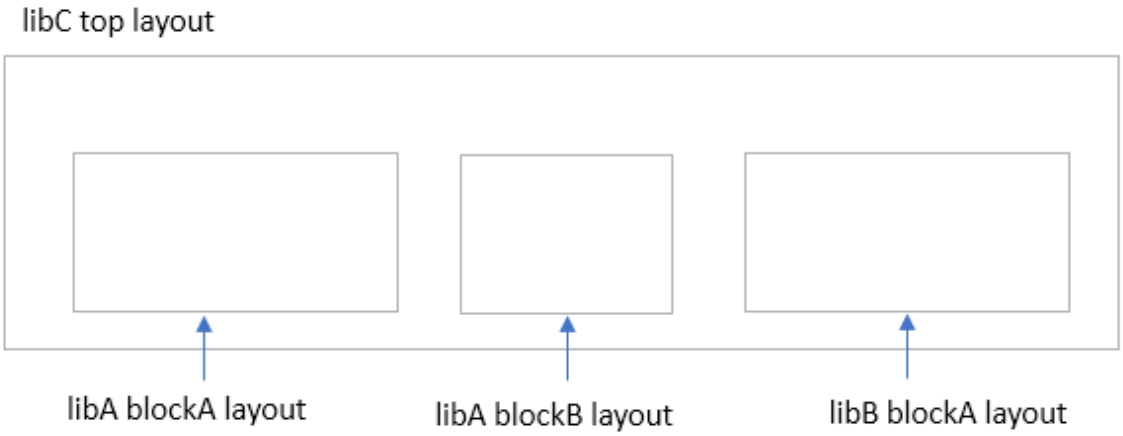
```
assembleDesign -topDesign {myLib top layout} -block {myLib blockA layout} -block {myLib blockB layout}

assembleDesign -block {myLib blockAa layout} -block {myLib blockBb layout}
```

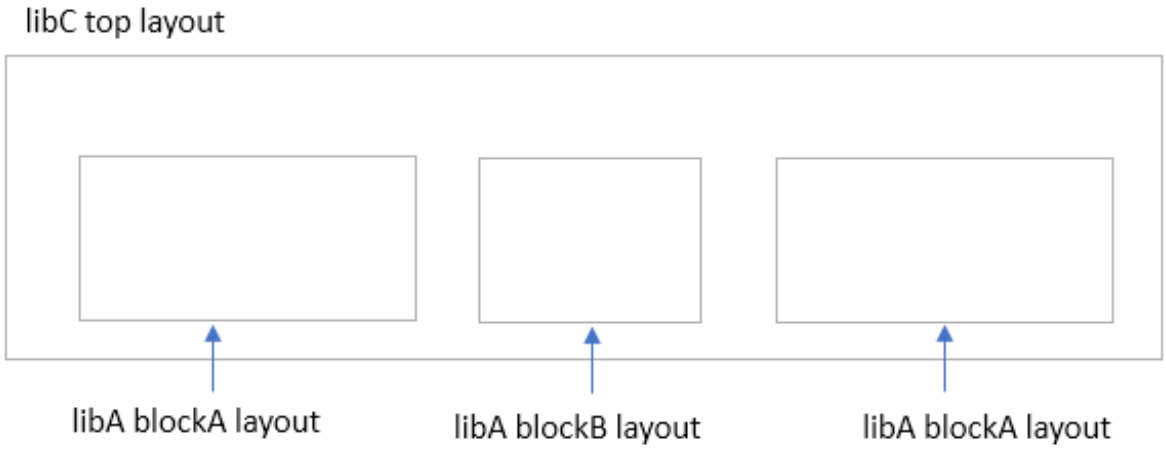
18.9.8 Limitations of assembleDesign

18.9.8.1 Innovus is not able to read instances with the same cell name but different library names

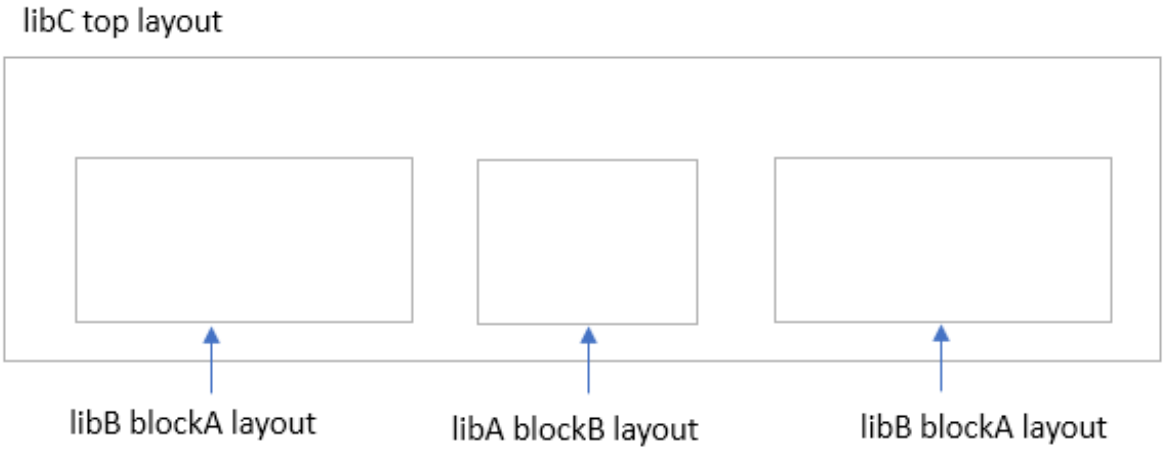
In Virtuoso, a design can have blocks specified with the same cell name but different design library names. However, if such a design is loaded into Innovus, Innovus will see both the cells as having the same design library. For example, consider a design, `{libC top layout}`, which contains three blocks as shown below. Two of the blocks have the same cell name `blockA` but are under different libraries, `libA` and `libB`.



After loading the design, Innovus will read in both cells as either `{libA blockA layout}` or `{libB blockA layout}`, depending on which block it reads first.



Or



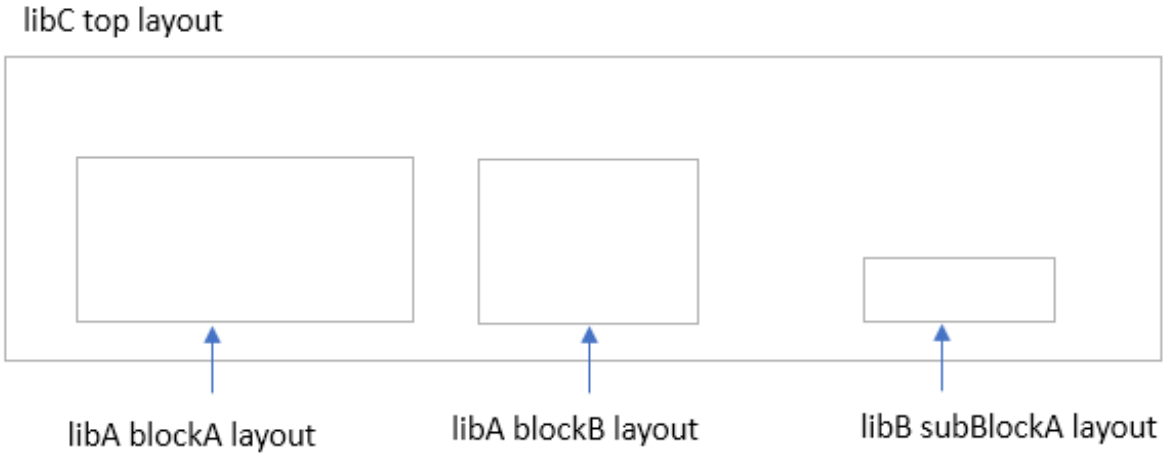
If Innovus reads both cells as `libA blockA layout`, during `init_design`, Innovus will issue a message like the following:

```
**WARN: (IMPOAX-745): Cell 'blockA' from cellview 'layout' has already been read in
from 'libA'. A cell can be read only once, so the second version will be ignored.
Review the setting of the init_oa_ref_lib variable and update it to remove the library
that has the duplicate cell.
```

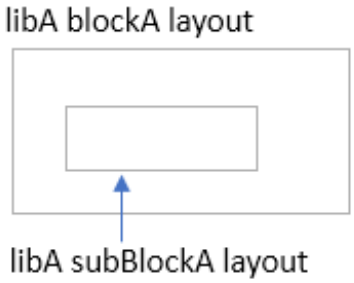
Thus, if `assembleDesign` is run to flatten `blockA`, the flattened content of both the blocks will be the same.

18.9.8.2 assembleDesign is not able to support instances with the same cell name but different library names

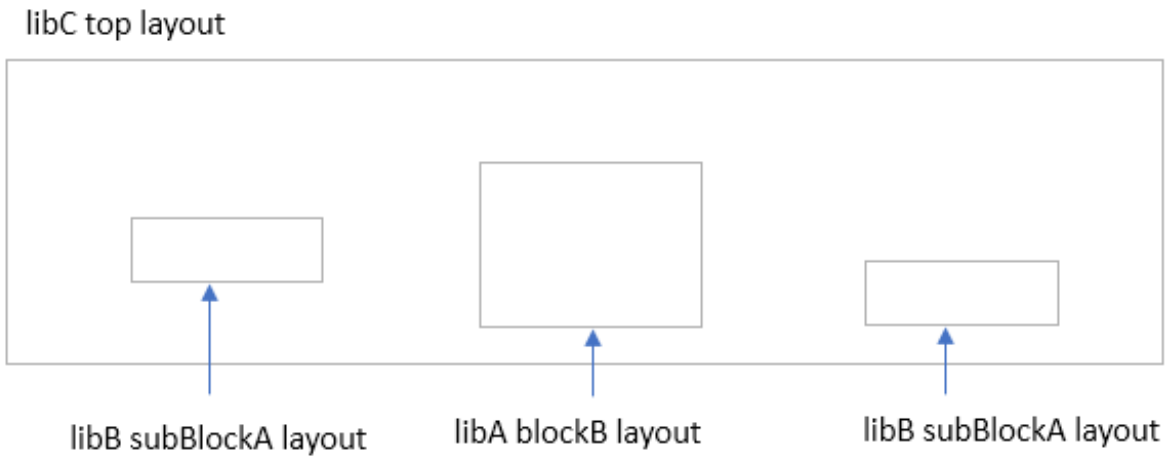
Consider a design, `{libC top layout}`, which contains the three blocks shown below:



Now suppose the block `{libA blockA layout}` contains a sub-block, `{libA subBlockA layout}`. This sub-block has the same cell name as the top-level block `{libB subBlockA layout}`. However, since the library name is different, the contents of both blocks are most likely different.



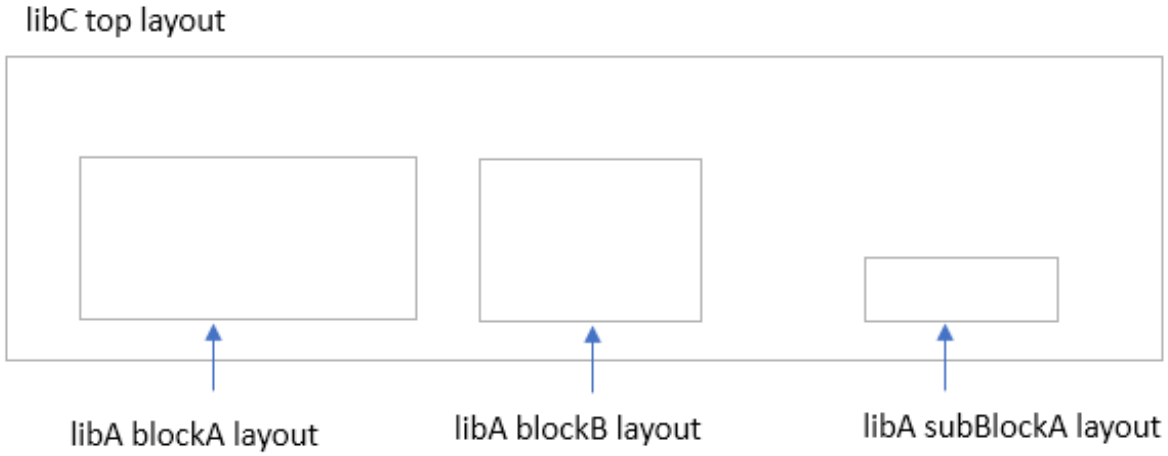
If `assembleDesign` is run to flatten blockA (`assembleDesign -block {libA blockA layout`), the resulting output will be as follows:



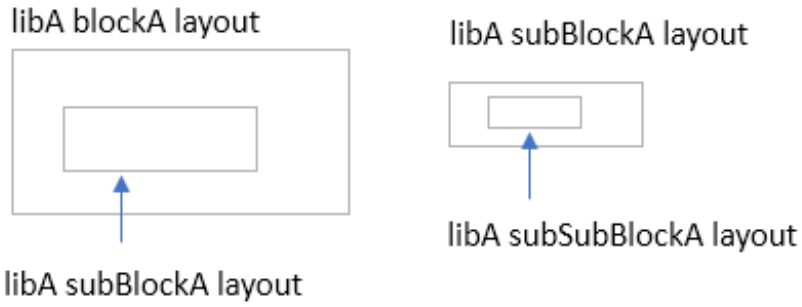
Notice that after the flattening, the cellview `{libB subBlockA layout}` appears in place of the cellview `{libA subBlockA layout}`. This is because the tool has already read in the cellview (`subBlockA`) earlier on during `init_design` and the library that it sees or binds to this cell is `libB`.

18.9.8.3 Handling of cells that appear at different hierarchy levels

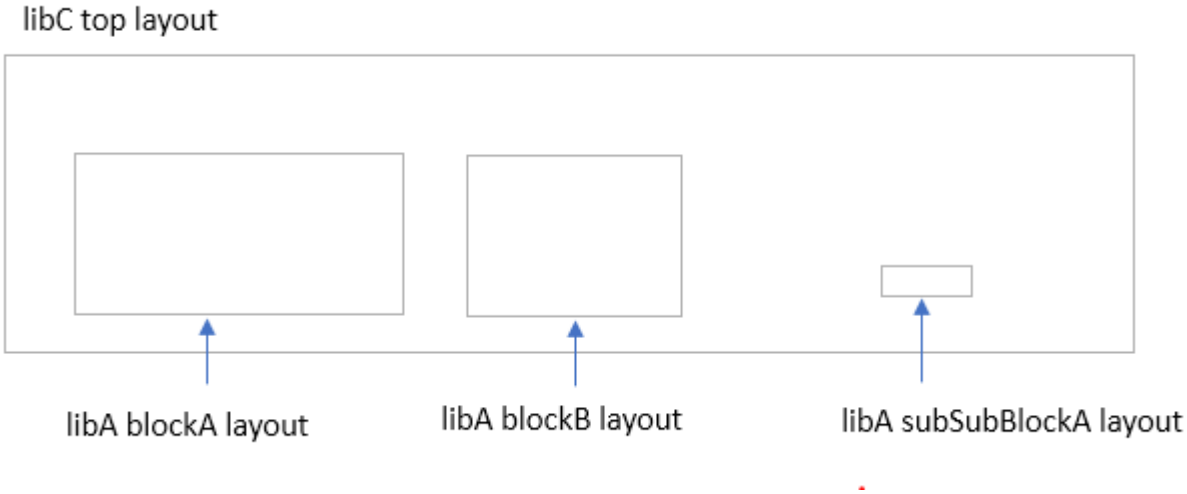
For cellviews that are being instantiated at a different hierarchy level of a design, the sequence in which the design is assembled with `assembleDesign` matters in ensuring that the flattening of the design is done successfully in Innovus. To explain the concept, consider a design that has the three blocks shown below:



The cellview `{libA blockA layout}` contains a cellview that is the same as the one at the top, `{libA subBlockA layout}`. The cellview `{libA subBlockA layout}` in turn contains a block `{libA subSubBlockA layout}`.



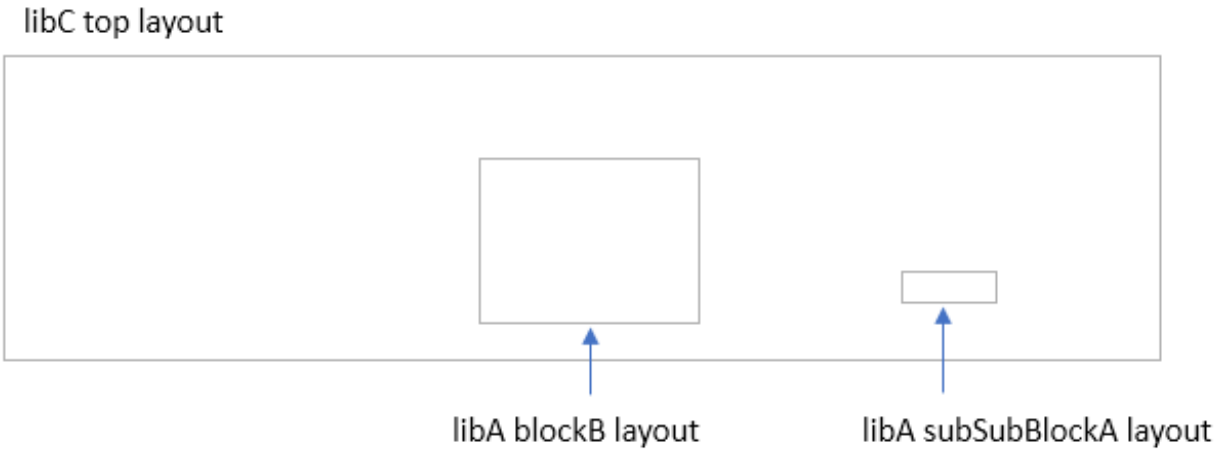
If the block `{libA subBlockA layout}` is flattened first (that is, the command `assembleDesign -block {libA subBlockA layout}` is run), the resulting output will be as follows. Notice that the cell `subSubBlockA` appears on the right.



However, if the following command is then run, there is nothing on the left side:

```
assembleDesign -block {libA blockA layout}
```

This is because after the first `assembleDesign` of `{libA subBlockA layout}`, a logical module of `subBlockA` is created. This means Innovus now interprets `subBlockA` as a logical module and not as a physical instance. This limitation comes from the Verilog netlist definition that Innovus used for its its logical netlist structure.

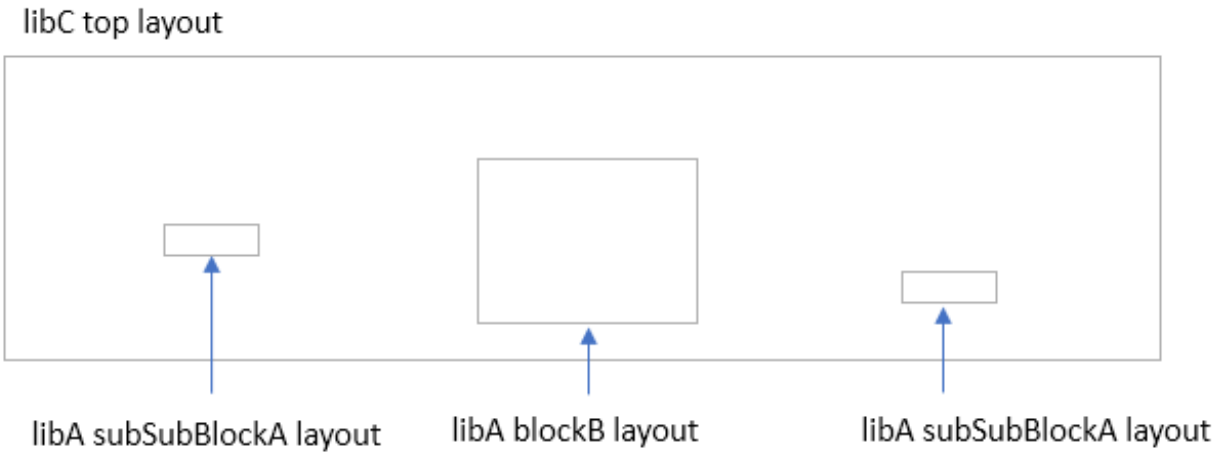


To flatten both blocks (blockA and subBlockA) correctly at the top and the child instance, the correct sequence of assembleDesign should be:

```
assembleDesign -block {libA blockA layout}

assembleDesign -block {libA subBlockA layout}
```

The resulting output is as follows:



18.10 Parasitic RC Extraction for Running MS-STA

Parasitic RC (resistance and capacitance) extraction of wiring interconnects is an important step in STA. In physical implementation, the logic instances are connected by the metal interconnects. The parasitic RC of the interconnect impacts the signal path delay. In a typical nanometer design, the parasitic RC of interconnect can account for the majority of the delay in the design.

18.10.1 Running Quantus Extraction with postRoute Engine for a Routed Design

Typically, for a design that is fully routed, the mode of extraction should be set to `postRoute` by specifying the following command statement:

```
setExtractRCMode -engine postRoute
```

You can use `setExtractRCMode -effortLevel` to further specify which extraction engine is to be used.

In the `postRoute` mode, `-effortLevel` can be specified as one of the following:

- `-effortLevel medium` to invoke the TQuantus extraction engine
- `-effortLevel high` to invoke the IQuantus extraction engine
- `-effortLevel signoff` to invoke the standalone Quantus extraction engine.

The TQuantus and IQuantus extraction engines are native to Innovus. To run the standalone Quantus extraction engine, the path for the Quantus binary has to be specified before invoking Innovus. All these engines require the Quantus techfile to be specified in the `viewDefinition.tcl` file.

18.10.2 Running Quantus Extraction with Signoff Effort Level

If the timing analysis is for signing off purpose, the `-effortLevel` option should be set to `signoff` to invoke the standalone Quantus extraction engine. The `signoff` effort level offers the highest level of accuracy. This is also the recommended option to handle a design that contains complex wire shapes.

The IQuantus extraction engine in Innovus is typically used for handling normal wires created by the router in the place and route environment and not for complex wires shapes that are created in Virtuoso. The following is a sample message if the design contains complex wire shapes and the effort level is not set to signoff.

```
**WARN: (IMPEXT-1452): Internal representation of wires of net 'por_h' at (231600
800740 3) is complex and may have overlapping geometries. So, IQuantus extraction is
not possible for this net. RC estimation will be used but any SPEF that is generated
subsequently will not include any RC's for this net. If this net falls in the critical
path, use the signoff extractor by setting the setExtractRCMode -effortLevel to signoff
for better accuracy.
```

Please note that if the design being analyzed contains macros that are represented as GDS because an accurate abstract is not available, the GDS representations of such macros would need

to be passed to Quantus when using the Quantus integration in Innovus to perform extraction. This is done using the following settings in Innovus:

```
setExtractRCMode -effortLevel signoff
setExtractRCMode -coupled true
setExtractRCMode -engine postRoute
setExtractRCMode -qrcCmdType partial
setExtractRCMode -qrcCmdFile macrogds.cmd
```

In the above example the `-qrcCmdType partial` indicates that the `macrogds.cmd` file needs to be merged with the regular Quantus command file. The user also needs to indicate the layermap file to be used by Quantus.

18.10.3 Using the Quantus Layer Map File

This layer map is required when running the RC extraction with the signoff effort level. The layer names in the Quantus technology file may be different from the layer names used by Innovus (e.g. the layer names used in the technology file). The Quantus layer map file provides layer mapping between the technology file of Innovus and the Quantus technology file.

The following shows an example of the layer map file:

```
extraction_setup \

    -technology_layer_map \

        Metal1              METAL_1 \
        Via1            VIA_1 \
        Metal2              METAL_2 \
        Via2            VIA_2 \
        Metal3              METAL_3 \
        Via3            VIA_3 \
        Metal4              METAL_4 \
        Via4            VIA_4 \
        Metal5              METAL_5 \
        Via5            VIA_5 \
        Metal6              METAL_6 \
        Via6            VIA_6 \
        Metal7              METAL_7 \
        Via7            VIA_7 \
        Metal8              METAL_8 \
        Via8            VIA_8 \
        Metal9              METAL_9 \
```

```
Via9            VIA_9 \
Metal10            METAL_10 \
Via10           VIA_10 \
Metal11            METAL_11
```

Note: When the IQuantus extraction engine is selected, you do not need to specify the Quantus layer map file because Innovus will auto-create it in this mode. However, in some complex cases that have non-metal routing layer, the automatically created map file may not be correct. In such a situation, you have to specify a layer map for it.

18.10.4 Auto-creation of the Quantus Layer Map File from Innovus

To get Innovus to create a Quantus layer map file to be used or edited for other purposes (e.g. for the signoff mode), select IQuantus as the extraction engine and set the effort level to high. Then, run RC extraction.

```
setExtractRCMode -engine postRoute -effortLevel high

extractRC
```

After RC extraction is complete, look for a directory named `extLogDir` in the current working directory. Then look for the latest sub-directory inside the `extLogDir` directory (if there are multiple sub-directories). The name of the sub-directory should start with IQuantus followed by the running date, time, and so on (e.g. IQuantus_06-Jul-2017_15:16:51_18410_Q6Gdq9).

In that sub-directory, look for a file named `extr.LP_pll.layermap.log`. This will contain the content of the layer map that was used for running IQuantus.

18.10.5 Running Signoff Quantus Extraction in the OpenAccess Mode

By default, the signoff Quantus extraction is based on a LEF/DEF flow. To run signoff Quantus extraction in the OpenAccess mode, specify:

```
setExtractRCMode -useQrcOAInterface true
```

18.10.6 Auto-creation of Input Files from Innovus To Run Standalone Quantus QRC

During the signoff mode, Innovus generates the design input, library inputs, technology inputs and the command file to Quantus. Then it will run RC extraction in the background and read back the resulting SPEF file into Innovus. By default, the above files are stored in a temporary directory and the directory will be deleted upon completion of the whole RC extraction process.

You can use the command, `write_extraction_spec` to generate the command file and a directory that stores other inputs files. Use the `-out_dir` option to specify the directory name that stores the input files.

For example, assume the following is specified:

```
setExtractRCMode -engine postRoute

setExtractRCMode -effortLevel signoff

setExtractRCMode -useQrcOAInterface true

setExtractRCMode -lefTechFileMap QRC.layermap

write_extraction_spec -out_dir standalone_qrc_dir
```

In this case, the Quantus command file (also called the CCL file), `qrc.cmd`, should be generated in the working directory. In the directory named `standalone_qrc_dir`, you should find the input files (for example, the tech file) for Quantus QRC.

18.10.7 Running Standalone Quantus QRC and Loading the Resulting SPEF Files Back to Innovus

To run Quantus QRC in the LEF/DEF mode with `qrc.def` as the design input, use:

```
qrc -cmd qrc.cmd qrc.def
```

To run Quantus QRC in the OpenAccess mode, run:

```
qrc -qrc.cmd
```

The OpenAccess-mode Quantus QRC run does not require an additional design input file because the specification of the library, cell, and view names of the cellview is present in the command file, `qrc.cmd`.

Note: If the OpenAccess design contains PCells, Quantus requires the PCell cache, generated using Virtuoso's *Express Pcell Manager*, to read in the PCells. The `CDS_ENABLE_EXP_PCELL` and `CDS_EXP_PCELL_DIR` environment variables are also consumed by Quantus for locating the PCell cache directory. For example, the following statements specify that the PCell cache is stored in the `./.expressPcells` directory.

```
setenv CDS_ENABLE_EXP_PCELL true

setenv CDS_EXP_PCELL_DIR ./.expressPcells
```

To load the resulting SPEF back into Innovus (assuming two SPEF files are created for two RC corners), the commands are:

```
spefIn -rc_corner rc_worst LP_pll_rc_worst.spef

spefIn -rc_corner rc_best LP_pll_rc_best.spef
```

For more information on Quantus QRC extraction, refer to the "RC Extraction" chapter of the *Innovus User Guide.*

Note: In certain cases, the designs coming from Virtuoso could contain macros for which a GDS file exists, but an accurate LEF abstract is not available. If you wish to do only timing analysis at the top level, you do not need to create a detailed abstract for the macros in the design that only have GDS information. To run standalone Quantus extraction and pass the GDS representation of the macros in the design to Quantus, the following syntax needs to be used in the Quantus command file.

```
input_db -type oa \

 -design_cell_name <cellname> <viewname> <Design library name> \
 -library_cell_name * abstract <reference library name> \
 -gds_file_list <files containing GDS description of Macro's>\
 -library_definitions_file <the name of the cds.lib ile where libraries are defined>
```

Note that if `-library_cell_name * abstract <reference library name> \` is not defined and `-gds_file_list` has been specified, Quantus will automatically use the layout view of the leaf-level cells, if available. In this case, if the layout views of the leaf-level cells have issues with pin, etc, Quantus may run into a problem and not extract certain nets.

18.11 Tips for Debugging the No Constrained Timing Path Issue Generated by report_timing

When you run `report_timing`, the timer may sometimes issue the following message instead of generating a timing report:

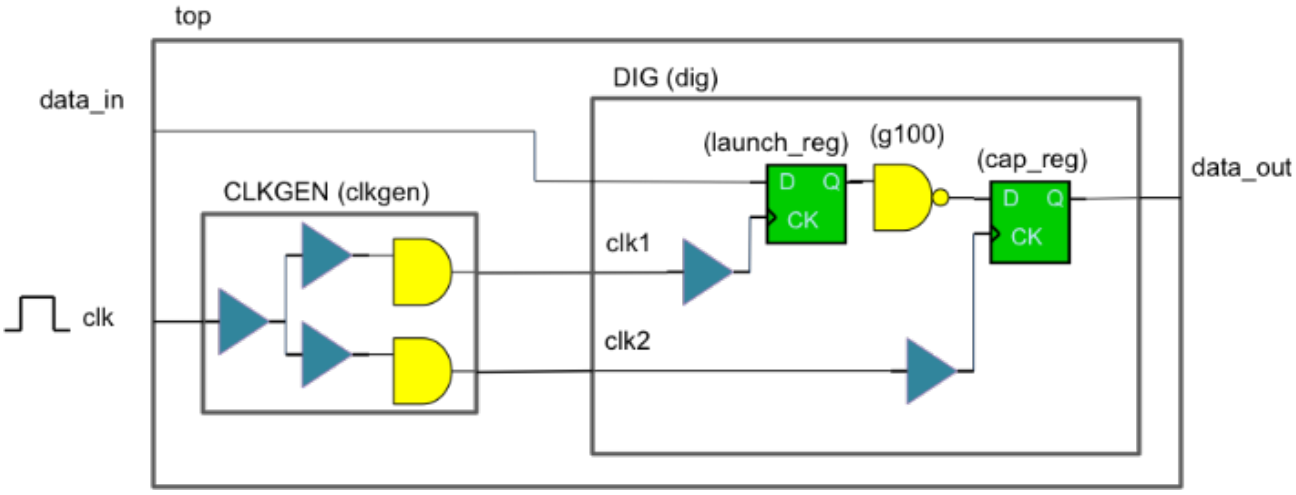
```
> report_timing -from wdac/a_reg/D

No constrained timing paths found.
Design may not be unconstrained or library is missing timing information.
```

This section will help you understand what this message means, the reasons why the timer issues such a message, and the ways to debug and fix the issue. However, before you do so, it is important to review the basic conditions that must be met for the timer to generate a timing report on a path. The following case study will help you understand these conditions.

18.11.1 Case Studies

The goal is to time a path between two registers (`dig/launch_reg` to `dig/cap_reg`) inside a digital block (`DIG`).



A timing constraint is specified at the primary input port (`clk`) of the top level design:

```
create_clock -name clk -period 10 [get_ports clk]
```

The clock signal has to propagate through a `CLKGEN` block before reaching the `DIG` block. Internally, the clock signal goes through some buffers and clock gating cells. As this is a hierarchical design, assuming the top-level design is already loaded, you need to run the `assembleDesign` command to flatten the `CLKGEN` and `DIG` blocks.

```
assembleDesign -block {designLib CLKGEN layout} -block {designLib DIG layout}
```

Assume that the two blocks are XL-compliant so that when they are flattened, there is logical connectivity for all instances on the clock and data paths of the two registers. With the timing library for all the standard cells loaded into the tool, the timer should see a valid path from the primary input port (`clk`) to both registers (`dig/launch_reg` to `dig/cap_reg`).

When the following command is run, a timing report such as the one given below is displayed:

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D
```

Path 1: MET Setup Check with Pin dig/cap_reg/CK

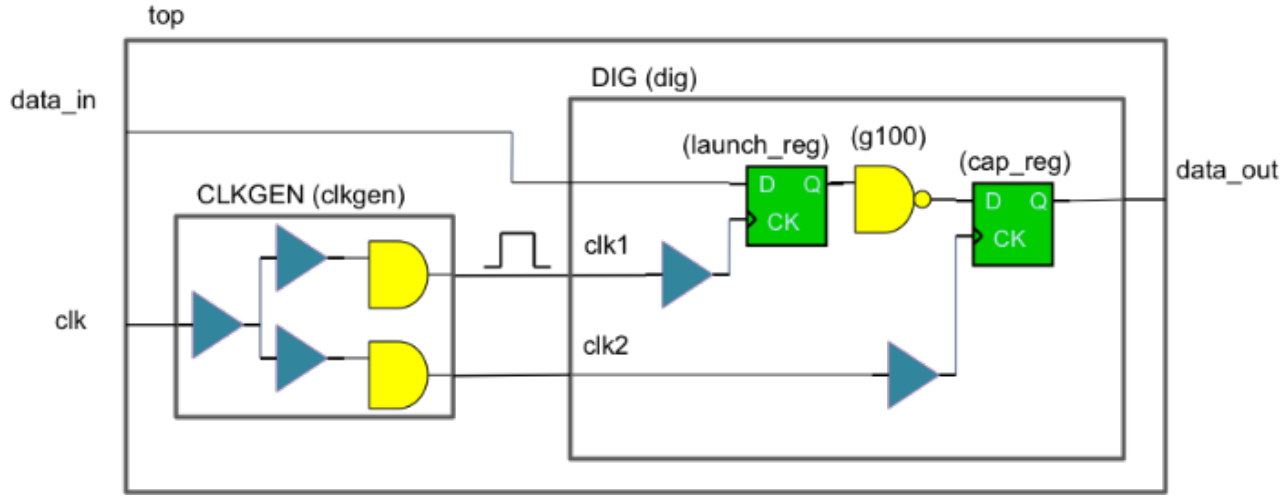
```
Endpoint:   dig/cap_reg/D    (v) checked with  leading edge of 'clk'
Beginpoint: dig/launch_reg/Q (^) triggered by  leading edge of 'clk'
Path Groups: {clk}
Analysis View: max
Other End Arrival Time          0.437
- Setup                         0.048
```

```
+ Phase Shift                  10.000
= Required Time                10.389
- Arrival Time                  0.818
= Slack Time                    9.571
     Clock Rise Edge                0.000
     + Clock Network Latency (Prop)  0.437
     = Beginpoint Arrival Time      0.437
     +-------------------------------------------------------------------+
     |    Instance    |     Arc     |  Cell   | Delay | Arrival | Required |
     |                |             |         |       |  Time   |   Time   |
     |----------------+-------------+---------+-------+---------+----------|
     | dig/launch_reg | CK ^        |         |       |   0.437 |   10.008 |
     | dig/launch_reg | CK ^ -> Q ^ | DFFRX1  | 0.315 |   0.752 |   10.322 |
     | dig/g100       | A ^ -> Y v  | NAND2X1 | 0.067 |   0.819 |   10.389 |
     | dig/cap_reg    | D v         | DFFRX1  | 0.000 |   0.818 |   10.389 |
     +-------------------------------------------------------------------+
```

18.11.1.1 Case 1 - the create_clock constraint is specified at the wrong point

What would be the result if the `create_clock` statement is specified at the pin level instead of the primary port level of the `DIG` block? For example:

```
create_clock -name clk -period 10 [get_pins dig/clk1]
```



When the same path is now timed, the timer reports no constrained timing path:

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D

No constrained timing paths found.
Design may not be unconstrained or library is missing timing information.
```

To debug the issue, run the command again with the additional option `-unconstrained` to check if the timer reports more information:

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D -unconstrained

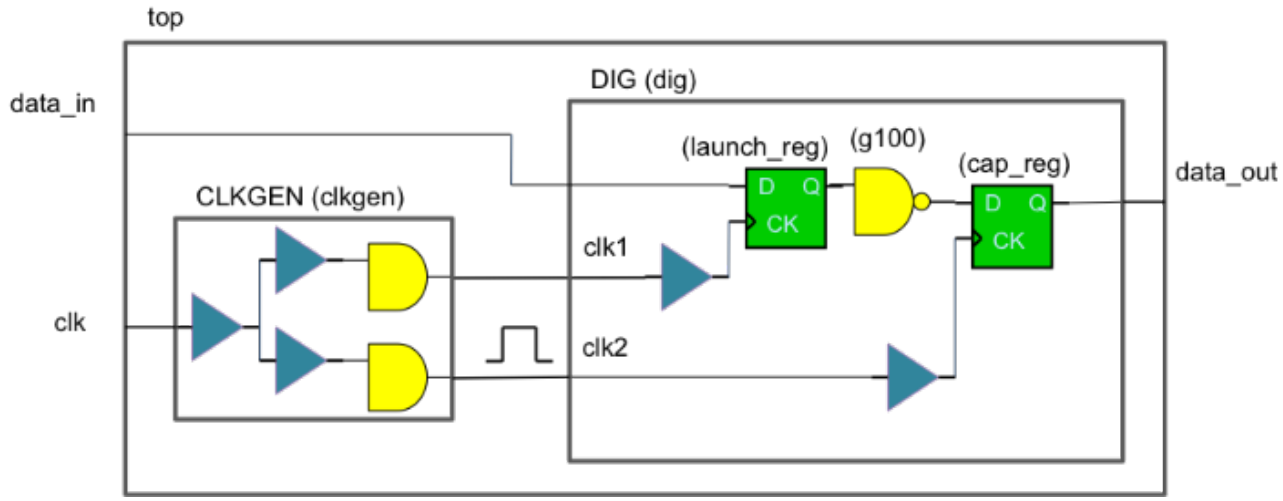
Path 1:Endpoint: dig/cap_reg/D (v) (unconstrained output)
 Beginpoint: dig/launch_reg/Q (^) triggered by leading edge of 'clk'
 Arrival Time                    0.450
 Analysis View: max
      + Clock Network Latency (Prop) 0.068
      = Beginpoint Arrival Time      0.068
      +-------------------------------------------------------+
      |    Instance    |     Arc     |   Cell   | Delay | Arrival |
      |                |             |          |       |   Time  |
      |----------------+-------------+----------+-------+---------|
      | dig/launch_reg | CK ^        |          |       |   0.068 |
      | dig/launch_reg | CK ^ -> Q ^ | DFFRX1   | 0.315 |   0.383 |
      | dig/g100       | A ^ -> Y v  | NAND2X1  | 0.067 |   0.450 |
      | dig/cap_reg    | D v         | DFFRX1   | 0.000 |   0.450 |
      +-------------------------------------------------------+
```

The *unconstrained output* shown in the report indicates that something is not right at the capturing end (or the capturing clock).

In fact, its clock path is not constrained. The capturing register is not receiving a clock signal.

Now, what if the `create_clock` statement is specified at the other clock input of the `DIG` block?

```
create_clock -name clk -period 10 [get_pins dig/clk2]
```



The timer reports no constrained timing path in this case as well.

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D

No constrained timing paths found.
Design may not be unconstrained or library is missing timing information.
```

To debug, run the command again with the `-unconstrained` option:

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D -unconstrained
```

This time, the timer reports that the beginning point is not constrained.

```
Path 1:Endpoint:  dig/cap_reg/D (v)
 Beginpoint: dig/launch_reg/Q (^) (unconstrained input)
 Arrival Time                    0.382
 Analysis View: max
      + Clock Network Latency (Ideal) 0.000
      = Beginpoint Arrival Time 0.000
      +-------------------------------------------------------------+
      |     Instance   |     Arc     |   Cell   | Delay | Arrival  |
      |                |             |          |       |   Time   |
      |----------------+-------------+----------+-------+----------|
      | dig/launch_reg | CK ^        |          |       |   0.000  |
      | dig/launch_reg | CK ^ -> Q ^ | DFFRX1   | 0.315 |   0.315  |
      | dig/g100       | A ^ -> Y v  | NAND2X1  | 0.067 |   0.382  |
      | dig/cap_reg    | D v         | DFFRX1   | 0.000 |   0.382  |
      +-------------------------------------------------------------+
```

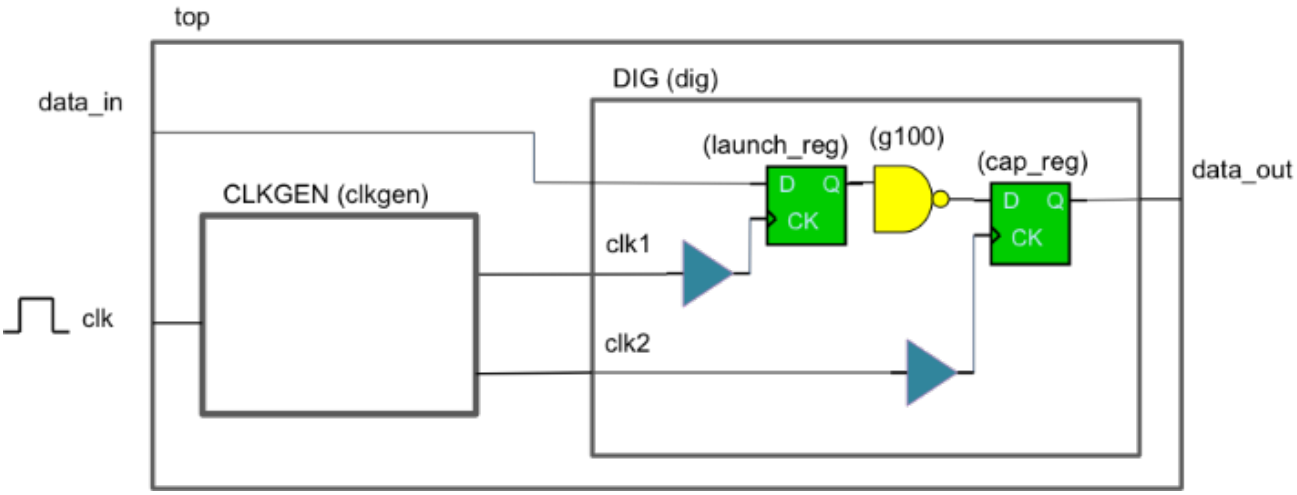
Both these examples highlight the case where the timing path is not fully constrained.

18.11.1.2 Case 2 - the CLKGEN block is not flattened

Consider another situation in which the `assembleDesign` command is run only on the `DIG` block.

```
> assembleDesign -block {designLib DIG layout}
```

In this case, the `CLKGEN` block is not flattened. As a result, the actual path becomes the following:



No constrained timing path will be reported if the same path is to be timed.

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D

No constrained timing paths found.
Design may not be unconstrained or library is missing timing information.
```

Run `report_timing` with the `-unconstrained` option:

```
report_timing -from dig/launch_reg/Q -to dig/cap_reg/D -unconstrained
```

It reports that the beginning point is not constrained properly.

```
Path 1:Endpoint: dig/cap_reg/D (v)
Beginpoint: dig/launch_reg/Q (^) (unconstrained input)
Arrival Time 0.381
Analysis View: max
     + Clock Network Latency (Ideal) 0.000
     = Beginpoint Arrival Time       0.000
     +-------------------------------------------------------------+
     |    Instance    |    Arc      |  Cell    | Delay | Arrival |
     |                |             |          |       |  Time   |
     |----------------+-------------+----------+-------+---------|
     | dig/launch_reg | CK ^        |          |       |   0.000 |
     | dig/launch_reg | CK ^ -> Q ^ | DFFRX1   | 0.313 |   0.313 |
     | dig/g100       | A ^ -> Y v  | NAND2X1  | 0.068 |   0.381 |
     | dig/cap_reg    | D v         | DFFRX1   | 0.000 |   0.381 |
     +-------------------------------------------------------------+
```

To further investigate whether the issue falls on the clock path, run the following command:

```
report_timing -unconstrained -clock_from clk
```

```
Path 1:Endpoint: clkgen/clk (v) (unconstrained output)
 Beginpoint: clk     (v) triggered by trailing edge of 'clk'
 Arrival Time               0.000
 Analysis View: max
      = Beginpoint Arrival Time 0.000
      +-----------------------------------------+
      | Instance |  Arc  |  Cell  | Delay | Arrival |
      |          |       |        |       |  Time   |
      |----------+-------+--------+-------+---------|
      |          | clk v |        |       |   0.000 |
      | clkgen   | clk v | CLKGEN | 0.000 |   0.000 |
      +-----------------------------------------+
```

The result shows that the clock propagation stops at the input of `CLKGEN` and is not able to reach the clock input of `launch_reg` and `cap_reg` of `DIG` block. In this case, the timing constraint is valid. However, the clock is not able to propagate through the `CLKGEN` block unless a timing library for the `CLKGEN` block is provided. If the `CLKGEN` block is flattened so that those standard cells (which have timing library loaded to the tool) on the clock path are brought to the top for timing analysis, then the path will become valid.

Use the `report_instance_library` command to check if an instance is bound with timing library.

```
> report_instance_library -instance clkgen

Instance      : clkgen
Analysis View : max
Power Domain  : -
Op Cond       : P->1.000000, V->0.900000, T->125.000000 (slow_oc)
```

The output shows that the `clkgen` instance is not bound with any timing library.

If a cell is bound with timing library, the name of the library file that is bound to it is displayed in the output. For example, if you run the same command for `dig/launch_reg`, the output is as follows:

```
> report_instance_library -instance dig/launch_reg

Instance              : dig/launch_reg
Analysis View         : max
Power Domain          : -
Library/Libset(early) : slow/slow_libset
Library File (early)  : gsclib045/timing/slow_vdd1v0.lib
```

```
Library/Libset(late)   : slow/slow_libset
Library File (late)    : gsclib045/timing/slow_vdd1v0.lib
Op Cond                : P->1.000000, V->0.900000, T->125.000000 (slow_oc)
Cell Type              : gateCell
```

Without the timing library, the timer is not able to understand the timing relationship between the input and output pins of the `CLKGEN` block. This example highlights one common reason why a path is viewed as an invalid or broken timing path by the timer. As a result, the timer is unable to time it.

18.11.2 Basic Conditions for Reporting the Timing on a Path

As seen in the case study, the timer can report the timing on a path only if the path is constrained with timing constraint. In other words, the following basic conditions must be met for the timer to report the timing on a path:

1. The timing constraint for the path to be timed exists and is valid.
2. The path covered by the constraint exists and is not broken.

So, what does the following message actually mean?

```
No constrained timing paths found.
Design may not be unconstrained or library is missing timing information.
```

'No constrained timing paths' can be interpreted as either 'No constraint' or 'No timing paths'.

- 'No constraint' means that there is no valid timing constraint for the paths of concern.
- 'No timing paths' means that the paths are broken or invalid.

18.11.3 Common No Constraint Situations

Here are some common situations where the timer either cannot find a timing constraint or the timing constraint found is not valid.

18.11.3.1 No timing constraint loaded

In the `assembleDesign` flow, make sure that the timing constraint file is specified using `create_constraint_mode` in `viewDefinition.tcl` . The example below shows how a timing constraint file, `func.sdc` , is specified.

```
create_constraint_mode -name func -sdc_files [list func.sdc]
```

In the FTM flow, make sure that the timing constraint file is specified using the -

`ilm_sdc_files` option, whether it is for the `create_constraint_mode` statement specified in `viewDefinition.tcl`, or for `update_constraint_mode`, which is a TCL command run after `init_design`. For example, if the timing constraint file name is `func_flat.sdc`, the statement or command should be:

```
create_constraint_mode -name func -sdc_files [list func_top.sdc] -ilm_sdc_files
{func_flat.sdc}

update_constraint_mode -name func -ilm_sdc_files {func_flat.sdc}
```

Refer to the "Load the top-level design" of the Running STA by Using the FTM section for more details on the difference between top-design-only timing constraint (i.e. `func_top.sdc`) and flat timing constraint (i.e. `func_flat.sdc`). The flat type (here, `func_flat.sdc`) should be used for the FTM flow.

In both flows, the specified constraint mode should be tied to an analysis view and the analysis view should be specified using the `set_analysis_view` command. Basically, each analysis view contains a constraint mode and a delay corner. For example, to include the `func` constraint mode in analysis views and specify the associated analysis views for both setup and hold analysis, use the following commands:

```
create_analysis_view -name func_worst -constraint_mode func -delay_corner dc_worst

create_analysis_view -name func_best -constraint_mode func -delay_corner dc_best

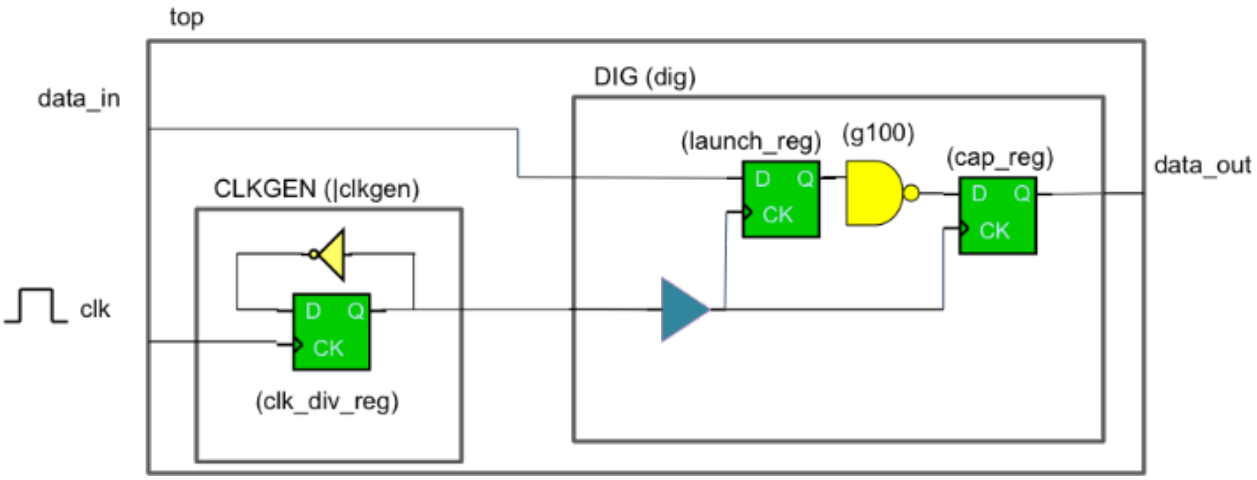
set_analysis_view -setup [list func_worst] -hold [func_worst func_best]
```

18.11.3.2 No appropriate timing constraint for each type of path

Depending on the type of the path to be timed, appropriate timing constraint statement has to be specified in the timing constraint file.

Register-to-register path

At least one `create_clock` constraint is needed to time a register-to-register path. Typically, when the clock path involves a clock divider or clock multiplier, the `create_generated_clock` constraint is required.



For the above diagram, the clocks can be specified as follows:

```
create_clock -name clk -period 10 [get_ports clk]

create_generated_clock -name clk_div -source [get_ports clk] -divide_by 2 [get_pins clkgen/clk_div_reg/Q]
```

Without the `create_generated_clock` constraint, the clock propagation stops at the `CK` pin of `clk_div_reg` and will not propagate through the `Q` pin of `clk_div_reg`.

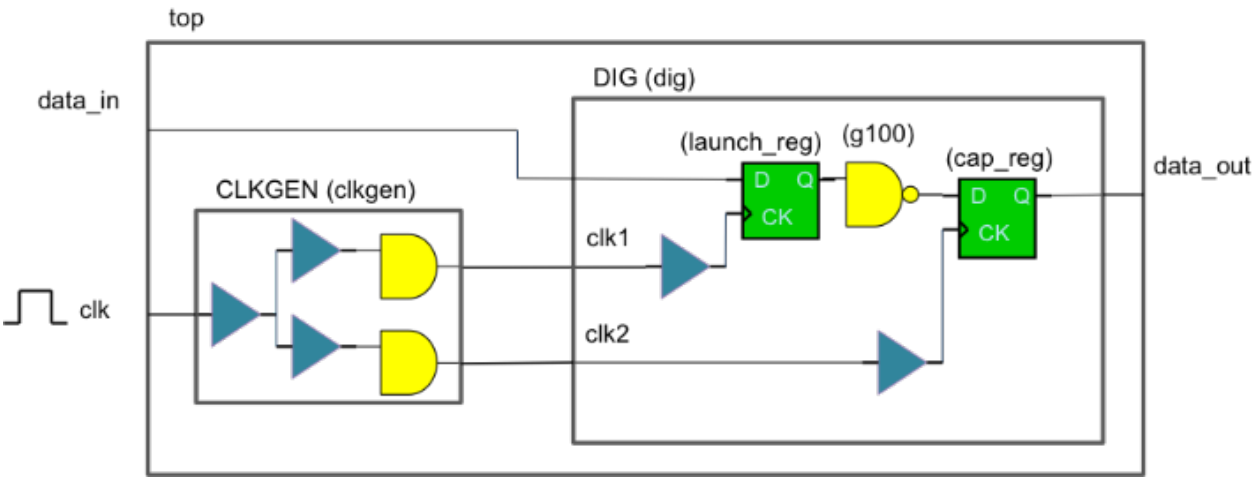
Input-to-register path

The data arrival times should be specified at the specified input ports, relative to the clock specified by the `-clock` option using the `set_input_delay` constraint.

For example:

```
create_clock -name clk -period 10 [get_ports clk]

set_input_delay 2.43 -clock [get_clocks clk] [get_ports data_in]
```



Note that, if `set_input_delay` is not specified, the tool is still able to report the timing, with an exception that the beginning point is not triggered by a clock (therefore, it shows @). One example is shown below:

```
> report_timing -to dig/launch_reg/D -from data_in

Path 1: MET Setup Check with Pin dig/launch_reg/CK

Endpoint: dig/launch_reg/D   (^) checked with leading edge of 'clk'
Beginpoint: data_in          (^) triggered by leading edge of '@'
Path Groups: {clk}
Analysis View: max
Other End Arrival Time          0.437
- Setup                         0.100
+ Phase Shift                  10.000
= Required Time                10.337
- Arrival Time                  0.000
= Slack Time                   10.337
     Clock Rise Edge 0.000      + Input Delay 0.000      = Beginpoint Arrival Time
0.000

         +-------------------------------------------------------------------+
        | Instance       | Arc       | Cell   | Delay | Arrival | Required |
        |                |           |        |       | Time    | Time     |
        |----------------+-----------+--------+-------+---------+----------|
        |                | data_in ^ |        |       | 0.000   | 10.337   |
        | dig/launch_reg | D ^       | DFFRX1 | 0.000 | 0.000   | 10.337   |
        +-------------------------------------------------------------------+
```

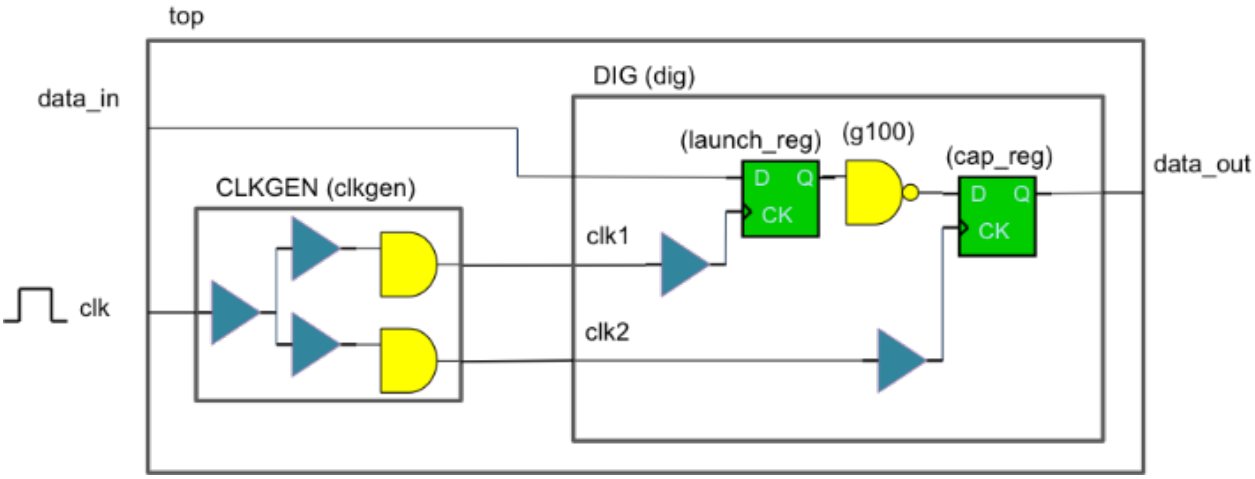
Register-to-output path

The data required times should be specified at the specified output ports, relative to the clock specified by the `-clock` option using the `set_output_delay` constraint. For example:

```
create_clock -name clk -period 10 [get_ports clk]

set_output_delay 1.23 -clock [get_clocks clk] [get_ports data_out]
```

If `set_output_delay` is not specified, the timer reports no constrained timing paths found. An example is shown below:




```
> report_timing -from dig/cap_reg/CK -to data_out

No constrained timing paths found.
Design may not be unconstrained or library is missing timing information.
```

To debug the issue, specify the `-unconstrained` option and run the command again.

```
> report_timing -from dig/cap_reg/CK -to data_out -unconstrained

Path 1:Endpoint: data_out    (^) (unconstrained output)
Beginpoint: dig/cap_reg/Q    (^) triggered by leading edge of 'clk'
Arrival Time                      0.744
Analysis View: max

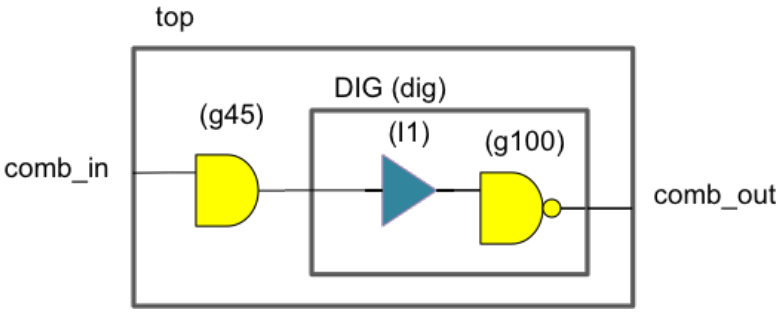
        + Clock Network Latency (Prop) 0.437

        = Beginpoint Arrival Time 0.437
        +-------------------------------------------------------+
        | Instance    | Arc         | Cell     | Delay | Arrival |
        |             |             |          |       | Time    |
        |-------------+-------------+----------+-------+---------|
```

```
| dig/cap_reg | CK ^        |          |        | 0.437  |
| dig/cap_reg | CK ^ -> Q ^ | DFFRX1   | 0.307 | 0.744   |
|             | data_out ^  |          | 0.000 | 0.744   |
+-------------------------------------------------------+
```

Combination path

A combination path involves no actual clock path. One common example of timing a combination path is to time a path from an input port to an output port that has no register in between.



For this type of path, you can specify the `set_input_delay` and `set_output_delay` constraints with a virtual clock. A virtual clock is a clock that is not associated with any part of the design.

```
create_clock -name v_clk -period 10
set_input_delay 4 -clock [get_clocks v_clk] [get_ports comb_in]
set_output_delay 4 -clock [get_clocks v_clk] [get_ports comb_out]
```

Alternatively, you may choose to specify `set_max_delay` to constrain such a type of path for setup time analysis and `set_min_delay` for hold time analysis.

```
set_max_delay 2 -from [get_ports comb_in] -to [get_ports comb_out]
set_min_delay 1 -from [get_ports comb_in] -to [get_ports comb_out]
```

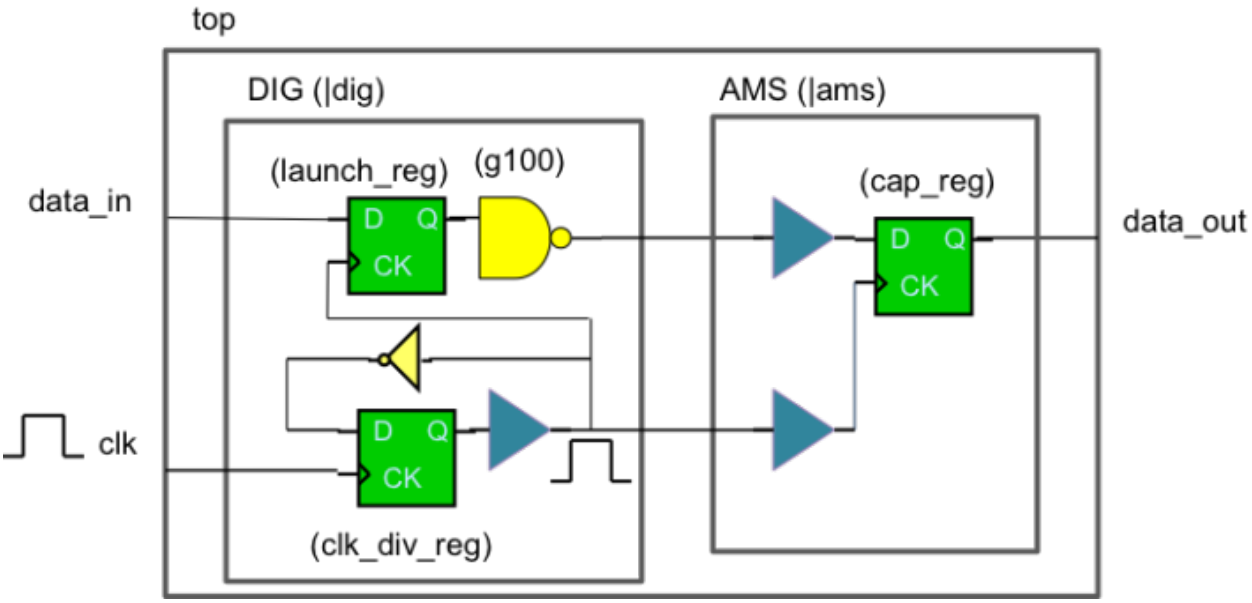
18.11.3.3 Design object specified in timing constraint does not match with the actual layout

A few examples of design objects are instances, primary ports, or pins of an instance. If the timer is not able to recognize a design object referenced in a timing constraint statement, that constraint statement is rejected. Some common situations are explained here:

Pipeline character in instance name of AOT design

If the top-level design is implemented in Virtuoso, the pipeline (|) character may be part of the instance name for blocks instantiated in the top-level design. In the mixed-signal OpenAccess (MSOA) flow, Innovus reads the design object from the physical layout cellview and not from the schematic cellview. If the timing constraint file is originally created based on the schematic, it may have to be modified to add | characters to the instance name to match with the corresponding name in the physical layout.

In the example below, the `DIG` block has been implemented in Innovus while the `AMS` block is a custom digital bock implemented by Virtuoso. Both blocks are integrated in Virtuoso, making it a schematic-on-top or Analog-on-Top (AOT) design.



Assume the original timing constraints that are based on the schematic diagram are as follows:

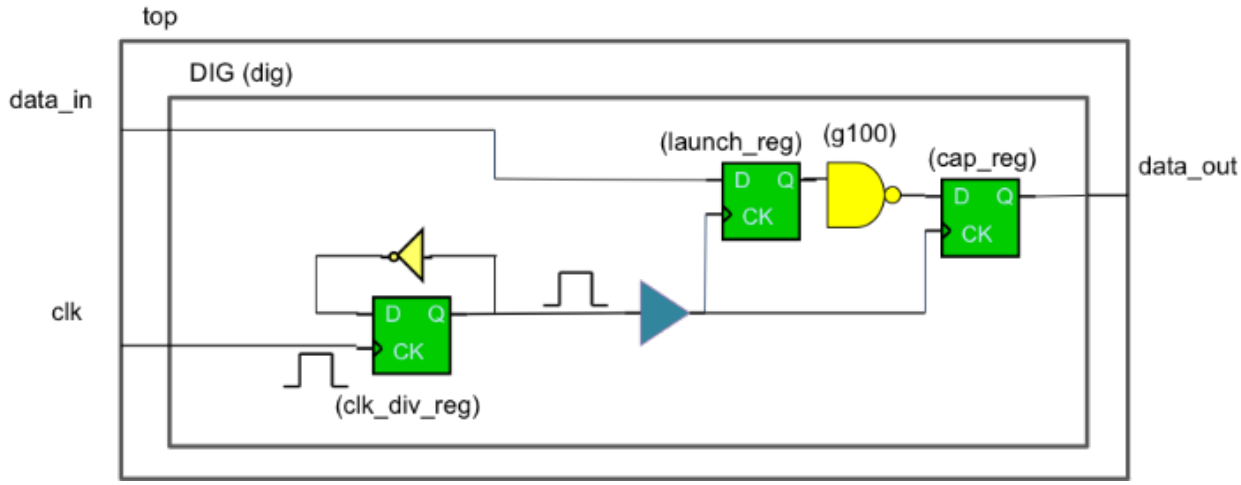
```
create_clock -name clk -period 10 [get_pins dig/clk_div_reg/CK]
create_generated_clock -name clk_div -source [get_pins dig/clk_div_reg/CK] -divide_by 2
[get_pins dig/clk_div_reg/Q]
```

However, to time this OpenAccess design in Innovus (or Tempus), the timing constraint should be as follows:

```
create_clock -name clk -period 10 [get_pins |dig/clk_div_reg/CK]
create_generated_clock -name clk_div -source [get_pins |dig/clk_div_reg/CK] -divide_by
2 [get_pins |dig/clk_div_reg/Q]
```

Timing constraint at the wrong design level

Another reason why the design object does not match could be that the timing constraint is meant for a different design level.



For example, for this schematic diagram, assume a timing constraint file with the following content is received from a designer:

```
create_clock -name clk -period 10 [get_pins clk_div_reg/CK]
create_generated_clock -name clk_div -source [get_pins clk_div_reg/CK] -divide_by 2
[get_pins clk_div_reg/Q]
```

By examining the content, one can easily see that the constraint is specified at the `DIG (dig)` level and not at the top level. Probably, the designer is the designer of the DIG block.

Whether `assembleDesign` is run on the `DIG` block (`assembleDesign -block {designLib DIG layout}`) or the FTM for the `DIG` block is created and specified (`specifyIlm -cell DIG -dir dig_FTM`), the required timing constraints should have an additional hierarchy (`dig`) for the specified instance name.

```
create_clock -name clk -period 10 [get_pins dig/clk_div_reg/CK]
```

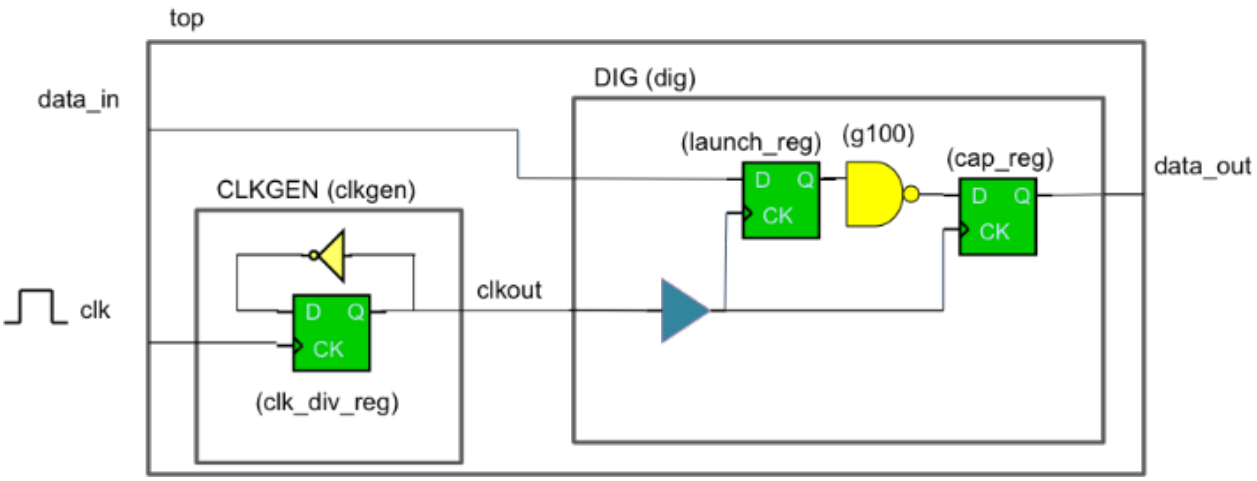
```
create_generated_clock -name clk_div -source [get_pins dig/clk_div-reg/CK] -divide_by 2
[get_pins dig/clk_div_reg/Q]
```

Block not flattened or FTM not specified

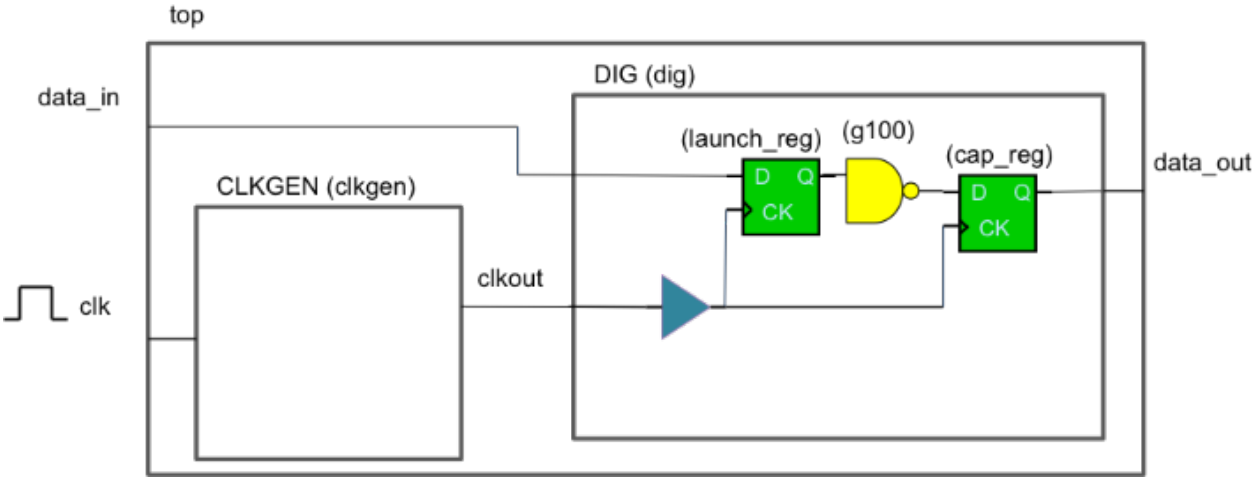
Consider the case below:

```
create_clock -name clk -period 10 [get_pins clkgen/clk_div_reg/CK]
```

```
create_generated_clock -name clk_div -source [get_pins clkgen/clk_div_reg/CK] -
divide_by 2 [get_pins clkgen/clk_div_reg/Q]
```



Here, the timing constraints are correct. However, if the `CLKGEN` block is not flattened using `assembleDesign` or an FTM is not created and specified for it, the diagram look likes the following and the timer does not accept these timing constraint statements. This is because the timer cannot find any instance in the physical layout cellview with name equal to `clkgen/clk_div_reg` .



The workaround is to either get a timing library for the `CLKGEN` block (which takes more effort and time) or modify the timing constraint as follows:

```
create_clock -name clk -period 20 [get_pins clkgen/clkout]
```

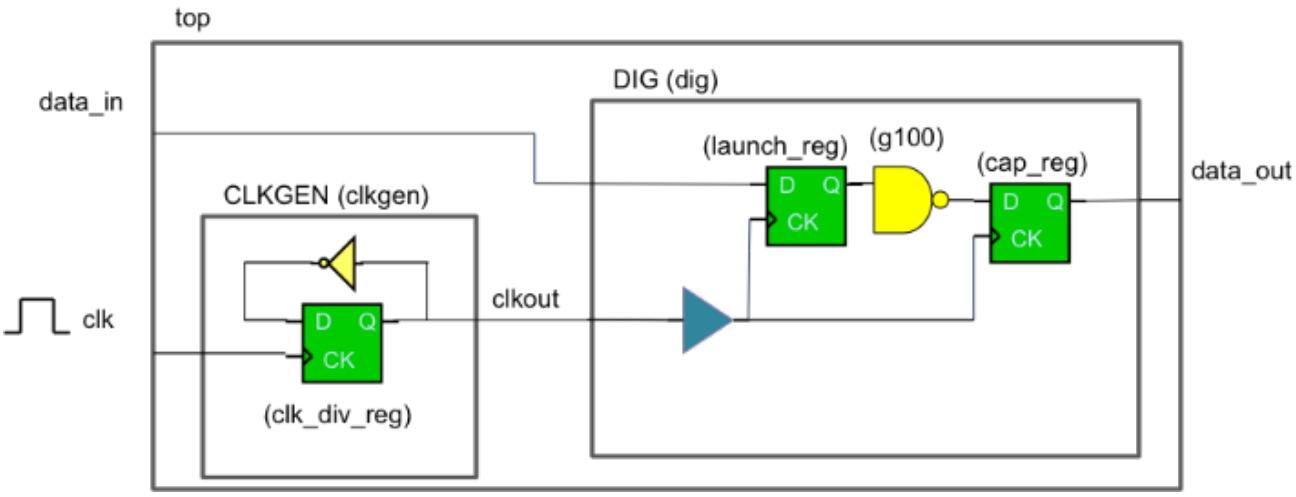
18.11.4 Checking the Validity of a Timing Constraint

How can you check whether a timing constraint is valid or, in other words, acceptable to the timer?

You can look at the log file and search for the timing constraint file name. When the timer parses a timing constraint file and detects something wrong with the constraints, it will issue warnings or

errors in the log file.

Let us use the previous case again to illustrate how you can check the validity of the timing constraints.



The timing constraints are:

```
create_clock -name clk -period 10 [get_ports clk]

create_generated_clock -name clk_div -source [get_ports clk] -divide_by 2 [get_pins clkgen/clk_div_reg/Q]
```

Assume these constraints are specified in a file called `simple.sdc`, and this timing constraint file is specified in `viewDefinition.tcl`.

18.11.4.1 Successful case for the assembleDesign flow

Assume the `assembleDesign` flow is run for the above case and the script is as follows:

```
set_db init_design_netlisttype {OA}

set_db init_oa_design_lib {designLib}

set_db init_oa_design_cell {top}

set_db init_oa_design_view {layout}

set_db init_oa_ref_lib {gsclib045}

set_db init_mmmc_file {viewDefinition.tcl}

init_design

setExtractRCMode -engine postRoute
```

```
setExtractRCMode -effortLevel high

assembleDesign -block {designLib DIG layout} -block {designLib CLKGEN layout}

source viewDefinition.tcl
```

Now, when you search for `simple.sdc` in the log file, you should see something like the following:

```
CTE reading timing constraint file 'simple.sdc' ...
```

Note - Ignore the message that is issued during `init_design`. At this point in time, errors related to timing constraints are expected because the `CLKGEN` block is not yet flattened. Therefore, the tool cannot find objects such as `clkgen/clk_div_reg/Q` in the layout.

To check whether the timing constraints are accepted by the timer, you can run `source viewDefinition.tcl` after `assembleDesign` is done.

If the constraints are read successfully, it should show something like the following:

```
CTE reading timing constraint file 'simple.sdc' ...
Current (total cpu=0:00:19.5, real=0:01:16, peak res=325.1M, current mem=660.9M)
INFO (CTE): Constraints read successfully.
```

18.11.4.2 Unsuccessful case for the assembleDesign flow

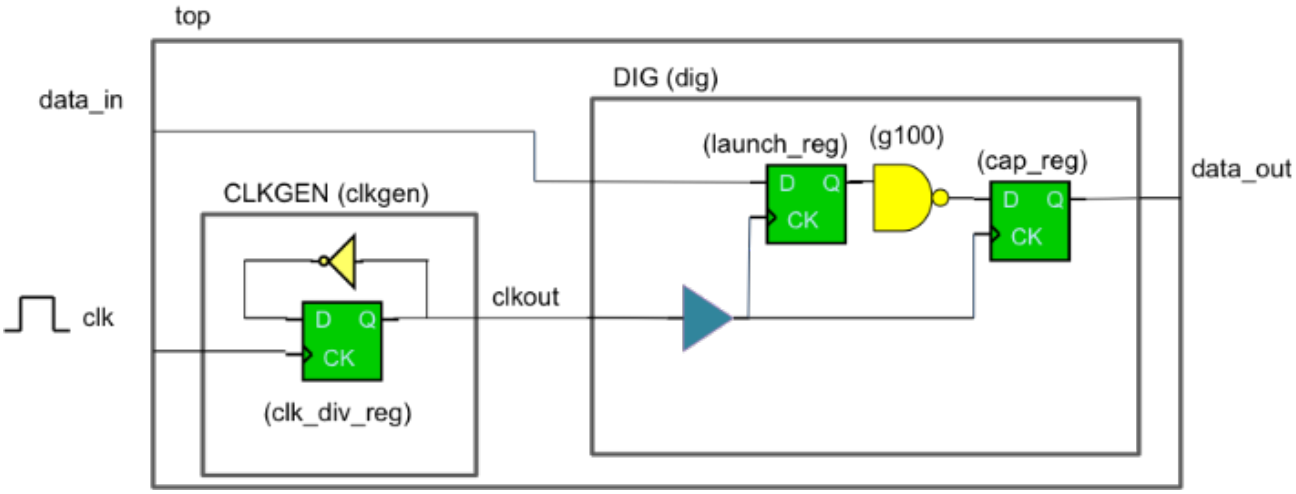
Now, suppose the `assembleDesign` command is run as follows:

```
assembleDesign -block {designLib DIG layout}
source viewDefinition.tcl
```

As the CLKGEN block is not flattened, the timer reports errors on the timing constraints when the `viewDefinition.tcl` is sourced again after `assembleDesign`.

```
CTE reading timing constraint file 'simple.sdc' ...
Current (total cpu=0:00:08.5, real=0:00:15.0, peak res=201.2M, current mem=544.5M)
**WARN: (TCLCMD-513): The software could not find a matching object of the specified
type for the pattern 'clkgen/clk_div_reg/Q' (File simple.sdc, Line 2).
**ERROR: (TCLCMD-917): Cannot find 'pins' that match 'clkgen/clk_div_reg/Q' (File
simple.sdc, Line 2).
**ERROR: (TCLCMD-917): Cannot find 'ports or pins' that match '' (File simple.sdc, Line
2).
INFO (CTE): read_dc_script finished with 1 WARNING and 2 ERROR.
```

18.11.4.3 Successful case for the FTM flow



Assume the FTM flow is run for the same case and the script is as follows:

```
set_db init_design_netlisttype {OA}

set_db init_oa_design_lib {designLib}

set_db init_oa_design_cell {top}

set_db init_oa_design_view {layout}

set_db init_oa_ref_lib {gsclib045}

set_db init_mmmc_file {viewDefinition.tcl}

init_design

setExtractRCMode -engine postRoute

setExtractRCMode -effortLevel high

specifyIlm -cell DIG -dir dig_FTM

specifyIlm -cell CLKGEN -dir clkgen_FTM

flattenIlm

update_constraint_mode -name func -ilm_sdc_files {simple.sdc}
```

Assume the timing constraint mode specified in `viewDefinition.tcl` is created as follows:

```
create_constraint_mode -name func -sdc_files [list simple.sdc]
```

When looking at the log file,ignore the errors and warning reported by the timer during `init_design`. At this point of time, the tool is not able to see any internal content inside the `CLKGEN` block and,

therefore, is expected to report errors. Look at the lines after `create_constraint_mode` and the constraints should have been read successfully.

```
<CMD> update_constraint_mode -name func -ilm_sdc_files {simple.sdc}
CTE reading timing constraint file 'simple.sdc' ...
Current (total cpu=0:00:13.2, real=0:00:17.0, peak res=316.0M, current mem=636.9M)
INFO (CTE): Constraints read successfully.
```

Note that the command `update_constraint_mode` is not necessary if the `-ilm_sdc_files` option is already specified in the timing constraint mode in `viewDefinition.tcl` as follows:

```
create_constraint_mode -name func -sdc_files [list simple.sdc] -ilm_sdc_files
{simple.sdc}
```

Then, check whether constraints are read successfully during the command `flattenIlm`.

```
<CMD> flattenIlm
Reading one ILM netlist ...
...
CTE reading timing constraint file 'simple.sdc' ...
Current (total cpu=0:00:11.9, real=0:00:17.0, peak res=305.0M, current mem=632.3M)
INFO (CTE): Constraints read successfully.
```

18.11.4.4 Unsuccessful case for FTM flow

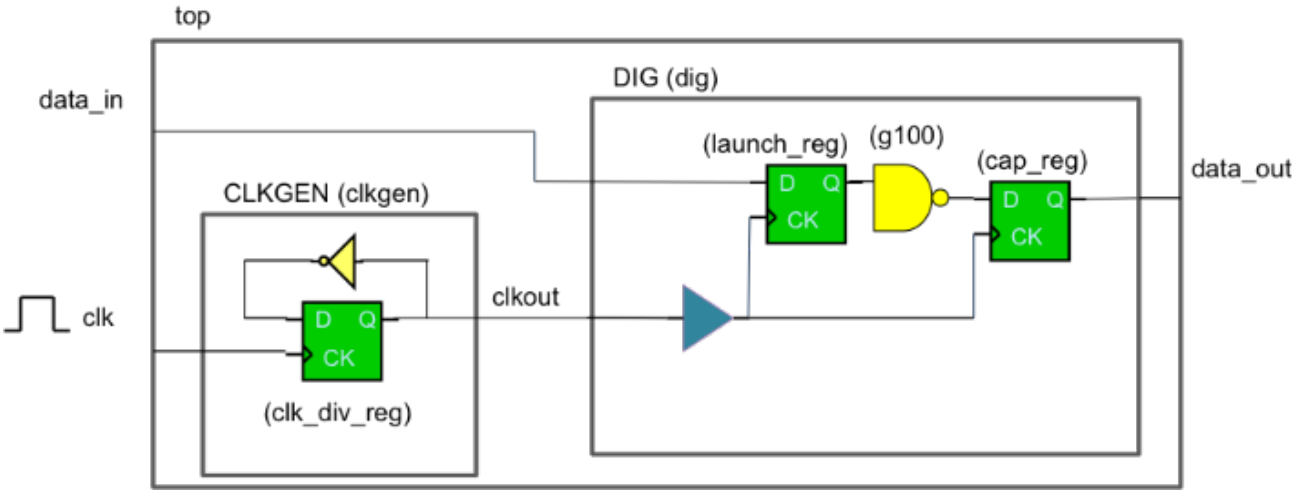
If you have specified the FTM only for the `DIG` block, the timer will report errors on the timing constraint when the `flattenIlm` or `update_constraint_mode` commands are run.

```
specifyIlm -cell DIG -dir dig_FTM
#specifyILM -cell CLKGEN -dir clkgen_FTM
flattenIlm

CTE reading timing constraint file 'simple.sdc' ...
Current (total cpu=0:00:08.5, real=0:00:15.0, peak res=201.2M, current mem=544.5M)
**WARN: (TCLCMD-513): The software could not find a matching object of the specified
type for the pattern 'clkgen/clk_div_reg/Q' (File simple.sdc, Line 2).
**ERROR: (TCLCMD-917): Cannot find 'pins' that match 'clkgen/clk_div_reg/Q' (File
simple.sdc, Line 2).
**ERROR: (TCLCMD-917): Cannot find 'ports or pins' that match '' (File simple.sdc, Line
2).
INFO (CTE): read_dc_script finished with 1 WARNING and 2 ERROR.
```

18.11.5 Checking the Existence of a Design Object in the Layout

When a timing constraint is not accepted by the timer because a design object is not found, how can you verify that the specified design object exists in the design layout? Or if there is a typo in the object name specified, how can you find the right name of the object?



In the above example, the tool returns the following message when reading the timing constraint:

```
**WARN: (TCLCMD-513): The software could not find a matching object of the specified
type for the pattern 'clkgen/clk_div_reg/Q' (File simple.sdc, Line 2).
```

To check if the object `clkgen/clk_div_reg` exists in the design, open the Design Browser from the *Tools* menu in Innovus. You can use the `*` character as a wild card.

Note - The Design Browser verification has to be done after the blocks are flattened. A block that has not been flattened is viewed as a blackbox logically by the tool.

In the following screenshot, `*clkgen/*reg*` is entered in the Design Browser and it shows up all instances that match with the input.



After checking the Design Browser and examining the `simple.sdc` file, you can see that the design object is not specified correctly in the following constraint:

```
create_generated_clock -name clk_div -source [get_ports clk] -divide_by 2 [get_pins clkgen/clk_div_reg/Q]
```

It should be:

```
create_generated_clock -name clk_div -source [get_ports clk] -divide_by 2 [get_pins |clkgen/clk_div_reg/Q]
```

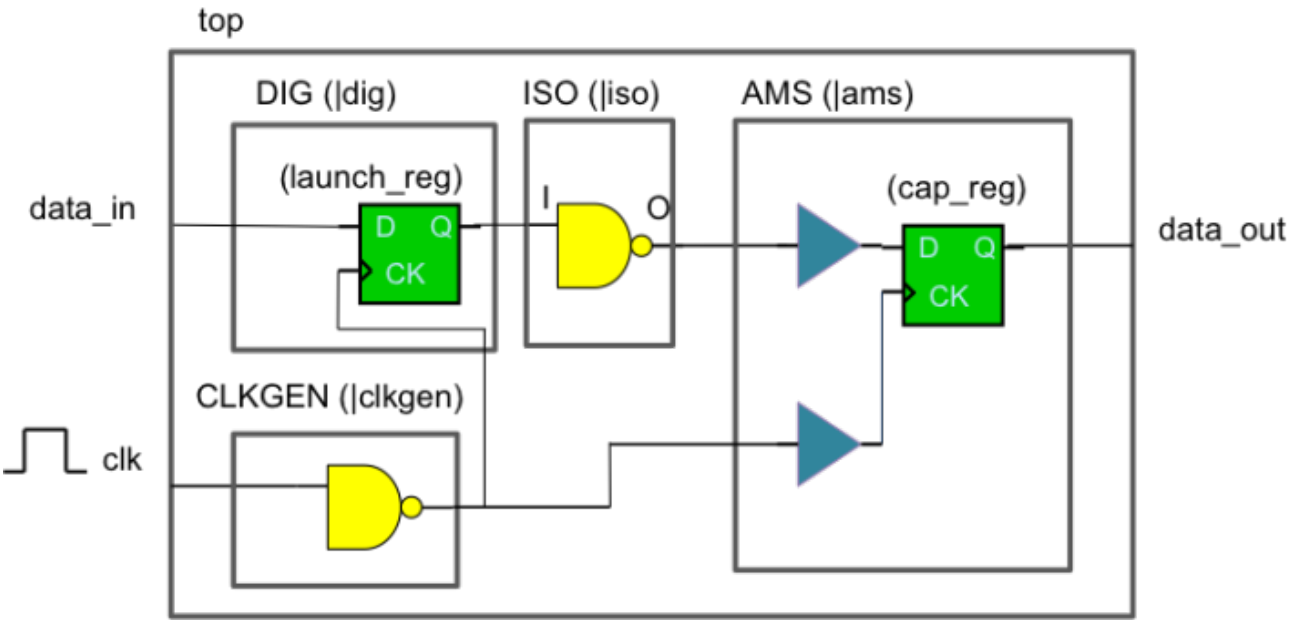
18.11.6 Common Invalid Timing Path Situations

Here are some common situations that could cause the timing path to be invalid. In each case, either the timing library is not present or the signal propagation is blocked.

18.11.6.1 Block not flattened or FTM not read

The diagram below shows a case where the data path is broken. The timing constraint is:

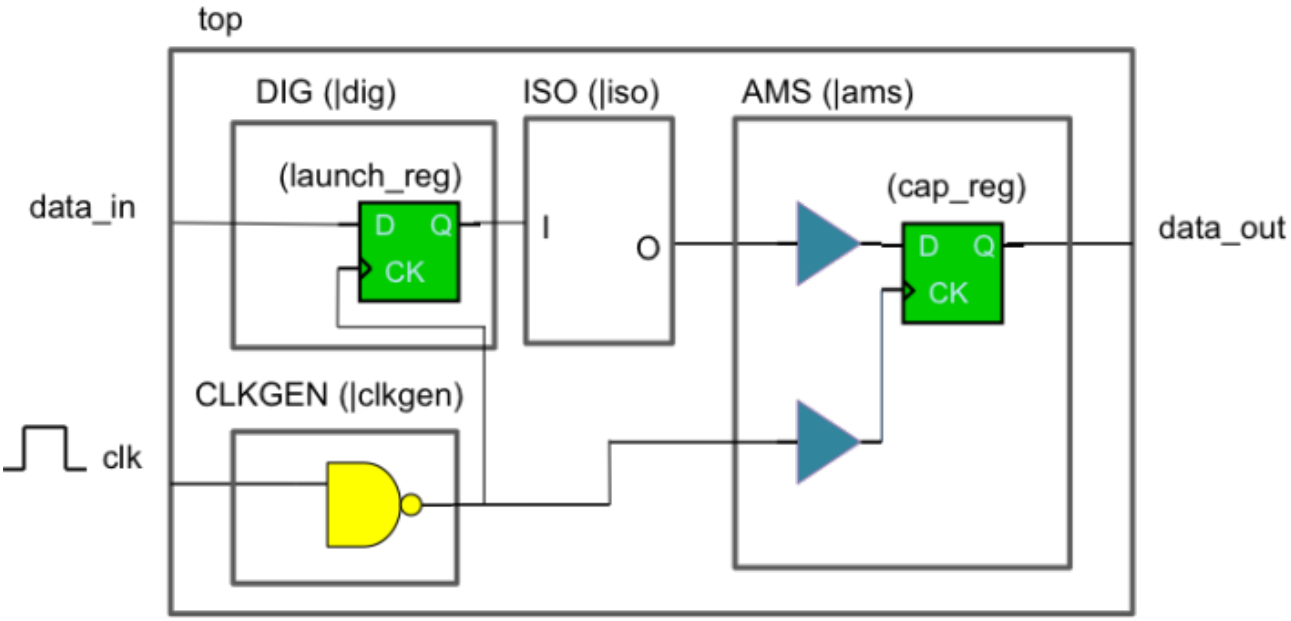
```
create_clock -name clk -period 10 [get_ports clk]
```



Assume the timing library for all the above standard cells is available and loaded. In the above case, either `assembleDesign` has to be run or FTM has to be specified for the `DIG` , `CLKGEN` , `ISO` and `AMS` blocks to report the timing between the two registers.

However, if the `ISO` block that contains the isolation cells (assuming this is a low power design) is

not flattened or its FTM is not specified, the path is viewed by the timer as follows:



As a result, the timer will report an unconstrained timing path if you run `report_timing` on the path:

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D

No constrained timing paths found.
Design may not be unconstrained or library is missing timing information.
```

To further investigate whether the propagation of the data signal stops at ISO cell, you can run the following:

```
> report_timing -from dig/launch_reg/Q -to dig/iso/I -unconstrained

Path 1:Endpoint: dig/cap_reg/D (v) (unconstrained output)
Beginpoint: dig/launch_reg/Q (^) triggered by leading edge of 'clk'
Arrival Time                    0.383
Analysis View: max
     + Clock Network Latency (Prop)   0.068
     = Beginpoint Arrival Time        0.068
   +-------------------------------------------------------------+
   |    Instance    |      Arc     |  Cell   | Delay | Arrival  |
   |                |              |         |       |  Time    |
   |----------------+--------------+---------+-------+----------|
   | dig/launch_reg | CK ^         |         |       |   0.068  |
   | dig/launch_reg | CK ^ -> Q ^  | DFFRX1  | 0.315 |   0.383  |
   | dig/iso        | I ^          | ISO 1   | 0.000 |   0.383  |
```

```
+-------------------------------------------------------+
```

The above report shows that the propagation of the data path stops at the input of the `ISO` cell, resulting in a broken path.

18.11.6.2 Cell not bound with timing library

When timing a path, the timer needs to understand the function and delay attributes of all leaf cells along the timing path. Therefore, all these leaf cells must have timing library representation. Typically, the issue of cell not bound with timing library occurs when the library set is not specified with a complete set of library files in the `viewDefinition.tcl` file.

For example, for designs with multiple timing library files, especially ones with multiple power domains (e.g. with different sets of supply voltages, or threshold voltages, for specific low power cells), a certain amount of effort and time is required to find out all the necessary timing library files to be loaded into the tool. Therefore, it is possible that in the first few runs of timing analysis, some of the timing library files are left out and not specified in the `viewDefinition.tcl` file.

To debug if a cell is bound with the timing library, run `report_instance_library` with the instance name specified.

The example below shows that the instance `dig/ram4k` is not bound with the timing library.

```
report_instance_library -instance dig/ram4k

Instance      : dig/ram4k
Analysis View : max
Power Domain  : -
Op Cond       : P->1.000000, V->0.900000, T->125.000000 (slow_oc)
```

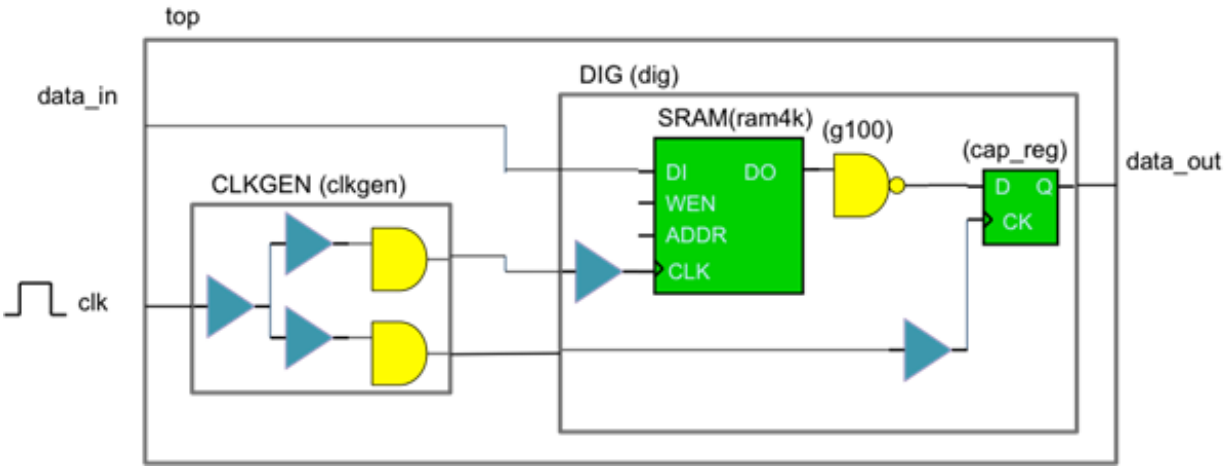
The example below shows that the instance `dig/ram4k` is bound with two timing library files, `custom_best.lib` and `custom_worst.lib`.

```
report_instance_library -instance dig/ram4k

Instance              : dig/ram4k
Analysis View         : max
Power Domain          : -
Library/Libset(early) : custom/1v32_min
Library File (early)  : /lib/custom_best.lib
Library/Libset(late)  : custom/1v08_max
Library File (late)   : /lib/custom_worst.lib
Op Cond               : P->1.000000, V->0.900000, T->125.000000 (slow_oc)
```

The following options suggest how the library set should be specified in the `viewDefinition.tcl` file for two specific situations:

Cell exists only inside a flattened block



In the above case, suppose you want to time the path from `dig/ram4k/DO` to `dig/cap_reg/D`. Assume the library information is as follows:

- The timing library model for the SRAM cell is contained in a file called `sram.lib`.
- The timing library model for all other standard cells is `slow.lib`.
- The physical library that stores the abstract cellview for the standard cells is `gsclib045`.

Specify all the timing library files, including the ones that have cells appearing only in the blocks to be flattened (e.g. SRAM), in the `viewDefinition.tcl` file:

```
create_library_set -name lib_worst -timing [list lib/slow.lib lib/sram.lib]
```

The following is a sample script to run the flattened flow.

```
set_db init_design_netlisttype {OA}

set_db init_oa_design_lib {designLib}

set_db init_oa_design_cell {top}

set_db init_oa_design_view {layout}

set_db init_oa_ref_lib {gsclib045}

set_db init_mmmc_file {viewDefinition.tcl}

init_design

setExtractRCMode -engine postRoute

setExtractRCMode -effortLevel high
```

```
assembleDesign -block {designLib DIG layout} -block {designLib CLKGEN layout}

report_timing -from dig/ram4k/DO -to dig/cap_reg/D
```

Cell exists only inside an FTM block

Similarly, in the FTM flow, specify all the timing library files in the `viewDefinition.tcl` upfront. An example is shown below:

```
create_library_set -name lib_worst -timing [list lib/slow.lib lib/sram.lib]
```

A sample script for the FTM flow is shown below:

```
set_db init_design_netlisttype {OA}

set_db init_oa_design_lib {designLib}

set_db init_oa_design_cell {top}

set_db init_oa_design_view {layout}

set_db init_oa_ref_lib {gsclib045}

set_db init_mmmc_file {viewDefinition.tcl}

init_design

specifyILM -cell DIG -dir dig_FTM

setExtractRCMode -engine postRoute -effortLevel high

flattenILM

update_constraint_mode -name func -ilm_sdc_files {simple.sdc}

report_timing -from dig/ram4k/DO -to dig/cap_reg/D
```

18.11.6.3 Bad timing library

The cell is bound with the timing library but the timing description in the timing library is incomplete. For example, the following cell appears to be a buffer or inverter but the pin's specification has no timing arc description:

```
    pin(A) {
        direction : input;
        capacitance : 0.008908;
    }

    pin(Y) {
        direction : output;
```

```
    capacitance : 0.0;
}
```

A proper specification for pin Y should describe its function and relationship to a particular input pin and should contain timing sense description that talks about unateness and timing look up tables for cell rise delay, rise transition, cell fall delay, and fall transition. A simple example is shown below.

```
pin(Y)  {
    direction : output;
    output_signal_level : RAIL_VDD;
    capacitance : 0;
    max_capacitance : 0.507904;
    function : "A";
    timing() {
      related_pin : "A";
      timing_sense : positive_unate;
      cell_rise(delay_template_2x2) {
        index_1 ("0.008, 0.28");
        index_2 ("0.01, 0.45");
        values ( \
          "0.079673, 0.877286", \
          "0.178436, 0.976339");
      }
      rise_transition(delay_template_2x2) {
        index_1 ("0.008, 0.28");
        index_2 ("0.01, 0.45");
        values ( \
          "0.056876, 1.66719", \
          "0.060035, 1.66718");
      }

      cell_fall(delay_template_2x2) {
        index_1 ("0.008, 0.28");
        index_2 ("0.01, 0.45");
        values ( \
          "0.079517, 0.996017", \
          "0.174628, 1.09146");
      }

      fall_transition(delay_template_7x7) {
```

```
            index_1 ("0.008, 0.28");
            index_2 ("0.01, 0.45");
            values ( \
              "0.060737, 1.87573", \
              "0.063819, 1.8791");
          }
        }
    }
```

18.11.6.4 Logical connectivity issue in the path

Typically, a logical connectivity issue occurs when the design or the block to be timed is not Virtuoso XL-compliant. Refer to the Requirements for Correct Connectivity Propagation section for details on connectivity propagation. Note that the logical connectivity issue is different from the physical connectivity issue. A physical connectivity issue can be as simple as a net not routed or the wire not touching the pin shape of a cell, causing it to be an open violation. However, a logical connectivity issue refers to, for example, the tool not being able to see any logical connection between two blocks. This could be because one of the blocks has no terminals or has terminals that have been left dangling logically, even though wires are physically connecting the two blocks.

Note: To avoid misunderstanding, this section uses the word "terminal" to refer to the input or output of a cell. In static timing analysis, the input or output of a cell is often referred as a "pin" or "port". In Virtuoso, the "pin" usually refers to the physical pin shape while "terminal" refers to the corresponding logical port.

To check for logical connectivity issue, you can:

1. Open Design Browser from the *Tools* menu in Innovus to check if the input or output of an instance is connected as expected.
2. Dump out a verilog netlist in Innovus using the `saveNetlist` command and view the content of the Verilog file to check the connection of the instances,
3. Load the top-level design in Virtuoso, select the instance, and select *Connectivity* -> *Nets* -> *Propagate…* from the top pull-down menu to view the logical connectivity of the terminals.

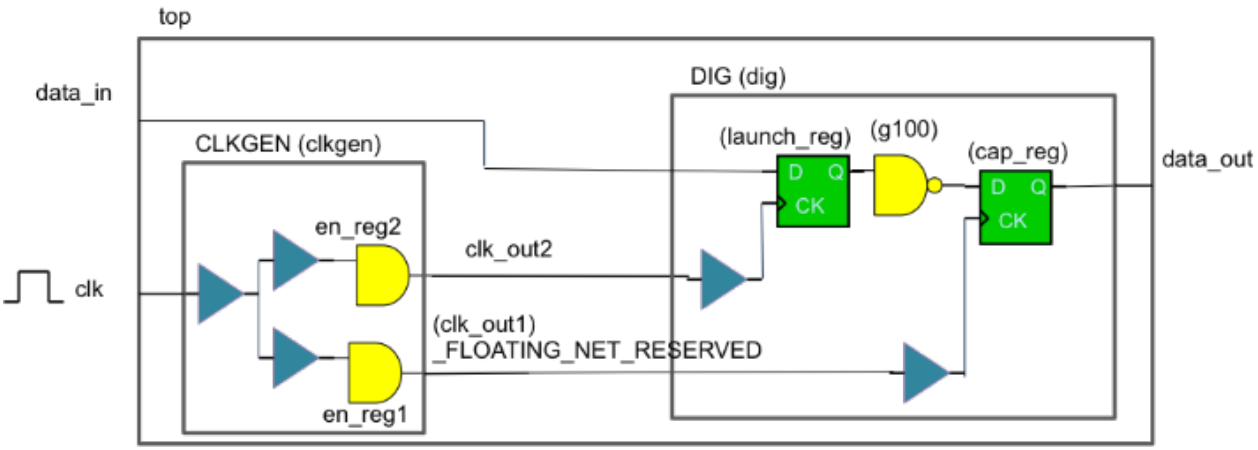
The following situations could lead to a logical connectivity issue in the path:

Terminal not connected or shape having no connectivity

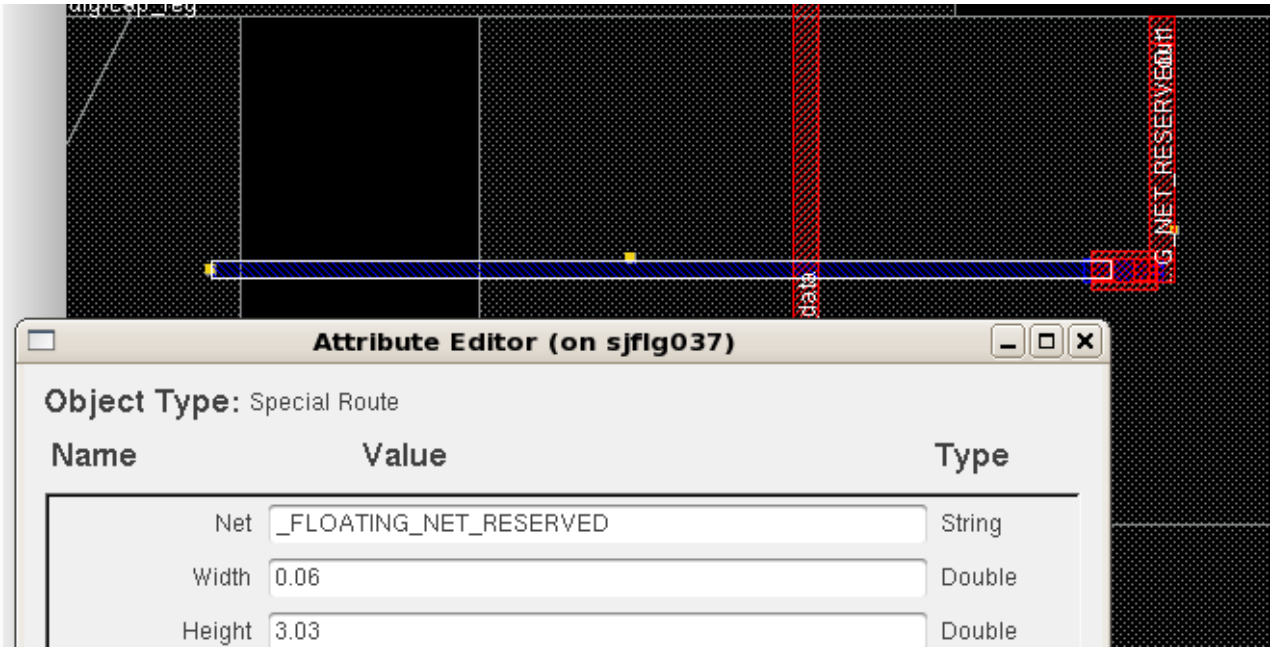
Typically, this issue occurs because the physical wire does not have an associated net name. In Virtuoso, it is usually referred as shape having no connectivity.

In the following case, a logical connectivity issue exists at the top-level net, `clk_out1`, which is

viewed as a floating net in Innovus.

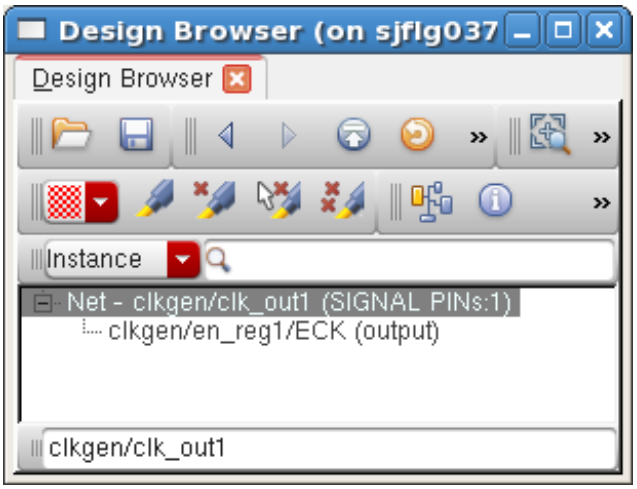


When you select the floating net in Innovus (left-click the wire) and view its attributes (press q), it is displayed as `FLOATING_NET_RESERVED`.



When you view the netlist or check in Design Browser, the output of the `clkgen/en_reg1` instance appears to be dangling.

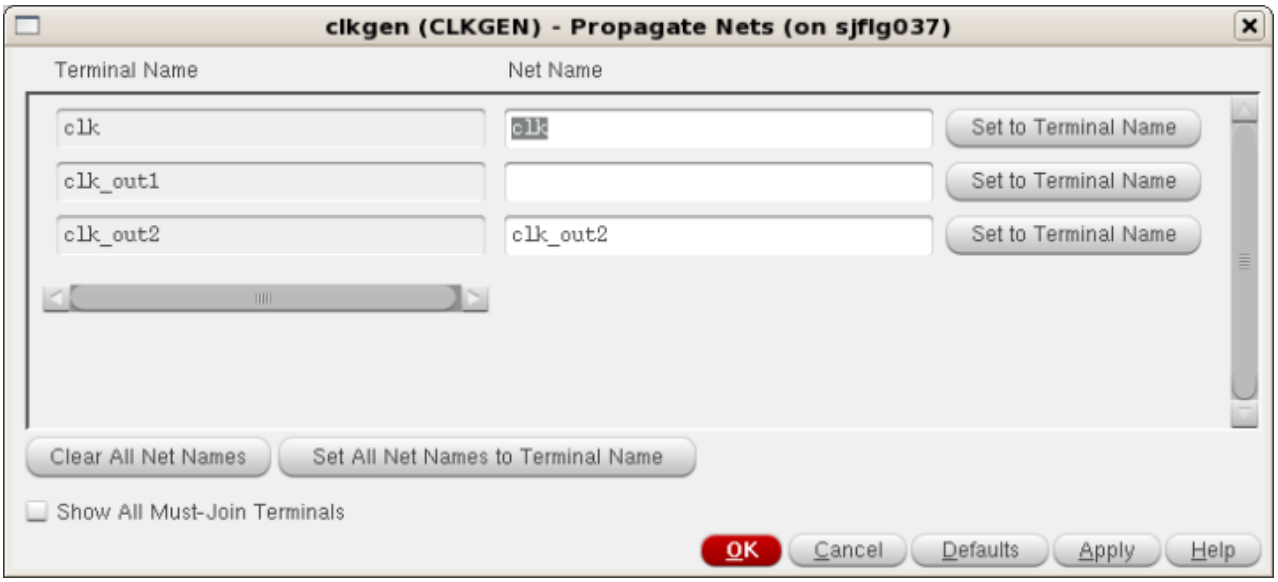
```
module top
...
CLKGEN clkgen (.clk_out2(clk_out2),
.clk(clk));
...
endmodule
```



However, as can be seen in the following screenshot, the output of the `clkgen/en_reg2` is connected to the top level net, `clk_out2`.



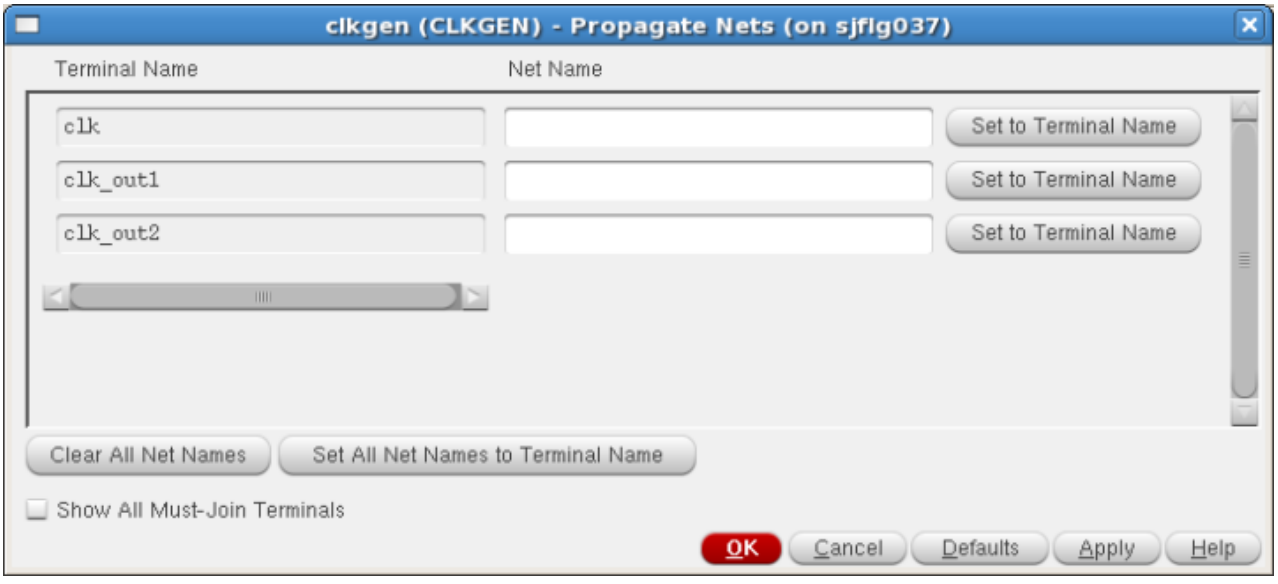
If you open the top-level design in Virtuoso and select the `CLKGEN` block, and then select *Connectivity -> Nets -> Propagate…* from the top pull-down menu, the following information can be seen:



In some cases, it could be that none of the terminals are connected, and the netlist displays as follows:

```
module top
CLKGEN clkgen ();
...
endmodule
```

In this case, when you open the top-level design in Virtuoso and select the `CLKGEN` block, and then select *Connectivity* -> *Nets* -> *Propagate…* from the top pull-down menu, the following information can be seen:



Such a situation could happen if the block is imported to Virtuoso from a GDSII file (which has no

connectivity information). Typically, for a block imported through GDSII, the content of the Verilog netlist generated from the block has the following characteristics:

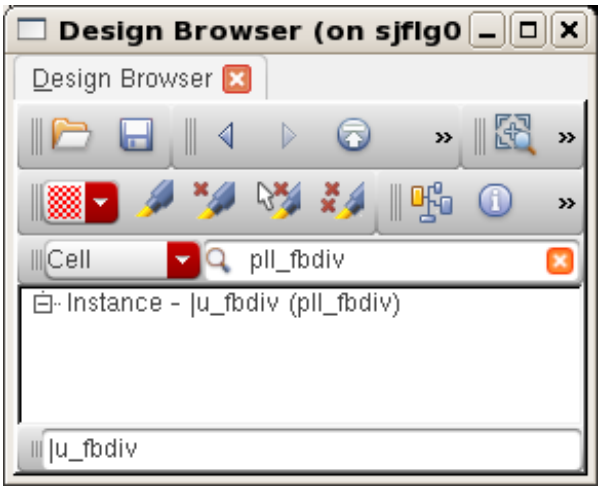
- The instance name starts with `I__xxxx` , where *xxxx* are numbers.
- No logical connection exists between instances.

An example is shown below:

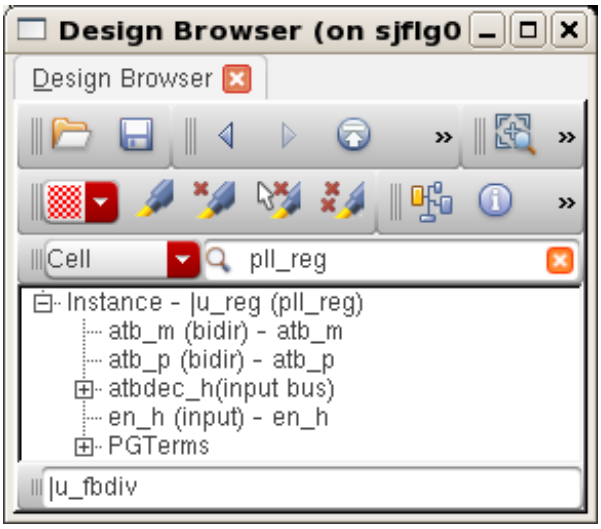
```
SDFFSHQX2 I__1729 ();
TLATNX1 I__1730 ();
TLATNX1 I__1731 ();
AOI222X1 I__1732 ();
...
```

Missing terminal

In the previous case, the block has terminals but the terminals are dangling. In this case, the block to be assembled has no terminals at all. This is another example of non-compliance of Virtuoso XL. To check if a cell has a terminal in Innovus, select the instance in the Innovus main window and open Design Browser from the *Tools* menu. Alternatively, just open the Design Browser and enter the cell name.



Here, the cell `pll_fbdiv` has no terminals. In contrast, the cell `pll_reg` in the screenshot below has several terminals.



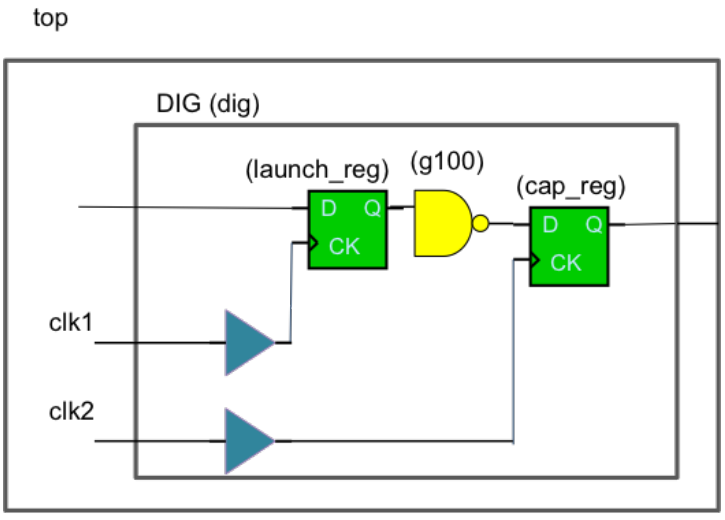
Remember to click the + symbol to the left of Instance to check if terminals exist for the cell. The + symbol becomes - when expanded.

18.11.6.5 Path not to be timed

In this case, the path is intended to be unconstrained, either by definition in the timing library or in the timing constraint file. Some situations where an unconstrained path may be used are listed below:

False path

A false path is a timing path that is not required to meet its timing constraints for the design to function properly. One typical example is an asynchronous path between multiple clocks.



In the timing constraint file, the constraints are as follows:

```
create_clock -name clk1 -period 10 [get_pins dig/clk1]
create_clock -name clk2 -period 10 [get_pins dig/clk2]
set_false_path -from [get_clocks clk1] -to [get_clocks clk2]

> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D

No constrained timing paths with given description found.
Paths may be unconstrained (try '-unconstrained' option) or may not exist.
```

To look for these kind of false paths, you can run `report_path_exceptions`:

```
> report_path_exceptions

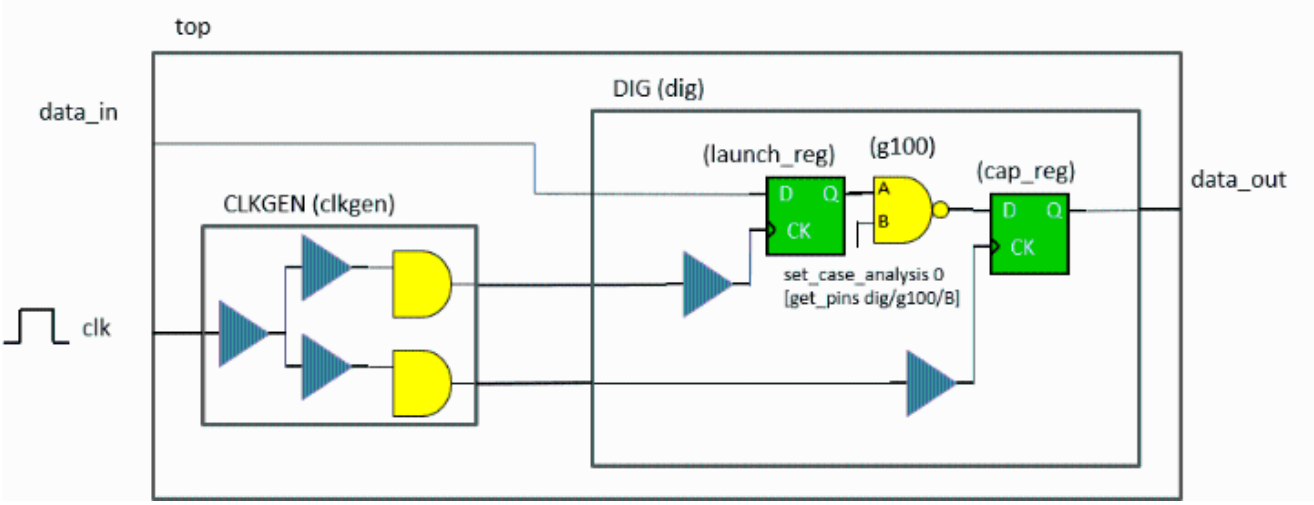
   +--------------------------------+
  | From   |   To   |  Late |  View |
  |        |        |       |  Name |
  |--------+--------+-------+-------|
  | "clk1" | "clk2" | false | max   |
  +--------------------------------+
```

Disabled timing arcs

Static timing analysis uses timing arcs to describe the cell delay from each input to each output, and the constraints between input pins. Disable a timing arc breaks a specific timing path. When a pin on a cell get disabled, all timing arcs from or to that pin is disabled. The timing arc or pin on a cell can be disabled by timing constraint, constant propagation during timing analysis, or by the timing library.

Examples of timing constraints that disable timing arcs are `set_disable_timing`, `set_case_analysis`, and `set_disable_clock_gating_check`.

Innovus provides TCL commands, such as `report_inactive_arcs` and `report_case_analysis` , to help you analyze such kinds of timing exceptions.



For the above example, assume the following constraints are specified in the timing constraint file:

```
create_clock -name clk -period 10 [get_ports clk]

set_case_analysis 0 [get_pins dig/g100/B]

> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D

No constrained timing paths with given description found.
Paths may be unconstrained (try '-unconstrained' option) or may not exist.
```

When you run `report_inactive_arcs` and `report_case_analysis`, the following reports are generated:

```
> report_inactive_arcs -type const

 +-------------------------------------------------------------------------------+
  | From        | To          | ArcType       | Sense           | Reason          | View |

  |-------------+-------------+---------------+-----------------+-----------------+------|
  | dig/g100/A | dig/g100/Y | combinational | negative_unate | dig/g100/Y = 1 | max  |

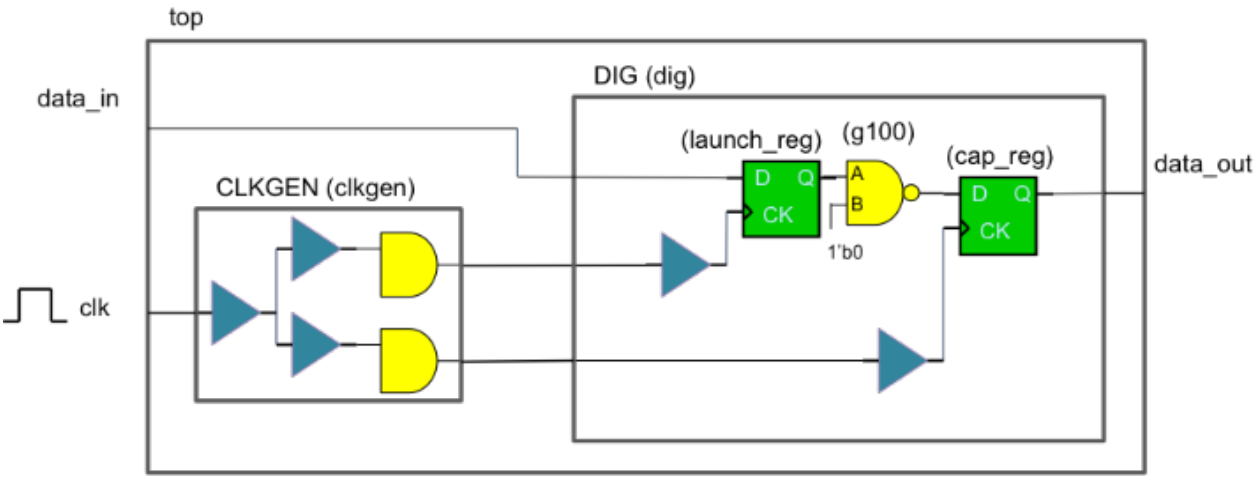
  | dig/g100/B | dig/g100/Y | combinational | negative_unate | dig/g100/B = 0 | max  |
  +-------------------------------------------------------------------------------+

> report_case_analysis

 +-------------------------------+
  |  Pin name  |   User    | View  |
  |            |   case    | Name  |
  |            | analysis  |       |
  |            |   value   |       |
  |------------+-----------+-------|
```

```
| dig/g100/B | 0          | max    |
+-------------------------------+
```

For the following example which is due to constant propagation, use `report_inactive_arcs` because `report_case_analysis` reports disabled arcs only for case analysis.



The timing constraint is:

```
create_clock -name clk -period 10 [get_ports clk]
```

```
> report_timing -from dig/launch_reg/Q -to dig/cap_reg/D
```

```
No constrained timing paths with given description found.
Paths may be unconstrained (try '-unconstrained' option) or may not exist.
```

When you run `report_inactive_arcs`, the following reports are generated.

```
> report_inactive_arcs -type const
```

```
 +-------------------------------------------------------------------------------------+
 | From        | To          | ArcType       | Sense           | Reason          | View |

 |-------------+-------------+---------------+-----------------+-----------------+------|
 | dig/g100/A | dig/g100/Y | combinational | negative_unate | dig/g100/Y = 1 | max  |

 | dig/g100/B | dig/g100/Y | combinational | negative_unate | dig/g100/B = 0 | max  |
 +-------------------------------------------------------------------------------------+
```

```
> report_case_analysis
```

`report_case_analysis` does not report anything because there is no `set_case_analysis` statement in the timing constraint.

19

Chip Finishing and ECO Flows

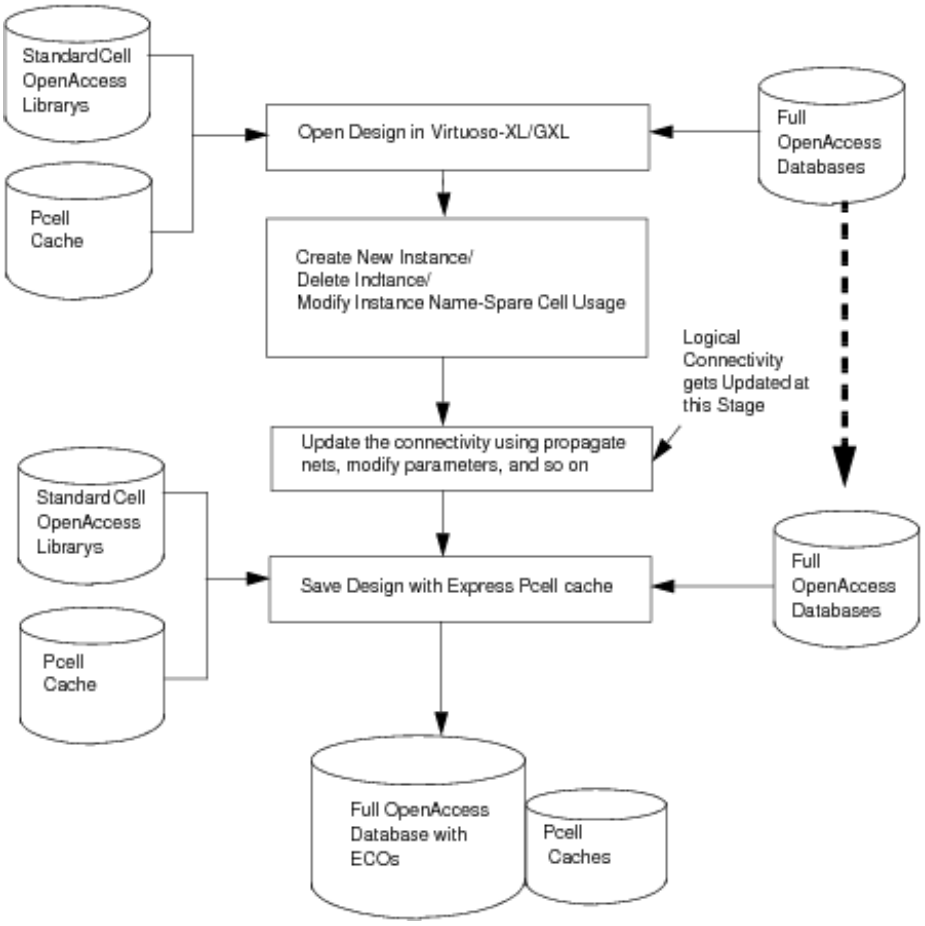
- Overview
- Virtuoso-Based ECO Flow
- Innovus-Based ECO Flow
  - Overview
  - Pre-Mask ECO Flow Steps
  - Post-Mask ECO Flow Steps
  - Example Post-Mask ECO Scenarios

19.1 Overview

Most of the custom design level sign-off analysis is done in Virtuoso. The sign-off timing analysis of the flat top-level digital portion of the design can be done in Innovus. The mixed-signal functional simulation gives functional performance of the design at any stage of the design cycle.

Metal filling, optimization of metal density can be done in Innovus also which can be taken into account by QRC during extraction so that the timing effects due to metal fill can be addressed.

19.2 Virtuoso-Based ECO Flow



Use the following steps to run the Virtuoso-based ECO flow when the design changes are made at the top-level schematic:

1. **Large ECO**: For large ECO changes, such as when the intent is to use only the existing floorplan, hard-block placement, reuse of power mesh, and so on, follow these steps:
    - Run `verilog2oa` to import the new netlist in a new cellview.
    - Use the *Load Physical View* command in Virtuoso-XL Floorplanner to read the information from the old physical database selectively.
2. **Small ECO**: To add/create a hierarchical ECO instance in Virtuoso in case of small ECOs:
    - If the schematic is updated with minor changes to the net or instances, these changes can be tracked and data can be updated automatically in the Virtuoso-XL environment. You can use the *Check Against Source* command to track the changes and the *Update Components and Nets* command to update the layout as per the new schematic.

- You can also make the changes manually. To create a hierarchical instance, use the symbol ` which indicates a module hierarchy present in the design. For example, an instance with name M1`A1 implies that a module hierarchy present with the A1 module inside the module M1.

For the VDI flow for AoT designs, use the following steps when there are changes in the digital block netlist or the floorplan:

- **Netlist has changes:** The digital block is implemented but there is a new netlist for the block.
  - If the netlist changes are large, use the `oaIn -filter` option to load in selective data like macro placement, power routing and power domains and pre-routes from the previously implemented cellview. To accomplish this, you need to initialize the new design in Innovus by invoking Innovus and using the new netlist to create a new cellview. You can then use `oaIn -filter` to selectively import physical objects from the older cellview, which was generated with the old Verilog netlist. For example, if you are interested in importing the block placement from the older cellview into the new cellview (the cellview created from the new Verilog netlist), use the following command:

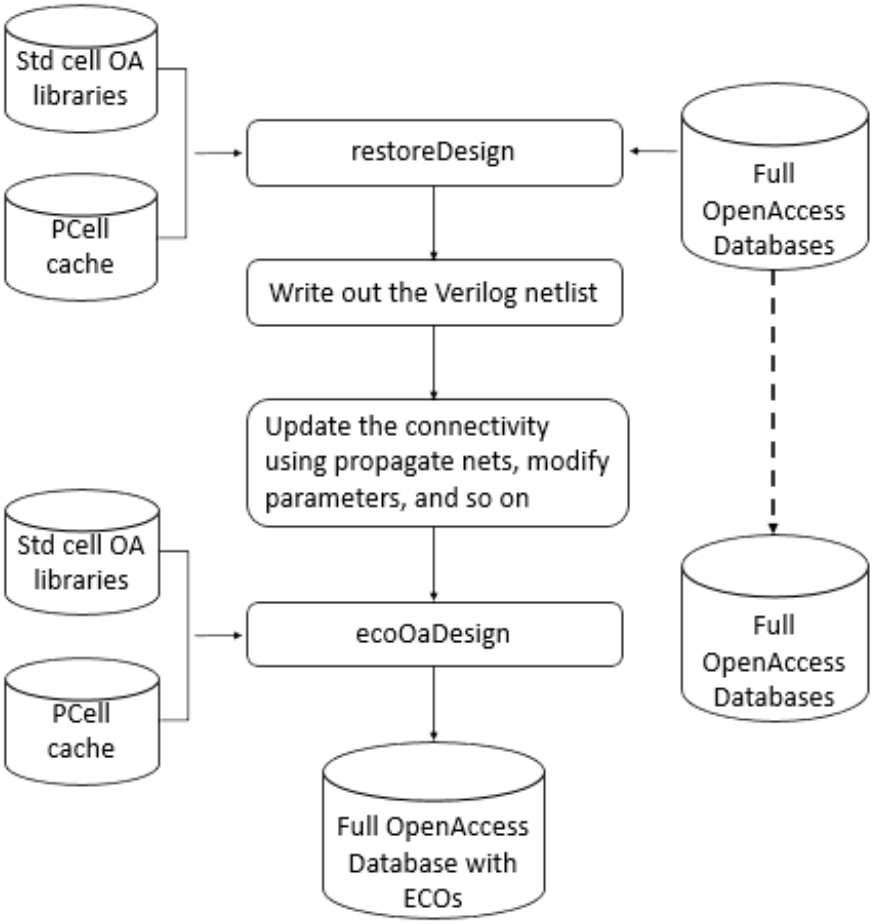
    ```
    oaIn lib cell view -filter block_insts
    ```

    Here, *`lib cell view`* refers to the old cellview.
  - If the netlist changes are small and you would like to use as much of the original cellview information, use the `ecoOaDesign` command.
- **Netlist remains same:** The digital block is implemented but the block shape is changed, or pins are moved or their shapes changed.
  - In this case, the Verilog netlist has not changed, so all that needs to be done is to import the new block boundary or pin location and update the previously implemented cellview. To do this, use the `oaIn -filter` command to load in the new block shape or pin location as follows:

    ```
    oaIn lib cell view -filter boundary
    ```

Due to ECO changes, if a net gets modified, it needs re-routing using automatic router, use Innovus platform to route using NanoRoute.

19.3 Innovus-Based ECO Flow



19.3.1 Overview

The digital ECO flow is a methodology used to integrate the netlist modifications on a given design with minimal changes in the physical database to minimize the mask costs. This is only practical if the amount of change is minimal. Changes can also be achieved by modifying the interconnections of the existing netlist.

In the post-mask ECO flow, when a design has been taped out and requires logical changes, the corrections are performed by changing the logic in the Verilog file written from the old taped-out database. The pre-existing spare cells or the existing logic is used to accomplish the logical changes expected so that the poly/diffusion and lower layers are not changed, and only the metal and via layer masks are modified. To save mask cost, you can direct the tool to perform routing changes only within the specified layer range. The expected top-level flow steps are:

1. Import the new Verilog file and pre-ECO design database into Innovus.

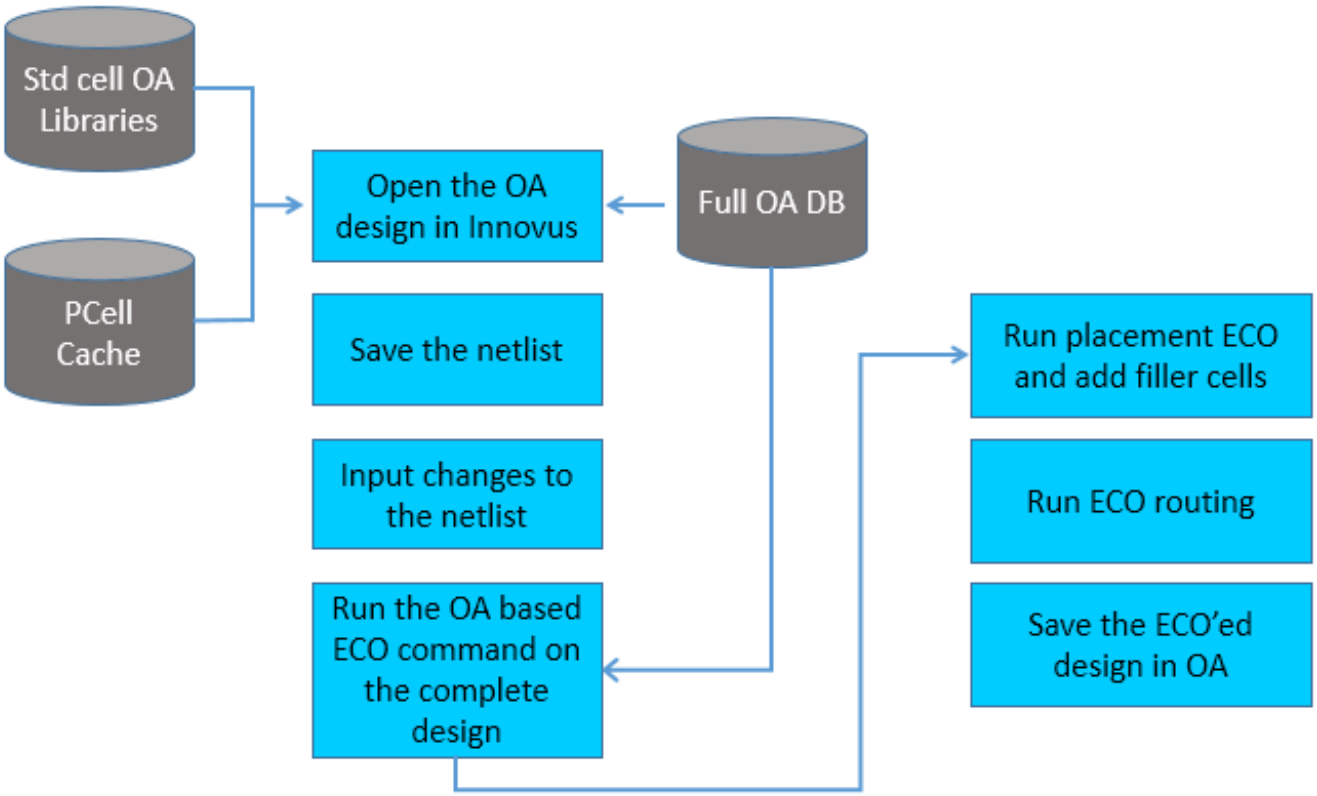
2. Map new cells on spare cells, and map the deleted cells to be available as spare cells.
3. Perform interactive ECOs such as `ecoSwapSpareCell`, if needed.
4. Run `ecoRoute` with only a few layers getting changed.

To complete the ECO flow, you must provide the following inputs:

- New Verilog with logical modifications completed.
- Pre-ECO design database.
- List of spare cell modules to be used for ECO changes. If spare cells have been specified in previous Innovus sessions before saving the OpenAccess database, then they would be restored during `ecoOaDesign` command.
- Permitted routing layers during `ecoRoute`.

19.3.2 Pre-Mask ECO Flow Steps

The following figure depicts the pre-mask ECO flow that implements changes made through the netlist.



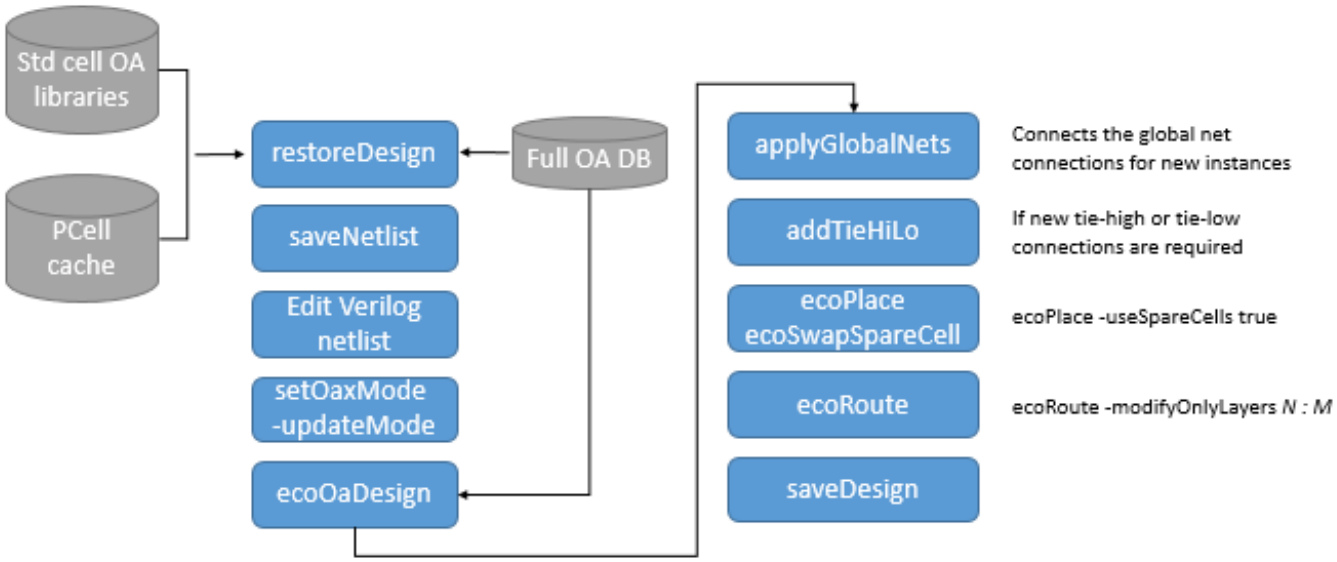
Follow these steps to run the pre-mask ECO flow:

1. Run `restoreDesign` to load pre-ECO library/cell/view. There is no need for a LEF technology library, as a common OpenAccess PDK can be used for the Innovus and Virtuoso environments, containing all technology information needed for the mixed signal implementation flow.
2. Run `saveNetlist` to write out the old Verilog (with parameterized cells (PCells) in it, if they exist in the design).
3. Exit Innovus.
   Note: If the old library/cell/view have no PCells, that is, it is a digital-only netlist, you can use the standalone utility `oa2verilog` directly. The Innovus commands `restoreDesign` and `saveNetlist` are required if PCells are present in the design.
4. Edit the Verilog netlist manually to implement the digital ECO changes.
5. Start a new Innovus session. Run `setOaxMode -updateMode true` if there are analog objects like PCells, MPPs, fig-groups, in the old OpenAccess database. If this is a digital-only design without Virtuoso-specific objects or constraints, you do not need to use `setOaxMode -updateMode`.
6. Run `ecoOaDesign` to read the new ECO Verilog netlist, which copies all of the floorplanning, placement, routing and spare bit marking from the pre-ECO OpenAccess database except those not existing in the new Verilog file. Instances existing only in pre-ECO library/cell/view would not be added in the new Innovus database.
7. Run the `applyGlobalNets` command to connect the global net connections for new instances. If your design is low-power-aware and you have a Common Power Format (CPF) file, use the `read_power_intent -cpf` and `commit_power_intent` commands to accomplish the above tasks.
8. Use `addTieHiLo` if new tie-high or tie-low connections are needed.
   `[ -cell "`*`tieHighCellName tieLowCellName`*`"]`
   `[ -createHierPort {true|false}]`
9. Run `ecoPlace` to place the new unplaced cells. According to requirements, the `deleteFiller -prefix FILL` and `deleteNotchFill` commands may be required to delete the existing filler cells and notch fills before running `ecoPlace`.
10. Optionally, you can run the `selectInst` and `placeInstance` commands to achieve fine-grain optimization in placement.

11. Use the `addFiller -cell {...}` command to add filler cells and then run `ecoRoute` as required in the flow. After pereforming ECO routing, you need to take care of DRC, DFM and DFY changes as per requirement. For example, running the `addMetalFill`, `fillNotch`, and `trimMetalFill` commands.
12. Run `saveDesign -cellview` to save the database in the new library/cell/view.

19.3.3 Post-Mask ECO Flow Steps

The following figure depicts the post-mask ECO flow.



Follow these steps to run the post-mask ECO flow:

1. Run `restoreDesign` to load the pre-ECO library/cell/view.
2. Run `saveNetlist` to write the pre-ECO Verilog file.
3. Exit the Innovus session.
   Note: If the pre-ECO library/cell/view has no PCells, that is, it is a digital-only netlist, you can use the standalone utility `oa2verilog` directly. The Innovus commands `restoreDesign` and `saveNetlist` are required if PCells are present in the design.
4. Edit the Verilog netlist manually to implement the digital ECO changes.
5. Start a new Innovus session. Use `setOaxMode -updateMode true` if there are analog objects like PCells, MPPs, fig-groups, in the pre-ECO OpenAccess database. If this is a digital-only

design without full-custom objects or constraints, you do not need to use `setOaxMode -updateMode`.

6. Run `ecoOaDesign` with the `-postMask` option to read the new ECO Verilog netlist, which copies all of the floorplanning, placement, routing and spare bit marking, from the pre-ECO OpenAccess database.
7. Run the `applyGlobalNets` command to connect the global net connections for new instances.
8. Run `addTieHiLo` if new tie-high or tie-low connections are needed:

   `[ -cell "`*`tieHighCellName tieLowCellName`*`"]`

   `[ -createHierPort {true|false}]`

   `-postMask`

   During this step in the post-mask mode, no new cells would be added. Only the already existing cells are used.
   If you have Common Power Format (CPF) file, use the `read_power_intent -cpf` and `commit_power_intent` commands to accomplish the above tasks.
9. Run `ecoPlace -useSpareCells true`, as required to map unplaced cells to spare cells.
10. Run `ecoSwapSpareCell` if required in the flow, when auto-mapping was not according to your expectations.
11. Run `ecoRoute -modifyOnlyLayers N:M`, as required in the flow.
12. Run `saveDesign -cellview` to save the database in the new library/cell/view.

19.3.4 Example Post-Mask ECO Scenarios

19.3.4.1 Pre-ECO Verilog

```
module spare ();
     // Internal wires
       wire LTIELO_1_NET;
       wire LTIELO_NET;
       TIELO LTIELO_1 (.Y(LTIELO_1_NET));
       TIELO LTIELO (.Y(LTIELO_NET));
       BUFX4 U6 (.A(LTIELO_NET));
       BUFX4 U7 (.A(LTIELO_1_NET));
endmodule

module TOP (Z, A);
    output Z;
    input A;
    // Internal wires
    wire net_pcell;

    inv_pcell I1 (.OUT(Z), .IN(net_pcell));
    BUFX8 U1 (.Y(net_pcell), .A(A));
    spare SU1 ();
endmodule
```

Note: In the pre-mask ECO flow, the newly added or modified instances will come as unplaced cells. The old cells (`INVXL`) are deleted from the database. The net physical geometries remain unchanged.

19.3.4.2 Post-ECO Verilog

```
module spare ();
   // Internal wires
     wire LTIELO_1_NET;
     wire LTIELO_NET;



   TIELO LTIELO_1 (.Y(LTIELO_1_NET));
   TIELO LTIELO (.Y(LTIELO_NET));
   BUFX4 U6 (.A(LTIELO_NET));
   BUFX4 U7 (.A(LTIELO_1_NET));
endmodule

module TOP (Z, A);
   output Z;
   input A;
   wire net_pcell, net1;
   inv_pcell I1 (.OUT(Z), .IN(net_pcell));
   BUFX4 ECO1 (.A(A), .Y(net1) );
   BUFX4 ECO2 (.A(net1), .Y(net_pcell) );
   BUFX8 U1 (.A(1'b0));
   spare SU1 ();
endmodule
```

Note: In the post-mask ECO flow, the newly added or modified instances (`BUFX2`) will come as unplaced cells. They can be mapped to a spare cell using the `ecoPlace -useSpareCells` parameter. The pre-ECO cell (INVXL) will remain in the database with placement information same as in the pre-ECO database. The physical geometries (nets) remain unchanged.

19.3.4.3 *Example Command Sequence*

```
ecoOaDesign designLib TOP layout -ecoVerilogFile eco.v -reportFile eco.rpt -postMask
applyGlobalNets
ecoPlace -useSpareCells true
addTieHiLo -postMask
ecoRoute -modifyOnlyLayers 1:2
```

20

OpenAccess Database Interoperability Checker

- Overview
- Running the OA DB Checker
- Check Library - Innovus Interoperability Library Checker
  - Technology DB Checker
  - Library DB Checker
  - Check Pins between Two Cellviews for the Remaster Instance
  - Report File Name
- Check Design - Innovus Interoperability Design Checker
  - CellView(s)
  - Report File Name
  - Checks
- Create Check File - Create Check Design File
- Viewing OA DB Checker Violation Markers in the Annotation Browser in Virtuoso
- Run Innovus - Innovus Launch GUI
- OA DB Checker Use Case Scenarios for Virtuoso Users

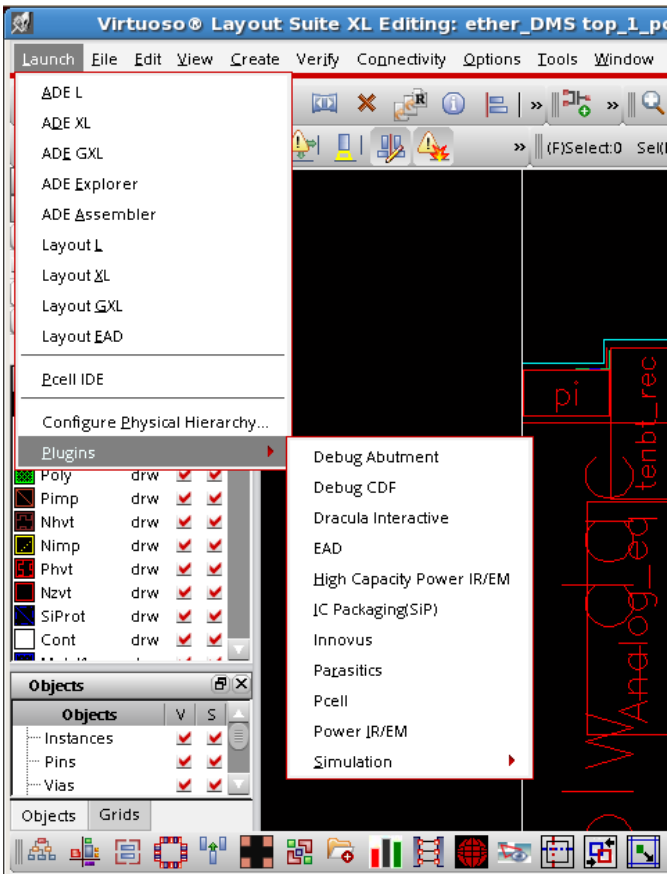
20.1 Overview

OpenAccess Database Interoperability Checker (OA DB Checker) is a SKILL-based utility that can be run in the Virtuoso environment to ensure that the OpenAccess database when brought inside Innovus™ Implementation System is compatible for place-and-route flow. The OA DB Checker is also referred as the Innovus Interoperability Checker. To ensure interoperability, the OpenAccess database should contain all the necessary database objects that are required by applications within Innovus. At the same time, it should not contain any database object that would stop the Innovus implementation flow.

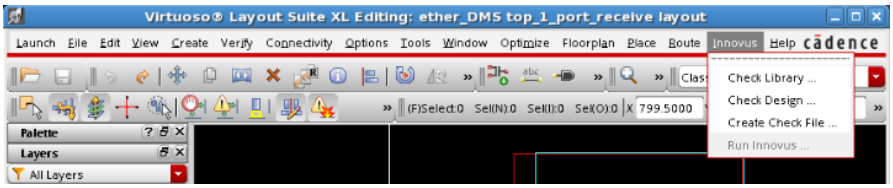
The OA DB Checker, 16.20.002 revision and above, is available through the *Innovus* plug-in in Virtuoso. To learn more about running the OA DB Checker, see Running the OA DB Checker.

20.2 Running the OA DB Checker

The OA DB Checker is available through the *Innovus* plug-in in the Virtuoso *Launch* menu. To launch the OA DB Checker, select *Innovus* from the *Launch* --> Plugins submenu in Virtuoso.



Once the plug-in is invoked, *Innovus* appears as a menu header on the Virtuoso window.

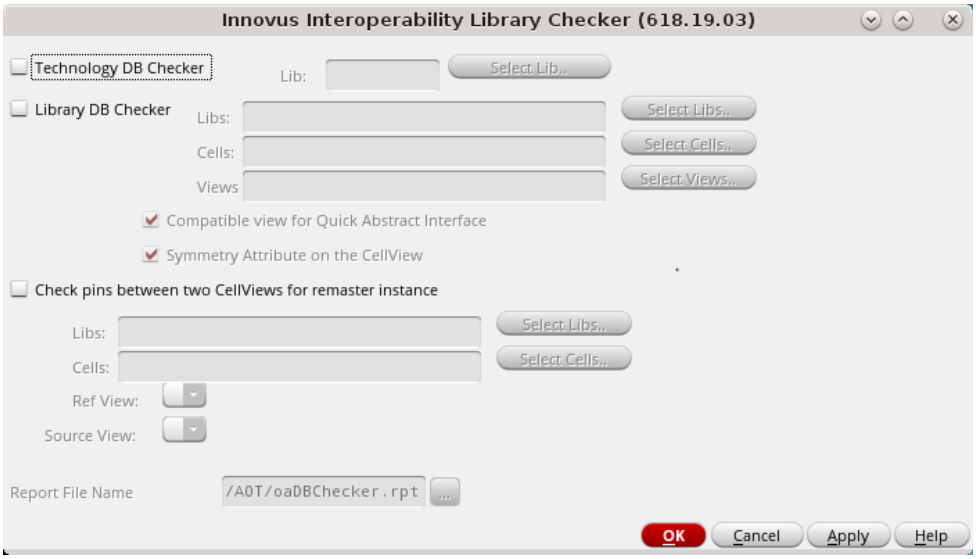


The *Innovus* menu in Virtuoso provides the following options:

- *Check Library* - Opens the Innovus Interoperability Library Checker form, which enables you to check the completeness and correctness of technology data and the reference library.
- *Check Design* - Opens the Innovus Interoperability Design Checker form, which enables you to check the correctness and completeness of the design library.
- *Create Check File* - Opens the Create Check Design File form, which enables you to generate a file containing the lib/cell/views from your design.
- *Run Innovus* - Opens the Innovus launch GUI form, which enables you to perform library and design checks and open the design in Innovus Implementation System. The *Run Innovus* option is enabled only if Innovus is in your installation path.

20.3 Check Library - Innovus Interoperability Library Checker

Select the *Innovus - Check Library* option from the Virtuoso menu bar to open the Innovus Interoperability Library Checker form.



The Innovus Interoperability Library Checker form contains the following sections:

- *Technology DB Checker*
- *Library DB Checker*
- *Check pins between two CellViews for remaster instance*
- *Report File Name*

20.3.1 Technology DB Checker

Checks for completeness of technology data in the specified library.

Many applications within Innovus require certain information to be present completely, consistently, and correctly in the OpenAccess technology database. This information includes definitions of layers, vias, routing, and DRC rules. Some of the completeness checks are tricky and vary from one application to other. For example, `LEFDefaultRouteSpec` and `LEFSpecialRouteSpec` are requirements for automatic routers but wire editors in Virtuoso do not require them.

To ensure that the required data is present in the technology database, the checker performs the following checks:

- The foundry constraint group exists in the technology.
- The `LEFDefaultRouteSpec` constraint group is present in the technology.
- Each pair of consecutive routing layers has a cut layer defined between them.
- The width constraint is specified in `LEFDefaultRouteSpec` for each routing layer. If the width constraint in `LEFDefaultRouteSpec` is less conservative than the foundry width value, the checker reports this to the user.
- Innovus always reads the spacing constraint from foundry. The spacing constraint must be specified in the foundry constraint group. If the value of a spacing constraint in `LEFDefaultRouteSpec` is different from its foundry value, Innovus reports the inconsistency and displays a message that the foundry spacing value will be used by the Innovus technology reader.
- If more than one spacing value is specified, the values must be increasing monotonically.
- Innovus always reads the layer pitch constraint from foundry. It must be specified in the foundry constraint group. If a pitch constraint is present in `LEFDefaultRouteSpec` with its value different from the foundry value, the inconsistency is reported.

  Note: Innovus technology reader uses the foundry spacing value.
- The `oacValidRoutingLayer` constraint is defined in `LEFDefaultRouteSpec` and specifies all the routing layers in the technology, except the

backside layer and the minimum capacitance layer.

- The `oacValidRoutingVias` constraint is defined in `LEFDefaultRouteSpec` and has at least one via between every two consecutive routing layers. TSV (via from metal to backside routing layer) and via to top routing layer may not be included in the valid vias list.
- The `LEFSpecialRouteSpec` constraint group is present in the technology and has a via rule for every two consecutive routing layers. Exceptions of top routing layer and via to top routing layer are not applicable on `LEFSpecialRouteSpec`.
- There are no PCell vias in the validVias list of the LEFDefaultrouteSpec (LDRS) and Non-Default Rules (NDRs). If any PCell via is present, the checker reports its name along with the associated constraint group during technology library checking. For example:

  ```
  Checking for PCELL vias in validVias list of NDRs....
  The validVias list of the constraint group 'virtuosoDefaultSetup' from library 'LIB' contains a pcell via 'M1V1M2_C'. The pcell via should be removed from the validVias list and, if necessary, replaced by a custom (fixed) via definition or a standard (paramaterizable) via definition.
  The validVias list of the LEFDefaultRouteSpec 'LEFDefaultrouteSpec' from library 'LIB' contains a pcell via 'M1V1M2_C'. The pcell via should be removed from the validVias list and, if necessary, replaced by a custom (fixed) via definition or a standard (paramaterizable) via definition.
  FAILED : Technology library 'LIB' fails certain checks.
  ```

  For interoperability with Innovus, you should remove the reported P-cell vias from the validVias list. If necessary, you can replace the P-cell via with a custom (fixed) via definition or a standard (paramaterizable) via definition.
- All custom vias available in the tech file should be available in the Incremental Technology Database (ITDB). If the checker finds references to custom vias in the tech file (under the LDRS section) but cannot find the custom via master in the ITDB/PDK, it issues a WARNING message such as the follows:

  ```
  WARNING: "Custom via found in the design for which there is no via cell master. Please make sure that the custom via cell masters are available in the library and then reload the design"
  ```
- The Layer function table contains the list of all routing and cut layers as defined in the technology database. Material type is checked for all routing and cut layers. All the layers defined in `validVias` and `validLayers` in `LEFDefaultRouteSpec` and `LEFSpecialRouteSpec` are included in the layer function table in the right sequence.
- The OpenAccess reader (and data-checker) specifically looks at `minSpacing` consistency in all the constraint groups used in the design. The `minSpacing` constraint values should be equal or greater than the values specified in foundry constraint groups.
- All the layers in the `LEFDefaultRouteSpec` should have consistent `ANTENNA` rules. Various values that should be defined for a layer are:
  - `ACCURRENTDENSITY`
  - `DCCURRENTDENSITY`
  - `ANTENNAMODEL`
  - `ANTENNAAREARATIO`
  - `ANTENNADIFFAREARATIO`
  - `ANTENNACUMAREARATIO`
  - `ANTENNACUMDIFFAREARATIO`
  - `ANTENNAAREAFACTOR`
  - `ANTENNACUMROUTINGPLUSCUT`
  - `ANTENNAFATEPLUSDIFF`
  - `ANTENNAAREAMINUSDIFF`
  - `ANTENNAAREADIFFREDUCEPWL`

  All these listed values should be equal for all the layers and cuts with respect to the foundry constraint group.
- The Virtuoso ASCII techFile should have the cutClass specification in the viaSpacing rule for a CUT layer.
- For advanced node data, spacing table should be defined in the OpenAccess database if cut classes are defined in the techFile.

The *Technology DB Checker* checks for all the occurrences of above situations and reports probable problems when the current view is taken to Innovus and brought back.

20.3.2 Library DB Checker

Checks for the correctness and completeness of the specified lib/cell/view as the reference library.

20.3.2.1 Compatible View for Quick Abstract Inference

Checks whether the specified lib(s)/cell(s)/view(s) can be used for reading the instance abstracts inside Innovus.

A proper dual view has `prBoundary` and pin shapes for all pins in the layout view. If `prBoundary` is not present, the quick abstract inference capability in Innovus will not populate the `SIZE` statement of the abstract, which might not give any meaningful shape to the instance. The shape of the instance is computed based on its geometry's bounding box. Similarly, if all the pin shapes are not present, the quick abstract inference capability in Innovus will assume no metal layer is assigned to that pin and the router will give an error when you try to route such nets using automatic routers, such as NanoRoute. A warning message is displayed for such cases while importing a design into Innovus. The checker checks whether or not a particular view is a valid dual view, which contains `prBoundary` and all pin shapes corresponding to its logical Verilog view.

As part of this check , if a pin is outside the `prBoundary`, the checker displays a warning. This is because, though legal, it is not advisable to have pins outside the `prBoundary`. A pin outside the boundary is often an indication that the `prBoundary` has not been updated to enclose wiring.

20.3.2.2 Symmetry Attribute on the Cellview

Checks for the existence of the `SYMMETRY` property on a given list of cellview(s).

For quick abstract inference and other applications within Innovus to work, there must be a `SYMMETRY` statement in any given lib/cell/view that is being used as the IP in a design. You would provide a list of libraries and a view name. The checker runs on all the cells inside those libraries and the given view name and reports whether or not each one has the `SYMMETRY` property set.

20.3.3 Check Pins between Two Cellviews for the Remaster Instance

Checks whether the pins and ports match between two given cellviews.

In the netlist-driven mixed-signal flow, the blackbox declaration of the AMS block is done and then you are allowed to re-bind the instance with the layout view during iterative stages of floorplanning and analysis flow. Innovus either crashes or shows inconsistent behavior if the number of terms in the abstract view of the blackbox does not match with that in the layout view. You need to make sure that these two views have exactly the same name and number of terms.

During remastering (if being done in Virtuoso), the tool gives a warning or error message for these kinds of mismatches.

20.3.4 Report File Name

Provides the report in text format. The PASS/FAIL status for each check is printed clearly at the end of the report in the final summary.

Here's a sample Innovus Interoperability Library Checker report:

```
################################################################################
#### Technology Lib Name: FEOAreflib8
#### @(#)$CDS: virtuoso version 6.1.7-64b 09/21/2016 22:40 (sjfhw307) $
#### OA DB Checker Version: 16.20.003
#### OA DB Checker Path: /grid/cic/ciccm_t1nb_004/CICCM_BUILDS1/IC6.1.7/main/lnx86/64/160921-
291/tools.lnx86/dfII/etc/tools/innovus/oaDBChecker.il
#### Report Generated on: Sep 22 15:00:11 2016
################################################################################

INFO (CHECKER_LIB-33): Only default vias, routing layers, pitch, routing width, routing direction, layer offset, wire extension
constraints are read from the constraint group set by "set init_oa_default_rule" or defaults to LEFDefaultRouteSpec and rest of
rules are read from foundry constraint group.

Checking the right sequence of cut layers in technology....PASSED

Checking 'LEFDefaultRouteSpec' Constraint Group for valid layers, appropriate spacing values etc.....PASSED

Checking LEFSpecialRouteSpec Constraint Group....PASSED.

Checking 'minSpacing' Constraint Group for valid layers, appropriate spacing values etc.....
INFO (CHECKER_LIB-11): 'minSpacing' Constraint Group does not have validLayers.

Checking for PCELL vias in validVias list of NDRs....NOT FOUND
```

```
FAILED : Technology library 'FEOAreflib8' fails certain checks.

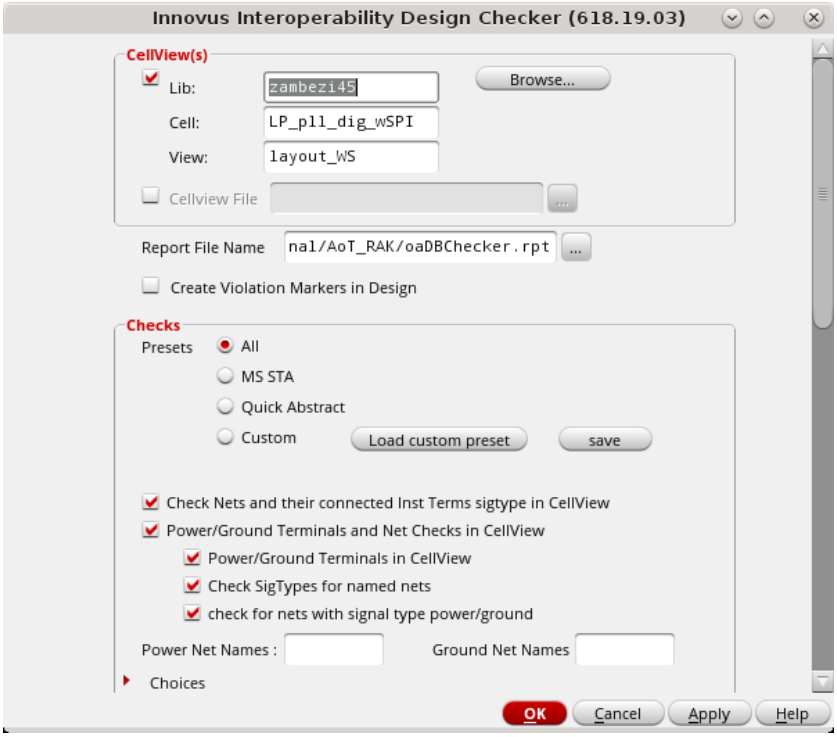

Final Summary
Type of checks PASSED FAILED
------------------------------------------------------------------------------

Tech Library:
LDRS Checks PASSED
LSRS Checks PASSED
Constraint Group Checks FAILED

Library Checks:
Compatible View Checks FAILED
Symmetry Attribute Checks FAILED
```

20.4 Check Design - Innovus Interoperability Design Checker

Select the *Innovus - Check Design* option from the Virtuoso menu bar to open the Innovus Interoperability Design Checker form.



The Innovus Interoperability Design Checker form checks for the correctness and completeness of the specified lib/cell/views as the design library. It contains the following sections:

- *CellView(s)*
- *Report File Name*
- *Checks*

20.4.1 CellView(s)

In this section, you specify the lib/cell/views that need to be checked for correctness and completeness as the design library. You can choose to specify either:

- A single lib/cell/view by specifying the appropriate values in the *Lib*, *Cell*, *View* fields in the *CellView(s)* section of the checker,
  Or
- Load a file containing a list of cell views by selecting the *Cellview File* check box and then specifying the file name in the adjacent field.
  If this method is used, the subsequent design checks are done on each view. With this method, the report has a separate header for each lib/cell/view followed by the checks made for that cellview.

20.4.2 Report File Name

Provides the report in text format. The PASS/FAIL status for each check is printed clearly at the end of the report in the final summary.

Here's a sample Innovus Interoperability Design Checker report:

```
################################################################################
#### @(#)$CDS: virtuoso version 6.1.8-64b 03/17/2022 12:01 (cpgsrv11) $
#### OA DB Checker Version: 618.19.03
#### OA DB Checker Path: /servers/inciccm_v31/ciccm_dev/IC6.1.8/ISR/lnx86/64/220317-
024/tools.lnx86/dfII/etc/tools/innovus/oaDBChecker.il
```

```
#### Report Generated on: Mar 30 17:49:57 2022
################################################################################

################################################################################
#### oaDBChecker Report
#### OA Design CellView: ether_DMS/top_1_port_receive/layout
#### Report Generated on: Mar 30 17:49:57 2022
################################################################################
Performing Check for existence of leading '|' char in instance names...PASSED


Performing checks for correct Bus information...PASSED


Performing Check for checking Sigtypes of Nets and their connected InstTerms...
ERROR (CHECKER_DESGN-59): (ether_DMS/top_1_port_receive/layout) contains a net 'VDDRX' of sigtype 'supply', but its isGlobal
attribute is not set to true. Make the net 'VDDRX' global by adding ! at the end of the net, otherwise there will be a problem
while saving the flattened view during saveDesign in Innovus. In some cases, it is required to override any logical
tiehigh/tielow connection with the physical implementation in the design. If the OpenAccess database is only used for analysis,
this message can be ignored. However, if the OpenAccess database is to be saved, there may be connectivity issues in the P/G
network. For more information, please refer to the oaDbChecker document in the mixed signal interoperability guide.
…
...FAILED


Performing Check for existence of PG terms in Netlist...
INFO (CHECKER_DESGN-27): Found terminal VDD with power sigType in the cellView.
INFO (CHECKER_DESGN-27): Found terminal GND with ground sigType in the cellView.
INFO (CHECKER_DESGN-27): Found terminal VDDRX with power sigType in the cellView.
INFO (CHECKER_DESGN-27): Found terminal GNDRX with ground sigType in the cellView.
...PASSED


Performing checks for SigTypes of named power and ground nets...
ERROR (CHECKER_DESGN-62): No power nets specified.
...FAILED

Performing Check for Power/Ground nets...
INFO (CHECKER_DESGN-98): Found net VDDRX with supply sigType in the cellView.
INFO (CHECKER_DESGN-98): Found net GNDRX with ground sigType in the cellView.
INFO (CHECKER_DESGN-98): Found net GND with ground sigType in the cellView.
INFO (CHECKER_DESGN-98): Found net VDD with supply sigType in the cellView.
...PASSED


Performing Check for existence of non-drawing shapes...PASSED


Performing Check for existence of non-drawing pin shapes...PASSED


Performing check for pins on non-routing layers in the design...PASSED


Performing check for presence of textDisplay objects in the design...PASSED


Performing Check for gapFill/fill/fillOPC purpose validity on routing layers...PASSED


Performing checks for presence of conic shapes in the design...PASSED


Performing check for status of interface bit for all blocks in the current view...PASSED


Performing checks for the incompatible wires/wire segments...PASSED


Performing Check for completeness of all Constraint Groups (NDRs) in the design...PASSED
```

```
Performing check for presence of pcell cache...PASSED

Performing Checks for XL compliancy...
WARNING (CHECKER_DESGN-39): Terminal 'BLW_DIS_2p5' found outside the bounding box of the design.
WARNING (CHECKER_DESGN-39): Terminal 'AEQ_DIS_2p5' found outside the bounding box of the design.
WARNING (CHECKER_DESGN-39): Terminal 'MR_AEQ_BYPASS' found outside the bounding box of the design.
...PASSED

Performing check for mustJoin terminals...PASSED

Performing checks for mismatch in instance instTerms and its master terminals...PASSED

Performing checks for trim metal shapes present in design...PASSED

Performing checks for pin shapes for terminals in the design...PASSED

Performing check for net/pin names with Verilog reserved keywords...PASSED

Performing check for net/pin names with System Verilog reserved keywords...PASSED

Performing checks for presense of symbolic wiring in PG nets in design...
WARNING (CHECKER_DESGN-102): Design has 'supply' net with net name VDDRX which does have symbolic wiring.
WARNING (CHECKER_DESGN-102): Design has 'supply' net with net name VDDRX which does have symbolic wiring.
WARNING (CHECKER_DESGN-102): Design has 'supply' net with net name VDDRX which does have symbolic wiring.
…
...PASSED

Perform checks for routing status of the design...PASSED

Performing check for pin placement status...PASSED

Performing check for existence of PRBoundary...PASSED

Performing check for multiple instantiation of same cell with different cellview...PASSED

Performing check for WSP interoperability...PASSED

Performing check for unbound instances...PASSED

Performing checks for placement status of hard macros...PASSED


Final Summary
Type of checks PASSED FAILED
------------------------------------------------------------------------------

Design Library Checks:
'|' char in instance names PASSED
Bus Annotation check PASSED
Check Nets and their Inst terms' sig type FAILED
Power/Ground Checks PASSED
Check signal types of named nets FAILED
Nets with signal type power or ground PASSED
```

```
Shapes on drawing purpose PASSED
Pins on drawing purpose PASSED
Pins on non-routing layer check PASSED
Presence of textDisPlay object check PASSED
Validity of gapFill/fill/fillOPC PASSED
Presence of conic shape check PASSED
Status of interface bit check PASSED
Unsupported routing shapes PASSED
Non default rules check PASSED
Show non default rules PASSED
MS constraints check PASSED
PCell cache check PASSED
XL Compliancy check PASSED
check for mustJoin terminals PASSED
Absence of instTerms in its master terminals PASSED
Existence of trim metal layers PASSED
Pin shape check for terminals PASSED
Nets or pins with same name as verilog reserved keywords PASSED
Nets or pins with same name as system verilog reserved keywords PASSED
Check PG nets with symbolic wiring PASSED
Design route status check PASSED
Pin placement status check PASSED
PR Boundary check PASSED
Instances inconsistency check PASSED
Check WSP interoperability PASSED
Check for unbound instances PASSED
Placement status check PASSED
```

20.4.2.1 Create Violation Markers in Design

Select the *Create Violation Markers in Design* check box if you want the OA DB Checker to create violation markers that can be displayed in the Annotation Browser window in Virtuoso. This makes it easier to view and debug violations as examining the log file for violations can be time consuming.

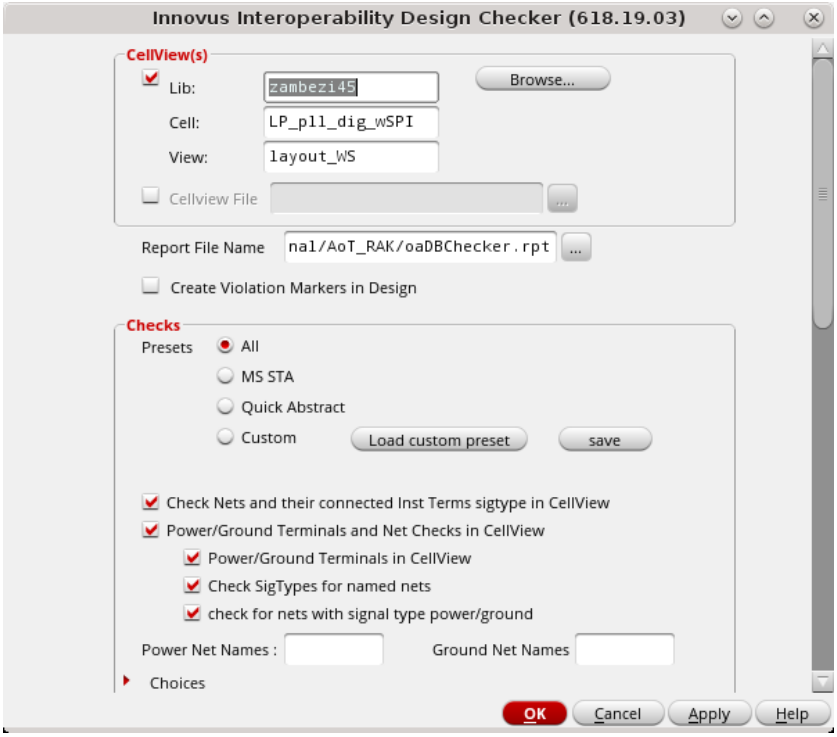
The marker creation feature is available in the latest Virtuoso ICADVM181 ISR/IC618 ISR release and in Innovus 20.1. This feature needs the INVS30 license.

For more information, see Viewing OA DB Checker Violation Markers in the Annotation Browser in Virtuoso.

Note: The marker creation feature is available only if you load OA DB Checker by using the plug-in method. It will not be available if you load OA DB Checker manually.

20.4.3 Checks

This section provides various options for checking the design library. The checks are categorized by possible mixed-signal flows. You can select a suitable check category from the *Presets* subsection. You can view list of available checks by expanding the *Choices* subsection.



20.4.3.1 Presets

Select one of the following check categories:

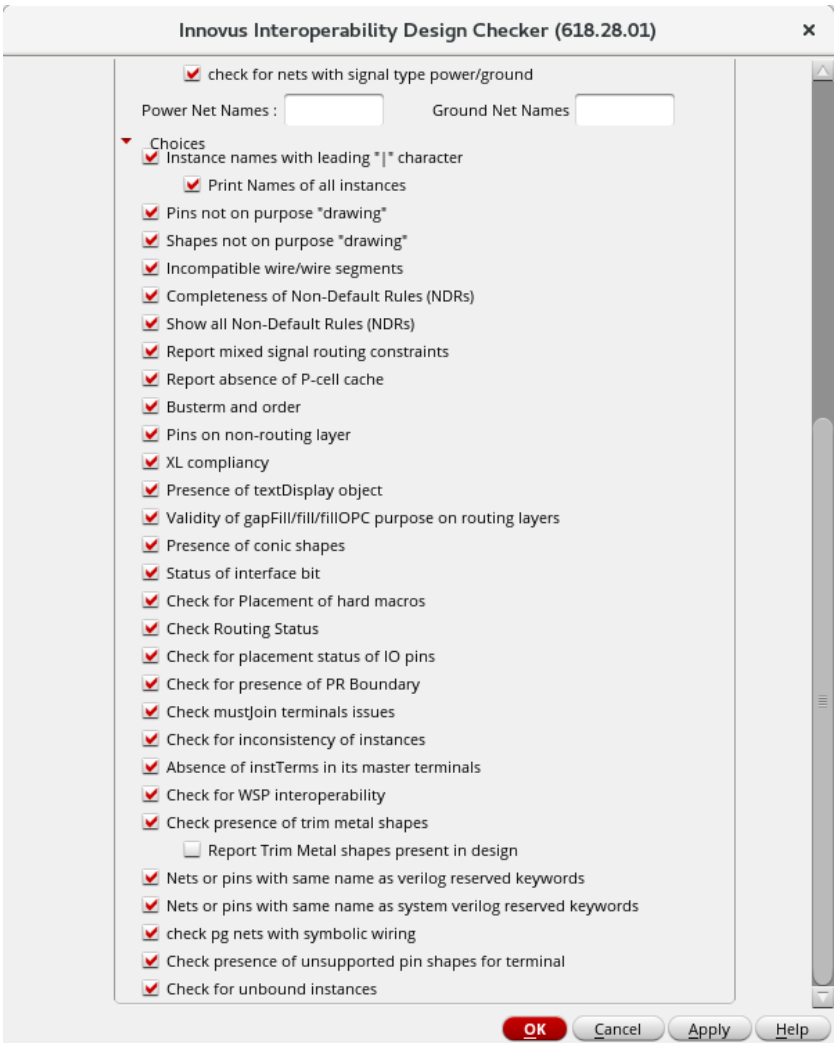
- *All* - This is the default setting. All possible design checks, listed under Choices, are carried out with this setting.
- *MS STA* - With this setting, the checks mandatory for static timing analysis of mixed-signal designs are selected by default. You can select additional checks from the *Choices* subsection.
- *Quick Abstract* - With this setting, the checks mandatory for quick abstract inference are selected by default. You can select additional checks from the *Choices* subsection.
- *Custom* - With this setting, you can select a combination of checks as per your requirements. Click *save* to save your custom settings. You can load previously saved settings using the *Load custom preset* button.

For each of the above preset categories, the following checks are turned on by default:

Check	Description
Check Nets and their connected inst Terms sigtype in CellView	Checks for signal type mismatches between all nets and the connected inst terms for a given cell view. For example, if a signal net is connected to a pin of a block but that pin is declared as a P/G pin, the checker will flag it as an error. Similarly, if a P/G net is connected to a non-P/G pin of a block, the checker flags it as an error.
Power/Ground Terminals and Net Checks in CellView	Checks the power and ground connections and nets in the OpenAccess library/cell/view. It includes the following sub-checks: • *Power/Ground Terminals in CellView* Detects the power and ground connections existing in the netlist data (EMH) of the OpenAccess library/cell/view. • *Check SigTypes for named nets* Checks signal types for the specified nets in the design. You can specify the power and ground nets in the *Power Net Names* and *Ground Net Name* fields, respectively. • *Check for nets with signal type power/ground* Reports all nets that are defined as either power or ground in the database.

20.4.3.2 Choices

Expand the *Choices* subsection to view the list of available checks.



The table below lists the available checks and their default setting for the preset categories *MS STA*, *Quick Abstract*, and *Custom*. Note that all the checks below are turned on by default for the *All* preset category.

Check	Description	MS STA	Quick Abstract	Custom
Instance names with the leading "|" character	Checks for the existence of instance names with the leading "|" character. Virtuoso-XL creates a leading pipe character in instance names when connectivity-driven layout generation process is followed using Configure Physical Hierarchy (CPH) and Generate Physical Hierarchy (GPH). This leads to an instance name mismatch given in the `.sdc` file. The checker finds all such cases. You can use `envSetVal("layoutXL" "prefixLayoutInstNamesWithPipe" 'boolean nil)` before running the schematic-driven layout generation process in Virtuoso-XL to get rid of occurrences of the extra leading character in instance names.	Off	Off	Off

Pins not on purpose "drawing"	Checks for the existence of pin shapes that are not on purpose drawing. Earlier versions of Innovus could not understand the pin layer and could not create physical shapes if pin shapes were not on the drawing purpose. In the current version of Innovus, any purpose other than drawing is converted to drawing in the round trip of OpenAccess data. There is no way to restrict the conversion within Innovus to put the pin back on the original layer purpose or drawing purpose. If the pin purpose is converted to drawing, we need a way to preserve the pin purpose for the top level. To retain pin purpose, use: `setOaxMode -pinPurpose {true | false}` in Innovus.	On	On	On
Shapes not on purpose "drawing"	Checks for the existence of shapes on non-routing layers in the design or on routing layers but on purpose other than "drawing". If you select the *Shapes not on purpose "drawing"* check box, the checker flags any blockages found on non-routing layers. In such cases, you must fix the tech and reload the design.	Off	Off	Off
Incompatible wire/wire segments	Checks for wire or wire segments that are not compatible with Innovus. Following types of routing shapes are allowed in Innovus: • Only manhattan routing is allowed for signal nets and they can have any of the following end extension: half-extend, zero-extend, or LEF wire extension value. • Manhattan and 45-degree routing is allowed for special nets. Custom end extension value is supported for manhattan while default style is supported for 45-degree special wires. • Only 45-degree and 90-degree shapes are supported in Innovus. If a wire or wire segment created in Virtuoso is not adhering to the half or zero extents rule, those nets when brought into Innovus are automatically changed to DEF SPECIALNETS wiring. This change can affect the physical design flow in Innovus. If you select the *Incompatible wire/wire segments* check box, the checker looks for any wires or wire segments in the design that do not adhere to the half or zero extents rule and flags these as incompatible with Innovus.	On	Off	Off
Completeness of Non-Default Rules (NDRs)	A Non-Default Rule (NDR) is a constraint group defined in OpenAccess design and/or technology database and associated to a net. From the Innovus perspective, a valid NDR contains definitions of the following items/rules: • `validLayers` The list of the number of metal layers might not be the same as that defined in `LEFDefaultRouteSpec`. However, it should be in a sequence and start from the bottom-most routing layer (M1). • `validVias` (Optional) • `width` (Optional) • `spacing` (Optional) • `minNumCuts` for each via layer (Optional) If the list of metal layers is lesser than that defined in `LEFDefaultRouteSpec`, Innovus reads and automatically completes the list for its in-memory reference. If an NDR is defined in a hierarchical fashion and refers to the other constraint groups defined in the same design library, or the referred technology libraries up to the base technology in the technology definition tree, Innovus reads the complete tree structure by iterating all the referenced libraries and constraint groups. It creates an equivalent NDR that contains the five rules--`validLayers`, `validVias`, `width`, `spacing`, and `minNumCuts`--listed above. If there are any conflicting values found during hierarchy traversal, Innovus uses the technology hierarchy to resolve the conflict by using the values defined in the upper hierarchy as against the values defined in the lower constraint groups. The constraint	Off	Off	Off

	groups are traversed as per their definition list including branches of the definition tree. The existence of the `validLayers` rule is mandatory. For the other rules, Innovus first goes to `LEFDefaultRouteSpec` and then to foundry, if not defined in NDR itself. So the checker will have to do the same thing as done by Innovus. The user runs the checker in Virtuoso Environment on a design and technology database to make sure that the constraint defined on the net is a valid NDR definition and the NDR can be used as a valid routing rule within Innovus. If you select *Completeness of Non-Default Rules (NDRs)*, SKILL iterates all the constraint groups and checks for its validity as an NDR. If the constraint is referring to other constraints (hierarchical constraint definition), the SKILL code is expected to traverse the complete list of constraints in the reference tree to fetch and complete the list of all variables. It would write the final NDR with values for the five rules--`validLayers`, `validVias`, `width`, `spacing`, and `minNumCuts`--in the text report file to show you the final flattened NDR. This is important from an Innovus user's perspective because it would give the flattened view of the NDR after hierarchy flattening, which Innovus does by default. When the *Completeness of Non-Default Rules (NDRs)* option is selected: • The checker iterates all constraints and checks the NDRs, and reports it as it would be visible in Innovus. • The checker flags NDRs that are not compatible with Innovus. An NDR is valid in Innovus if the routing layers used by the NDR match those of the LDRS used in the design. If the layers do not match, the checker issues a warning that the NDR is not compatible with Innovus and that any net having the NDR as a constraint group will be converted into a special net in Innovus. • If the number of layers defined in the NDR is lesser than that in `LEFDefaultRouteSpec`, Innovus completes the NDR definition for database completeness but directs NanoRoute to use only the layers specified in the NDR. For example, if the NDR has `validLayer` (M1 M2 M3) and `LEFDefaultRouteSpec` contains M1 to M5, the constraint using the specific NDR would be routed using only M1 to M3 by NanoRoute. This is displayed as a message that the net Nx on which you have associated a specific NDR contains only M1 to M3 as `validLayers` and NanoRoute would use only M1 to M3 for completing the routing of this net even though technology allows M1 to M5. • If a constraint on a net contains only `minNumCuts` or `minWidth/spacing` rules without having `validLayer`, the checker displays a message that constraint is not valid and you need to add validLayers to make it a valid NDR. • If there is any hierarchical NDR (NDRs referring to other NDRs), the checker is expected to display the following warning message: `%s is a hierarchical constraint. Innovus will create a flattened constraint in non-update mode. To preserve the constraint as-is and avoid Innovus from creating an equivalent flattened constraint, use setOaxMode -updateMode true within Innovus sessions.` • If an NDR name contains any special character like " " (blank space) in their names, the checker reports these names.			
Shows all Non-Default Rules (NDRs)	Reports all NDRs in the design.	Off	Off	Off
Report mixed signal routing constraints	Checks for the existence of valid mixed-signal routing constraint in the OpenAccess database. In the flow, OpenAccess database might have been available from Virtuoso or Innovus. Therefore, interoperability of OpenAccess database using Innovus might lose some of the objects that are not supposed to be interoperated. The OpenAccess DB checker checks for the completeness of a constraint that is going to be used by all the tools	Off	Off	Off

including Innovus.

Note: If multiple constraints, such as differential pair, match length, and bus, exist on same net, the checker generates an error stating that they cannot be applied on same net. The names of the constraints and the net or net group on which the conflict is occurring are reported in the error message.

Differential Pair
Iterates on all `diffPair` groups and identifies the number of such constraints that exist in an OpenAccess database.

Following items exist in a `diffPair` constraint:

- If a `diffPairGroup` has NDR rule3, `net1` has NDR rule1 and `net2` has NDR rule2 associated, then
- If rule1 is not equal to rule2, the checker displays an error message stating that rule1 and rule2 should be same.
- If rule3 is not present, rule1 == rule2 and is promoted to become rule3 on `diffPairGroup`.
- If all rules are present, rule3 takes precedence. For this `diffPairGroup`, `net1` has rule1 and `net2` has rule2 (which is same as rule1) but rule3 will be used for this `diffPairGroup`. All the rules--rule1, rule2, and rule3--if present must be same.
- All the valid layers, valid vias, width, spacing, and `minNumCut` definitions should come from a valid NDR for the `diffPair` constraint.
- A reflexive rule has `minSpacing` for each layer and tolerance defined in each `diffPair` group. If not, the values are picked from `LEFDefaultRouteSpec` and then `foundry`.

  If tolerance values are not present, the following warning message is displayed.
  `"Tolerance value for diffPair %s is not specified. Default of 20% will be used."`

  When reflexive rules are not found on `diffPair`, the following warning message is displayed.
  `"Within group rules were not specified. Innovus will use NDR or LDRS or foundry for within group spacing rules"`
- If trans-reflexive rule is present, the following warning message is displayed:
  `"Diffpair %s has a outside group rule attached which is not understood by Innovus, hence this constraint will not be preserved by Innovus during roundtrip of OA data in non-update mode. To preserve such rules, it is recommended to use setOaxMode -updateMode true in Innovus sessions. For diffPair group to other nets spacing, Innovus will use NDR or foundry rules".`
- In a mixed-signal constraint, the following three can have constraints/NDRs associated with it: Design Constraint Group (DCG), Group to Outside (Trans-Reflexive), and Within Group(Reflexive) rules.
  - If a constraint/NDR is associated with DCG, it takes precedence over others. You can specify values to DCG directly or can associate an NDR/constraint.
  - If DCG is empty and tran-reflexive and reflexive rules are present, the group to outside values come from the trans-reflexive rule and the within group rule comes from the reflexive rule.
  - If DCG is empty, trans-reflexive is empty, and reflexive rule is present, the Within group rules come from the reflexive rule, the group to outside rule comes from the `LEFDefaultRouteSpec/foundry` search path, and tolerance gets the default value.

- If all three are empty, default values are used plus the `LEFDefaultRouteSpec/foundry` search path is used for other rule values.

Existence of trans-reflexive rule displays a warning because Innovus cannot handle or use them. Innovus will interoperate the values in the update mode and lose them in the non-update mode.

Shielding
An oacShieldingConstraintGroupType contains the following values: oacMinSpacing (Shield Gap), oaxMinWidth (shield width), msOverHang, msShieldStyle, msConnectSupply and msConnectSupplyDistance.

- No override constraint group required in DCG. NDR is not mandatory. So, no message is needed from checker.
- If any of the following: `msOverHang`, `msShieldStyle`, `msConnectSupply` or `msConnectSupplyDistance` are not present, an error message is displayed and you are prompted for these values.
- Shielding constraint can be applied on `oaNet` and `oaNetGroup`. Warning situations are:
- When some nets of the group have shielding constraint and some are empty.
- When all nets do not have same shielding constraint.

Diff Pair with Shield

- Referring to the `diffPair` example above, `oaShieldNet1` and `oaShieldNet2` on boths nets, `net1` and `net2`, must be same. If not, an error message is displayed.
- Both the nets - `net1` and `net2` should point to the same shielding net constraint group. If not, an error message is displayed.
- The checker checks for the existence of shield attributes on the nets contained in the `diffPairGroup` to determine and display a message to the user about the values that would be used for shielding the nets.

Match Nets
Checker iterates on all matchedNetGroup and checks for the following:

- Nets `net1` through `netN` are either empty or have identical content in NDRs.
- Reflexive and trans-reflexive requirements are same as `diffPair` above.
- All the valid layers, valid vias, width, spacing, and `minNumCut` definitions come from a valid NDR for the `matchLength` constraint.

OpenAccess Net Group
Checker iterates on all oacNetGroupPurposeType and checks for the following:

- If an NDR is present on the group and the group members, all nets should have same NDR. Checker displays an error message if any NDR is different.
- Reflexive and trans-reflexive requirements same as `diffPair` above.

BUS Group
This combines N nets into a standard oaBusNet. The checker checks and reports any inconsistency in the construction of the oaBusNet.

- If DCG contains an NDR and group members either do not have any NDR or have same NDR, all the nets would use the same NDR rules.
- NDRs, if present, should be same on all group members as well as the DCG of the BUS group. If all the NDRs are not identical, it displays an error message with the net names that have different NDRs.

<table>
<tr>
<td>Report absence of P-cell cache</td>
<td>Checks for missing P-cell cache.

When a design is brought into Innovus, you might find WARNING and ERROR messages related to missing P-cell information. You then need to return to the Virtuoso environment, dump the P-cell cache, and then reload the design in Innovus. To prevent this situation, select the Report absence of P-cell cache check box in the checker. The checker then warns you about the missing P-cell cache so that you can dump the cache before loading the design in Innovus.

Note that if there are no P-cells in the design, the checker reports PASS as given in (1) in the table below. It starts checking for the cache only if there are P-cells in the design. In such a case, sub-checks are reported as given in (2) and (3).
<table>
<tr><th>#</th><th>Scenario</th><th>Errors/Warnings reported by OA DB checker</th></tr>
<tr><td>1</td><td>No PCells

(PCell variables could be set or unset)</td><td>Reports check as <code>Passed</code>.</td></tr>
<tr><td>2</td><td>PCells present + PCell variables are not set</td><td><code>ERROR : Design has pcell instances. set environment variable CDS_ENABLE_EXP_PCELL true before loading design into Innovus
ERROR : Design has pcell instances. set environment variable CDS_EXP_PCELL_DIR before loading into Innovus</code></td></tr>
<tr><td>3</td><td>PCells present + Cache does not exist in the PWD + CDS_EXP_PCELL_DIR not set</td><td><code>ERROR : Design has pcell instances. set environment variable CDS_EXP_PCELL_DIR before loading into Innovus
ERROR : PCell cache ./.expressPcells does not exists. Please dump the cache before loading design in Innovus.</code></td></tr>
</table>
</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>Busterm and order</td>
<td>Checks for bus terminals with missing ordering information.

If you select the Busterm and order check box, the checker checks bus annotation and reports any bus terminals (busTerms) that do not have ordering (busOrder) information.</td>
<td>On</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>Pins on non-routing layer</td>
<td>Checks for the existence of the pins on non-routing layers in the design. This usually happens if you have used the layers outside of the <code>validLayers</code> in the <code>LEFDefaultRouteSpec</code>.

If you select the Pins on non-routing layer check box, the checker flags any pins found on non-routing layers. For example, if the <code>validLayers</code> in the <code>LEFDefaultRouteSpec</code> has M1 and M2 as routing layers and pins are found on non-routing layers, such as M1PN and M2PN, the checker flags these pins as follows:

<code>FAILED: Pins found in non-routing layers. Please fix the tech and reload the design.</code>

In such cases, you must fix the tech and reload the design.</td>
<td>On</td>
<td>On</td>
<td>On</td>
</tr>
</table>

XL compliancy	Performs *XL compliance* checks. If you are working on a design or part of a design that needs to be analyzed or modified in Innovus, Innovus needs to be able to extract the connectivity information from the design. Some older designs implemented in Virtuoso may not have the connectivity required by Innovus. In some cases, the top level layout that is brought into Innovus may have a PR boundary and some pin formation but not have full connectivity. In such cases, you can select this check box to perform *XL compliance* checks such as:
	• Reports the missing connectivity for the shapes/terminals of the block(s).
	• Reports shapes drawn manually without any nets attached to it
	If you select the *Check for XL compliancy* check box, the checker flags connectivity issues as follows:
	`Checking for XL compliancy….`
	`WARNING: Terminal abc  does not have any net connection.`
	`WARNING: Terminal def does not have any net connection.`
	`..`
	`WARNING: SHAPE <bbox1> found without any net connection.`
	`WARNING: SHAPE <bbox2> found without any net connection.`
	`..`
	`WARNING: Terminal cde found outside design boundary`
	`..`
		Off	Off	Off

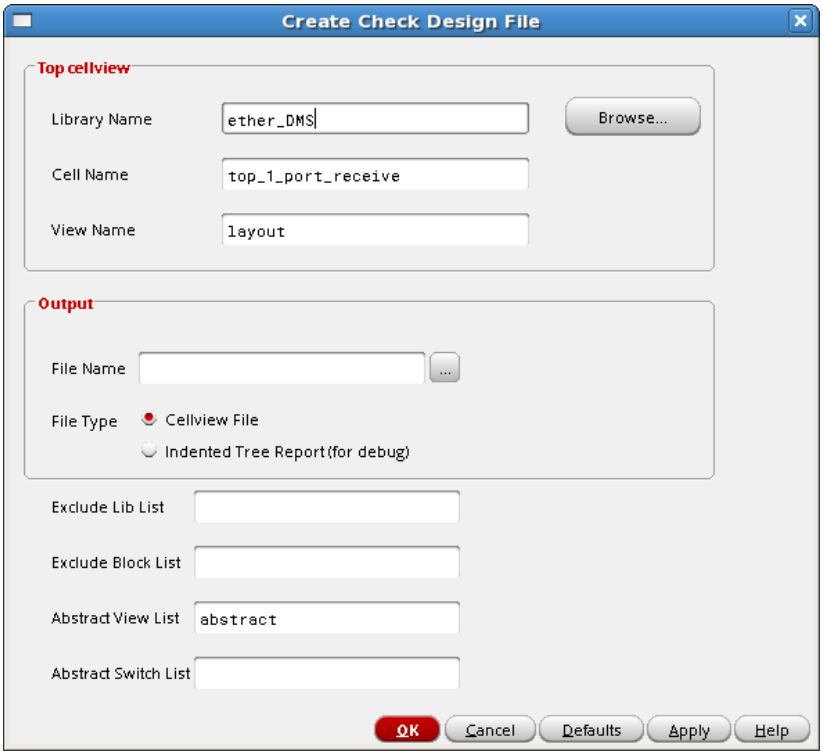
Presence of textDisplay object	Checks for the presence of oaTextDisplay objects that are not connected to pins in the design. If you select this option, the checker throws an error mentioning that textDisplay is not supported in Innovus and therefore will not be round-tripped. textDisplay connected to pins are round-tripped correctly and are therefore not reported in this check.	Off	Off	Off
Validity of gapFill/fill/fillOPC purpose on routing layers	Checks for the validity of gapFill, fill, and fillOPC purpose on routing layers. Only data that is valid can be selected or turned off when viewing the cellview in the Virtuoso window. If the gapFill, fill, fillOPC purpose on any routing layer is not valid, the checker displays a warning such as follows: `WARNING : 'gapFill' purpose on routing layer 'M1' is not valid.` In such cases, you would need to set the layer/purpose pair to valid before you can select or turn off visibility of the data.	Off	Off	Off
Presence of conic shapes	Checks for the presence of conic shapes in the design. If you select this option and a conic shape is present in the design, the checker throws an error mentioning that conic shapes will not be read into Innovus and will not be round tripped.	Off	Off	Off
Status of interface bit	Checks for status of interface bit.	On	On	On
Check for Placement of hard macros	Reports hard macros without `FIXED` or `COVER` status. Typically, the hard macros in an AoT design have the placement status as `NONE` or `PLACED`. However, while running the ECO or optimization flow in Innovus, the `ecoPlace` command requires all the hard macros to have the `FIXED` or `COVER` status. If you enable Design Library checks and select the Check for Placement of hard macros check box, the checker flags hard macros that do not have placement status as `FIXED` or `COVER`. You can then change the status of the reported macros to `FIXED` by using a script before loading the design in Innovus.	Off	Off	Off

Check Routing Status	Checks whether or not Innovus will see the design as routed. If a block is routed manually in Virtuoso, it can be VXL clean but still be seen as unrouted when loaded in Innovus because it does not have any OpenAccess routes in the DB. In such a case, when you try to run extraction, Innovus errors out with the following message: `Design must be routed before running ""` If you enable Design Library checks and select the new *Check Routing Status* check box, the checker flags any blocks that Innovus can view as unrouted. In such cases, you may need to update the design status manually before loading the design in Innovus.	On	Off	Off
Check for placement status of IO pins	Checks whether the placement status of IO pins is set to `NONE`. I/O pins that have the placement status set to `NONE` in Virtuoso are translated to `UNPLACED` in Innovus. Although such pin shape will still be visible in the GUI, NanoRoute will not route them. Similarly, the `verify_drc` command in Innovus will not include such pins in DRC checking. If the *Check for placement status of IO pins* check is enabled, the OA DB Checker provides a warning on encountering an I/O pin with placement status set to `NONE`.	Off	Off	Off
Check for presence of PR Boundary	Checks for the presence of the place & route boundary (PR Boundary). This check may not be required during static timing analysis (STA) checks. If the *MS STA* preset category is selected, the *Check for presence of PR Boundary* option is turned off by default.	Off	Off	Off
Check mustjoin terminal issues	Checks and reports the MUSTJOIN attribute on the PG terminals of a cell. PG terminals with the MUSTJOIN attribute lead to issues while assembling the design in Innovus.	Off	Off	Off
Check for inconsistency of instances	Reports instances that have the same cell name but different library names.	Off	Off	Off
Absence of instTerms in its master terminals	Checks and reports mismatches between instTerms and master terminals. Such mismatches can occur if connectivity extraction was not run in Virtuoso after remastering instances. By enabling this check in the OA DB Checker, you can fix any reported mismatches within Virtuoso Layout Editor before importing or assembling the design in Innovus.	Off	Off	Off
Check for WSP interoperability	Checks the design for Width Spacing Pattern (WSP) regions. Designs with WSP regions cannot be read into Innovus so enabling this check provides an early warning.	Off	Off	Off
Check presence of trim metal shapes	Checks for the presence of trim metal shapes. Presence of trim metal shapes may cause interoperability issues if they are not aligned to tech LEF rules.	Off	Off	Off
Nets or pins with same name as verilog reserved keywords	Checks whether the design has any pins or nets with names that clash with Verilog reserved keywords. By modifying the names of such pins and nets, you can avoid errors later on while saving the design in Innovus.	Off	Off	Off
Nets or pins with same name as system verilog reserved keywords	Reports pins and nets using systemVerilog reserved names. Correct the reported issues before opening the design in Innovus.	Off	Off	Off
Check pg nets with symbolic wiring	Checks for PG nests that use symbolic wiring.	Off	Off	Off
Check presence of unsupported pin shapes for terminal	Reports any unsupported pin shapes for a terminal.	Off	Off	Off

Check for unbound instances	Reports unbound instances in the design.	Off	Off	Off

20.5 Create Check File - Create Check Design File

Select the *Innovus - Create Check File* option from the Virtuoso menu bar to open the Create Check Design File form.



Use the Create Check Design File form to generate a file containing the lib/cell/views from your design. You can choose to extract lib/cell/views hierarchically. The output file can then be used to run the OA DB Checker.

The Create Check Design File form contains the following fields:

Field		Description
Top cellview		
	Library Name	Specifies the library name for the top cell of your design.
	Cell Name	Specifies the cell name for the top cell of your design.
	View Name	Specifies the view name for the top cell of your design.
Output		
	File Name	Specifies the name and location of the output file.
	File Type	Specifies the type of file to be output. Choose from one of the following options: • C*ellview File* - This is the default file type. It generates a file contains a list of cell views for the OA DB Checker run. • *Indented Tree Report (for debug)* - Use this option for debugging.
Exclude Lib List		Specifies the list of libraries to be excluded from the cellview list input to the OA DB Checker.

Exclude Block List	Specifies the list of blocks to be excluded from the cellview list input to the OA DB Checker.
Abstract View List	Specifies the list of abstract views to be input to the OA DB Checker.
Abstract Switch List	Specifies the abstract switch list.

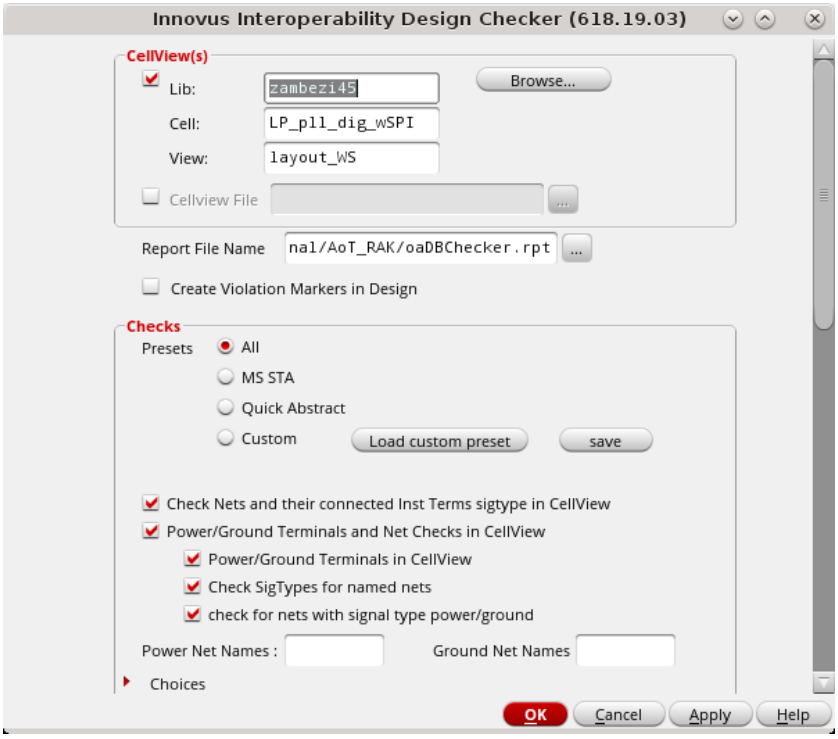
20.6 Viewing OA DB Checker Violation Markers in the Annotation Browser in Virtuoso

The OA DB Checker logs all violations in the checker log files. However, examining the log file for violations can be time consuming. In this release, the OA DB Checker has been enhanced to create violation markers, which can be displayed using the Annotation Browser window in Virtuoso.

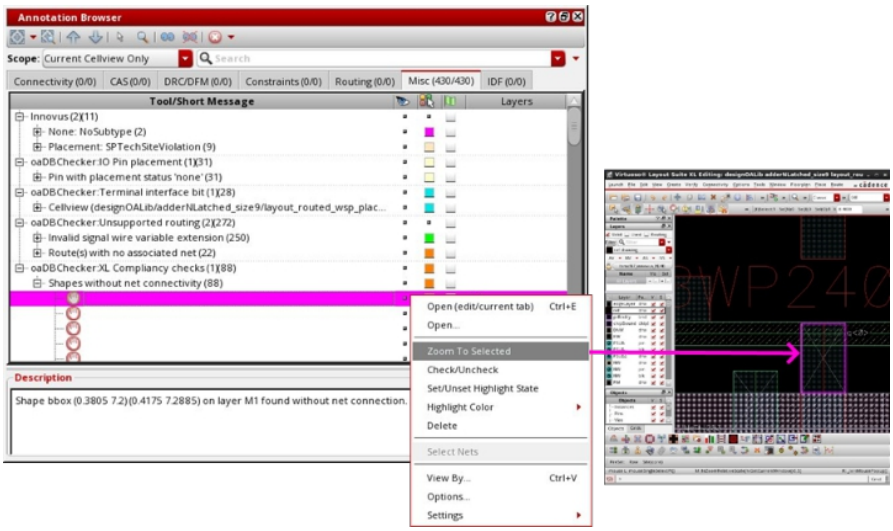
The marker creation feature is available in the latest Virtuoso ICADVM181 ISR/IC618 ISR release and in Innovus 20.1.

To view OA DB Checker violation markers in Annotation Browser, follow the steps given below:

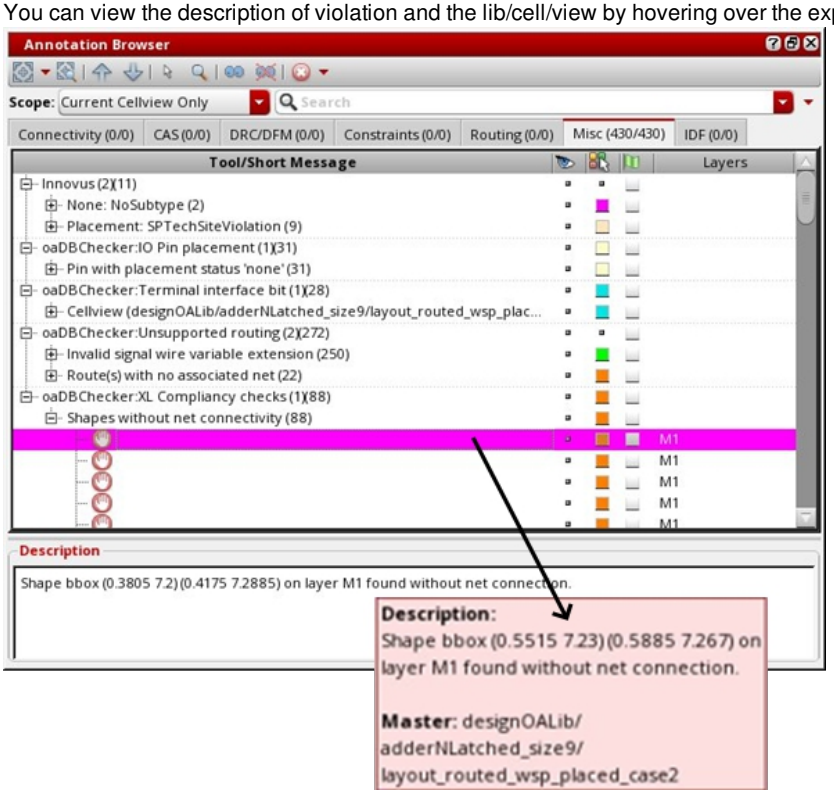
1. Load OA DB Checker by using the plug-in method and select *Check Design* from the *Innovus* menu in Virtuoso.
   Note: The marker feature will not be available if you load OA DB Checker manually.
2. Select the *Create Violation Markers in Design* check box in the Innovus Interoperability Design Checker form.



   Note: The marker feature is available only in the Innovus Interoperability Design Checker form (*Innovus* -> Check Design) and not in the Innovus Interoperability Library Checker form (*Innovus -> Check Library*).
3. Specify other options, as required, and click *OK*. As you have selected the marker feature, the INVS30 license will be checked out.
4. The violations are listed automatically in the Annotation Browser window on the *Misc* page. Violations are listed by category, such as *Route(s) with no associated net* and *Shapes without net connectivity*. You can expand a category, select a specific violation, and zoom into its bounding box.



5. You can view the description of violation and the lib/cell/view by hovering over the expanded violation:



Note: As of this release, all marker violations are reported on the *Misc* page of the Annotation Browser window. If you ran any DRC check earlier on the current view, the OA DB Checker marker violations appear along with other DRC violations.

20.7 Run Innovus - Innovus Launch GUI

Select the *Innovus - Run Innovus* option from the Virtuoso menu bar to open the Innovus launch GUI form. The *Run Innovus* option is enabled only if Innovus Implementation System is in your installation path.



In the Innovus launch GUI form, you choose the mode in which you want to launch Innovus. To learn about these modes, see the Virtuoso Digital Implementation chapter.

20.8 OA DB Checker Use Case Scenarios for Virtuoso Users

The table below depicts various scenarios for Virtuoso users with *Innovus* plug-in installation:

Scenario	Check Library and Check Design	Create Check File	Run Innovus
Default (No environment variable present; no Innovus installation)	Loads `oaDBChecker.il` from the dfII hierarchy	Loads `userCreateCheckDesignFile.il` from the dfII hierarchy	Disabled
`OA_CHECKER_DIR` is set to the directory containing the latest `oaDBChecker.il` file; no Innovus installation	Loads `oaDBChecker.il` from the directory specified by `OA_CHECKER_DIR`: • If the `oaDBChecker.il` revision is earlier than 16.20.002, the `oaDBChecker.il` from the dfII hierarchy is loaded. • If `oaDBChecker.il` is not present in the path specified by the `OA_CHECKER_DIR` variable, the `oaDBChecker.il` file from the dfII hierarchy is loaded and a warning is displayed.	Loads `userCreateCheckDesignFile.il` from the dfII hierarchy	Disabled
`OA_CHECKER_DIR` is set to the directory containing the latest `oaDBChecker.il` file; Innovus installation path is present in the `PATH` environment variable	Loads `oaDBChecker.il` from the directory specified by `OA_CHECKER_DIR`: • If the `oaDBChecker.il` revision is earlier than 16.20.002, the `oaDBChecker.il` from the dfII hierarchy is loaded. • If `oaDBChecker.il` is not present in the path specified by the `OA_CHECKER_DIR` variable, the `oaDBChecker.il` file from the dfII hierarchy is loaded and a warning is displayed.	Loads `userCreateCheckDesignFile.il` from the dfII hierarchy	*Run Innovus* is enabled. Loads `virLaunchInnovusIC61x.il` from the Innovus installation path

21

oaZip Utility

- The oazip Utility to Compress/Decompress Databases
  - Command Syntax
  - Arguments

21.1 The oazip Utility to Compress/Decompress Databases

OpenAccess 22.42 and later releases support the ability to save the design databases in a library in a compressed form. Tools based on OpenAccess 22.41 or earlier releases (such as IC6.1.5), may need to have the designs in the library decompressed with the `oazip` utility before the designs can be accessed.

This utility provides the following functionality:

- It processes the OpenAccess databases in a library and compresses them. The compression control value of the library is updated.
- It processes the OpenAccess databases in a library and decompresses the ones that are in compressed form. The compression control value of the library is reset or updated.
- It provides the value of the compression control attribute of a library.
- It scans the OpenAccess databases in a library and reports the databases that do not match the compression control attribute of the library.
- It scans the OpenAccess databases in a library and updates any databases that did not match the compression control attribute of the library.

21.1.1 Command Syntax

To run `oazip`, enter the following:

```
oazip -lib library {-compress|-decompress|-check|-query|-update} [Optional Arguments]
```

21.1.2 Arguments

You can use the `-help` or `-h` argument to display command line help. The command line arguments are described in the table below.

Required Argument	
`-lib <name>`	This required argument specifies the name of the library to process. If this argument is not specified, an error message will be displayed.
One of the Following Arguments is Required	
`-compress`	If this option is specified, the utility will process the OpenAccess databases in a library, compress the ones in uncompressed form, and reset the compression control value of the library.
`-decompress`	If this option is specified, the utility will process the OpenAccess databases in a library, uncompress the ones in compressed form, and reset the compression control value of the library.
`-query`	If this option is specified, the utility will report whether the compression control is specified for the library and what level it is set to.
`-check`	If this option is specified, the utility will report the OpenAccess databases in the library that are inconsistent with the compression control setting of the library. If there is no compression control specified, the utility will list the databases that are in compressed form. If the compression control is specified, the utility will list those databases that are either in uncompressed form or were written using a compression level different than what the compression control is set to.
`-update`	If this option is specified, the utility will process the OpenAccess databases in a library and update the ones that are inconsistent with the compression control value of the library.
Optional Arguments	
`-h` or `-help`	Display usage information.

`-compressLevel <level>`	This option specifies the compression level to use for the library. Compression levels refer to the amount of effort the compression algorithm will use to when compressing data. Higher values do not necessarily correspond to better compression efficiency. Compression levels are specified by an integer value between `1` and `9`, inclusive. The default value of `1` is suitable for most applications.
`-logFile <file>`	Specifies the log filename. If this option is omitted, the log filename defaults to `oazip.log`.
`-noInfo <msgIds>`	Suppresses the specified INFO messages. msgIds is a quoted, space separated list of numbers. Each number in the list represents the numerical portion of the ID for the message you want to suppress. None of the numbers in the list may be zero. Suppressed messages do not appear on the terminal or in the log file.
`-noWarning <msgIds>`	Suppresses the specified WARNING messages. `msgIds` is a quoted, space separated list of numbers. Each number in the list represents the numerical portion of the ID for the message you want to suppress. None of the numbers in the list may be zero. Suppressed messages do not appear on the terminal or in the log file and are not included in the total of WARNING messages displayed in the summary.

`-templateFile<file>`	Specifies a file containing arguments to oazip. You can specify a template file instead of entering a string of arguments on the command line.
	
	If you specify a template file, arguments on the command line have precedence over arguments specified in the file. So, if the same argument exists in the template file and in the command line, the translator uses the argument on the command line.
	
	Specify arguments in a template file as follows:
	• Enter arguments in the template file without a dash (-) before the argument.
	• Enter each argument and value pair on a single line.
	• Separate the argument from its value using a space or a tab.
	• Designate comment lines with a `#` sign as the first character in the line.
	
	Sample Template File
	
	`# oazip` command line arguments:
	
	`lib libName`
	
	`logFile myoazip.log`
	
	`compress`

`-v`	Prints tool, format, and library version information.
`-version`	Prints tool and format version information.

22

Voltage Dependent Rule Interoperability

- Overview
- VDR Syntax

22.1 Overview

You can create Voltage Dependent Rules (VDRs) which can be associated with signal nets using the Common Power Format (CPF). The layer purpose per voltage is carried with the VDR rules itself. With this understanding, NanoRoute and WireEditor simply use the DRC rules coming from specified VDR and voltage values on the nets coming from CPF.

Innovus allows the LEF `VOLTAGESPACING` rule through a LEF 5.8 property (and an equivalent OpenAccess technology rule) to support voltage-dependent spacing rules.

Innovus does not directly support layer-purpose on each routing-shape in the Innovus database. Instead it depends on net voltage levels specified through a CPF file. All the shapes of a given net inherit the voltage for that net.

Loading a CPF file will assign voltage levels to power/ground nets and implicitly their associated signal nets belonging to the same power domain. If no voltage levels are loaded (no CPF), then the VDRs have no effect inside Innovus.

Note: The VDR file is required only with CPF. In the case of Unified Power Format (UPF), the voltage specs are part of the view definition files and so the VDR file is not required.

22.2 VDR Syntax

The following is syntax for VDR in the ASCII technology file:

```
techPurposes(

 ;( PurposeName Purpose# Abbreviation )
  ;( ----------- -------- ------------ )
   ( lv 100 lv 'parent "drawing" 'voltageRange ( 0.0 1.5 ) 'description "low-voltage" )
   ( hv 101 hv 'parent "drawing" 'voltageRange ( 1.5 3.3 ) 'description "high-voltage"
)
.
.
)

constraintGroups(

;( group [override] )
;( ----- ---------- )
( "foundry" nil

spacings(

( minSpacing ("Metal1","lv") 1.5 'ref "M1.S.LV" 'description "M1 LV spacing = 1.5" )
( minSpacing ("Metal1","hv") 3.3 'ref "M1.S.HV" 'description "M1 HV spacing = 3.3" )
.
.
)
```

The corresponding LEF syntax is:

```
LAYER Metal1

...

VOLTAGESPACING TOCUT ABOVE

     0       0.75

     1.5     0.95

     3.3     1.25 ;

VOLTAGESPACING
```

```
    0       0.7

    1.5     0.9

    3.3     1.2 ;

.....

END Metal1
```

You can distinguish between high voltage and low voltage wires by using different layer-purpose pairs (LPPs) for routing in Virtuoso.

In Innovus, the VDR is read from the OpenAccess database through its native constructs.

The applications that support VDRs within Innovus are NanoRoute and verifyGeometry. Basic VDR rules are part of the technology rules and are not modified within Innovus.

23

Useful Tips

- Removing Pipe Character from Instance Names
- Viewing PCells in Innovus
- Global Net Name Collision Resolution
- NanoRoute Support for Nets
- Saving Blackboxes in OpenAccess
- Importing the Power and Ground Nets Connections Using Verilog Netlist
- Turning Off Power and Ground Connections in a Netlist
    - verilogAnnotate Command Syntax
- Generating Power and Ground Pins
- Using Abstract Views in Innovus
- Performing Power Routing Outside Innovus
- Generating Abstracts with Antenna Information
- Mapping Virtuoso Bind Keys to Innovus Bind Keys
- SKILL to TCL Mapping
- Generating a Verilog Netlist for Innovus from a Virtuoso Schematic
    - Netlisting Options
    - Schematic and Hierarchy Editor Setup
    - Netlist Generation
    - Problems and Solutions
    - Filtering Power and Ground Nets in Verilog
    - Netlisting Options
    - Schematic Modifications

- Limitation due to Split Bus and Bundle with `Merge All'
- Creating a Non-Default Rule in Virtuoso and then Using it in Innovus
  - Using Virtuoso
  - Using Innovus
- Troubleshooting Common Errors in Innovus OpenAccess Flow
  - IMPOAX-717 Error
- Working with Old Versions of OpenAccess Data

23.1 Removing Pipe Character from Instance Names

Generate physical hierarchy prefixes the '|' pipe character to instance names in the generated layout. This change in instance names can cause interoperability issues in certain flows where tools are name specific.

For example, suppose you have an OA-based VDI flow where the top level schematic is created from a verilog netlist using `verilog2oa`. Now, when you create a layout from the schematic view, the layout netlist prefixes all instances with "|". This means if the instance names are `I0`, `I1`, `I2`, and so on in the schematic, they are changed to |I0, |I1, |I2 and so on in the generated layout.

To avoid such issues, set the following variable in the .cdsinit file before starting Virtuoso session:

```
envSetVal("layoutXL" "prefixLayoutInstNamesWithPipe" 'boolean nil)
```

When you do so, the layout generation flow from schematic using *Generate From Source* does not prefix the '|' pipe character to instance names in the layout database.

23.2 Viewing PCells in Innovus

To view parameterized cells (PCells) in Innovus, set the following variables on UNIX prompt before starting the Innovus session:

```
setenv CDS_ENABLE_EXP_PCELL TRUE

setenv CDS_EXP_PCELL_DIR <directory path>/.expressPcells
```

23.3 Global Net Name Collision Resolution

If the module `dtmf_chip_block` has a local net name `?VDD?` that collides with a global net name `?VDD?`, then a local instance terminal power-connection requires access to the global net that is blocked by the local net with the same name. Therefore, a new global net `?OAX_VDD_1?` will be created while reading a design inside Innovus, as an equivalent net to the `?VDD?` global net and will be used for these power-connections. The two nets are effectively the same nets.

To avoid this collision, do not use local net names that collide with global net names.

23.4 NanoRoute Support for Nets

NanoRoute supports the following two use models for pre-routed nets.

- If a via does not match the net's non-default rule (NDR) via list, it is still allowed.
  Note: Presently, there is a workaround that forces it to become a special via but NanoRoute will change to fully support it without changing it to a special via.
- If a routing segment has an NDR that does not match the net's NDR, it will be allowed.
  Note: If NanoRoute is used to route a net (for ECO routing or so on), it will rip-up all the odd vias and routing segments, and reroute them. There are a variety of ways NanoRoute can avoid routing a net:
- By setting `setAttribute -skip_routing` to `true` for that net.
- By marking all the route segments as `FIXED`, and so on.

23.5 Saving Blackboxes in OpenAccess

A blackbox must be committed into a partition before saving it to OpenAccess. After doing this, a cell type will not remain as blackbox, but will get saved as a block.

23.6 Importing the Power and Ground Nets Connections Using Verilog Netlist

Innovus can infer global net connectivity from Verilog netlist so that you may not specify it through the multiple `globalNetConnect` commands. To enable this:

- All undefined PG connections should be defined by using `globalNetConnect`.
- All power and ground net names must be defined in the configuration file.
- Use the `-override` parameter such that `globalNetConnect` does not override the PG connections in the netlist.
- Use the new `-netlistOverride` parameter. Doing so, the physical netlist will reflect overrides but the logical netlist will not reflect such changes.

23.7 Turning Off Power and Ground Connections in a Netlist

The `oa2verilog` command contains a mechanism to determine which terminals of the cells in a library should be included in the output netlist. If a cell library has power and ground terminals, but you prefer to generate a Verilog netlist that does not contain power and ground terminals, run the `verilogAnnotate` command on a reference library containing cells to indicate the ports that should be output in the Verilog netlist.

23.7.1 verilogAnnotate Command Syntax

```
verilogAnnotate
-refLibs libList
-verilog fileList
[-libDefFile file]
[-logFile file]
[-noInfo msgIds]
[-noWarning msgIds]
[-refViews viewList]
[-templateFile file]
[-tolerate]
[-h | -help]
[-v]
[-version]
```

23.8 Generating Power and Ground Pins

Power and ground pins are created in three ways:

- When you create a power mesh in Innovus, power and ground pins are automatically generated in VDI.
- In `sroute`, you have an option to create pins at the intersection of the power grid and `prBoundary`.
- Use the `createPGPin` command in Innovus.

23.9 Using Abstract Views in Innovus

Innovus is designed to use abstract views. If the cells remastered with layout views are read in Innovus, then errors might appear with `verifyGeometry`, and so on.

23.10 Performing Power Routing Outside Innovus

Power routes created using Virtuoso might not have the `stripe` attribute that is needed by `sroute` in Innovus. To set the shape attribute on all the special wires (power routes) of `$my_nets` to `stripe`, use the following command in Innovus:

```
dbSet $my_nets.sWires.shape stripe
```

23.11 Generating Abstracts with Antenna Information

To generate abstracts with antenna information, you can use either `write_lef_abstract` (after `verifyProcessAntenna`), `save_abstract` (after `verifyProcessAntenna`), or use the abstract tool. If you want to do cover obstruction style modeling for the completed partition, use `write_lef_abstract`/`save_abstract`. For detailed modeling, you can use the abstract tool.

For determining metal density, you can run `verifyMetalDensity -saveToDB` before `write_lef_abstract`/`save_abstract`.

Note: Using non-overlapping windows is recommended for this usage.

23.12 Mapping Virtuoso Bind Keys to Innovus Bind Keys

There is an internal map that tries to find Innovus equivalent commands for the user's Virtuoso bindkeys and assigns them to the keys according to the user's Virtuoso setup. For example, if a Virtuoso bindkey uses `hiRedraw()` as bindkey instead of the default key in Innovus, the Innovus behavior aligns in a way that it now maps to the `redraw` command in Innovus.

The following table describes the mapping between Virtuoso bind keys to Innovus bind keys.

Virtuoso Action	Innovus Action
`leHiClearRuler()`	`cleanRuler`
`leHiMerge()`	`mergeWire`
`leEditDesignProperties()`	`summaryReport`
`leHiReShape()`	`resizeMode`
`leHiSearch()`	`getWireInfo`
`hiRedo()`	`redo`
`hiZoomOut()`	`zoomOut`
`geSingleSelectPoint()`	`selectMode`
`leHiCopy()`	`copySpecialWire`
`leHiEditDisplayOptions()`	`popUpEdit`
`hiZoomAbsoluteScale (hiGetCurrentWindow())`	`fit`
`leHiCreateRuler()`	`createRuler`
`leHiMove()`	`moveWireMode`
`leHiCreateVia()`	`addViaMode`
`leHiEditProp()`	`attributeEditor`

leHiRotate()	rotateInstance
leHiStretch()	stretchWireMode
leUndo()	undo
hiPrevWinView (hiGetCurrentWindow())	previousView
hiZoomIn()	zoomIn
geScroll (nil \\\"n\\\" nil)	panUp
geScroll (nil \\\"s\\\" nil)	panDown
geScroll (nil \\\"w\\\" nil)	panLeft
geScroll (nil \\\"e\\\" nil)	panRight
geSave()	saveDesign
leHiDelete()	deleteSelected
cancelEnterFun()	cancel
geDeselectAllFig()	deselectAll
leSetFormSnapMode (\\\"90XFirst\\\")	snapFloorplan
hiRedraw()	redraw
leHiSplit()	splitWire

23.13 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 Action
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.

23.14 Generating a Verilog Netlist for Innovus from a Virtuoso Schematic

This section describes the various settings available to generate Verilog netlist for Innovus from a Virtuoso schematic. These settings are categorized as:

- Netlisting Options
- Schematic and Hierarchy Editor setup
- Netlist Generation

23.14.1 Netlisting Options

The following table describes the variable settings that need to be done in the `.simrc` file:

Settings	Function
`simVerilogFlattenBuses=nil`	Preserves the bus syntax in order to avoid the individual bits to be created instead of the bus syntax.
`simVerilogNetlistExplicit=t`	Dumps explicit connection which is required for Innovus.
`simVerilogGenerateLogicalVerilog=t`	Generates a logical verilog netlist for a design.
`simVerilogGenerateSingleNetlistFile=t`	Generates a single netlist instead of splitting them into multiple ones (one netlist per module).
`vlogExpandIteratedInst=t`	Expands the iterated instances for Innovus.
`vlogifCompatibilityMode="4.0"`	Dumps out the Verilog version.

23.14.2 Schematic and Hierarchy Editor Setup

- Ensure that there are no PCells at the top-level schematic. If you don't want the power and ground connections to be a part of the netlist, see the Filtering Power and Ground Nets in Verilog section.
- In the Hierarchy Editor (control schematic traversal), set the `viewList` and the `stopList` in

order to traverse the design correctly. For each instance make sure to define the `view to use' accurately.

- `viewToUse' symbol
  Instances with `viewToUse' specified as symbol are interpreted in Innovus as follows:
- If no Verilog netlist exists and `abstract' view exists, it is considered as an IP (stop at the IP level).
- If Verilog netlist exists and no `abstract' exists, it is considered as a module with no physical representation (physically flatten).
- If no Verilog netlist exists and no `abstract' exists, it is considered as an empty module with no physical representation.

23.14.3 Netlist Generation

Generate the netlist using *CIW--Tools--NC-Verilog* or from Virtuoso Schematic Editor using *Launch--Simulation--NC-Verilog.*

Now, use the *Commands-Generate Netlist* menu command after initializing the design.

Following is the SKILL equivalent of the NC-Verilog based command:

- `deInstallApp(getCurrentWindow() "NC-Verilog")`
- `vtoolsIseGNCForm->simSimButton->value= nil`
- `vtoolsIseGNCForm->simElabButton->value= nil`
- `vtoolsIseGNCForm->simCompileButton->value= t`
- `vlogifNCInitDesign()`
- `vlogifNCNetlist()`
  The `vlogifNCNetlist()` skill function generates the verilog netlist from the schematic from which it has been run.

For more information on generating a netlist, see *Virtuoso NC-Verilog Environment User Guide*.

23.14.4 Problems and Solutions

- If you have the inherited connections dumped out (*`inh_xx`* ports).
  Solution: Read the Filtering Power and Ground Nets in Verilog.

- Verilog netlist is naming array instance as `INST_n_` but the instance name in Virtuoso is `INST(n)`.
  Solution: Type the following:

  ```
  hnlMapInstName = `("!" "#" "$" "%" "&" "(" ")" "*" "+" "," "-" ( "." "_") ("/"
  "_") ":" ";" ( "<" "(") "=" (">" ")" ) "?" "@" "[" "\\" "]" "^" "`" " "{" "|" "}"
  "~")
  ```

  You need as well to escape manually iterated instances due to the (): AND2_C \I5(0) (.g(net7[0]), .A(A), .B(B));

23.14.5 Filtering Power and Ground Nets in Verilog

The following methods can be used to filter power and ground nets in Verilog:

- Netlisting Options
- Schematic modifications

23.14.6 Netlisting Options

Set the following in the `.simrc` file:

```
simVerilogGenerateLogicalVerilog=t
```

This option could be used in addition to the ones mentioned in the Schematic Modifications section.

23.14.7 Schematic Modifications

Perform the following schematic modifications:

- Modify Schematic views
- Modify Symbol views
- Schematic modifications: Manual
- Schematic Modifications: Automated

23.14.7.1 Modify Schematic views

1. Set the signal type `power` to specify power nets.
2. Set the signal type `ground` to specify ground nets.
3. Set the signal type to `tieHi`/`tieLo` on signals which need to be of these types.

Note: Ensure that the nets which are not power or ground are set to a signal type other than `power`, `ground`, `tieHi` or `tieLo`.

23.14.7.2 Modify Symbol views

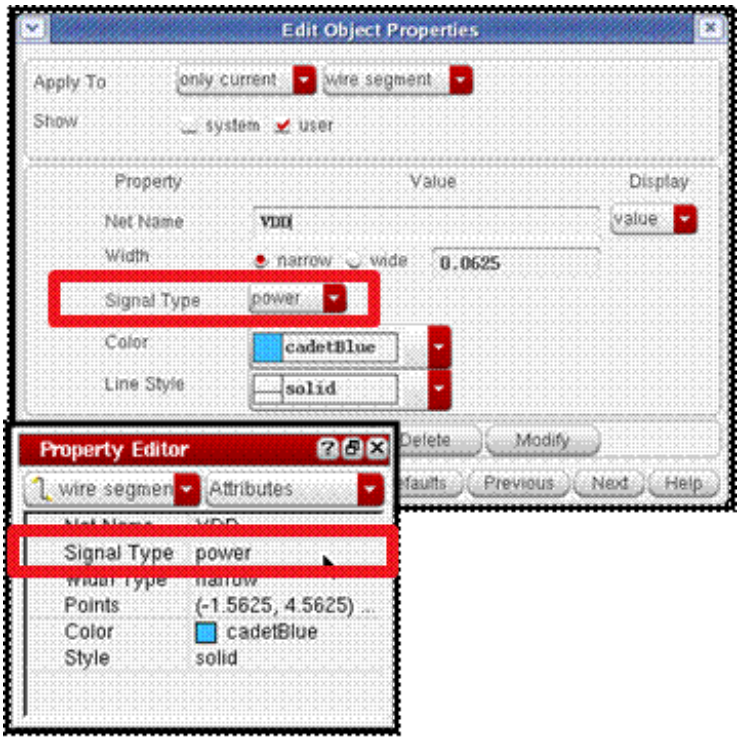
1. Set the signal type to `power` for the power symbol pins.
2. Set the signal type to `ground` for the ground symbol pins.
3. Set the signal type to `tieHi`/`tieLo` on signals which need to be of that type

Note: Ensure that the pins which are not power or ground are set to signal type different than `power`, `ground`, `tieHi` or `tieLo`.

23.14.7.3 Schematic modifications: Manual

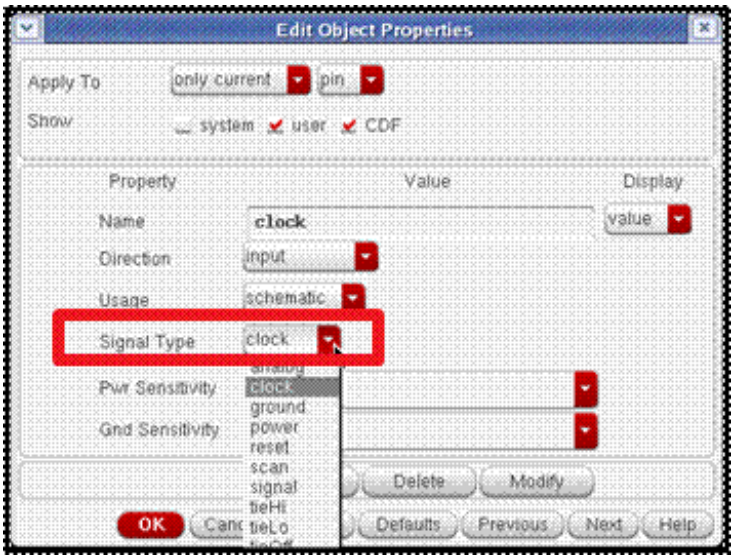
Schematic Net signal type

1. Select the Net in the Navigator or in the canvas.
2. Use the *Edit Object Properties* window or the *Property Editor* assistant to change the signal type to `power` or `ground`.



Symbol Pin Signal Type

1. Select the pin in the navigator or in the canvas.
2. Change the signal type to `power` or `ground`.



23.14.7.4 Schematic Modifications: Automated

1. Register the power and ground nets for the auto assignment of signal type.

   ```
   ciRegisterNet("supply" '("vdd" "vdd!") ?regexNetNames '("^vdd"))
   ciRegisterNet("ground" '("gnd" "gnd!") ?regexNetNames '("^gnd"))
   ```

2. Check the current registered nets.

   ```
   ciGetNetNames("supply")
   ciGetNetNames("ground")
   ```

3. Set the SKILL variable to enable the auto assignment of signal type based on registered net during check and save.

   ```
   schSetEnv(postProcNetSigTypes t)
   ```

4. Activate the signal type consistency check into the schematic during check and save.

   ```
   schSRCForm->tabField->page7->enableSigTypeChecks->value= t
   ```

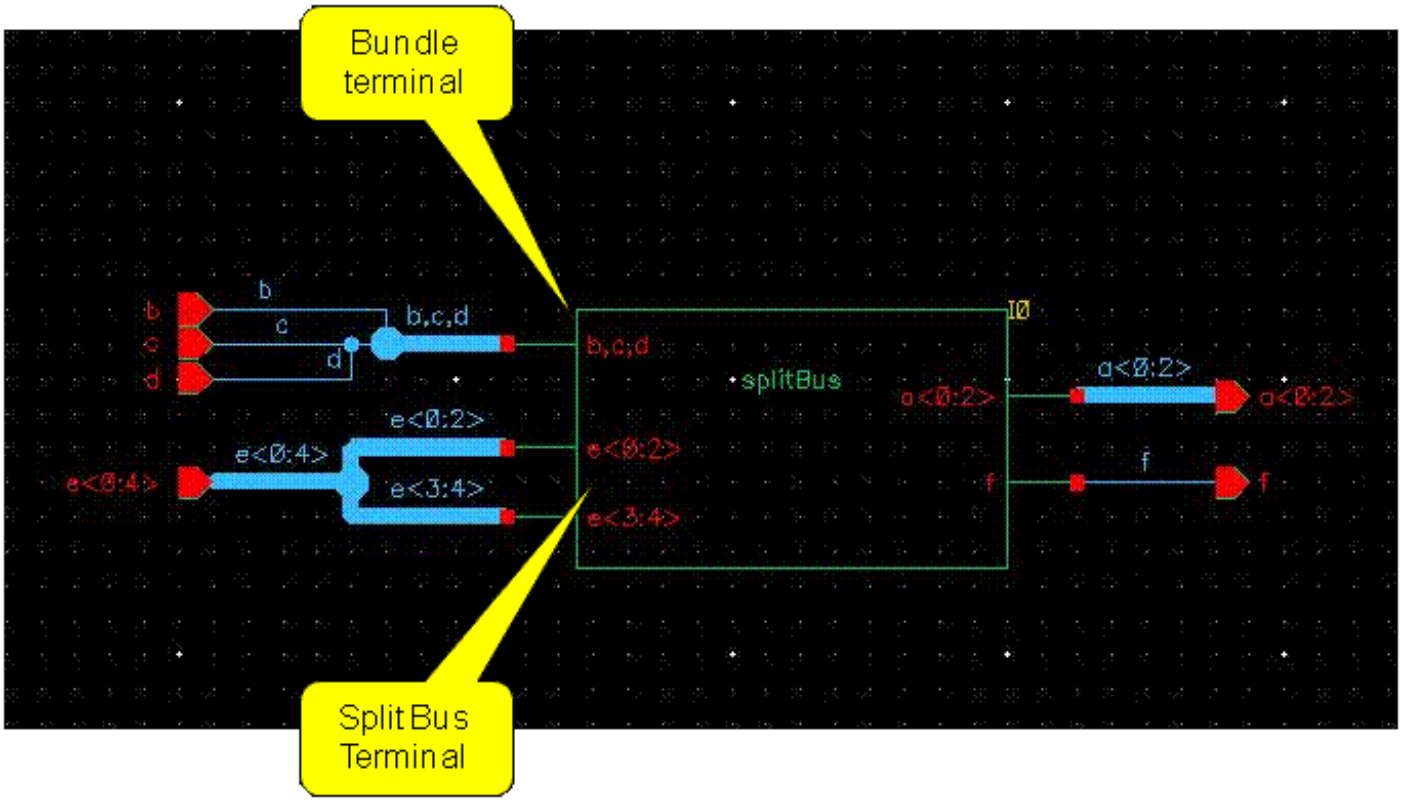
5. Run the schematic hierarchical check and save.

   ```
   schHiCheckHier()
   ```

23.14.8 Limitation due to Split Bus and Bundle with `Merge All'

You could face a problem when you use a mix of split bus and bundle with the *Merge All* option of NC-Verilog.

The following picture illustrate the type of configuration which could lead to a problem.


Bundle terminal
b
c
d
b,c,d
e<0:4>
e<0:2>
e<3:4>
b,c,d
splitBus
a<0:2>
f
a<0:2>
f
Split Bus Terminal

23.15 Creating a Non-Default Rule in Virtuoso and then Using it in Innovus

You can create a valid NDR in Virtuoso and then use it in Innovus using the following steps:

23.15.1 Using Virtuoso

1. Run Virtuoso
2. Select *Tools - Library manager.*
3. Select the Lib/Cell/View of the design.
4. Once the view is open, Select *Layout GXL.*
5. From the *Workspace* option, run the *Virtuoso Constraint Editor*.
6. Open the *Process Rule Editor*. You create NDRs in the *Process Rule Editor*.
7. Select the *Design* radio button. This option is selected because we want to create an NDR which is a part of the design.
8. Specify the NDR name in the *Create Constraint Group* option. For example, `myNDR3`.
9. Click on + to create the NDR.
10. In the next step, we need to assign properties to the NDR to make it a valid NDR in Innovus. In Innovus, an NDR is considered to be valid if it has valid routing layers defined.
11. Select *Minimum Width*, *Minimum Spacing Same Layer*, *Valid Layers*, *Valid Vias* from the *Create Process Rule* text box and select + to assign this property to the NDR.
12. Click on *Value* to assign layers. Select Metal1, Metal 2...Metal 6 and click *Update*.
13. To define valid vias in the NDR, select vias and click *Update*.
14. To specify minimum width or spacing for the valid layers, select minimum width/minimum spacing and specify a value.
15. You can also copy process rules from other constraint groups using *Copy Process Rules* and *Paste* buttons.
16. After pasting the rules, if needed, select and modify the values.
17. Select *Close*.
18. In the constraints manager, select nets.
19. Create a net class constraint.
20. Change default group of the net class to the one you created. For example, `myNDR3`.
21. Save the design in Virtuoso.
22. Exit Virtuoso.

23.15.2 Using Innovus

1. Run Innovus.
2. Open the same cellview for which you created the NDR using *File - Restore Design* or *File – Import Design*.
3. Open the *Integration Constraints Editor*.
4. To view the NDR created in Virtuoso, select `myNDR3`. It has same values which you specified in Virtuoso.
5. In this procedure, you created an NDR in Virtuoso, assigned nets with the NDR, save the design, and import the design in Innovus with the same values.

23.16 Troubleshooting Common Errors in Innovus OpenAccess Flow

23.16.1 IMPOAX-717 Error

```
**ERROR: (IMPOAX-717):  There should be a cut layer after layer 'mxx'. Check the order
of layers in the technology information read from OA database.
```

Cause: The technology file needs to have a cut layer in between any two routing layers defined. In this case, either layer *mxx* does not have the cut layer defined or the same layer is defined in the tech multiple times, perhaps with a different width or thickness.

Solution: To investigate this issue further, dump the tech in Virtuoso and look for the presence of a cut layer between any two routing layers. If the error is being caused because of multiple definitions of the same metal layer (with different width/thickness), the tech file might not have been created properly. This method of defining multiple version of the same metal layer is referred to as a Multi Metal Stack Option (MMSO).

The proper way of creating a tech file with multiple metal stack is to create a LEFdefaultRouteSpec (LDRS) in the OpenAccess tech for each metal-stack. The LDRS for a metal stack will have only the layers used for that specific stack. Once this is done, Innovus can choose the correct metal stack for a particular design by using the following command.

```
set init_oa_default_rule LEFdefaultRouteSpec_name
```

23.17 Working with Old Versions of OpenAccess Data

A useful utility to run on old versions of OpenAccess data is *oaScan.* It can be found in both the Innovus and Virtuoso hierarchies. It scans the contents of a library and checks for inconsistencies in the OpenAccess design, tech, and DMData databases. Such inconsistencies in data can cause the software to hang, crash, or work incorrectly. You can run *oaScan* on cells/designs to to repair these inconsistencies and debug random crashes and hangs.

Executable Path:

<INNOVUS or VIRTUOSO Installation>/tools/bin/oaScan

Usage:

Syntax: `oaScan`

```
-lib libName
[-libPath libPath]
[-cell cellName]
[-view viewName]
[-viewType viewTypeName]
[-dataModel version]
[-repair]
[-verbose]
[-forceLibPath] #Ignores check for .oalib in libPath
[-fixWarnings]
[-noWarnings]
[-logFile file]
[-templateFile file] #File that collects command options
[-h | -help]
[-version]
```

Document Location:

<INNOVUS or VIRTUOSO Installation>/doc/oascan/oascan.pdf