cadence®

Integrity 3D-IC User Guide

Product Version 21.16
Last Updated in November 2022

© 2022 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

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

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

All other trademarks are the property of their respective holders.

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

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

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

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

Contents

About This Manual 14
Audience 14
How This Manual Is Organized 14
Conventions Used in This Manual 15
Related Documents 16
Additional System Planner Documentation 16
Additional Layout Implementation Documentation 17
Additional Learning Resources 18
Online Trainings 18
Videos 18
1 20
Introduction to Integrity 3D-IC 20
Overview 20
Integrity 3D-IC Workflow 21
2 24
Starting Integrity 3D-IC 24
Invoking Integrity 3D-IC 24
Tcl Command Line 26
iHDB Root Directory 27
3 28
The Integrity Hierarchical Database (iHDB) 28
Overview 28
iHDB Directory Structure 29
Basic iHDB Commands 30
Setting the iHDB root directory 30
Creating a DB 30
Restoring a DB 31
Restoring a top-level DB and loading the module model of its child device or partition 31
Loading a child device or partition’s module model with add-ons 31
iHDB Module Model and Add-On Types 32
Multi-Die Mode of the Layout View 33

Design Stack 33
System Planner View Displaying the Die Floorplan 34
System Planner View Displaying Die Placement 35
Layout View (Multi-die Mode) 36
4 44
Understanding the System Planner Environment 44
Introduction 45
System Planner Terminology and Data Model 45
Terminology 45
Using the System Planner Workspace 51
Using the System Planner Toolbar 53
Using the Canvas 54
Using the Layers and Objects Views 57
Using the Flow Manager 60
Using the Device Hierarchy 62
Using the Attribute Editor 67
Using the Selection Filter 68
Configuring Tool Preferences 70
Report and Log Files 71
Scripting 71
5 72
Working with Designs in System Planner 72
Defining Substrates 74
Creating Substrates Interactively 74
Creating Substrates by Importing a LEF Tech or CSV Layers File 76
Creating Padstacks - Pads, Bumps, and TSVs 80
By Importing LEF Pads and Pad Macros 81
By Using LEF Macros 83
By Creating Bumps Interactively 83
Viewing and Modifying Padstacks 88
Creating Devices 89
By Using the Create Device Option 90
By Importing a Die Abstract File 95
By Importing LEF/DEF Files 97
By Importing a Die CSV File 99
By Importing a Pins List CSV File 100

By Importing a System Planner "OIO" Project File 101
By Importing a Die Text "TXT" File 101
Changing the Origin of a Device 103
Changing the Size of a Device 103
Floorplanning Devices 104
Flipping Devices 105
Moving Devices 105
Copying Devices 106
Using the Device Table Editor 108
Using the Attribute Editor 109
Importing a CSV Devices File 109
Using the Ruler Mode to Measure 109
Defining the Design Stackup 110
Creating Contact Layers and Contact Pads 112
Creating Contact Layers 113
Creating Contact Pads 117
Managing Contact Pads 120
Creating Contact Devices 120
Running Inter-Substrate Connectivity Checks 121
Exploring Results 123
Defining or Importing Netlist Connectivity 124
Understanding the Top-Level Design Requirements 125
Work at an Abstract Level using Black Box Die Macros 126
Creating the Top-Level Design Hierarchy 127
Importing Verilog Netlists 129
Creating Connectivity with Term Files 130
Importing Term List and Term Map Files 132
Creating Terms Using the Netlist Editor 134
Exploring Connectivity with the Netlist Editor 136
Exploring Connectivity with the Graph Net/Term Commands 140
Working with Power and Ground Nets 142
Defining Differential Pairs 145
Defining Net Personality Groups 147
Creating Bumps and TSVs 152
Using the Bumps for All Layer Shapes Command 152
Creating a Pin Pattern for Bumps and TSVs 156
Assigning Power and Ground Nets 161

Assigning Nets to Bumps 162
Creating TSVs Using the Fanout Vias Command 164
Optimized Bump Assignment with the Connection Optimization Command 166
Interfacing with Other Implementation Tools 169
Creating Die Symbols for Package Design 169
Exporting Implementation Files 182
6 187
Interoperability Commands and Features 187
7 188
Tcl Interaction and Commands 188
Executing Tcl Commands for the Layout Window 188
Executing Tcl Commands for the System Planner Window 188
Controlling the Planner Command Log Mode 188
Checking the Version 189
Integrity-Specific Tcl Commands 190
clear_floorplan_for_proto_seeds 190
create_proto_seeds 191
get_proto_seeds 193
proto3d::assign_shadow_bump 194
proto3d::create_3d_stacked_proto_shapes 196
proto3d::create_module_wrapper 197
proto3d::get_obj_die_id 198
proto3d::insert_zero_delay_cell 199
proto3d::set_obj_die_id 201
proto3d::unfold_floorplan 202
proto_design 203
pull_from_planner 205
push_to_planner 206
set_proto_design_mode 207
update_proto_seeds 209
System Planner Tcl Commands 211
align_contacts 216
align_device_placement 218
align_selected 218
assign_bumps 220
assign_pg_bumps 221

assign_terms 223
check_cell_uniqueness 223
check_contacts 224
check_design 227
check_device_grid 228
check_io_info 229
check_pin_grid 230
check_place 230
classify_nets 232
clear_all_rulers 233
clear_selection 233
close_design 234
commit_connection_optimization 235
complete_tcl_cmd 235
connect 237
connect_contact_pin 237
copy_device 239
copy_pin 241
create_bump 243
create_bump_replication 248
create_bump_template 250
create_contact_layer 251
create_contacts 254
create_current_region 257
create_design 259
create_device 261
create_device_grid 267
create_device_template 268
create_diff_pair 271
create_external_contact_layer 272
create_feedthru 273
create_group 274
create_layer 275
create_marker 276
create_modInst 278
create_module 278
create_module_model 280

create_net 281
create_package 282
create_padstack 288
create_pin_grid 289
create_pin_matrix_pattern 290
create_place_blockage 294
create_projective_pins 296
create_region 299
create_route_blockage 300
create_stitch_zone 301
create_substrate 302
create_term 303
create_via 305
delete_contact_layer 306
delete_contacts 308
delete_current_region 310
delete_external_contact_layer 310
delete_group 312
delete_marker 312
delete_nets 314
delete_obj 314
delete_padstack 317
delete_region 318
delete_stitch_zone 318
delete_substrate 319
deselect_obj 320
edit_bulk_nets 322
edit_bulk_terms 324
eval_bsh 327
get_db 328
get_obj_in_area 347
get_property 349
get_selection 351
get_transform_shapes 351
group 353
gui_canvas 354
gui_fit 355

gui_hide 355
gui_redo 356
gui_set_obj_color 357
gui_set_visible_layer 358
gui_set_visible_obj 359
gui_set_visible_substrate 361
gui_show 362
gui_undo 363
gui_zoom 363
is_top_design 364
list_select_properties 366
man 366
move_device 367
move_pin 369
place_inst 370
place_io_info 371
read_allegro 374
read_bga_text 377
read_bump_locations 379
read_def 381
read_design_stack 383
read_device_placement 384
read_die_text 385
read_io_info 387
read_lef 388
read_module_model 389
read_net_interface 390
read_net_list 391
read_net_map 391
read_oio 393
read_pin_label 393
read_pin_physical_list 394
read_power_intent 395
read_property 396
read_systemlvs_cdl_port_config 397
read_term_list 398
read_term_map 398

read_verilog 399
refresh_canvas 400
refresh_pin_map 402
remove_connections 403
rename_matrix_objs 403
run_lec 406
run_systemlvs 407
run_thermal 408
select_obj 409
set_connection_optimization 412
set_db 413
set_era_die_config 415
set_era_pg_net_config 417
set_era_result_user_preference 418
set_era_user_preference 419
set_hierarchy 420
set_lec_library 421
set_lec_netlist 422
set_lec_user_preference 423
set_message 424
set_module_model 425
set_preference 425
set_property 426
set_systemlvs_design_check 428
set_systemlvs_interface_check 431
set_systemlvs_netlist_usage 434
set_thermal_ambient_condition 435
set_thermal_contact_attribute 436
set_thermal_heat_sink 438
set_thermal_layer_material 439
set_thermal_molding_compound 441
set_thermal_package 442
set_thermal_power 444
set_thermal_tech 446
set_thermal_test_board 447
set_thermal_user_preference 448
set_vsrc_layers 449

show_connections 450
snap_cell 450
source_bsh 451
space_selected 452
swap_pin_nets 455
synthesize_boundary 455
test_same_oio 456
test_same_text 457
thermal::view_result 458
unassign_bumps 460
ungroup 461
ungroup_device 462
uniquify_cell 462
unplace_obj 464
update_contact_layer 465
update_current_region 467
update_device 469
update_device_template 474
update_padstack 476
update_shrink_factor 478
update_stitch_zone 479
update_substrate 480
view_lec_result 481
view_systemlvs_result 482
write_allegro 483
write_bga_text 484
write_bump_locations 485
write_contact_pin_map_file 486
write_def 487
write_design_stack 487
write_device_placement 488
write_die_text 489
write_edtext_file 491
write_oio 492
write_pin_label 492
write_pin_physical_list 493
write_ploc_file 494

write_power_intent 494
write_term_list 496
write_term_map 497
write_to_gif 497
write_verilog 499
8 500
Implementation Design Navigation and Management 500
Using the Multi-Die Physical Layout View 500
Using the Color Panel in the Multi-Die Mode 501
Switching between an Individual Die and 3D-IC Top 503
9 505
Memory-on-Logic Flow 505
Overview of 3D-IC Memory-on-Logic Flow 506
Recommended 3D-IC Memory-on-Logic Flow 508
Concurrent Multi-Die Placement 509
Creating Netlist Hierarchy for Memory and Logic Dies 509
Assigning Shadow Bumps 510
Inserting a Zero Delay Cell 511
Physically Unfolding the Design 512
Creating Bumps on Each Die 512
10 513
3D Early Floorplan Synthesis 513
Overview of 3D Early Floorplan Synthesis 513
Benefits of 3D Early Floorplan Synthesis 514
General 3D-EFS Flow 514
Using Proto Seeds 515
Defining Proto Seeds 516
Adding Constraints for Proto Seeds 517
Placing the Proto Seeds 517
Generating and Optimizing Partition Shapes 518
Partitioning the Design 519
Logical Partitioning 519
Physical Partitioning 520
11 522
Using the Interposer Route Form 522

Interposer Route 522
Interposer Route - Basic Fields and Options 523
Related Commands 525
Routing Pattern 526
Routing Pattern - Basic Fields and Options 526
Bus Group Diagnosis 528
Bus Group Diagnosis - Basic Fields and Options 529
Related Commands 530
Interlayer Shielding 530
Interlayer Shielding - Basic Fields and Options 530
Related Commands 531
12 533
Configuring and Launching System Level Analysis 533
Using The Flow Manager 533
Thermal Analysis 535
Early Rail Analysis 544
Static Timing Analysis 554
System LVS 557

About This Manual

The Cadence® Integrity™ 3D-IC product provides designers a single platform for both 3D-IC system-level planning and layout implementation. This manual describes how to use Integrity 3D-IC to handle complex, multi-die designs.

Audience

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

How This Manual Is Organized

The *Integrity 3D-IC User Guide* explains the Integrity 3D-IC workflow, touching on both the system-level planning in System Planner and design implementation in the layout window

This guide focuses on the specific System Planner features and commands that enable 2.5D- and 3D-IC designs. These typically involve multi-die designs that contact each other directly or using silicon interposers or info type substrates. The term 3D-IC is used generically throughout this guide to describe any 2.5D-IC and 3D-IC stacked die or interposer design.

The package design can also be created and managed as a part of the System Planner design project. System Planner enables co-design of the package substrate, including C4 microbump creation, ball pad assignment, and package netlist creation. System Planner is well integrated into the Cadence® Allegro® Package Designer Plus (APD Plus) package design environment to implement the package substrate. The process of creating the die symbols needed for package design is included in this guide. The System Planner features specific to package design are not covered in this guide. Refer to the *Package Codesign with the Integrity 3D-IC Platform RAK* for more information on package design capabilities with System Planner and APD Plus.

The layout window provides all the place, route, and optimization capabilities needed by a digital IC designer. It provides 3D-IC and 2.5D-Interposer-specific features needed for designing a digital die or interposer of the 3D system.

Conventions Used in This Manual

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

`text`	Indicates text that you must type exactly as shown. For example: `get_selection`
`text`	Indicates information for which you must substitute a name or value. In the following example, you must substitute the name of a specific file for *`file_name`*: `read_def` *`file_name`*
text	Indicates the following: • Text found in the graphical user interface (GUI), including button labels and field names • Terms that need special emphasis • Titles of manuals, other documents, and their sections
`[ ]`	Indicates optional arguments. In the following example, you can specify none, one, or both of the bracketed arguments: `command [-arg1] [arg2 value]`
`[ | ]`	Indicates an optional choice from a mutually exclusive list. In the following example, you can specify any of the arguments or none of the arguments, but you cannot specify more than one: `command [arg1 | arg2 | arg3 | arg4]`
`{ | }`	Indicates a required choice from a mutually exclusive list. In the following example, you must specify one, and only one, of the arguments: `command {arg1 | arg2 | arg3}`

`{[ ] [ ]}`	Indicates a required choice of one or more items in a list. In the following example, you must choose one argument from the list, but you can choose more than one: `command {[arg1] [arg2] [arg3]}`
. . .	Indicates that you can repeat the previous argument.
. . .	Indicates an omission in an example of computer output or input.
Command - Subcommand	Indicates a command sequence, which shows the order in which you choose commands and subcommands from the GUI menu. In the following example, you choose *View* from the menu, *Zoom* from the submenu, and then *In* from the displayed list: *View – Zoom – In*

Related Documents

For more information about Integrity 3D-IC, see the following documents. You can access these and other Cadence documents using the Cadence Help documentation system.

Additional System Planner Documentation

Check the following RAKs at Cadence Online Support:

- *Designing with the Integrity 3D-IC Platform RAK:* Introduces the Integrity 3D-IC platform that integrates 3D design planning, implementation, and system analysis in a single, unified environment.
- *RISC-V Digital 3DIC Design and Analysis Using Integrity 3D-IC Platform*: Describes the 3D-IC digital implementation flow with the Integrity 3D-IC platform. This RAK will guide you to complete the system planning and die implementation of a two-die RISC-V design with a top-down methodology.

- *RISC-V System LVS Flow Using Integrity 3D-IC Platform*: Provides an introduction and guideline of System LVS on the Integrity 3D-IC platform.
- *RISC-V Thermal Flow Using Integrity 3D-IC Platform*: Provides guidelines for performing Thermal analysis on the Integrity 3D-IC platform.
- *RISC-V ERA PSDL Flow Using Integrity 3D-IC Platform*: Provides guidelines for performing the early-stage rail analysis (ERA) PSDL flow on the Integrity 3D-IC platform.
- *RISC-V ERA DEF Flow Using Integrity 3D-IC Platform*: Provides guidelines for performing the ERA DEF flow on the Integrity 3D-IC platform.
- *RISC-V Digital Quantus Flow Using Integrity 3D-IC Platform*: Demonstrates the Quantus 3D-IC digital extraction flow with the Integrity 3D-IC platform.
- *RISC-V STA Flow Using Integrity 3D-IC Platform*: Provides guidelines for performing static timing analysis (STA) on the Integrity 3D-IC platform.

Additional Layout Implementation Documentation

- *Innovus Text Command Reference*
  Describes the Innovus text commands, including syntax and examples.
- *Innovus User Guide*
  Provides information on using Innovus and iHDB.
- Innovus Menu Reference
  Provides information specific to the forms and commands available from the Innovus graphical user interface.
- README file
  Contains installation, compatibility, and other prerequisite information, including a list of Cadence Change Requests (CCRs) that were resolved in this release. You can read this file online at downloads.cadence.com.

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

Additional Learning Resources

Online Trainings

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

You can also write to training_enroll@cadence.com.

Videos

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

- Innovus Video Library

1

Introduction to Integrity 3D-IC

- Overview
- Integrity 3D-IC Workflow

Overview

Integrity™ 3D-IC provides designers a single platform for both 3D-IC system-level planning and implementation.

- System Planner helps designers to set up and integrate all the stacking devices of a 3D design, including digital dies, analog dies, packages, and PCB boards. Compared with the layout view's massive design data, System Planner provides a lightweight view of the 3D-IC system, focusing on the die-to-die stacking configuration, floorplanning, die-to-die interface design, and optimization.
- The layout implementation works on a full Innovus DB and provides all the place, route, and optimization capability that a digital IC designer needs. It provides 3D-IC and 2.5D-Interposer-specific features needed for designing a digital die or interposer of the 3D system. In addition, Integrity 3D-IC also provides a multi-die layout view, so that designers can check multiple dies with different process nodes in a single GUI and display the system analysis result for debugging and locating cross-die design problems.
  The multi-die layout DB also leaves the interface for future extension of real 3D place & route support.
- Integrity 3D-IC gives designers the flexibility to switch between System Planner and layout implementation seamlessly. Design data can be pushed to or pulled from the planner if die floorplan, bumps, TSVs, IO pads, hard macros are changed so that the change is always synced up among all the stacking devices.
- Integrity 3D-IC also works as a cockpit to connect to Virtuoso and Allegro for analog IC, Package and PCB design, and implementation. The Flow Manager can launch system-level analysis with minimal user inputs, including STA (Tempus/Quantus), Power/IREM (Voltus),

Thermal (Celcius) and LVS (Pegasus). It greatly reduces a designer's effort and makes system-level analysis easily accessible so that the analysis result comes back driving the system PPA improvement.

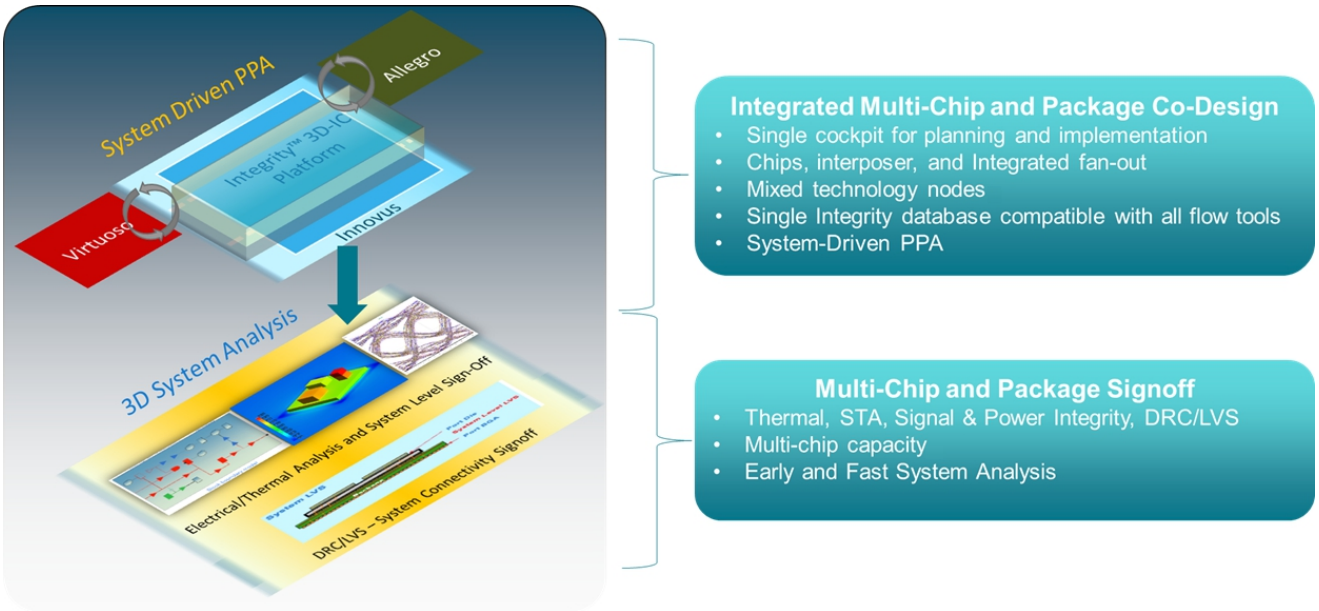
- The design data of each 3D-IC device is saved and organized under a single Integrity database directory. The native Integrity DB Tcl commands make it easy to read, write, update, display, and overlay design DBs and analysis results. The Integrity database is a natural extension of the Innovus hierarchical DB. So, it also supports a digital die that itself is a hierarchical design.

Integrity 3D-IC Workflow

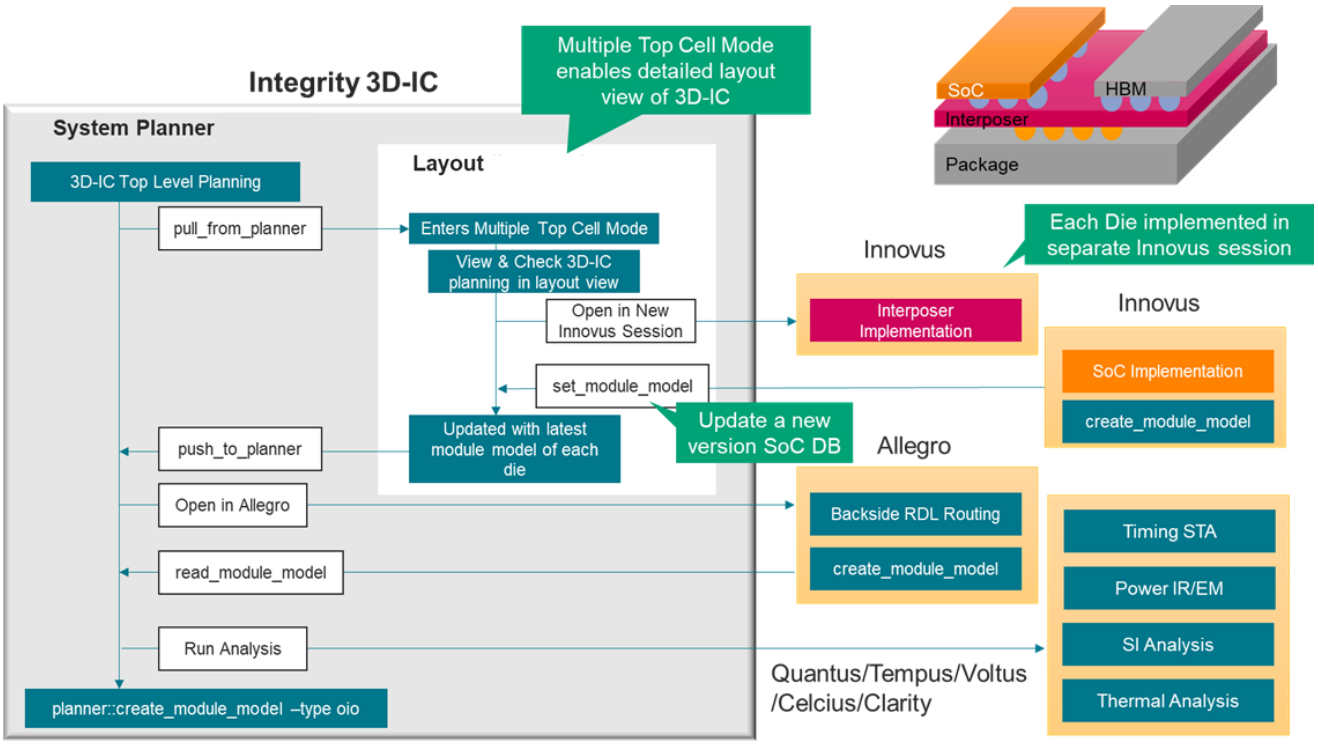
Integrity 3D-IC is a single cockpit for 3D system planning and implementation. Depending on the design style, data readiness, and components of the 3D system, Integrity 3D-IC provides a flexible workflow. Designers can start with high-level design inputs, such as chip size, die-to-die connectivity, and die stacking information, to set up the entire 3D system in System Planner, then push down each die's database to layout implementation. Designers can also start with the existing DB of a die, push it to System Planner to create the other dies to be stacked with it, including the bumps and TSVs between the die interface.

Digital dies can be implemented in Integrity 3D-IC directly after pulling the DB from System Planner. It can also be implemented in another Innovus session by loading the data from Integrity Hierarchical Database (iHDB). Analog die and Package design can be seamlessly opened in Virtuoso and Allegro from the System Planner. After implementation is completed, DB is updated back to System Planner through iHDB.

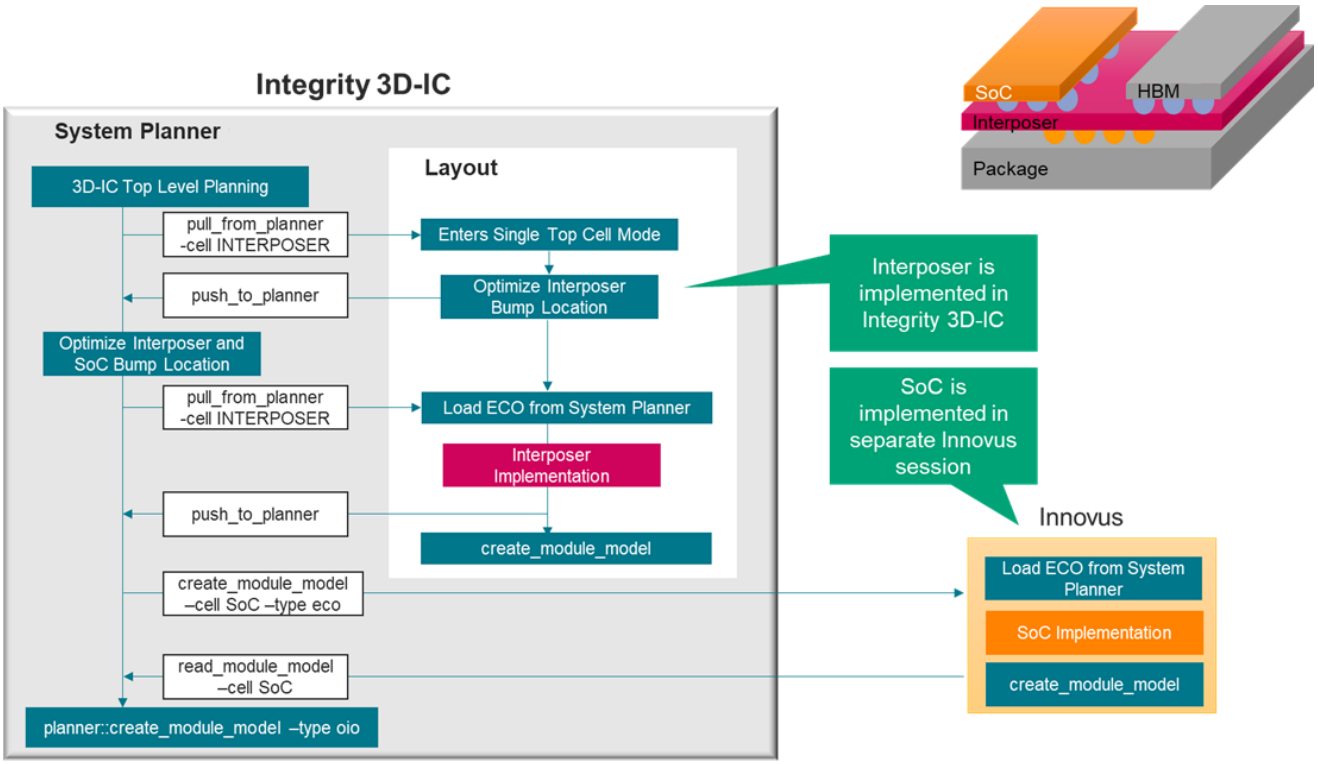
3D System Analysis was a pain point for 3D-IC designers in the past. With Integrity 3D-IC, the Flow Manager in the System Planner provides an easy way to prepare the data for each die and launch system-level analysis from a single cockpit.



Use the Multiple Top Cell mode to display stacked dies in the layout view:



Use the Single Top Cell mode in the layout view to implement one die and push back to planner:


Integrity 3D-IC
System Planner
3D-IC Top Level Planning
pull_from_planner -cell INTERPOSER
push_to_planner
Optimize Interposer and SoC Bump Location
pull_from_planner -cell INTERPOSER
push_to_planner
create_module_model –cell SoC –type eco
read_module_model –cell SoC
planner::create_module_model –type oio
Layout
Enters Single Top Cell Mode
Optimize Interposer Bump Location
Load ECO from System Planner
Interposer Implementation
create_module_model
Interposer is implemented in Integrity 3D-IC
SoC is implemented in separate Innovus session
Innovus
Load ECO from System Planner
SoC Implementation
create_module_model
SoC
HBM
Interposer
Package

2

Starting Integrity 3D-IC

- Invoking Integrity 3D-IC
- Tcl Command Line
- iHDB Root Directory

Invoking Integrity 3D-IC

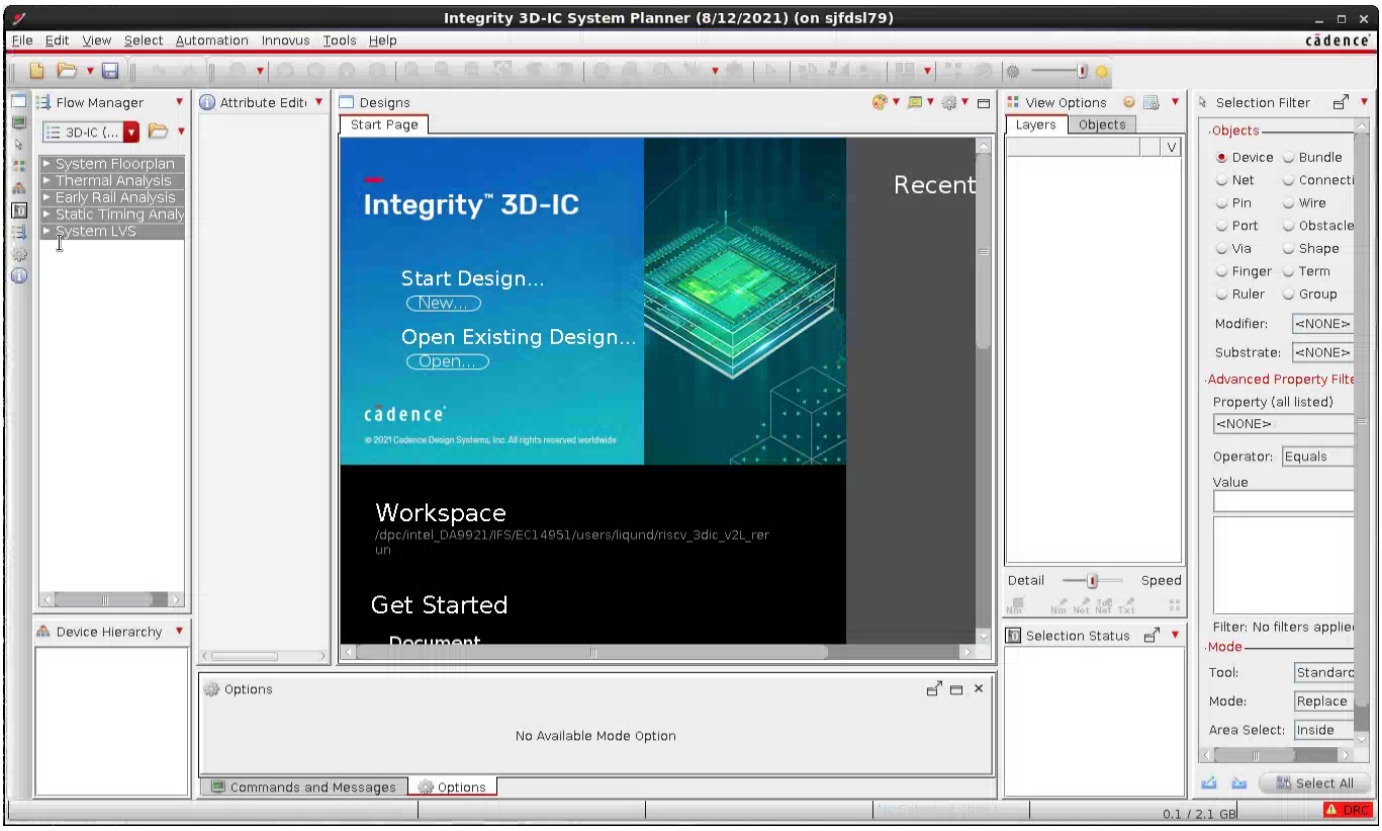
To start the software, type:

```
%integrity_3dic
```

Once the software is launched, the command line prompt shows:

```
integrity_3dic 1>
```

By default, the System Planner window pops up.



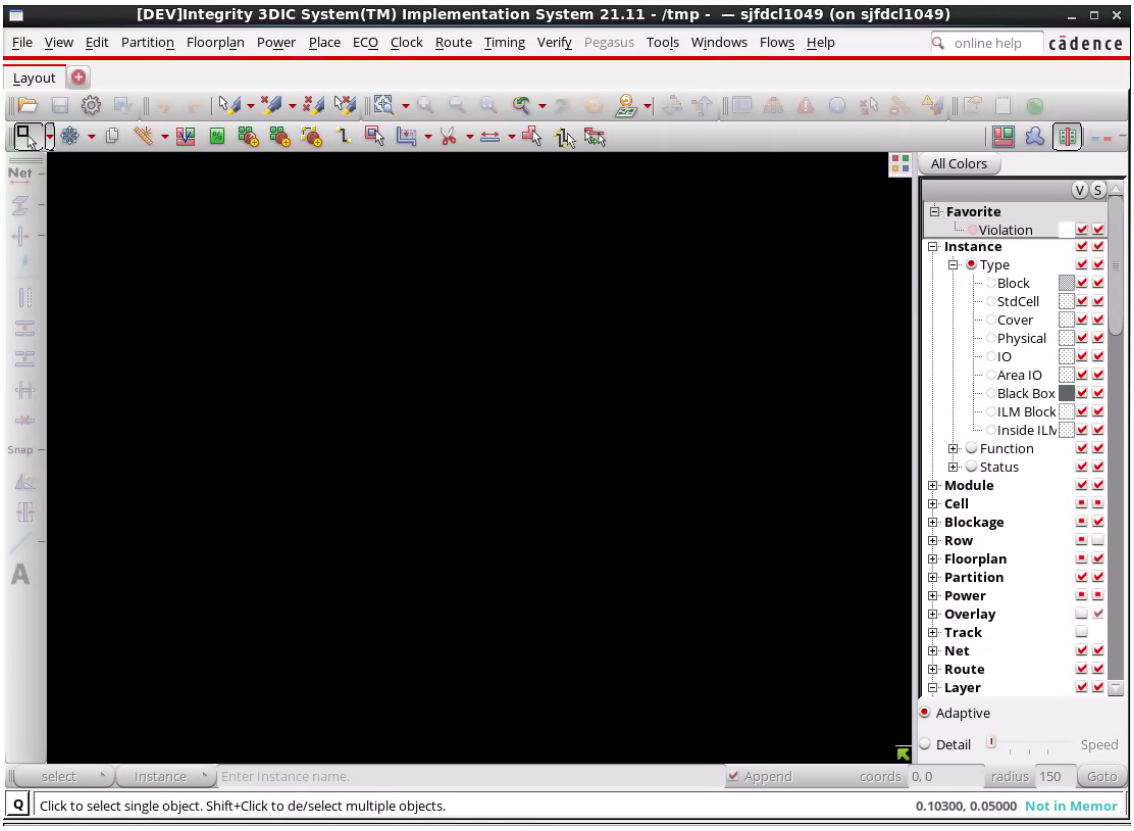
To pop up the Layout window (Implementation), type:

```
integrity_3dic 1> win
```

in the default mode or

```
integrity_3dic 1> gui_show
```

in the Stylus mode.



Tcl Command Line

At the Integrity 3D-IC command line, both the System Planner and Implementation Tcl commands can be used. They apply to the System Planner and Layout window, respectively.

To use System Planner commands, start with the `planner::` Tcl namespace prefix. For example:

```
planner::create_contact_layer -from_device top_die -from_layer M10 -to_device bot_die -to_layer M5
```

If there are multiple lines of the System Planner commands, they can be called through an `eval_planner` statement. For example:

```
eval_planner {     create_contact_layer -from_device top_die -from_layer M10 -to_device bot_die -to_layer M5     create_contacts -from_device top_die -from_layer M10 \           -to_device bot_die -to_layer M5 \           -cell_type device -cell FRONTBUMP }
```

To use the Implementation commands, input the commands directly at the prompt, just like in Innovus™ Implementation System. Integrity 3D-IC is compatible with Innovus Tcl scripts.

iHDB Root Directory

The Integrity Hierarchical Database (iHDB) is the center of the on-disk DB save, restore, and interoperability functions between System Planner, implementation, and analysis tools.

> ⚠ Recommended Practice
> Set the correct iHDB root directory every time you start the software.

To set the iHDB root directory:

```
set_module_model -default_dir iHDBRootDir
```

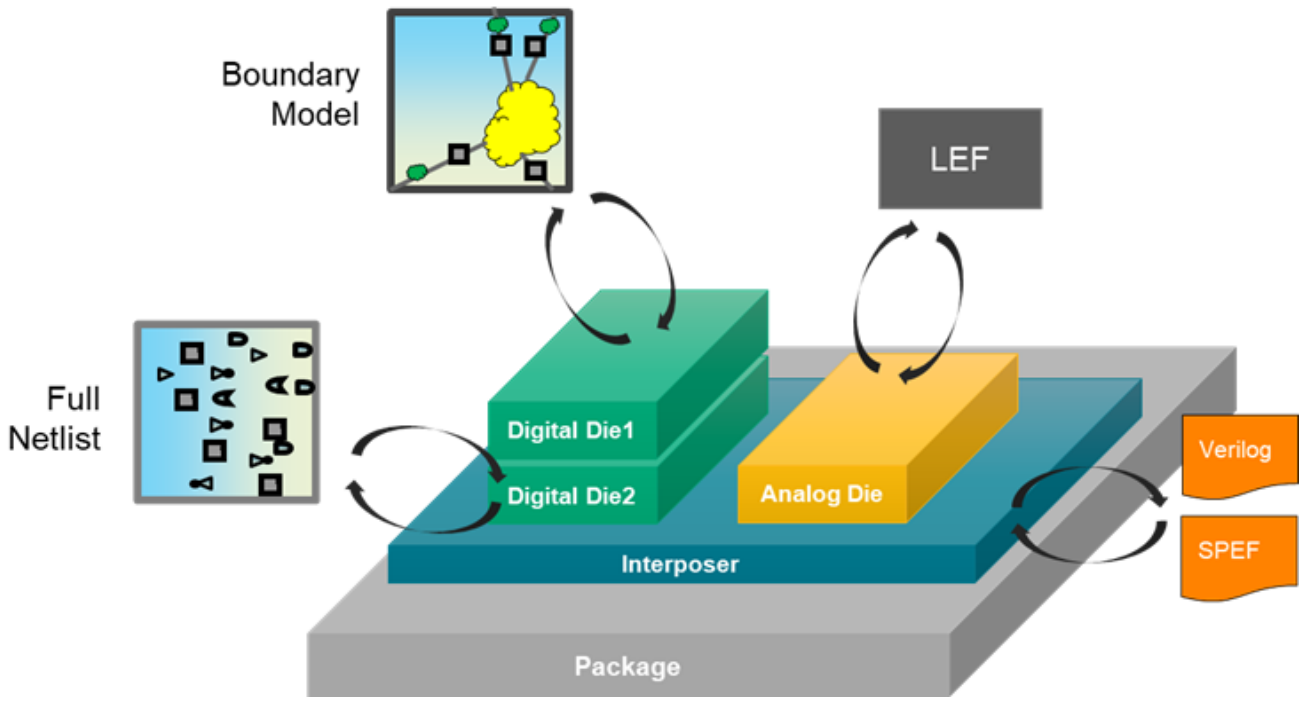
3

The Integrity Hierarchical Database (iHDB)

- Overview
- iHDB Directory Structure
- Basic iHDB Commands
- iHDB Module Model and Add-On Types
- Multi-Die Mode of the Layout View

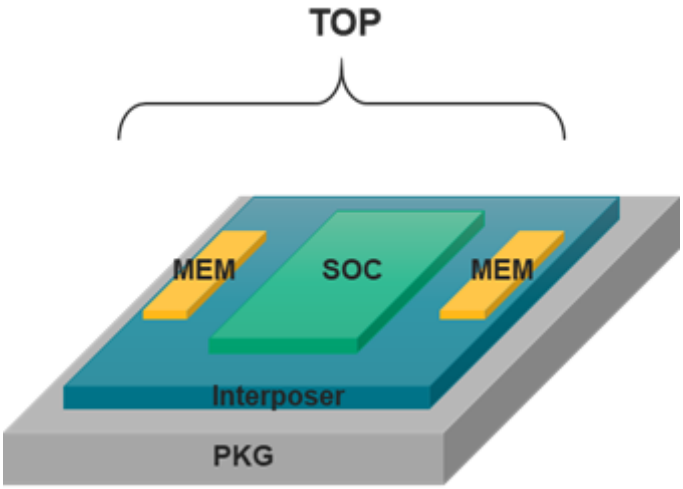
Overview

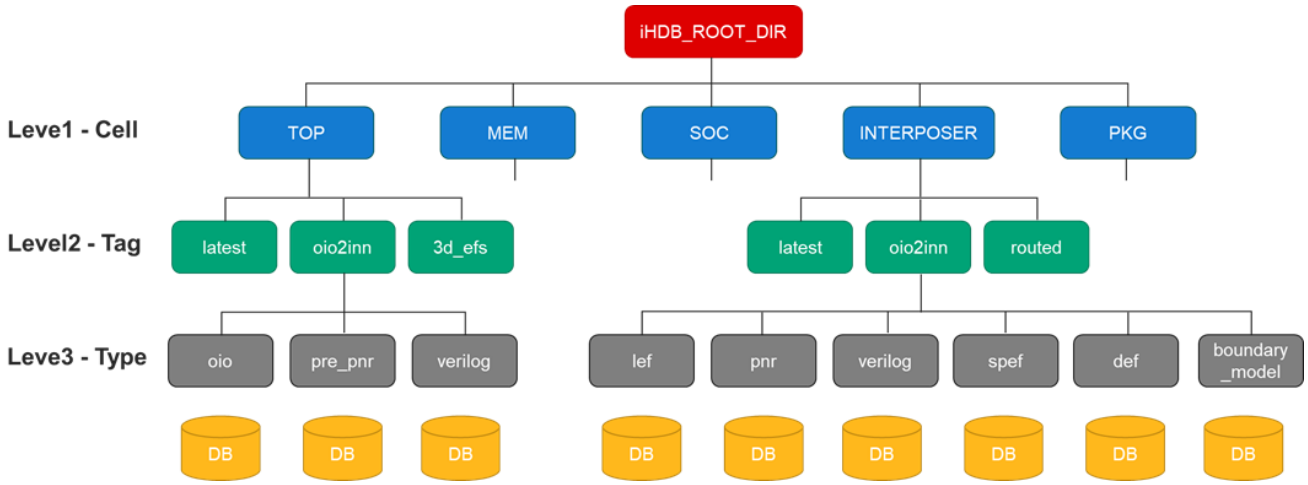
The Integrity Hierarchical Database (iHDB) enables you to manage and navigate through the hierarchical structure of multiple devices of different types easily in a 2D- or 3D-IC design. It is an on-disk database container and management system, which is top-cell (module) oriented with built-in dependency rules. Each cell module can create or be specified with different types of models, from full place-and-route (pnr) view to a high-level abstract, such as LEF, SPEF, or boundary model. iHDB also provides a common user interface to save or restore the DB across all supported tools, no matter what the type of the DB. The cell-tag-type directory structure allows easy version control management and repository synchronization across regions.



iHDB Directory Structure

The iHDB directory structure of a sample 3D-IC design is depicted below. The 3D-IC system top level has four devices under it: SOC, MEM, Interposer, and PKG. Each device may be designed by different engineers on different tools.





The 3D-IC iHDB is also compatible with the Innovus™ Implementation System iHDB used in the 2D hierarchical flow. A digital die in a 3D-IC system can still have its hierarchical structure of multiple partition levels. Each partition can have its cell directory under the iHDB root.

Basic iHDB Commands

Setting the iHDB root directory

Set the root directory of iHDB when you start a design for the first time.

Example:

```
set_module_model -default_dir ./DBS
```

Creating a DB

Save a DB of the current design with the specified type of model abstract.

Examples:

Save four different types of DB for the `prects` step.

```
create_module_model -tag prects -type lef -allow_port_mismatch
```

```
create_module_model -tag prects -type {pnr mmmc spef}
```

Restoring a DB

Load a saved DB. If `-tag` is not specified, the latest DB will be loaded. If `-type` is not specified, `pnr` type DB will be loaded by default.

Examples:

- Load a `pre_pnr` type DB written from Integrity 3D-IC System Planner:

  ```
  restore_module_model -cell bottom_die -tag oio2inn -type pre_pnr
  ```

- Load the latest `pnr` type DB for `top_die`:

  ```
  restore_module_model -cell top_die
  ```

Restoring a top-level DB and loading the module model of its child device or partition

For 3D-IC, it enables the layout view to enter the multiple-die mode with physical hierarchy. For a 2D hierarchical design, each partition can be specified with different types of module models, such as `lef` or `ilm`.

Example:

Load a 3D-IC full design with routed DB of both the top and bottom dies.

```
restore_module_model TOP -tag oio2inn -type pre_pnr

set_module_model -cell top_die   -tag pnr_routed -phys_hier

set_module_model -cell bottom_die -tag pnr_routed -phys_hier

commit_module_model
```

Loading a child device or partition's module model with add-ons

The add-on model is a supplementary model that can be loaded with an existing model or a design.

Example:

Read the 3D-IC top level and the Innovus-routed DB of the two dies with `spef` add-on, to prepare the DB for system STA and timing debug.

```
restore_module_model -cell TOP -tag bump_assignment -add_ons {mmmc spef}
set_module_model -cell top_die -tag pnr_routed -add_ons {spef} -phys_hier
set_module_model -cell bottom_die  -tag pnr_routed -add_ons {spef} -phys_hier
commit_module_model
```

iHDB Module Model and Add-On Types

The following table lists the most frequently used iHDB module model types in a 3D-IC design. Some module model types are supported as add_on models, which can be associated with its `pnr` type model.

Type	Is an add_on model?	Description
`pre_pnr`	No	pre_pnr DB written from System Planner
`eco`	Yes	eco DB written from System Planner
`pnr`	No	Innovus DB
`lef`	No	LEF abstract
`verilog`	No	Verilog netlist
`def`	No	DEF file
`gds`	Yes	GDSII file
`mmmc`	Yes	Multi-mode multi-corner definition
`spef`	Yes	SPEF file from RC extraction
`boundary_model`	Yes	Boundary model for timing analysis
`ilm`	No	Interface logic model
`oio`	No	System Planner DB for 3D-IC system top

Multi-Die Mode of the Layout View

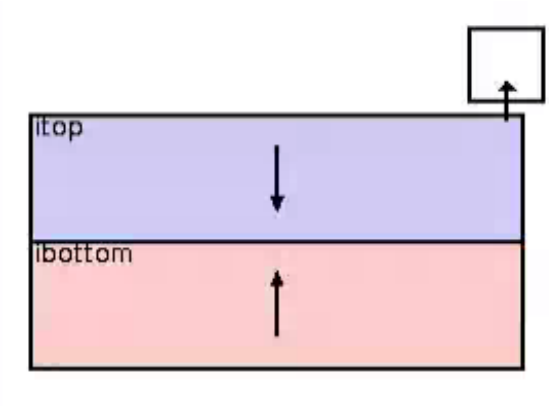
Integrity 3D-IC allows multiple dies to be loaded and displayed in the layout view. It helps you check the 3D-IC stack configuration and bump or TSV assignment. You can also use Integrity 3D-IC to debug the cross-die timing path, based on the actual placed and routed DB.

To initialize the multi-die mode from System Planner, set up each die's tech LEF, netlist, and floorplan and configure the design stack of 3D-IC. Then, use the `pull_from_planner` command to sync up the DB to the layout view.

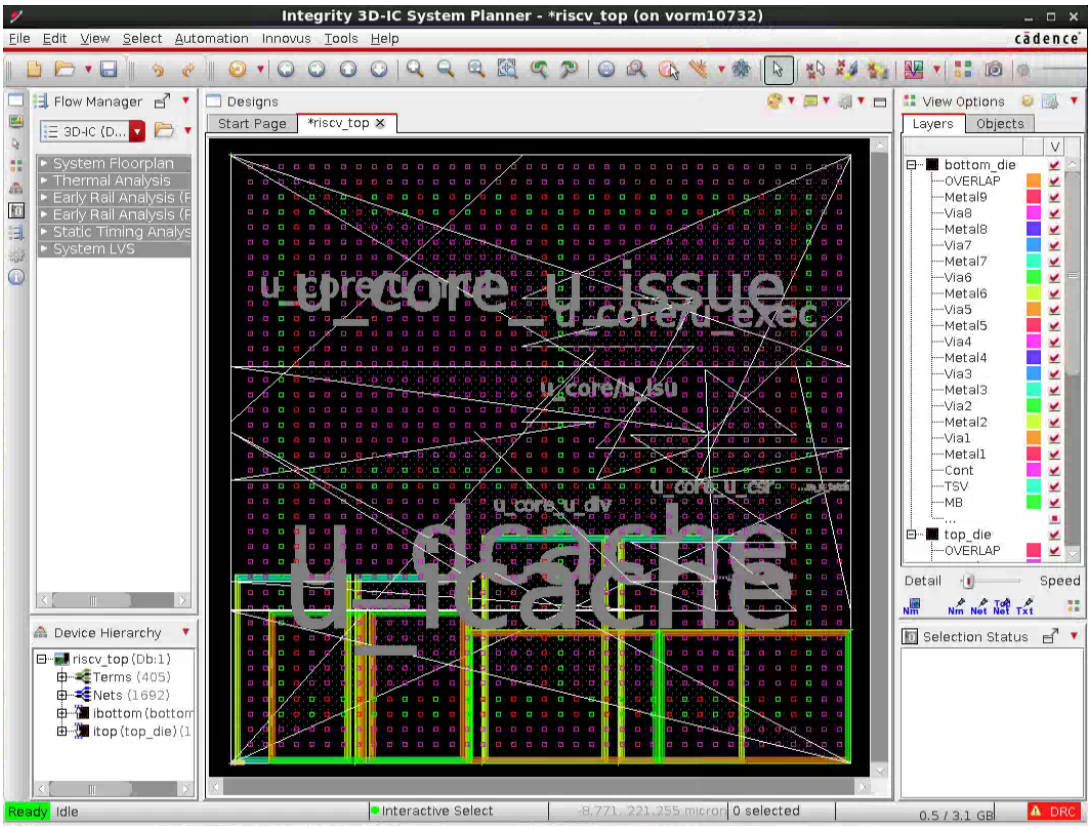
To initialize the multi-die mode from the layout view of the existing Innovus DB (pnr type) of each die, load the top-level netlist and the design stack configuration. Then, use `set_module_model -phys_hier` to specify the pnr type model of each die with the physical hierarchy view.

```
set_module_model -default_dir ./DBS

set init_verilog TOP.v

read_design_stack_config -xml TOP.xml

init_design

set_module_model -cell top_die -tag routed -phys_hier

set_module_model -cell bottom_die -tag routed -phys_hier

commit_module_model
```

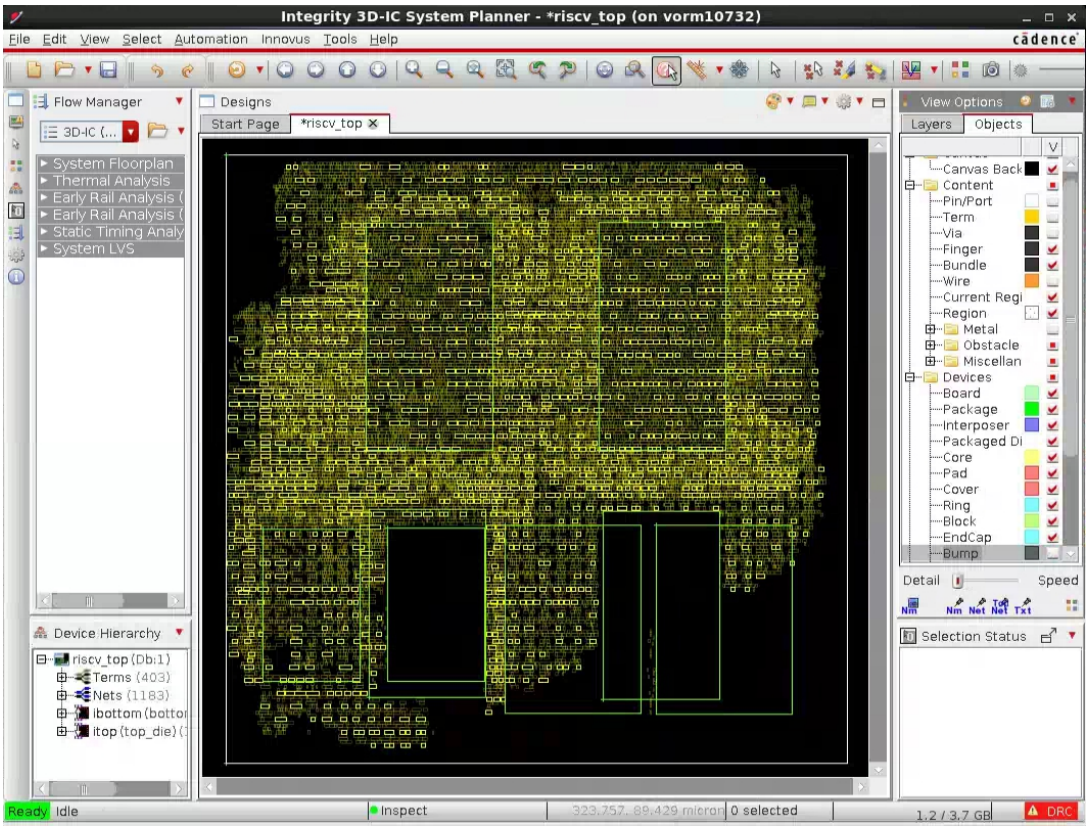
Design Stack



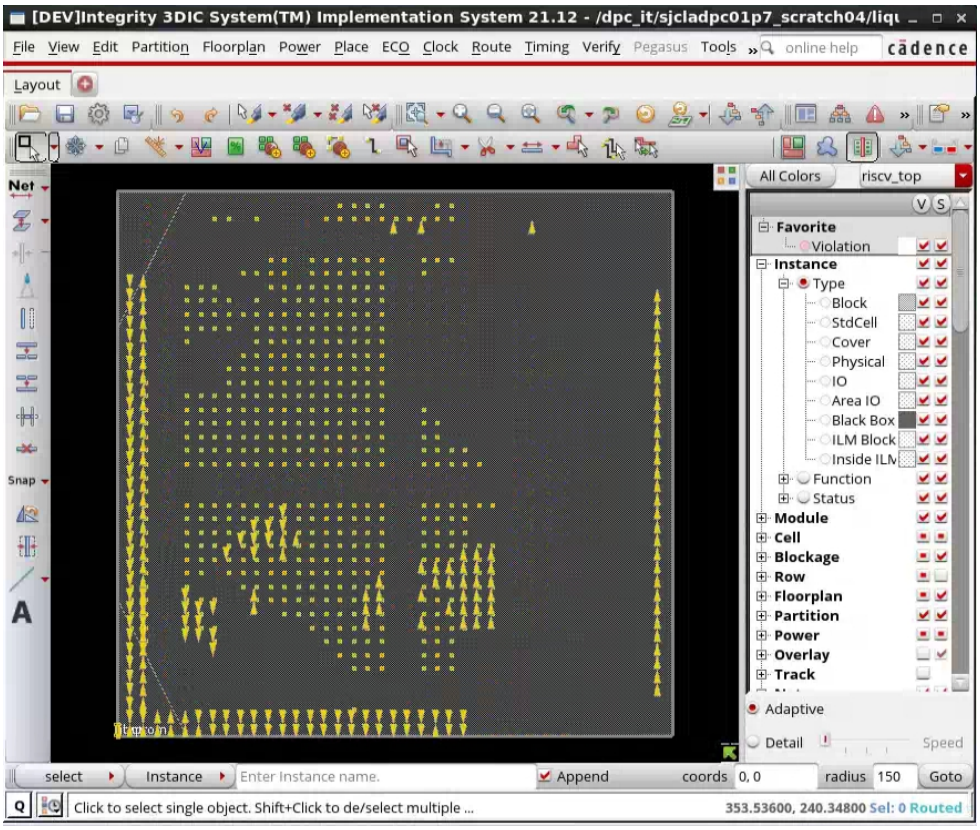
System Planner View Displaying the Die Floorplan



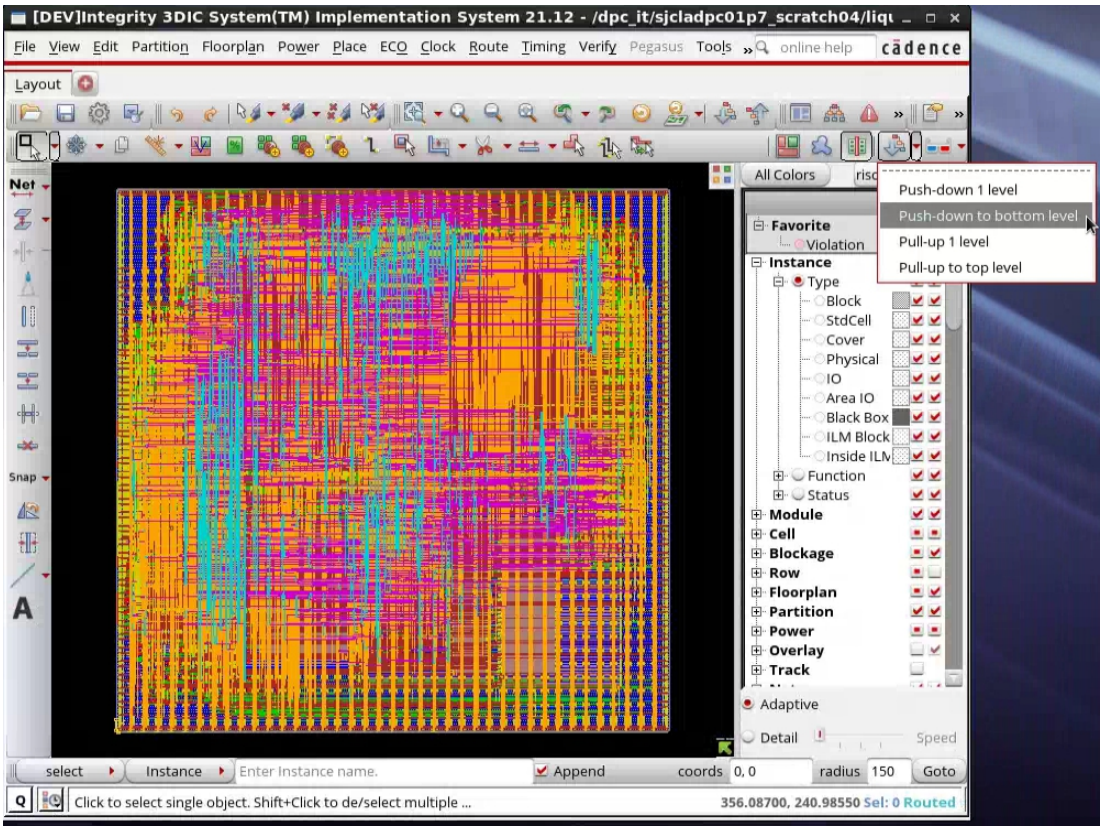
System Planner View Displaying Die Placement



Layout View (Multi-die Mode)

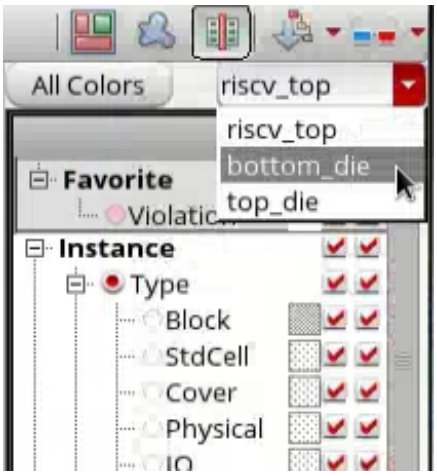


Click the *Push-down 1 level* or *Push-down to bottom level* option, as shown below, to display the physical layout of the multiple dies under 3D-IC top.



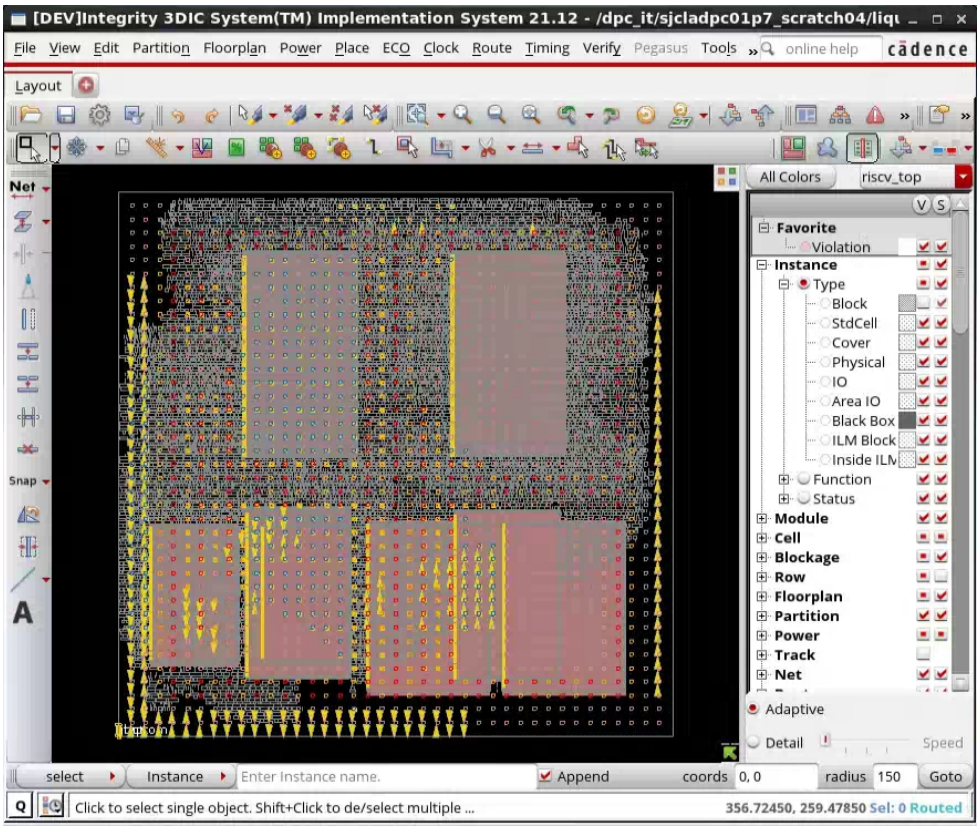
The die name pulldown list on the Layer Control panel sets the die that the panel controls.

In the following example, `riscv_top` controls the layer visibility of the 3D-IC top. As in the top netlist, there is no standard cell or wire. The top-level visibility controls only each die's LEF block instantiation and I/O terminals. `bottom_die` and `top_die` control the layer visibility of the corresponding dies.

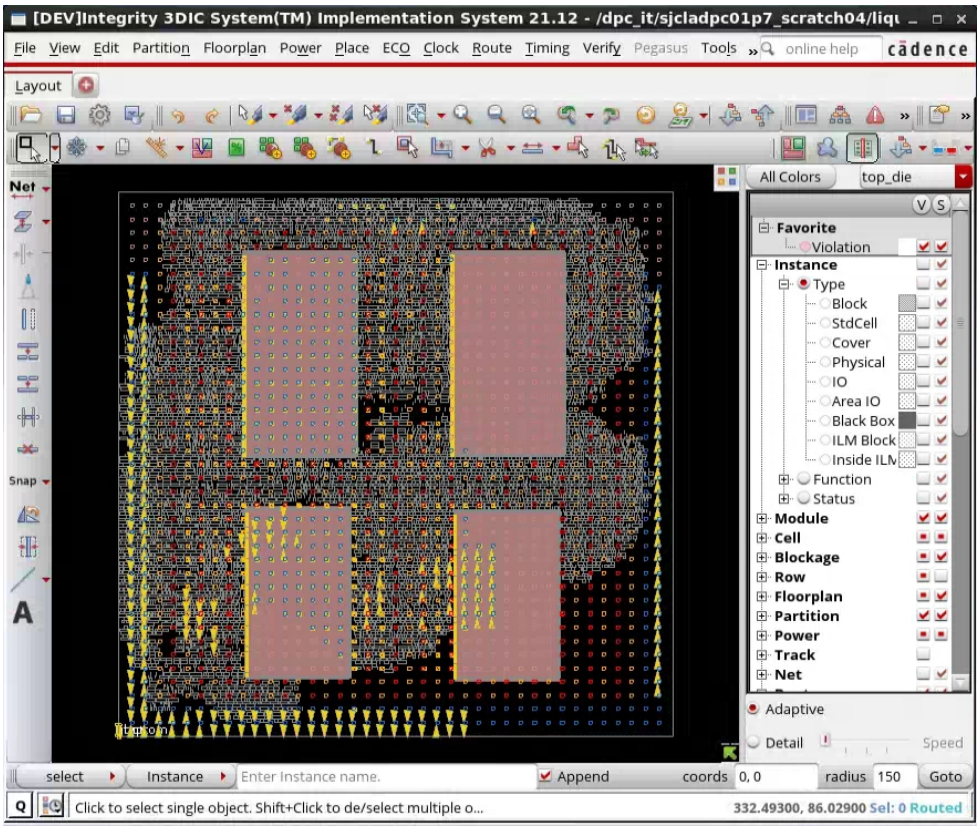


Turn off the LEF block and wires for both dies to check the macros and standard cells and bumps.

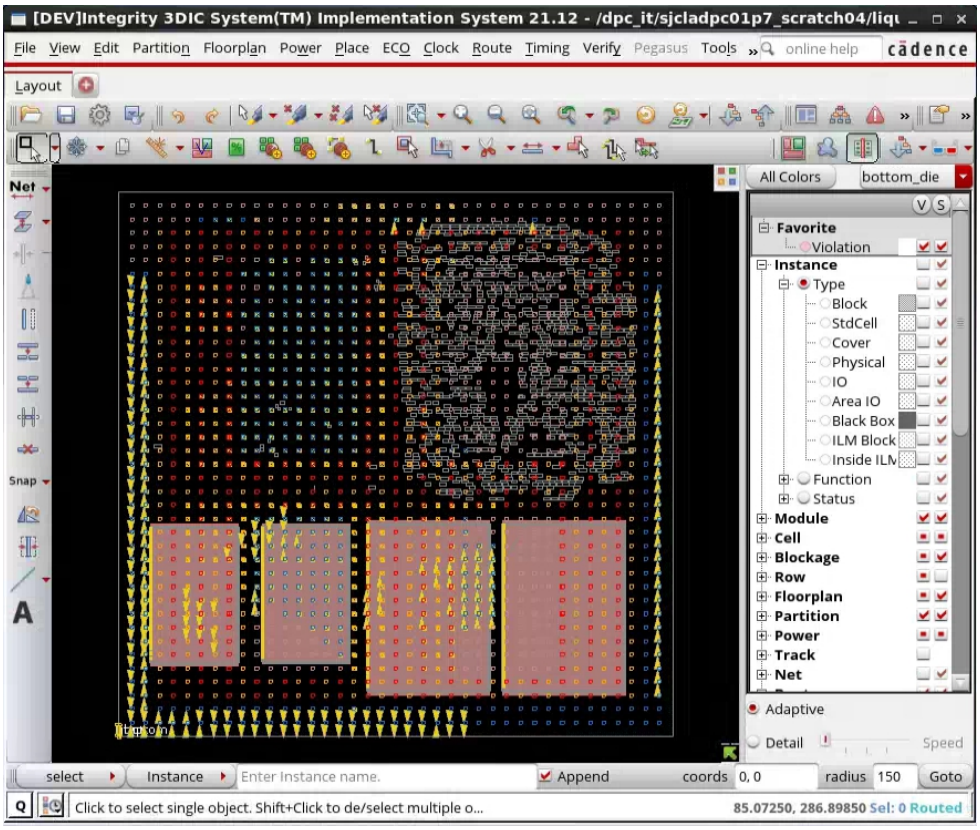
The top die's instances are flipped automatically as the die is defined as face-down in the design stack configuration.



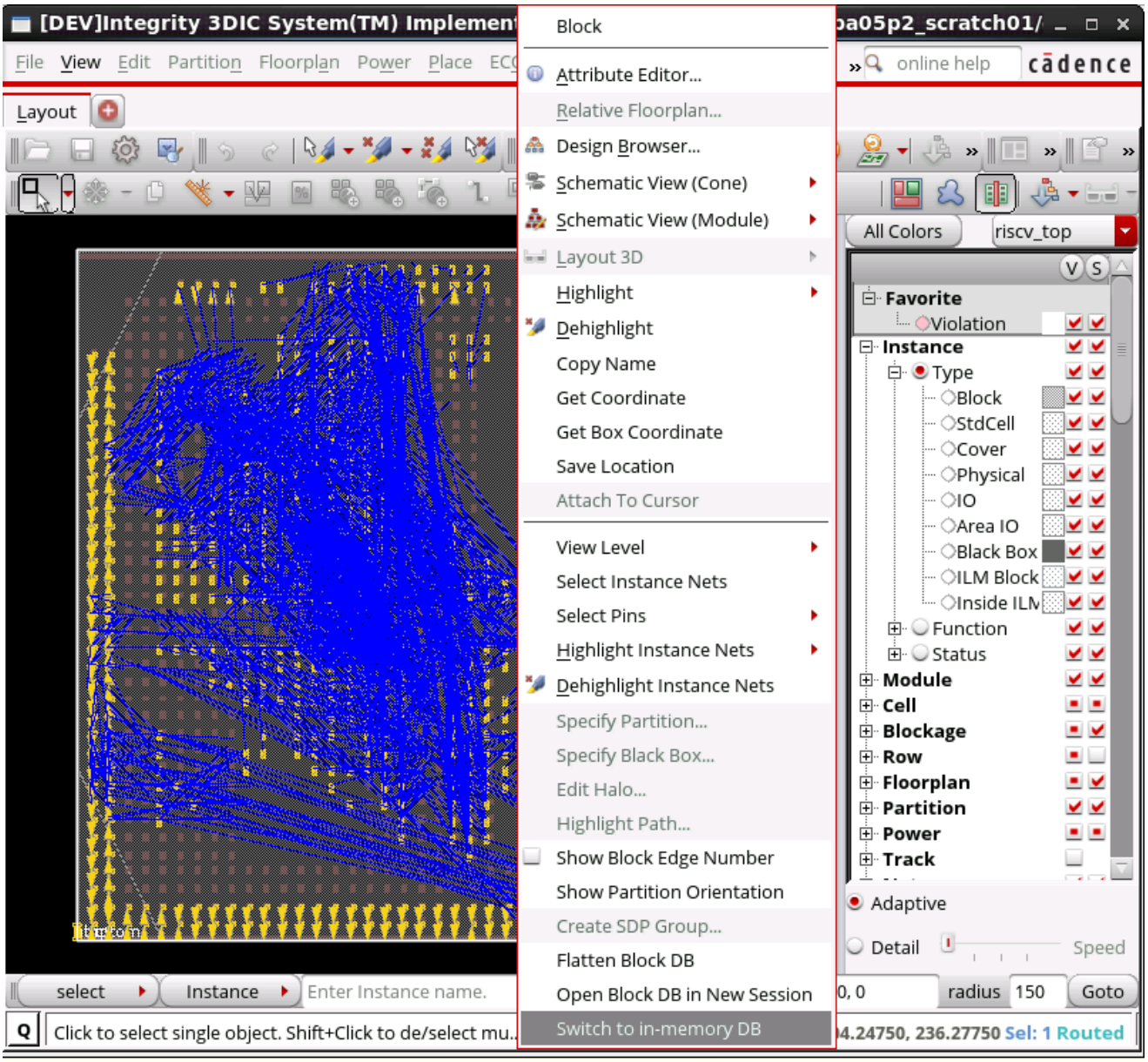
Turn off `top_die` visibility to see only `bottom_die`:



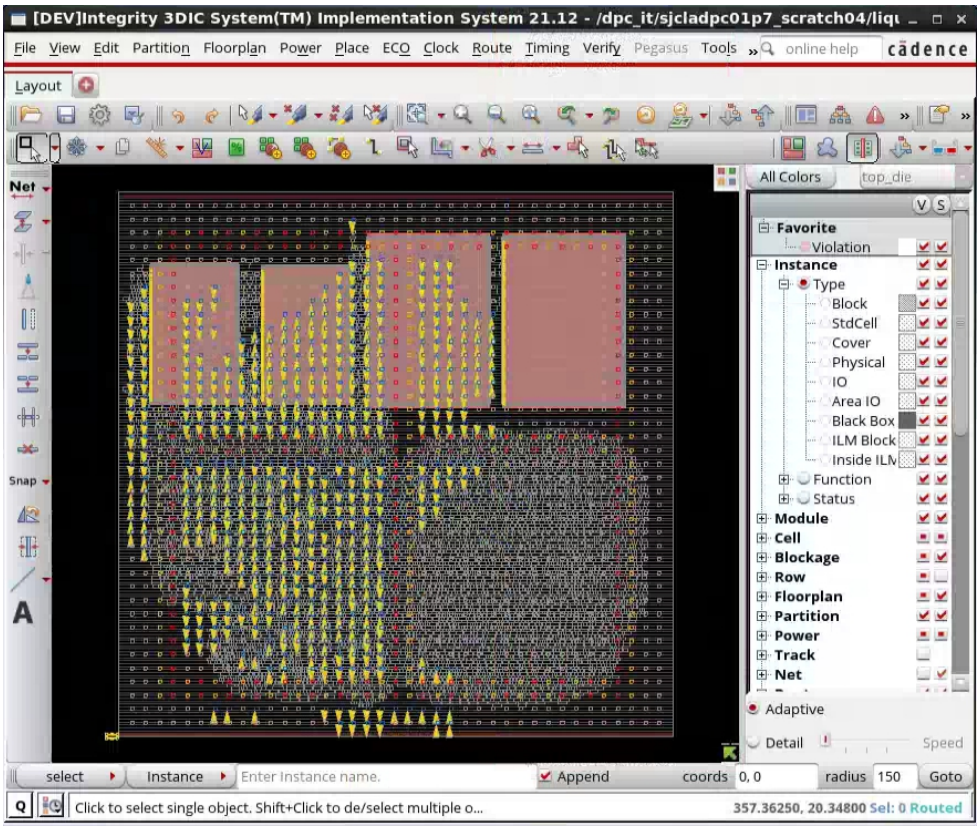
Turn off `bottom_die` visibility to see only `top_die`:



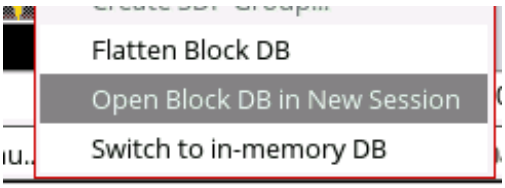
To switch the current top cell into an individual die, right click the die LEF block and then select *Switch to in-memory DB*.



The layout view switches to the selected die. The die will be displayed in its R0 orientation.



The *Open Block DB in New Session*option on the right-click menu allows you to open the die in a new session to start implementation or apply ECO.



4

Understanding the System Planner Environment

This chapter introduces the System Planner environment. It describes the terminology, data model, views, and some of the options within the System Planner environment.

- Introduction
- System Planner Terminology and Data Model
  - Terminology
- Using the System Planner Workspace
  - Using the System Planner Toolbar
  - Using the Canvas
  - Using the Layers and Objects Views
  - Using the Flow Manager
  - Using the Device Hierarchy
  - Using the Attribute Editor
  - Using the Selection Filter
  - Configuring Tool Preferences
- Report and Log Files
- Scripting

Introduction

System Planner is a very flexible tool that enables several different design methodologies used for IC and package design. The unique data structure provides a platform to build and model system-level designs consisting of multiple substrates, devices, and physical design types. Different die devices from varying process technologies can be aggregated together for system-level planning and management.

System Planner manages both the physical and logical relationships between the different devices. Contact layers and contact bumps can be created to establish and manage the physical contact points for each net. Nets can be propagated and mapped between contacted devices to form die-to-die connectivity, as well as connectivity to the top-level system design. The system-level netlist can be assembled by creating or importing netlists or pin-mapping files for each device and for the top-level design. Sensical top-level net names can be defined and managed through the design cycle. These capabilities make the System Planner environment ideal for 3D-IC system design creation, assembly, analysis, and management. System Planner is used to assemble, configure, and simultaneously manage multiple-die design projects.

System Planner Terminology and Data Model

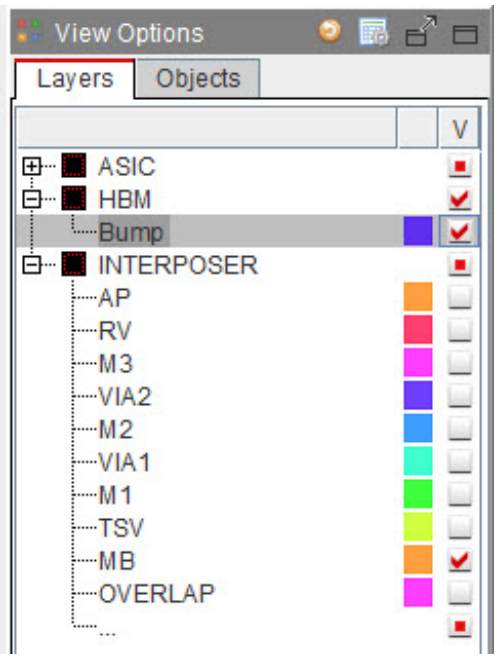
Terminology

This section describes the various objects and requirements when designing in System Planner.

Substrate

A substrate is essentially a layer stackup definition. System Planner enables designs to be assembled from multiple fab technologies, each typically with a unique substrate. Multiple devices can also use the same substrate.

The first step in creating a design in System Planner is to define one or more substrates. A substrate can consist of a single bump pad layer, as shown below under *HBM*. The substrates are displayed and managed in the *Layers* view.



Pin, Pad, Bump, Padstack, and TSV

System Planner supports a variety of pin types and terminologies such as pad, bump, and TSV to enable support for both package and IC design tools. Depending on whether you are designing for IC design tools or package design tools, you will be creating new pins as *Pin* or *Device* type. To enable you to use the different processes and terminologies, the various dialog boxes will use terminologies to suit the methodology.

- Use pins of type *Device* when targeting IC design tools. A bump is defined as a device in the System Planner data model. When using *Device*-type pins, bump terminology is used.
- Use pins of type *Pin* when targeting the package tool, APD Plus. When using*Pin*-type pins, pad terminology is used. A padstack is a pad definition for any given substrate. Pad sizes are defined on the desired layers in the substrate. Padstacks are assigned to pins. Both can be created and managed through a variety of methods described in this guide. Each device can use multiple padstacks.

Through Silicon Vias (TSVs) are defined as pins with a padstack definition.

Device

All design objects are considered devices in System Planner. Devices are displayed and manipulated in the Device Hierarchy panel. The device type dictates how the device is handled and displayed in the design. Typical device types include DIE, MACRO, GROUP, CORE, COVER, BUMP, and BUMPPAD. Package type devices are also described in the guide. The device type is displayed and can be modified in the Attribute Editor panel.

Device Template

Each device is created with a unique device template. Devices can be copied, and the copied devices share a common device template. Any changes to one device will affect the other copies as well. The device template is displayed and can be renamed in the Attribute Editor.

Contact Layer and Contact Pads

System Planner manages the contact point relationship between different devices using contact layers. Contact layers can be created using several methods. Use the *Edit – Contact Layer* and *Edit – Design Stack* commands to define or modify contact layers. A contact layer is also defined automatically when a hierarchy is created in the Device Hierarchy. Typically, when a contact layer is being created, the option to create aligned contact pads on the contacted device is presented.

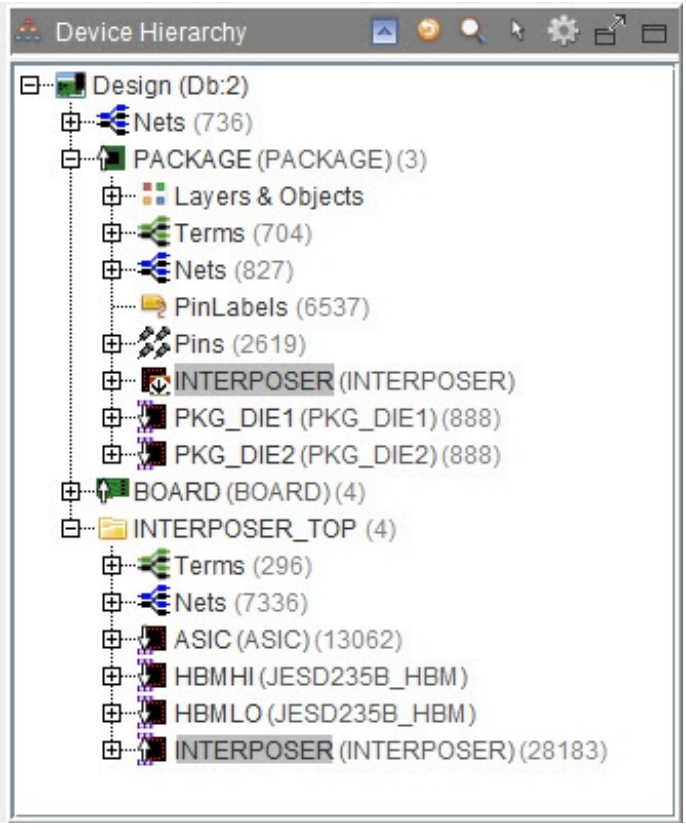
Creating contact layers also affects the netlist of the design. A contact layer can propagate the nets from the source device to the target device. There are features available to propagate and/or map the nets into the contacted device.

Contact Device

Creating a contact device creates a copy of the device in the Device Hierarchy, consisting of only the pad contact pattern and net definitions. This is sometimes useful when multiple copies of a device are needed. Using a contact device enables a single contact layer to be used for all the copies. You can copy contact devices while using the original device to manage all the copies.

> ⚠ Contact devices are not recommended for 3D-IC designs because the original device is often located outside the design hierarchy, while the contact devices are within the hierarchy. These devices add a layer of complexity to the design and interfere with the Verilog netlist requirements. Use these devices primarily for package design applications. You can achieve contact device functionality by copying the device itself within the Device Hierarchy. The copied devices share a common device template.

Contact devices are also useful when a co-design project has multiple levels of hierarchy and a device needs to exist in more than one hierarchy, as shown below with the INTERPOSER contact device under the PACKAGE hierarchy.

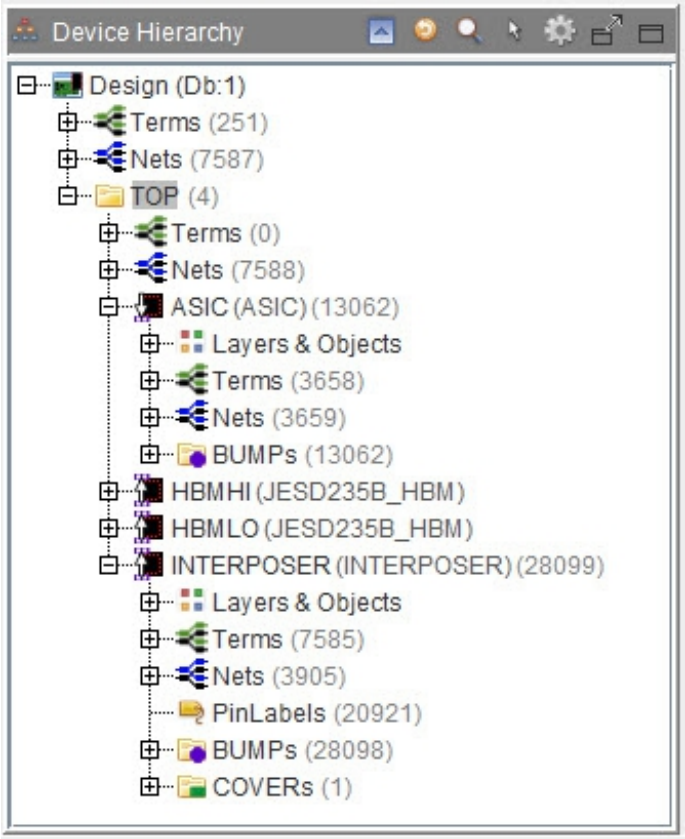


In the above example, the INTERPOSER and PACKAGE designs each have their own hierarchy, while the INTERPOSER device is also represented in the PACKAGE design hierarchy using a contact device.

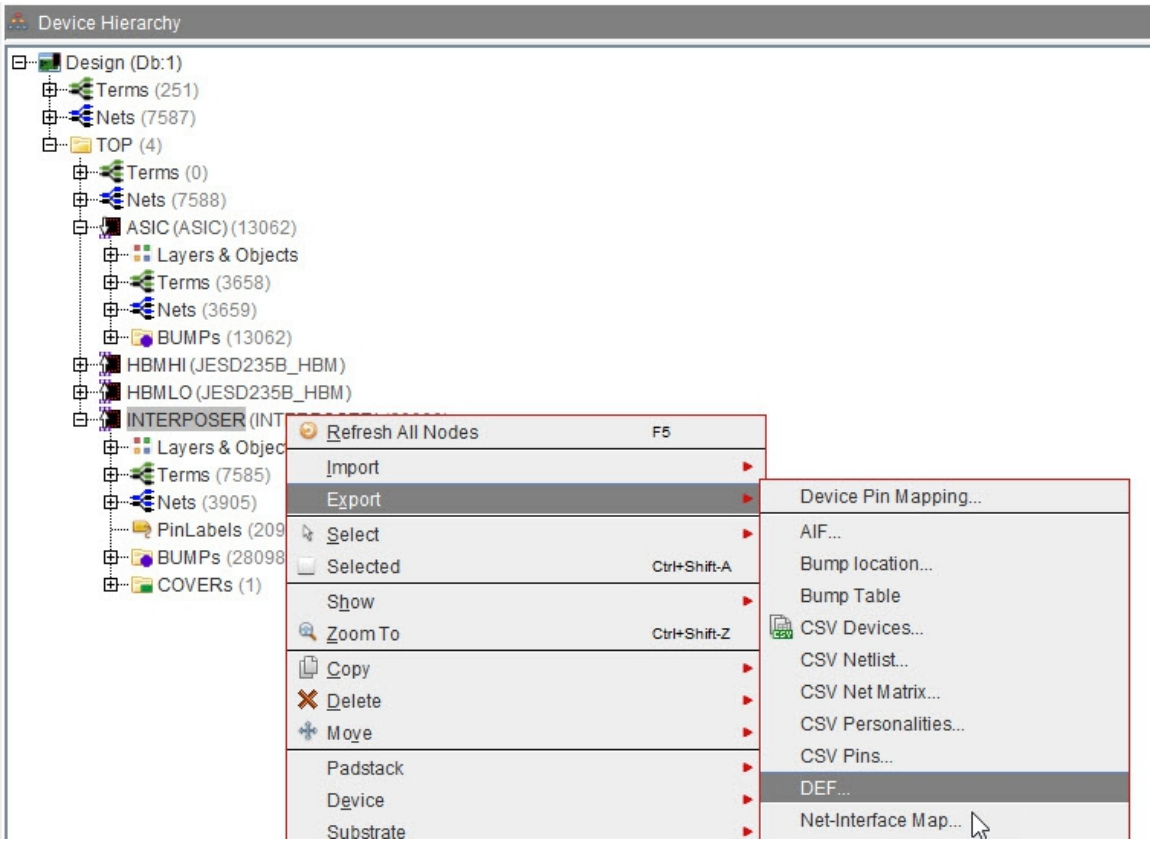
Contact devices are sometimes used when a device's contacts are buried in a package design layer within the substrate. Contact devices are represented with a unique icon , as shown above for the INTERPOSER device under PACKAGE.

Device Hierarchy

The Device Hierarchy panel is used frequently during a typical design process. It displays the devices in the design as well as the hierarchical relationship between the devices. It also displays the terms, nets, and pins/bumps that exist for each device, as shown below.



Select any device and right-click to access an object-oriented set of commands to perform on the selected device. These commands include many import and export options to manage the implementation requirements of the device, as shown below.



Netlist

System Planner has a robust set of capabilities for netlist generation, mapping, and management. Each device has a set of named pins that can equate to the net names for the device. Device net names can also be derived from imported sources, such as Verilog or CSV pin lists. They can also be derived by creating levels of device hierarchy and then propagating nets up and down the levels. There are net prefix and mapping options to map devices with differing net names across contacted devices. Contact layers can also be used to propagate and map nets from one device to another.

Integrity-3D-IC designs require a Verilog term-based netlist. This netlist requirement dictates different considerations and formats. These Verilog requirements are described in the "*Defining or Importing Netlist Connectivity*" section of the Working with Designs in System Planner chapter. Users familiar with Verilog should be familiar with the requirements for terms and macros.

Package designs prepared for APD Plus do not use Verilog netlists. Therefore, the netlist and data format requirements are less stringent.

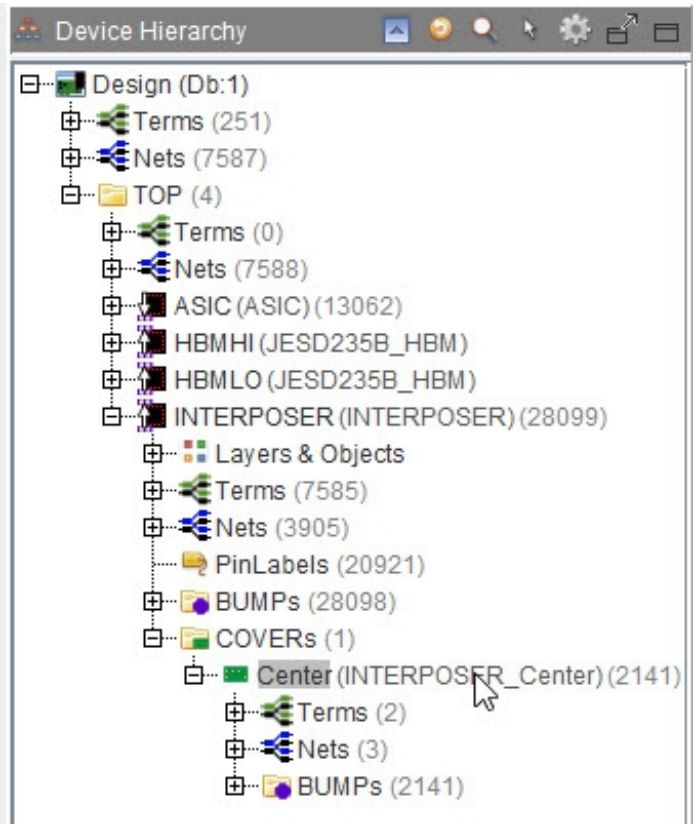
Cover

A cover is a group of pins that exist in a device and is referred to in this guide as a cover device. Cover devices are typically used to define a power and ground pattern of pins but can also be used for other purposes.

Using the *Create Pins or Devices Pattern* command will create a *COVERs*-level of hierarchy under the device in the Device Hierarchy panel. Multiple cover patterns can be created and managed under the *COVERs* folder. Cover devices can be manipulated independently of the device.

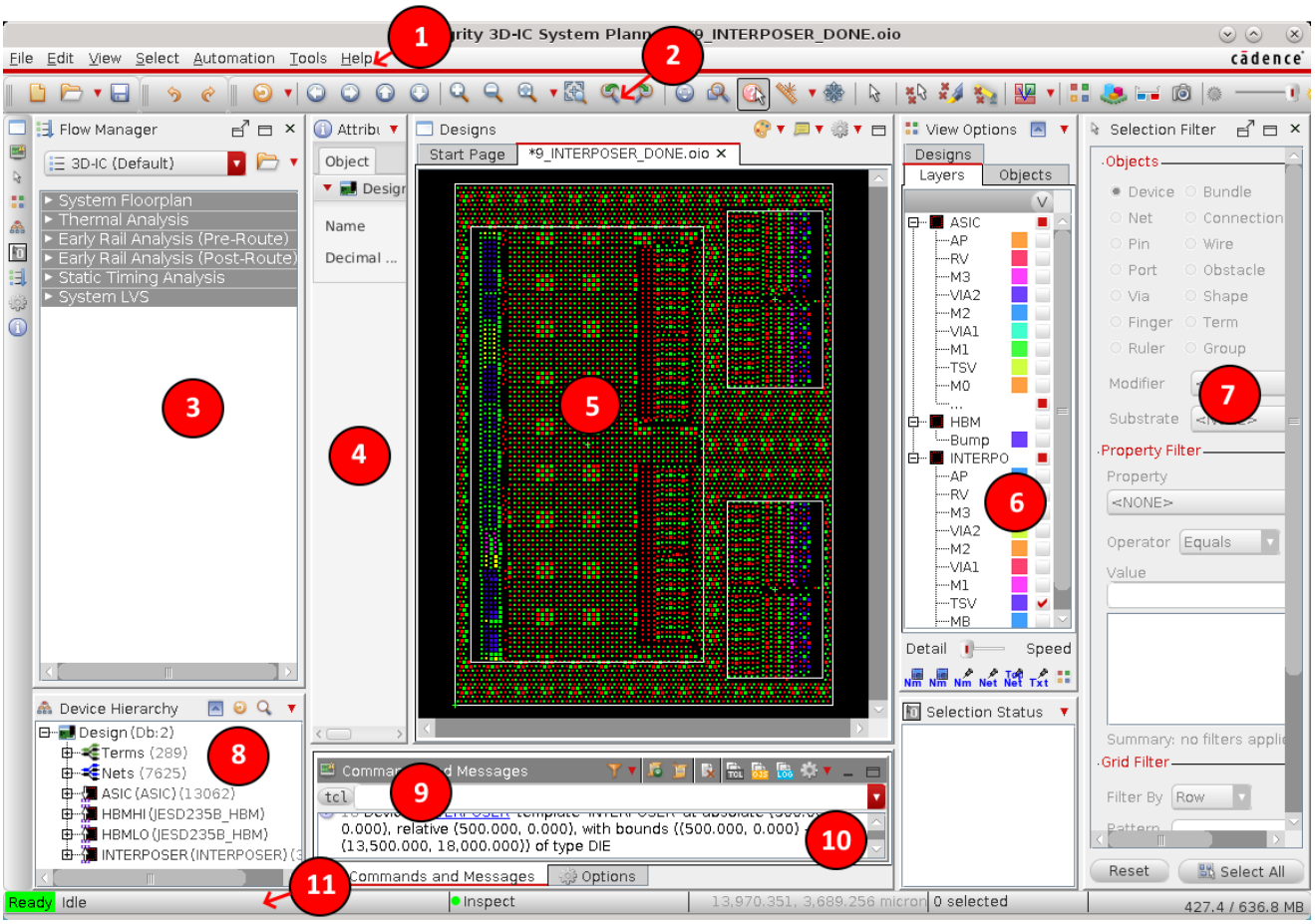
A cover device creates an additional level of netlist hierarchy under the device, as shown below. This may not be desirable, so cover devices can be dissolved into the parent-level netlist. The *Ungroup* command can be used to move the cover pins into the parent-level device netlist hierarchy and dissolve the cover device. Once ungrouped, the pins are no longer contained in or manipulated as a group. They become individual device pins.

For more details, refer to the "*Creating Bumps and TSVs*" section of the Working with Designs in System Planner chapter.



Using the System Planner Workspace

The following image shows various System Planner interface elements, which are identified in the number chart shown below. These elements are described in detail within this section.

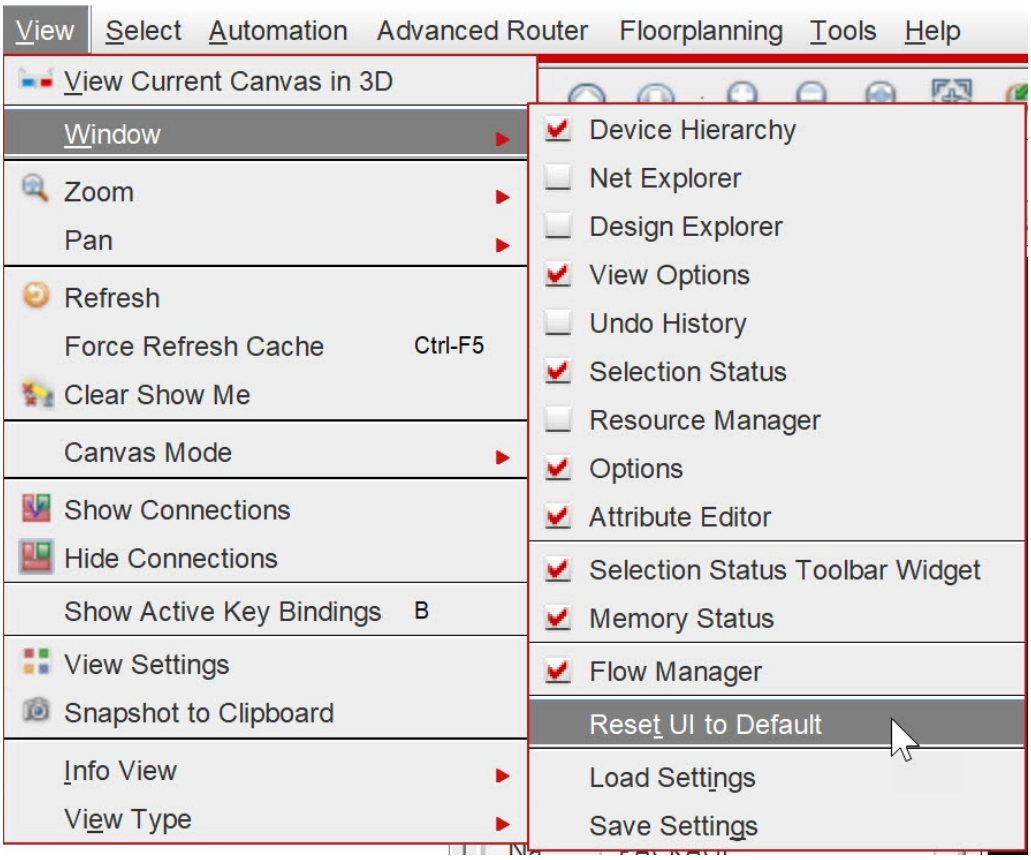


#	Element
1	Main Menu
2	Main Toolbar
3	Design Flows
4	Attribute Editor
5	Canvas
6	Layers and Objects Filter
7	Selection Filter
8	Design Hierarchy
9	Command Prompt
10	Messages Window
11	Status Bar

Many of these elements have common Windows widgets in the upper right corner to close, minimize, maximize, or float the element.

The default view configuration is designed to offer the best access to design information and command controls. System Planner stores the last used view configuration for each user and will attempt to open each time with the last used configuration.

The display of the various elements can be controlled by selecting or deselecting the check boxes in the *View – Window* submenu, as shown below. You can choose *View – Window – Reset UI to Default* to return all windows to the default state.



Using the System Planner Toolbar

The System Planner toolbar provides quick access to commonly used commands and view controls. The toolbar icons are described below and are numbered from left to right.

Theme	#	Toolbar Icon
File/Design	1	New - Open a new design
	2	Open - Browse to select and existing design to open
	3	Open Pulldown Menu - Select from previously opened designs
	4	Save - Save active design
Action	5	Undo - Undo last step
	6	Redo - Redo last step
Pan	7	Refresh - Redraw canvas
	8	Pan Left - Pan canvas to the left
	9	Pan Right - Pan canvas to the right
	10	Pan Up - Pan canvas up
	11	Pan Down - Pan canvas down
Zoom	12	Zoom In - Zoom the canvas in
	13	Zoom Out - Zoom the canvas out
	14	Zoom Fit - Zoom fit the canvas
	15	Zoom Selected - Zoom the canvas to fit selected objects
	16	Zoom Previous - Zoom the canvas to the previous zoomed area
	17	Zoom Next - Zoom to canvas to the next stored zoomed area
Modes	17	Navigation Mode - Continuous panning mode
	18	Zoom Mode - Continuous zoom mode
	19	Inspect Mode - Continuous click to inspect objects mode
	20	Ruler Mode - Continuous click to draw rulers mode
	21	Interactive Move Mode - Continuous click to move mode
Select	22	Interactive Select Mode - Continuous click to select mode
	23	Clear Selection - Deselects all selected objects
	24	Clear Highlight - Clears all highlighted objects

Using the Canvas

The System Planner canvas is the main graphic display area and is used to interact with the physical design.

Mouse Actions in the Canvas

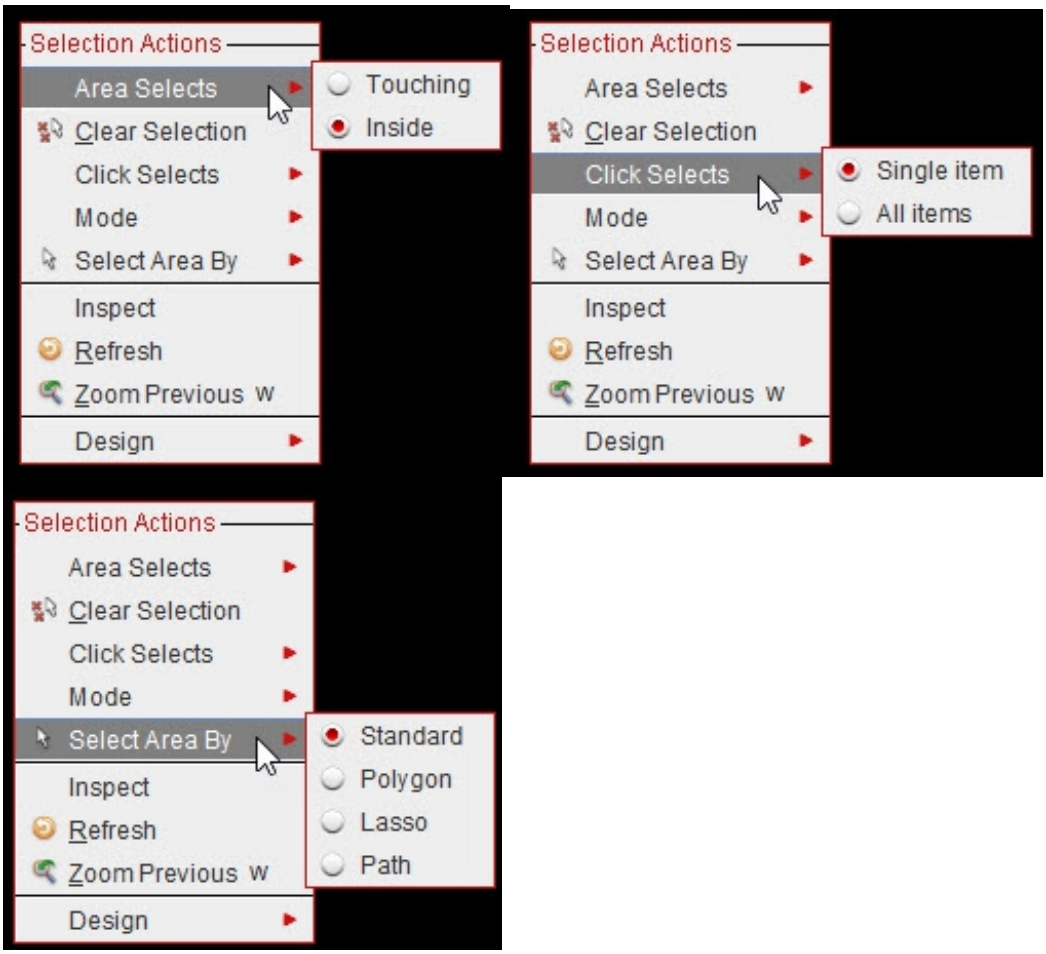
Selecting Objects

Rectangles can be drawn on the canvas to select multiple objects in the range.

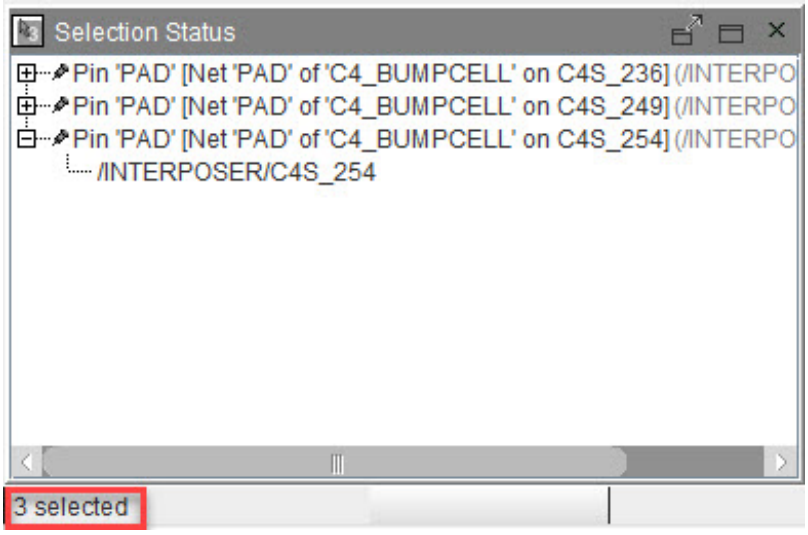
> ⓘ Use `Shift` and `Ctrl` to select multiple objects.

Objects must be visible on the canvas to allow them to be selected. Be sure to adjust the layer visibility in the *Layers* view accordingly.

There are additional selection controls under the right-click menu on the canvas, as shown below. These options are self-descriptive and are not explained in detail.



The number of selected objects is displayed in the status bar at the bottom and in the*Selection Status* panel, as shown below.



Right-Click Menus

Right-clicking an object opens a pop-up menu of available commands for that object.

The available commands are sometimes determined by the view in which the right-click was performed. Some of these menus extend with multiple levels of expanded selections. All the available commands are not covered in this guide.

Zooming and Panning

To zoom in/out, you can do the following:

- Use the mouse wheel to zoom in and out.
- Use the right mouse button (RMB) to draw a rectangle to zoom to that specific area.
- Use the middle mouse button (MMB) or wheel for panning.

The mouse zoom speed can be set on the *User Interface* page of the Preferences form (*Tools – Preferences*).

Setting the Default View Mode

The *Default View Mode* setting controls the default mode to use when the cursor is on the canvas. The modes dictate the usage and display of the cursor on the canvas. Once a mode is selected, it remains active until another mode is selected. If undesired cursor behavior is detected, click the *Interactive Select Mode* icon on the toolbar. The *Interactive Select Mode* is used primarily with typical 3D-IC design activities.

You can change the default view mode on the *Design* page of the Preferences form (*Tools – Preferences*).

The available *Default View Mode* settings are:

- *Inspect* – Provides information for selected objects.
- *Ruler* – Measures the distance between selected points.
- *Navigate* – Pans to browse the design.
- *Zoom Region* – Zooms into and out of areas using zoom rectangles.
- *Interactive Select* (preferred) – Selects objects.

You can switch to any of these modes by selecting the corresponding icon on the toolbar.

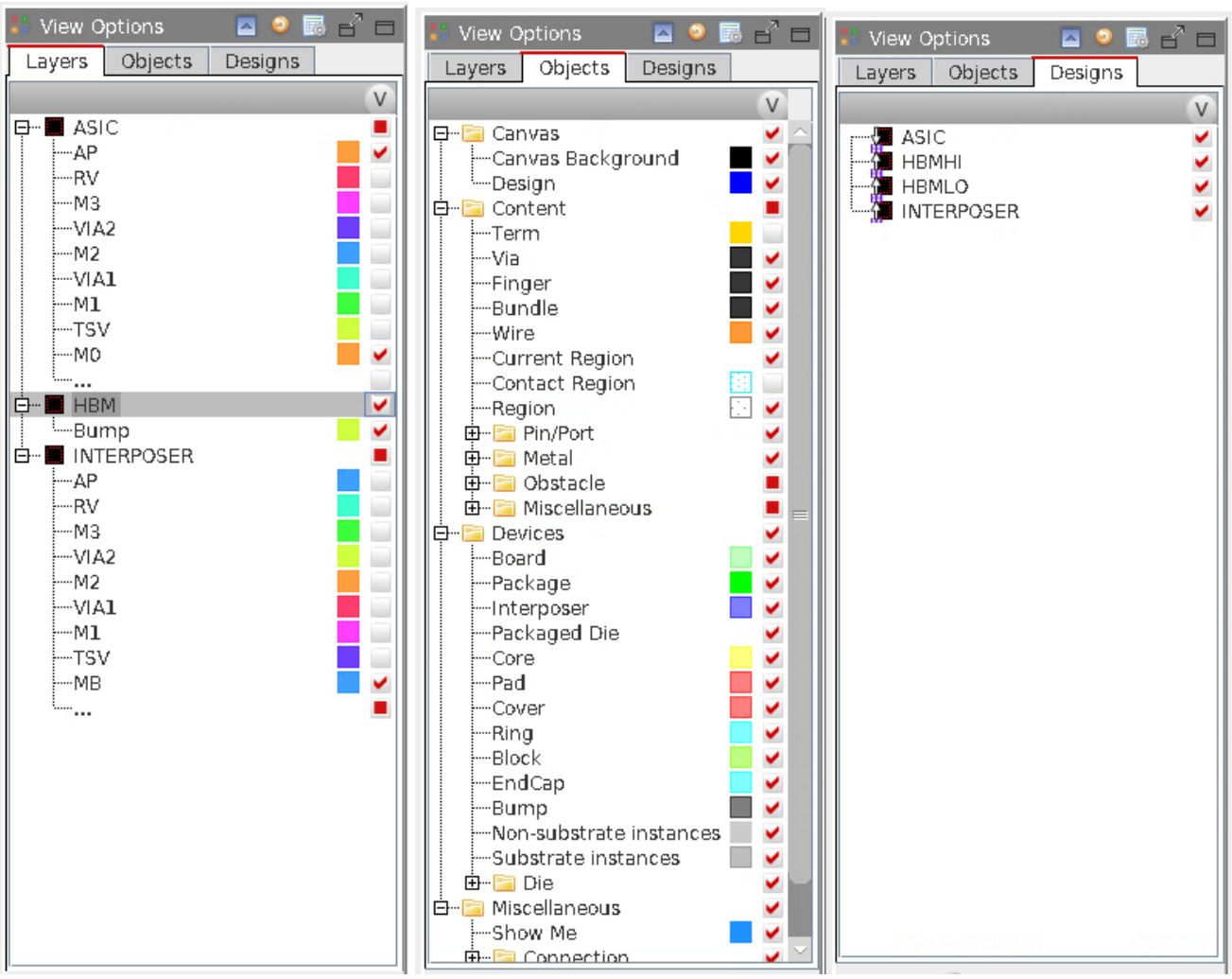
Setting the Canvas Background Theme

System Planner includes the *Classic Dark* and *Classic Light* canvas background themes. This guide uses the *Classic Dark* background theme in the images.
To change the background to a dark theme, select *Classic Dark* from the drop-down menu of the *Change the canvas theme* icon ( ) at the top of the canvas.

Using the Layers and Objects Views

The View Options panel is displayed by default to the right of the System Planner canvas. It has three tabs, as shown below:

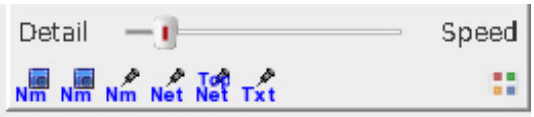
- *Layers* - Allows you to control the visibility of the layers in various substrates
- *Objects* - Allows you to control the visibility of different types of objects in the canvas
- *Designs* - Allows you to control the visibility of the designs in the canvas



Selecting the various layer or object toggle switches will turn on or off the display on canvas for that layer or object. In the *Layers* view, next to each substrate, a top-level toggle controls visibility for all the layers in that substrate. Clicking on the colored squares will present a color palette to enable the selection of a new color for that object or layer.

Additional right-click capabilities are provided for each substrate in the*Layers* view. These include quickly toggling on/off the visibility for all layers, substrate renaming, and layer modification using the *Layers Table*. The Layers Table can also be opened by clicking on the*Layers Table* icon ( ) in the *Layers* view header.

Additional view options are provided under the*Layers* and *Objects* views, as shown below.



The *Detail <> Speed* slider enables configuration of the level of detail displayed in the canvas

versus the speed it takes to refresh the screen. Setting it all the way to the left will enable pins to be displayed with decreased zoom levels.

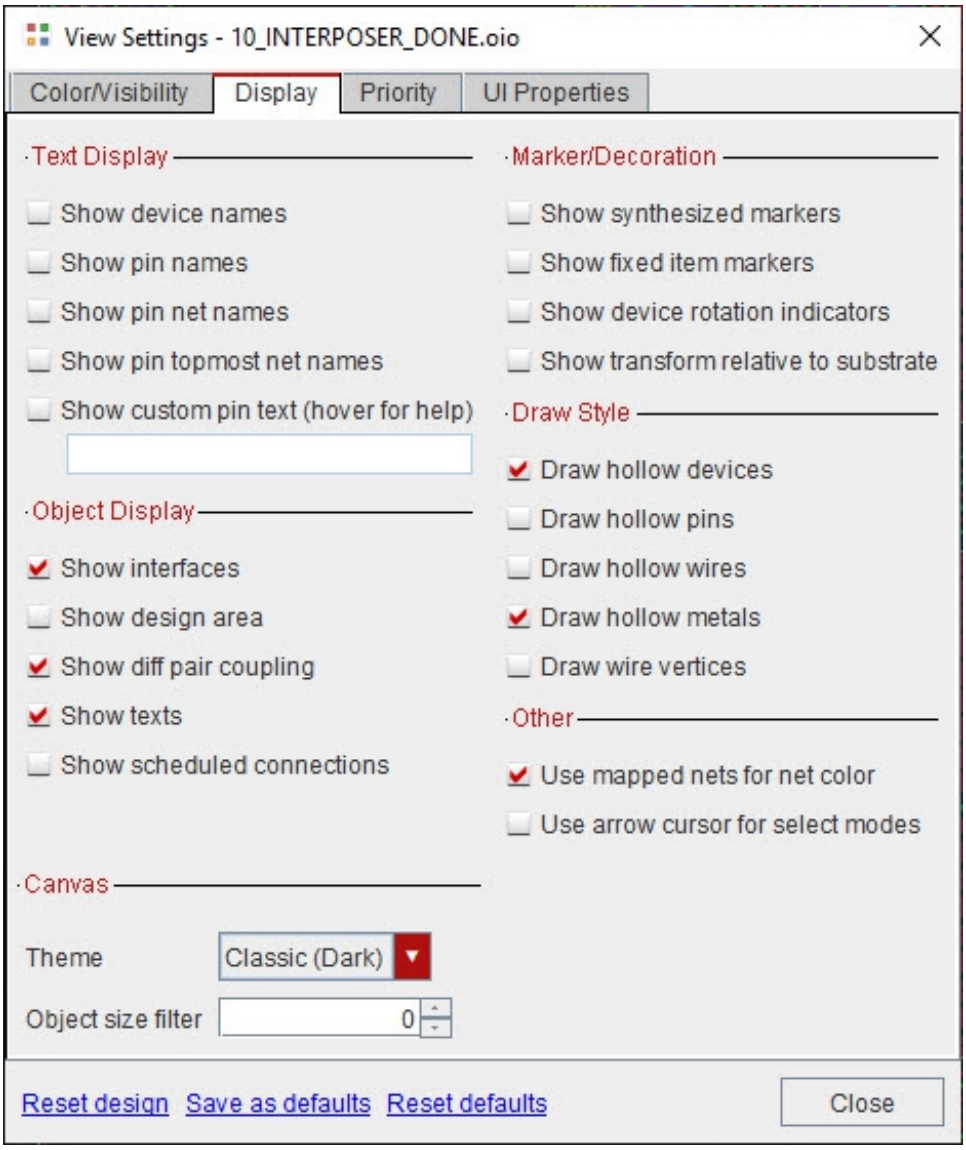
The additional view toggles are described below:

- – Toggles the display of device names.
- – Toggles the display of device template names.
- – Toggles the display of pin names.
- – Toggles the display of net names on the pin.
- – Toggles the display of the top-level net name.
- – Toggles the display of the custom pin text.

Configuring View Settings

Additional canvas visibility control options are available in the View Settings form, which can be accessed by selecting the *View – View Settings* command or the *View Settings* toolbar icon ( ). The view settings control display behavior defaults, display priorities, and name text display. Many of these are related to other methodologies or tool capabilities and are not covered here. Default settings are recommended for *Priority* and *UI Properties* to ensure proper tool behavior.

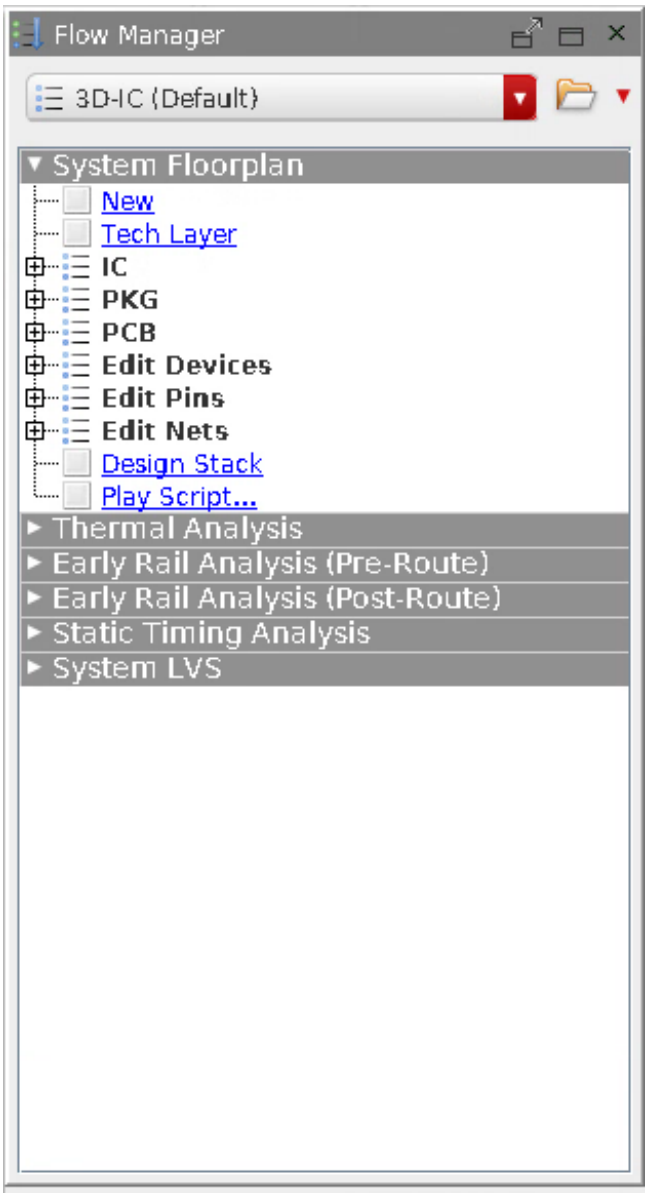
The settings most relevant to 3D-IC designs are under the *Display* tab, as shown below. Most of these are self-explanatory and are not covered in detail in this guide.



Using the Flow Manager

The Flow Manager panel is used to organize and present commands relevant to certain types of design objects or design tasks. These commands are also found in the *Edit* menu. The Flow Manager is used to simplify access to commonly used commands.

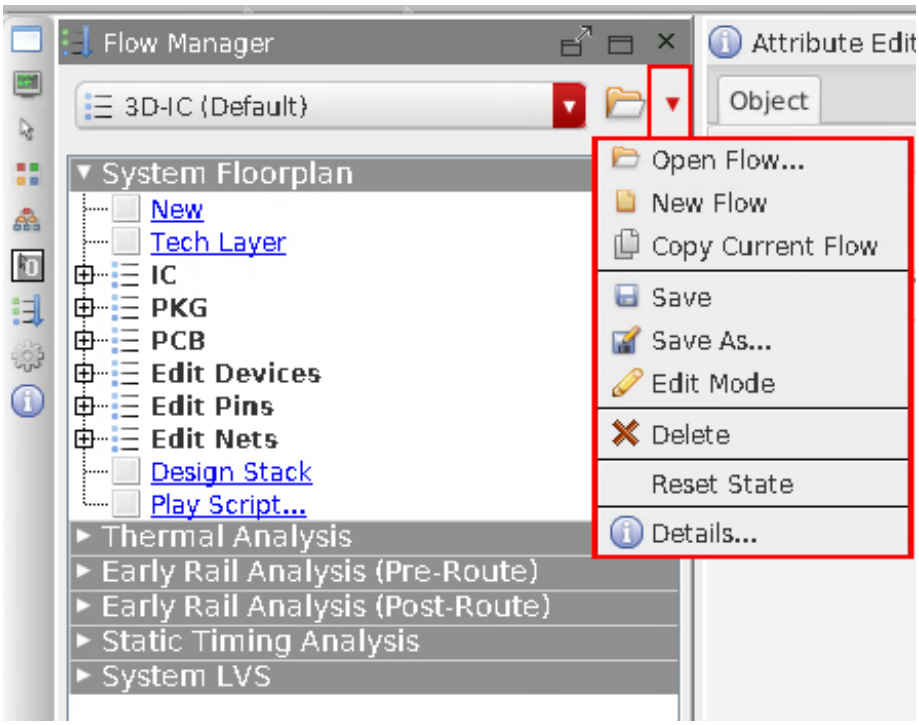
Double-clicking the *Hierarchy Expand* icon ( ) presents the commands relevant to that type of object, as shown below. Double-clicking expanded menus collapses them.



Once a command has been executed from the Flow Manager, a green check mark is displayed in its check box. This is helpful when using design flows with a specific set of steps. The *Reset State* command in the *More* ( ) menu can be used to clear these markers.

Using Custom Design Flows

The *More* ( ) menu in the Flow Manager header provides capabilities to import new designs and manage and modify design flows, as shown below.



Custom design flows are used for different types of design methodologies. New design flows may become available in future releases of System Planner to enable specific functions.

Using the Device Hierarchy

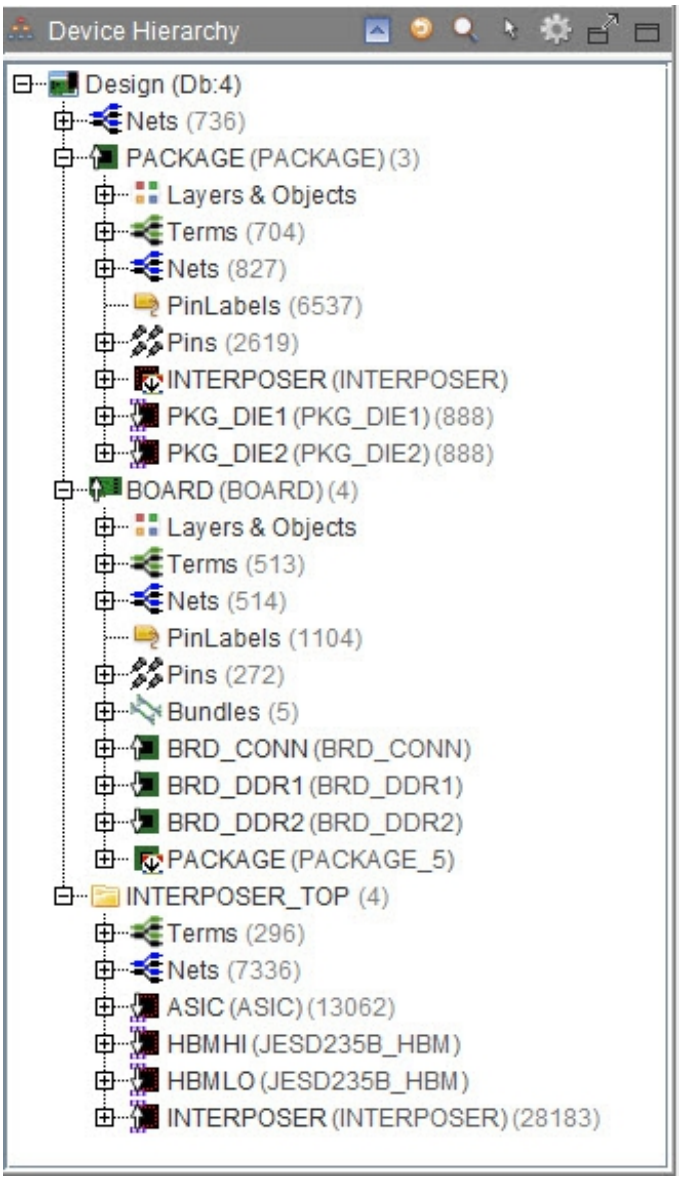
The Device Hierarchy panel is used to display, select, and manipulate objects in the design. It is the most commonly used view when performing typical design activities.

Understanding Devices

All main design elements such as die, interposers, packages, bumps, and boards are considered devices in System Planner and are displayed in the Device Hierarchy.

Actions can be taken on any device, by selecting it and using the right-click menu commands. Data can be imported and exported for each selected device. This enables a system design to be aggregated, while still maintaining each device independently. The terms, nets, pins, bumps, and other aspects of each device are displayed under the device, as shown below.

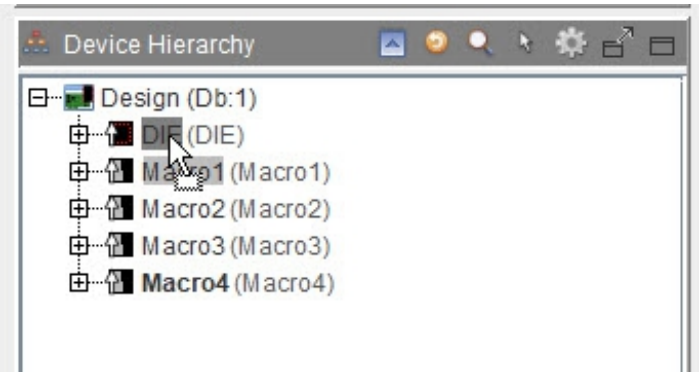
.



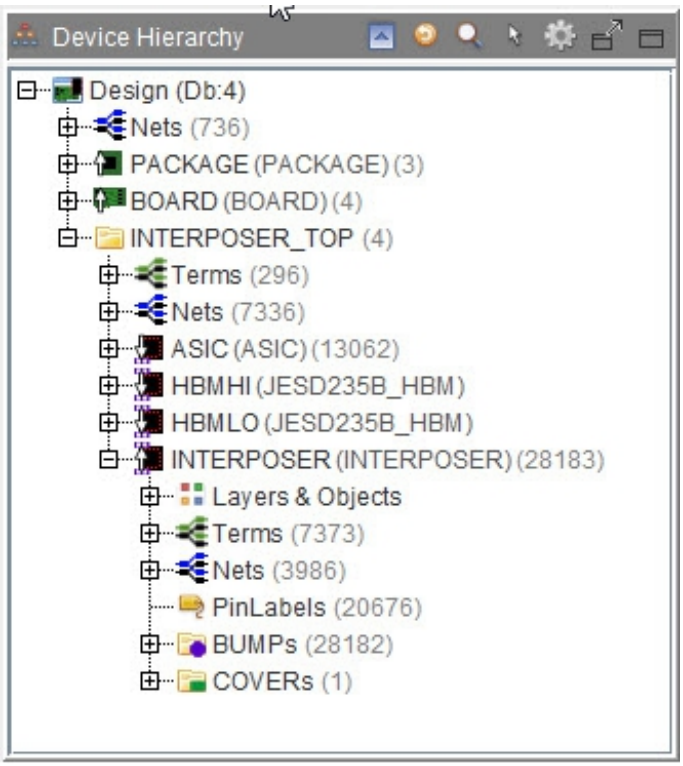
The image above represents a complex design with an interposer design, a package design, and a board design, all represented in a single, system-level System Planner project. Note the levels of hierarchy – the *Design* (top-level), the *PACKAGE*, the *BOARD*, and the *INTERPOSER*. The devices that are contained in each of these levels are displayed under them, as seen with the INTERPOSER with an ASIC and two HBMs. By expanding the hierarchy, the *Terms*, *Pins*, *Nets*, *Bumps*, and so on for each device are displayed.

Creating Hierarchy

Relationships can be established between devices by creating levels of hierarchy. Children of a parent die are all associated with the parent die design and are contained in its netlist.
Hierarchy can be created by right-clicking a device and choosing*Move – Device to New Parent*or by dragging the device an placing on top of another device in the Device Hierarchy, as shown below.



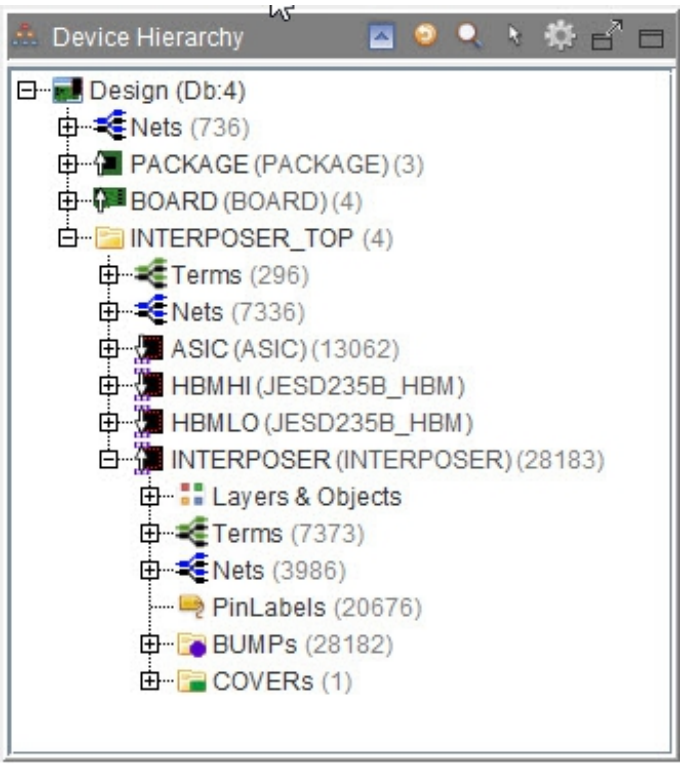
⚠ Designs that are expected to be implemented with Innovus need the hierarchy to be created slightly differently than those targeting APD Plus or Virtuoso. Innovus designs with multiple dies require a top-level netlist that instantiates each of the dies in the design, as shown in the image below. Notice the extra *INTERPOSER_TOP* level of the hierarchy. Each die is instantiated as a macro in the Verilog netlist, including the INTERPOSER.



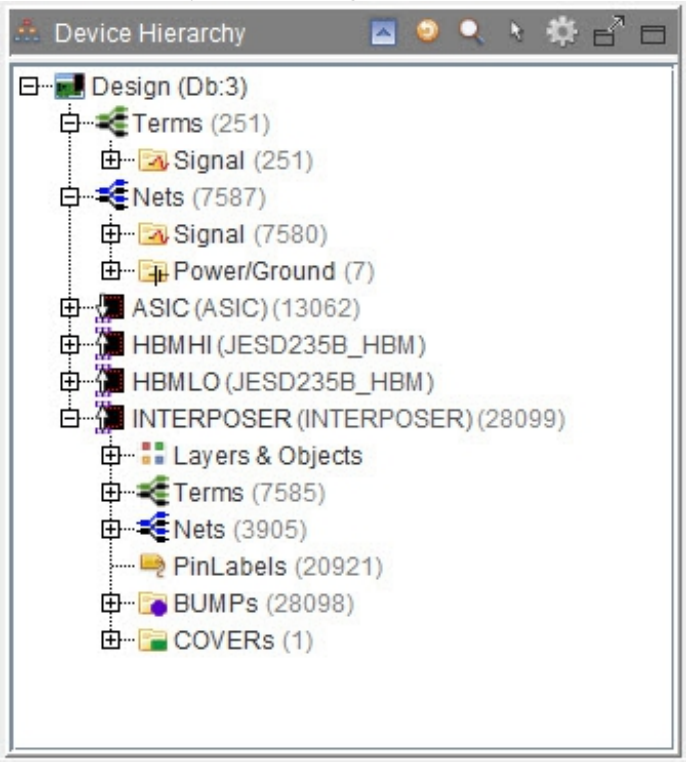
When importing Verilog netlist(s) into an existing System Planner design with existing devices, the module names and hierarchy must match exactly with the System Planner device names and structure, including the top-level module name.

Using the Group Devices Command to Create Hierarchy

A parent-level of the hierarchy can be created for any selected group of devices by using the*Edit – Devices – Group Devices* command. The INTERPOSER_TOP group shown below was created using this method. The group will initially be created with a default name of *UserGroup*. This name can be changed by selecting the group and using the Attribute Editor to enter a new directory and template name.



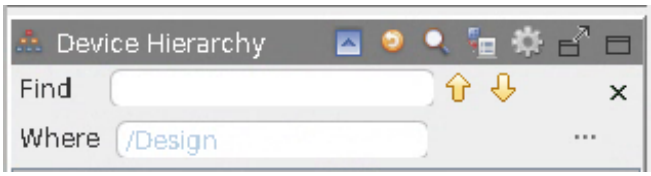
The top-level *Design* can also be used as the top-level netlist, as shown below. Notice the*Nets* and *Terms* directories are populated with top-level nets. The top-level terms define the external nets for the system design, such as those that connect to the package design, for example.



In the above image, this INTERPOSER design project uses *Design* as the top-level netlist. All dies are on the same level as the Interposer in the hierarchy. This must be represented like this in the Verilog netlist. The default top-level *Design* name can be defined when creating a new project.

Using the Device Hierarchy Toolbar

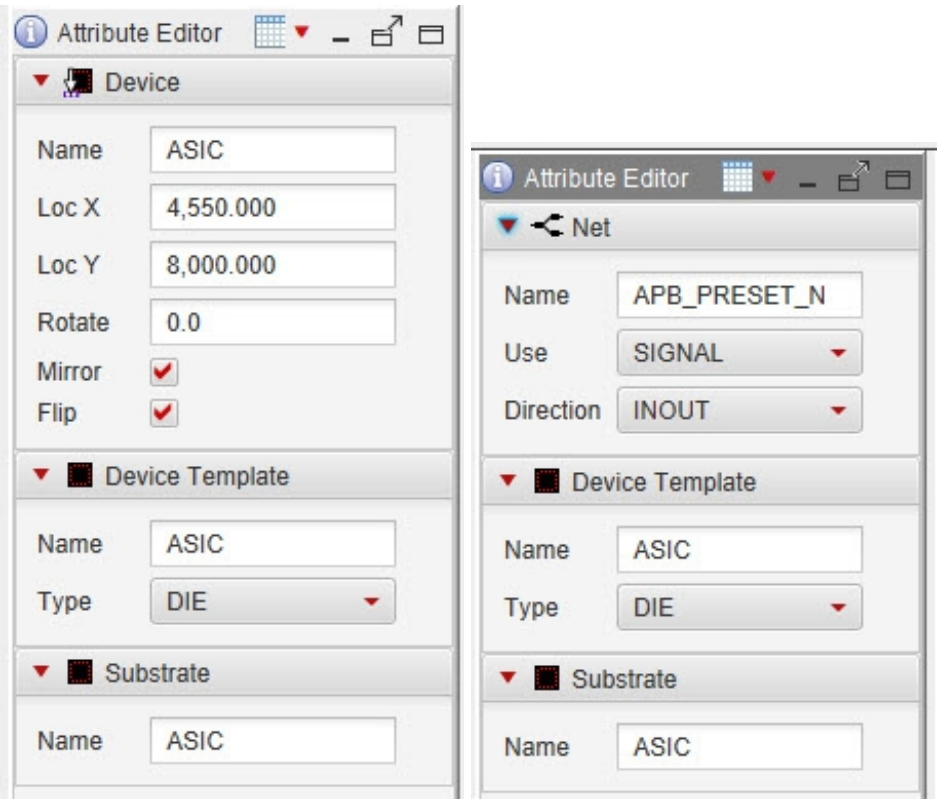
The Device Hierarchy panel contains toolbar icons in the view banner offering view-specific functionalities, as shown and described below:



- *Collapse Nodes* – Collapses all expanded hierarchy and returns the view to a default state.
- *Refresh All Nodes* – Redraws hierarchy to reflect any name changes or device manipulation.
- *Search* – Presents a *Find* field to enter the text string to be searched and a *Where* field to enter the location to be searched, as shown above.
- *Search/Pin Selected* – Scrolls to display the pre-selected objects.
- *Properties* – Provides configuration options for the view.

Using the Attribute Editor

The Attribute Editor is used to display and modify attributes for any selected object, as shown below.

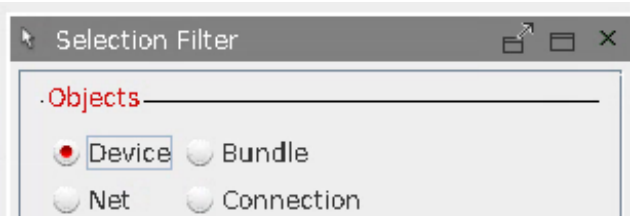


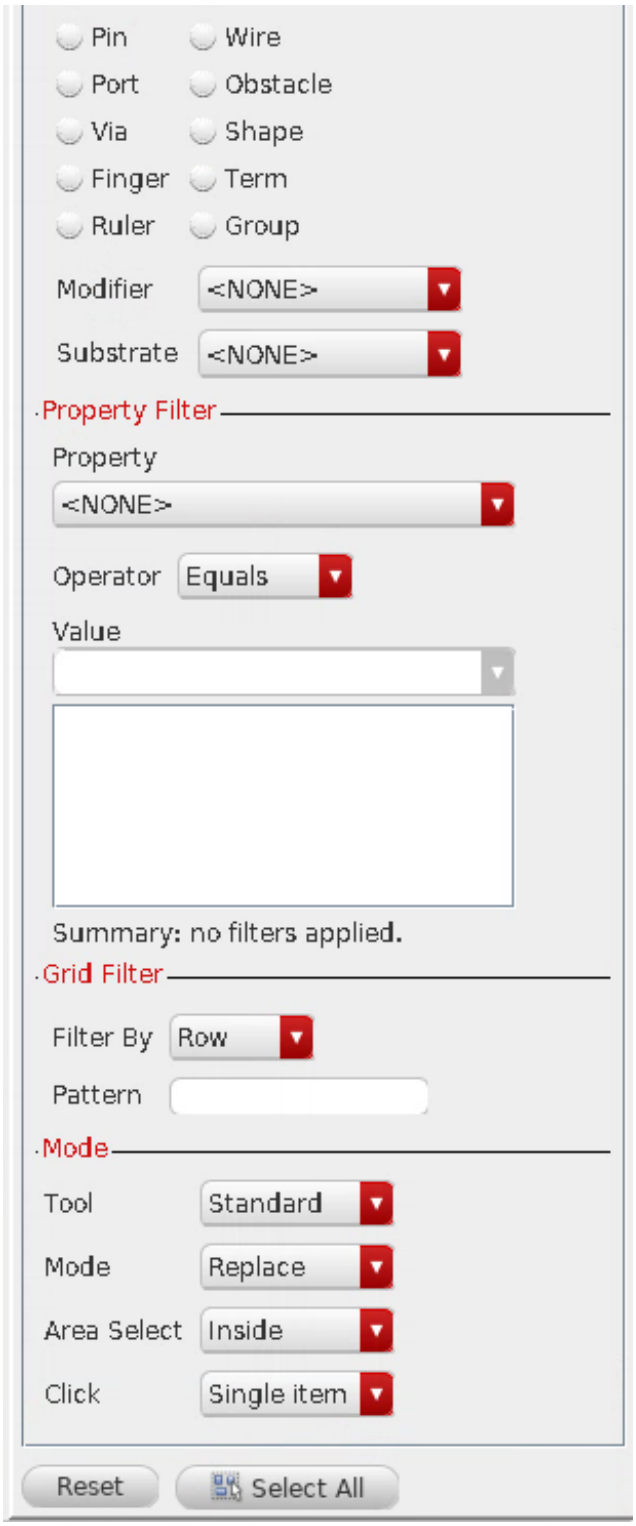
The attributes are fairly self-explanatory, and most are not covered in detail here. The object attributes can be modified by typing new values or selecting from the drop-down menus. Ensure the intended object is selected and displayed in the Attribute Editor before making any changes. Click the *Refresh All Nodes* icon on the Device Hierarchy toolbar to see any recent changes.

The *Device Template – Type* value can be used to change the type of object for a selected device. Changing a DIE to a MACRO is an example of the functionality of this attribute. This will also change the device type icon displayed in the Device Hierarchy. Many of the values available under the *Type* drop-down menu are not used in traditional 3D-IC design.

Using the Selection Filter

The Selection Filter is displayed on the far right of the System Planner window and allows specific object types to be selected on the canvas. A variety of filters allow you to make a specific selection of desired objects.





The various *Selection Filter* options are described below:

- *Objects* – Defines the type of objects to be selected.

- *Modifier* – Defines specific selection conditions for the selected object type.
- *Substrate* – Defines the specific substrate from which to select objects. Leaving it set to <NONE> will select objects from all substrates.
- *Property Filter* – Enables filtering based on specific properties and values for the objects to be selected. Many of these values are not used in traditional 3D-IC design and are not covered here. A common property filter used for selecting pins is *Net Class* to filter Signal, Power, and Ground pins.
- *Mode* – Enables various methods for selecting objects. The values are self-explanatory and are also available from the right-click menu on the canvas. The *Intersect* mode is used to enter additional filter modifiers. To use the *Intersect* mode, first select objects using the primary filter settings. Then set the mode to *Intersect*, set secondary filter settings, and select the objects again to apply the second filter settings to the first set of selected objects.

Configuring Tool Preferences

Configuration settings for the System Planner environment are available under *Tools -Preferences*. Many of the settings are for other methodologies and legacy related. The settings relevant to 3D-IC design are covered below.

General

- *Application Configuration Directory* – Specifies the alternate System Planner configuration directory. This controls how System Planner behaves and is intended for custom environment developers. Recommended to leave this as the default value.

The other general settings are self-explanatory.

User Interface

- *GUI Startup Directory* – Defines where to invoke System Planner and to write .log and .cmd files.
- *Default Net Filter* – Defines the type of nets to display in the Device Hierarchy. These options are also available in the Device Hierarchy right-click menu for *Nets*.
- *Mouse wheel zoom speed* – Controls the zoom in/out speed when using the mouse wheel.

Other settings are either self-explanatory or not applicable.

Report and Log Files

System Planner creates the following three files in the GUI startup directory each time it is invoked:

- `integrity_planner.msg.log` - Contains the information displayed in the *Commands and Messages* view within the System Planner session.
- `integrity_planner.cmd.log` - Contains a history of the commands run during the System Planner session.
- `integrity_planner.tcl.log` - Contains a tcl version of the history of the commands run during the System Planner session.

An extra number is appended after the file name if there is a name collision. For example, `integrity_planner.msg.log2`, `integrity_planner.cmd.log2`, and `integrity_planner.tcl.log2`.

Scripting

System Planner scripts can be written in Java. They can also be created interactively within the tool by using the *Tools – Record Script* command. They can be replayed by using the *Tools – Play Script* command. This guide does not contain detailed scripting instructions.

5

Working with Designs in System Planner

This chapter sequentially describes the various steps involved in the typical die design process. The System Planner features for each step are described in detail.

- Defining Substrates
  - Creating Substrates Interactively
  - Creating Substrates by Importing a LEF Tech or CSV Layers File
- Creating Padstacks - Pads, Bumps, and TSVs
  - By Importing LEF Pads and Pad Macros
  - By Using LEF Macros
  - By Creating Bumps Interactively
  - Viewing and Modifying Padstacks
- Creating Devices
  - By Using the Create Device Option
  - By Importing a Die Abstract File
  - By Importing LEF/DEF Files
  - By Importing a Die CSV File
  - By Importing a Pins List CSV File
  - By Importing a System Planner "OIO" Project File
  - By Importing a Die Text "TXT" File
  - Changing the Origin of a Device
  - Changing the Size of a Device
- Floorplanning Devices
  - Flipping Devices
  - Moving Devices

- Copying Devices
- Using the Device Table Editor
- Using the Attribute Editor
- Importing a CSV Devices File
- Using the Ruler Mode to Measure

- Defining the Design Stackup
- Creating Contact Layers and Contact Pads
  - Creating Contact Layers
  - Creating Contact Pads
  - Managing Contact Pads
  - Creating Contact Devices
- Running Inter-Substrate Connectivity Checks
  - Exploring Results
- Defining or Importing Netlist Connectivity
  - Understanding the Top-Level Design Requirements
  - Work at an Abstract Level using Black Box Die Macros
  - Creating the Top-Level Design Hierarchy
  - Importing Verilog Netlists
  - Creating Connectivity with Term Files
  - Importing Term List and Term Map Files
  - Creating Terms Using the Netlist Editor
  - Exploring Connectivity with the Netlist Editor
  - Exploring Connectivity with the Graph Net/Term Commands
  - Working with Power and Ground Nets
  - Defining Differential Pairs
  - Defining Net Personality Groups
- Creating Bumps and TSVs

- Using the Bumps for All Layer Shapes Command
- Creating a Pin Pattern for Bumps and TSVs
- Assigning Power and Ground Nets
- Assigning Nets to Bumps
- Creating TSVs Using the Fanout Vias Command
- Optimized Bump Assignment with the Connection Optimization Command
- Interfacing with Other Implementation Tools
  - Creating Die Symbols for Package Design
  - Exporting Implementation Files

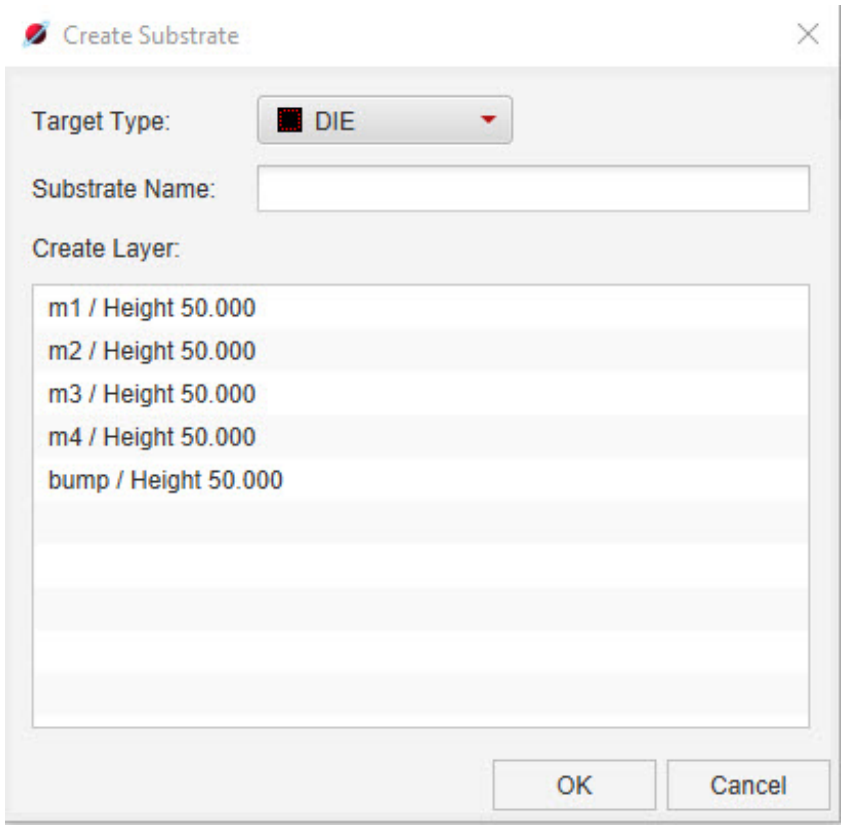
Defining Substrates

A substrate is essentially a layer stackup definition. Various devices will likely have different substrates. The first step in creating designs in System Planner is to create a substrate for targeting a device(s). Substrates can be created interactively or by importing a LEF tech file or CSV file.

Creating Substrates Interactively

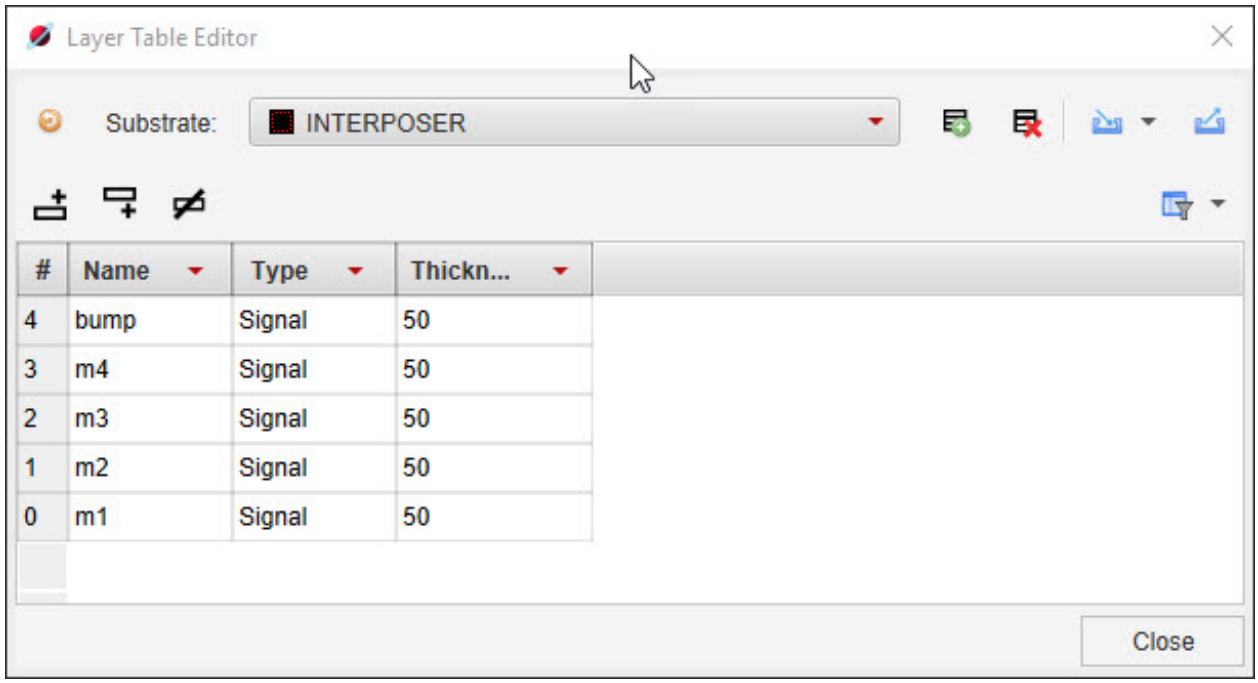
To create a substrate interactively:

1. Choose either the *Edit – Layers* command from the menu bar or the *Layer Stackup* command from the Flow Manager.
2. A confirmation form appears if no substrates exist in the design. Click *Yes* to create a substrate. By default, this opens the Create Substrate dialog, as shown below.



3. Update the following in the Create Substrate dialog:
   a. *Target Type:* Specify the substrate type from the pull-down menu, which includes the DIE, PACKAGE, BOARD, and SPACER options.
   b. *Substrate Name:* Specify a unique name for the substrate being created.
4. Click *OK* to create the default layer stack for the substrate type and name entered and open the Layer Table Editor, as shown below.

> ⓘ A default layer stackup is defined and displayed for each substrate type. The default layer stackup can be modified after the substrate is created.



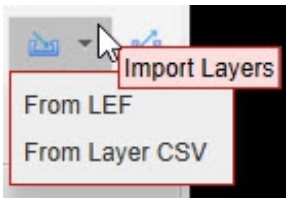
In the Layer Table Editor, you can:

- modify the name, type, or thickness of selected layers by either entering or selecting a value in the layer's *Name*, *Type*, and *Thickness* fields, respectively.
- modify the layer stackup by using the *Insert layer above*, *Insert layer below*, and *Delete layer* ( ) icons to create or delete layers, as required.
- create new substrates in a selected layer by clicking the *Create new substrate* icon ( ).
- delete substrates in a selected layer by clicking the *Delete substrate* icon ( ).

Creating Substrates by Importing a LEF Tech or CSV Layers File

Choose either the *Edit – Layers* command from the menu bar or the *Layer Stackup* command from the Flow Manager to open the Layer Table Editor dialog.

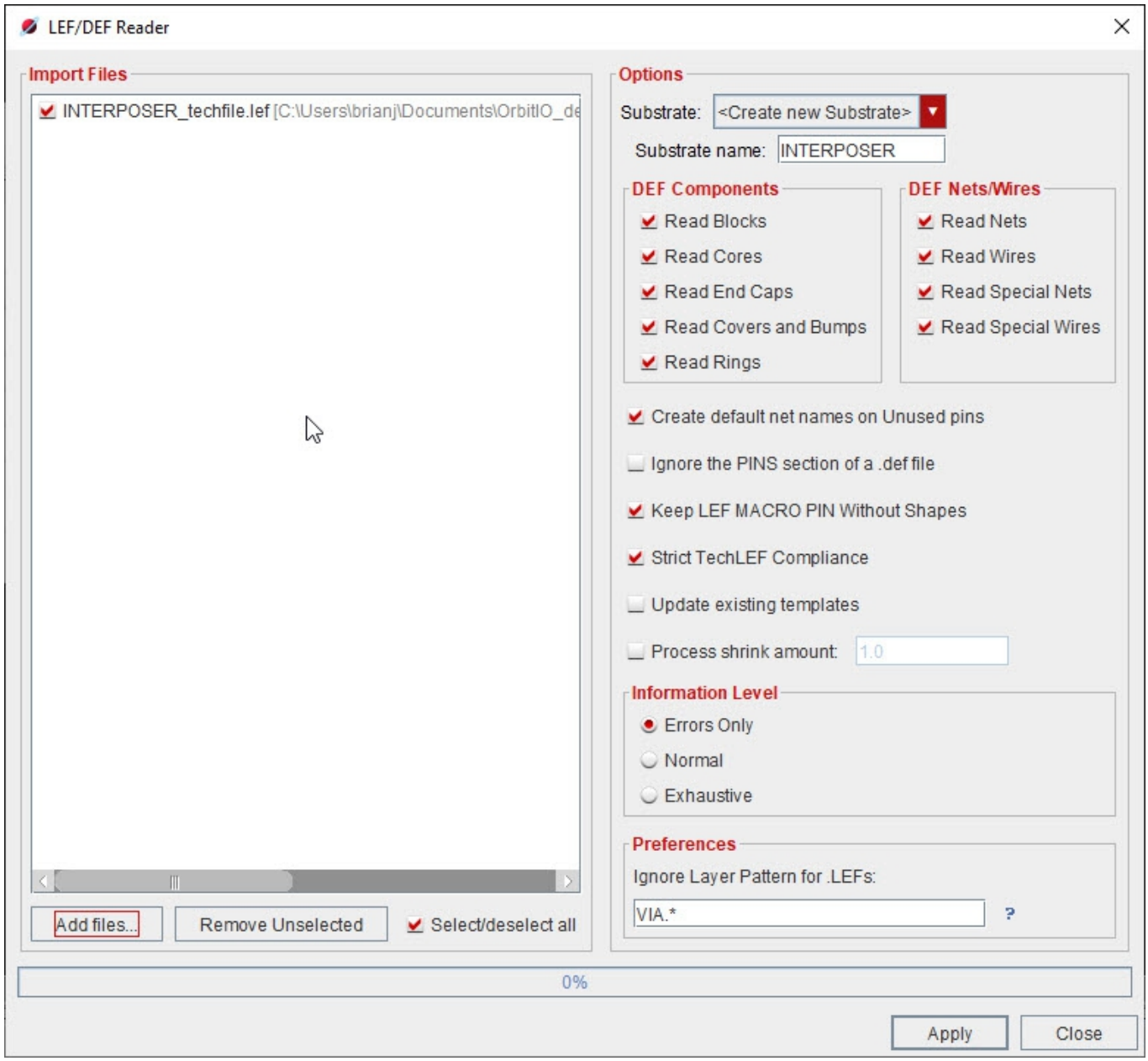
From the *Import Layers* pull-down menu, choose either the *From LEF* or *From Layer CSV* option.



By Importing a LEF Tech File

To create a substrate by importing a LEF tech file:

1. Choose *From LEF* from the *Import Layers* pull-down menu. This opens the *LEF/DEF Reader* dialog, as shown below.



2. Enter the name of the new substrate to be created in the *Substrate name* field. This field is available only when *Create new Substrate* is selected from the *Substrate* pull-down menu. LEF and DEF files can be imported for existing substrates by selecting the target substrate from the *Substrate* pull-down menu.

3. Use *Add files* to browse and select LEF files. Once LEF files are selected, they appear in the *Import Files* list. This dialog can also be used to import LEF pads, macros, and DEF layout information. The LEF tech file should be the first file shown in the *Import Files* list and the DEF file should be last. Right-click a file to view options to adjust the file import order.

Using the same LEF/DEF files that are being used for implementation, ensure matching layer stackups between System Planner and Innovus implementation. The LEF tech file is automatically passed back and forth when System Planner is invoked from Innovus.

4. Click *Apply* to create the new substrate based on the LEF tech file.

Importing a CSV Layers File

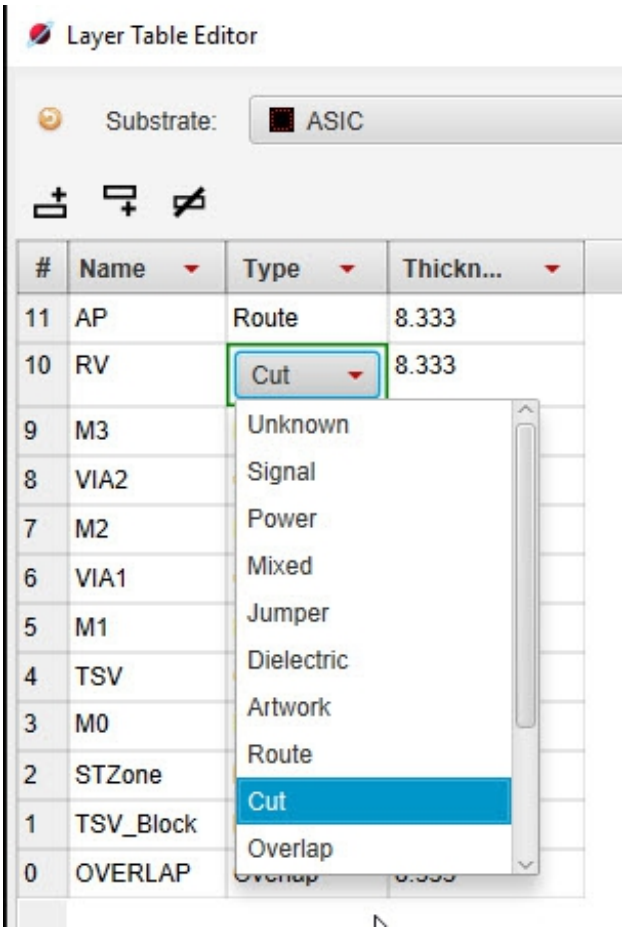
To create a substrate by importing a LEF tech file, choose *From Layer CSV* from the *Import Layers* pull-down menu and browse to a CSV file. Click *Open* to create and display the new substrates defined in the CSV file in the *Layers* view.

The Layers file CSV format is shown below.

Order	Substrate	Name	Height	Type
0	HBM	Bump	50	Signal
0	ASIC	AP	8.333333	Route
10	INTERPOSER	AP	9.090909	Route
9	INTERPOSER	RV	9.090909	Cut
8	INTERPOSER	M3	9.090909	Route
7	INTERPOSER	VIA2	9.090909	Cut
6	INTERPOSER	M2	9.090909	Route
5	INTERPOSER	VIA1	9.090909	Cut
4	INTERPOSER	M1	9.090909	Route
3	INTERPOSER	TSV	9.090909	Cut
2	INTERPOSER	MB	9.090909	Route
1	INTERPOSER	PO	9.090909	MasterSlice
0	INTERPOSER	OVERLAP	9.090909	Overlap

The *Order* column defines the way the substrates will appear in the *Layers* view from top to bottom. Notice that *HBM* and *ASIC* only have a single contact pad layer defined.

The available layer *Types* can be displayed and set by expanding one of the *Type* fields in the *Layer Table Editor*, as shown below.



Creating Padstacks - Pads, Bumps, and TSVs

Different methodologies use different terms, such as pads, bumps, ubumps, and Through Silicon Vias (TSVs), to describe the pads in the design. A padstack is simply a definition of the various layers, shapes, and a sizes for each pad used in the design. You can create padstacks ahead of time and then use them in the devices that you create. Padstacks can also be created interactively as devices are being created by entering the geometries and parameters.

Padstacks target a specific substrate and can define pads on the substrate for one or more layers. Once defined, they appear in the appropriate dialogs for selection when creating devices, bottom side bumps, and cover groups. A padstack can be used to define contact pads, pads on multiple substrate layers, and TSVs.

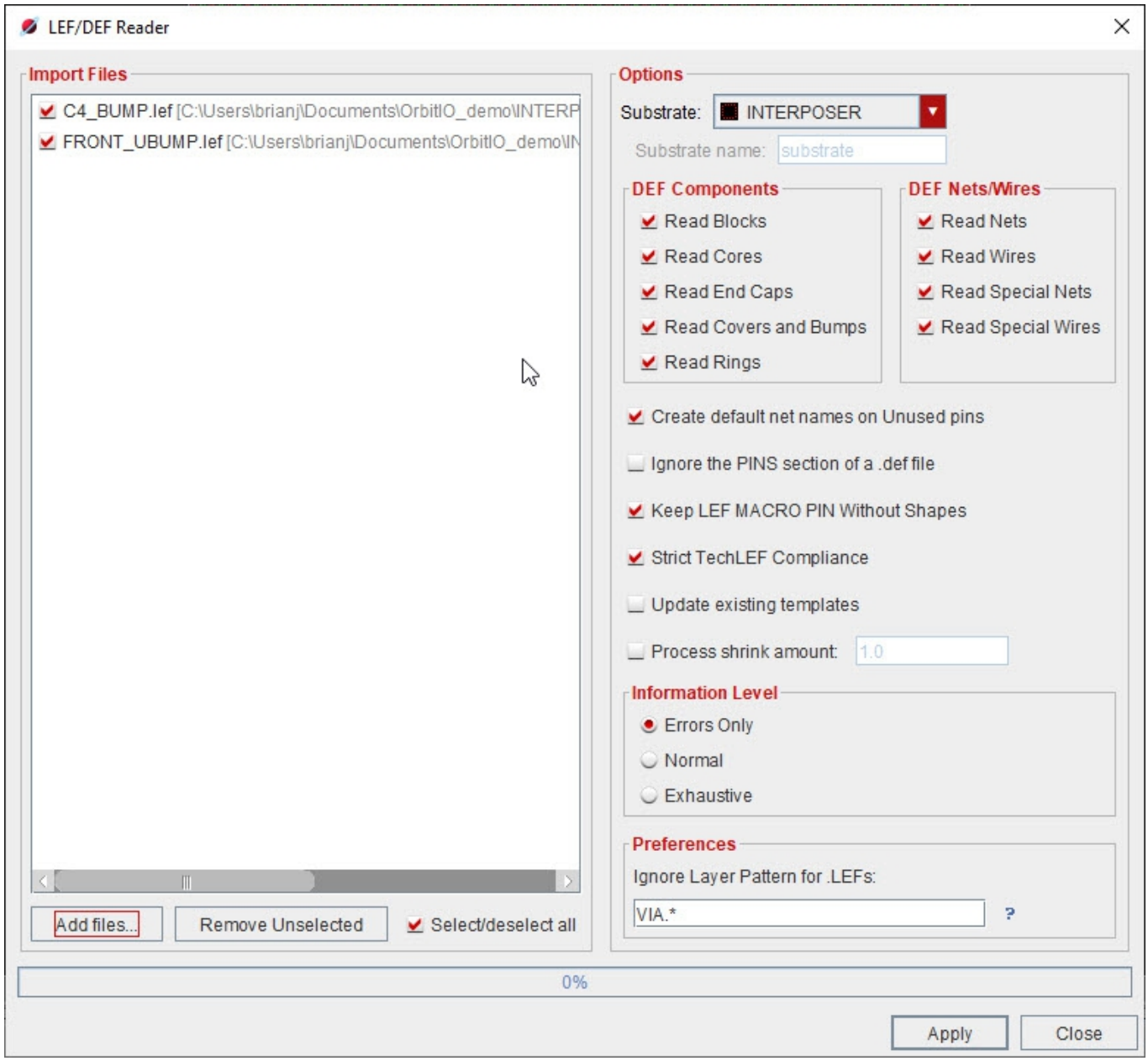
By Importing LEF Pads and Pad Macros

The recommended method for creating padstacks in System Planner for 3D-IC designs is to import LEF pad files. LEF pad files enable the layer characteristics used for implementation and analysis. Using the same LEF pads that are being used for implementation ensures matching pads between System Planner and the Innovus implementation. They are also automatically passed when System Planner is invoked from Integrity 3D-IC.

You can import LEF pad files while creating the substrate, as detailed in the Defining Substrates section of this guide.

You can also import LEF pad files separately by following these steps:

1. Choose either *File – Import – LEF/DEF* from the menu bar or *IC – Import – LEF/DEF* from the Flow Manager. This opens the *LEF/DEF* Reader dialog as shown below.



2. Select the substrate for which the padstack is to be created in the*Substrate* field.
3. Use *Add files* to browse and select LEF pad files. Once LEF pad files are selected, they appear in the *Import Files* list. Right-click a file to view options to adjust the file import order.
4. Click *Apply* to create padstacks based on the LEF file content.

By Using LEF Macros

You can also import LEF format macros. This enables multiple pins and pin/pad configurations to be created as a group of objects acting as a single pad device. An example may be several top-layer pads and vias funneling down to one C4 bump for increased current capacity.

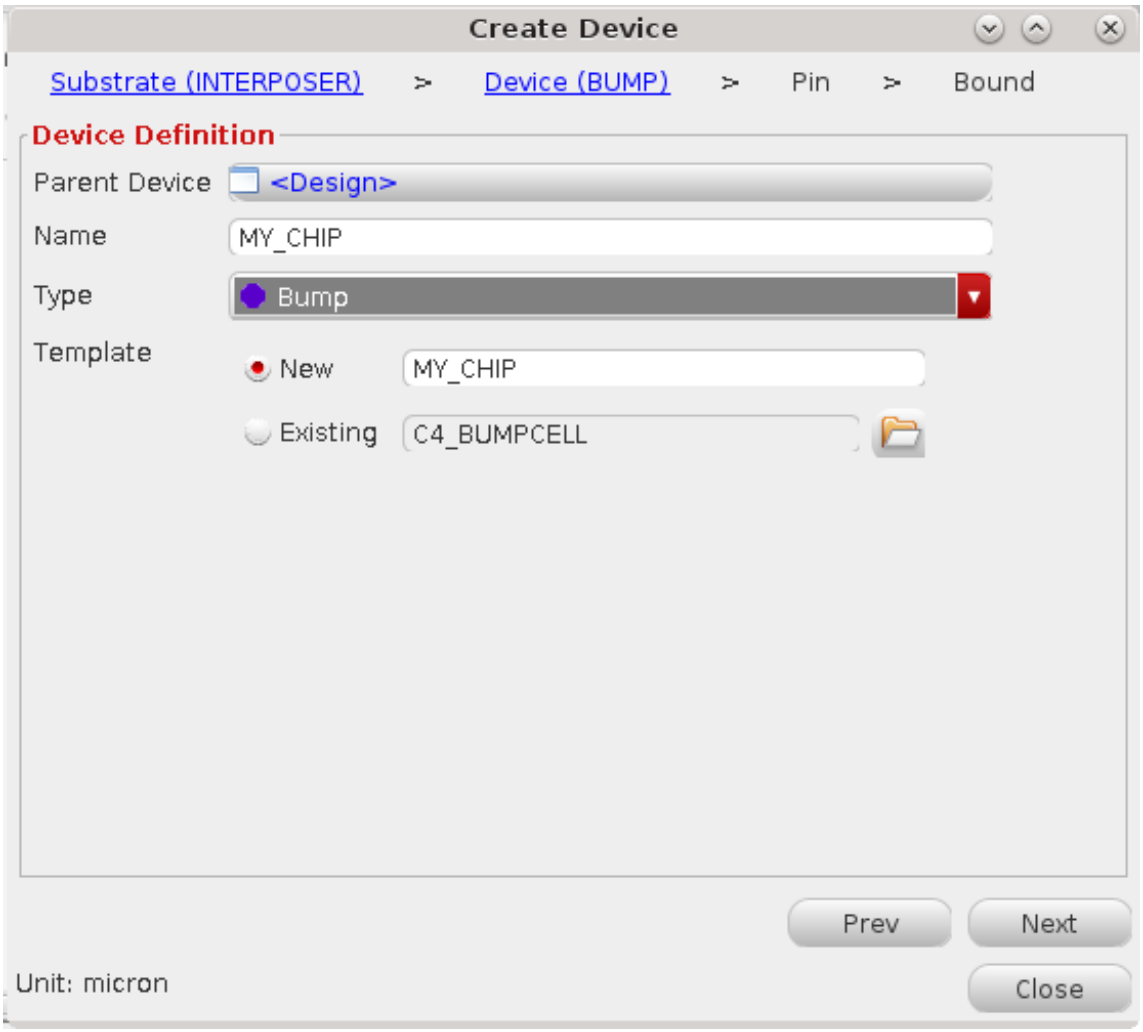
By Creating Bumps Interactively

You can create padstacks either interactively as a standalone device or when creating a device.

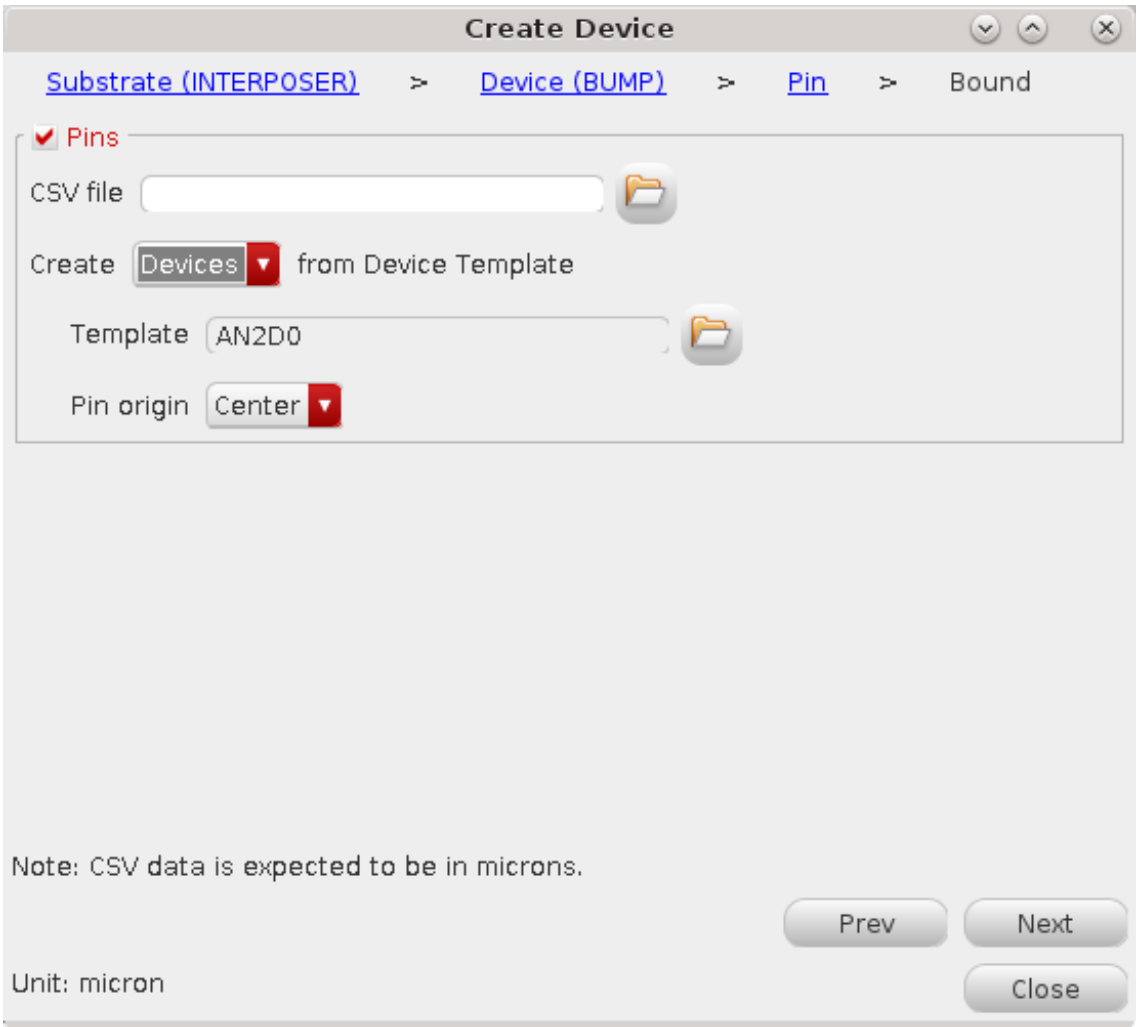
Creating Pad Devices

To create a pad device, do the following:

1. Choose *Edit – Devices – Create Device* from the menu bar or *IC – Create Device* from the Flow Manager.
2. Select the target substrate from the *Substrate* pull-down menu and click *Next* to continue.
3. Set the *Type* to *Bump*, as shown below. Either enter a new, unique device template name, or select an existing device template in the Template section. Click *Next* to continue.



4. The next pane of the dialog is shown below.

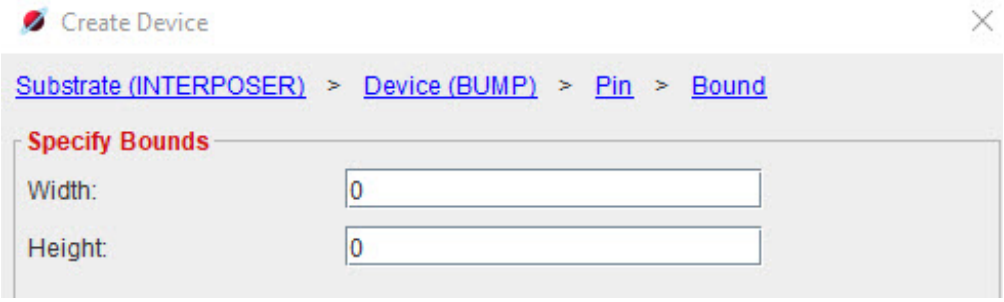


This pane enables the pins for the device to be imported using a CSV format file. The following fields are available in this pane:

Create	Specifies the type of pins to be created. Select *Devices* from the pull-down menu for 3D-IC designs (Innovus) or *Pins* for Package design (APD).
Template	Defines the padstack name. The padstack will appear with this name in subsequent dialogs.
Pin origin	Specifies the origin of the pins.

After making your selections, click *Next* to continue to the next pane.

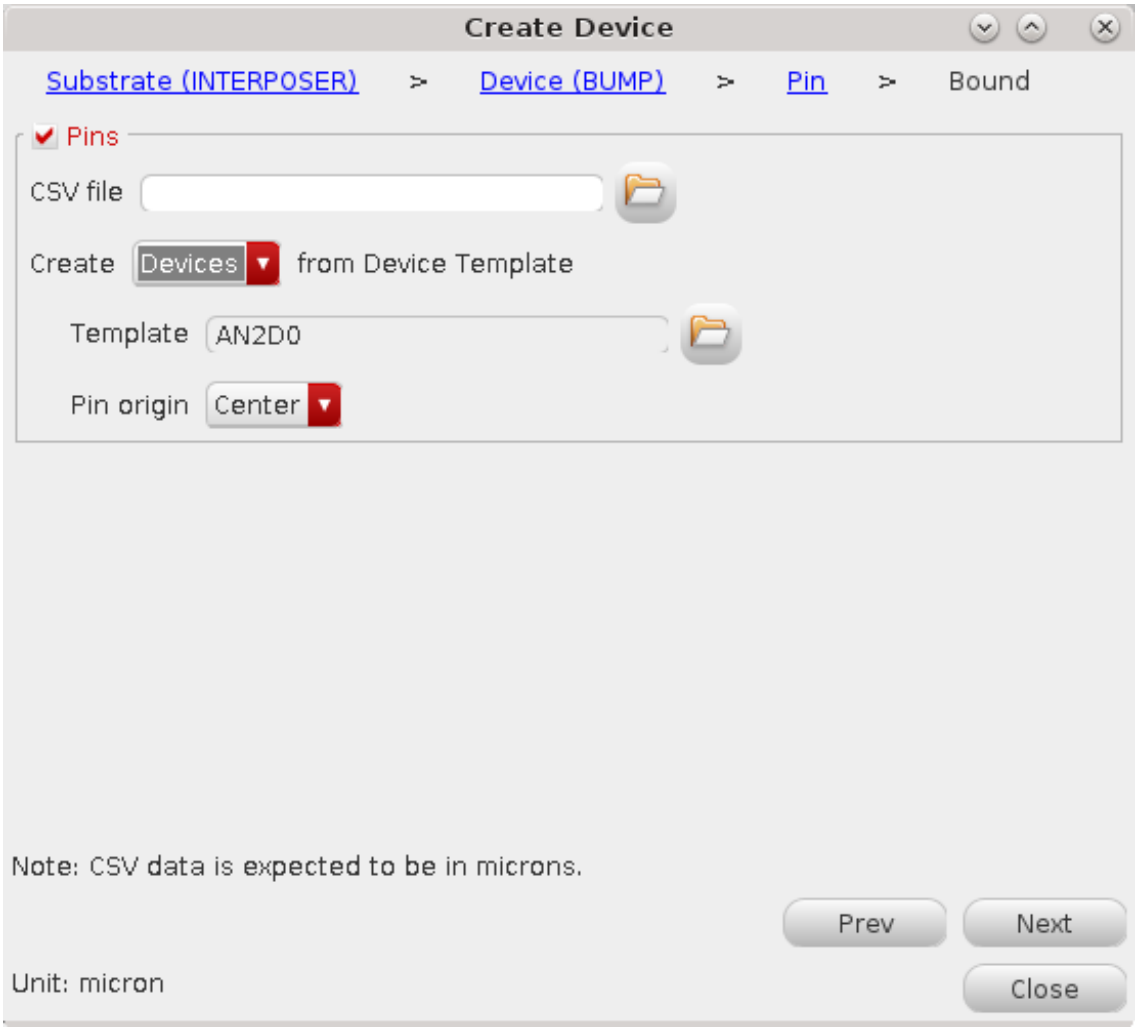
5. In the next pane that is displayed, specify the width and height.



If you leave the *Width* and *Height* fields with the default `0` values as shown above, the minimum bounding box size will be used to contain the pad object.

Creating Padstacks on-the-fly while Creating Devices

The *Pins* pane of the *Create Device* dialog enables pin parameters to be defined as the Die device is being created, as shown below.



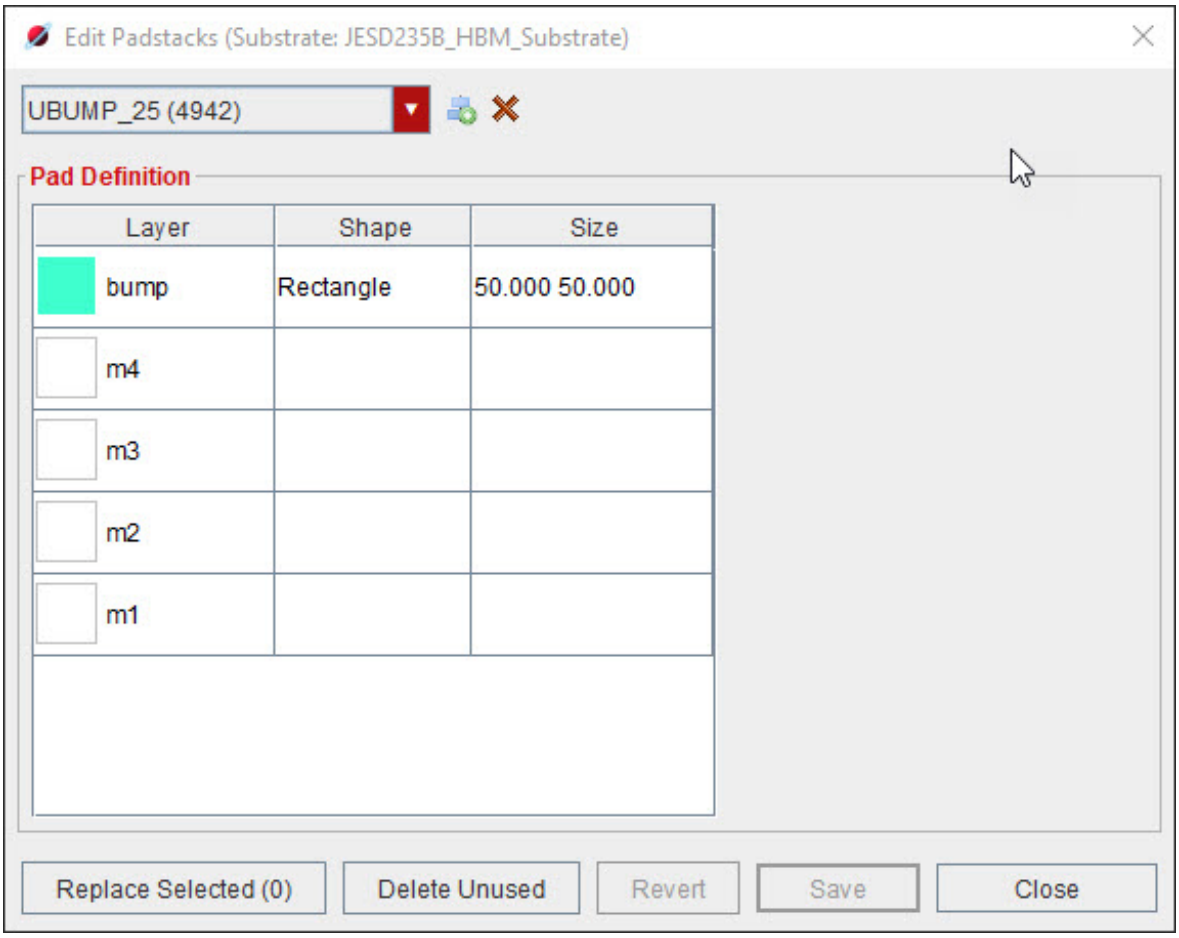
Specify the following in the dialog:

- Select either *Pins* or *Devices* from the *Create* pull-down menu. *Devices* style pads are used or 3D-IC designs, while *Pins* are used for package design.
- Define the padstack name in the *Template* field. The padstack will appear with this name in subsequent dialogs.

The device will be created with the padstack parameters defined.

Defining Padstacks while Importing Device Source Files

When creating devices by importing TXT or CSV format files that do not have pads defined in them, the *Edit Padstacks* dialog is presented to enable the pad size to be defined, as shown below.

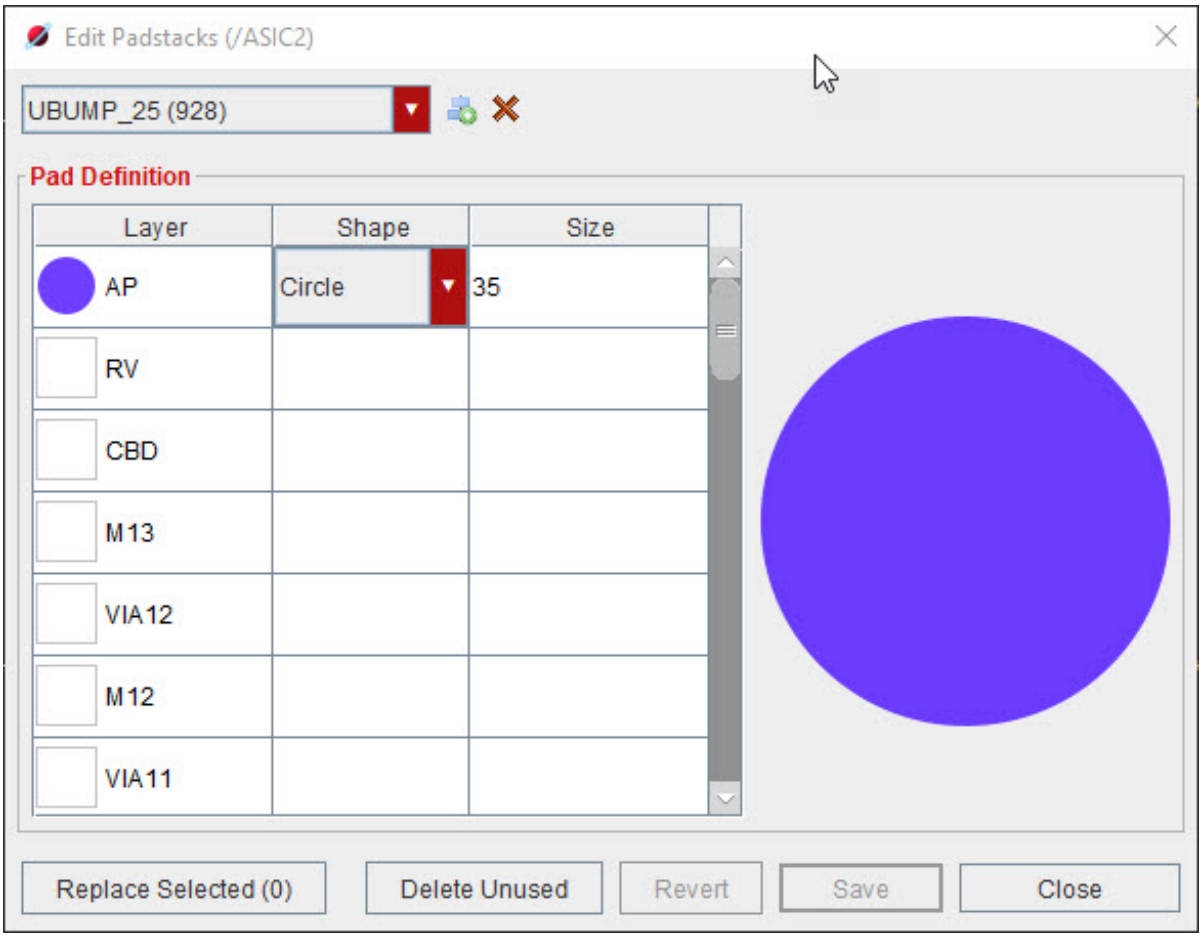


The device will be created with the padstack parameters defined.

When creating a device from a CSV pin list file, the file can include a PADSTACK column, where one or more padstack names can be defined. The *Create Device – Pins* pane enables a single initial pad parameter to be defined. Each of the padstack names defined in the imported CSV file will be created with this default configuration. They can then be modified with the correct parameters using the *Padstack – Edit Padstack* command, as shown in the next section.

Viewing and Modifying Padstacks

Padstacks can be viewed and modified by right-clicking any device with the target substrate in the *Device Hierarchy* and choosing *Padstack – Edit Padstacks* from the shortcut menu. This opens the *Edit Padstacks* dialog, as shown below.

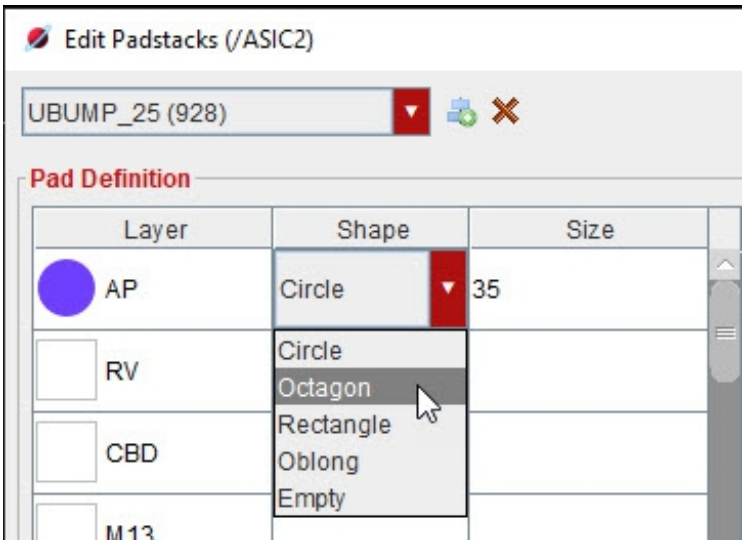


Existing padstacks defined for the substrate appear in the pull-down menu at the top of the dialog, with the following exception.

> ⓘ Padstacks created by importing LEF files will not appear in the Edit Padstack dialog. Such pads can be modified only by editing the LEF files directly, which is preferred so as to keep the LEF files as the source.

Choose a padstack from the pull-down menu, and the pad configuration for each layer on the substrate is displayed.

Pad shapes can be defined on any layer by clicking in the*Shape* field and choosing a value from the pull-down menu, as shown below.



Pad sizes can be entered by clicking in the *Size* field and entering the value. The *Enter* key must be pressed for the change to take effect.

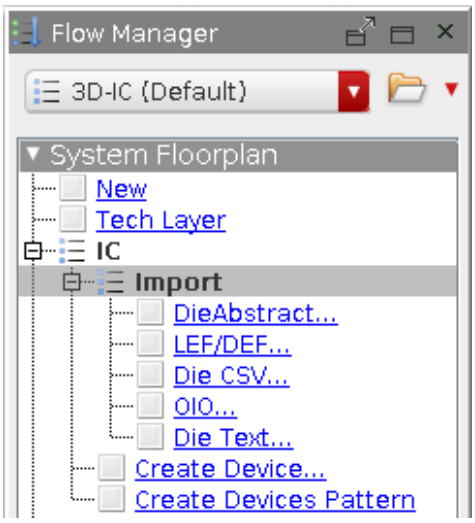
New padstacks can be created by clicking the *Create new padstack* icon ( ).

Padstacks can be deleted by clicking the *Delete current selected padstack* icon ( ).

Creating Devices

Several methods are available to create devices in System Planner. This section focuses on creating Die type devices. Package and Macro type devices are covered in the *Designing with the Integrity 3D-IC Platform RAK* and the *ASIC Macro and Bump Floorplanning RAK*.

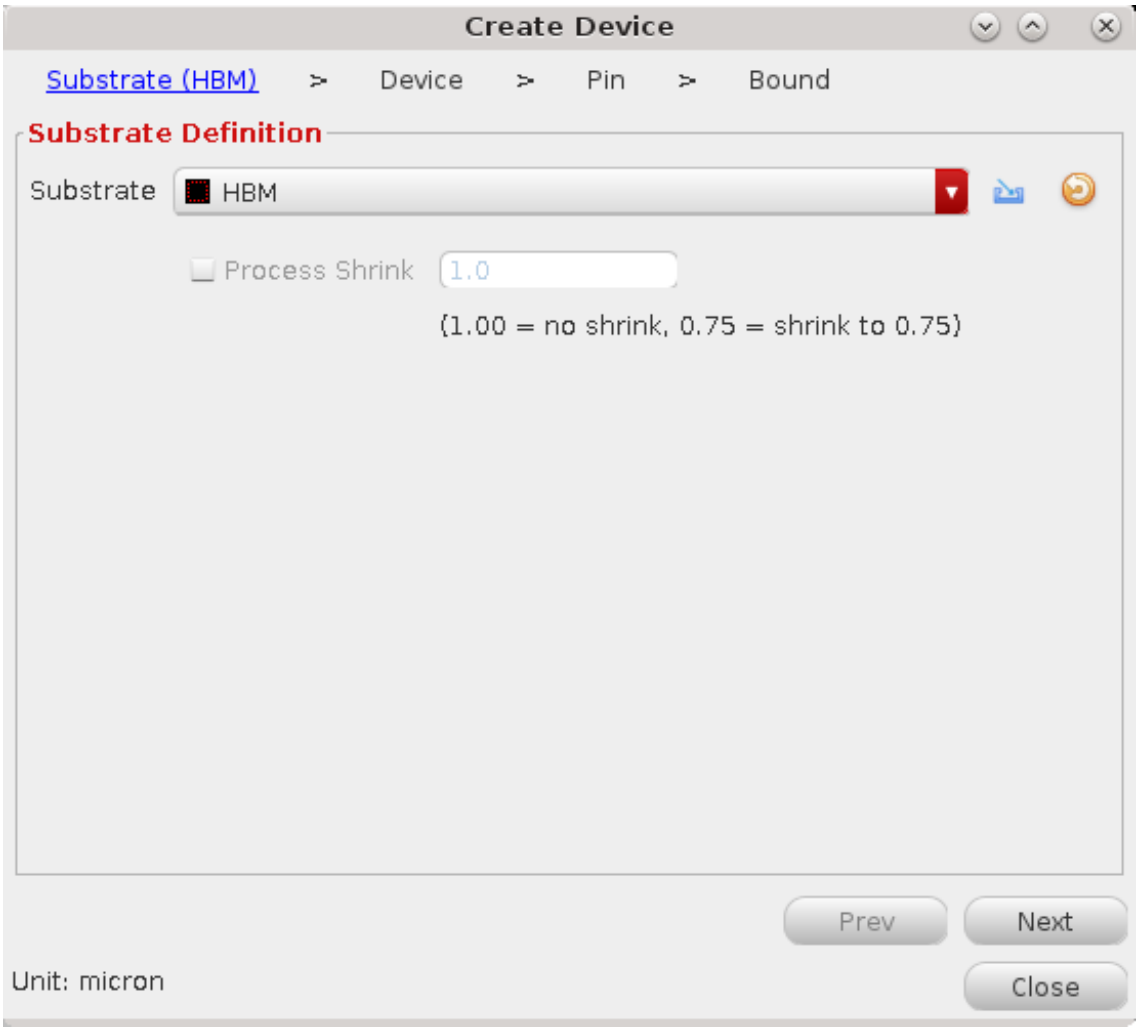
Select *File – Import* from the menu bar or expand the *IC – Import* folder in the *Flow Manager* to view the options, as shown below.



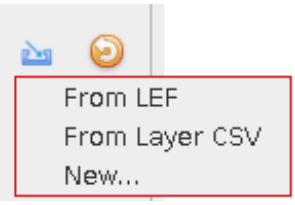
By Using the Create Device Option

The Create Device command can be used to create initially empty devices, such as Interposers or Packages. It can also be used to create devices using a CSV pin list file. Options for default pad configurations are presented.

1. Choose either *Edit – Devices – Create Device* from the menu bar or *IC – Create Device* in the *Flow Manager* to invoke the first pane of the *Create Device* dialog, as shown below.



2. Devices need to target a specific substrate.
   a. Specify one of the following in the dialog:
      - Select an existing target substrate for the device *Substrate*.
      - Create a new substrate in *Import layers* either interactively by selecting *New* or by importing a LEF or CSV file, as shown below.



   b. In the *Process Shrink* field, you can specify the intended shrink percentage. The pin coordinates will be scaled appropriately upon import.

c. Click *Next* to continue to the second pane of the dialog.

3. In the next pane that is displayed, specify the following:

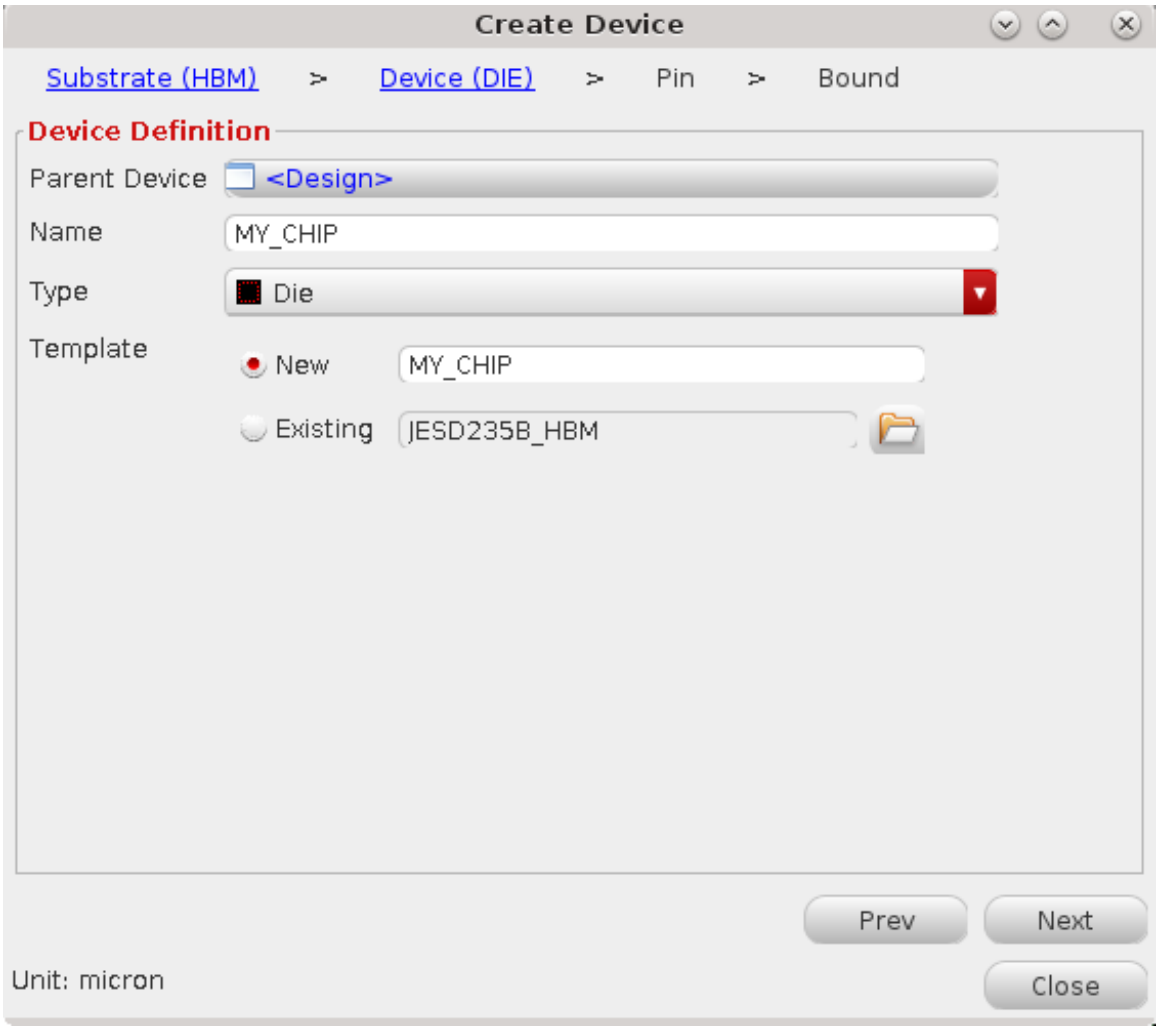
   a. If the device is to be created under a specific parent, select the existing parent in the *Parent Device* field. The default is to create the device at the top-level of the design.

   b. Select the type of device to create using the *Type* pull-down menu.

   c. Either enter a new, unique device template name, or select an existing device template in the *Template* section.

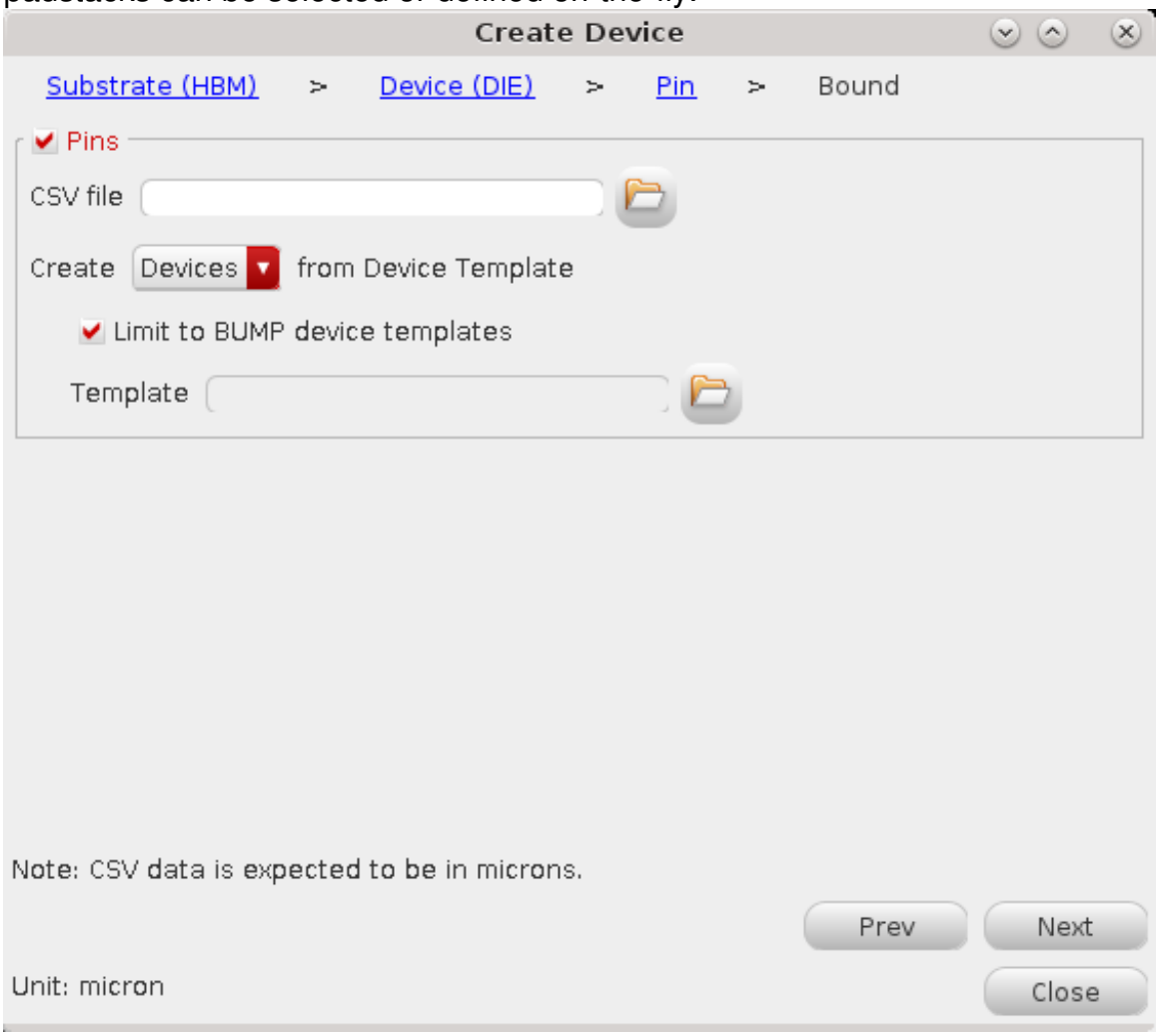
   d. Enter a unique name for the new device in the *Name* field.

   e. Click *Next* to continue.



4. In the next pane that is displayed, select *Pins* and either import a CSV pins list file containing the device pins or define the default pad parameters for the pins in the device. Existing

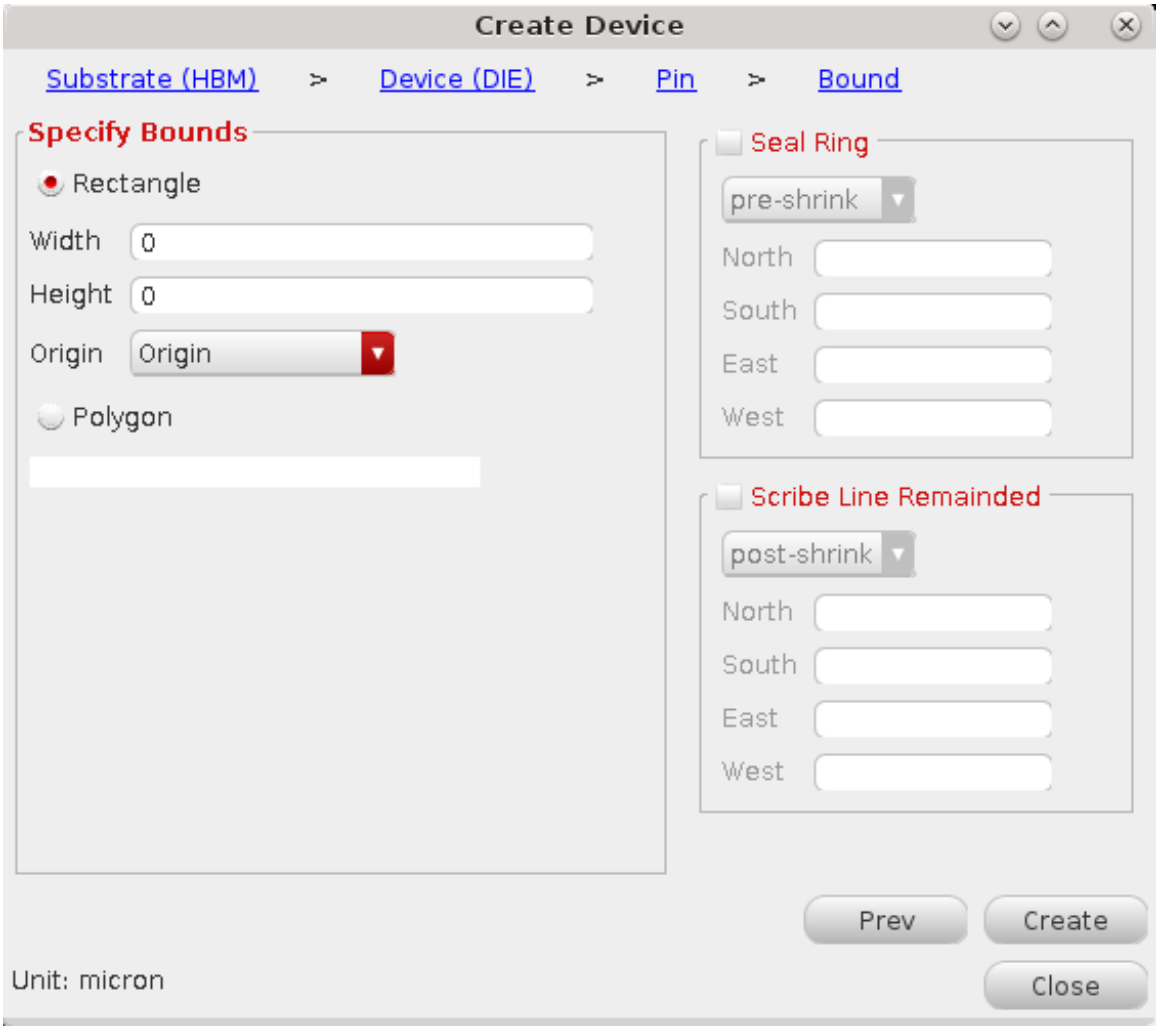
padstacks can be selected or defined on-the-fly.



> Leave the *Pins* check box unchecked if you do not want to import a pins list or define default pad sizes.

5. If you have opted to select the *Pins* check box, specify the following in the dialog:
   a. Specify a CSV pins list in the *CSV file* field. The CSV pin list file format is described in the Importing a CSV Pin List section of this guide.
   b. Select either *Pins* or *Devices* from the *Create* pull-down menu. *Devices* style pads are typically used for 3D-IC designs, while *Pins* are used for package designs.
   c. Click *Next* to continue.
6. Specify the following in the fourth pane of the dialog, as shown below.

a. Specify the device size by entering values for *Width* and *Height*. Using the default 0 values will create a device size based on the minimum bounding box around all device pins.



In the *Seal Ring* and *Scribe Line Remainded* fields, you can enter the *North*, *South*, *East*, and *West* distance values. The values should be based on the existing device outline.

7. Click *Create* to complete creating the device based on the defined inputs.
   If required, you can first dick *Prev* to cycle back through the dialog panes, for example, to modify any settings or to create additional devices using some of the data already defined.
   Click *Close* to close the *Create Device* dialog without creating the device.

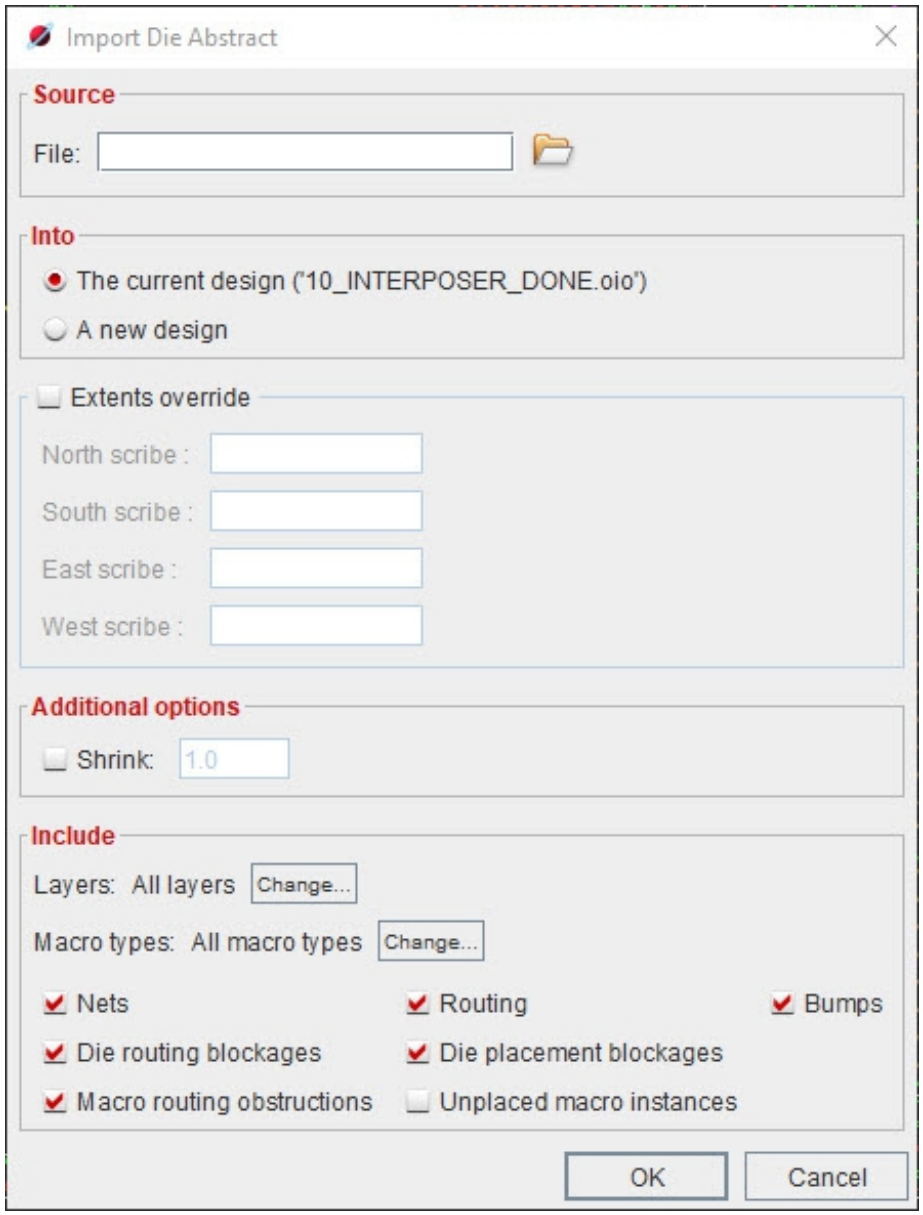
By Importing a Die Abstract File

A die abstract file is a Cadence-defined format designed to pass single-pad-layer die information easily between the various layout tools including APD Plus, Innovus, and Virtuoso. All these tools can also be used to create a die abstract file. System Planner also has an *Export – Die Abstract* capability. Die abstract format is sometimes preferred when extra security is needed when sending files such as LEF/DEF between different design groups or companies.

Die abstract files use the *.xda* extension.

To create a device by importing a die abstract file:

1. Choose either *File – Import – Die Abstract* from the menu bar or *IC – Import – Die Abstract* from the *Flow Manager* to invoke the *Import Die Abstract* dialog, as shown below.



2. Specify the following in the dialog:
   a. Specify the die abstract file (`.xda`) in the *File* field.
   b. Choose to import the specified die abstract into either the existing design project or a new project in the *Into* section.
   c. Specify the device boundary scribe information in the *Extents override* section.
   d. Specify a *Shrink* value in the *Additional options* section. The shrink processing will take place during the import process.

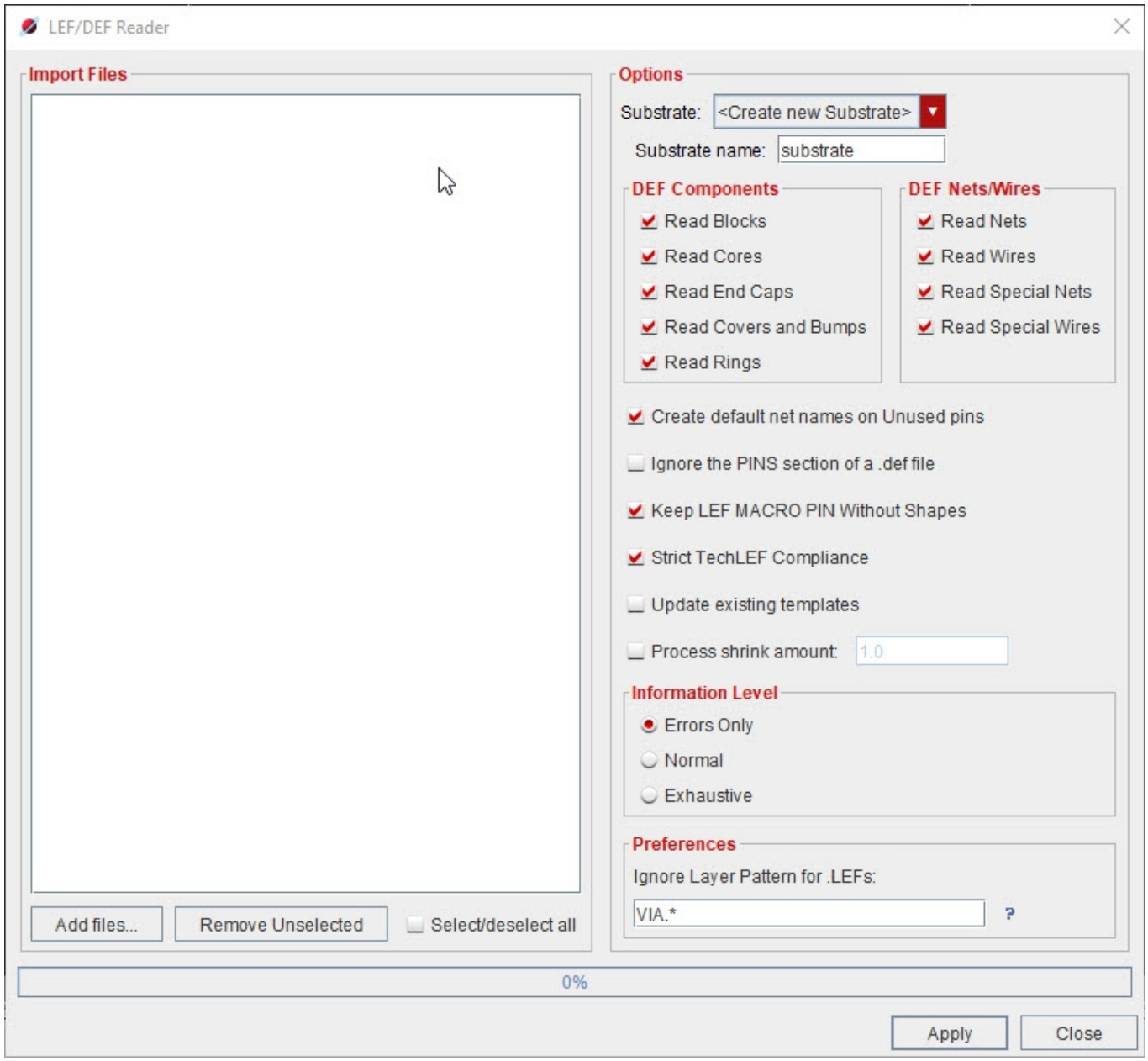
   e. Specify the data to be imported from the die abstract file in the *Include* section. This relies on the data being included in the file when it was created.

3. Click *OK* to create the device based on the input die abstract file data. A new substrate will be created to support the newly imported device.

By Importing LEF/DEF Files

To create a device by importing a LEF/DEF file:

1. Choose either *File – Import – LEF/DEF* from the menu bar or *IC – Import – LEF/DEF* from the *Flow Manager* to invoke the *LEF/DEF Reader* dialog, as shown below.



2. Specify the following in the dialog:
   a. Select an existing substrate to target the device or create a new substrate for the device in the *Substrate* pull-down menu.
   b. Enter a name in the *Substrate name* field if you selected *<Create new substrate>*.
   c. Click *Add files* to browse and select LEF/DEF source files. Once files are selected, they appear in the *Import Files* list. There are right-click options to adjust the file import order.

The LEF tech file should be the first file shown in the *Import Files* list, followed by any LEF pad files and then the DEF file. Right-click a file to view options to adjust the file import order.

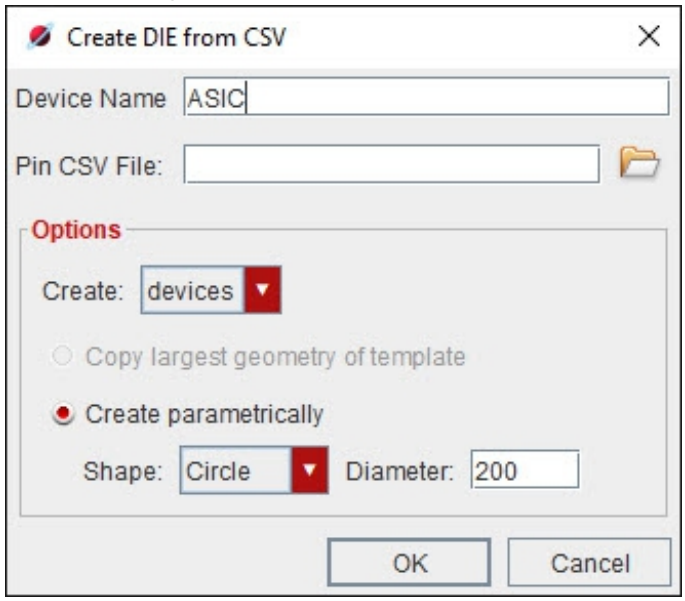
d. Specify the remaining options in the dialog, as needed.

3. Click *Apply* to create the device with the specified LEF/DEF data.

By Importing a Die CSV File

Devices can be created by importing a die format CSV file:

1. Choose either *File – Import – Die CSV* from the menu bar or *IC – Import – Die CSV* from the *Flow Manager* to invoke the *Create DIE from CSV* dialog, as shown below.



2. Specify the following in the dialog:

   a. Enter a name for the device in the *Device Name* field.

   b. Select an input pin format CSV file in the *Pin CSV File* field. To see the format for the Pin CSV file, click here.

   c. From the *Create* pull-down menu under *Options*, specify the pin style. Select *devices* style pins for IC design or *pins* style pins for package design.

   d. Define pad parameters in the *Create parametrically* section to create padstacks. Alternatively, to use an existing padstack, select the *Copy largest geometry of template* option. If you select this option, the available padstacks are presented for

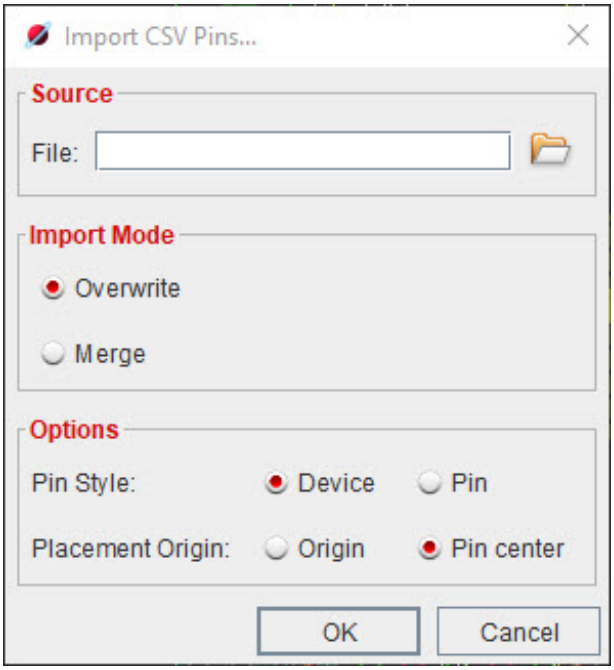
selection from a pull-down menu.

3. Click *OK* to create the device based on the input file data. A new substrate will be created to support the newly imported device.

By Importing a Pins List CSV File

Devices can be created by importing a pin list format CSV file. The pin list can be specified when using the *Create Device* command. Refer to the *Using the Create Device* section of this guide for more information.

Pins list CSV files can also be imported for a device by right-clicking the device in the *Device Hierarchy* and choosing the *Import – CSV Pins* command from the shortcut menu. The *Import CSV Pins* dialog is presented, as shown below.



Complete the dialog as follows:

1. In the *File* field, browse to select the CSV pins list file to be imported.
2. In the *Import Mode* section, select *Overwrite* to replace existing device pins or *Merge* to merge the imported pins with the existing device pins.
3. In the *Pins Style* section, select the *Device* style to create pins for IC layout or the *Pin* style to create pins for package layout.
4. In the *Placement Origin* section, specify the origin for the pins being imported.

5. Click *OK* to import the device pins with the defined options.

The format for the pin list CSV file is displayed below. The required columns are highlighted. The other columns are optional.

	A	B	C	D	E	F	G	H	I	J	K	L
1	PIN_NUMBER	PIN_NAME	X	Y	NET_NAME	PADSTACK	PIN_USE	PIN_DIRECTION	FIXED	PIN_PERSONALITY	DEVICE_PIN_TERM	DEVICE_PIN_TEMPLATE
2	ASIC_A128	VSS	1558.001	15346.5	HBM2_PHY_HI_DQF[79]	UBUMP_25	SIGNAL	INOUT	FALSE		PAD	FRONT_UBUMP
3	ASIC_A139	VDDIO	1398.001	15346.5	HBM2_PHY_HI_DQF[77]	UBUMP_25	SIGNAL	INOUT	FALSE		PAD	FRONT_UBUMP
4	ASIC_A154	VSS	1238.001	15346.5	HBM2_PHY_HI_WDQSF_C[2]	UBUMP_25	SIGNAL	INOUT	FALSE		PAD	FRONT_UBUMP
5	ASIC_A165	VDDIO	1078.001	15346.5	VDDIO	UBUMP_25	SIGNAL	INOUT	FALSE		PAD	FRONT_UBUMP
6	ASIC_A181	VSS	914	15346.5	VSS	P5608	SIGNAL	INOUT	FALSE		PAD	FRONT_UBUMP

By Importing a System Planner "OIO" Project File

A device from an existing System Planner project can be used as an input to another project. The input `.oio` file must first be created by right-clicking the device in the source project and choosing *Export – OIO*. The exported file can then be used as the source for importing the device into another project. The OIO file being imported must be prepared this way.

To import a System Planner `.oio` source file, choose *File – Import – OIO* from the menu bar or *IC – Import – OIO* from the *Flow Manager*. A file browser is presented to choose the input `.oio` file. A device is created based on the data in the source `.oio` file.

The *File – Import – OIO* command will not accept `.oio` files for existing designs.

By Importing a Die Text "TXT" File

A device can be created by importing a text format file:

1. Choose *File – Import – Die Text* from the menu bar or *IC – Import – Die Text* from the *Flow Manager* to invoke the *Import Die Txt* dialog, as shown below.

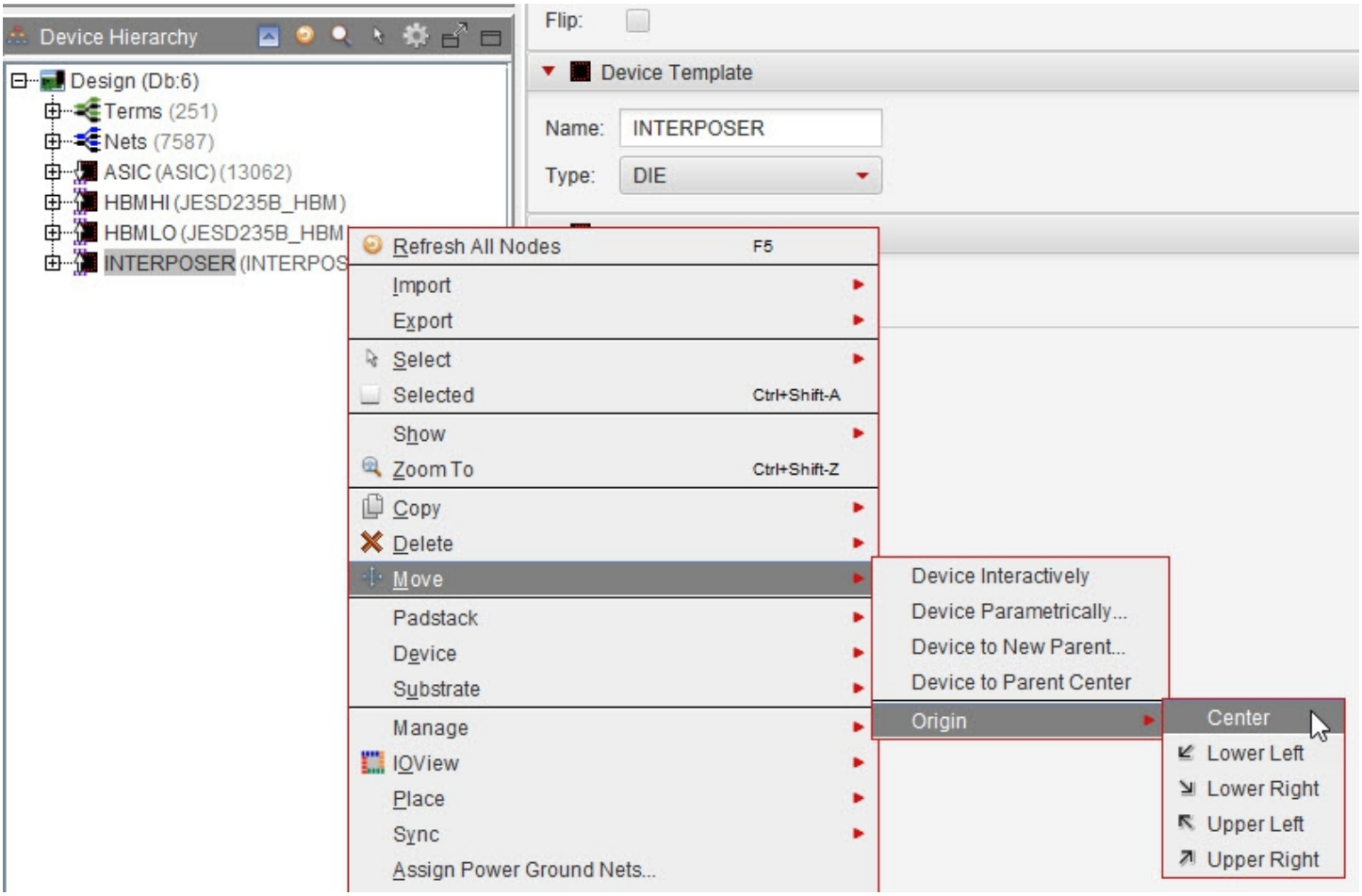
2. Specify the following in the dialog:
   a. In the *File* field, browse to select the input text format file.
   b. Set the *Coordinates* as either *Absolute* or *Relative*, depending on how the source text file was created.
   c. Create a new substrate or select an existing project substrate from the*Substrate* pull-down menu.
   d. Select the *Pin Style*. Choose *Device* style pins for IC layout.
   e. Click inside the *Parent Device* field to open the Select Root form, where you can select an existing device as parent for the new device in the *Device Hierarchy*. The default is the top-level of the design.
   f. Define a *Process shrink* value for the shrink processing to take place during import.
3. Click *OK* to complete creating the device based on the text file data.

The format for the TXT file is displayed below. The*Net Name* column is optional.

```
Pin Number      Padstack        X Coord         Y Coord         Rotation        Pin Use         Net Name

1               UBUMP_25        -1608.0000      3011.2500       0.000           BI
2               UBUMP_25        -1608.0000      2956.2500       0.000           BI
3               UBUMP_25        -1608.0000      2901.2500       0.000           BI
4               UBUMP_25        -1608.0000      2846.2500       0.000           BI
5               UBUMP_25        -1608.0000      2791.2500       0.000           BI
6               UBUMP_25        -1608.0000      2736.2500       0.000           BI
```

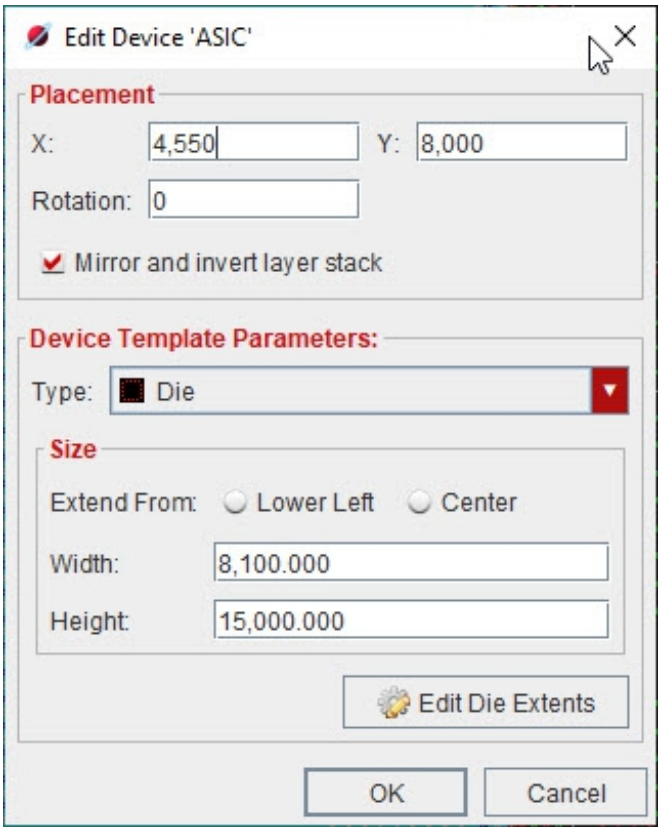
Changing the Origin of a Device

Occasionally, the origin of a device needs to be changed to enable alignment, exact placement, or easier rotation of the device in place when floorplanning. To change the origin of a device, right-click the device in the Device Hierarchy and choose *Move – Origin – <New origin setting>*, as shown below.



Changing the Size of a Device

The size of an existing device can be modified by selecting the device and right-clicking to choose the *Device – Edit – Device* command. The *Edit Device* dialog is presented, as shown below.



Modify the *Width* and *Height* fields to change the die size.

The die size modifications are dependent on the origin of the device. The origin will not move during resizing. If the device origin is set to center, the die size gets changed uniformly from the center of the device. If the device origin is set to lower left, the die size will be changed in the X Y directions from that point, and so on.

Floorplanning Devices

There are several features to manipulate the placement of devices in the design. Ultimately, the connectivity between the devices in the design can help dictate the proper placement, orientation, and alignment.

System Planner provides system-level connectivity display across the design. Several connectivity optimization features are available to help reduce connection crossings and length.

Flipping Devices

Flip-chip devices are often imported or created with their original orientation and need to be flipped to accurately interface with their contacted device. To flip a device, right-click the device in the Device Hierarchy and choose the *Flip in Place* command. The icon displayed next to the device in the Device Hierarchy panel will now point down rather than up.

Moving Devices

Several commands and options are available to adjust the placement for devices.

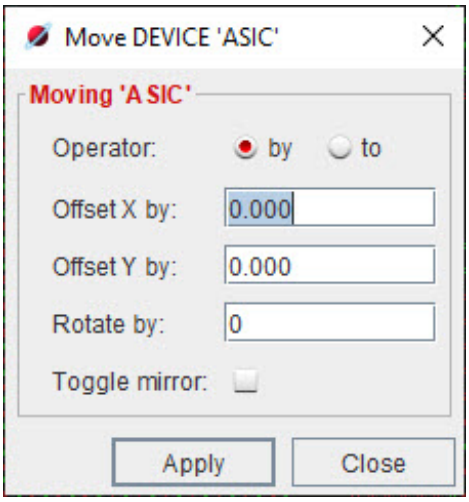
Moving Devices Interactively

Right-click a device in the Device Hierarchy panel and choose *Move – Device Interactively* to begin dragging the device on the canvas. Right-clicking will show options such as *Rotate* and *Mirror*. Clicking on the canvas will place the devices at that location.

The *Interactive Move Mode* can also be used when moving devices. Select the main toolbar *Interactive Move Mode* icon ( ) to enter the mode. Objects can be selected for the move by clicking on them in the canvas. The mode remains active until another mode is selected, so different objects can be continually selected for the move. The *Selection Filter* is not activated while using this mode, so be very careful that you are moving the intended object. For example, a pin can get inadvertently selected when a device was intended for the move.

Moving Devices Parametrically

Right-click a device in the Device Hierarchy and choose *Move – Device Parametrically* to invoke the *Move DEVICE <device>* dialog, as shown below.



The options are self-explanatory. Clicking *Apply* or pressing the *Enter* key will make the move.

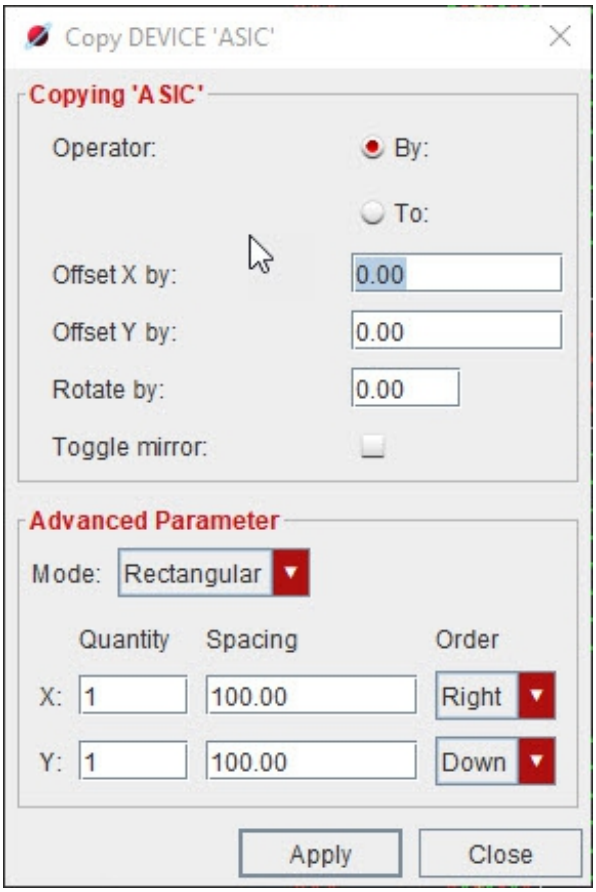
The device(s) remain selected and additional moves can be performed by entering new values in the dialog and clicking *Apply* or pressing the *Enter* key.

Copying Devices

Devices can be copied in the design. Copied devices share the *Device Template* of the original, meaning that changes to one will affect the others.

Copying Devices to a Location

Right-click a device in the Device Hierarchy and choose *Copy – Device to Location* to invoke the *Copy DEVICE <device>* dialog, as shown below.

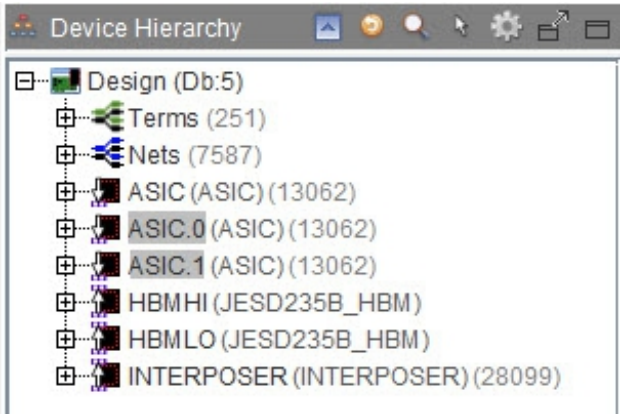


The options in the *Copying* section are self-explanatory.

The options in the *Advanced Parameter* section enable you to make multiple copies of the device in a matrix by entering the matrix values.

Clicking *Apply* or pressing the *Enter* key will copy the device.

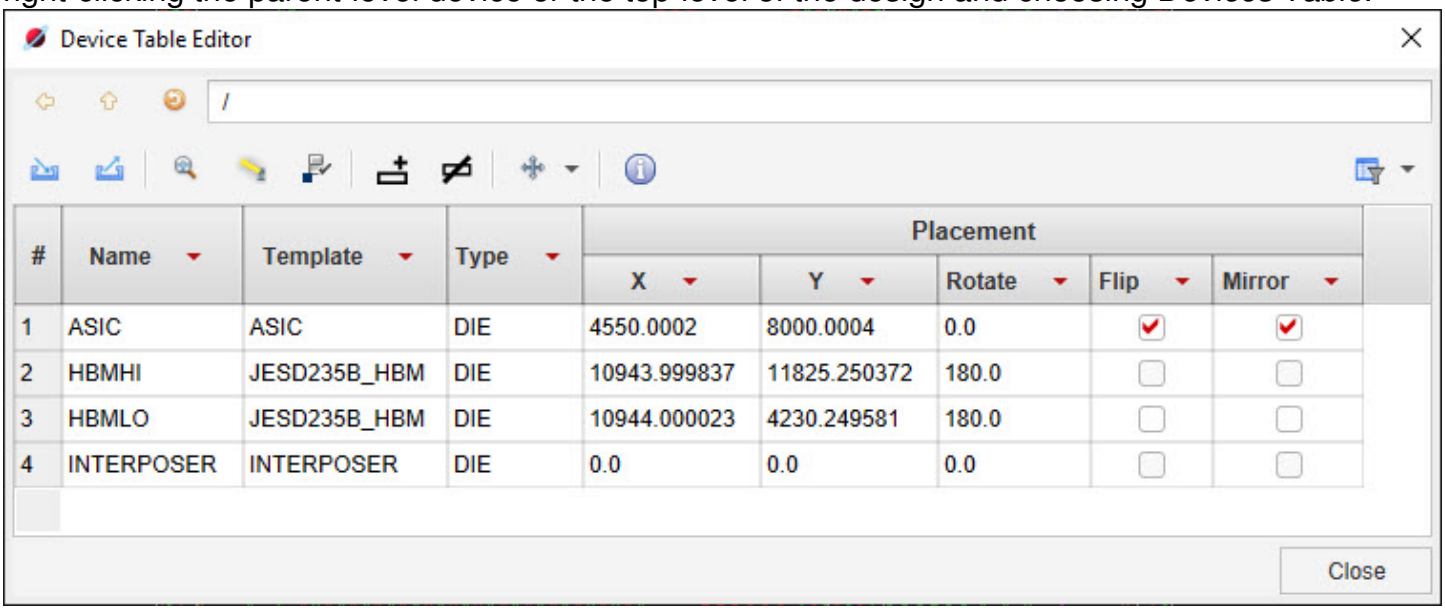
The copies of the device will be displayed at the same level as the original in the Device Hierarchy. The copies receive default names consisting of the original name with *.0*, *.1*, … extensions, as shown below. The names can be changed by selecting the device and entering a new name in the *Attribute Editor*.



The device(s) remain selected and additional copies can be performed by entering new values in the dialog and clicking *Apply* or pressing the *Enter* key.

Using the Device Table Editor

The Device Table Editor is available at each level of the hierarchy in the Device Hierarchy panel by right-clicking the parent-level device or the top-level of the design and choosing *Devices Table*.



#	Name	Template	Type	Placement				
				X	Y	Rotate	Flip	Mirror
1	ASIC	ASIC	DIE	4550.0002	8000.0004	0.0	✔	✔
2	HBMHI	JESD235B_HBM	DIE	10943.999837	11825.250372	180.0	☐	☐
3	HBMLO	JESD235B_HBM	DIE	10944.000023	4230.249581	180.0	☐	☐
4	INTERPOSER	INTERPOSER	DIE	0.0	0.0	0.0	☐	☐

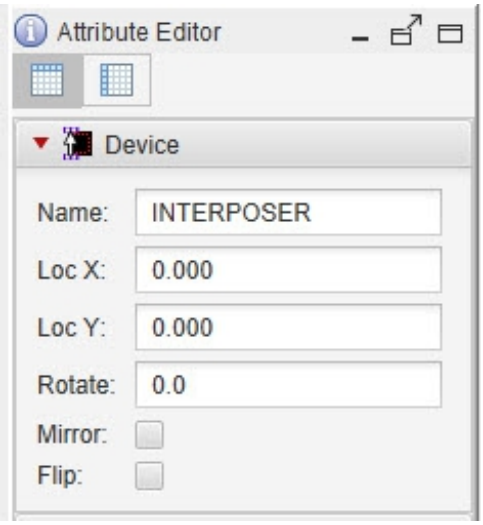
The top field enables navigation within the Device Hierarchy.

The icons enable importing and exporting of a CSV placement file; zooming, highlighting, or selection of the desired device on the canvas; and the creation, removal, or movement of devices.

The *Placement* location and orientation can be modified by selecting the appropriate field and entering a new value. The *Enter* key must be pressed to make the change.

Using the Attribute Editor

Selecting a device will display the device's placement information in the Attribute Editor, as shown below.



The values can be modified to adjust the device's placement and orientation.

Importing a CSV Devices File

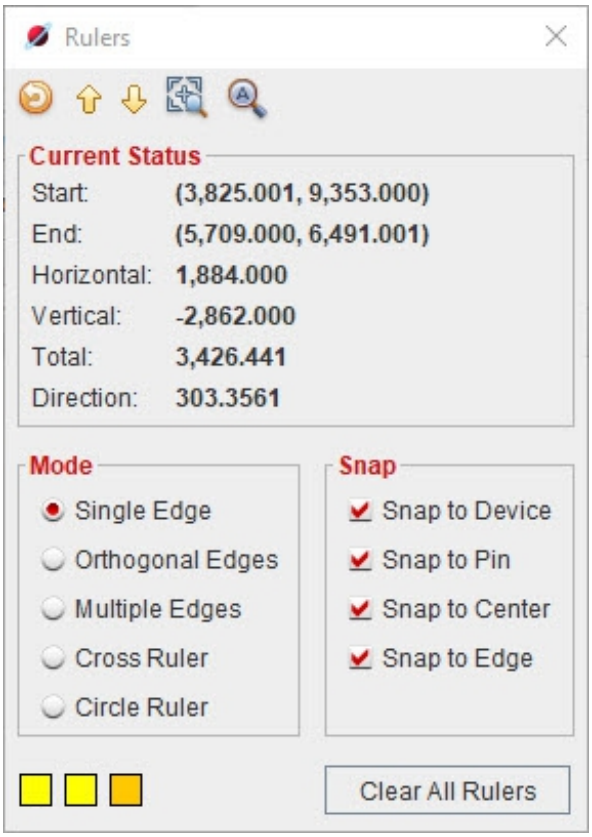
Placement can also be imported using a CSV devices placement file. Right-click any parent level of hierarchy in the Device Hierarchy, choose *Import – CSV Devices*, and select an input CSV placement file. The design placement will be adjusted per the data in the placement file.

The CSV Devices file format is shown below.

Device	Substrate	Device_Template	X	Y	Flip	Mirror	Rotate
ASIC	ASIC	ASIC	4550	8000	TRUE	TRUE	0
INTERPOSER	INTERPOSER	INTERPOSER	0	0	FALSE	FALSE	0
HBMHI	HBM	JESD235B_HBM	10944	11825.25	FALSE	FALSE	180
HBMLO	HBM	JESD235B_HBM	10944	4230.25	FALSE	FALSE	180

Using the Ruler Mode to Measure

The *Ruler Mode* can be used to measure and align devices exactly. Select the *Ruler Mode* icon ( ) on the main toolbar to invoke the *Rulers* dialog, as shown below.



Rulers are created by clicking in the canvas to measure between selected objects. As they are created, the *Current Status* values are updated with the appropriate distances, as shown above.

Multiple rulers can be created and displayed on the canvas at once. They can be cycled through by using the Up and Down arrow icons.

The *Mode* section provides options for the type of ruler desired.

The *Snap* section provides options for object snapping. Ensure that you are zoomed in close enough to snap to and select the desired objects. This is particularly important when snapping to pin centers.

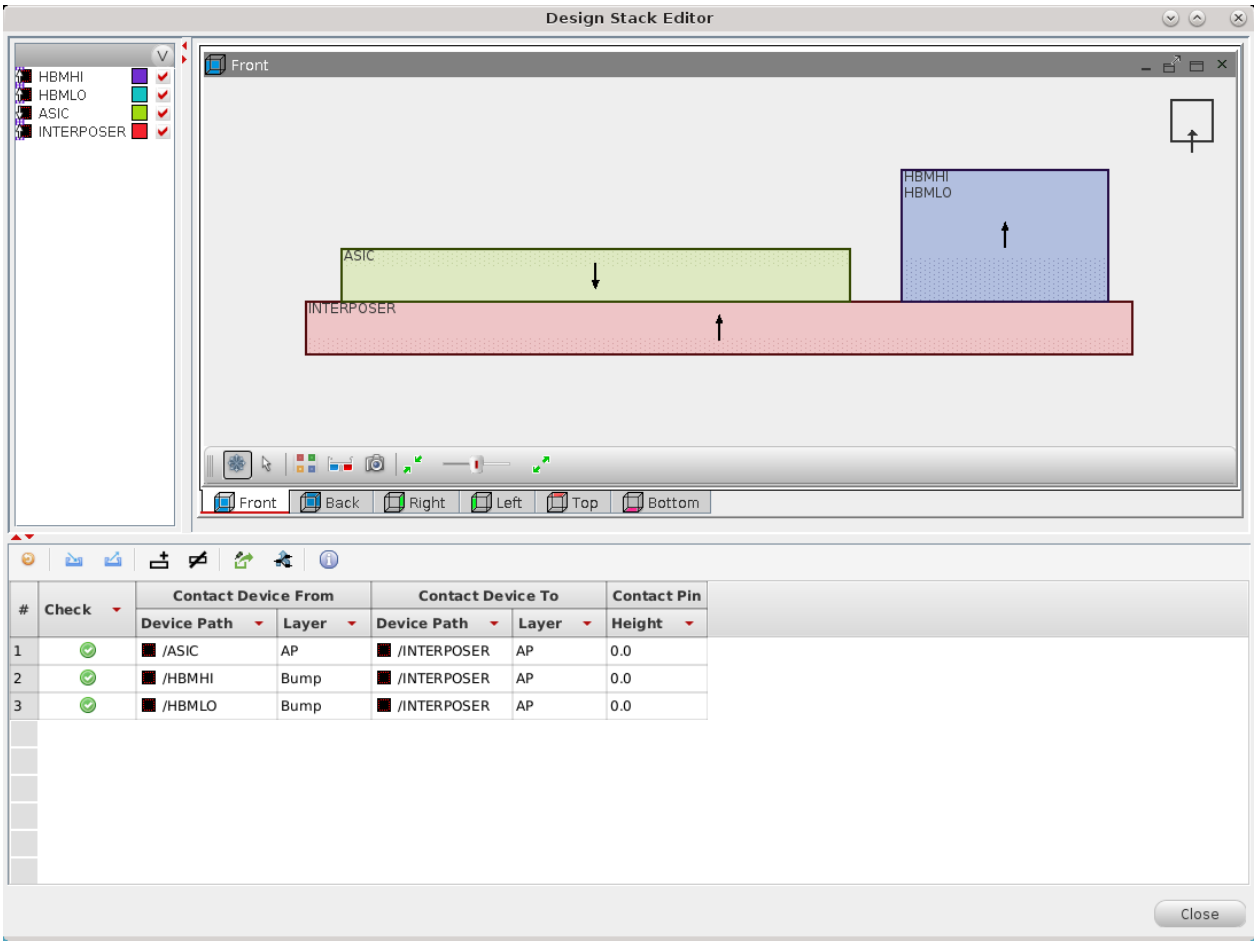
Ruler colors can be defined by clicking on the color palette squares.

The *Clear All Rulers* button will erase all current Rulers.

You can exit from the *Ruler Mode* by selecting any other mode from the main toolbar.

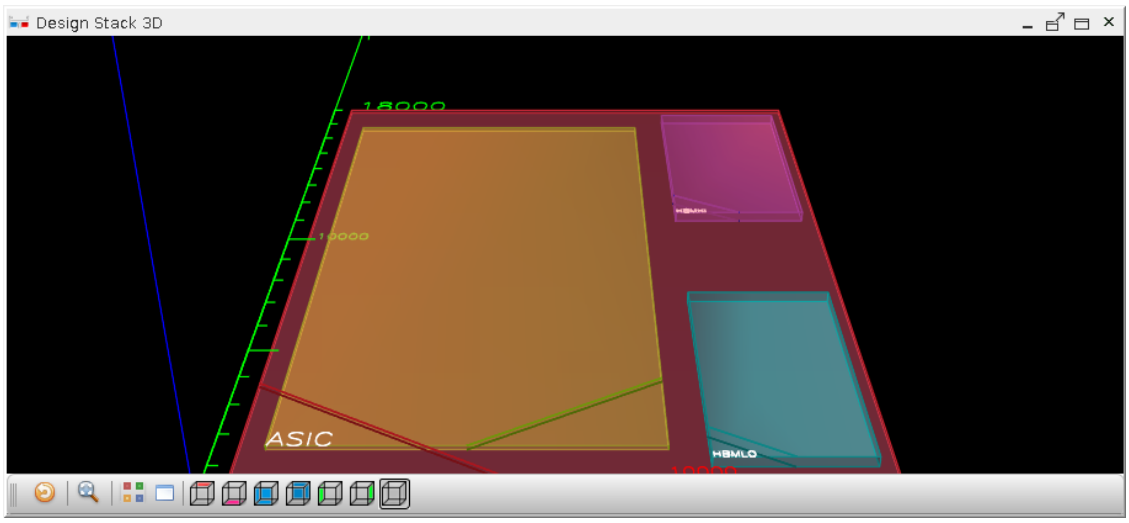
Defining the Design Stackup

The Design Stack Editor is used to visualize the die physical locations in the stack. Select *Edit – Design Stack* to invoke the Design Stack Editor, as shown below.



The Design Stack Editor has two panes:

- The upper pane depicts the die physical locations in the stack as a diagram. For the most part, the diagram is used only as a visual aid and not for actually determining the contact layers. Devices can be selected from the list on the left, or by clicking them in the diagram. Devices can be moved up and down in the stack by dragging them in the diagram. Right-click options enable the addition of spacers, device mirroring, and device height adjustment.
  The device height can also be adjusted interactively by selecting the device on the diagram and dragging it up or down to a new height.
  By default, the *Front* perspective is displayed, as shown above. The view perspective can be changed by clicking any of the other buttons, such as *Back* or *Right*, at the bottom of the pane. The view zoom level can be adjusted with the slide bar.
  The *Layout 3D* button launches a 3D representation of the die stack. The resulting window should be closed if it fails to render the design.



- The lower pane displays the Contact Layer chart. Refer to the Creating Contact Layers and Contact Devices section for more information about using the chart.

Creating Contact Layers and Contact Pads

Contact layers define the relationship between the devices in the design that contact each other. They can be defined between die, die to interposer or info, die to package, and package to board. When creating a contact layer, the source layer and contacted layers are selected. They can be any layer in the substrate and do not have to be the top and bottom layers.

By default, contact pads are created from all the pins in the source device. A selected set of pins can also be used to create contact pads between two devices. This is helpful when only a portion of the device pins contact the other device, such as with bridge applications. It is also helpful if multiple pad sizes are used in the source device. Each pad size can be selected using *Selection Filter* and the corresponding contact pad can be created.

Creating a contact layer also establishes a netlist relationship between the two devices in the design. When creating a contact layer with contact pads, the child device nets automatically appear in the parent-level netlist. The net names can be redefined with net prefixes or mapped to existing top-level nets after the initial contact layer and contact pads are created. A lot of options are available for defining the netlist relationship and mapping of the two devices.

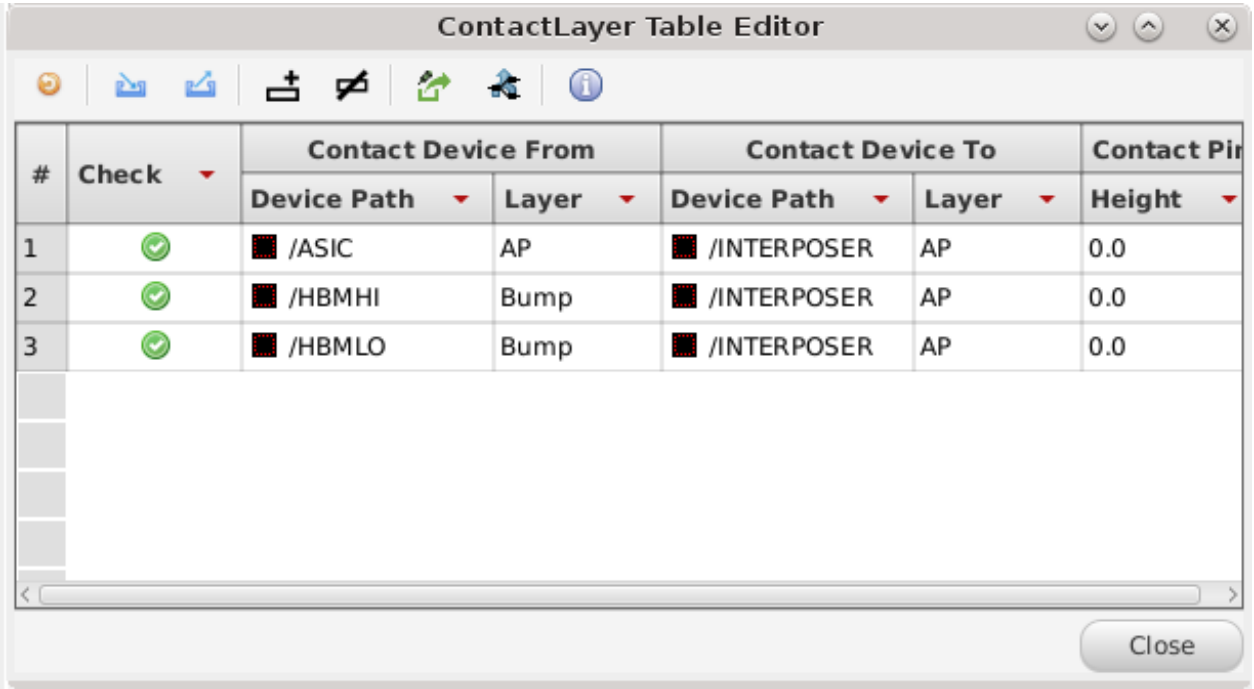
Refer to the Defining or Importing Netlist Connectivity section for more information.

Creating Contact Layers

Contact layers can be created using several methods. The *Edit – Contact Layer* and *Edit – Design Stack* commands allow contact layers to be defined in the*Contact Layer* chart. A contact layer is also defined automatically when a hierarchy is created in the Device Hierarchy.

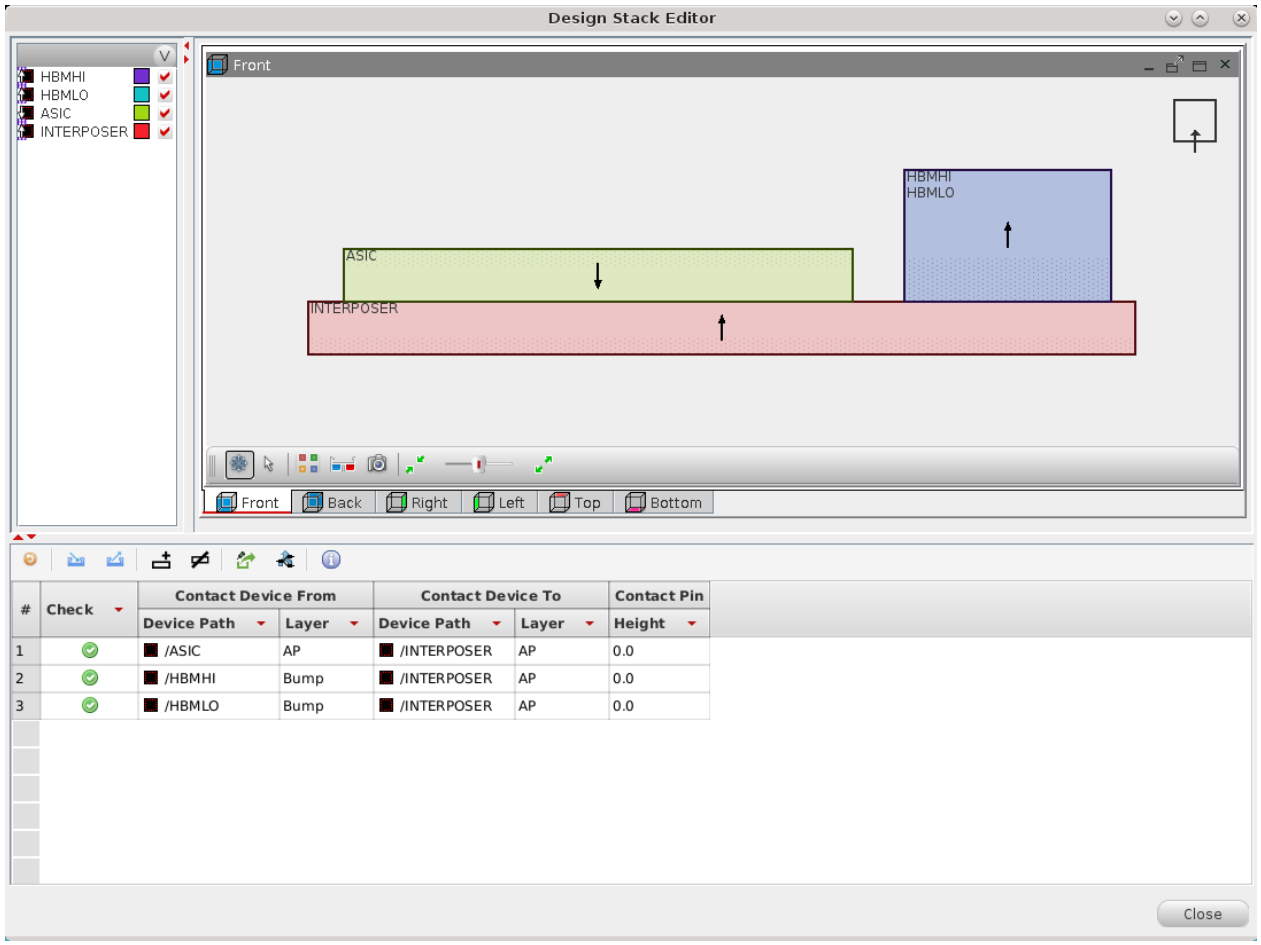
To create a contact layer, follow these steps:

1. Select the *Edit – Contact Layer* command to invoke the ContactLayer Table Editor, as shown below.



#	Check	Contact Device From		Contact Device To		Contact Pir
		Device Path	Layer	Device Path	Layer	Height
1	✓	/ASIC	AP	/INTERPOSER	AP	0.0
2	✓	/HBMHI	Bump	/INTERPOSER	AP	0.0
3	✓	/HBMLO	Bump	/INTERPOSER	AP	0.0

Alternatively, you can use the *Edit – Design Stack* command. In this case, the Contact Layer table is displayed in the lower pane of the Design Stack Editor as shown below.



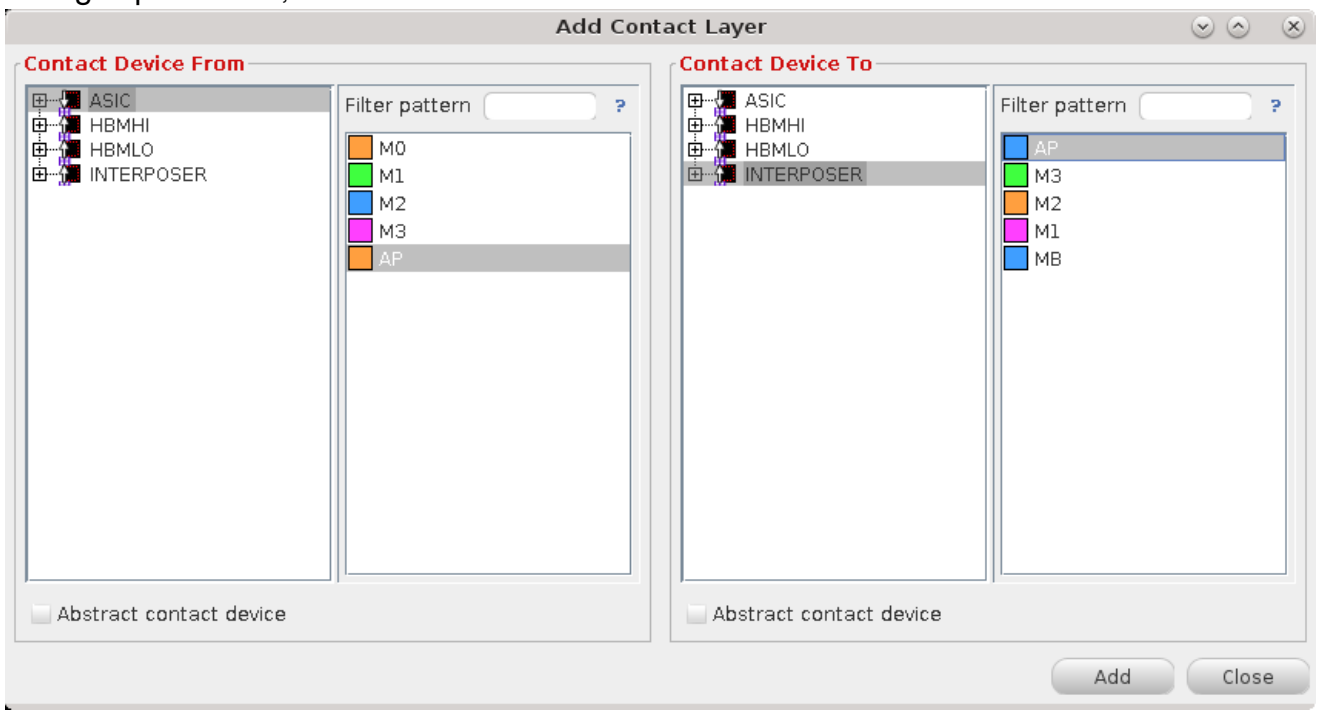
#	Check	Contact Device From		Contact Device To		Contact Pin
		Device Path	Layer	Device Path	Layer	Height
1	✓	/ASIC	AP	/INTERPOSER	AP	0.0
2	✓	/HBMHI	Bump	/INTERPOSER	AP	0.0
3	✓	/HBMLO	Bump	/INTERPOSER	AP	0.0

As contact layers are created, they appear in the contact layer chart, as shown above. Notice in this example, the *Contact Device From* column shows the *ASIC AP* layer and the two *HBM Bump* layers contacting the *Contact Device To - INTERPOSER AP* layer.
The toolbar icons provide the following functionality:

- *Refresh*: Refreshes the chart to reflect any recent changes.
- *Import/Export Contact Layer CSV*: Allows import and export of contact layers.
- *Add Contact Layer*: Invokes the *Add Contact Layer* dialog to add a new contact layer.
- *Delete Contact Layer*: Removes the selected contact layer.
- *Write contact pin map file*: Exports a CSV file containing the pin mapping between the devices for the selected contact layer.

- *Auto-Connect Nets*: Automatically derives connectivity for all aligned pads on the two contact devices.
- *Details*: Invokes the Details dialog for the selected device.

2. To create a new contact layer, click the *Add Contact Layer* ( ) icon. The *Add Contact Layer* dialog is presented, as shown below.

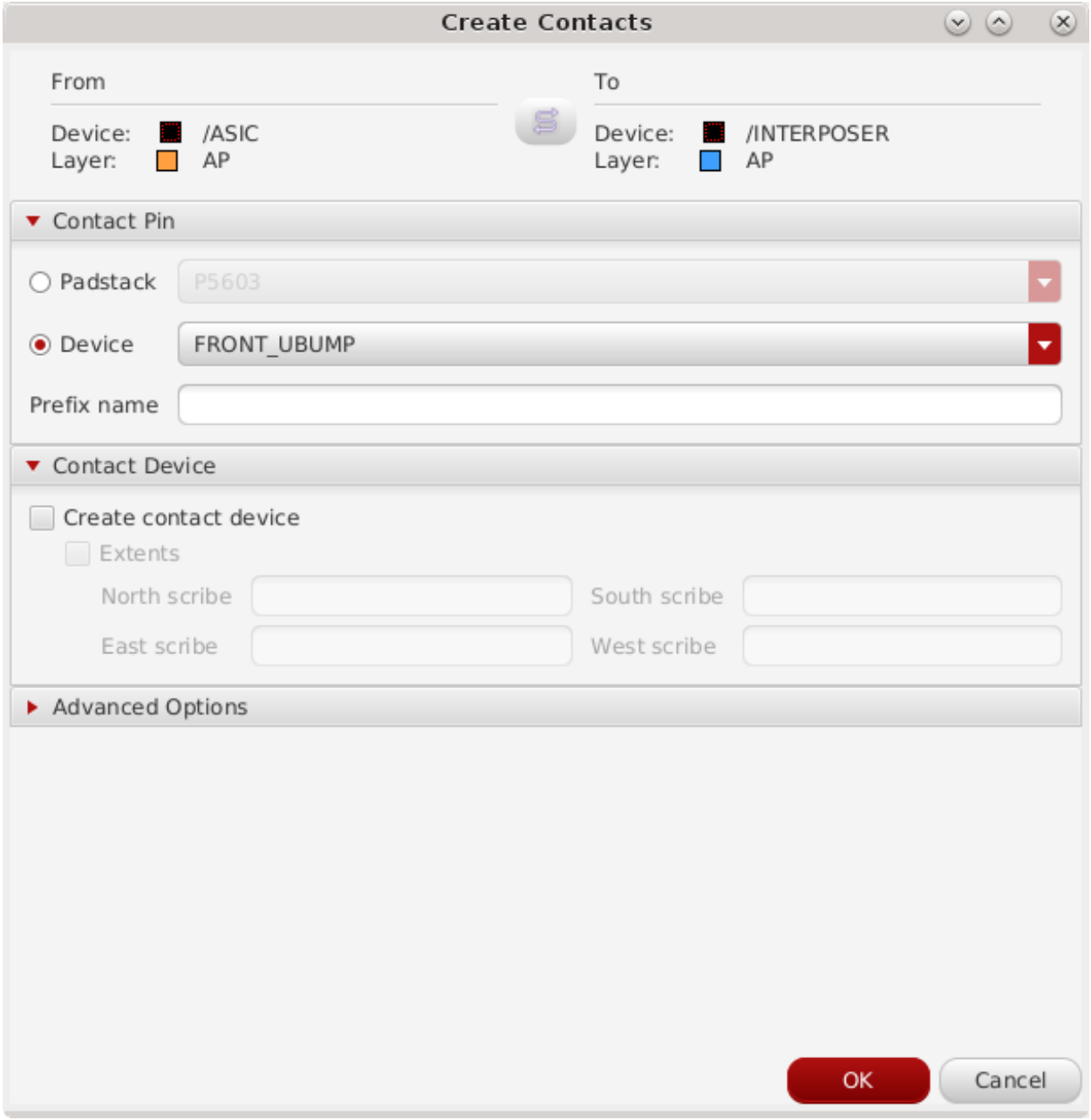


3. Select the source in the *Contact Device From* list on the left and the corresponding contacting layer; then select the target from the *Contact Device To* list on the right and the corresponding contacted layer, as shown above with the two *AP* layers.
4. Click the *Add* button to create the contact layer and display the following dialog to create contact pads or a contact device.

5. Clicking *No* will simply establish the contact layer without creating contact pads or a contact device. They can be added later, if desired. The child device nets are not automatically

propagated to the parent-level netlist. They do appear under the parent and are available for propagation and mapping to parent-level nets.

Clicking *Yes* will present the Create Contacts dialog, as shown below.



Creating Contact Pads

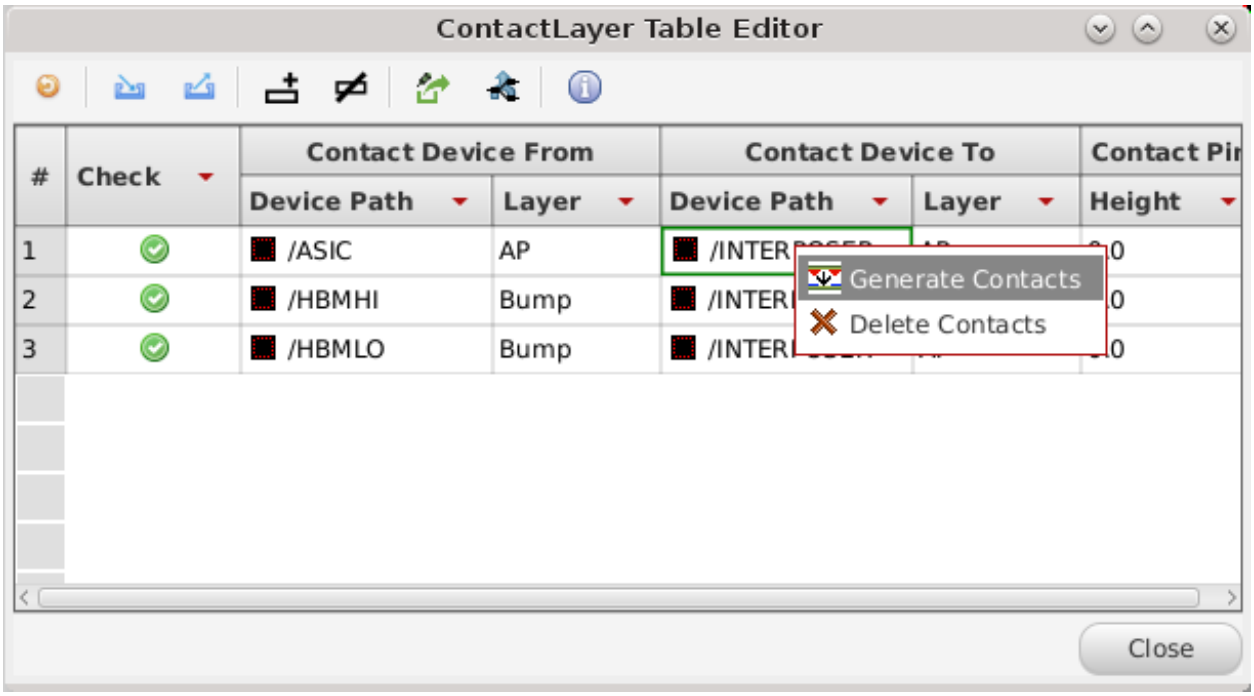
Creating a contact layer provides the option to also create contact pads on the contacted device, as shown above. This option may only be presented if the two devices are placed with one device within the boundary of the other. In other words, with one on top of the other in the floorplan.

Contact pads can be created using an existing padstack defined in the contacted substrate, or they can be created on-the-fly to match the geometry of the source contact pads. They can also be a TSV padstack.

By default, contact pads are created from all the pins in the source device. A selected set of pins can also be used to create contact pads between two devices. This is helpful when only a portion of the device pins contact the other device, such as with bridge applications. It is also helpful if multiple pad sizes are used in the source device. Each pad size can be selected using *Selection Filter* and the corresponding contact pads can be created selectively.

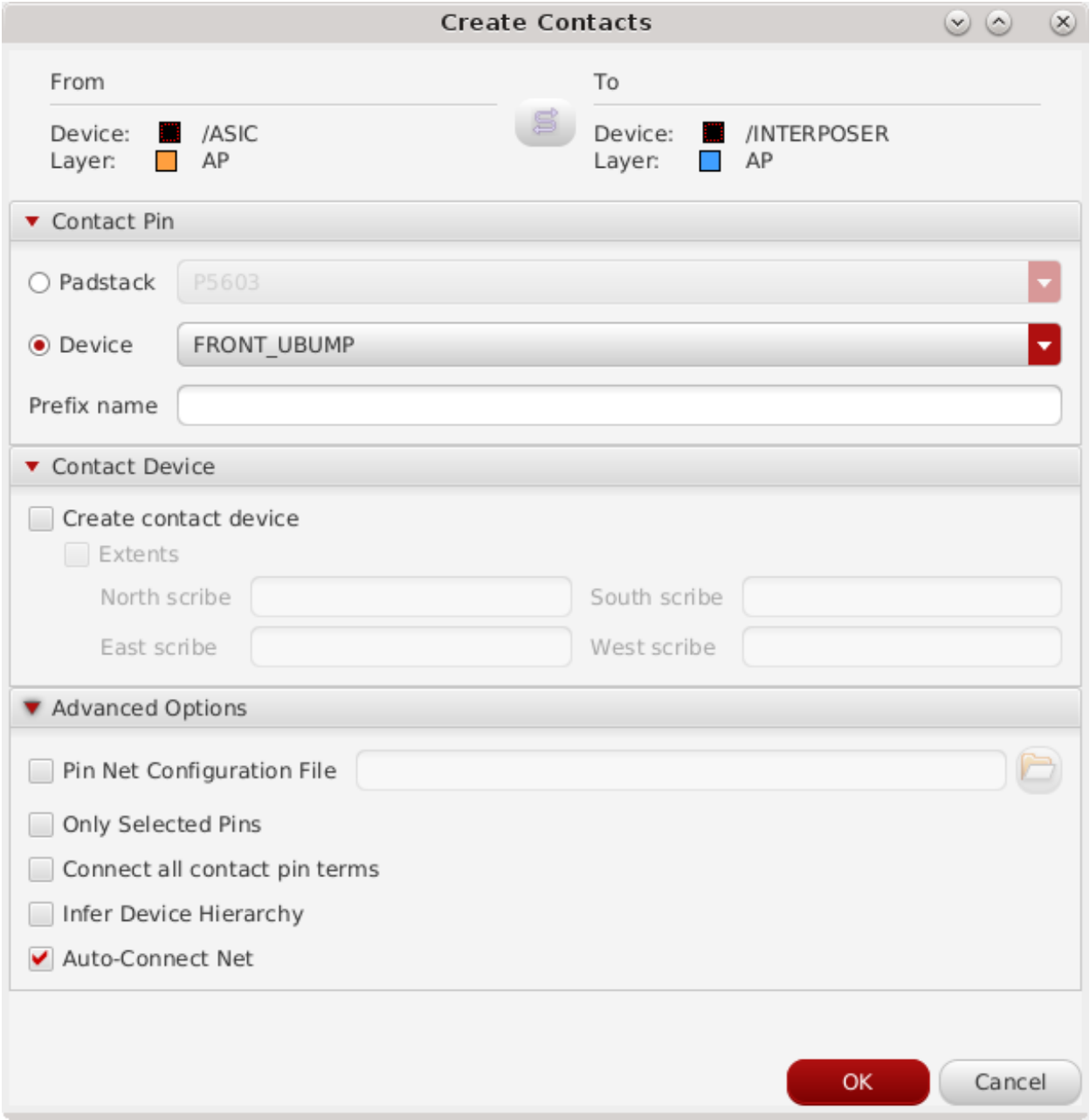
Contact pads can be created in two ways:

- By clicking *Yes* when prompted after creating a contact layer, as described in the Creating Contact Layers section above.
- By right-clicking the appropriate *Device Path* in the ContactLayer Table Editor and choosing *Generate Contacts*, as shown below.



Ensure that the proper *Device Path* in the chart is selected for the correct device. Selecting the *ASIC* device path for *Contact Device A* in the above image would attempt to create contact pads on the ASIC and not the INTERPOSER.

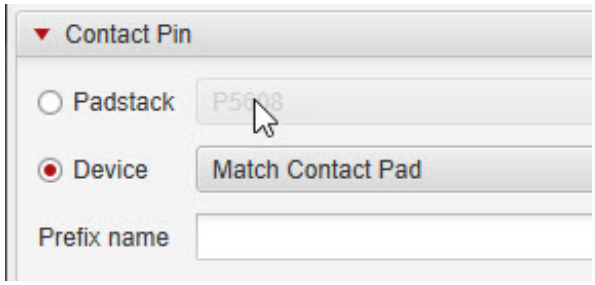
With either method, the *Create Contacts* dialog is presented, as shown below.



In this dialog:

- The top section confirms the source (*From*) and destination (*To*) layers for creating the contact pads.
- In the *Contact Pin* section, select the padstack to be used for the contact pads.
  Select *Device* for IC layout and *Padstack* for package design with APD. If a padstack exists in

the contacted device substrate on the appropriate layer, it will automatically be displayed in the dialog, as shown above with FRONT_UBUMP. If no padstack exists the *Match Contact Pad* option can be used to create a new padstack, as shown below. This option may not be able to create complex pad shapes used in the source device. Padstacks should be created in the contacted device substrate in this case.



- The *Prefix name* field can be used to add a name prefix to all the term and net names being pushed into the contacted device.
- Use the *Contact Device* option to create a contact device, as described in theCreating Contact Devices section below. This option is rarely used for typical IC designs.
- The *Advanced Options* are as follows:
  - *Pin Net Configuration File* – Presents a browser to select a pin configuration file. Pin configuration files are used to customize the generated contact pin's name or the net/term of the contact pins. The format of the csv file is same as the pin map csv, as shown below. Columns "Pin A" and "Pin B" are required, others are optional.

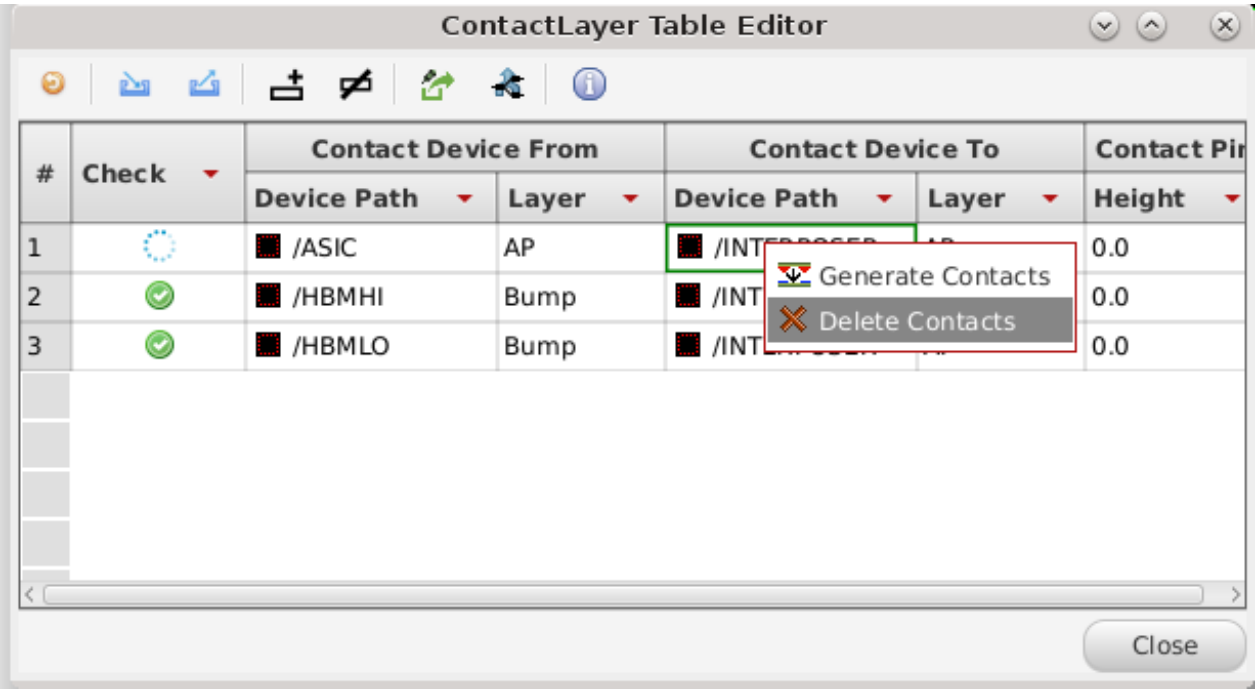
PinPath A	Pin A	Term A	Net A	PinPath B	Pin B	Term B	Net B
.	A128	VSS	VSS	.	ASIC_A128	ASIC_VSS	ASIC_VSS
.	A139	VDDIO	VDDIO	.	ASIC_A139	ASIC_VDDIO	ASIC_VDDIO

  - *Only Selected Pins* – Creates contact pads for only a selected set of device pins and not all the device pins. This option is often used with bridge type applications where only a portion of the device pins contact another device.
  - *Connect all contact pin terms* - Connects all the terms of the contact pins.
  - *Infer Device Hierarchy* – Creates the contact device as a hierarchical child under the contacted device and leaves the original outside of the hierarchy. This option is used along with creating a contact device. This is often preferred when using contact devices.
  - *Auto-Connect Net* – Pushes the source device nets into the contacted device and will assume connectivity for any matching net names. This check box is selected by default.

Managing Contact Pads

System Planner will attempt to maintain contact pads through changes in the design. When a device with contact pads is moved, a prompt is presented to also move the contact pads. As net names are changed or removed, the contact pads are adjusted accordingly. If major placement changes are being made, delete the contact pads and regenerate them after the moves are completed.

Occasionally, contact pads need to be removed or regenerated. Before regenerating, it is advised to first remove the existing contact pads. To remove contact pads, right-click the appropriate *Contact Device* in the ContactLayer Table Editor and choose *Delete Contacts*, as shown below.



Ensure that the correct *Device Path* in the chart is selected for the device on which you desire to remove contact pads. Selecting the *ASIC* device path for *Contact Device A* in the above image would attempt to delete contact pads on the ASIC and not the INTERPOSER.

Creating Contact Devices

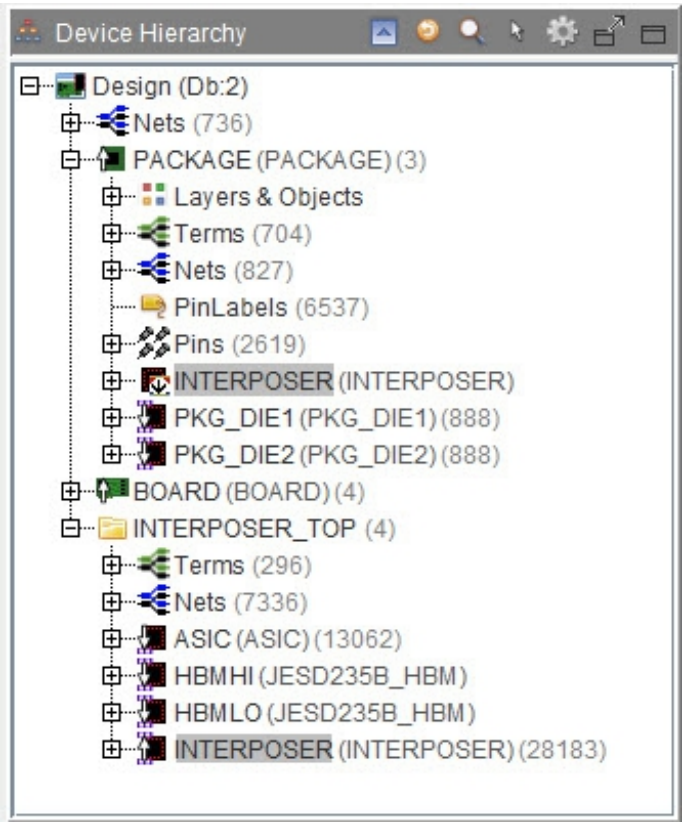
Contact devices are rarely used with typical 3D-IC designs, as they affect the Verilog netlist hierarchy. This same functionality can also be achieved by using copies of the device that share a common device template.

Creating a contact device creates a copy of the device in the Device Hierarchy consisting of only the pad contact definition and pattern. This is sometimes useful when multiple copies of a device are needed. Contact devices can be copied while the original device is used to manage all the copies. The original device is often located outside of the design hierarchy, while the contact device is within it.

Using a contact device ensures a single contact layer/device is used.

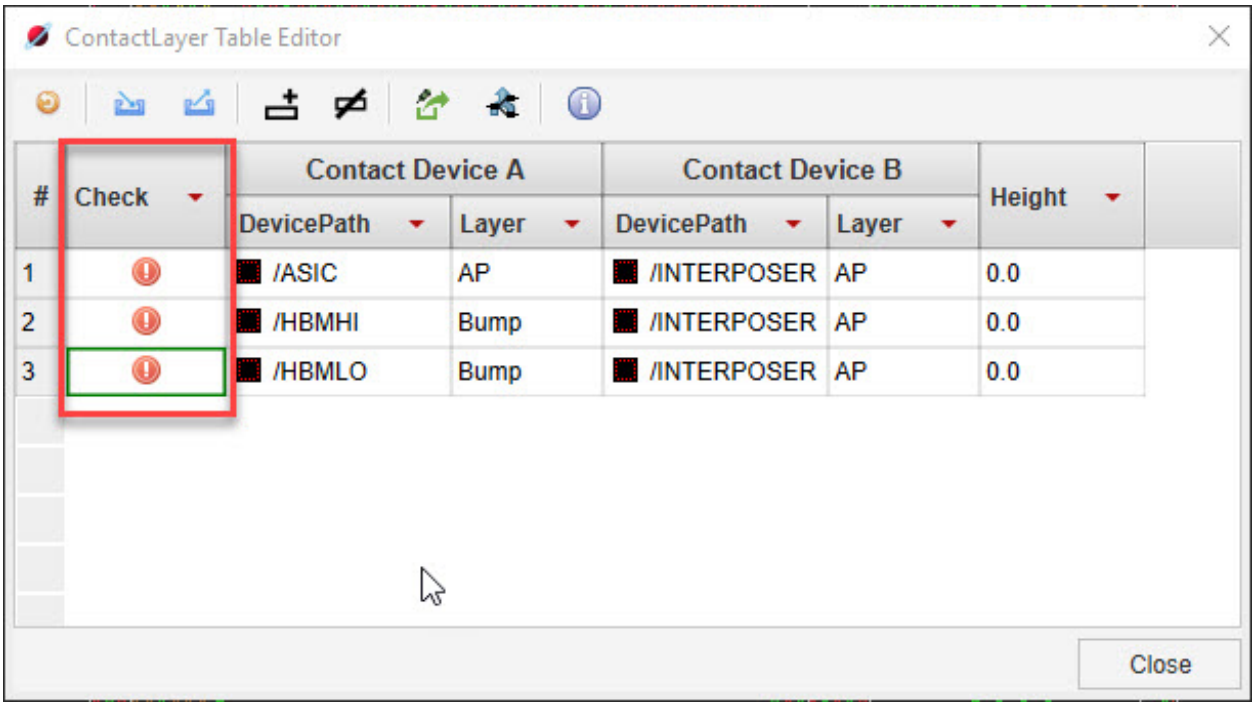
Contact devices are also sometimes used when a device's contacts are buried in a layer within the target substrate.

Contact devices are also useful when a co-design project has multiple levels of hierarchy and a device needs to exist in both, as shown below with the INTERPOSER contact device under the PACKAGE hierarchy.

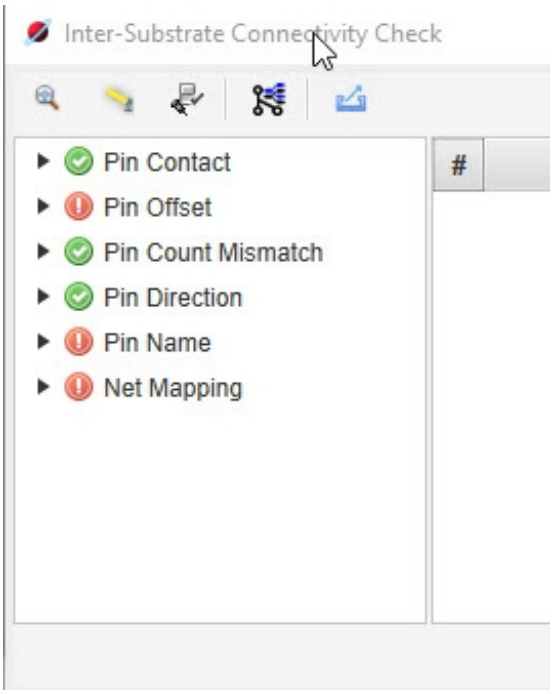


Running Inter-Substrate Connectivity Checks

As contact layers are defined, a set of Inter-Substrate Connectivity Checks are automatically performed. The *Check* column in the ContactLayer Table Editor displays a red or green circle to indicate the status of these checks, as shown below.



Clicking on the circle will present the results in*the Inter-Substrate Connectivity Check*table, as shown below. Refer to the Exploring Results section for more information.

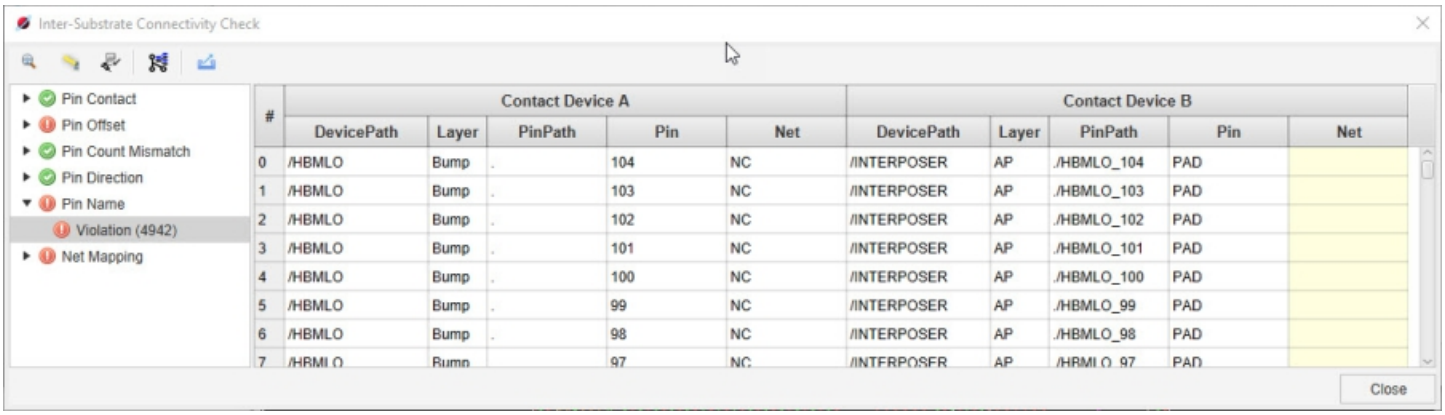


The checks can also be performed by selecting *Tools – Check Inter-Substrate Connectivity*. The *Check Inter-Substrate Connectivity*dialog is presented, as shown below.

You can select the rules required to be run as part of the inter-substrate connectivity check in this dialog. For example, select the *Pin contact* check to ensure that the pins are contacting each other on each device. Similarly, select the *Pin offset* option to verify that the contacted pins meet the maximum allowable pin offset limit, which you can set in the field provided.

Exploring Results

The Inter-Substrate Connectivity Check chart is presented after the rules have been run, as shown below.



#	Contact Device A					Contact Device B				
	DevicePath	Layer	PinPath	Pin	Net	DevicePath	Layer	PinPath	Pin	Net
0	/HBMLO	Bump	.	104	NC	/INTERPOSER	AP	./HBMLO_104	PAD	
1	/HBMLO	Bump	.	103	NC	/INTERPOSER	AP	./HBMLO_103	PAD	
2	/HBMLO	Bump	.	102	NC	/INTERPOSER	AP	./HBMLO_102	PAD	
3	/HBMLO	Bump	.	101	NC	/INTERPOSER	AP	./HBMLO_101	PAD	
4	/HBMLO	Bump	.	100	NC	/INTERPOSER	AP	./HBMLO_100	PAD	
5	/HBMLO	Bump	.	99	NC	/INTERPOSER	AP	./HBMLO_99	PAD	
6	/HBMLO	Bump	.	98	NC	/INTERPOSER	AP	./HBMLO_98	PAD	
7	/HBMLO	Bump		97	NC	/INTERPOSER	AP	./HBMLO_97	PAD	

Expanding the rule categories on the right and selecting a rule with violations will display those violations in the table, as shown above. The violations are often shown highlighted in yellow, as shown above.
The toolbar presents the following functionality:

- *Zoom to* will zoom into the canvas to show the design object violating the selected rule.
- *Show me* will highlight the object violating the selected rule on the canvas, often with a dragged connection line.
- *Select* will select the design object that the violation is reporting.
- *Graph net* will open the Graph net dialog with the offending net.
- *Export to CSV* will export the current results to a CSV file.

Defining or Importing Netlist Connectivity

System Planner supports multiple methodologies and design tools with differing netlist requirements. Connectivity can be defined using a variety of methods. System Planner simply needs a pad with a net name to establish connectivity for Allegro® Package Designer Plus, Allegro® PCB Editor, and Virtuoso® Layout.

For Integrity 3D-IC, a top-level Verilog term-based netlist is required for implementation. Connectivity can be established by importing one or more Verilog netlists or by preparing term list and term map files, as described in this section.

The imported Verilog netlists for each die should only contain the structural part of the design, including external signals and macro-level connections.

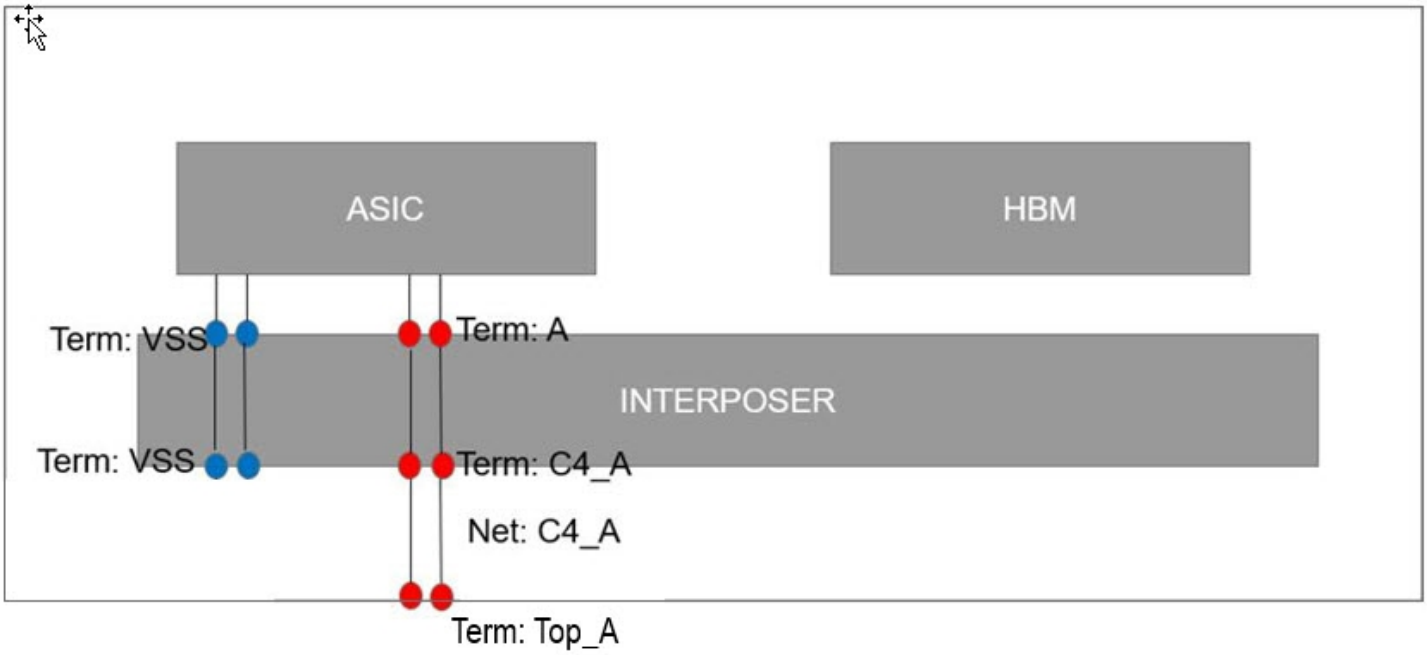
System Planner has no use for internal macro connectivity. Including internal macro connectivity in the imported Verilog netlists may result in overly large project files and reduced tool performance.

As the Verilog files are imported, System Planner will populate existing devices with matched names with the net connectivity from the Verilog module. If devices do not exist, System Planner will create them automatically. If macros exist in the device Verilog files, they are automatically created under the COREs directory in the Device Hierarchy, as shown in the Importing Verilog Netlists section.

The recommended sequence is to create devices in System Planner before importing the Verilog netlists.

Understanding the Top-Level Design Requirements

Each die, interposer, info, and so on must be instantiated as a module declaration in a top-level netlist. Defining each device with a Verilog module declaration ensures it is handled as a unique device, and not a macro, in iHDB and Innovus. The top-level netlist needs to contain terms for each external net, such as those connecting to the package substrate, as shown below.

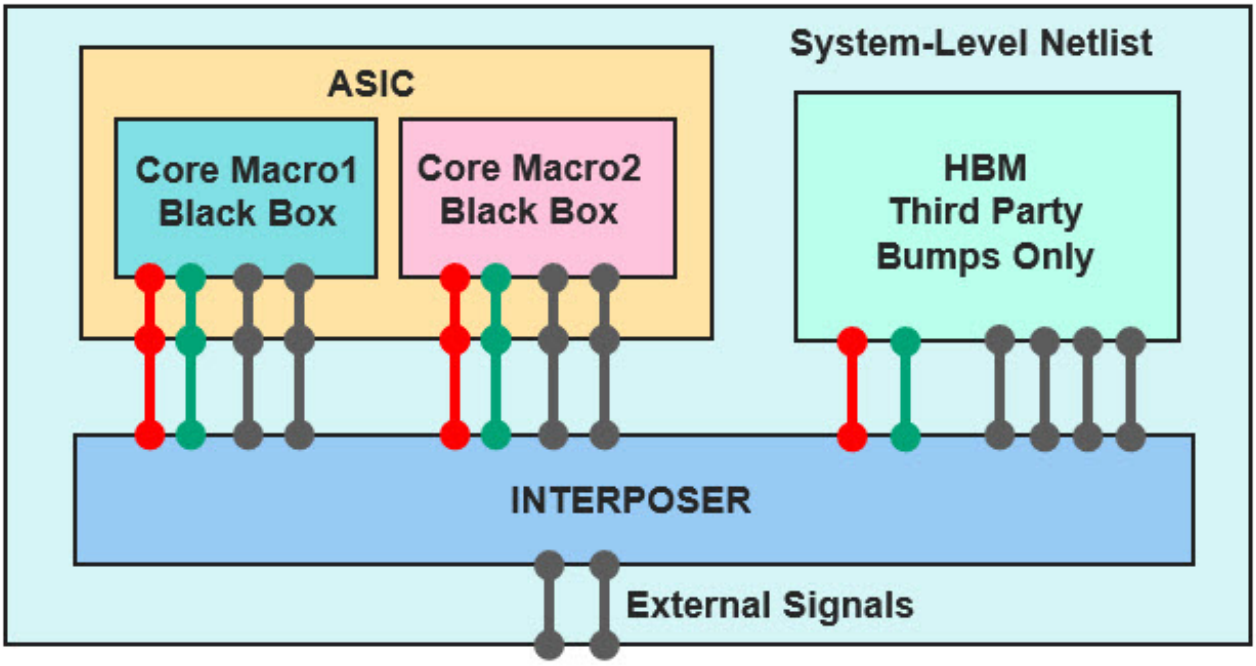


The top-level netlist is represented with the largest rectangle shown above encompassing the entire design. The *Top_A* term, in this case, represent the signals that are connected externally to the package substrate.

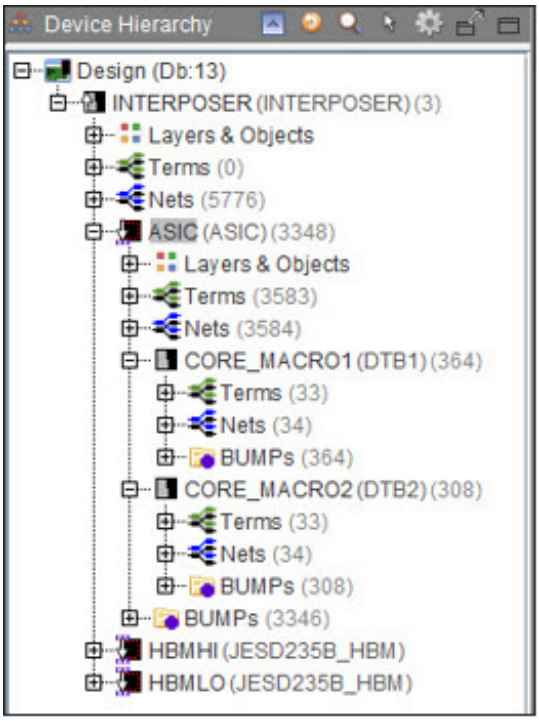
⚠ To facilitate easy selection of these external signals for C4 bump generation, they should be named with a unique net name prefix (for example, **C4_**).

Work at an Abstract Level using Black Box Die Macros

System Planner is best used for 3D-IC design assembly with a structural netlist that contains black boxes for internal die core macros, as shown below. This reduces the database size, improves performance, and also enables die designers to work in parallel with the top-level design planning efforts. This also allows for optimization of die bumps based on die to die or external connections. The die core macro structure will match the Verilog netlist structure exactly.



Work with Structural Netlist Only

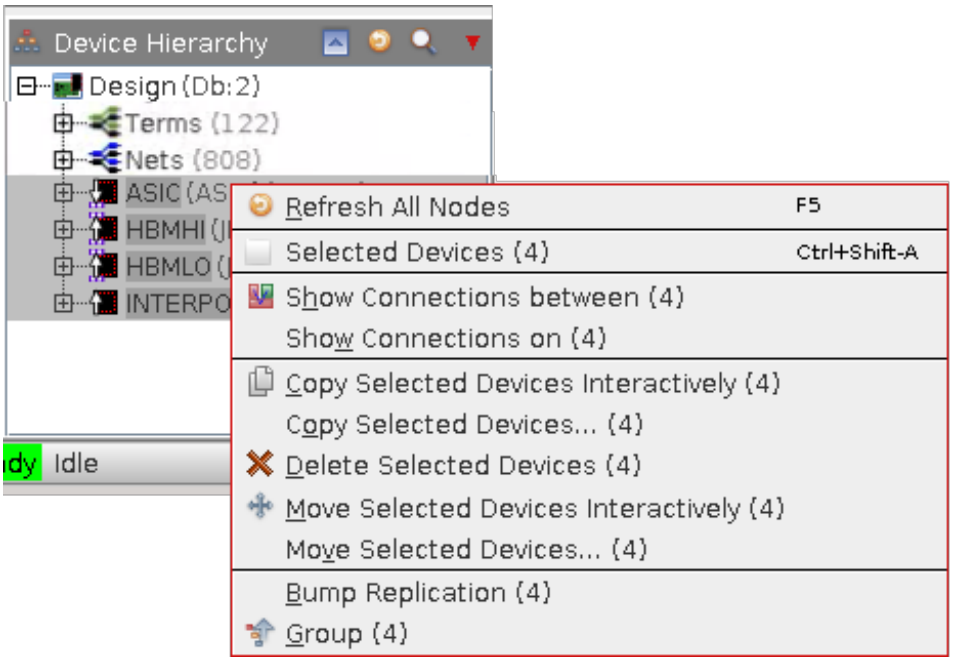


Creating the Top-Level Design Hierarchy

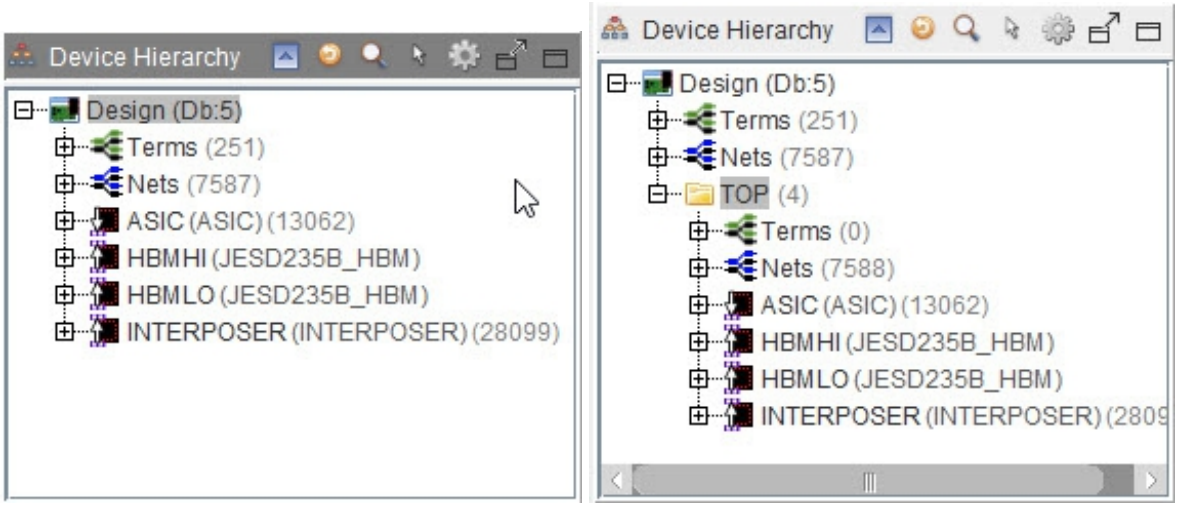
Prior to defining connectivity, the hierarchical relationship between the devices must be established. The device names and hierarchical relationship in System Planner must match the Verilog files exactly, including the top-level.

There are two ways to create a top-level design. When a new project is created in System Planner, a dialog is presented to enter the top-level name. The default is *Design*, as shown in the image on the left below.

You can also use the *Group* command to establish a top-level design. Select the devices that are to be grouped and right-click to choose *Group* from the shortcut menu.



Alternatively, select *Edit – Devices – Group Selected* from the menu bar. On clicking the *Group* command, the top-level folder is created with *UserGroup* as the default name. You can change the group and template name by selecting it and entering a new name in the Attribute Editor. In the image on the right below, the top-level folder name has been changed to *TOP*.



The Verilog top-level module name must match the name of the top-level in the System Planner project.

The examples above represent the recommended device hierarchy, with each device on the same level of hierarchy as the children of a top-level design.

Importing Verilog Netlists

Connectivity can be established by importing one or more Verilog netlists. If Verilog is being imported into an existing System Planner design with devices and hierarchy already defined, the module names and hierarchical format must exactly match the ones in the file.

As the Verilog files are imported, connectivity is populated in any existing devices with names matching the Verilog module.

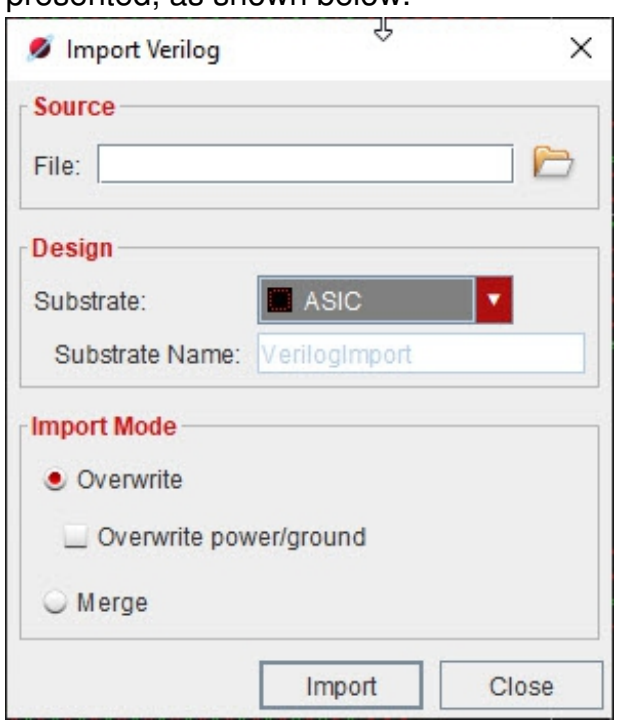
If devices do not exist, System Planner will create them automatically.

> The recommended practice is to create devices in System Planner before importing the Verilog netlists.

If macros exist in the device Verilog files, they are automatically created under the COREs directory in the Device Hierarchy.

To import a top-level or device-level Verilog netlist:

1. Right-click the project top-level and choose *Import – Verilog*. The *Import Verilog* dialog is presented, as shown below.



2. Specify the Verilog file to be imported in the*File* field.
3. Use the *Design* section of the dialog to specify different use models. Verilog can be imported

into an existing System Planner design that has matching device names and hierarchy or it can be imported to create the devices and hierarchy based on the input file content.

- If Verilog is being imported into a System Planner design that has existing matching devices, the *Substrate* entries are ignored and the netlist is imported and applied accordingly.
- If the Verilog netlist is being imported to automatically create the System Planner devices, select a *Substrate* to create the devices. Select *Create New* and then enter *Substrate Name* to create a new substrate.

4. In the *Import Mode* section, select *Merge* to merge the imported netlist with the existing netlist in the project. Alternatively, you can overwrite the existing netlist and also overwrite the power/ground nets in the project.
5. Click *Import*.

Creating Connectivity with Term Files

If a top-level Verilog file is not available, connectivity can be defined by preparing term list and map files. To create term-based connectivity that enables Verilog netlist output for Innovus, a physical terminal "term" is required for each die contact pad. Terms can be defined by importing term list files and also when contact pads are created. Each net is propagated to the top-level netlist as terms are created. The terms are then mapped to each other using the term map files to create connectivity.

Terms are created by importing a term list for each device. Net and term prefix names can be used to make the names unique or to better identify the nets at the top-level. The term names defined in the term lists must be the same names used when creating the contact pads for each die, because these act as the physical terms.

This process is described and used in the *Designing with the Integrity 3D-IC Platform RAK*.

You can also create terms easily for most devices by using the *Netlist Editor*. Refer to the Creating Terms Using the Netlist Editor section for more information.

Preparing Term List Files

Term list files are used to map device term names to top-level net names. You can use these term list files to map differing die net names to a single net name in the top-level netlist. This is the method used to connect die with differing net names.

It is often easier to prepare a separate term list and term map file for each die in the design, as well as for the top-level external nets. If the net names you wish to connect differ between dies, you

must first decide which net name you would like to use in the top-level netlist.

To prepare a starter term list file, right-click the device in the Device Hierarchy and select the *Export – Term List* command.

An example of completed term list file is shown below.

	A	B	C	D
1	Term	Net	Type	Use
2	HBMHI_AERRA	HBMHI_AERRA	INOUT	SIGNAL
3	HBMHI_AERRB	HBMHI_AERRB	INOUT	SIGNAL
4	HBMHI_AERRC	HBMHI_AERRC	INOUT	SIGNAL
5	HBMHI_AERRD	HBMHI_AERRD	INOUT	SIGNAL
6	HBMHI_AERRE	HBMHI_AERRE	INOUT	SIGNAL
7	HBMHI_AERRF	HBMHI_AERRF	INOUT	SIGNAL
8	HBMHI_AERRG	HBMHI_AERRG	INOUT	SIGNAL
9	HBMHI_AERRH	HBMHI_AERRH	INOUT	SIGNAL
10	HBMHI_ARFUA0	HBMHI_ARFUA0	INOUT	SIGNAL
11	HBMHI_ARFUA1	HBM2_PHY_HI_ARFUA1	INOUT	SIGNAL
12	HBMHI_ARFUA2	HBMHI_ARFUA2	INOUT	SIGNAL
13	HBMHI_ARFUA3	HBM2_PHY_HI_ARFUA3	INOUT	SIGNAL
14	HBMHI_ARFUB0	HBMHI_ARFUB0	INOUT	SIGNAL

Here:

- The *Term* column defines the name of the term to be defined for the device at the top-level. In this example, the *HBMHI_* prefix was added to the device net names when the contact pads were created to make the names unique compared to other HBMs.
  As contact pads are created, terms are also created matching the net names. In this case, the term names used in the term list file must match the contact pad names in the design.
- The *Net* column defines the top-level net with which to connect the term. Notice in the example above, the highlighted line 11 maps the term to a different top-level net name from another die. This is how connectivity is established between die with different net and term names.
- The *Type* column defines the net and term type. The available values are IN, OUT, INOUT, and FEEDTHRU.
- The *Use* column defines the type of term. The values include ANALOG, CLOCK, GROUND, POWER, REESET, SCAN, SIGNAL, TIEOFF, NC, and UNKNOWN. Typically SIGNAL, POWER, and GROUND are the only values used for 3D-IC designs.

Preparing Term Map Files

A term map file maps child (die) and parent term names. It is often easier to create a separate term map file for each device in the design.

An example of term map file is shown below:

	A	B	C
1	Child Device	Child Term	Parent Net
2	ASIC	APB_PCLK	ASIC_APB_PCLK
3	ASIC	APB_PRESET_N	ASIC_APB_PRESET_N
4	ASIC	AUX_ATB_CORE_0	ASIC_AUX_ATB_CORE_0
5	ASIC	AUX_ATB_CORE_1	ASIC_AUX_ATB_CORE_1
6	INTERPOSER	ASIC_APB_PCLK	ASIC_APB_PCLK
7	INTERPOSER	ASIC_APB_PRESET_N	ASIC_APB_PRESET_N
8	INTERPOSER	ASIC_AUX_ATB_CORE_0	ASIC_AUX_ATB_CORE_0
9	INTERPOSER	ASIC_AUX_ATB_CORE_1	ASIC_AUX_ATB_CORE_1

In the example above, the ASIC Child Device is sitting on top of an INTERPOSER. For easier file preparation, both child devices are included in this single file because they use the same top-level net names. In the term map file:

- *Child Device* defines the child device from which the term is being mapped.
- *Child Term* defines the term name in the child device to map.
- *Parent Net* defines the top-level net to which the child term is to be mapped.

In this example, the nets all map to matching parent nets with the *ASIC_* prefix. The *ASIC_* prefix was added to the child net names when the contact pads were created, which is why the INTERPOSER *Child Term* names include the *ASIC_* prefix, as shown above.

Importing Term List and Term Map Files

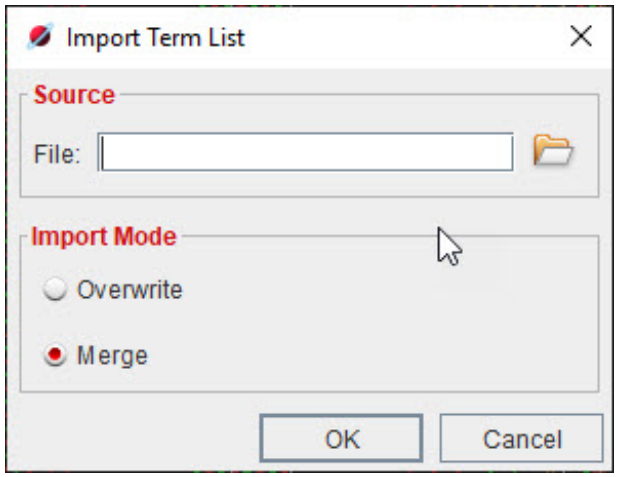
After the term list and term map files are prepared, they can be imported into the System Planner design.

Importing Term List Files

Term list files are imported relative to the device for which the terms are being created.

To import a term list file for a device:

1. Right-click the device in the Device Hierarchy and choose *Import – Term List* to invoke the Import Term List dialog, as shown below.



2. Select one or more term list files for the device in *File*.
3. In the *Import Mode* section, specify whether you want to merge with or overwrite existing terms.
4. Click *OK* to import the term list files.

Review the messages echoed to the *Command and Messages* view to ensure a proper import.

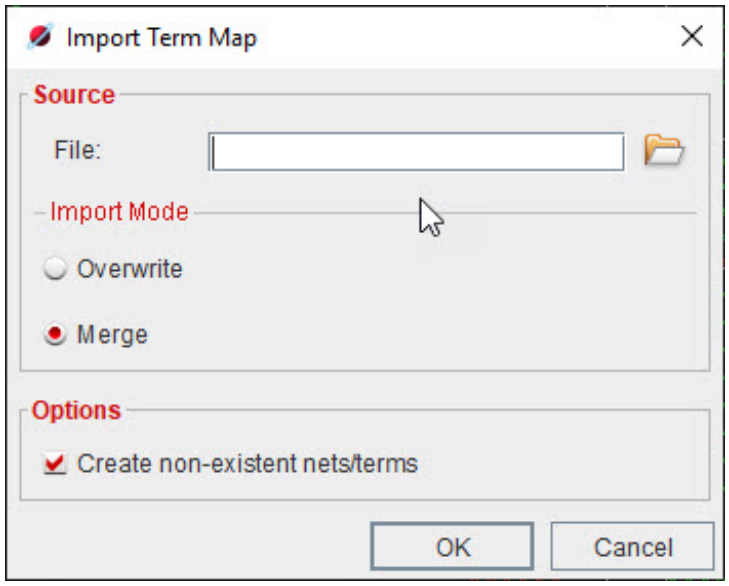
Importing Term Map Files

Term map files are imported at the top-level since the terms are being mapped to the top-level net names.

⚠ Ensure that you import term map files with the top-level of the design hierarchy.

To import a term-map file:

1. Right-click the top-level *Design* and choose *Import – Term Map* to invoke the Import Term Map dialog, as shown below.



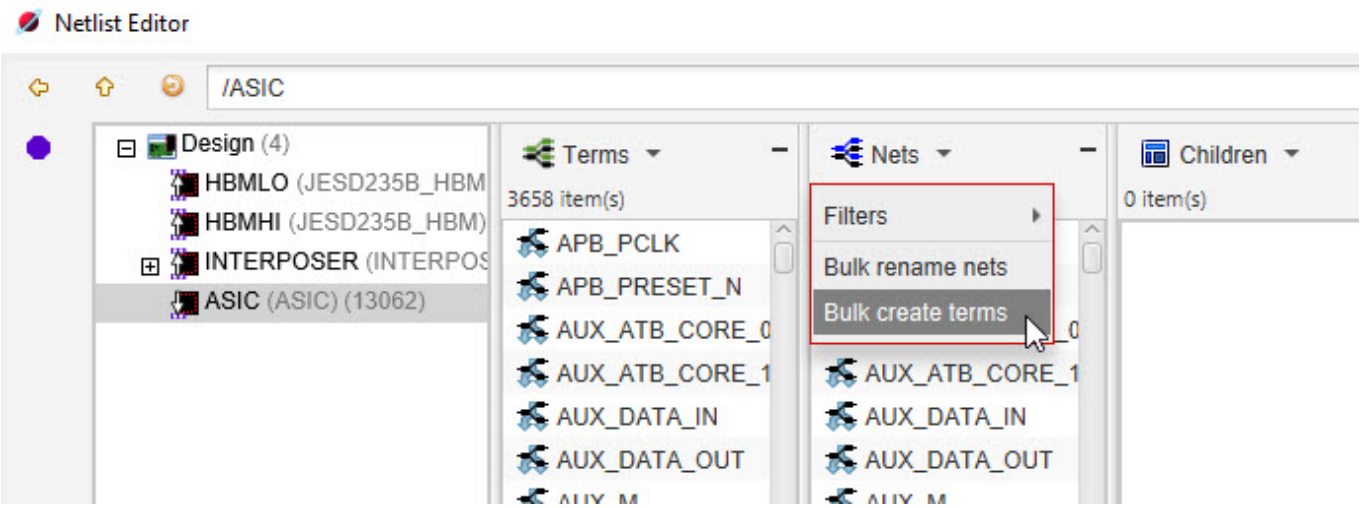
2. Select one or more term map files for the design in*File*.
3. In the *Import Mode* section, specify whether you want to merge with or overwriteexisting mapping.
4. Select *Create non-existent nets/terms* to create terms for objects in the map file that do not exist in the design.
5. Click *OK* to import the term map files.

Review the messages echoed to the*Command and Messages* panel to ensure a proper import.

Creating Terms Using the Netlist Editor

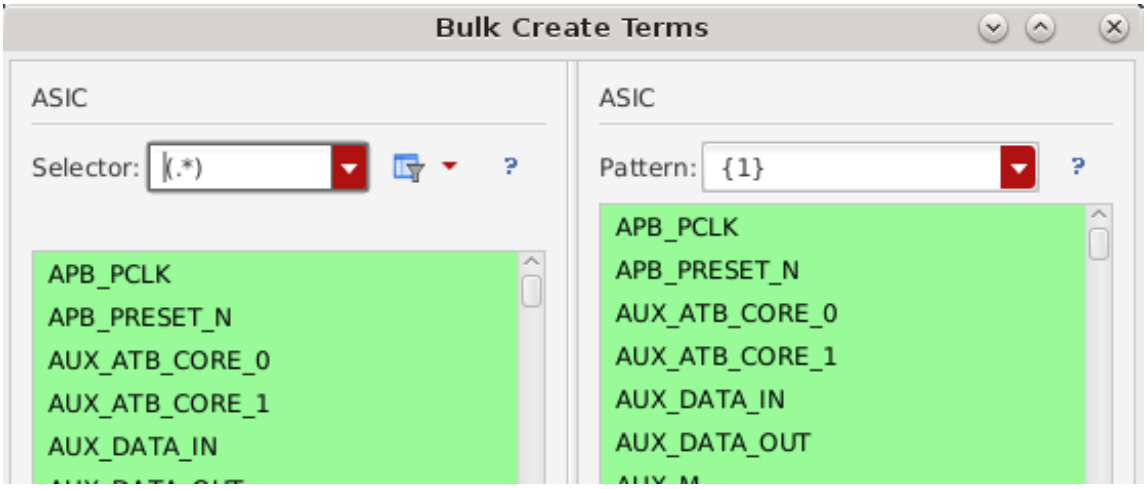
Terms can also be created easily for most devices automatically using the Netlist Editor:

1. Right-click the device in the Device Hierarchy and choose*Netlist* to invoke the Netlist Editor, as shown below.



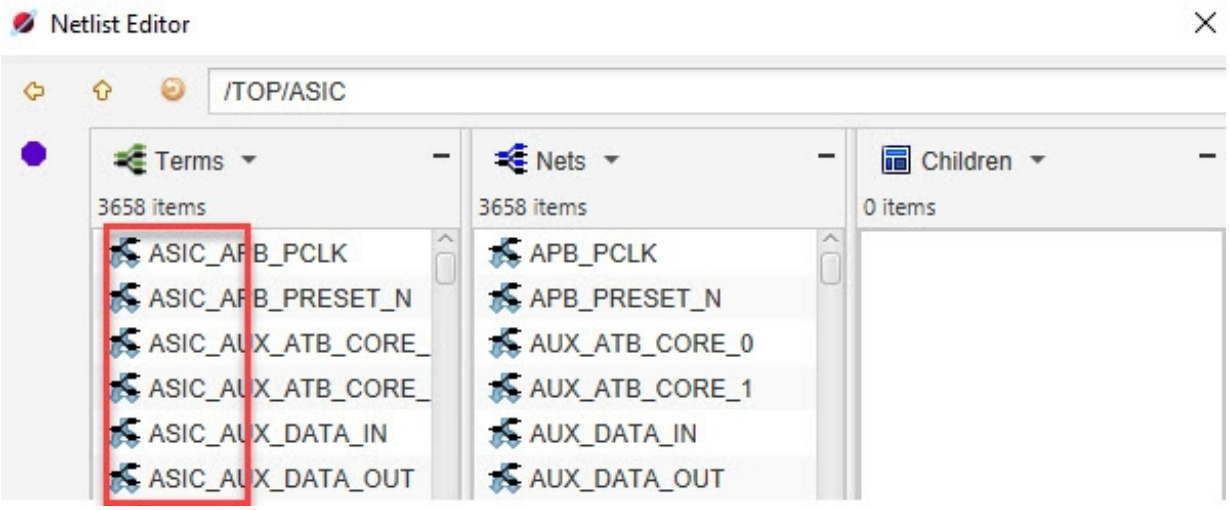
2. You can display any level of the Device Hierarchy inthe Netlist Editor by entering the path in the top field, as shown above. The *Up to* arrow can be used to push up a level of the hierarchy. The nets and terms at each level are displayed, and pull-down menu commands are provided to create terms and nets and to map nets interactively.

3. To create terms, choose *Bulk create terms* from the *Nets* pull-down menu, as shown above. The Bulk Create Terms dialog is invoked, as shown below.



4. Nets can be selected using regular expressions for term creation. Enter a term-name prefix, such as *ASIC_*, as shown above.

5. Click *OK* to create the terms, as shown below.



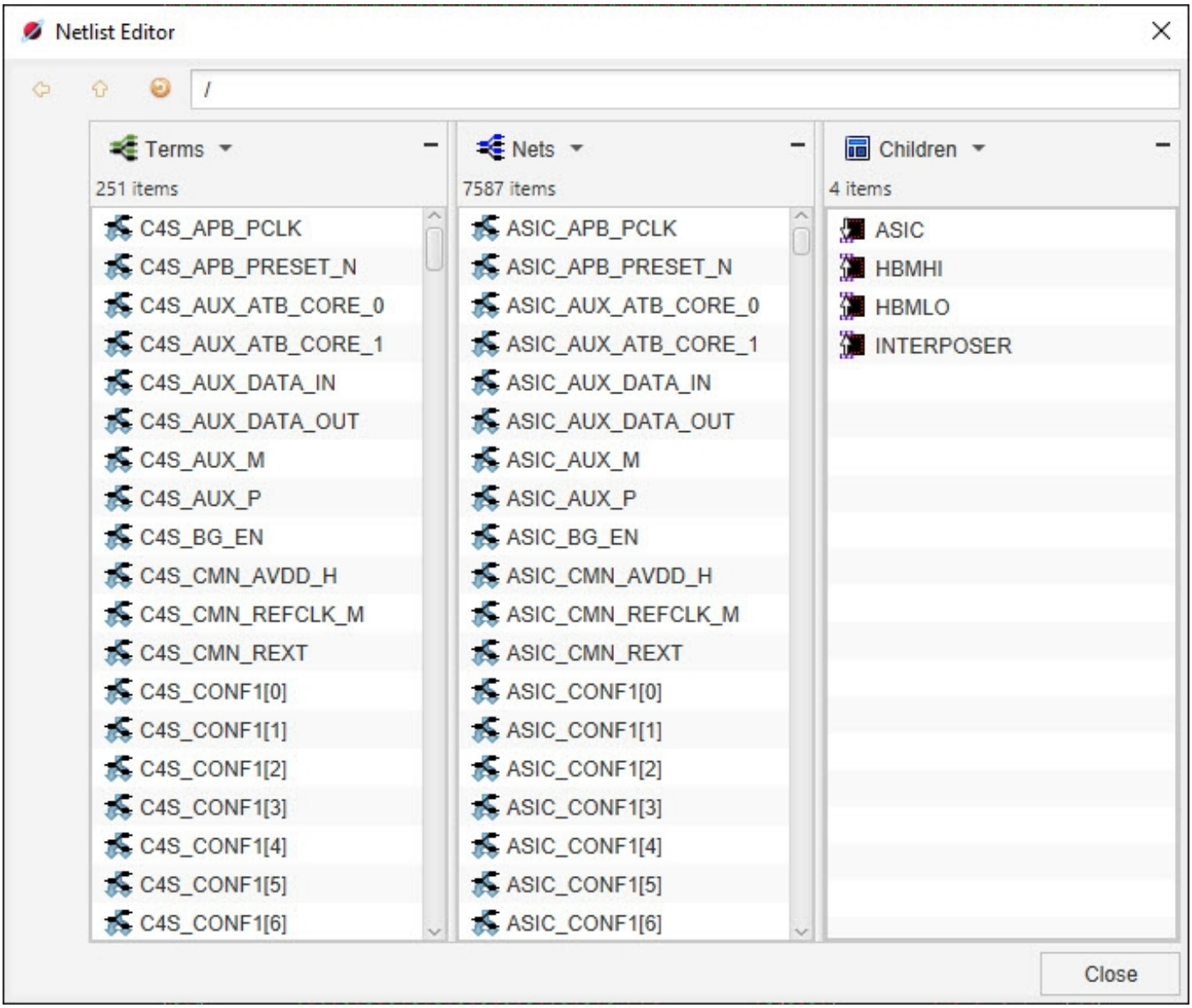
Note that the terms were created with the designated *ASIC_* name prefix.

Exploring Connectivity with the Netlist Editor

Connectivity can be explored using two methods, the *Netlist Editor* and the *Graph Net or Graph Term* commands.

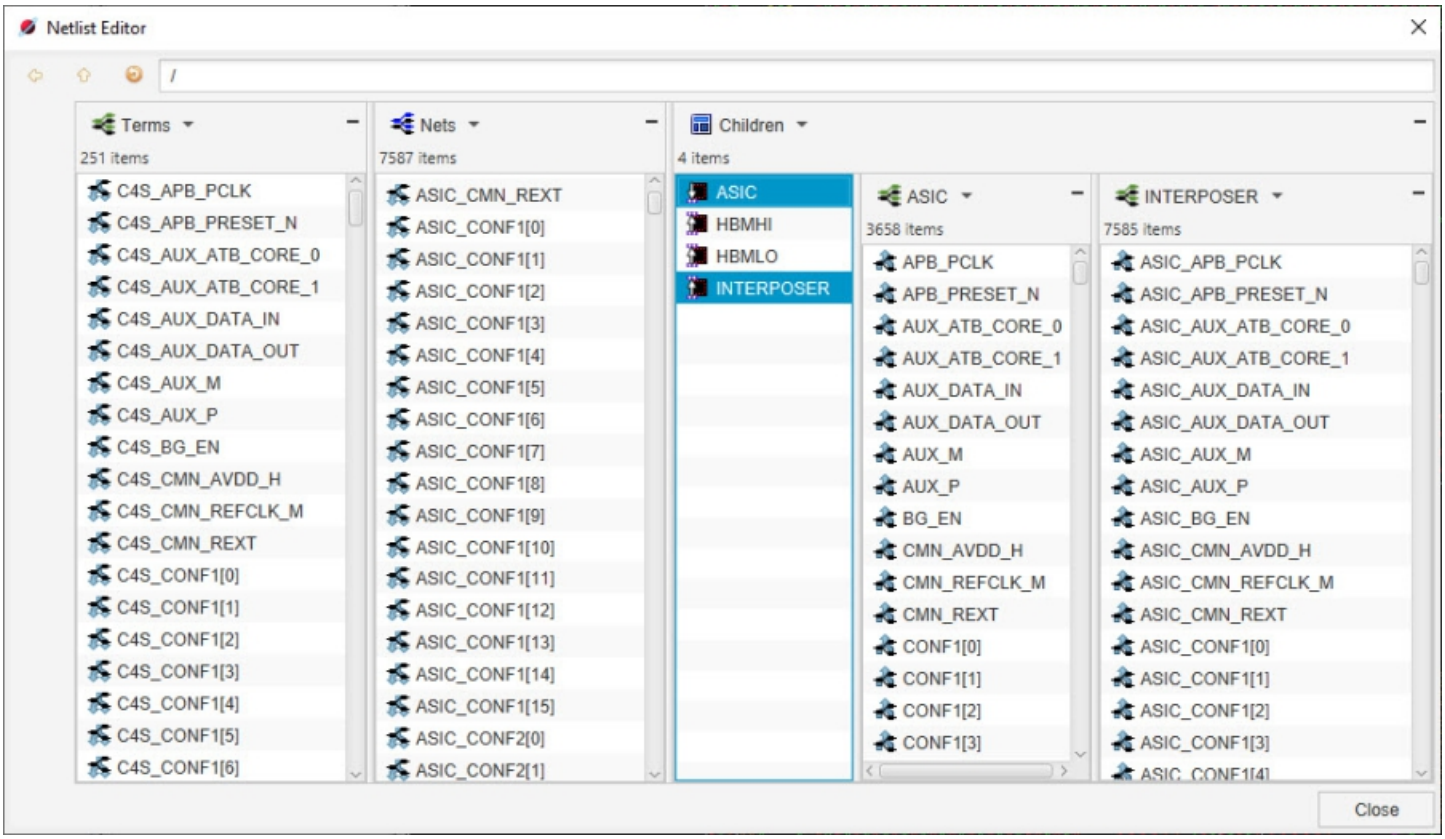
Using the Netlist Editor

Invoke the Netlist Editor at any device or hierarchy level by right-clicking it and selecting the *Netlist* command.



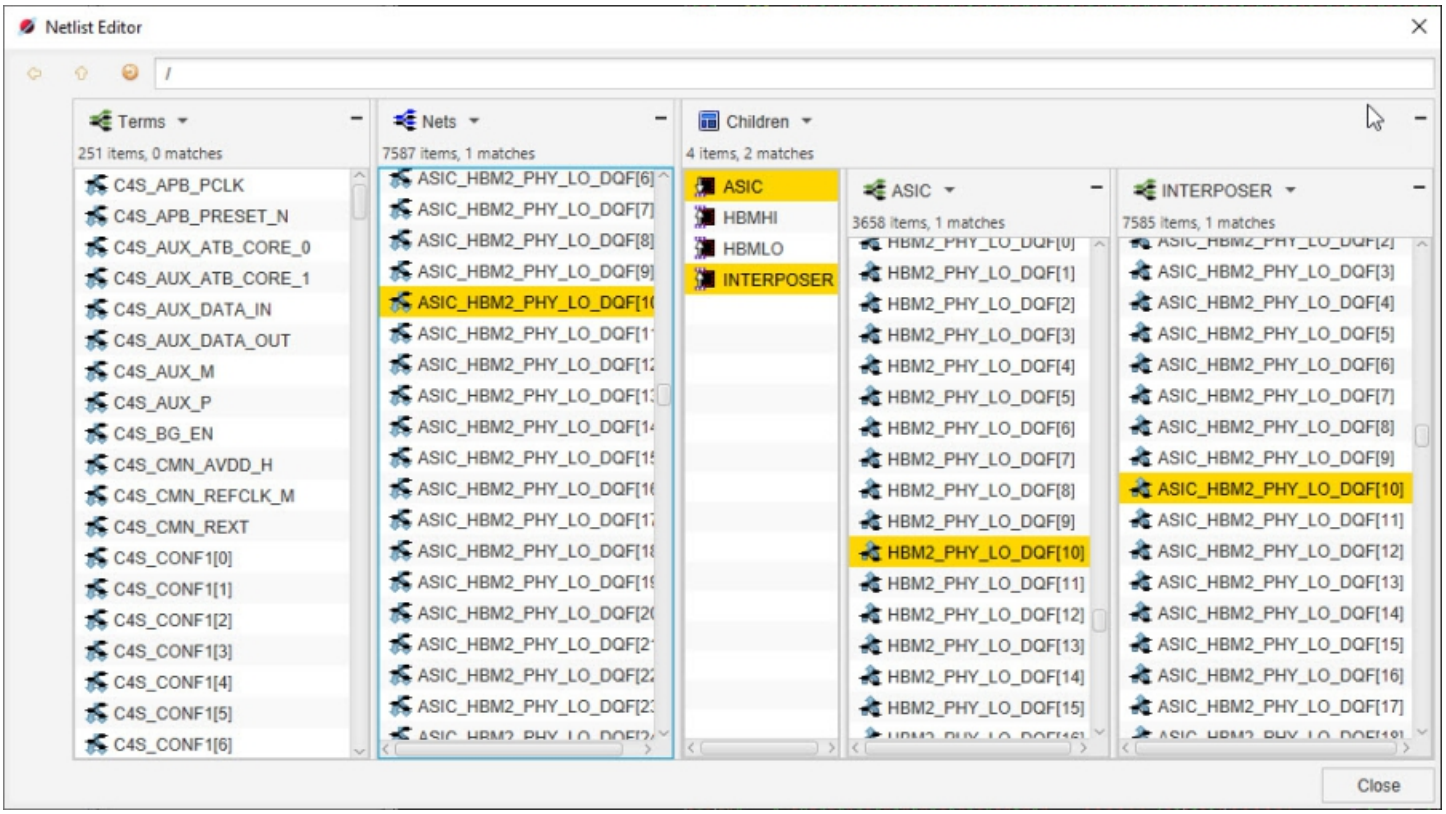
The Netlist Editor displays the *Nets*, *Terms*, and *Children* for the level of device hierarchy displayed in the top field.

Double-click on an object in the *Children* list to display child nets, as shown below.

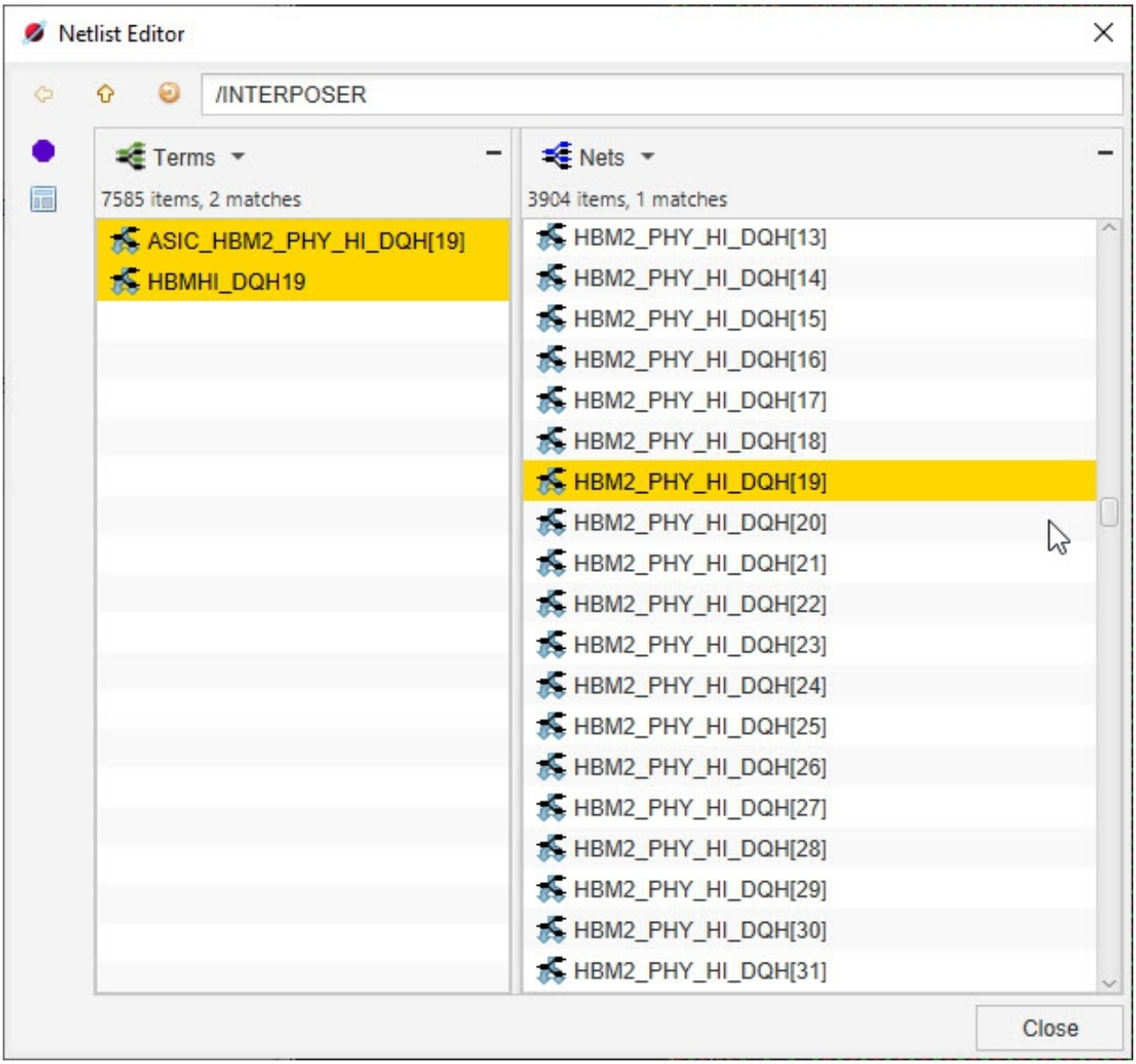


> ⓘ All Netlist Editor features are not included in this guide. Many of the features involve net mapping capabilities used for package design and are not applicable for term-based Verilog netlist mapping.

To explore connectivity, double-click any *Net* or *Term* name in the list columns. The nets that are mapped are centered and highlighted in the list, as shown below.



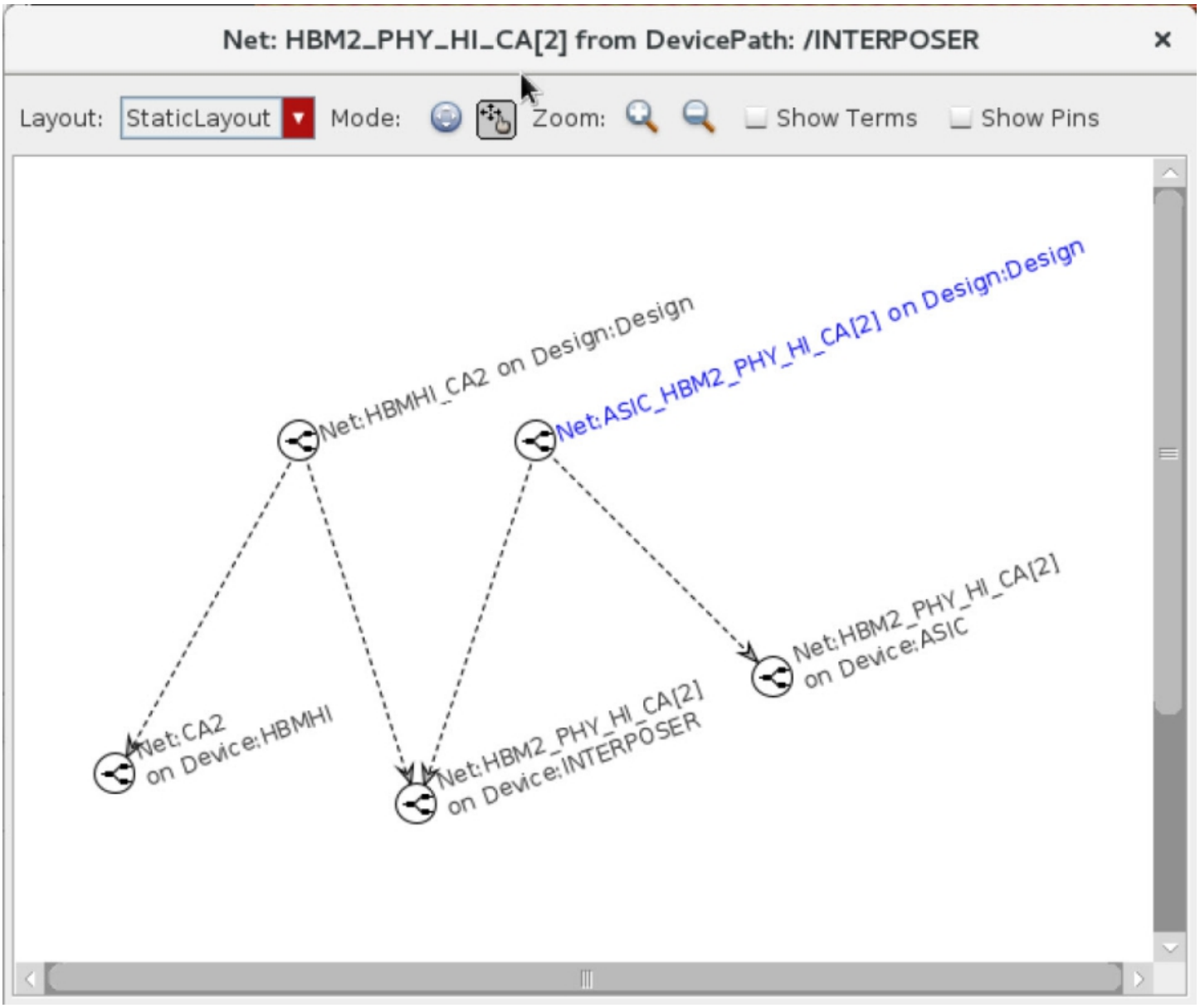
To view *Nets* mapped to multiple terms, select *Filters – By condition – is matched* under the *Terms* pull-down menu. Double-click any *Nets* in the column and notice the mapped *Terms* are displayed and highlighted, as shown below.



This method can be used to explore the net mapping within the design.

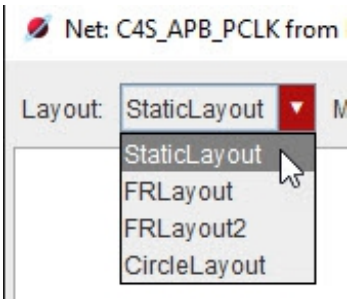
Exploring Connectivity with the Graph Net/Term Commands

A graph of the connectivity for any net can be displayed by right-clicking the net or term and choosing *Graph Net* or *Graph Term*. The Net Graph is invoked, as displayed below.



The Net Graph displays the names and types for all the objects on the net. The various objects on the net can be dragged and moved to make the graph more readable.

The *Layout* pull-down menu provides options for displaying the net, as shown below.



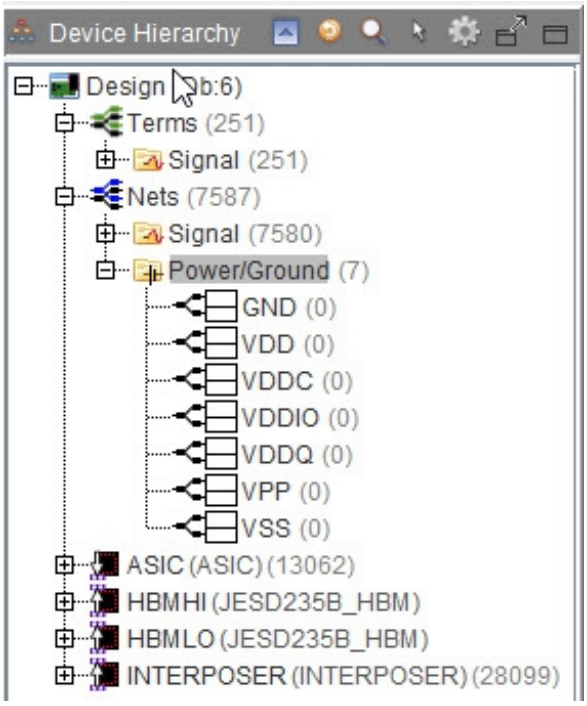
The two *Mode* icons enable selection between *Pan* mode and *Pick and move vertices* mode.

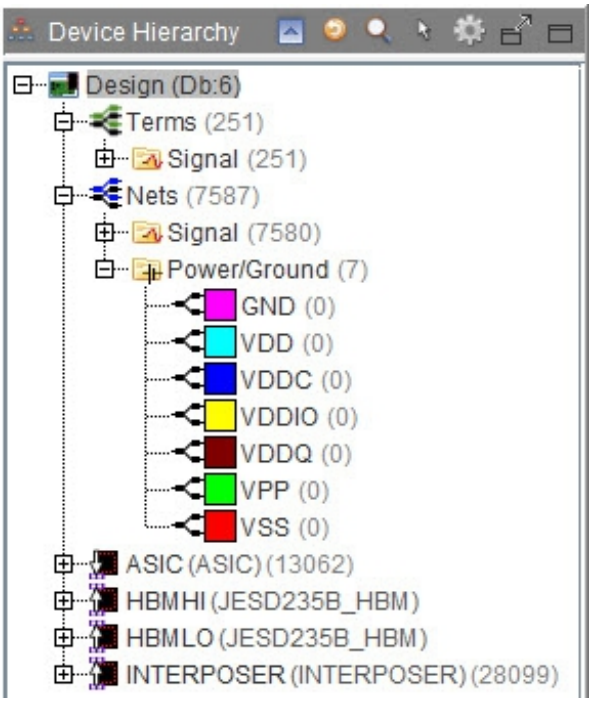
The *Show Terms, Show Pins,* and *Show Vias* options can be used to toggle display of these objects.

Working with Power and Ground Nets

Power and Ground (PG) nets are treated in a different way to signal nets in System Planner. The PG nets are not displayed when connectivity is displayed by selecting *Display Connection* ( ) from the main toolbar. They are filtered out to better show the signal nets. The PG nets are displayed under a separate *Nets – Power/Ground* folder in the Device Hierarchy, as shown below.

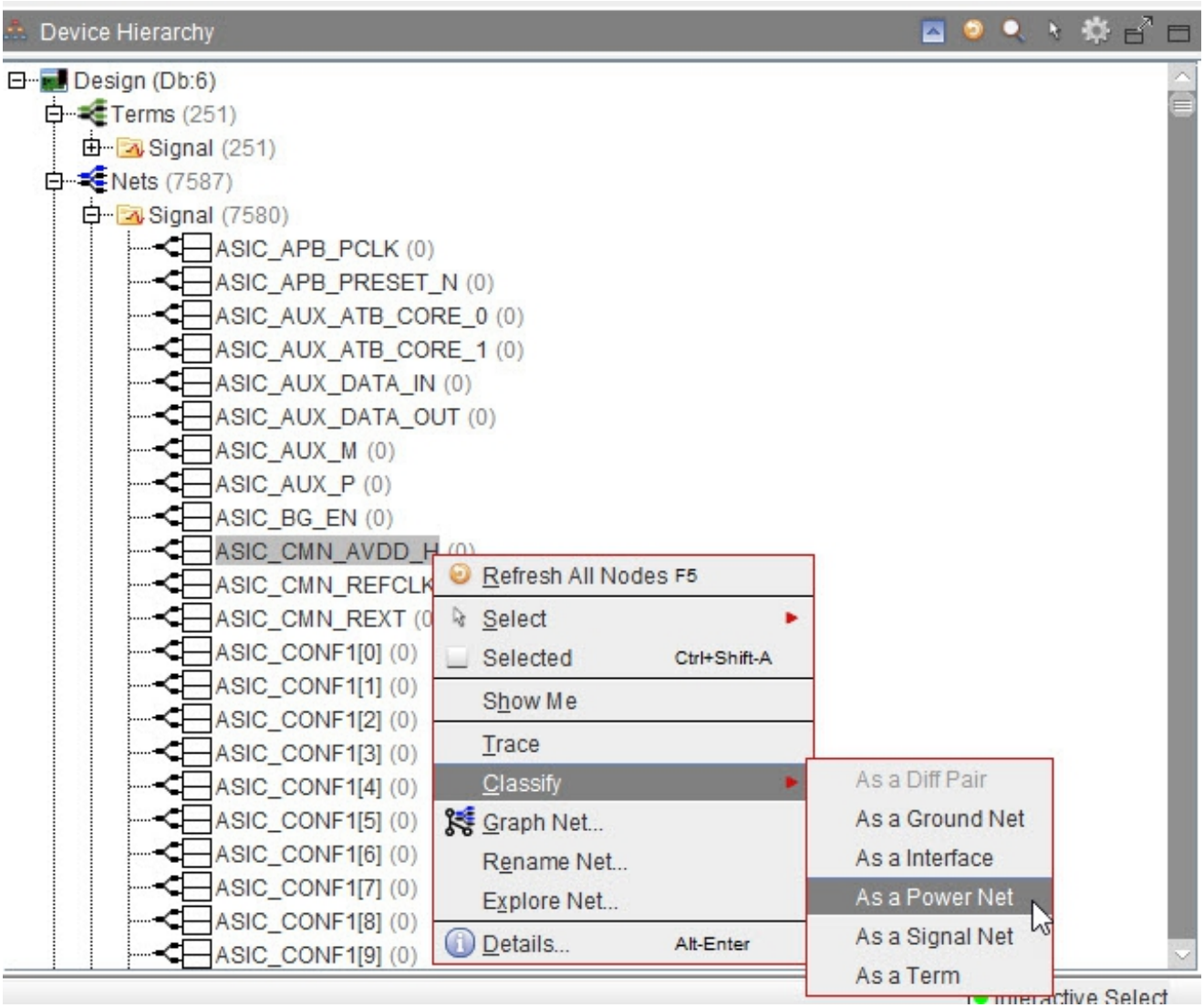
PG nets can be colored by selecting the white box next to the net and selecting a new color from the color palette presented, as shown below on the right.





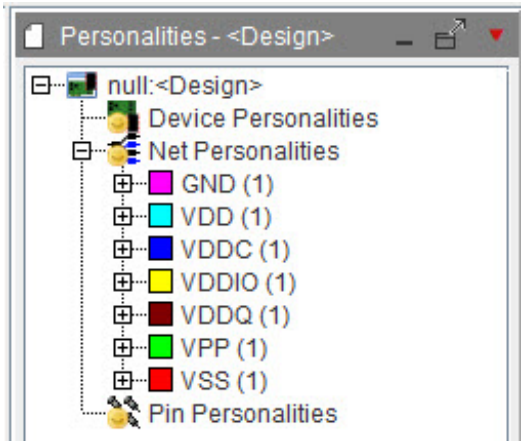
PG nets are identified as Power or Ground during the import of a Verilog netlist. They can also be identified when using the *Import – CSV Pins* command by using the *PIN_USE* column with *POWER* or *GROUND* field entries.

Any net can be interactively defined as a Power or Ground net by right-clicking the net in the Device Hierarchy and choosing *Classify – As a Power/Ground Net*, as shown below.



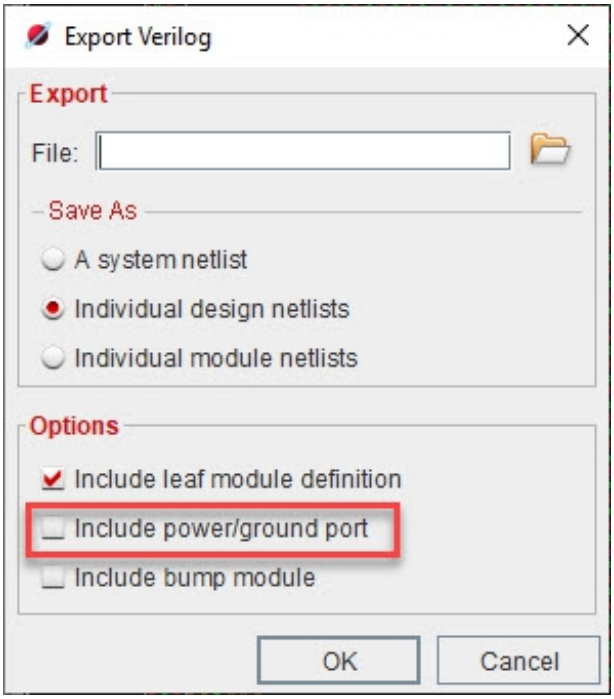
PG nets defined using the *Classify* command will result in the net being displayed under the *Power/Ground* folder. The net connections are then not displayed when using *Display Connection*.

Nets classified as PG nets automatically get a *Net Personality* assigned, allowing them to be colored for easier identification, as shown below. Colors can be changed by selecting the colored box and selecting a new color from the color palette presented.



Nets with over a 100 connections are automatically treated as PG nets, and their net connections are not displayed when using *Display Connection.*

The *Export Verilog* command allows PG net ports to be filtered out using the*Include power/ground port* option, as shown below.

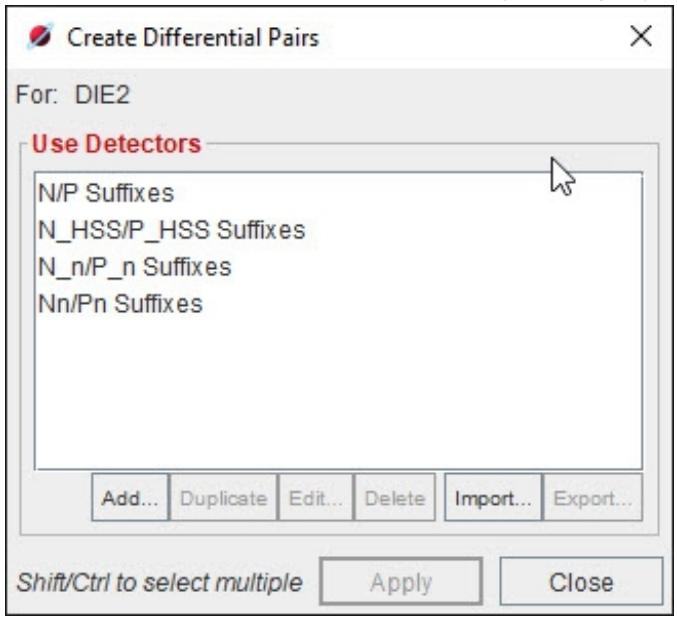


Defining Differential Pairs

Differential pair (diff pair) signals can be identified in two ways in System Planner. First, regular expressions can be used to define net naming strategies that will automatically filter all nets and classify any with matching name patterns as diff pairs. For example, all matching nets with *N* and *_P* suffixes can be defined easily as diff pair groups. Diff pair groups can also be imported for any device by right-clicking it in the Device Hierarchy and choosing *Import – Personalities*.

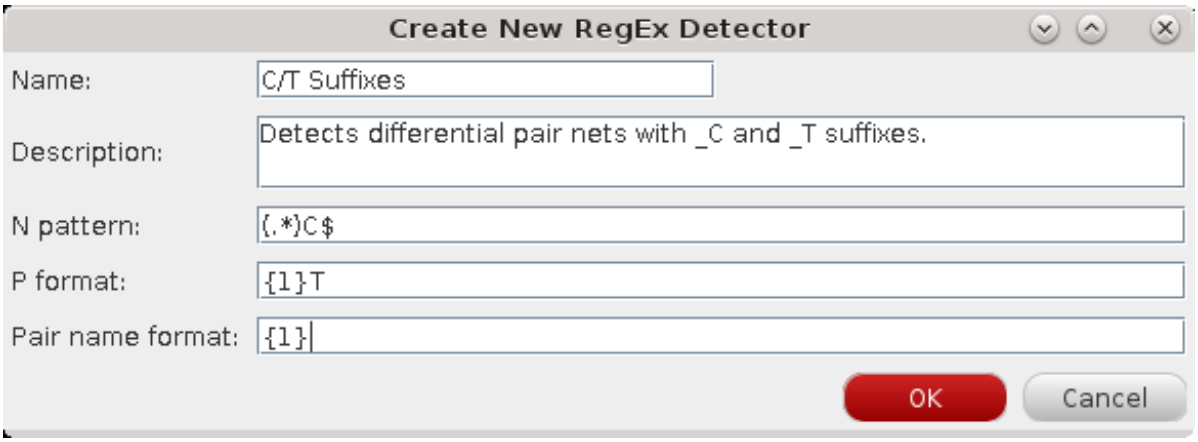
To define diff pair nets based on the net naming strategies,

1. Right-click any level of hierarchy in the Device Hierarchy and choose the*Device – Create Differential Pairs* command.
2. The Create Differential Pairs dialog is displayed, as shown below.



   Four common diff pair naming filters, referred as *Use Detectors*, are supplied by Cadence. The Cadence supplied use detectors cannot be edited.

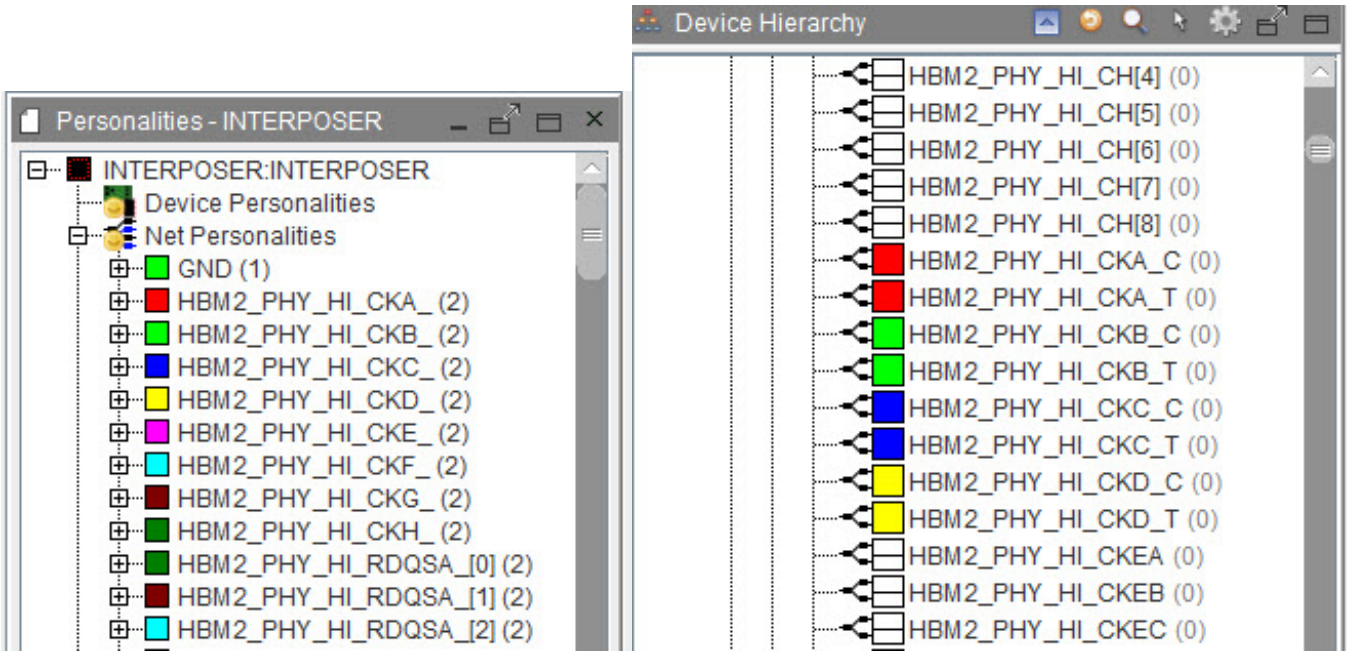
   - New use detectors can be created using the *Add* button and entering the required information. An example is shown below.



This example will create a new use detector recognizing net names with *_C* and *_T* suffixes as diff pairs. The regular expression filters can be used to recognize a wide range of naming strategies for diff pairs.

- You can copy the supplied use detectors by using the *Duplicate* button and edit the copies to see how to define use detector regular expressions.
- Previously created use detectors can be modified by using the *Edit* button. Although Cadence supplied use detectors cannot be edited, you can copy them and then modify the copies.
- Use detectors can be saved and reused in other designs. An `xml` format file is written with the *Export* command, and the use detectors can be used again by using the *Import* command.

3. Select the required use detectors from the list and click the *Apply* button. This will scan the selected device for nets matching the selected use detectors and classify those nets as diff pairs. The net name minus the use detector characters are used as the diff pair name. For example, the *C* and *T* are removed to form the diff pair net group name, as shown below. Diff pair signals receive a *Net Personality* and are also displayed differently in the *Device Hierarchy*, as shown below.



Diff pair net groups can also be imported in a net personality CSV file. Right-click the desired device in the Device Hierarchy and choose *Import – CSV Personalities* to classify the defined net groups as diff pairs automatically.

Refer to the Defining Net Personality Groups section for more information.

Defining Net Personality Groups

System Planner uses the term Personality to define a group of nets, pins, or devices so as to better identify them. As an example, power and ground nets receive a net personality to enable their coloring for easy identification.

Personalities other than power and ground are rarely used in a typical 3D-IC design. Therefore, all personality features are not described in this guide. Net personalities can be used to identify key interfaces for floorplanning and visualization. Nets assigned to net personality groups will display the connectivity rats in the color specified for the net personality.

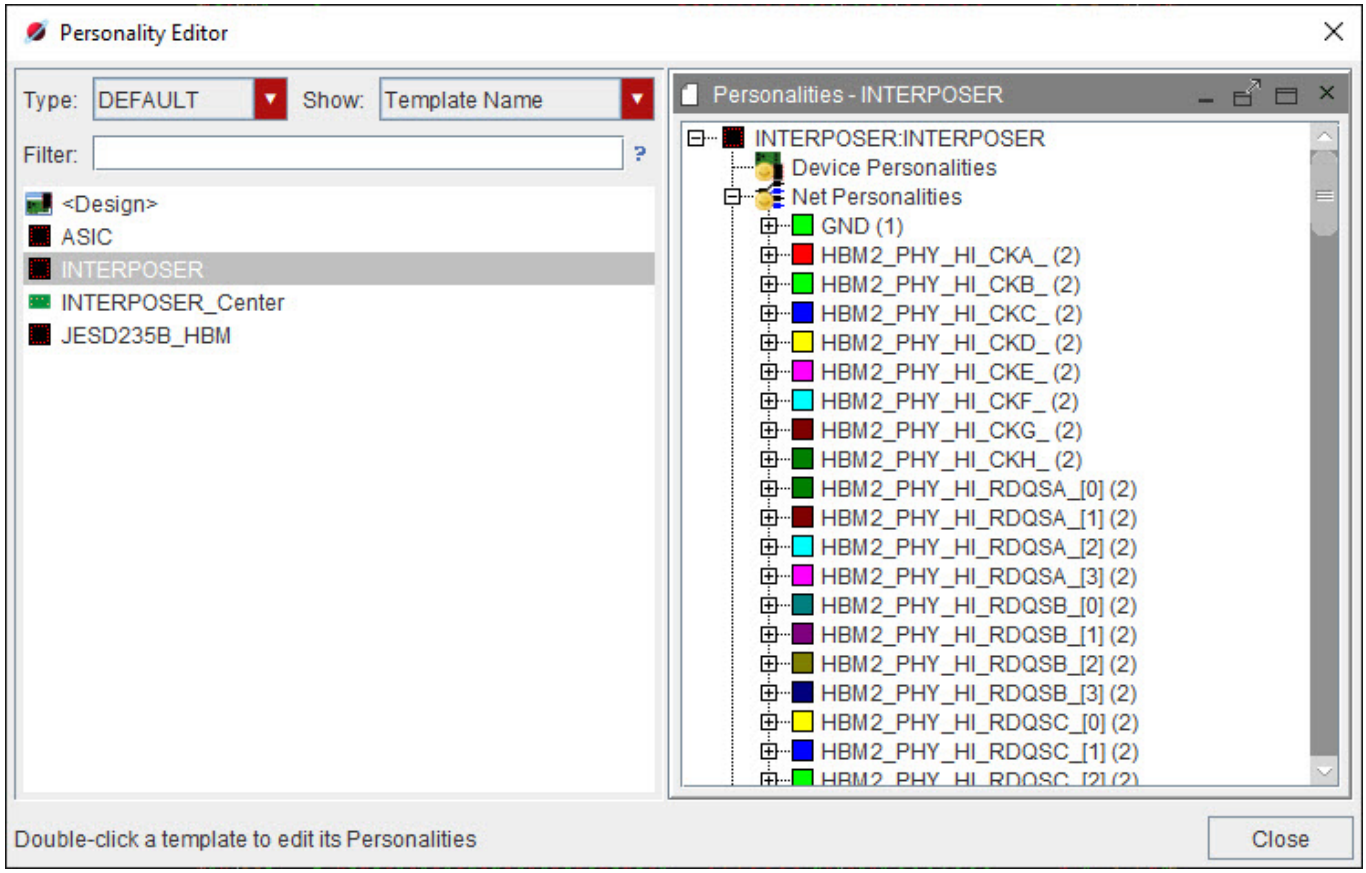
Differential pair signals are also identified as net personality groups. Several net personality features are described in this guide. Refer to the Defining Differential Pairs section for more information on defining differential pair net personalities.

Interactively Creating Personalities

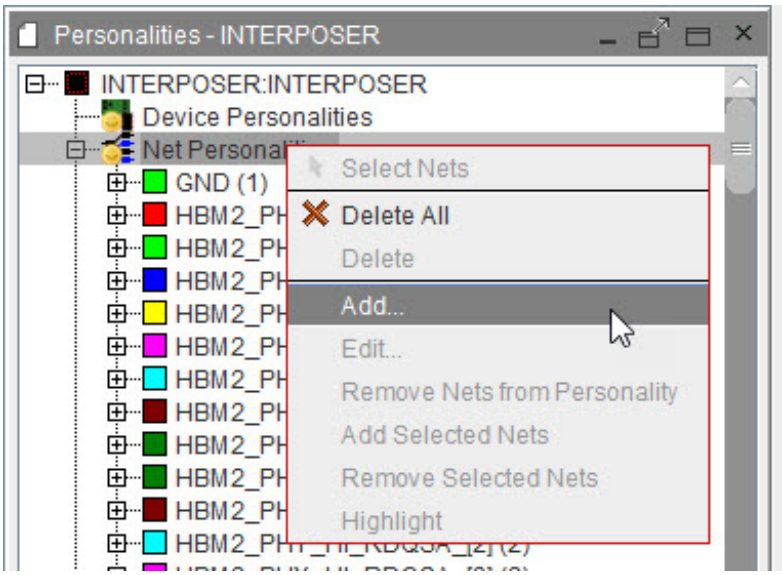
Personalities can be created interactively and assigned to design objects or by importing a CSV personalities file.

To create a personality interactively:

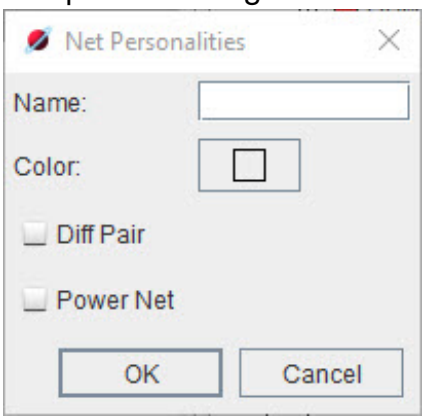
1. Select *Edit – Personalities* from the menu bar to invoke the Personalities Editor, as shown below.



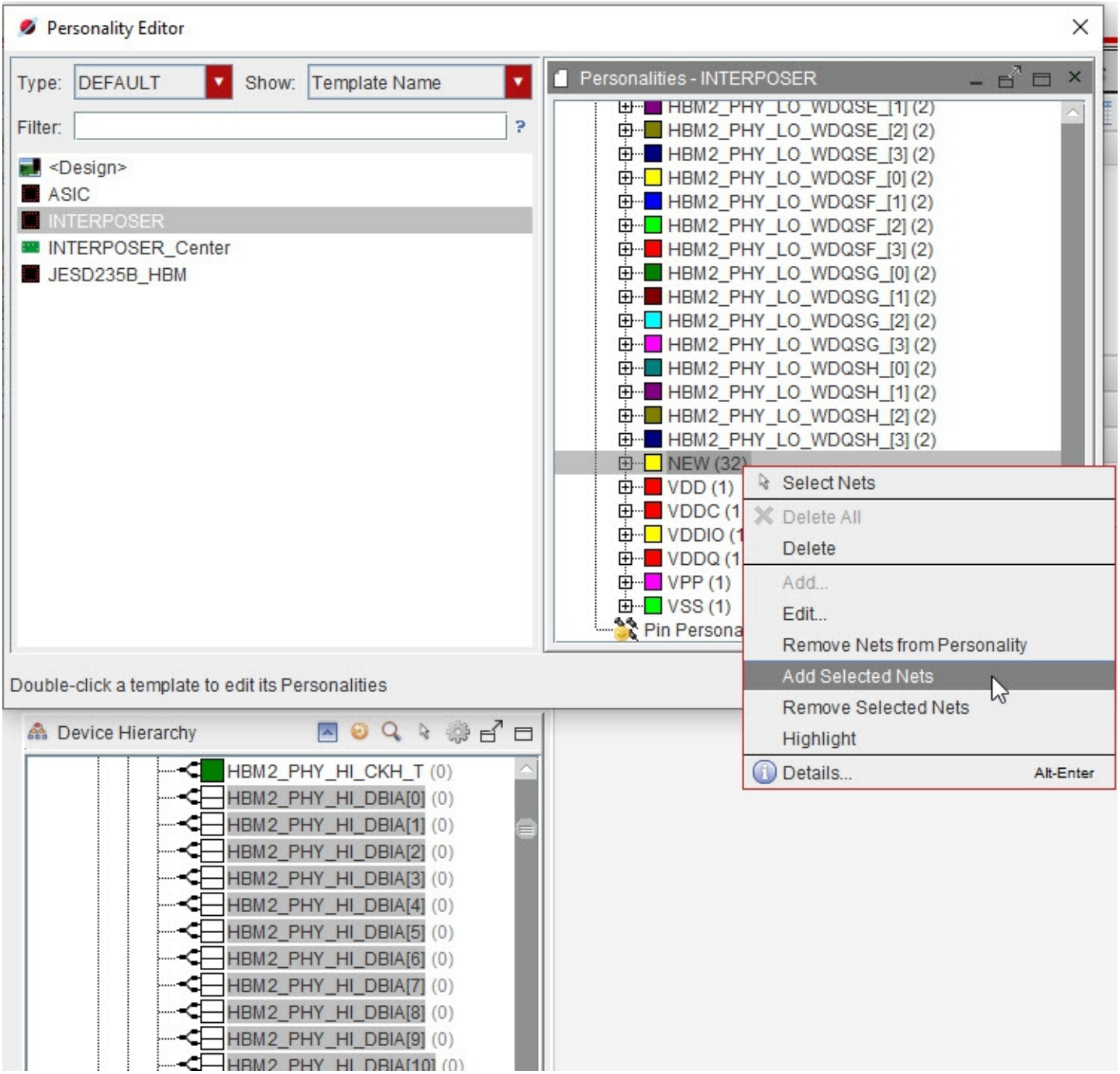
2. Double-click a device in the list on the left pane to populate its existing device personalities on the right. Nets classified as power or ground, as well as differential pairs, are automatically assigned a net personality group and are displayed, as shown above.
3. To create a new personality group, right-click the*Net Personalities* folder and select *Add*, as shown below.



4. The *Net Personalities* dialog is presented. Specify the relevant details and click OK to complete creating the new net personality group.



To assign nets to a personality group, first select the nets in the Device Hierarchy. Next, right-click the nets and choose *Selected Nets*. Then right-click the personality group in the Personality Editor and right-click to choose *Add Selected Nets*, as shown below.

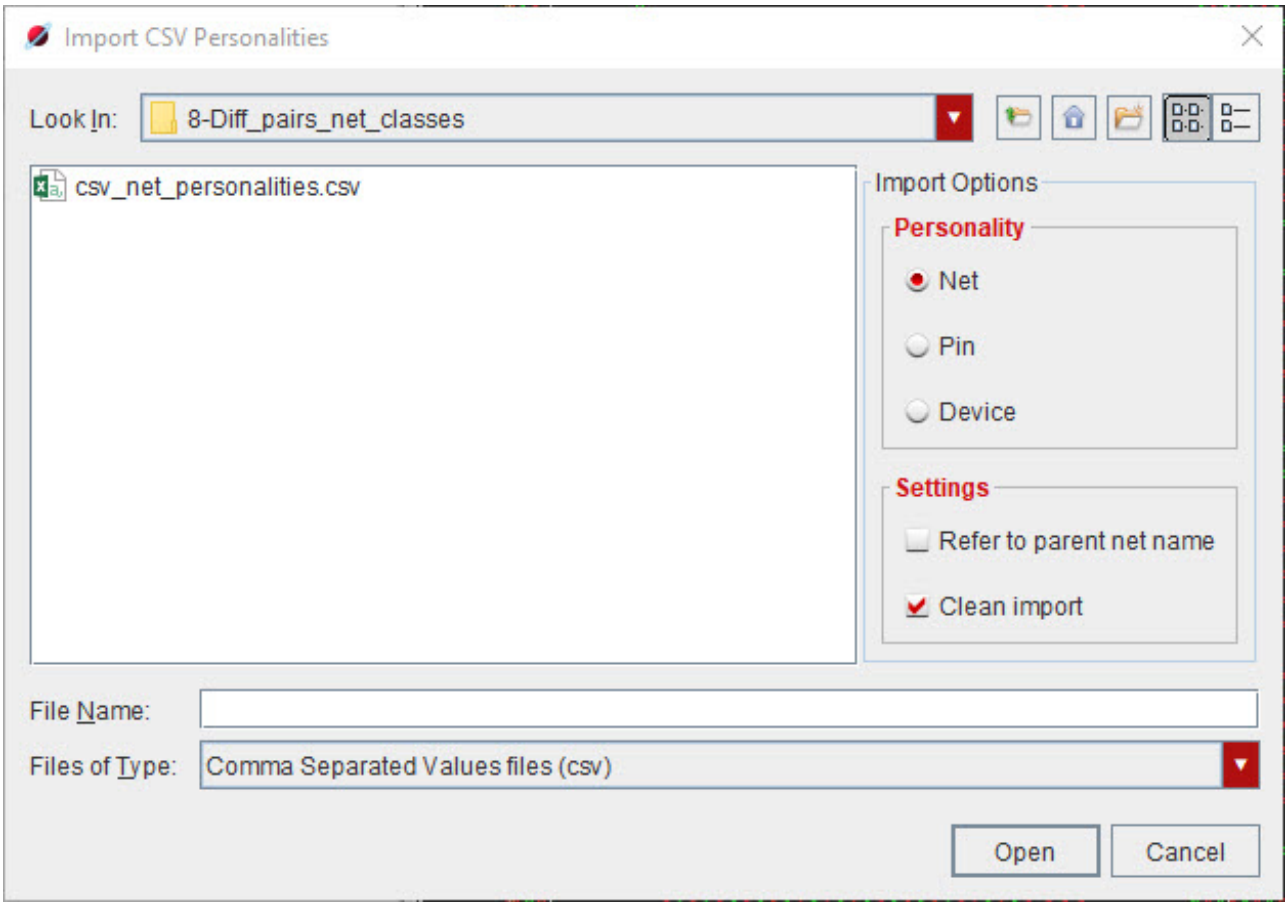


Importing CSV Personalities

Personality groups can be imported in a CSV personalities file.

To import personality groups,

1. Right-click the device in the Device Hierarchy and choose *Import – CSV Personalities*. The Import CSV Personalities dialog is presented, as shown below.



2. Browse and select the CSV personalities file in *Look in*.
3. In the *Personality* section, select the type of personality to import.
4. Confirm or modify the settings in the *Settings* section:
   - *Refer to parent net name*– Uses the parent net name to identify the net names in the design.
   - *Clean import* – Removes all current personalities defined in the design and replaces them with the imported groups. Deselecting this check box will merge the imported personalities with those currently defined in the design.
5. Click *Open* to import and define the personality groups to the objects in the design.

An example of a CSV personality file is shown below.

	A	B	C	D	E	F	G	H
1	Net	Group	Power Net	Diff Pair	Voltage	Color	Power Net Color	Diff Pair Color
2	ASIC_VDDIO		VDDIO		1.2		YELLOW	
3	ASIC_VSS		VSS		0		GREEN	
4	GND		GND		0		GREEN	
5	HBM2_PHY_HI_CKA_C			HBM2_PHY_HI_CKA_				RED
6	HBM2_PHY_HI_CKA_T			HBM2_PHY_HI_CKA_				
7	HBM2_PHY_HI_CKB_C			HBM2_PHY_HI_CKB_				GREEN
8	HBM2_PHY_HI_CKB_T			HBM2_PHY_HI_CKB_				
9	HBM2_PHY_HI_CKC_C			HBM2_PHY_HI_CKC_				BLUE
10	HBM2_PHY_HI_CKC_T			HBM2_PHY_HI_CKC_				

Creating Bumps and TSVs

System Planner has several capabilities to create bumps and bump patterns for use with bottom layer signal pads, power and ground patterns, and TSVs.

- Create bumps on any layer by selecting a group of pins and using the *Bumps for All Layer Shapes* command.
- Create patterns of pins for power and ground matrixes, signals, and TSVs by using the *Create Pin or Device Pattern* command.
- Generate TSVs for a selected set of pins by using the *Fanout Vias* command.
- Auto-assign existing bumps with nets to optimize routing by reducing net crossings by using the *Connection Optimization* command.

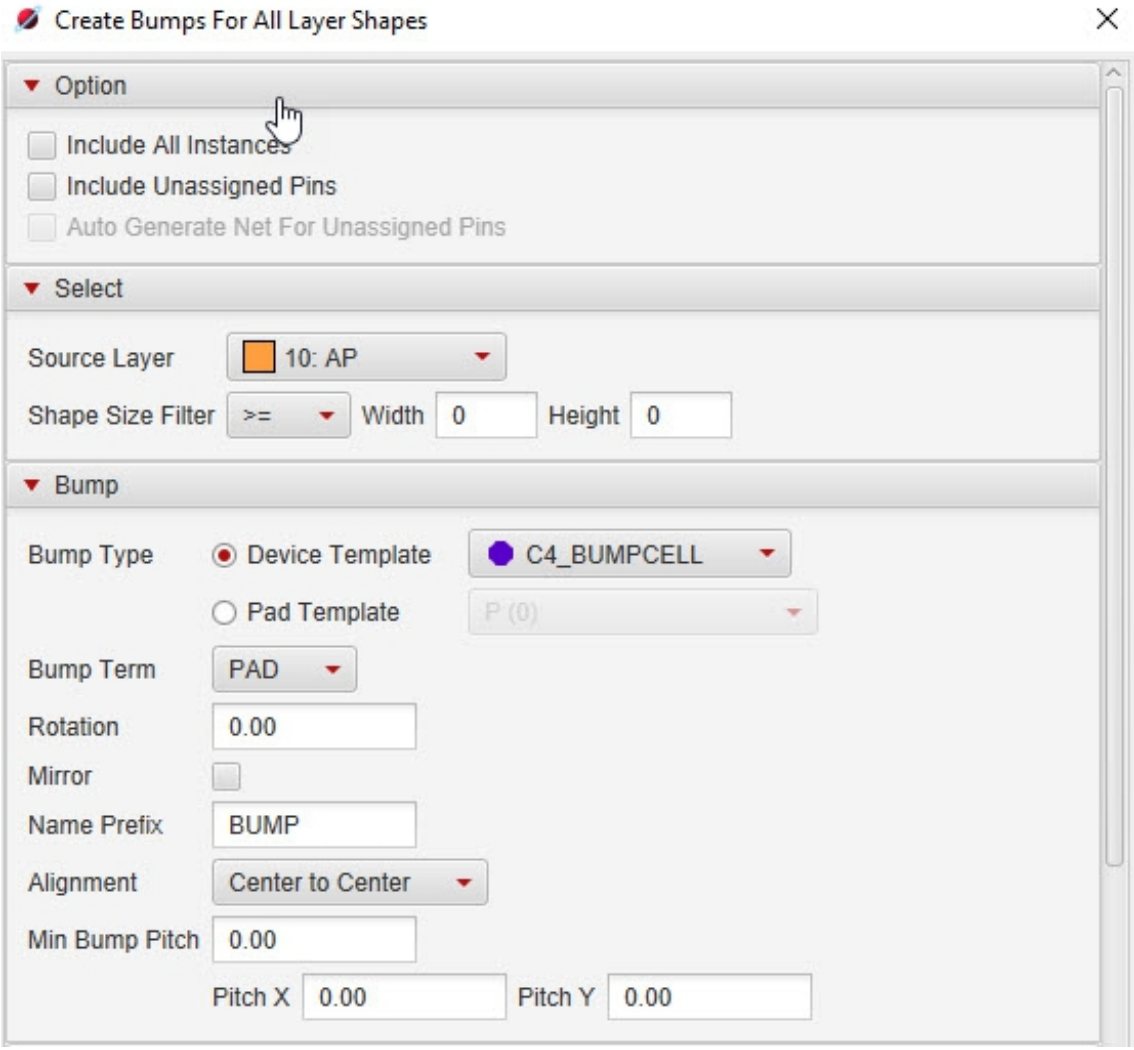
These commands are covered in detail in the following sections.

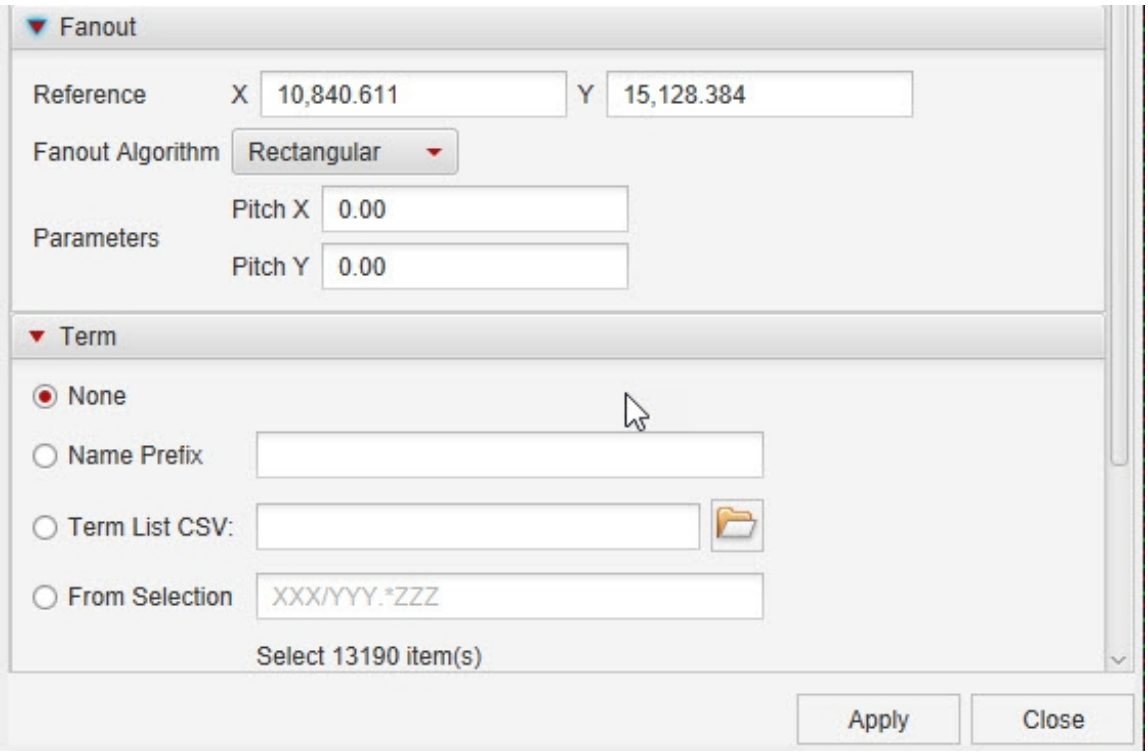
Using the Bumps for All Layer Shapes Command

Bumps can be automatically created on any layer for selected pins. You must create a padstack for the bump on the target layer before using the *Bumps for All Layer Shap*es command. These bumps are typically vertically aligned with the selected pins but can be fanned out as well.

1. Select a group of pins to generate bumps using any selection method. Adjusting visible layers and using the *Selection Filte*r is often useful for this purpose to ensure only the source layer pins are selected.
2. Once the desired pins are selected, right-click on the canvas to choose the *Bumps for All Layer Shap*es command. The *Create Bumps For All Layer Shapes* dialog is presented, as

shown below.


Create Bumps For All Layer Shapes
Option
Include All Instances
Include Unassigned Pins
Auto Generate Net For Unassigned Pins
Select
Source Layer
10: AP
Shape Size Filter
>=
Width
0
Height
0
Bump
Bump Type
Device Template
C4_BUMPCELL
Pad Template
P (0)
Bump Term
PAD
Rotation
0.00
Mirror
Name Prefix
BUMP
Alignment
Center to Center
Min Bump Pitch
0.00
Pitch X
0.00
Pitch Y
0.00

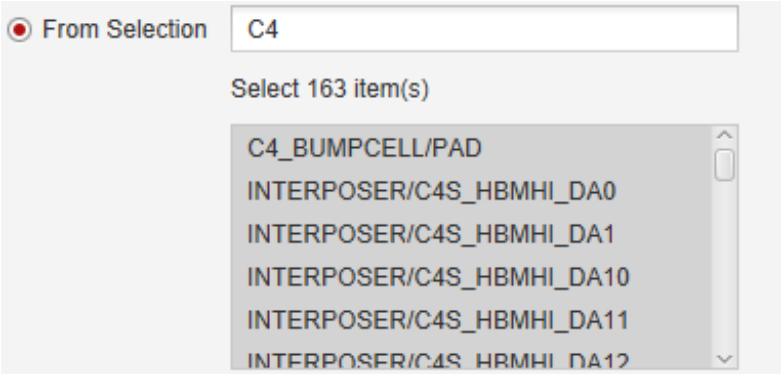


3. Most of the dialog options are optional and are explained below. Click*Apply* to generate the bump pads based on the specified options.

Create Bumps for All Layer Shapes Options

Include All Instances	Creates bumps for all pins with the same net.
Include Unassigned Pins	Creates bumps for unconnected pins.
Source Layer	Identifies the substrate layer for the selected source pins.
Shape Size Filter	Filters the shapes on the selected source layer to use for creating bumps.
Bump Type – Device Template	Identifies the padstack to be used for the bumps for IC design.

Bump Type – Pad Template	Identifies the padstack to be used for the bumps for package design.
Bump Term	Identifies the term of bump device template to be used for connecting the source pin. In most cases, a bump template only has one term. However, some special testcases have multiple terms in a bump template.
Rotation	Defines the rotation value for the bump pad.
Mirror	Mirrors bumps by Y-axis.
Name Prefix	Enables a name prefix to be entered for the bump pad name.
Alignment	Defines the alignment between the selected pins and the bump pads.
Min Bump Pitch	Defines the minimum spacing between generated bump pads.
Pitch X Pitch Y	Defines a specific X/Y grid pitch to generate the bump pads.
Fanout	
Reference	Identifies the starting anchor point to be used for placing bumps.
Algorithm	Defines the direction for the fanout routes.
Parameters	Defines a specific X/Y grid pitch to generate the fanouts.
Term	
None	Does not generate terms for the bump pads.
Name Prefix	Defines a name prefix for the bump pad terms.
Term List CSV	Defines a term list file to be used for the bump pads.

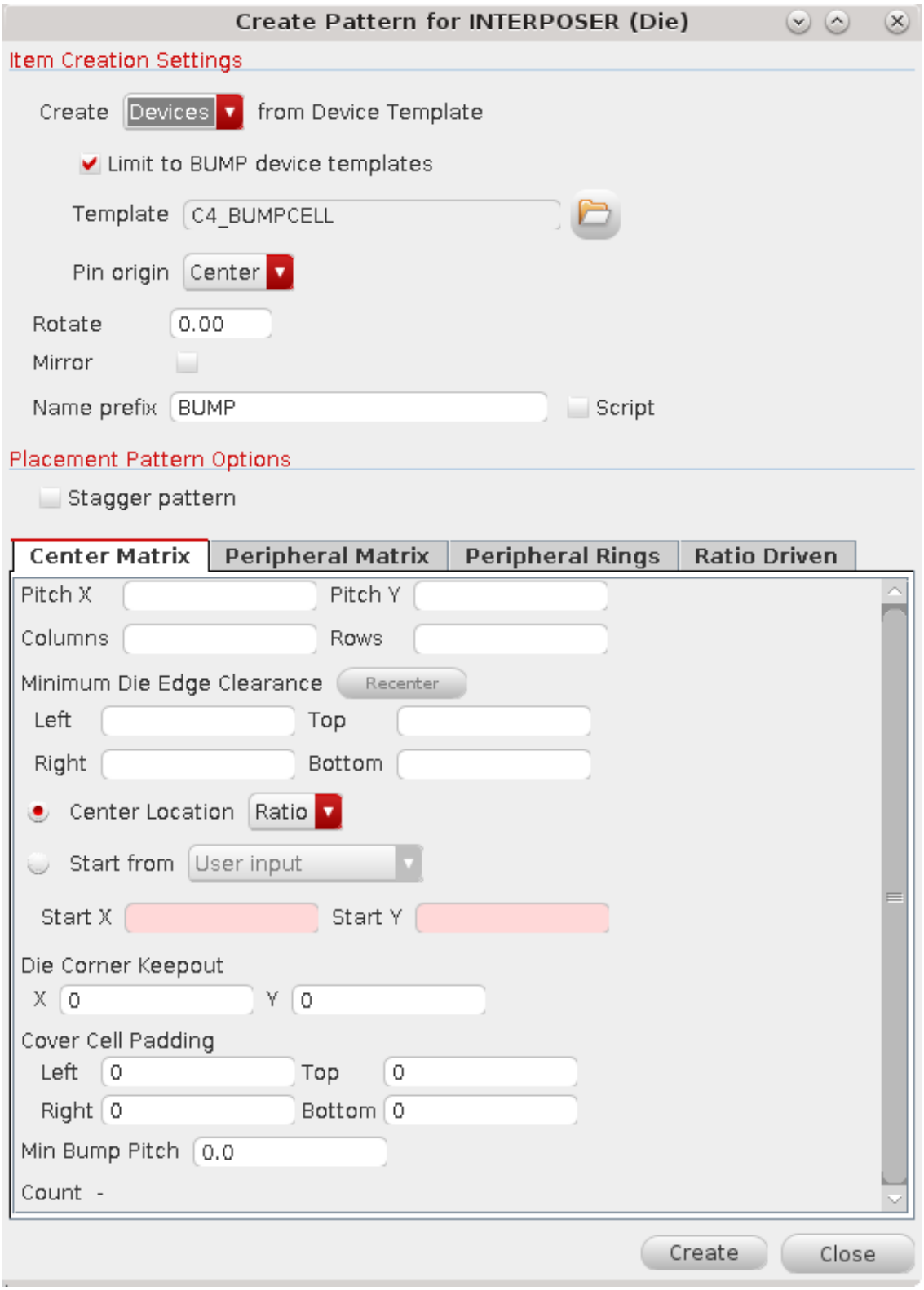
From Selection	Selects the existing terms by a keyword for the bump pads. 

Creating a Pin Pattern for Bumps and TSVs

Patterns of pins can be created to be used for a variety of reasons. The pattern itself can be created in several styles. Pattern styles ranging from simple matrixes to complex shapes that automatically avoid placed die devices can be generated.

To generate a pin pattern:

1. Right-click the device in which pins are to be created in the Device Hierarchy and choose the *Create Pin or Device Pattern* command. The Create Pattern for <selected device> dialog is presented, as shown below.



In this dialog, you can specify various options, including the placement pattern, for generating the pin pattern. The options in the dialog are explained here.

2. Click the *Create <placement pattern option chosen> Matrix* button to generate the pin pattern based on the specified options.
3. Click *Close* to close the dialog.

The pin pattern groups are created under the selected device in the *Device Hierarchy* in a folder called *COVERs*. For more details on *COVERs*, refer to Working with the COVERs Pin Groups.

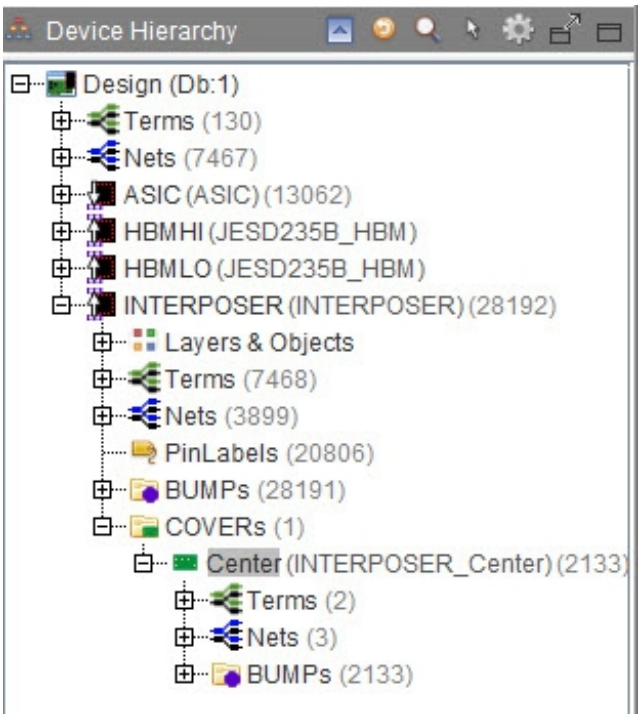
Create Pattern for <selected device> Options

Item Creation Settings	
Create	Presents the options to generate *Device* style pins for IC design or *Pins* for package design.
Limit to BUMP device template	Restricts the choice to only BUMP type templates.
Template	Provides template options from existing pads.
Pin origin	Specifies the pin origin.
Rotate	Rotates the pads by the specified value as they are generated.
Mirror	Mirrors pads by Y-axis.
Name prefix	Appends the specified name prefix to the bump pad name. The default prefix is *Bump*. Blank out this field if you do not want the default prefix.
Script	Provides the option to generate the pins' names by a script.
Placement Pattern Options	
Stagger pattern	Generate the pins in a staggered fashion, if selected.

Center Matrix	Generate a uniform rectangular pin matrix starting at the center of the device. The group of pins can be moved after they are generated. Many of the options are self-explanatory. The *Minimum Die Edge Clearance* fields define the spacing for voids created around exiting placed die, which is often used for power and ground cover patterns. Entering 0 will ignore placed die and create the pattern under the die. The *Count* at the bottom shows the number of pins to be generated based on the options entered.
Peripheral Matrix	Generates a matrix ring pattern of pads around the periphery of the device. The size of the pattern depends on options entered with pads created from the outside die edges toward the center of the device. Many of the options are self-explanatory. The *Minimum Die Edge Clearance* fields define the gap between the outside edge of the selected die to the first ring of pads created. The values need to be negative values. For example, entries of -50 will create a 50-micron space between the outside edge of the die and the first ring of pads.
Peripheral Rings	Creates pin rings with varying rotation options for each side of the die. For example, rectangular pads can have different rotation values specified for each side of the die. This is primarily used for wire bonds or package pins and is not recommended for uniform pad geometries.
Ratio Driven	This option is not applicable for 3D-IC.

Working with the COVERs Pin Groups

Each pin pattern group created using the *Create Pin or Device Pattern* command is listed under the *COVERs* folder in the Device Hierarchy, as shown below with the *Center* pattern group. This example shows a power and ground matrix, which explains why only three *Nets* are shown under it (VSS, VDD, NC).



The pins in a pin pattern group can be manipulated as a group in the Device Hierarchy. You can select a group and move, copy, or rotate it as required.

To select all the pins in the group, right-click the group and choose the*Select – All Pins* command. Selecting all pins is often required to feed other commands, such as the *Assign Power and GroundNets* command.

Pin patterns are created with no connect (NC) pins. Signals can be assigned to the pads using a variety of methods, which are further described in the following sections:

- Power and Ground assignment patterns can be assigned using the*Assign Power and GroundNets* command.
- Individual nets can be assigned to any selected set of pins by using the*Set Selected Pins' Nets* command in the right-click or shortcut menu.
- Nets can be automatically assigned in a route-optimized manner by using the*Connectivity Optimization* command.

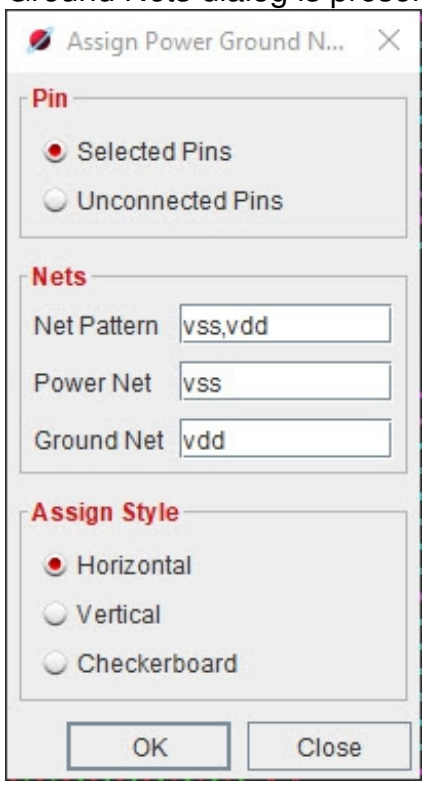
Pin groups listed under the COVERs folder are also located in a COVERs level of hierarchy in the netlist. This is often undesired from a logic netlist perspective. These pin groups can be dissolved and merged into the parent-level netlist. This is recommended after the pins are placed and assigned to nets. To dissolve a pin cover group, right-click the group and choose the *Ungroup* command. Once done, the *COVERs* group will disappear and the pins will be displayed in the parent-level netlist and the *Pins* folder.

Assigning Power and Ground Nets

Power and ground (PG) nets can be assigned automatically to a selected group of pins. Options are provided for net assignment patterns, including allowing NC signal pins, to be left within the pattern.

The *Assign Power Ground Nets* command can assign PG nets on a selected set of pins or to the remaining NC pins in the design using the following steps:

1. Select the desired pins using any selection method, including the *Selection Filter*. To select all the pins in an existing *COVERs* pin group, right-click the group and choose choose *Select – All Pins*.
2. Next, right-click and choose the *Assign Power Ground Nets* command. The *Assign Power Ground Nets* dialog is presented, as shown below.



The dialog options are as follows:

Selected Pins	Assigns PG nets to the selected set of pins.
Unconnected Pins	Assigns PG nets to the remaining NC pins in the design.

Net Pattern	Defines the pattern of net names to auto-assign to the pins. If net names are entered that do not exist in the design, new nets will be created and added to the device's *Nets > Power/Ground* folder. If NC signal pins are desired in the pattern, enter a net pattern as shown in the following example: *VSS, NetUnused, NetUnused, VDD, VSS, …* The *NetUnused* net name will leave as many NC pins as defined in the assignment matrix.
Power Net	Designates any new design net name entered as a *Power* type net.
Ground Net	Designates any new design net name entered as a *Ground-type* net.
Horizontal	Assigns the name names in a horizontal style.
Vertical	Assigns the name names in a vertical style.
Checkerboard	Assigns the name names in a staggered checkerboard style.

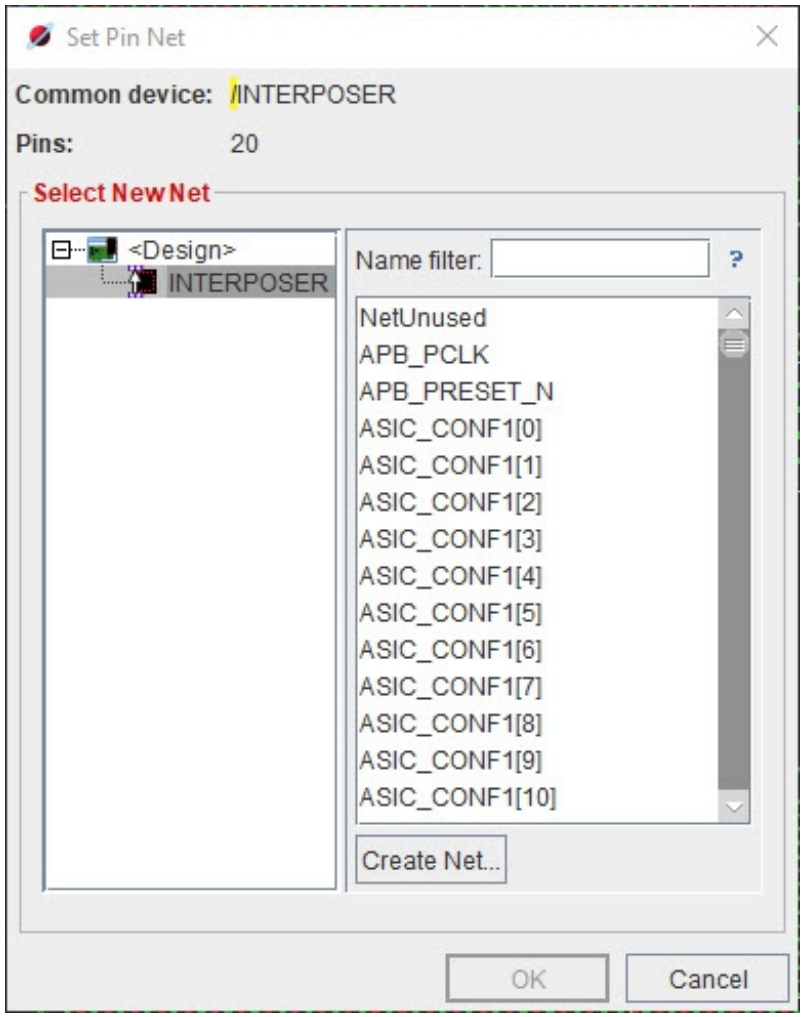
3. Click *OK* to assign the nets based on thespecific options.

Assigning Nets to Bumps

Any net name can be assigned to individual pins or to a selected group of pins.

To assign net names:

1. Select a pin or group of pins using any selection method, and right-click to choose the *Set Selected Pins' Nets* command. The *Set Pin Net* dialog is presented, as shown below.



2. To assign net names, ensure the proper design level is selected in the hierarchy tree on the left. Select the required parent level in the left pane to display the nets contained in that level of netlist hierarchy in the right pane.
3. If required, filter the list of nets displayed by entering the filter pattern in the *Name Filter* field. You can also add new net names to the design by clicking the *Create Net* button and entering a new net name in the dialog presented.
4. Select the desired net name from the list of nets in the right pane.

> Pins can also be unassigned by selecting the *NetUnused* net in the list in the right pane. The pins are stripped of existing net name assignments and receive the *NetUnused* net name, which indicates that it is unconnected.

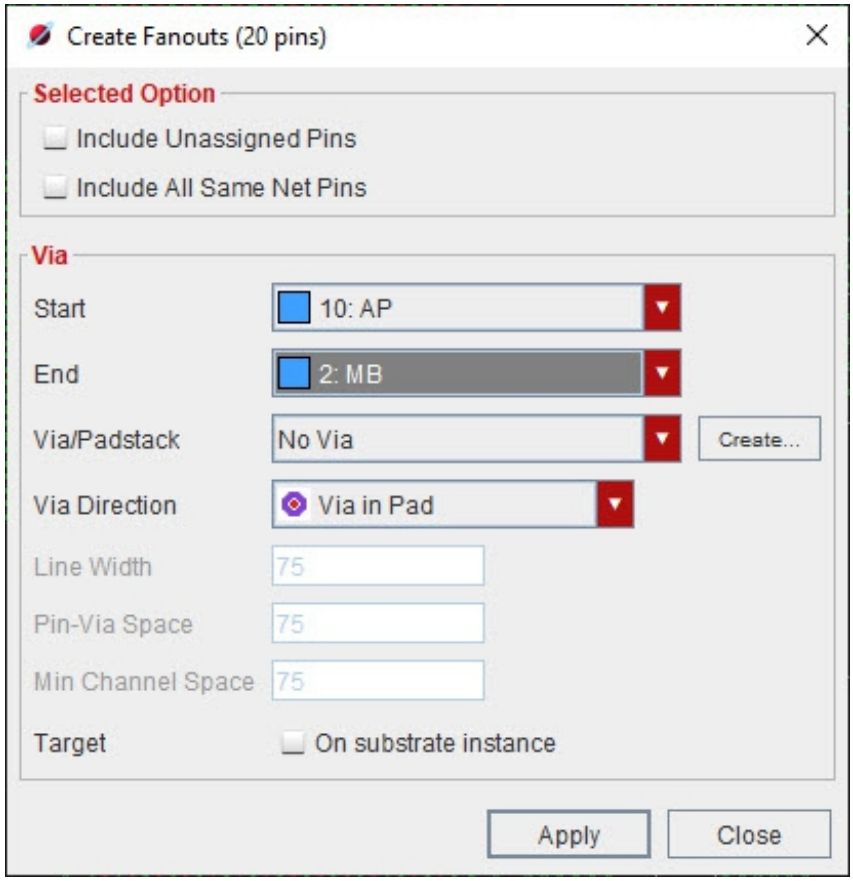
5. Click *OK* to assign the selected net name to the pins.

Creating TSVs Using the Fanout Vias Command

TSVs can be generated automatically on any selected set of pins. A padstack can be created beforehand for the TSVs, or it can be created interactively.

To create TSVs:

1. Right-click the required pin or group of pins and choose the*Fanout Vias* command. The Create Fanouts dialog is presented, as shown below.



2. Set the following options in the dialog, as required:

Include Unassigned Pins	Generates TSVs on selected pins, including the ones without nets assigned.

Include All Same Net Pins	Generates TSVs on all selected pins with the same net name. For example, if 10 *VSS* pins are selected, a TSV will be created for each pin instead of a single TSV for the net.
Start	Specifies the layer from which the TSV should originate. This is typically the layer that has the selected pads.
End	Specifies the layer on which to end the TSV.
Via/Padstack	Specifies the padstack to be used. If padstacks exist in the design that match the *Start* and *End* layer selections, they will be displayed in the list. To create a new padstack, select the *Create* button. The *Edit Padstack* dialog is presented to enter padstack values. Once a new padstack is created, it will be listed in the pull-down menu.
Via Direction	Defines the location for the TSVs to be created. This command is also used for other methodologies, including package design. Selecting any option other than *Via in Pad* will create vias offset from the pad center with routes on the *Start* layer. The *Via in Pad* option is recommended for IC design and will create the TSVs in the center of the selected pins.
Line Width, *Pin-Via Space*, *Min Channel Space*	Define parameters for the fan out routing and via spacing when using options other than *Via in Pad*. The options are self-explanatory.

Target – On substrate instance	If enabled, creates vias in the substrate instance (DIE, PACKAGE, BOARD) template. If not selected, System Planner will create vias in the pin's device template.

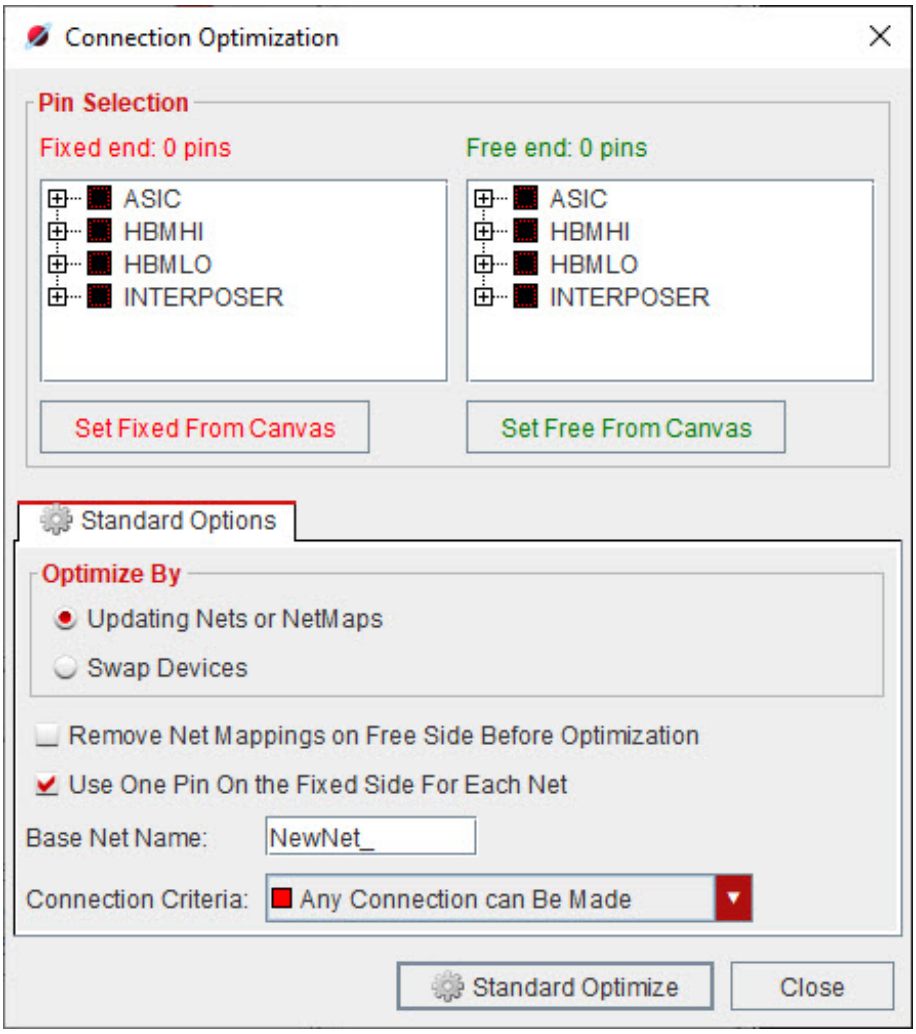
3. Click *Apply* to generate TSVs based on the specified options.

Optimized Bump Assignment with the Connection Optimization Command

Groups of pins can be selected and automatically assigned nets so as to reduce net crossing and length to optimize routing. Pin assignment is done to eliminate net crossing and length. The *Connection Optimization* command is used to perform the optimized assignment. This feature is used for a variety of purposes, including ASIC bump assignment within the die, die to die bump alignment, package ball assignment, and bottom layer C4 bump assignment. It is described for some of these purposes in both the *Designing with the Integrity 3D-IC Platform RAK* and the *ASIC Macro and Bump Floorplanning RAK*.

To perform the optimized assignment:

1. Select the *Automation – Connection Optimization* command from the menu bar to invoke the Connection Optimization dialog, as shown below.



The command works on a set of *Fixed end* pins that are used to drive the automated pin assignment of the *Free end* pins. The *Fixed end* pins are displayed in the left column and the *Free end* pins on the right.

2. Select pins in either column by using any pin selection method, including the*Selection Filter* options. Alternatively, right-click the desired nets in the Devoice Hierarchy and choose *Select – All Pins.*
   Pins can also be selected by traversing the device trees in either column to choose existing System Planner *Interfaces* or *Bundles*. Typically, those features are used for package design only and not covered in this guide. Refer to the *Designing with the Integrity 3D-IC Platform RAK* for more information on using *Interfaces* and *Bundles*.

3. After pins are selected, click the *Set Fixed From Canvas* or *Set Free from Canvas* button to load the form with the selected pins. The count of pins selected appears in the dialog next to *Fixed end* and *Free end* at the top of the dialog.

4. Confirm or update the following *Standard Options* in the dialog, as required:

Updating Nets or NetMaps	Ensures that newly assigned nets are assigned to the *Free end* pins and that any existing mapped nets are updated accordingly.
Swap Devices	Allows devices to be swapped to achieve optimized pin assignments.
Remove Net Mappings on Free Side Before Optimization	Optimizes and reassigns pins on both ends of the net.
Use One Pin On the Fixed Side For Each Net	Enables you to control net assignment in cases where multiple selected pins have the same net name. Select this check box to assign only one free-end pin for any number of selected fixed-end pins with the same net name. Deselect this check box to assign one free-end pin for every fixed-end pin selected, regardless of whether the selected pins have the same net name.
Base Net Name	Defines a net name prefix for newly assigned pins.
Connection Criteria	Defines the connection criteria to be used for assignment. The *Any Connection Can Be Made* option is the default and is the recommended choice for assigning 3D-IC pins. The other options, including *Personality Names must match* and *Auto Bundle Instructions Must Match* support existing personality groups and bundles in the design. These features are primarily used for package design and are not covered in this guide.

5. Select the *Standard Optimize* button to assign pins using the options defined in the dialog. An

example of typical results is shown below.

Interfacing with Other Implementation Tools

System Planner is well integrated within the Innovus®, Allegro® Package Designer Plus, and Virtuoso® implementation tools. There are several levels of integration, including directly launching or importing the System Planner project into the implementation tools. File transfer interaction is also enabled through a variety of file format import and export capabilities. The typical 3D-IC methods are described below.

- The recommended methodology for 3D-IC designs is to invoke System Planner from Innovus to create and manage the system-level design.
- The *Creating Die Symbols for Package Design* section describes the process to create the package topside C4 bumps for any die touching the package substrate, and then exporting the files needed to create the layout symbols in APD. System Planner can also be also used to floorplan the package die and optimally assign package ball pads. System Planner contains many other features specific to package design and APD integration. For more information, refer to the *Designing with the Integrity 3D-IC Platform RAK*.
- The Package substrate design can also be co-designed in the System Planner along with the 3D-IC design. Refer to the *Designing with the Integrity 3D-IC Platform RAK* for more information.
- The *Exporting Implementation Files* section describes some of the typical file formats used to exchange data between System Planner and implementation tools.

Creating Die Symbols for Package Design

System Planner can be used to co-design the package along with the die. System Planner can also be used to floorplan the package die and optimally assign the package ball pads. System Planner contains many other features specific to package design and APD integration. This methodology is not contained in this guide. For more information, refer to the *Designing with the Integrity 3D-IC Platform RAK*.

The section describes the process to create die text files used to create the symbols used in APD package design for any die touching the package substrate. This process is often performed just once to create the symbol files needed for package design. The package device may not be desired in the System Planner project going forward, so this process can be done in a copy of the

design or removed afterward.

Creating a Package Device

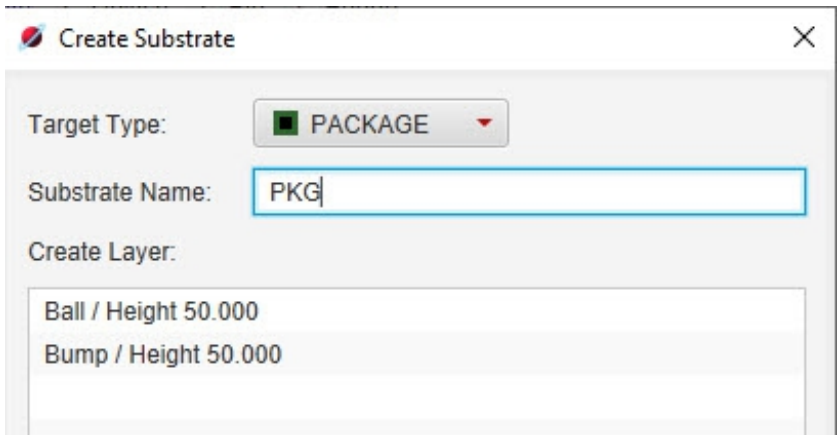
The first step in the process of creating die symbols is to create an empty package device in the design. The package balls and other package-relevant data can also be included, but they are not covered in this guide. The package device is created empty for this process.

A simple default substrate layer stack is created since you are only concerned with the top-layer to generate the contact bumps. If you have a substrate for the package in a CSV or LEF tech file, it can also be imported and used but is not really needed.

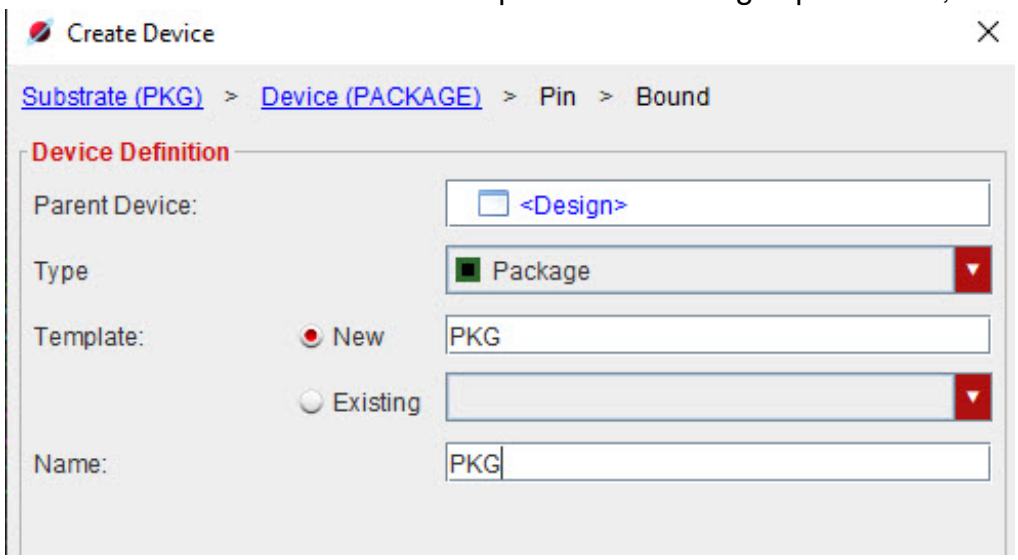
To create an empty package device:

1. Select *Edit – Devices – Create Device* from the menu bar to invoke the *Create Device* dialog, as shown below.

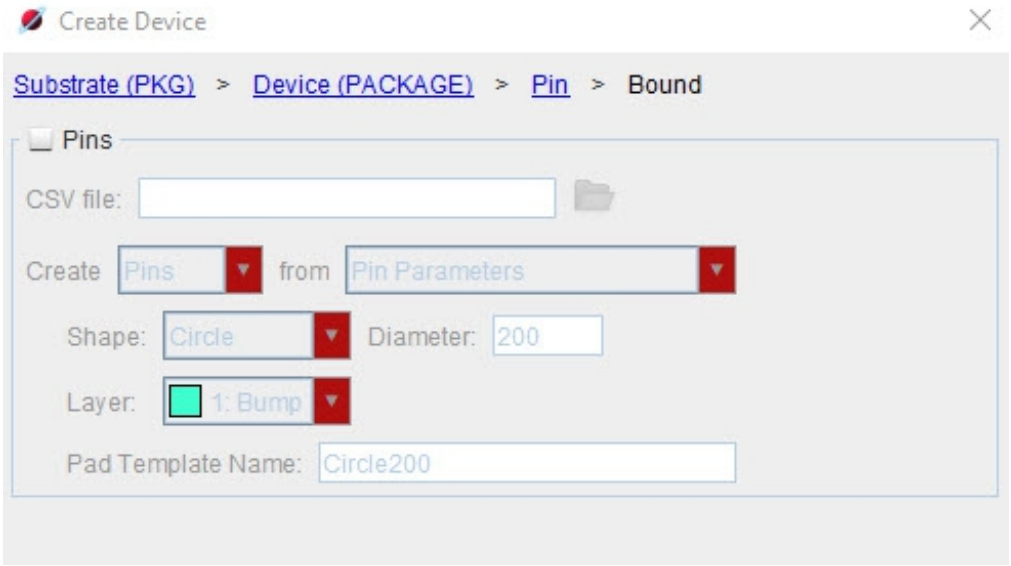
2. If a substrate for the package device has been previously imported or created, select it from the *Substrate* pull-down menu. Else, to create a new package substrate, click the *Create Substrate* icon ( ) and select *New*. The *Create Substrate* dialog is presented, as shown below.



3. Select *PACKAGE* from the *Target Type* pull-down menu and enter a package device name in the *Substrate Name* field, as shown above. Click *OK* to create the package substrate with the default layer stackup.
4. Notice the package substrate you created is now available in the*Substrate* pull-down menu in the Create Device dialog. Select the new package substrate from the *Substrate* pull-down menu and click *Next*. The second pane of the dialog is presented, as shown below.



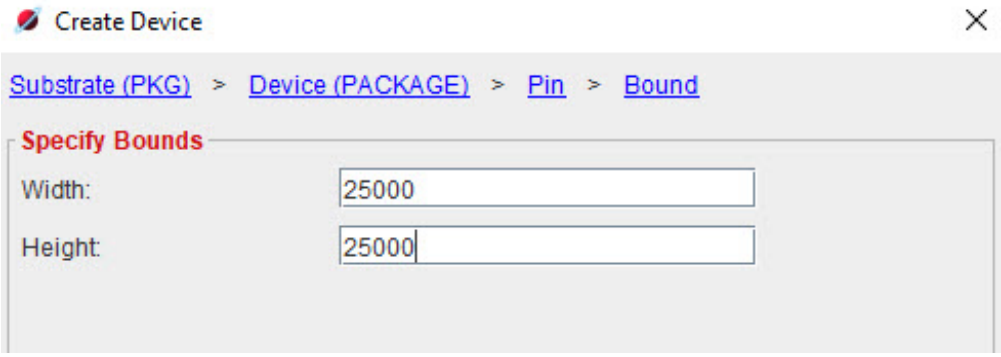
5. Select *Package* from the *Type* pull-down menu, enter names in the*Template* and *Name* fields and click *Next*. The third pane of the Create Device dialog is presented, as shown below.



6. The third pane enables pins to be entered from a CSV pins file and pad geometries to be defined. For example, this is where package balls could be imported on the *Ball* layer. Since

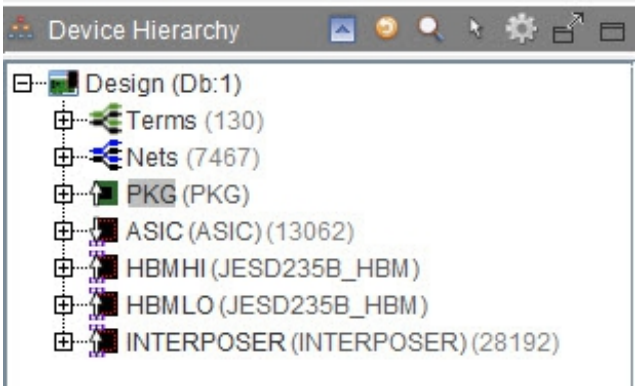
we are creating an empty package, click *Next* to continue.

7. In the fourth pane of the Create Device dialog, you can define the package size. Enter values in the *Width* and *Height* fields, as shown below.



For the proper die size to be output in the die text file, enter the size of the die device, not the package device. It also may not be important to use a realistic size, if not known. The output pin files for the APD symbols will use X-Y coordinates based on the origin defined for the package. The device size will be included in the die text file but can be changed easily by the package designer.

8. Click *Create* to create the package device and click *Close*. The new package device is now displayed under the top-level design in the Device Hierarchy, as shown below.



Configuring and Placing the Package Device

The process described below is intended to be used to create the files specific to creating a die symbol for use in APD for package design. It is not intended to floorplan the die in an actual intended location on the package. The process involves aligning the origins of the two devices to produce X-Y coordinates for the pins within the die symbol. The process needs to be repeated for each die sitting on the package substrate.

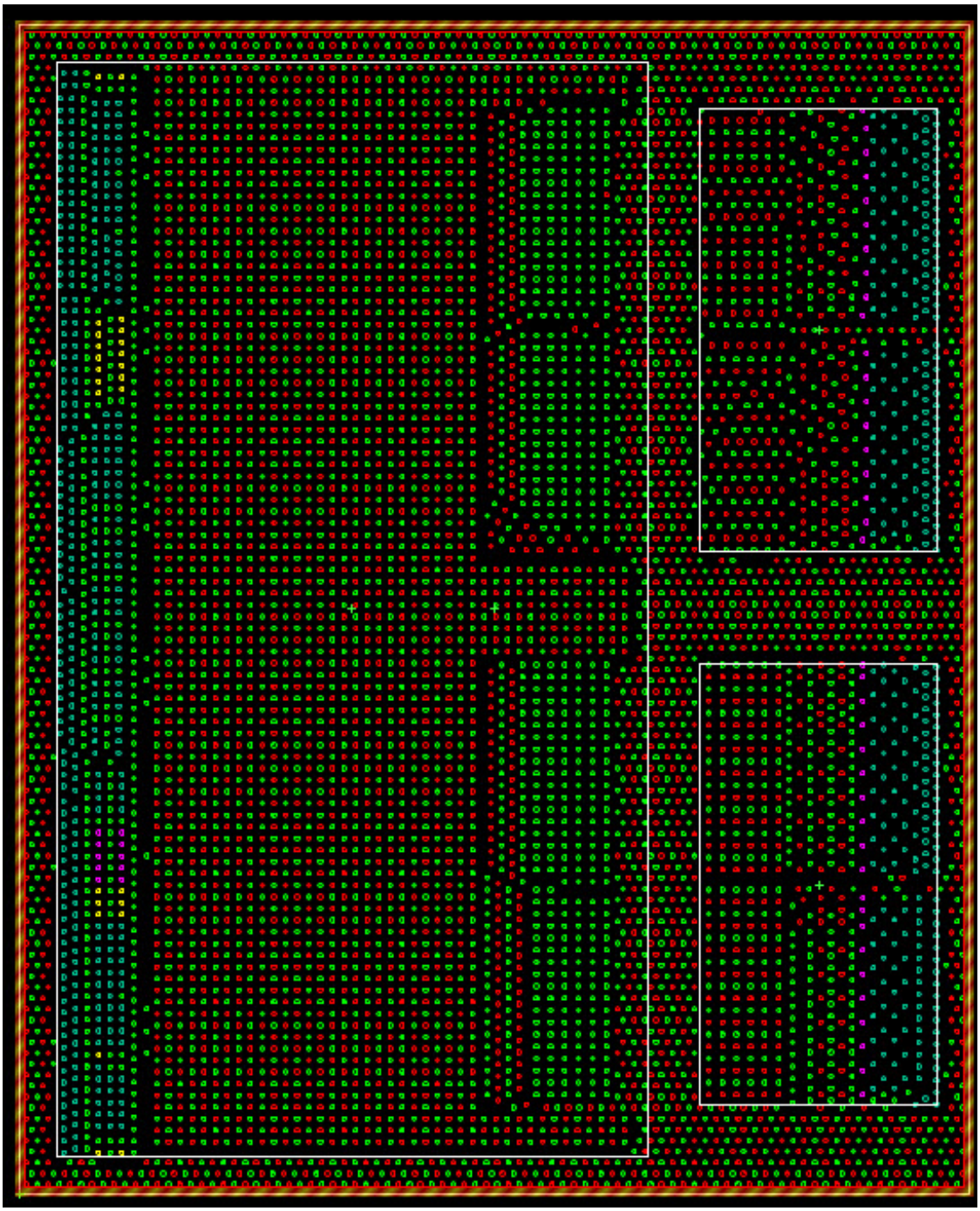
Defining the Package Device Origin

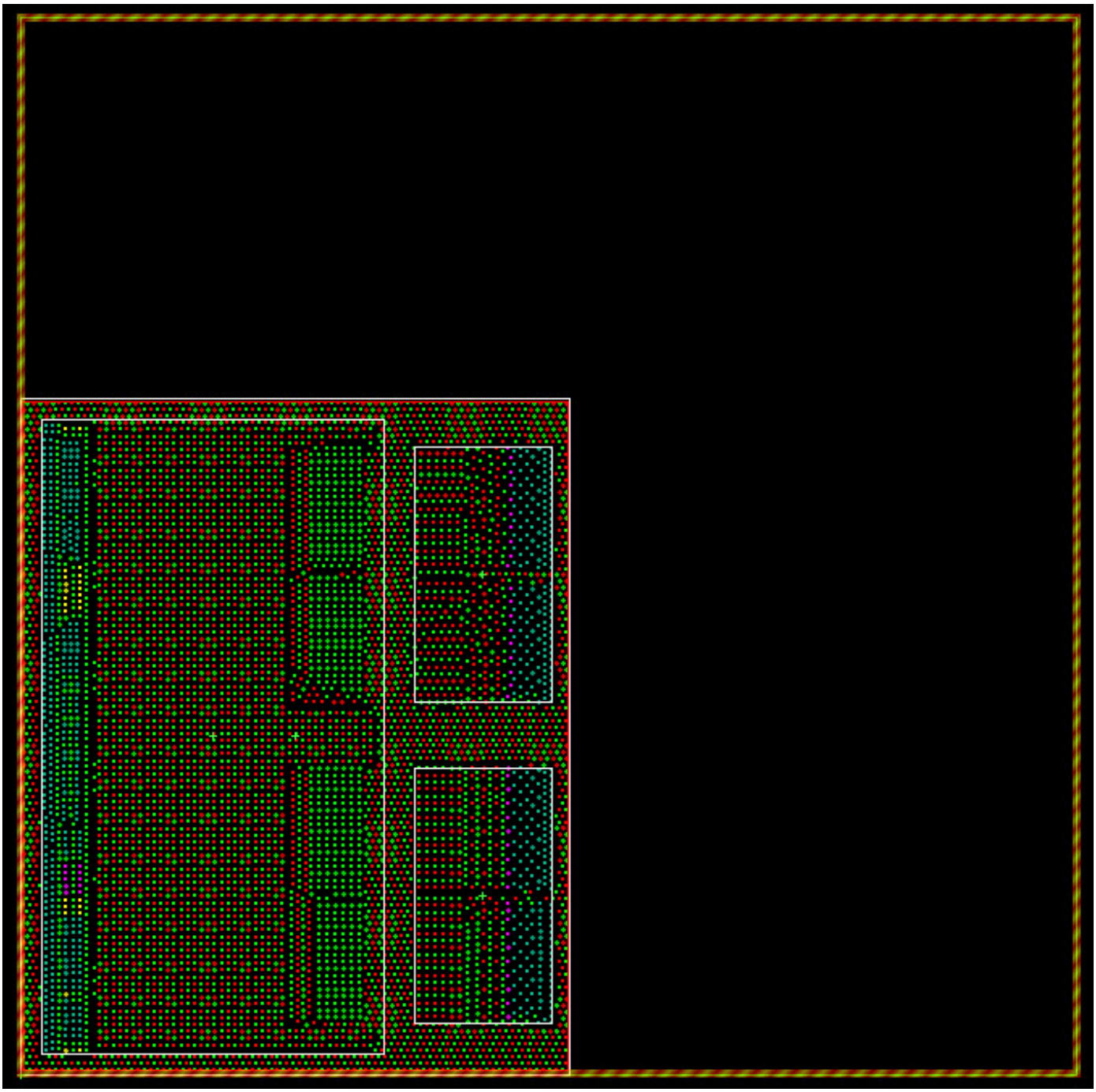
The package device origin and placement location need to be defined to ensure proper X-Y coordinates are output to the die text file. Typically, package designers need all positive coordinates in the die text file to create the die symbols, so a lower left origin is preferred. The package device typically gets created with a center origin by default. To define the package origin, right-click the device in the Device Hierarchy and choose *Move – Origin – Lower Left.*

Adjust the Package Device Placement

The package device placement needs to be placed in a location to properly generate the correct X-Y coordinates in the pin list files for the die. This is not the same placement used to floorplan the die on the substrate. The package placement is being used to generate die symbol-specific coordinates, not package-specific coordinates. The package designer uses these files to create symbols for the die on the package substrate. Once the symbols are created, the package designer can move them to the desired locations on the package substrate.

The typical package placement for die symbol creation aligns the two same size devices exactly, or just the lower-left corners of the package and die, as shown in the two examples below.



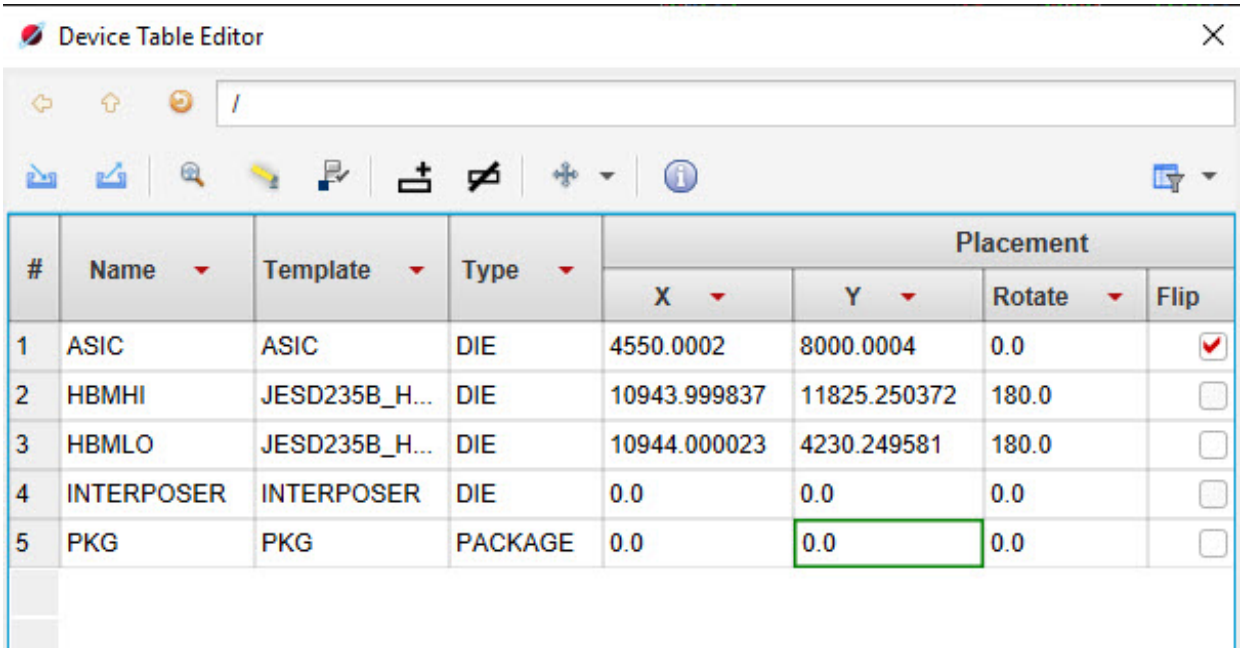


In the example above, the Interposer and the package origins are aligned in the lower left and will generate X-Y coordinates based on that origin.

There are several methods to adjust the package device placement, including the Attribute Editor, Devices Table Editor, and the *Move – Device* Parametrically command in the Device Hierarchy.

If the origin of the die device is also lower left, select the package device and change the *Loc X* and *Loc Y* fields in the Attribute Editor to *0*.

Another method is to select the parent-level of the hierarchy above the package device in the Device Hierarchy, and right-click to choose *Devices Table*. The *Device Table Editor* is presented, as shown below.



#	Name	Template	Type	Placement			
				X	Y	Rotate	Flip
1	ASIC	ASIC	DIE	4550.0002	8000.0004	0.0	✔
2	HBMHI	JESD235B_H...	DIE	10943.999837	11825.250372	180.0	
3	HBMLO	JESD235B_H...	DIE	10944.000023	4230.249581	180.0	
4	INTERPOSER	INTERPOSER	DIE	0.0	0.0	0.0	
5	PKG	PKG	PACKAGE	0.0	0.0	0.0	

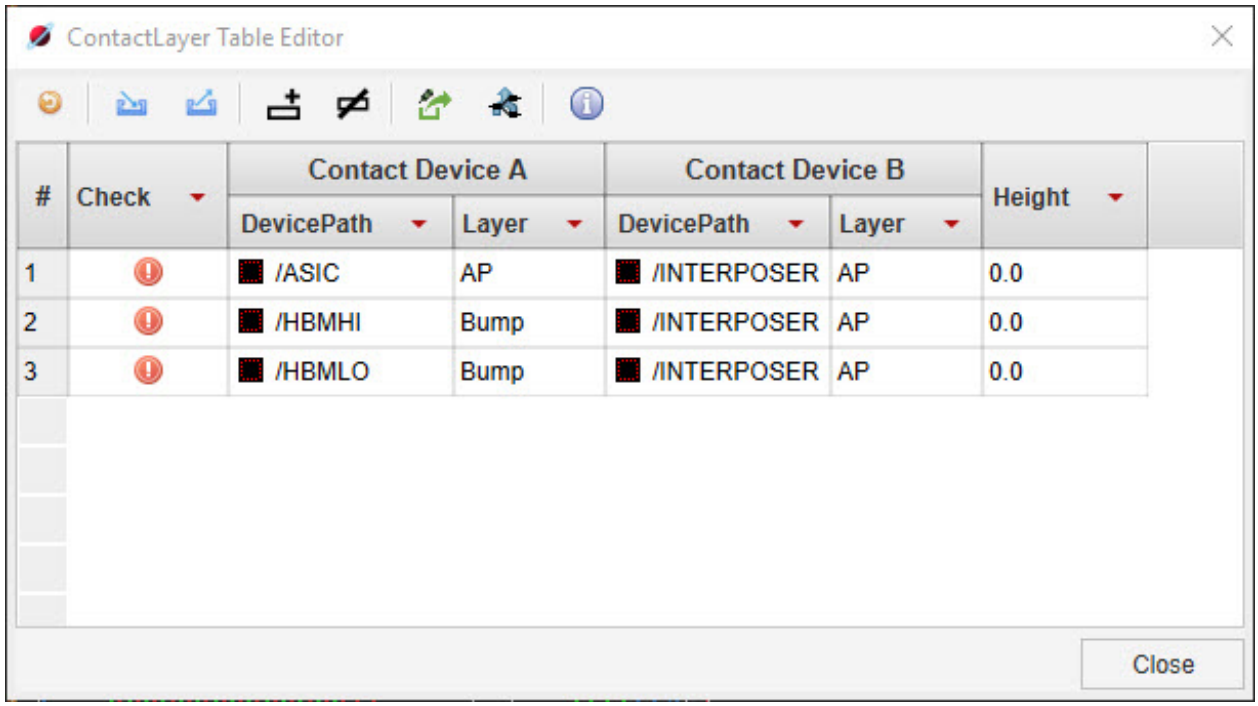
The fields in the *X* and *Y* columns can be selected and modified for any device to reflect new device locations. You must press the *Enter* key after specifying a new value in any field for the change to take effect.

Creating Package Substrate Contact Pads

After aligning the package device origin with the die device, the next step is to generate contact pads on the package substrate to match the die locations. This is done by creating a contact layer between the two devices. As a part of creating the contact layer, contact pads can also be generated.

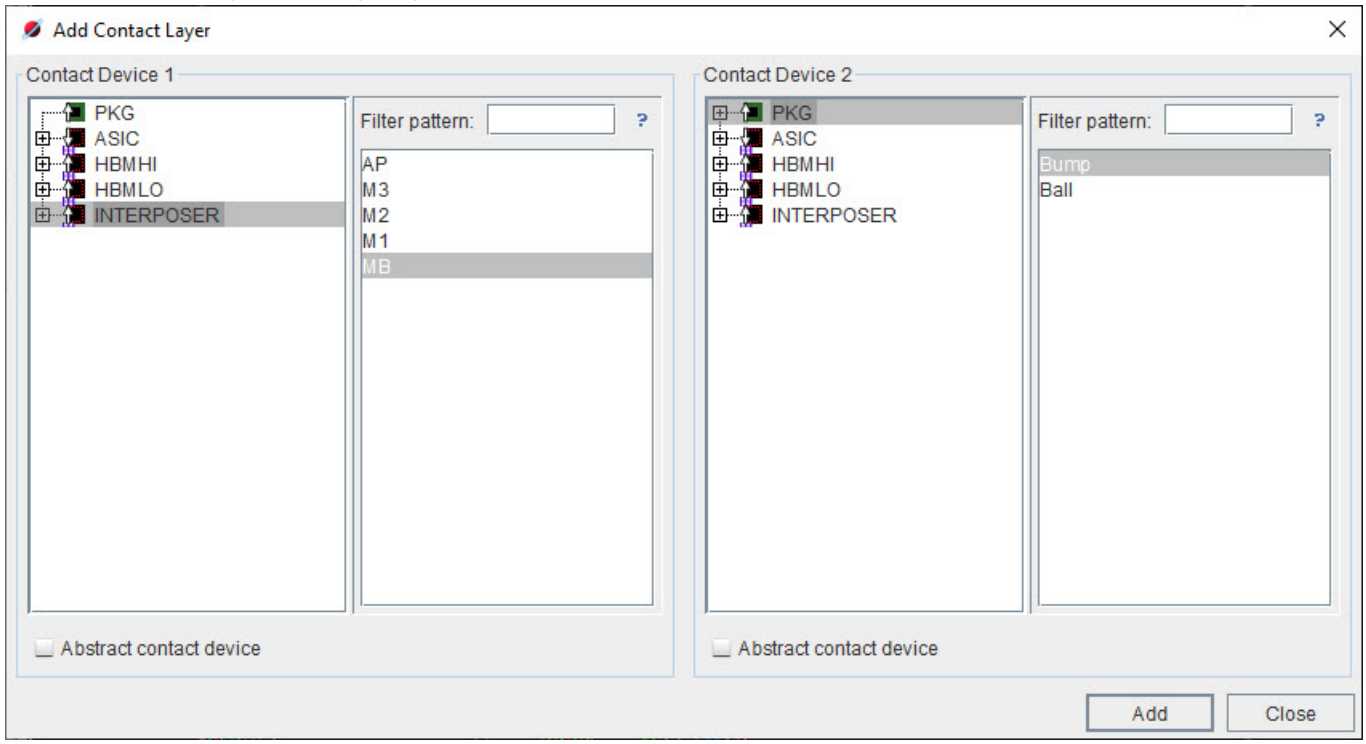
To create the contact layer and pads:

1. Select *Edit – Contact Layer* from the menu bar. The Contact Layer Editor is presented, as shown below.



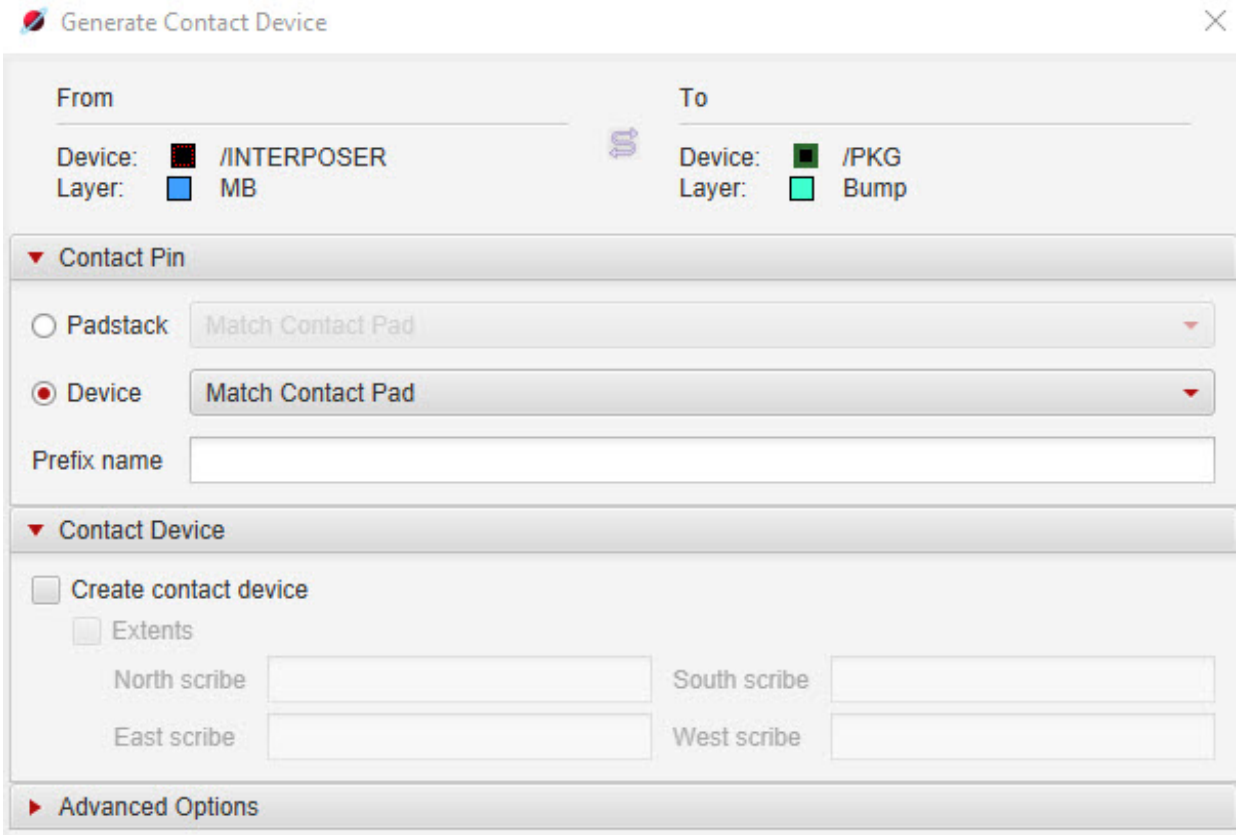
The Contact Layer Editor is used to create and manage all the contact layers defined in the design. It displays all existing contact layers defined in the design.

2. To create the new package to die contact later, click the *Add Contact Layer* icon ( ) . The *Add Contact Layer* dialog is presented, as shown below.



3. The Add Contact Layer dialog is used to establish contact relationships between two devices. To create a new contact layer, select the source device and existing pad layer from the *Contact Device 1* column and the destination device and layer from the *Contact Device 2* column. In the example above, *INTERPOSER* and *MB* are selected in the *Contact Device 1* column and *PKG* and *Bump* are selected from the *Contact Device 2* column. This will establish the physical relationship and contact layers between the two devices.
4. Click *Add*. This opens the Contact Device dalog shown below.

5. Click *Yes* to create the contact pads on the package substrate. Clicking *No* will establish the contact relationship, but not generate contact pads. On clicking *Yes*, the Generate Contact Device dialog is displayed, as shown below.

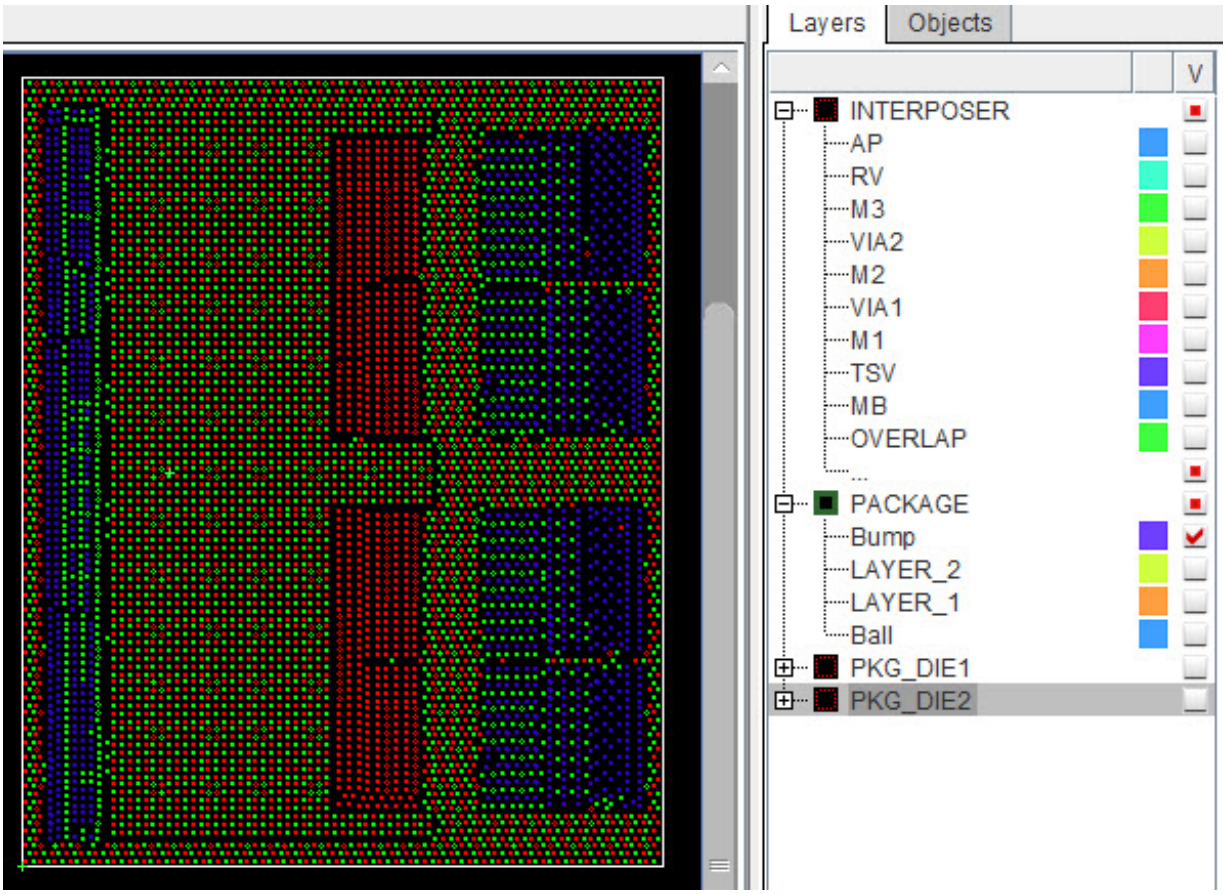


6. Select *Device* and use the default *Match Contact Pad* option and then click *OK* to generate

the contact pads on the package substrate.

7. Close the Add Contact Layer dialog.

In the *Layers* view in the View Options panel, right-click anywhere and choose the *All Layers off* command, and then select the package substrate layer used to generate the contact pads. In this example, it is the *Bump* Layer. Notice the pads are displayed on the package substrate layer.



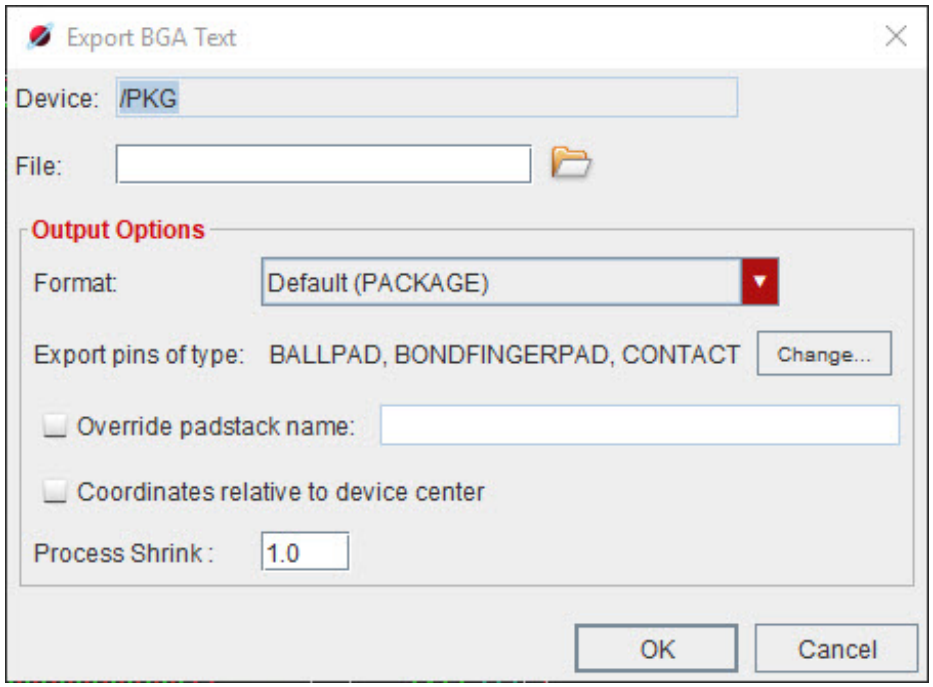
Optionally, you can turn on net names by clicking the *Toggle display of pin net names* icon (Net) on the View Options panel and zoom in to view the nets assigned to the pads.

Exporting a Die Text File for the Die Symbol Creation

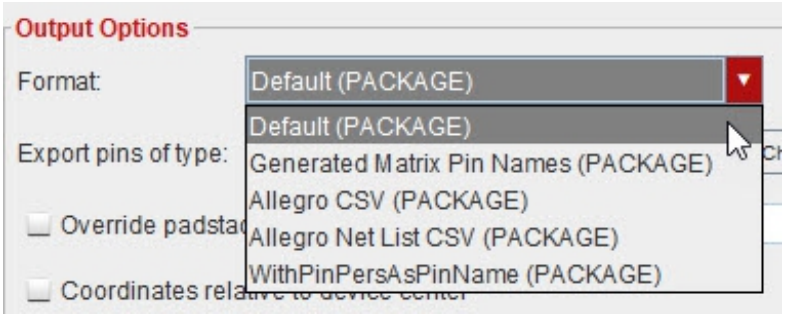
The desired file format for PAD symbol creation is a die text file. The CSV pins file format can also be used, but it does not include the device outline.

To generate the die text file for the die pins:

1. Right-click the package device in the Device Hierarchy and choose *Export – Text*. The Export BGA Text dialog is presented, as shown below.



2. Confirm or update the various option settings in the dialog, as required:

Device	Displays the device from which to extract the pin locations. It should display the package device.
File	Specifies the name and location of the exported text file.
Format	Specifies the output format for the text file. Choose the desired format from the pull-down menu, as shown below.  The options are self-explanatory. Files can be exported with the various options to explore the formats. The *Default (PACKAGE)* option can be used for APD symbol creation.

Export pins of type	Lists the type of pins to be included in the file. The *Device* type pins that were created earlier for the package contact pads require BUMPPAD to also be included. Click the *Change* button and select *BUMPPAD* from the list.
Override padstack name	Enables an alternate padstack name to be used in the file.
Coordinates relative to device center	Produces X Y coordinates relative to the package device center.
Process Shrink	Adjusts coordinates based on the shrink value entered.

3. Click *OK* to create the file based on the options selected.

An example of the *Default* format die text file is displayed below.

```
File: C:\Users\Documents\interposer_die_text.txt
Date: Thu Jan 14 15:30:19 MST 2021
Units: microns, 3 decimal places
Name: PKG
RefDes: PKG
Origin: (0.0 0.0)
Rotation: 0.000000
Pad Layer: Bump
Extents: ((0.000000 0.000000) (25000.000000 25000.000000))

Pin Number      Padstack        X Coord         Y Coord         Rotation        Net Name
C4S_11          C4_PAD          1029.9995       13659.001       0               ASIC_DTB1[24]
C4S_110         C4_PAD          1398.0          5261.5005       0               ASIC_TAP_TCK
C4S_111         C4_PAD          1078.0005       3666.5005       0               ASIC_DTB2[15]
C4S_112         C4_PAD          1238.0005       3666.5005       0               ASIC_DTB2[16]
C4S_113         C4_PAD          1398.0005       3666.5005       0               ASIC_DTB2[17]
C4S_114         C4_PAD          1078.0005       3506.5005       0               ASIC_DTB2[12]
C4S_115         C4_PAD          1238.0005       3506.5005       0               ASIC_DTB2[13]
C4S_116         C4_PAD          1398.0005       3506.5005       0               ASIC_DTB2[14]
C4S_117         C4_PAD          1078.0005       3346.5005       0               ASIC_DTB2[9]
C4S_118         C4_PAD          1238.0005       3346.5005       0               ASIC_DTB2[10]
```

The die text file can then be used in APD to generate the symbols needed for the package design. The pads will include the net names needed for netlist creation.

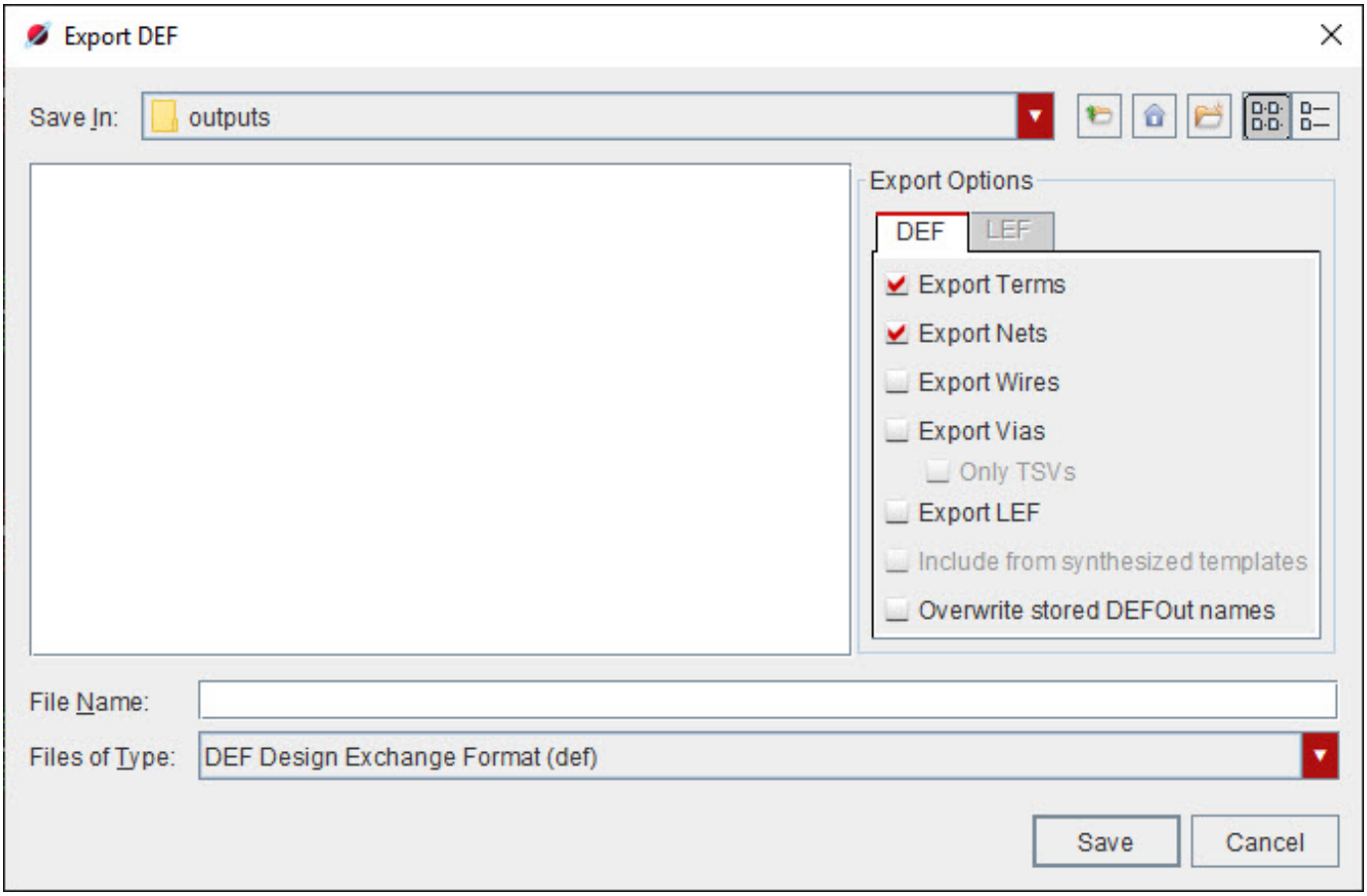
Exporting Implementation Files

System Planner enables the export of a variety of file formats to exchange data with various layout tools. These files can be exported for specific devices in the design by right-clicking the device in the Device Hierarchy and selecting the desired format from the list shown in the *Export* submenu.

Several of the common formats used for 3D-IC are described below.

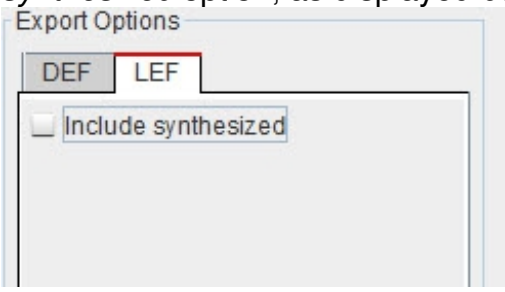
Exporting LEF/DEF Files

To export LEF or DEF format files, right-click the device in the Device Hierarchy and choose *Export – DEF*. The *Export DEF* dialog is presented, as shown below.



Confirm or update the following option settings in the dialog, as required:

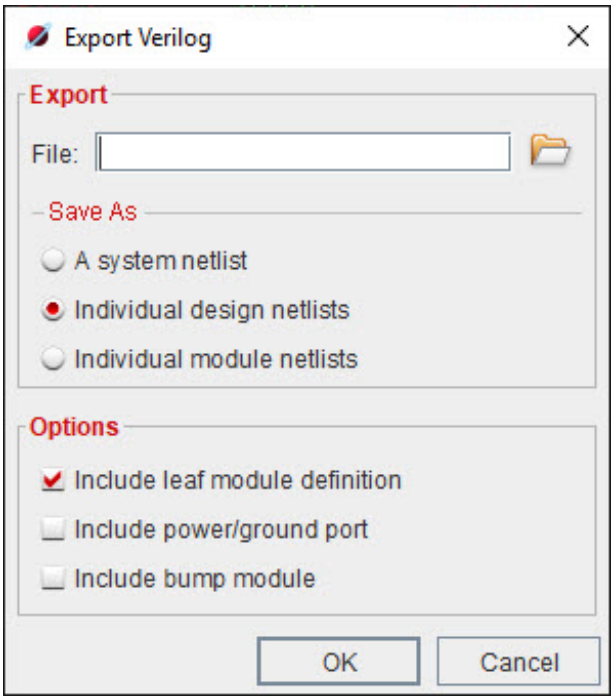
Save In	Allows you to browse to the directory to which you want to write the exported file(s). The large pane below displays the files and directories in the specified *Save In* directory.

File Name	Specifies a file name for the exported file.
Files of type	Defaults to the DEF format to export the file in the proper format.
Export Options	
Export Terms	Includes terms in the exported DEF file.
Export Nets	Includes nets in the exported DEF file.
Export Wires	Includes wires in the exported DEF file.
Export Vias	Includes vias in the exported DEF file. Select the *Only TSVs* option to filter vias to only TSVs.
Export LEF	Exports LEF format files as well for the tech file layer stackup, pads used on the selected device substrate, and other LEF design data. Selecting this option enables the LEF tab to be selected. The LEF tab includes the *Include synthesized* option, as displayed below. 
Include from synthesized templates	Includes devices of synthesized templates (non-LEF Macro) in the exported DEF file.
Overwrite stored DEFOut names	Overwrites DEFOut names previously stored in the design with new names; otherwise, existing names will be exported to the DEF file.

Click *Save* to create the LEF/DEF files based on the options selected.

Exporting Verilog Files

To export Verilog format files, right-click the device or the top-level of hierarchy in the Device Hierarchy and choose *Export – Verilog*. The *Export Verilog* dialog is presented, as shown below.



The dialog options are self-explanatory.

Click *OK* to create the files based on the options selected.

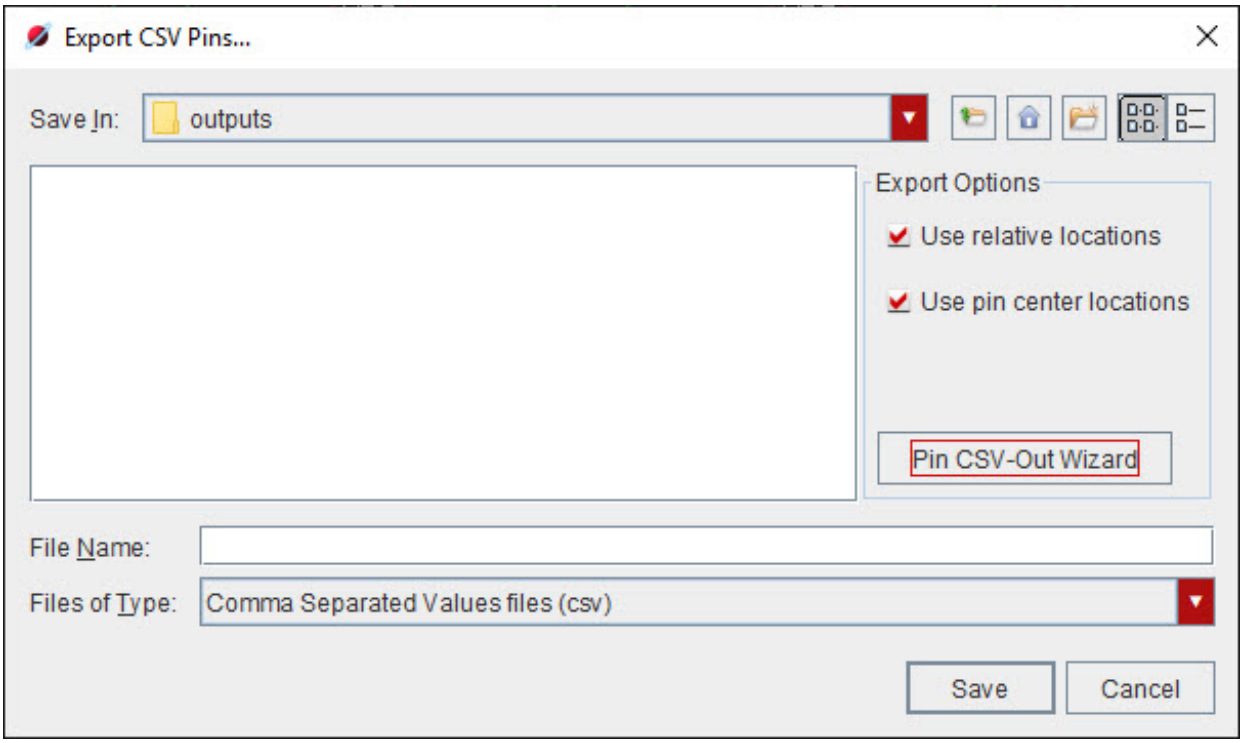
Exporting a CSV Pins File

System Planner provides the following commands to export various types of design information in the CSV file format. Most are self-explanatory or not applicable for typical 3D-IC designs.

Exporting a CSV Pins File for a Device

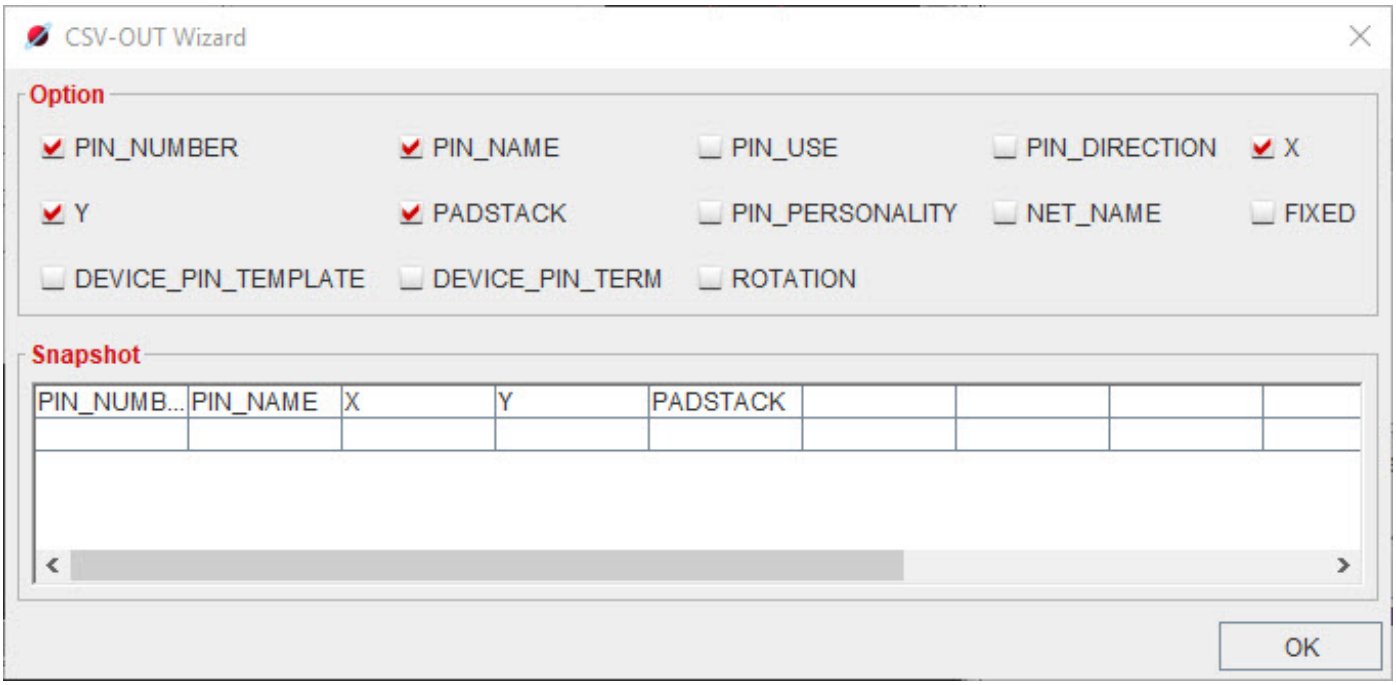
The CSV pins list file is often used to import or export pin or bump locations for a device.

To create a CSV pins list file, right-click the device in the Device Hierarchy and choose *Export – CSV Pins*. The *Export CSV* Pins dialog is presented, as shown below.



The options are self-explanatory.

Clicking the *Pin CSV-Put Wizard* button enables the content of the file to be defined, as shown below.



The format of the file to be created is displayed in the*Snapshot* area.

Options that are selected in the Wizard will remain as the default going forward.

Clicking *OK* in the *Export CSV Pins* dialog will create the file based on the options selected.

Exporting a CSV Devices File

The CSV devices file is used to save the placement locations for the devices exported. It can be imported to replicate the placement later in other designs. All devices under the selected device or level of the hierarchy will be included in the file.

During import, if devices exist in the file that is not in the design, they will be created in the design.

To create a CSV devices file, right-click the required device or level of hierarchy in the Device Hierarchy and choose *Export* – CSV *Devices*. The *Export CSV Devices* dialog provides the options to enter an output file name and the desired origin to be used for the X Y coordinates in the file.

6

Interoperability Commands and Features

Integrity 3D-IC is comprised of System Planner and Layout Implementation.

System Planner accesses all the devices in a 3D-IC system and provides a system-level environment for performing 3D-stack planning and die-to-die, die-to-package I/O, bump and TSV generation and assignment. It is also a cockpit to launch various system-level analysis and connect to other implementation tools.

Layout Implementation is designed for digital die implementation, which shares most of the functions with Innovus. It is also enabled with multiple top cell mode that can display the stacked dies in the layout window with all the instances and routing details.

System planning and digital die implementation usually have quite different requirements. System planning needs a lightweight view of each device that focuses mainly on floorplan, IO, and IP blocks. It does not need to see all the standard cell placement and routing. In contrast, implementation works on all the objects of a digital die and needs a much larger capacity. Therefore, Integrity 3D-IC has two separate views for planning and layout implementation. The planner and layout views can work independently, and Integrity 3D-IC provides an easy way to sync up between the two views.

When a digital die is implemented or has an existing database, such as in the bottom-up flow, you can use the `push_to_planner` command to view it in System Planner and create or optimize the bumps with other dies or packages.

When the bumps are created or optimized for a digital die in System Planner, you can use the `pull_from_planner` command to pull it from planner to layout for further implementation, such as place and route.

For more information on these commands, see Integrity-Specific Tcl Commands.

Integrity 3D-IC allows you to push to or pull from planner for a single die or multiple dies (multiple top cell mode).

7

Tcl Interaction and Commands

The Integrity 3D-IC command line accepts Tcl commands for both the Layout and System Planner windows.

- Executing Tcl Commands for the Layout Window
- Executing Tcl Commands for the System Planner Window
- Controlling the Planner Command Log Mode
- Checking the Version

Executing Tcl Commands for the Layout Window

To execute a Tcl command for the Layout window, follow exactly the same method as in Innovus™ Implementation System.

For example:

```
integrity_3dic 1> selectInst block1
```

For Integrity-specific Tcl Commands, see Integrity-Specific Tcl Commands.

For details on other Innovus Tcl Commands, including syntax and examples, refer to *Innovus Text Command Reference*.

Executing Tcl Commands for the System Planner Window

To execute a Tcl command for the System Planner window, use the `planner` name space.

For example:

```
integrity_3dic 1> planner::create_design -name Design
```

Alternatively, use `eval_planner` for multiple commands:

```
integrity_3dic 1> eval_planner { create_design -name Design1
    create_substrate -lefs "all.lef" -name die1
    read_def die1.def -substrate die1
    }
```

For details on the Tcl commands for System Planner, see System Planner Tcl Commands.

Controlling the Planner Command Log Mode

To control the way messages are displayed for the `planner` name space commands, use the `planner::command_log_mode` variable.

```
set planner::command_log_mode value
```

The possible values are:

- `0`: Neither logs nor displays the messages on the screen.
- `1`: Logs the messages but does not display them on the screen.
- `2`: Logs and displays the messages on the screen.

Default: `1`

For example, to display planner command messages both in the log and on the screen, use the following command:

```
set planner::command_log_mode 2
>> 2
```

Checking the Version

To check the version, use the following command:

```
integrity_3dic -version
```

Output has the following format:

```
Detected cadence.com domain. This build is not fully tested or qualified. Use it at your own risk.
@(#)CDS: Integrity 3DIC System v21.15-e085_1 (64bit) 08/18/2022 14:35 (Linux 3.10.0-693.el7.x86_64)
@(#)CDS: NanoRoute 21.15-e085_1 NR220815-0224/21_15-UB (database version 18.20.592) {superthreading v2.17}
@(#)CDS: AAE 21.15-s026 (64bit) 08/18/2022 (Linux 3.10.0-693.el7.x86_64)
@(#)CDS: CTE 21.15-s029_1 () Aug 17 2022 21:25:24 ( )
@(#)CDS: SYNTECH 21.15-s008_1 () Aug 1 2022 09:52:19 ( )
@(#)CDS: CPE v21.15-s060
@(#)CDS: IQuantus/TQuantus 21.1.1-s867 (64bit) Sun Jun 26 22:12:54 PDT 2022 (Linux 3.10.0-693.el7.x86_64)
@(#)CDS: OA 22.60-p076 Wed May 18 13:45:29 2022
@(#)CDS: SGN 20.10-d001 (01-Jun-2020) (64 bit executable, Qt5.9.0)
@(#)CDS: RCDB 11.15.0
@(#)CDS: STYLUS 21.12-s013_1 (08/15/2022 08:25 PDT)
@(#)CDS: SystemPlanner-21.15Rel-8679 (21.15) (2022-08-18 17:08:47+0800)
```

Integrity-Specific Tcl Commands

- clear_floorplan_for_proto_seeds
- create_proto_seeds
- get_proto_seeds
- proto3d::assign_shadow_bump
- proto3d::create_3d_stacked_proto_shapes
- proto3d::create_module_wrapper
- proto3d::get_obj_die_id
- proto3d::insert_zero_delay_cell
- proto3d::set_obj_die_id
- proto3d::unfold_floorplan
- proto_design
- pull_from_planner
- push_to_planner
- set_proto_design_mode
- update_proto_seeds

clear_floorplan_for_proto_seeds

```
clear_floorplan_for_proto_seeds
[-help]
```

Cleans up the floorplan for proto seed placement. Placement and route blockages will be removed. The tool will keep fixed top-level macros in the floorplan and define them as proto seeds with `soft_fixed` constraints.

Parameters

`-help`	Prints a brief description for each `clear_floorplan_for_proto_seeds` parameter. *Data_type*: optional

Example

- The following command clears the floorplan for proto seed placement:
  `clear_floorplan_for_proto_seeds`

create_proto_seeds

```
create_proto_seeds
[-help]
[-group_top_logic [-max_area_of_top_logic_group float]]
[-groups {list_of_instGroups}]
[-hinsts list_of_hinsts| -from_partitions]
[-macros list_of_macros]
[-util float]
```

Enables you to specify placement seeds easily. Integrity 3D-IC supports incremental operation in which you can specify seeds with multiple `create_proto_seeds` commands.

Parameters

`-help`	Prints a brief description for each `create_proto_seeds` parameter. *Data_type*: optional
`-from_partitions`	Creates proto seeds from partitions in the design. *Data_type*: bool, optional
`-group_top_logic`	Creates proto seeds for top logic. When `-group_top_logic` is specified, the tool will group all logics that are not part of proto seeds into a group, and define it as a proto seed. *Data_type*: bool, optional
`-groups {list_of_instGroups}`	
	Specifies a list of instance group names as placement seeds. *Data_type*: optional
`-hinsts list_of_hinsts`	
	Specifies a list of hierarchical instance names as placement seeds. Supports wildcards. *Data_type*: string, optional
`-macros list_of_macros`	
	Specifies a list of top-level macros as seeds. Supports wildcards. **Note**: Top-level macros are the macros whose parent hierarchical modules are not defined as proto seeds. For lower-level macros, you should define their parent hierarchical modules as proto seeds. *Data_type*: string, optional
`-max_area_of_top_logic_group float`	
	Specifies the maximum area of each top logic group. *Data_type*: float, optional
`-util float`	Specifies the target utilization for created proto seeds. *Data_type*: float, optional

Example

- The following command creates instGroups as proto seeds for glue logic. The maximum area of each instGroup is `1000` um² and the utilization of the proto seeds will be set to `0.5`:
  ```
  create_proto_seeds -group_top_logic -util 0.5 -max_area_of_top_logic_group 1000
  ```

Related Command

- `update_proto_seeds`

get_proto_seeds

```
get_proto_seeds
[-help]
[-die_id integer]
[-not]
[objs]
[-ptr]
[parameter_name]
```

Displays information about the constraints of proto seeds.

Parameters

`-help`	Prints a brief description for each `get_proto_seeds` parameter. *Data_type*: optional
`-die_id` *`integer`*	Specifies the die ID. *Data_type*: int, optional
`objs`	Specifies a list of object names. The names are limited to the seeds created. *Data_type*: string, optional
`-not`	Returns the inverse list. *Data_type*: bool, optional
`-ptr`	Returns the pointer. *Data_type*: bool, optional
`parameter_name`	Returns information on the specified parameter. The allowed values are: `-name | -util | -spacing | -aspect_ratio_range | -honor_master | -orient | -cluster | -exclusive_cluster | -blocked_layers | -ignore_keepout_blockage | -soft_fixed | -independent` For example, use `-name` to get the seed name.

Examples

- The following command returns the proto seeds whose die id is `1`:

  ```
  get_proto_seeds -name -die_id 1
  ```

- The following command returns the target utilization of the proto seed `M1`:

  ```
  get_proto_seeds M1 -util
  ```

Related Commands

- `proto3d::set_obj_die_id`
- `proto_design`
- `update_proto_seeds`

proto3d::assign_shadow_bump

```
proto3d::assign_shadow_bump
[-help]
[-exclude_macro inst|hInst+]
[-honor_place_blockage pblk]
[-out_file filename]
-ref_hinsts inst|hInst+
[-skip_pg_nets]
{[-pitch_x float -pitch_y float] [-bumpCell cell_name] [ -use_existing_bump_file in_file]}
[[[-offset_x float] [-offset_y float] [-pitch_x float] [-pitch_y float] [-bumpCell cell_name]] | [-use_existing_bump_file
in_file]]
```

Generates the shadow bump location file.

⚠ The `proto3d::assign_shadow_bump` command is a Beta feature. Contact the Cadence support team for usage details.

Parameters

`-help`	Prints a brief description for each `proto3d::assign_shadow_bump` parameter. *Data_type*: optional
`-bumpCell` *`cell_name`*	
	Specifies the bump cell to be inserted. *Data_type*: string, optional
`-exclude_macro` *`inst|hInst+`*	
	Excludes the macros in the specified die; the tool will not assign bumps in the area of macros. *Data_type*: (inst|hInst)+, optional
`-honor_place_blockage` *`pblk`*	
	Honors user-specified placement blockages. This option supports wild cards, so `*` means to honor all placement blockages. *Data_type*: string, optional
`-offset_x` *`float`*	Specifies the offset `x` of the die hinst box. *Data_type*: float, optional
`-offset_y` *`float`*	Specifies the offset `y` of die hinst box. *Data_type*: float, optional
`-out_file` *`filename`*	
	Specifies the name of the bump location file. *Data_type*: string, optional
`-pitch_x` *`float`*	Specifies the pitch `x` of the die hinst box. *Data_type*: float, optional

`-pitch_y` *`float`*	Specifies the pitch `y` of the die hinst box. *Data_type*: float, optional
`-ref_hinsts` *`inst|hInst+`*	
	Specifies the list of ref die hinsts names to which the shadow bump is to be assigned. *Data_type*: (inst|hInst)+, required
`-skip_pg_nets`	
	Skips the generation of power/ground (PG) bumps in the bump file. *Data_type*: boolean, optional
`-use_existing_bump_file` *`in_file`*	
	Uses an existing bump file to assign shadow bumps. **Note**: `-use_existing_bump_file` cannot be used with options `-offset_x`, `-offset_y`, `-pitch_x`, `-pitch_y`, and `-bumpCell`. *Data_type*: string, optional

Example

- The following command generates a bump location file referring to the placement of the `itop` die hinst, the offset for `x` and `y` is `8` um, and the pitch for x and y is `6 um`.

```
proto3d::assign_shadow_bump -ref_hinsts itop -offset_x 8 -offset_y 8 -out_file shadow_bump_loc.xy -pitch_x 6-pitch_y 6
```

proto3d::create_3d_stacked_proto_shapes

```
proto3d::create_3d_stacked_proto_shapes
[-help]
[-channel_width float]
[-max_iteration integer]
[-min_step_x float]
[-min_step_y float]
```

Generates and optimizes partition shapes per die.

Parameters

`-help`	Prints a brief description for each `proto3d::create_3d_stacked_proto_shapes` parameter. *Data_type*: optional
`-channel_width` *`float`*	Specifies the uniform channel for the design. *Data_type*: float, optional
`-max_iteration` *`integer`*	Specifies the maximum iteration for proto shapes optimization. *Data_type*: integer, optional
`-min_step_x` *`float`*	Specifies the minimum step x. *Data_type*: float, optional
`-min_step_y` *`float`*	Specifies the minimum step y. *Data_type*: float, optional

proto3d::create_module_wrapper

```
proto3d::create_module_wrapper
[-help]
-bottom_die_cell cell_name
-bottom_die_hinst hinst_name
[-skip_feedthrough_insertion]
[-skip_port_cloning]
-top_die_cell cell_name
-top_die_hinst hinst_name
[-from_die_id | -user_top_die_instgroup string]
```

Partitions the design netlist logically. Use the `proto3d::create_module_wrapper` command to:

- Reorganize the original netlist by creating two new top-level hinsts that represent the top and bottom die of the design. By default, the tool puts all of the hard macros into the top die and all the standard cells into the bottom die. All the primary I/Os connect to the bottom die.
- Create feedthrough nets and ports. This is needed because any net connecting the top-level I/Os to the top die must cross the bottom die. The net will become a "feedthrough" net in the bottom die.
- Clone multiple fanout ports on the top die. If a net drives multiple macros and you need to do the DRV fixing in the standard cell (bottom) die, you need to make sure all the macro die input ports have only one fanout in the macro die.

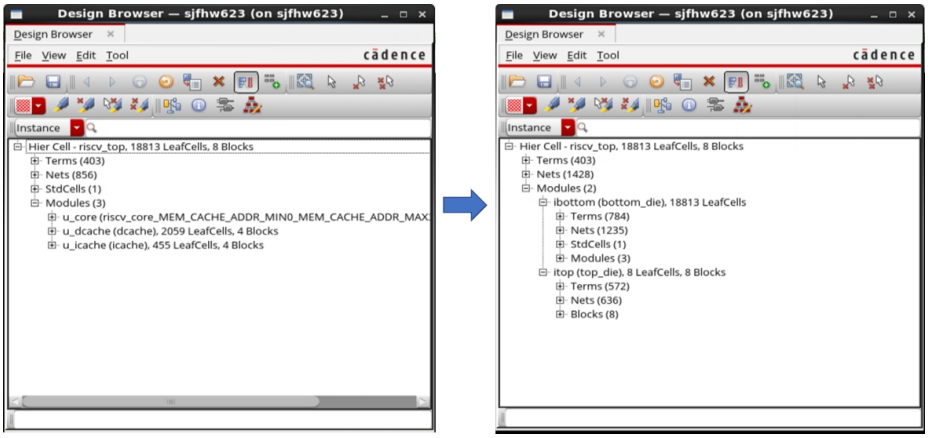
Parameters

`-help`	Prints a brief description for each `proto3d::create_module_wrapper` parameter. *Data_type*: optional
`-bottom_die_cell` *`cell_name`*	Specifies the cell name for the stdcell die. *Data_type*: string, required
`-bottom_die_hinst` *`hinst_name`*	Specifies the hinst name for the stdcell die. *Data_type*: string, required
`-from_die_id`	Specifies that the objects with the same die id will be grouped in the same module wrapper. The instance whose die id is `1` will be put on the bottom die and the instance whose die id is `2` will be put on the top die. *Data_type*: bool, optional
`-skip_feedthrough_insertion`	Turns off feedthrough insertion. *Data_type*: bool, optional
`-skip_port_cloning`	Turns off port cloning. *Data_type*: bool, optional
`-top_die_cell` *`cell_name`*	Specifies the cell name for the macro die. *Data_type*: string, required
`-top_die_hinst` *`hinst_name`*	Specifies the hinst name for the macro die. *Data_type*: string, required
`-user_top_die_instgroup` *`string`*	Specifies the name of an instGroup whose members will be placed on the top die. *Data_type*: string, optional

Example

- The following command will create an hinst `itop` with the cell name `top_die` to include all the macros and an hinst `ibottom` with the cell name `bottom_die` to include all the standard cells.

  ```
  proto3d::create_module_wrapper -top_die_hinst itop -top_die_cell top_die -bottom_die_hinst ibottom -bottom_die_cell
  bottom_die
  ```



proto3d::get_obj_die_id

```
proto3d::get_obj_die_id
[-help]
inst|hInst|pBlkg|group|bndry+
```

Returns the die ID of specified objects.

Parameters

`-help`	Prints a brief description for each `proto3d::get_obj_die_id` parameter. *Data_type*: optional
`inst|hInst|pBlkg|group|bndry+`	Specifies the list of objects for whichi die ID is to be returned. *Data_type*: `(inst|hInst|pBlkg|group|bndry)+`, required

Related Commands

- `proto3d::set_obj_die_id`
- `proto_design`
- `update_proto_seeds`

proto3d::insert_zero_delay_cell

```
proto3d::insert_zero_delay_cell
[-help]
-bottom_die_hinst hinst_name
[-insert_tsv [-logical_change_only ]]
[-insert_tsv [-zdc_tsv_size float]]
[-io_buffer cell_name]
[-loc_file string | -logical_change_only ]
-top_die_hinst hinst_name
[-zdc_size float]
```

Inserts a zero delay cell (ZDC) based on the shadow bump location file and updates MMMC.

Note that the `zdc.lef` and `zdc.lib` files are generated automatically.

The following settings will be set for inter-die net between the shadow bumps:

- Annotate zero delay
- Don't touch
- Skip routing

Parameters

`-help`	Prints a brief description for each `insert_zero_delay_cell` parameter. *Data_type*: optional
`-bottom_die_hinst` *`hinst_name`*	Specifies the stdcell die hinst name. *Data_type*: string, required
`-insert_tsv`	Specifies that TSV cells should be inserted in the design. *Data_type*: bool, optional
`-io_buffer` *`cell_name`*	Specifies the cell name for the io buffer. *Data_type*: string, optional
`-loc_file` *`string`*	Specifies the bump location file name. *Data_type*: string, optional
`-logical_change_only`	Specifies that TSV cells should be inserted in the logical-only mode. *Data_type*: bool, optional
`-top_die_hinst` *`hinst_name`*	Specifies the macro die hinst name. *Data_type*: string, required
`-zdc_size` *`float`*	Specifies the size of the zero delay cell. *Data_type*: float, optional
`-zdc_tsv_size` *`float`*	Specifies the size of the zero delay tsv cell. *Data_type*: float, optional

Example

- The following command inserts ZDC cells in both the `itop` and `ibottom` die hinsts based on the bump location file `shadow_bump_loc.xy`. The size of the ZDC cells is `2um*2um`, and an IO buffer using the cell `BUFX4` will be inserted along the ZDC in the bottom die.
```
proto3d::insert_zero_delay_cell -loc_file shadow_bump_loc.xy -top_die_hinst itop -bottom_die_hinst ibottom -io_buffer BUFX4
-zdc_size 2
```

proto3d::set_obj_die_id

```
proto3d::set_obj_die_id
[-help]
[-all | -objects (inst|hInst|pBlkg|group|bndry)+]
-id integer
```

Sets the die ID of specified objects.

Parameters

`-help`	Prints a brief description for each `proto3d::set_obj_die_id` parameter. *Data_type*: optional
`-all`	Sets the die ID of all the objects in the specified die. *Data_type*: bool, optional
`-id integer`	Specifies the die ID to be set for the specified objects. *Data_type*: uint, required
`-objects (inst|hInst|pBlkg|group|bndry)+`	
	Defines the specified objects on the specified die. *Data_type*: `(inst|hInst|pBlkg|group|bndry)+`, optional

Examples

- The following command sets the die ID for hinst`M1` as `1`:
  `proto3d::set_obj_die_id -objects M1 -id 1`
- The following command sets the die ID for the instance`M2/a1` as `2`:
  `proto3d:set_obj_die_id -objects M2/a1 -id 2`

Related Commands

- `proto3d::get_obj_die_id`
- `proto_design`
- `update_proto_seeds`

proto3d::unfold_floorplan

```
proto3d::unfold_floorplan
[-help]
[-design_stack_type {face2face face2back}]
-flip_die_hinst name
[-gap width]
```

Unfolds the design physically. The `proto3d::unfold_floorplan` command:

- Expands the floorplan by 2x vertically. The command unfolds the top die to the upper half of the new floorplan so that it no longer overlaps the bottom die. The tool will mirror the location and flip MX for each individual instance and ZDC accordingly because the top die is facing down.
- Create a fence for the two top-level hinsts. Each fence covers half the floorplan.
- Create a placement and route blockage band between the two top-level hinsts.

All I/Os stay on the bottom die.

Parameters

`-help`	Prints a brief description for each `proto3d::unfold_floorplan` parameter. *Data_type*: optional
`-design_stack_type {face2face face2back}`	
	Specifies the design stack type. *Default*: `face2face` *Data_type*: enum, optional
`-flip_die_hinst name`	Specifies the flip die name. *Data_type*: string, required
`-gap width`	Specifies the minimum channel width, which is the width of the route and placement blockage band between the two dies. *Default*: 5 or 6 *stdcell_row_height (Depends on the `-rowSiteHeight` constraint in `setFPlanMode`. *Data_type*: float, optional

Example

- The following command unfolds the die named `MACRO_DIE` to the upper half of the new floorplan, so that it no longer overlaps the bottom die:
  ```
  proto3d::unfold_floorplan -flip_die_hinst MACRO_DIE
  ```

Related Command

- `setFPlanMode`

proto_design

```
proto_design
[-help]
[-constraints name]
[-summary file_name]
```

Places proto seeds for 3D-IC scenarios. `proto_design` will assign a die ID for each proto seed, based on seeds placement.

`proto_design` can support pre-defined die IDs and pre-fixed objects. For example, the following commands set the die ID of the seed `u_icache` as `2` and add the `-soft_fixed` constraint to the proto seed:

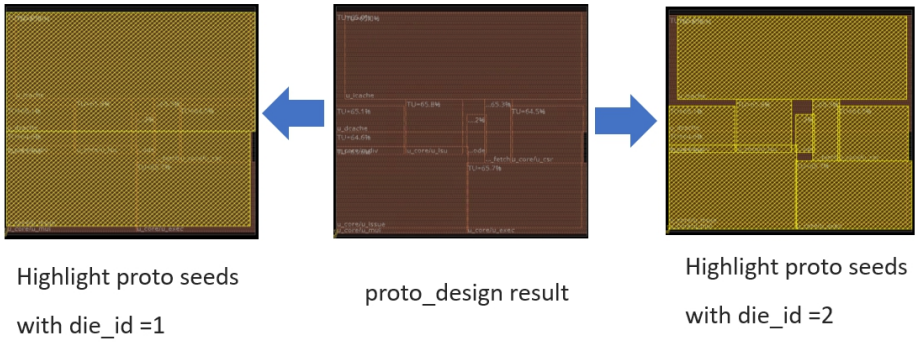
```
update_proto_seeds u_icache -soft_fixed
```

```
proto3d::set_obj_die_id -objects u_icache -id 2
```

`proto_design` will honor the pre-defined die ID and constraint defined above.

Note: The preplaced module's boundary constraint need to be set as `fence`.

The following diagram shows the proto seed placement result.



Parameters

`-help`	Prints a brief description that includes type and default information for each `proto_design` parameter. For a detailed description of the command and all of its parameters, use the man command: `man proto_design`
`-constraints` *`name`*	Specifies the constraint file name (string). Uses the constraints specified in the constraint file for fence creation and module guide generation. A constraint file is a text file that can contain a list of seeds used when generating fences and module guides, and basic relative, spacing and orientation constraints to follow during macro placement. *Data_type*: string, optional
`-summary` *`file_name`*	Specifies the name of the file in which the summary output report with some checkPoints like violation summary, TU of modules, master/clones shape are written. *Data_type*: string, optional

Example

- The following example invokes proto_design with the specified constraint file proto_user_constraint.txt file with all the default settings:

```
set_proto_design_mode -effort high
proto_design -constraints proto_user_constraint.txt
```

Related Commands

- proto3d::set_obj_die_id
- set_proto_design_mode
- update_proto_seeds

pull_from_planner

```
pull_from_planner
[-help]
[-cell dieCellName]
[-through_tag string]
```

Pulls a design from System Planner into the Integrity 3D-IC layout window.

Parameters

`-help`	Prints a brief description for each `pull_from_planner` parameter. *Data_type*: optional
`-cell` *`dieCellName`*	Specifies the cell name of the die to be pulled from System Planner to the layout window. Without this option, all digital dies with tech LEF are pulled from System Planner and layout view enters the multi-die mode. *Data_type*: string, optional
`-through_tag` *`string`*	Specifies the tag name of the module_model. *Data_type*: string, optional

Related Command

- `push_to_planner`

push_to_planner

```
push_to_planner
[-help]
[-cell dieCellName]
[-reset]
[-target_tag tagName]
```

Pushes a design from the Integrity 3D-IC layout window to System Planner.

Parameters

`-help`	Prints a brief description for each `push_to_planner` parameter. *Data_type*: optional
`-cell` *`dieCellName`*	Specifies the cell name of the die to be pushed to System Planner. *Data_type*: string, optional
`-reset`	Resets the System Planner DB. *Data_type*: bool, optional
`-target_tag` *`tagName`*	Specifies the tag name of the System Planner DB to be updated. *Data_type*: string, optional

Related Command

- `pull_from_planner`

set_proto_design_mode

```
set_proto_design_mode
[-help]
[-cover_fixed_macros {true|false}]
[-effort {low|high}]
[-flexmodel_constraint_type {guide|region|fence}]
[-keep_guide {true|false}]
[-number_of_dies integer]
[-place_macro {true|false}]
[-remove_overlap {true|false}]
[-reset]
[-seed_boundary_distance_factor float]
[-z_length_cost float]
```

Controls certain aspects of how the `proto_design` command generates an initial floorplan.

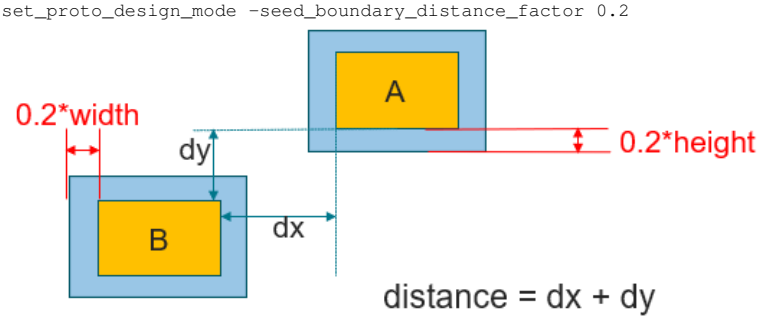
Parameters

`-help`	Prints a brief description for each `set_proto_design_mode` parameter. *Data_type*: optional
`-cover_fixed_macros {true|false}`	
	Forces the flexModel to cover fixed macros belonging to it when using the new planDesign algorithm. *Default*: `false` *Data_type*: bool, optional
`-effort {low|high}`	Specifies the `proto_design` effort. *Default*: `low` *Data_type*: enums, optional
`-flexmodel_constraint_type {guide|region|fence}`	
	Converts the flexModel to the specified constraint type in `proto_design`. *Default*: `region` *Data_type*: optional
`-keep_guide {true|false}`	Retains the guides. *Default*: `true` *Data_type*: bool, optional
`-number_of_dies integer`	Specifies the number of dies. Currently, the tool supports only 2 dies for a 3D-IC design. *Default*: `1` (min=1, max=2) *Data_type*: optional
`-place_macro {true|false}`	

	Controls whether `proto_design` does macro placement. *Default*: `true` *Data_type*: bool, optional
`-remove_overlap {true|false}`	
	Controls whether `proto_design` removes overlaps. *Default*: `true` *Data_type*: bool, optional
`-reset`	Resets all or specified options to the default value. *Data_type*: optional
`-seed_boundary_distance_factor` *`float`*	
	Specifies a scale factor to calculate the net-length for proto seeds. By default, the tool calculates the distance from the center to center of proto seeds. You could specify a factor to calculate the distance from the boundary to boundary of proto seeds. *Default*: 0.5 (min=0, max=0.5)
`-z_length_cost` *`float`*	Specifies the cost for inter-die connection. Increasing the `z_length_cost` will decrease the number of cross-die connections. *Default*: `1` (min=0, max=1000000) *Data_type*: optional

Example

- The following command specifies a scale factor of 0.2 for calculating the net-length for proto seeds:
  `set_proto_design_mode -seed_boundary_distance_factor 0.2`



Related Command

- `proto_design`

update_proto_seeds

```
update_proto_seeds
[-help]
[{objs}]
[-aspect_ratio_range {min max}]
[-blocked_layers string]
[-cluster string | -exclusive_cluster string]
[-delete]
[-honor_master string]
[-ignore_keepout_blockage]
[-independent]
[-orient {R0 R90 R180 R270 MX MX90 MY MY90}]
[-reset string]
[-soft_fixed]
[-spacing float]
[-util float]
```

Updates all or specified proto seeds by specifying constraints, such as utilization, aspect ratio, and soft fixed.

Note: The tool will report an ERROR if the specified seeds are nested.

Parameters

`-help`	Prints a brief description for each `update_proto_seeds` parameter. *Data_type*: optional
`-aspect_ratio_range {min max}`	Specifies the aspect ratio (width/height) in {*min max*} format of specified hinsts. *Default range*: `{0.33 3.0}` *Data_type*: point, optional
`-blocked_layers string`	Specifies the routing layers that are blocked on top of the seeds. *Data_type*: string, optional
`-cluster string`	Specifies the cluster name. *Data_type*: string, optional
`-delete`	Deletes the specified objects that are no longer placement seeds. *Data_type*: bool, optional
`-exclusive_cluster string`	Specifies the exclusive cluster name. This option will be honored by cluster_seeds_into_groups. *Data_type*: string, optional
`-honor_master string`	Specifies which hinst is the object master. This option should only be specified for a clone hinst. *Data_type*: string, optional
`-ignore_keepout_blockage`	Indicates that the specified objects can be placed inside the area of blockages that have `keepout` as a prefix in their names. *Data_type*: bool, optional
`-independent`	Specifies the objects that should not be merged or clustered with other objects. This option will be honored by cluster_seeds_into_groups. *Data_type*: bool, optional

`{objs}`	Specifies the list of object names to be updated. The names are limited to the created seeds. Wild cards are supported. *Default*: All seeds *Data_type*: string, optional
`-orient {R0 R90 R180 R270 MX MX90 MY MY90}`	
	Specifies valid macro or clone orientations. *Default*: `R0` *Data_type*: enum, optional
`-reset` *`string`*	Specifies the constraint that will be reset to the default value.`-reset` must be specified as the first option. *Data_type*: string, optional
`-soft_fixed`	Indicates that the specified objects are preplaced objects. *Data_type*: bool, optional
`-spacing` *`float`*	Specifies the min spacing in um. *Data_type*: float, optional
`-util float`	Specifies the maximum target utilization. Range is `0.0` to `1.0`. *Default*: `0.7` (70%) *Data_type*: float, optional

Example

- The following command deletes all placement seeds:
  `update_proto_seeds -delete`

Related Command

- `create_proto_seeds`

System Planner Tcl Commands

The the `planner` name space commands are listed below.

- align_contacts
- align_device_placement
- align_selected
- assign_bumps
- assign_pg_bumps
- assign_terms
- check_cell_uniqueness
- check_contacts
- check_design
- check_device_grid
- check_io_info
- check_pin_grid
- check_place
- classify_nets
- clear_all_rulers
- clear_selection
- close_design
- commit_connection_optimization
- complete_tcl_cmd
- connect
- connect_contact_pin
- copy_device
- copy_pin
- create_bump
- create_bump_replication
- create_bump_template
- create_contact_layer
- create_contacts
- create_current_region
- create_design
- create_device
- create_device_grid
- create_device_template
- create_diff_pair
- create_external_contact_layer
- create_feedthru

- create_group
- create_layer
- create_marker
- create_modInst
- create_module
- create_module_model
- create_net
- create_package
- create_padstack
- create_pin_grid
- create_pin_matrix_pattern
- create_place_blockage
- create_projective_pins
- create_region
- create_route_blockage
- create_stitch_zone
- create_substrate
- create_term
- create_via
- delete_contact_layer
- delete_contacts
- delete_current_region
- delete_external_contact_layer
- delete_group
- delete_marker
- delete_nets
- delete_obj
- delete_padstack
- delete_region
- delete_stitch_zone
- delete_substrate
- deselect_obj
- edit_bulk_nets
- edit_bulk_terms
- eval_bsh
- get_db
- get_obj_in_area
- get_property
- get_selection

- get_transform_shapes
- group
- gui_canvas
- gui_fit
- gui_hide
- gui_redo
- gui_set_obj_color
- gui_set_visible_layer
- gui_set_visible_obj
- gui_set_visible_substrate
- gui_show
- gui_undo
- gui_zoom
- is_top_design
- list_select_properties
- man
- move_device
- move_pin
- place_inst
- place_io_info
- read_allegro
- read_bga_text
- read_bump_locations
- read_def
- read_design_stack
- read_device_placement
- read_die_text
- read_io_info
- read_lef
- read_module_model
- read_net_interface
- read_net_list
- read_net_map
- read_oio
- read_pin_label
- read_pin_physical_list
- read_power_intent
- read_property
- read_systemlvs_cdl_port_config

- read_term_list
- read_term_map
- read_verilog
- refresh_canvas
- refresh_pin_map
- remove_connections
- rename_matrix_objs
- run_lec
- run_systemlvs
- run_thermal
- select_obj
- set_connection_optimization
- set_db
- set_era_die_config
- set_era_pg_net_config
- set_era_result_user_preference
- set_era_user_preference
- set_hierarchy
- set_lec_library
- set_lec_netlist
- set_lec_user_preference
- set_message
- set_module_model
- set_preference
- set_property
- set_systemlvs_design_check
- set_systemlvs_interface_check
- set_systemlvs_netlist_usage
- set_thermal_ambient_condition
- set_thermal_contact_attribute
- set_thermal_heat_sink
- set_thermal_layer_material
- set_thermal_molding_compound
- set_thermal_package
- set_thermal_power
- set_thermal_tech
- set_thermal_test_board
- set_thermal_user_preference
- set_vsrc_layers

- show_connections
- snap_cell
- source_bsh
- space_selected
- swap_pin_nets
- synthesize_boundary
- test_same_oio
- test_same_text
- thermal::view_result
- unassign_bumps
- ungroup
- ungroup_device
- uniquify_cell
- unplace_obj
- update_contact_layer
- update_current_region
- update_device
- update_device_template
- update_padstack
- update_shrink_factor
- update_stitch_zone
- update_substrate
- view_lec_result
- view_systemlvs_result
- write_allegro
- write_bga_text
- write_bump_locations
- write_contact_pin_map_file
- write_def
- write_design_stack
- write_device_placement
- write_die_text
- write_edtext_file
- write_oio
- write_pin_label
- write_pin_physical_list
- write_ploc_file
- write_power_intent
- write_term_list

- write_term_map
- write_to_gif
- write_verilog

align_contacts

```
align_contacts
[-help]
[-add_missing_pins]
[-align_nets]
[-contact_layer cl]
[-delete_extra_pins]
[-fix_offset]
[-fixed_device device]
[-fixed_layer layer]
[-ground_pin ground_pin]
[-nc_pin nc_pin]
[-power_pin power_pin]
[-reverse]
[-signal_pin signal_pin]
[-sync_device device]
[-sync_layer layer]
```

Aligns the location of specified contacts.

Parameters

`-help`	Prints a brief description of each `align_contacts` parameter. *Data_type*: optional
`-add_missing_pins`	Adds contact pins for the floating pins based on the existing physical locations. *Data_type*: optional
`-align_nets`	Finds contact pins with inconsistent net connection based on the existing physical locations and reconnects the nets. *Data_type*: optional
`-contact_layer cl`	Specifies the contact layer object instead of the fixed-sync config. *Data_type*: optional
`-delete_extra_pins`	Deletes pins that do not have a contact pin on the basis of the existing physical locations. *Data_type*: optional
`-fix_offset`	Aligns the center of the contact pins based on the existing physical locations. *Data_type*: optional

`-fixed_device` *`device`*	Specifies the device on which the fixed contact layer is located. *Data_type*: optional
`-fixed_layer` *`layer`*	Specifies the fixed contact layer. *Data_type*: optional
`-ground_pin` *`ground_pin`*	
	Specifies the contact pin matched for ground pins when `-add_missing_pins` is specified. *Data_type*: optional
`-nc_pin` *`nc_pin`*	Specifies the contact pin matched for NC pins when `-add_missing_pins` is specified. *Data_type*: optional
`-power_pin` *`power_pin`*	Specifies the contact pin matched for power pins when `-add_missing_pins` is specified. *Data_type*: optional
`-reverse`	Specifies the reverse sync order for a specific contact layer object. *Data_type*: optional
`-signal_pin` *`signal_pin`*	
	Specifies the contact pin matched for signal pins when `-add_missing_pins` is specified. *Data_type*: optional
`-sync_device` *`device`*	Specifies the device on which the contact layer is to be synced. *Data_type*: optional
`-sync_layer` *`layer`*	Specifies the layer to which the contact layer is to be synced. *Data_type*: optional

Example

- The following command finds contact pins and aligns their location:

```
align_contacts -fixed_device /daughter_die \
               -fixed_layer AP \
               -sync_device /Interposer \
               -sync_layer AP
```

align_device_placement

```
align_device_placement
[-help]
-refer referred_device
-selected selected
-type {x_center | y_center | left | right | top | bottom}
```

Aligns the selected devices as specified with respect to the referred device

Parameters

`-help`	Prints a brief description of each `align_device_placement` parameter. *Data_type*: optional
`-refer referred_device`	
	Specifies the referred devices. *Data_type*: required
`-selected selected`	
	Specifies the selected devices to be aligned. *Data_type*: required
`-type {x_center | y_center | left | right | top | bottom}`	
	Specifies the alignment type. *Data_type*: enum, required

Example

- The following command aligns the `/top/SoC` device by `y_center` to the referred devices:

```
align_device_placement -refer {/top/HBM1 /top/HBM2 /top/HBM3 /top/HBM4}
                       -selected {/top/SoC}
                       -type y_center
```

Related Commands

- `read_device_placement`
- `write_device_placement`

align_selected

```
align_selected
[-help]
[-refer_to_first]
-side {center | middle | left | right | top | bottom}
```

Aligns the selected objects as specified. Use this command after selecting two or more objects.

Parameters

`-help`	Prints a brief description of each `align_selected` parameter. *Data_type*: optional
`-refer_to_first`	
	Specifies that the first selected object is the reference object. *Data_type*: optional
`-side {center | middle | left | right | top | bottom}`	
	Specifies the vertical and horizontal alignment for the selected objects. Choose one of the following: • `right`: Specifies vertical right alignment. • `left`: Specifies vertical left alignment. • `center`: Specifies vertical center alignment. • `top`: Specifies horizontal top alignment. • `bottom`: Specifies horizontal bottom alignment. • `middle`: Specifies horizontal middle alignment. *Data_type*: enum, required

Example

- The following command aligns the selected objects vertically to the right:
  `align_selected -side right`
- The following command aligns all selected objects horizontally such that their bottom edges are aligned with the bottom of the first selected object:
  `align_selected -side bottom -refer_to_first`

Related Commands

- `clear_selection`
- `deselect_obj`
- `get_selection`
- `list_select_properties`
- `select_obj`
- `space_selected`

assign_bumps

```
assign_bumps
[-help]
[-assign_start_point {SW | NW | NE | SE}]
[-assign_style {horizontal | vertical}]
-selected
-terms term_list
```

Assigns bumps.

Parameters

`-help`	Prints a brief description for each `assign_bumps` parameter. *Data_type*: optional
`-assign_start_point {SW | NW | NE | SE}`	
	Specifies the corner from which assignment starts. *Data_type*: enum, optional
`-assign_style {horizontal | vertical}`	
	Specifies the assign pattern. You can specify `horizontal` or `vertical` as the assignment pattern. *Data_type*: enum, optional
`-selected`	Specifies only selected pins. *Data_type*: required
`-terms term_list`	Specifies the term names on substrate instance. *Data_type*: required

Example

- The following set of commands assigns bumps from selected pins:
```
select_obj -type pin -rect {1695.451851 2338.115741 2041.271541 2687.157354}
assign_bumps -selected -assign_style vertical -terms {yy_hidi* yy_clock*}
```

Related Command

- `assign_pg_bumps`
- `unassign_bumps`

assign_pg_bumps

```
assign_pg_bumps
[-help]
-assign_style {horizontal | vertical | checker_board}
-device_path path
[-ground_nets ground]
-net_pattern pattern
[-power_nets power]
[-selected]
```

Assigns power and ground bumps.

Parameters

`-help`	Prints a brief description for each `assign_pg_bumps` parameter. *Data_type*: optional
`-assign_style {horizontal | vertical | checker_board}`	
	Specifies the style of bump assignment. You can specify `horizontal`, `vertical`, or `checker_board` as the assignment style. *Data_type*: enum, required
`-device_path path`	Specifies the target device path. *Data_type*: required
`-ground_nets ground`	Specifies the list of ground nets, separated by commas. *Data_type*: optional
`-net_pattern pattern`	Specifies the pattern. *Data_type*: required
`-power_nets power`	Specifies the list of power nets, separated by commas. *Data_type*: optional
`-selected`	Specifies only selected pins. *Data_type*: optional

Examples

- The following set of commands assigns PG bumps from selected pins.

```
select_obj -type pin -all
assign_pg_bumps -assign_style checker_board \
                -selected \
                -device_path /MyPackage \
                -net_pattern vss,vdd \
                -power_nets vdd \
                -ground_nets vss
```

- The following command assigns PG bumps from unconnected pins.

```
assign_pg_bumps -assign_style checker_board \
                -device_path /MyPackage \
                -net_pattern {vss,vdd}
```

Related Commands

- `assign_bumps`
- `unassign_bumps`

assign_terms

```
assign_terms
[-help]
[-only_contacts]
[-only_floating]
```

Assigns terms as specified. By default, the tool assigns the contact pins first and then assign the floating terms.

Parameters

`-help`	Prints a brief description for each `assign_terms` parameter. *Data_type*: optional
`-only_contacts`	Assigns terms based only on the contact and netlist of pins. *Data_type*: optional
`-only_floating`	Assigns only floating terms to floating pins. *Data_type*: optional

Example

- The following command assigns only floating terms to floating pins.
  ```
  assign_terms -only_floating
  ```

check_cell_uniqueness

```
check_cell_uniqueness
[-help]
[-cell cell]
[-out_file file_name]
-pin_grid grid
[-selected]
```

Checks cell uniqueness and whether the selected devices are able to share a cell. Checks if the target devices (set by using `-selected`) have the same offset from *`pin_grid`* (set with `-pin_grid`). The devices with the same group name are able to use the same template for place and route (PnR). The group name shows up in the log or report file. Devices with the same group name means they have the same offset from *`pin_grid`* (considering rotation and mirroring).

Parameters

`-help`	Prints a brief description for each `check_cell_uniqueness` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell. All instances of the specified cell are checked. *Data_type*: optional
`-out_file` *`file_name`*	Specifies the location of the report file. *Data_type*: file, optional
`-pin_grid` *`grid`*	Specifies the name of the pin grid to be checked. *Data_type*: required
`-selected`	Specifies selected objects. *Data_type*: optional

Examples

- The following command checks the uniqueness of selected devices with the pin grid named `pin_grid_9x9`:

```
# When there is no need to uniquify or snap, refDevPath is an empty string.
# When uniquify or snap is necessary, refDevPath is the recommended device path to be
# used as specified by '-refer' of the 'snap_cell' command.
set refDevPath [check_cell_uniqueness -selected -pin_grid pin_grid_9x9]
```

- The following command checks the uniqueness of all instances of cell `N5:FCCC_N5` with the pin grid named `pin_grid_9x9`:

```
# When there is no need to uniquify or snap, refDevPath is an empty string.
# When uniquify or snap is necessary, refDevPath is the recommended device path to be
# used as specified by '-refer' of the 'snap_cell' command.
set refDevPath [check_cell_uniqueness -cell N5:FCCC_N5 -pin_grid pin_grid_9x9]
```

Related Commands

- `snap_cell`
- `uniquify_cell`

check_contacts

```
check_contacts
[-help]
[-check_net_mapping {yes | no}]
[-check_pin_contact {yes | no}]
[-check_pin_count_mismatch {yes | no}]
[-check_pin_offset {yes | no}]
[-check_term_direction {yes | no}]
[-contact_overlap_percentage overlap_percentage]
[-pin_offset pin_offset]
[-show_result_gui]
[-interface_name interface_name
 |{-from_device device -from_layer layer -to_device device -to_layer layer}]
[-export_csv export_csv [-prefix name]]
```

Checks contacts as specified.

Parameters

`-help`	Prints a brief description for each `check_contacts` parameter. *Data_type*: optional
`-check_net_mapping {yes | no}`	
	Checks whether the physical connectivity of the net matches its logic connectivity, when set to `yes`. *Data_type*: optional
`-check_pin_contact {yes | no}`	
	Checks whether the port on the contact layer has a contact pad, when set to `yes`. *Data_type*: optional
`-check_pin_count_mismatch {yes | no}`	
	Checks whether a pin has only one contact pin, when set to `yes`. *Data_type*: optional
`-check_pin_offset {yes | no}`	
	Checks the offset of the contact pin based on the center of the pad, when set to `yes`. *Data_type*: optional
`-check_term_direction {yes | no}`	
	Checks the direction of two contact pins, when set to `yes`. *Data_type*: optional
`-contact_overlap_percentage` *`overlap_percentage`*	
	Specifies the minimum rectangular bounds overlap percentage for identifying contact ports. *Default*: `0` *Data_type*: optional
`-export_csv` *`export_csv`*	
	Specifies the export file directory of the report. *Data_type*: file, optional
`-from_device` *`device`*	Specifies the device from which the contact layer is created. *Data_type*: file, optional
`-from_layer` *`layer`*	Specifies the layer from which the contact layer is created. *Data_type*: optional
`-interface_name` *`interface_name`*	
	Specifies the contact layer by given name. *Data_type*: optional
`-pin_offset` *`pin_offset`*	

	Specifies the maximum offset between pins in user unit. *Data_type*: optional
`-prefix` *name*	Specifies the prefix name of the exported csv files. This parameter can be specified only if `-export_csv` has been specified. *Default*: empty string *Data_type*: optional
`-show_result_gui`	Displays the checking result UI. *Data_type*: optional
`-to_device` *device*	Specifies the device to which the contact layer is created. *Data_type*: optional
`-to_layer` *layer*	Specifies the layer to which the contact layer is created. *Data_type*: optional

Example

- The following set of commands sets the output directory, checks contacts as specified, and prints each contact layer in the format `{{from_device from_layer to_device to_layer}*}` using `get_db`:

```
set testDir "./test_top1_interposer_output/";
check_contacts \
    -from_device /UserGroup/FCCC_TOP_1 \
    -from_layer MB \
    -to_device /UserGroup/FCCC_Interposer \
    -to_layer UBM \
    -check_pin_contact yes \
    -check_pin_offset yes \
    -pin_offset 0.01 \
    -check_pin_count_mismatch no \
    -check_net_mapping no \
    -check_term_direction no \
    -export_csv $testDir

# Combine the get_db command to output each contact layer format: {{from_device from_layer to_device to_layer}*}
get_db current_design .contact_layers
```

  The output is as follows:
  `{{/ASIC_IPG AP /INTERPOSER AP}{/JESD2358B-HBM_Ballout Bump /INTERPOSER AP}}`

Related Commands

- `create_contact_layer`
- `create_contacts`
- `delete_contacts`

check_design

```
check_design
[-help]
```

Checks the database definition and fixes illegal definitions.

Parameters

`-help`	Prints a brief description for each `check_design` parameter. *Data_type*: optional

Related Commands

- `close_design`
- `create_design`
- `is_top_design`

check_device_grid

```
check_device_grid
[-help]
[-cell cell]
-grid grid
[-out_file file_name]
[-selected]
```

Checks the specified device grid. If youselect a device before running the command, it checks whether the selected device is on the specified grid.

Parameters

`-help`	Prints a brief description for each `check_device_grid` parameter. *Data_type*: optional
`-cell cell`	Specifies the device cell. All instances of this cell will be checked. *Data_type*: optional
`-grid grid`	Specifies the name of the device grid to be checked. *Data_type*: required
`-out_file file_name`	Specifies the location of the report file. *Data_type*: file, optional
`-selected`	Specifies selected objects. *Data_type*: optional

Examples

- The following command checks whether the selected devices are on the specified device grid:
  `check_device_grid -selected -grid device_grid_51x51 -out_file {/path/to/report}`
- The following command checks all instances of the cell`N5:FCCC_N5_PB` on the specified device grid:
  `check_device_grid -cell N5:FCCC_N5_PB -grid device_grid_51x51 -out_file {/path/to/report}`

Related Command

- `create_device_grid`

check_io_info

```
check_io_info
[-help]
-device_path device_path
[-exception_file exception_file]
file
[-out_file out_file]
-type {order}
```

Checks the order based on the IO info file.

Parameters

`-help`	Prints a brief description for each `check_io_info` parameter. *Data_type*: optional
`-device_path` *`device_path`*	
	Specifies the target device path. *Data_type*: instance path, required
`-exception_file` *`exception_file`*	
	Specifies the exception file. *Data_type*: file, optional
`file`	Specifies the IO info file path. *Data_type*: file, required
`-out_file` *`out_file`*	Specifies the report file path. *Data_type*: file, optional
`-type {order}`	Specifies the check type. *Data_type*: enum, required

Example

- The following command checks the order based on the specified IO info file:
  `check_io_info -device_path /DIE1 -file {IOinfo_v1_2020_08_18 _pg.csv} -type order -exception_file excption.txt -out_file test.txt`

Related Commands

- `place_io_info`
- `read_io_info`

check_pin_grid

```
check_pin_grid
[-help]
[-cell cell]
-grid grid
[-out_file file_name]
[-selected]
```

Checks the specified pin grid. If youselect pin/bumps before running the command, it checks whether the selected objects are on the specified grid.

Parameters

`-help`	Prints a brief description for each `check_pin_grid` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the bump cell. All bump instances of this cell will be checked. *Data_type*: optional
`-grid` *`grid`*	Specifies the name of the pin grid to be checked. *Data_type*: required
`-out_file` *`file_name`*	Specifies the location of complete violation report file. The entire violation log is not printed out by default to avoid performance overhead in large designs. Use the `out_file` option to view the complete report. *Data_type*: file, optional
`-selected`	Specifies selected objects. *Data_type*: optional

Examples

- The following command checks whether the selected pins and bumps are on the specified pin grid:
  ```
  check_pin_grid -selected -grid pin_grid_8x8 -out_file {/path/to/report}
  ```
- The following command checks all instances of the bump cell`bottom_die:TSVBUMP` on the specified pin grid:
  ```
  check_pin_grid -cell bottom_die:TSVBUMP -grid pin_grid_8x8 -out_file {/path/to/report}
  ```

Related Command

- `create_pin_grid`

check_place

```
check_place
[-help]
```

Checks the placement rule and generates DRC markers.

Parameters

`-help`	Prints a brief description for each `check_place` parameter. *Data_type*: optional

classify_nets

```
classify_nets
[-help]
[-module module]
-nets nets
-type {signal | power | ground}
```

Classifies nets as signal, power, or ground nets.

Parameters

`-help`	Prints a brief description for each `classify_nets` parameter. *Data_type*: optional
`-module` *`module`*	Classifies the nets in the specified module. *Data_type*: module, optional
`-nets` *`nets`*	Lists the names of the nets that you want to classify. You can use a wildcard (*) to specify all nets. You must use braces to enclose the list of nets. *Data_type*: required
`-type {signal | power | ground}`	
	Specifies the type of the net as `signal`, `power`, or `ground`. *Data_type*: enum, required

Example

- The following command specifies that the `VCC` net and the nets prefixed with `VD` for `INTERPOSER` devices are of type `power`.

  ```
  classify_nets -nets {VCC VD*} -module INTERPOSER -type power
  ```

clear_all_rulers

```
clear_all_rulers
[-help]
```

Clears all rulers in the current canvas view.

Parameters

`-help`	Prints a brief description for each `clear_all_rulers` parameter. *Data_type*: optional

Related Commands

- `gui_hide`
- `gui_fit`
- `gui_redo`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `gui_show`
- `gui_undo`
- `gui_zoom`
- `refresh_canvas`

clear_selection

```
clear_selection
[-help]
```

Clears the current selection.

Parameters

`-help`	Prints a brief description for each `clear_selection` parameter. *Data_type*: optional

Related Commands

- `deselect_obj`
- `get_selection`
- `list_select_properties`
- `select_obj`

close_design

```
close_design
[-help]
```

Closes the design and releases the database.

Parameters

`-help`	Prints a brief description for each `close_design` parameter. *Data_type*: optional

Related Commands

- `check_design`
- `create_design`
- `is_top_design`

commit_connection_optimization

```
commit_connection_optimization
[-help]
[-swap_devices]
```

Optimizes the specified connection.

Parameters

`-help`	Prints a brief description for each `commit_connection_optimization` parameter. *Data_type*: optional
`-swap_devices`	Specifies optimization by swapping devices. *Data_type*: optional

Examples

- The following set of commands sets the first selected pin as the source of the fixed sided, then sets the second selected pin as the source of the free side, and finally commits the connection:

```
select_obj -type pin -area_operation contain -rect {1777 1058 1828 1275}
set_connection_optimization -fixed_type selection

clear_selection
select_obj -type pin -area_operation contain -rect {2368 1126 2474 1299}
set_connection_optimization -free_type selection

commit_connection_optimization -swap_devices
```

- The following set of commands sets first specified floorplan as source of the fixed side, then sets the second specified floorplan as the source of the free side, and finally commits the connection:

```
set_connection_optimization -fixed_type floorplan -fixed {/BGA1/DIE1 [null]:DDR0_data3:FC_DESIGN-02_1:DIE5}
set_connection_optimization -free_type floorplan -free {/BGA1 [null]:DDR0_data3:FC_DESIGN-02_1:18X18_BGA}

commit_connection_optimization
```

Related Commands

- set_connection_optimization

complete_tcl_cmd

```
complete_tcl_cmd
[-help]
cmd
[-cursor cursor]
```

Returns all the candidates in a string for the working command.

Parameters

`-help`	Prints a brief description for each `complete_tcl_cmd` parameter. *Data_type*: optional
`cmd`	Specifies the working command. *Data_type*: required
`-cursor` *`cursor`*	Specifies the working cursor position. Default is the length of the command. *Data_type*: optional

Example

- `complete_tcl_cmd {get_db c .name} -cursor [string length {get_db c}]`
  `==> {current_design cmd cmd_file}`

connect

```
connect
[-help]
-cell cell
-net net
-pin pin
```

Connects a specific net to a specific pin.

Parameters

`-help`	Prints a brief description for each `connect` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell. *Data_type*: required
`-net` *`net`*	Specifies the net name. *Data_type*: required
`-pin` *`pin`*	Specifies the pin name. *Data_type*: required

Examples

- The following command connects the `VSS_1` pin of the `INTERPOSER:INTERPOSER` cell to the `VSS` net:

  ```
  connect -cell INTERPOSER:INTERPOSER -net VSS -pin VSS_1
  ```

- The following command connects the `A1` pin of the `MyPackage` cell to the `VDD` net:

  ```
  connect -cell MyPackage -net VDD -pin A1
  ```

connect_contact_pin

```
connect_contact_pin
[-help]
[contact_layer]
[-from_device device]
[-from_layer layer]
[-to_device device]
[-to_layer layer]
```

Connects contact pins on all or specified layers.

Parameters

`-help`	Prints a brief description for each `connect_contact_pin` parameter. *Data_type*: optional
`contact_layer`	Specifies the contact layer object instead of using the from-to config. *Data_type*: optional
`-from_device` *`device`*	Specifies the device from which the contact layer is created. *Data_type*: optional
`-from_layer` *`layer`*	Specifies the layer from which the contact layer is created. *Data_type*: optional
`-to_device` *`device`*	Specifies the device to which the contact layer is created. *Data_type*: optional
`-to_layer` *`layer`*	Specifies the layer to which the contact layer is created. *Data_type*: optional

Examples

- The following command specifies the contact layer by using the from-to config:
```
connect_contact_pin -from_device /Chip1 -from_layer AP -to_device /INTERPOSER -to_layer AP
```
- The following command auto-connect nets for the all the contact layers in the db:
```
connect_contact_pin
```
- The following command specifies the contact layer by passing a ContactLayer object:
```
set ct [create_contact_layer -from_device /Chip1 -from_layer AP -to_device /INTERPOSER -to_layer AP]
connect_contact_pin $ct
```

copy_device

```
copy_device
[-help]
[-align_contacts]
[-direction {up | down | left | right}]
[-distance distance]
[-offset offset]
[-point point]
-selected
```

Copies the selected device as specified.

Parameters

`-help`	Prints a brief description of each `copy_device` parameter. *Data_type*: optional
`-align_contacts`	Aligns the contact objects. *Data_type*: optional
`-direction {up | down | left | right}`	
	Specifies the vertical and horizontal movement for the objects being copied. *Data_type*: enum, optional
`-distance` *`distance`*	Specifies the distance, in microns, that the objects will be copied in the specified direction. *Data_type*: optional
`-offset` *`offset`*	Specifies the offset points by which the objects will be copied. *Data_type*: optional
`-point` *`point`*	Specifies the point to which the objects will be copied. *Data_type*: optional
`-selected`	Copies the selected devices. *Data_type*: required

Examples

- The following command makes a copy of the selected device and moves the copy by the specified offset points:
  `copy_device -selected -offset {50.0 30.0}`
- The following command copies the selected device to the specified point:
  `copy_device -selected -point {0.0 0.0}`
- The following command copies the selected device and pastes it to the right by 50 microns:
  `copy_device -selected -distance 50.0 -direction right`

Related Commands

- `create_device`
- `move_device`
- `move_pin`
- `update_device`

copy_pin

```
copy_pin
[-help]
[-align_contacts]
[-direction {up | down | left | right}]
[-distance distance]
[-offset offset]
[-point point]
-selected
```

Moves the selected pin as specified.

Parameters

`-help`	Prints a brief description of each `copy_pin` parameter. *Data_type*: optional
`-align_contacts`	Aligns the contact pins. *Data_type*: optional
`-direction {up | down | left | right}`	
	Specifies the direction of movement for the pins. *Data_type*: enum, optional
`-distance distance`	
	Specifies the distance, in microns, that the pins will be copied in the specified direction. *Data_type*: optional
`-offset offset`	Specifies the offset points by which the pins will be copied. *Data_type*: optional
`-point point`	Specifies the point to which the pins will be copied. *Data_type*: optional
`-selected`	Copies the selected pins. *Data_type*: required

Examples

- The following command makes a copy of the selected pin and moves the copy by the specified offset points:
  `copy_pin -selected -offset {50.0 30.0}`
- The following command copies the selected pin to the specified point:
  `copy_pin -selected -point {0.0 0.0}`
- The following command makes a copy of the selected pin and moves the copy to the right by 50 microns:
  `copy_pin -selected -distance 50.0 -direction right`

Related Commands

- `move_device`
- `move_pin`

create_bump

```
create_bump
[-help]
[-algo_pitch_x x]
[-algo_pitch_y y]
[-alignment {center | origin}]
[-all_instances]
[-all_unassign]
[-auto_net_assign]
[-bump_term keystr]
-bump_type {device | pin}
[-fanout_algo {polar | rectangular}]
[-grid grid]
[-min_pitch min_pitch]
[-min_pitch_x min_pitch_x]
[-min_pitch_y min_pitch_y]
[-mirror]
[-name_prefix prefix]
[-offset x_y]
[-orient {r0 | r90 | r180 | r270}]
[-reference_point point]
[-shape_filter_operation {lesseq | greatereq}]
[-shape_filter_size width_height]
-source_layer layer_key
[-stitch_zone stitch_zone]
-template keystr
[-term_assign {none | param | keep | bidir}]
[-term_list term_list]
[-term_list_file term_list_file]
[-term_prefix term_prefix]
```

Creates bumps based on specified options. This command isused for inter-substrate projection.

Parameters

`-help`	Prints a brief description for each `create_bump` parameter. *Data_type*: optional
`-algo_pitch_x x`	Specifies pitch x for fanout algorithm. *Data_type*: optional
`-algo_pitch_y y`	Specifies pitch y for fanout algorithm. *Data_type*: optional
`-alignment {center | origin}`	
	Specifies the position by which bumps will align. *Data_type*: enum, optional
`-all_instances`	Includes all instances. *Data_type*: optional
`-all_unassign`	Includes unassigned pins. *Data_type*: optional

`-auto_net_assign`	Autogenerates net for unassigned pins. *Data_type*: optional
`-bump_term` *`keystr`*	Specifies the term of bump for mapping. *Data_type*: optional
`-bump_type {device | pin}`	
	Specifies the type of bump to be created. *Data_type*: enum, required
`-fanout_algo {polar | rectangular}`	
	Specifies the fanout algorithm. *Data_type*: optional
`-grid` *`grid`*	Specifies the name of the pin grid. New bumps are snapped to the closest grid point. *Data_type*: optional
`-min_pitch` *`min_pitch`*	
	Specifies the minimum spacing between two bumps. *Data_type*: optional
`-min_pitch_x` *`min_pitch_x`*	
	Specifies the minimum x-spacing between two bumps. *Data_type*: optional
`-min_pitch_y` *`min_pitch_y`*	
	Specifies the minimum y-spacing between two bumps. *Data_type*: optional
`-mirror`	Specifies that the bump should be mirrored about its Y-axis. *Data_type*: optional
`-name_prefix` *`prefix`*	Specifies the prefix of generated names. *Data_type*: optional
`-offset` *`x_y`*	Specifies the offset for bump placement. *Data_type*: optional
`-orient {r0 | r90 | r180 | r270}`	
	Specifies the orientation of the bump. *Data_type*: enum, optional
`-reference_point` *`point`*	
	Specifies the coordinates for fanout algorithm. *Data_type*: optional
`-shape_filter_operation {lesseq | greatereq}`	

	Specifies the operation of the shape. *Data_type*: enum, optional
`-shape_filter_size` *`width_height`*	
	Specifies the threshold for shape width and height. *Data_type*: optional
`-source_layer` *`layer_key`*	
	Specifies the source layer of pins. *Data_type*: required
`-stitch_zone` *`stitch_zone`*	
	Specifies the stitch zone. *Data_type*: optional
`-template` *`keystr`*	Specifies the template of the bump. *Data_type*: required
`-term_assign {none | param | keep | bidir}`	
	Specifies the term assignment. *Data_type*: enum, optional
`-term_list` *`term_list`*	
	Specifies the term list for the bumps. *Data_type*: optional
`-term_list_file` *`term_list_file`*	
	Specifies the term list CSV file to generate the terms for the bumps. *Data_type*: optional
`-term_prefix` *`term_prefix`*	
	Specifies the term name prefix for the bumps. *Data_type*: optional

Examples

- The following command creates basic bumps:

```
create_bump -source_layer INTERPOSER:AP \
                      -shape_filter_operation greatereq \
                      -shape_filter_size {0 0} \
                     -bump_type device \
                     -template INTERPOSER:C4_BUMPCELL \
                     -bump_term PAD \
                     -name_prefix BUMP \
                     -alignment center \
                     -min_pitch_x 150 \
                     -min_pitch_y 150 \
```

```
                    -fanout_algo rectangular \
                    -algo_pitch_x 100 -algo_pitch_y 100
```

- The following set of commands specify a term list:

```
# select referred pins with specific term name from substrate instances by -prop_values
select_obj -type pin -all -select_type replace \
            -prop_name term -prop_values {INTERPOSER/C4S_DA5 INTERPOSER/C4S_PAD_TP2 ...} \
            -prop_operation equals -area_operation contain


create_bump -source_layer INTERPOSER:AP \
            -shape_filter_operation greatereq \
            -shape_filter_size  {0 0} \
            -bump_type device \
            -template INTERPOSER:C4_BUMPCELL_105x82 \
            -bump_term {PAD} \
            -name_prefix {C4S_} \
            -alignment center \
            -min_pitch_x $BUMP_PITCH_X \
            -min_pitch_y $BUMP_PITCH_Y \
            -fanout_algo rectangular \
            -algo_pitch_x 0 \
            -algo_pitch_y 0 \
            -term_list {INTERPOSER/C4S_DA5 INTERPOSER/C4S_PAD_TP2 ...} # create pin labels by these terms of substrate
instance.
```

- The following command specifies a stich zone:

```
create_bump -source_layer INTERPOSER:AP \
            -shape_filter_operation greatereq \
            -shape_filter_size {0 0} \
            -bump_type device \
            -template INTERPOSER:C4_BUMPCELL \
            -bump_term PAD \
            -name_prefix BUMP \
            -alignment center \
            -min_pitch_x 150 \
            -min_pitch_y 150 \
            -fanout_algo rectangular \
            -algo_pitch_x 0 \
            -algo_pitch_y 0 \
            -offset {0 0} \
            -stitch_zone C4Bump
```

- The following command specifies a pin grid:

```
create_bump -source_layer INTERPOSER:AP \
            -shape_filter_operation greatereq \
            -shape_filter_size {0 0} \
            -bump_type device \
            -template INTERPOSER:C4_BUMPCELL \
            -bump_term PAD \
            -name_prefix BUMP \
            -alignment center \
            -min_pitch_x 150 \
            -min_pitch_y 150 \
```

```
-fanout_algo rectangular \
-algo_pitch_x 0 \
-algo_pitch_y 0 \
-grid pin_grid_9x9
```

Related Commands

- create_projective_pins
- create_pin_matrix_pattern

create_bump_replication

```
create_bump_replication
[-help]
-from from
-selected
[-term_list_file term_list_file | -term_list term_list]
-to to
```

Creates bump replications on selected bump devices.

Parameters

`-help`	Prints a brief description for each `create_bump_replication` parameter. *Data_type*: optional
`-from` *`from`*	Specifies the from path as reference. *Data_type*: instance path, required
`-selected`	Specifies that bump replications are to be created only for the selected pins. *Data_type*: required
`-term_list` *`term_list`*	Specifies the term list for the bumps. Wildcards are supported. *Data_type*: optional
`-term_list_file` *`term_list_file`*	Specifies the term list CSV file to generate. *Data_type*: file, optional
`-to` *`to`*	Specifies the to path as placed region. *Data_type*: instance path, required

Examples

- The following set of commands create bump replications from a term list file:

```
select_obj -rect {5769.633674 6541.38499 9507.100374 13398.5553} -type device -select_type replace
           -prop_name {Device-template type} -prop_values BUMP -prop_operation equals
           -area_operation contain -modifier <NONE>

create_bump_replication -selected \
                        -from /top/JESD235B-HBM_Ballout \
                        -to /top/JESD235B-HBM_Ballout.0 \
                        -term_list_file term_list_Interposer_C4.csv
```

- The following set of commands create bump replications from a term list, without using wildcards:

```
select_obj -rect {5769.633674 6541.38499 9507.100374 13398.5553} -type device -select_type replace
           -prop_name {Device-template type} -prop_values BUMP -prop_operation equals
           -area_operation contain -modifier <NONE>

create_bump_replication -selected \
                        -from /top/JESD235B-HBM_Ballout \
                        -to /top/JESD235B-HBM_Ballout.0 \
                        -term_list {INTERPOSER/C4S_DA0 INTERPOSER/C4S_DA1 INTERPOSER/C4S_DA10 INTERPOSER/C4S_DA11
INTERPOSER/C4S_DA12}
```

- The following set of commands create bump replications from a term list, using wildcards:

```
select_obj -rect {5769.633674 6541.38499 9507.100374 13398.5553} -type device -select_type replace
           -prop_name {Device-template type} -prop_values BUMP -prop_operation equals
           -area_operation contain -modifier <NONE>

create_bump_replication -selected \
                        -from /top/JESD235B-HBM_Ballout \
                        -to /top/JESD235B-HBM_Ballout.0 \
                        -term_list {INTERPOSER/C4S*}
```

create_bump_template

```
create_bump_template
[-help]
-name name
[-origin {lower_left | center | origin}]
-pad_template pad_template
[-size width_height]
-substrate substrate
```

Creates a bump template.

Parameters

`-help`	Prints a brief description of each `create_bump_template` parameter. *Data_type*: optional
`-name` *`name`*	Specifies the name of the bump template. *Data_type*: required
`-origin {lower_left | center | origin}`	
	Specifies the origin position. *Default*: `origin` *Data_type*: enum, optional
`-pad_template` *`pad_template`*	
	Specifies the name of the pad template. *Data_type*: required
`-size` *`width_height`*	Specifies the template size. If not specified, the tool creates a 5*5 square by default. *Data_type*: optional
`-substrate` *`substrate`*	Specifies the substrate to which the bump template belongs. *Data_type*: required

Examples

- The following command creates a bump template named `bump` using the `BPAD` pad template for the `INTERPOSER` substrate:

```
create_bump_template -name bump -substrate INTERPOSER -pad_template BPAD -origin center -size {50 50}
```

Related Commands

- `create_bump`
- `create_device_template`

create_contact_layer

```
create_contact_layer
[-help]
[-auto_connect_net {all | existing_contacts | new_contacts | none}]
[-cell cell]
[-cell_type {pin | device}]
-from_device device
-from_layer layer
[-interface_name interface_name]
[-only_selected_pins]
[-pin_height pin_height]
[-pin_material pin_material]
[-pin_net_config file]
[-pin_underfill_material pin_underfill_material]
[-prefix name]
[-stitch_zone stitch_zone]
-to_device device
-to_layer layer
[-create_contact_device]
```

Creates a contact layer.

Parameters

`-help`	Prints a brief description for each `create_contact_layer` parameter. *Data_type*: optional
`-auto_connect_net {all | existing_contacts | new_contacts | none}`	
	Specifies the mode for creating net mapping between the contact devices. The available modes are: • `all` - Auto-connects nets for both existing contacts and the new contacts. • `existing_contacts` - Auto-connects nets for the existing contacts on the contact layer. The behavior is same as clicking on the auto-connect net button in the *Edit > Contact Layer* dialog. • `new_contacts` - Auto-connects nets when creating contacts. • `none` - Specifies that net mapping is not to be created between the contact devices. *Default*: `new_contacts` *Data_type*: enum, optional
`-cell cell`	Specifies the cell used for the contact device. *Data_type*: optional
`-cell_type {pin | device}`	
	Specifies the type of the cell used for the contact device. *Data_type*: enum, optional
`-create_contact_device`	
	Specifies whether or not to create a contact device. *Data_type*: optional

`-from_device device`	
	Specifies the device from which the contact layer is created. *Data_type*: required
`-from_layer` *`layer`*	Specifies the layer from which the contact layer is created. *Data_type*: required
`-interface_name` *`interface_name`*	
	Specifies the unique short name for the contact layer. *Data_type*: optional
`-only_selected_pins`	
	Creates a contact device for only the selected pins. *Data_type*: optional
`-pin_height` *`pin_height`*	
	Specifies the height of the contact (mm). *Data_type*: distance, optional
`-pin_material` *`pin_material`*	
	Specifies the pin material. *Data_type*: optional
`-pin_net_config` *`file`*	
	Specifies a configuration file for the customized net of pin. *Data_type*: file, optional
`-pin_underfill_material` *`pin_underfill_material`*	
	Specifies the pin underfill material. *Data_type*: optional
`-prefix` *`name`*	Specifies the prefix of the created contact pins. *Data_type*: optional
`-stitch_zone` *`stitch_zone`*	
	Specifies the stitch zone. *Data_type*: optional
`-to_device` *`device`*	Specifies the device to which the contact layer is created. *Data_type*: required
`-to_layer` *`layer`*	Specifies the layer to which the contact layer is created. *Data_type*: required

Examples

- The following command creates only a contact layer:

```
create_contact_layer -from_device {/top/ASIC_IPG} \
                     -from_layer AP \
                     -to_device {/top/INTERPOSER} \
                     -to_layer AP
```

- The following command creates a contact layer with contact devices/pins:

```
create_contact_layer -from_device /Chip1 -from_layer AP \
    -to_device /INTERPOSER -to_layer AP \
    -cell_type device \
    -cell BUMP_INTERPOSER_SIGNAL \
    -prefix {}
```

- The following command creates a contact layer with matched pad:

```
create_contact_layer -from_device /Chip1 -from_layer AP \
    -to_device /INTERPOSER -to_layer AP \
    -cell_type pin \
    -cell {} \
    -prefix {}
```

- The following command creates a contact layer with the short name`inter_chip_f2f`:

```
create_contact_layer -interface_name inter_chip_f2f \
        -from_device /Chip1 -from_layer AP \
        -to_device /INTERPOSER -to_layer AP
```

Related Commands

- `create_contacts`
- `check_contacts`
- `delete_contact_layer`

create_contacts

```
create_contacts
[-help]
[-cell cell]
-cell_type {pin | device}
[contact_layer]
[-connect_all_contact_pin_terms]
[-dont_auto_connect_net]
[-from_device device]
[-from_layer layer]
[-ground_cell ground_cell]
[-interface_name interface_name]
[-mode {overwrite | merge}]
[-nc_cell nc_cell]
[-only_selected_pins]
[-orient {r0 | r90 | r180 | r270}]
[-pin_net_config file]
[-power_cell power_cell]
[-prefix name]
[-reverse]
[-stitch_zone stitch_zone]
[-to_device device]
[-to_layer layer]
[-create_contact_device]
```

Creates contacts based on specified parameters.

Parameters

`-help`	Prints a brief description for each `create_contacts` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell used for the contact device. *Data_type*: optional
`-cell_type {pin | device}`	
	Specifies the type of the cell used for the contact device. *Data_type*: enum, required
`-connect_all_contact_pin_terms`	
	Connects all the terms on the contact device pin to the contact device level net. *Data_type*: optional
`contact_layer`	Specifies the contact layer object instead of a 'from-to' configuration. *Data_type*: optional
`-create_contact_device`	
	Specifies whether or not to create a contact device. *Data_type*: optional
`-dont_auto_connect_net`	

	Specifies that net mapping will not be created between the contact devices. *Data_type*: optional
`-from_device` *`device`*	Specifies the device from which the contact layer is created. *Data_type*: optional
`-from_layer` *`layer`*	Specifies the layer from which the contact layer is created. *Data_type*: optional
`-ground_cell` *`ground_cell`*	
	Specifies the cell used for ground pins. *Data_type*: optional
`-interface_name` *`interface_name`*	
	Specifies the contact layer to be used by the given name. *Data_type*: optional
`-mode {overwrite | merge}`	
	Specifies the mode for creating contacts. *Default*: `merge` *Data_type*: enum, optional
`-nc_cell` *`nc_cell`*	Specifies the cell used for NC pins. *Data_type*: optional
`-only_selected_pins`	Creates contacts for only the selected pins. *Data_type*: optional
`-orient {r0 | r90 | r180 | r270}`	
	Specifies the rotation of contact bumps/pins. *Data_type*: enum, optional
`-pin_net_config` *`file`*	Specifies a configuration file for the customized net of pin. *Data_type*: file, optional
`-power_cell` *`power_cell`*	
	Specifies the cell used for power pins. *Data_type*: optional
`-prefix` *`name`*	Specifies the prefix of the created contact pins. *Data_type*: optional
`-reverse`	Specifies the reverse created order of specific contact layer object. *Data_type*: optional
`-stitch_zone` *`stitch_zone`*	
	Specifies the stitch zone. *Data_type*: optional

`-to_device` *`device`*	Specifies the device to which the contact layer is created. *Data_type*: optional
`-to_layer` *`layer`*	Specifies the layer to which the contact layer is created. *Data_type*: optional

Examples

- The following command creates contacts using the 'from-to' configuration:

```
create_contacts -from_device /Chip1 -from_layer AP \
                                -to_device /INTERPOSER -to_layer AP \
                                -cell_type device \
                                -cell BUMP_INTERPOSER_SIGNAL \
```

- The following set of commands creates contacts using the specified contact layer object:

```
### PASS contact layer object ###
set ct [create_contact_layer -from_device /Chip1 -from_layer AP \
    -to_device /INTERPOSER -to_layer AP]

create_contacts $ct \
    -cell_type device \
    -cell BUMP_INTERPOSER_SIGNAL \
    -prefix {}
```

- The following command creates contacts with matched pad:

```
create_contacts -from_device /Chip1 -from_layer AP \
                                -to_device /INTERPOSER -to_layer AP \
                                -cell_type pin \
                                -cell {} # empty cell for matched pad creation
```

- The following command creates contacts with a stitch zone constraint:

```
create_contacts -from_device top/ASIC_IPG -from_layer AP \
    -to_device top/INTERPOSER -to_layer AP \
    -cell_type device \
    -cell FRONT_UBUMP \
    -stitch_zone uBump
```

- The following command creates contacts using the short name, or given name, of the contact layer:

```
create_contacts  -interface_name inter_chip_f2f \
        -cell_type device \
        -cell BUMP_INTERPOSER_SIGNAL \
        -prefix {}
```

Related Commands

- `check_contacts`
- `create_contact_layer`
- `delete_contacts`

create_current_region

```
create_current_region
[-help]
-cell cell
[-current current]
-layer layer
-name name
-net net
[-power power]
[-power_density power_density]
-region region
[-threshold threshold]
[-voltage voltage]
```

Creates a current region with the specified parameters.

The total power of a current region is decided by the following, in descending order of priority:

1. Specified power of the region (`-power` value).
2. Specified power density of the region times the area (`-power_density * -region`)
3. Specified current of the region times the net's voltage (`-current * -voltage`)

Note that:

- The total power computation will be used for power mapping of Celsius.
- The current value of the net will be used for the ERA flow in Voltus.

Parameters

`-help`	Prints a brief description for each `create_current_region` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell owner. *Data_type*: required
`-current` *`current`*	Specifies the current level in mA. *Data_type*: optional
`-layer` *`layer`*	Specifies the name of the layer for the current region being updated. *Data_type*: required
`-name` *`name`*	Specifies the name of the current region to be created. *Data_type*: required
`-net` *`net`*	Specifies the name of the net(s) for the current region. *Data_type*: required
`-power` *`power`*	Specifies the power value in W. *Data_type*: optional
`-power_density` *`power_density`*	
	Specifies the power density in W/mm^2. *Data_type*: optional
`-region` *`region`*	Specifies the rectangular area for the current region. *Data_type*: required
`-threshold` *`threshold`*	Specifies the threshold of nets in V. *Data_type*: optional
`-voltage` *`voltage`*	Specifies the voltage of nets in V. *Data_type*: optional

Examples

- The following command creates a current region named `region1` for the `VSS` net with the specified parameters:

  ```
  create_current_region -cell INTERPOSER -name region1 -net VSS -current 150mA -layer Metal5 -region  "10 10 50 50"
  ```

- The following commands create current regions for multiple nets with the specified parameters:
  - `create_current_region -cell INTERPOSER -name region1 -net {VSS VDDC} -current {150mA 150mA} -layer M2 -region  "10 10 50 50"`
  - `create_current_region -cell INTERPOSER -name region1 -net {VSS VDDC} -current 150mA -layer M2 -region  "10 10 50 50"`
  - `create_current_region -cell INTERPOSER -name region1 -net {VSS VDDC} -current { 150 150} -layer M2 -region { 10 10 50 50} -power 5`

Related Commands

- `delete_current_region`
- `update_current_region`

create_design

```
create_design
[-help]
[-name name]
[-unit_dbu count]
[-unit_dbu_micron per_micron]
[-unit_decimal_place places]
[-unit_name unit_name]
```

Creates a design with the specified name or with no name.

Parameters

`-help`	Prints a brief description for each `create_design` parameter. *Data_type*: optional
`-name` *name*	Specifies the name to be used for the design. The value may be null. *Data_type*: optional
`-unit_dbu` *count*	Specifies the number of internal database units per distance unit. *Data_type*: optional
`-unit_dbu_micron` *per_micron*	
	Specifies the number of internal database distance units per micron. *Data_type*: optional
`-unit_decimal_place` *places*	
	Specifies the number of decimal places to display. For example, to display four decimal places, specify `4`. *Data_type*: optional
`-unit_name` *unit_name*	
	Specifies the distance unit name. For example, to store mils in picometers, specify `mil`. *Data_type*: optional

Example

- The following command creates a design named `top` that stores microns in picometers and displays four decimal places:

```
create_design -name top \
              -unit_name micron \
              -unit_decimal_place 4
```

Related Commands

- `check_design`
- `close_design`
- `is_top_design`
- `set_hierarchy`

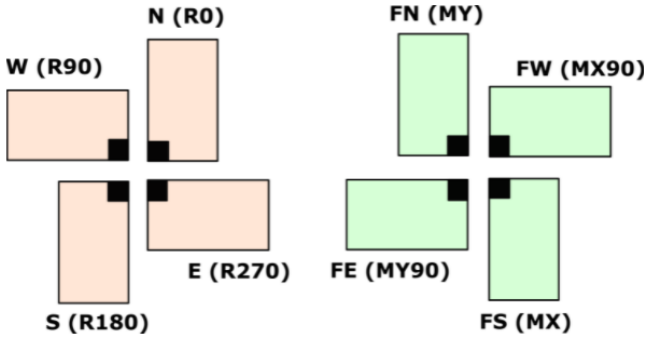
create_device

```
create_device
[-help]
[-height height]
[-module module]
-name name
[-origin {lower_left | center | origin}]
-parent parent
[-pin_csv file]
[-pin_device_template_name pin_device_template_name]
[-pin_layer layer]
[-pin_origin {center | origin}]
[-pin_pad_name pin_pad_name]
[-pin_shape{circle | square | octagon | rect | oblong }]
[-pin_shape_args width_height]
[-pin_template pin_template]
[-pin_type{regular | bump}]
[-polygon polygon]
[-size width_height]
[-substrate substrate]
[-type {die | pkg | board | cover |  block | core | endcap | bump | pad |
       | ip_component | mechanical | ring | hardmacro | softmacro | stdcell
       | contact_device | bondfinger | thermal_component | unknown} ]
[-z z]
[ [[-rotation rotate] [-mirror {x | y | no}] [-flip {yes | no}]]
 |[[-orientation {r0 | r90 | r180 | r270 | my | my90 | mx | mx90}] [-face_up {yes | no}]]]
[-location point | -llx llx -lly lly | -cx cx -cy cy]
[-llx llx -lly lly [-include_extent]]
{-template template | -template_name template_name}
```

Creates a device with the specified name and parent template.

Parameters

`-help`	Prints a brief description of each `create_device` parameter. *Data_type*: optional
`-cx` *cx*	Moves the center of the current device to the specific x-coordinate of the parent. *Data_type*: optional
`-cy` *cy*	Moves the center of the current device to the specific y-coordinate of the parent. *Data_type*: optional
`-face_up {yes | no}`	Specifies the flip and mirror x of the device. *Default*: `yes` *Data_type*: enum, optional
`-flip {yes | no}`	Flips the device and inverts the layer stack. *Data_type*: enum, optional
`-height` *height*	Sets the height of the device. *Data_type*: optional

`-include_extent`	Specifies whether to consider the seal ring and scribe to adjust the geometry information, which affects the `-llx -lly` result. *Data_type*: enum, optional
`-llx` *`llx`*	Moves the lower-left x-coordinate of the device. *Data_type*: optional
`-lly` *`lly`*	Moves the lower-left y-coordinate of the device. *Data_type*: optional
`-location` *`point`*	Sets the location for the device. *Data_type*: optional
`-mirror {x | y | no}`	Mirrors the device about its y-axis. *Data_type*: enum, optional
`-module` *`module`*	Specifies that the device is to be created using the existing module. You should not specify this parameter with the `-template` parameter. *Data_type*: module, optional
`-name` *`name`*	Specifies the name of the device. *Data_type*: required
`-orientation {r0 | r90 | r180 | r270 | my | my90 | mx | mx90}`	
	Specifies the orientation of the device. The image below shows the meaning of each orientation value.  *Data_type*: enum, optional
`-origin {lower_left | center | origin}`	
	Specifies the origin position. *Data_type*: enum, optional
`-parent` *`parent`*	Specifies the parent template of the device. *Data_type*: required
`-pin_csv` *`file`*	Specifies the CSV file name of the pin list. *Data_type*: file, optional
`-pin_device_template_name` *`pin_device_template_name`*	

	Specifies the name of the created device template. *Data_type*: optional
`-pin_layer` *`layer`*	Specifies the layer on which the pins are located. *Data_type*: optional
`-pin_origin {center | origin}`	
	Specifies the pin origin. *Default*: `center` *Data_type*: enum, optional
`-pin_pad_name` *`pin_pad_name`*	
	Specifies the name of the created pad template. *Data_type*: optional
`-pin_shape {circle | square | octagon | rect | oblong}`	
	Specifies the shape of the placed pin. *Data_type*: enum, optional
`-pin_shape_args` *`width_height`*	
	Specifies the shape width/height of the placed pin. *Data_type*: optional
`-pin_template` *`pin_template`*	
	Specifies the template of the placed pin. *Data_type*: optional
`-pin_type {regular | bump}`	
	Specifies the type of the importing pin. *Data_type*: enum, optional
`-polygon` *`polygon`*	Specifies the polygon boundary of the device. Format: `{x1 y1 x1 y2 ... xn yn}`
`-rotation` *`rotate`*	Rotates the device in degrees in a counter-clockwise/clockwise direction. For example, `90 cw` or `135 ccw` *Default*: clockwise
`-size` *`width_height`*	Specifies the device size. If not specified, the tool creates a 5*5 square by default. *Data_type*: optional
`-substrate` *`substrate`*	Specifies the substrate by which the device is created. *Data_type*: optional
`-template` *`template`*	Specifies an existing template for creating the device. You must specify either an existing template using this option or a new one using the `-template_name` option. *Data_type*: optional

`-template_name` *`template_name`*	
	Specifies the name of a new template. You must specify either a new template using this option or an existing one using the `-template` option. *Data_type*: optional
`-type {die | pkg | board | cover |  block | core | endcap | bump | pad | ip_component | mechanical | ring | hardmacro | softmacro | stdcell | contact_device | bondfinger | thermal_component | unknown}`	
	Specifies the type of the new template. Options are: • `die` - Represents a die design, such as an Interposer, ASIC, or an HBM die. This implementation is usually used for tools such as Innovus and Virtuoso. The design can be defined using a LEF/DEF or XDA file. • `pkg` - Represents a package design, such as a BGA or InFO package. This implementation is typically used for the Allegro platform. The design can be defined using a `.mcm`, `.dra`, or a `.sip` file. • `board` - Represents a board design, such as a PCB. This implementation is typically used for Allegro PCB designer. The design can be defined using a `.brd` file. • `cover` - Represents a cover device. This type maps to the `MACRO CLASS COVER` definition. Usually, power/ground bumps are grouped on a cover device. • `block` - Represents a predefined macro used in hierarchical design. This type maps to the `MACRO CLASS BLOCK` definition. • `core` - Represents a standard cell used in the core area. This type maps to the `MACRO CLASS CORE` definition. • `endcap` - Represents a macro placed at the ends of core rows. This type maps to the `MACRO CLASS ENDCAP` definition. • `bump` - Represents a physical-only cell that has bump geometries and pins. This type maps to the `MACRO CLASS COVER BUMP` definition. • `pad` - Represents an I/O pad. This type maps to the`MACRO CLASS PAD` definition. • `ip_component` - Represents a third party component, such as an IP component. • `mechanical` - Specifies mechanical symbols that represent the physical objects that can be placed in a board design and are not part of the netlist. Examples are mounting holes, tooling holes, and board stiffeners. • `ring` - Represents a large macro that has an internal power mesh and only exposes the powerpin shapes that form a ring along the macro boundary. This type maps to the `MACRO CLASS RING` definition.

	• `hardmacro` - Represents a predefined macro used in a hierarchical design. This type maps to the `MACRO CLASS BLOCK BLACKBOX` definition. • `softmacro` - Represents a predefined macro used in a hierarchical design. This type maps to the `MACRO CLASS BLOCK SOFT LEF` definition. • `stdcell` - Represents a standard cell. • `contact_device` - Represents the footprint of the device used with contact layer definition. • `bondfinger` - Represents a bond finger pin. • `thermal_component` - Represents the physical objects placed between the die or package in a design stack in thermal analysis and the related thermal flow. • `unknown` - Represents an unknown type. *Data_type*: enum, optional
`-z` *z*	Sets the z-location of the design stacking. *Data_type*: optional

Examples

- The following command creates a device named `INTERPOSER_TEST` from an existing template:

```
create_device -name INTERPOSER_TEST \
              -template {INTERPOSER:INTERPOSER} \
              -parent {PKG:top} \
              -type die
```

- The following command creates a new device and a new template with a specific boundary:

```
create_device -name INTERPOSER_TEST \
              -substrate INTERPOSER \
              -template_name {INTERPOSER_2} \
              -parent {PKG:top} \
              -type die \
              -size {1000 2000}
```

- The following commands create packages under `Design`:
  -
```
create_device -name InFO \
                -substrate InFo \
                -template_name {InFO} \
                -parent {[null]:<Design>} \
                -size {7000 2500} \
                -type pkg
```
  -
```
create_device -name PKG_LL \
                -substrate PKG_SUB \
                -template_name PKG \
                -type pkg
                -parent top \
                -size {1000 2000} \
                -origin lower_left
```

- The following command creates a device with a polygon outline:

```
create_device -name INTERPOSER_TEST \
    -substrate INTERPOSER \
    -template_name {INTERPOSER_2} \
    -parent {PKG:top} \
    -polygon {0 0 1000 0 1000 2000 800 2000 800 1800 200 1800 200 2000 0 2000}
```

Related Commands

- `create_device_template`
- `move_device`
- `update_device`

create_device_grid

```
create_device_grid
[-help]
-cell cell
-name name
-start_x start_x
-start_y start_y
-pitch_x pitch_x
-pitch_y pitch_y
```

Creates a device grid with the specified details.

Parameters

`-help`	Prints a brief description for each `create_device_grid` parameter. *Data_type*: optional
`-cell cell`	Specifies the cell for which the device grid is to be created. *Data_type*: required
`-name name`	Specifies the name of the grid. *Data_type*: required
`-pitch_x pitch_x`	Specifies the pitch x. *Data_type*: required
`-pitch_y pitch_y`	Specifies the pitch y. *Data_type*: required
`-start_x start_x`	Specifies the reference point of `pitch_x`. *Data_type*: required
`-start_y start_y`	Specifies the reference point of `pitch_y`. *Data_type*: required

Example

- The following command creates a device grid with the specified details:
  ```
  create_device_grid -cell N5:FCCC_N5 -name device_grid_LCM_153 -start_x 0.5 -start_y 0.5 -pitch_x 153 -pitch_y 153
  ```

Related Command

- `check_device_grid`

create_device_template

```
create_device_template
[-help]
[-module module]
-name name
[-origin {lower_left | center | origin}]
[-pin_csv file]
[-pin_device_template_name pin_device_template_name]
[-pin_layer layer]
[-pin_origin {center | origin}]
[-pin_pad_name pin_pad_name]
[-pin_shape{circle | square | octagon | rect | oblong }]
[-pin_shape_args width_height]
[-pin_template pin_template]
[-pin_type{regular | bump}]
[-polygon polygon]
[-size width_height]
-substrate substrate
[-type {die | pkg | board | cover | block | core | endcap | bump | group | pad | partition_block
        | spacer | ip_component | mechanical | ring | hardmacro | softmacro | stdcell
        | contact_device | bondfinger | thermal_component | unknown} ]
```

Creates a device template.

Parameters

`-help`	Prints a brief description of each `create_device_template` parameter. *Data_type*: optional
`-module` *`module`*	Specifies that the device is to be created using the existing module. *Data_type*: module, optional
`-name` *`name`*	Specifies the name of the device. *Data_type*: required
`-origin {lower_left | center | origin}`	
	Specifies the origin position. *Data_type*: enum, optional
`-pin_csv` *`file`*	Specifies the CSV file name of the pin list. *Data_type*: file, optional
`-pin_device_template_name` *`pin_device_template_name`*	
	Specifies the name of the created device template. *Data_type*: optional
`-pin_layer` *`layer`*	Specifies the layer on which the pins are located. *Data_type*: optional
`-pin_origin {center | origin}`	

	Specifies the pin origin. *Default*: `center` *Data_type*: enum, optional
`-pin_pad_name` *`pin_pad_name`*	
	Specifies the name of the created pad template. *Data_type*: optional
`-pin_shape {circle | square | octagon | rect | oblong}`	
	Specifies the shape of the placed pin. *Data_type*: enum, optional
`-pin_shape_args` *`width_height`*	
	Specifies the shape width/height of the placed pin. *Data_type*: optional
`-pin_template` *`pin_template`*	
	Specifies the template of the placed pin. *Data_type*: optional
`-pin_type {regular | bump}`	
	Specifies the type of the importing pin. *Data_type*: enum, optional
`-polygon` *`polygon`*	Specifies the polygon boundary of the device template. Format: `{x1 y1 x1 y2 ... xn yn}` *Data_type*: optional
`-size` *`width_height`*	Specifies the device template size. If not specified, the tool creates a 5*5 square by default. *Data_type*: optional
`-substrate` *`substrate`*	Specifies the substrate by which the device template is created. *Data_type*: required
`-type {die | pkg | board | cover | block | core | endcap | bump | group | pad | partition_block | spacer | ip_component | mechanical | ring | hardmacro | softmacro | stdcell | contact_device | bondfinger | thermal_component | unknown}`	
	Specifies the type of the new template. *Data_type*: enum, optional

Examples

- The following command creates a device template named `MyChip` from the specified pin CSV file:

```
create_device_template -name MyChip \
    -substrate Info_design \
    -pin_csv pin.csv
```

- The following command creates a device template named `INTERPOSER_2` of the specified size:

```
create_device_template -name INTERPOSER_2 \
              -substrate INTERPOSER \
               -size {1000 2000}
```

- The following command creates a device template named `INFO_11` of type package with a specific origin:

```
create_device_template -name INFO_11 \
              -substrate INFO_SUB \
              -template_name {InFO} \
               -size {1000 2000} \
              -type pkg \
              -origin lower_left
```

Related Commands

- `create_device`
- `update_device_template`

create_diff_pair

```
create_diff_pair
[-help]
-cell owner
-n_pattern n
-name_pattern name
-p_pattern p
```

Creates a differential pair for the specified device.

Parameters

`-help`	Prints a brief description for each `create_diff_pair` parameter. *Data_type*: optional
`-cell` *`owner`*	Specifies the owner cell. *Data_type*: required
`-n_pattern` *`n`*	Specifies the n pattern in regex for the matching net. name. *Data_type*: required
`-name_pattern` *`name`*	Specifies the pair name of the diff pair in regex. *Data_type*: required
`-p_pattern` *`p`*	Specifies the p pattern in regex for the mate net of n-pattern. *Data_type*: required

Examples

- The following command creates a differential pair with match net(BSIENN, BISENP), and differential pair name`BSIEN`:

  ```
  create_diff_pair -cell thor -n_pattern {(.*)N$} -p_pattern {{1}P} -name_pattern {{1}}
  ```

- The following command creates a differential pair with match net(BSIEN_C[5], BSIEN_T[5]), and differential pair name`BSIEN_[5]`:

  ```
  create_diff_pair -cell thor -n_pattern {(.*)C\[(.)\]$} -p_pattern {{1}T[{2}]} -name_pattern {{1}[{2}]}
  ```

Related Information

- *Defining Differential Pairs* section in Working with Designs in System Planner

create_external_contact_layer

```
create_external_contact_layer
[-help]
-device device
-layer layer
```

Creates an external contact layer object.

Parameters

`-help`	Prints a brief description for each `create_external_contact_layer` parameter. *Data_type*: optional
`-device` *`device`*	Specifies the device on which the external contact layer is created. *Data_type*: required
`-layer` *`layer`*	Specifies the layer on which the external contact layer is created. *Data_type*: required

Example

- The following command creates an external contact layer object on the specified device on the `UBMB` layer:
  `create_external_contact_layer -device /u_info_1 -layer UBMB`

Related Command

- `delete_external_contact_layer`

create_feedthru

```
create_feedthru
[-help]
[paths]
```

Creates feedthrough.

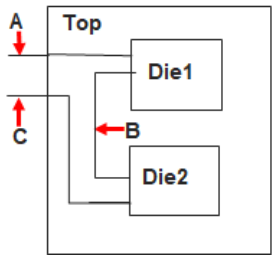
For implementation and LVS, the connectivity in a logical netlist needs to match the physical routing path. The net connection should go through the contact design. The nets in the designs that do not contact each other should not be connected directly in the netlist. In addition, the top-level ports should come from a specific design (and layer) based on the design stacking.

The feedthrough engine auto-creates a feedthrough netlist based on the original connectivity, contact layers, and pin mapping information. All the design-to-design connections connect directly through the defined contact layers. Only the specific design(s) that have contact to the top in the design stacking have a direct connection to the top ports.

When creating feedthrough, the feedthrough engine reuses the existing ports (terms) and wires (nets) on the design and creates new terms/nets if needed. No ports are added or deleted at the top level, and none of the original ports (terms) on the design are deleted.

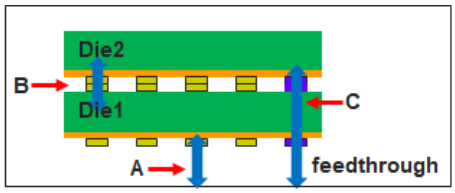
Example of Netlist Optimization with Feedthrough

Consider the original netlist depicted below:



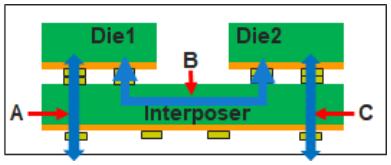
Here, `A` connects the ports of `Die1` to a top-level port directly. `B` is a connection between `Die1` and `Die2`, and `C` connects the ports of `Die2` to a top-level port directly.

One possible scenario is shown below. In the physical design stacking, `Die1` and `Die2` contact each other through bumps on the contact layers. Only the bumps on the backside of `Die2` connect to the fanout or package:



In the optimized netlist, connection `A` will remain the same because `Die1` can connect to the top-level ports directly. Connection `B` will also not be changed because `Die1` and `Die2` are in direct contact with each other. However, connection `C` now goes through the physical contact path from `Die2` through `Die1` to the top port. A new port is created on `Die1`, and it connects to the `Die2` port through a new top net. The `Die2` ports do not connect to the top ports directly after feedthrough creation.

Another possible scenario is that `Die1` and `Die2` are not in direct contact in the physical design stacking. Both `Die1` and `Die2` are in contact with the Interposer, as shown below. In this scenario, only the bumps on the backside of the Interposer connect to the package.



In the optimized netlist, connections `A` and `C` go through the physical contact path from `Die1` and `Die2`, respectively, through the Interposer to the top port. Connection `B` goes through the physical contact path from `Die1` through the Interposer to `Die2`. Two new ports are created on the

Interposer to connect `Die1` and `Die2`. In addition, a new net is created in the Interposer to connect the created ports in Interposer.

Parameters

`-help`	Prints a brief description for each `create_feedthru` parameter. *Data_type*: optional
paths	Specifies the path and layer of devices for the top terms. The format is: `{{path:layer} ...}` The path is the device connected to the top terms, and the layer is the port layer for the top terms. *Data_type*: optional

Example

- The following command creates feedthrough using the specified device path and layer:
  `create_feedthru {U1:bump}`

create_group

```
create_group
[-help]
-cell cell
-name name
[-pattern pattern]
[-region region]
```

Creates a group with specified cells and name.

Parameters

`-help`	Prints a brief description for each `create_group` parameter. *Data_type*: optional
`-cell` *cell*	Specifies the cell owner. *Data_type*: required
`-name` *name*	Specifies the name of the group to be created. *Data_type*: required
`-pattern` *pattern*	Specifies the component patterns. *Data_type*: optional
`-region` *region*	Specifies the the region for the group. *Data_type*: optional

Examples

- The following command creates a group named `par1` with the `DIE1` cell and `par1` patterns:

  ```
  create_group -name par1 -cell DIE1 -pattern {par1}
  ```

- The following command creates a group named `par` with the `DIE1` cell and the specified component patterns:

  ```
  create_group -name par -cell DIE1 -pattern {part* par1 par2}
  ```

- The following command creates a group named `par1` with the `DIE1` cell and `par1` region:

  ```
  create_group -name par1 -cell DIE1 -region par1
  ```

Related Command

- `delete_group`

create_layer

```
create_layer
[-help]
-name name
-substrate substrate
```

Creates a layer with the specified name and substrate.

Parameters

`-help`	Prints a brief description for each `create_layer` parameter. *Data_type*: optional
`-name` *`name`*	Specifies the name of the layer to be created. *Data_type*: required
`-substrate` *`substrate`*	Specifies the name of the substrate for the layer. *Data_type*: required

Example

- The following command creates a layer named `AP` with the `INTERPOSER` substrate:

  ```
  create_layer -name AP -substrate INTERPOSER
  ```

create_marker

```
create_marker
[-help]
[-bbox bbox]
[-cell cellName]
[-description description]
[-layer layerName]
[-location location]
[-poly poly]
[-sub_type subTypeName]
[-tool toolName]
[-type typeName]
```

Creates markers for violations in the database.

Parameters

`-help`	Prints a brief description for each `create_marker` parameter. *Data_type*: optional
`-bbox` *`bbox`*	Specifies the area of the bounding box of the marker. *Data_type*: optional
`-cell` *`cellName`*	Specifies the owner of the marker. *Data_type*: optional
`-description` *`description`*	Specifies the description of the marker. *Data_type*: optional
`-layer` *`layerName`*	Specifies layers. *Data_type*: optional
`-location` *`location`*	Specifies the location of the marker. *Data_type*: optional
`-poly` *`poly`*	Specifies the area of the bounding polygon of the marker. *Data_type*: optional
`-sub_type` *`subTypeName`*	Specifies the marker subtype. *Data_type*: optional
`-tool` *`toolName`*	Specifies the source software application. *Default*: Other *Data_type*: optional
`-type` *`typeName`*	Specifies the marker type. *Data_type*: optional

Examples

- The following command generates markers for the tool named `pegasus` with type `alignment` and subtype `pin_to_pin`:
  ```
  create_marker -location {100 100} -cell Design -layer {N3::metal1 N5::metal2} -tool pegasus -type alignment -sub_type pin_to_pin
  ```
- The following command generates markers within the specified bounding box for the cell `N5_DIE` on the specified layers:
  ```
  create_marker -bbox {100 100 200 200} -cell N5_DIE -layer {metal1 metal2}
  ```

Related Command

- `delete_marker`

create_modInst

```
create_modInst
[-help]
-module module
-name name
-parent parent
```

Creates a module instance.

Parameters

`-help`	Prints a brief description for each `create_modInst` parameter. *Data_type*: optional
`-module` *`module`*	Specifies the module to be used for the new module instance. *Data_type*: module, required
`-name` *`name`*	Specifies the name of the new module instance. *Data_type*: required
`-parent` *`parent`*	Specifies the parent for the new module instance. *Data_type*: module, required

Example

- The following command creates a module instance named `inst` with parent `parentMod`:
  `create_modInst -module mod -name inst -parent parentMod`

Related Command

- `create_module`

create_module

```
create_module
[-help]
-name name
```

Creates a module.

Parameters

`-help`	Prints a brief description for each `create_module` parameter. *Data_type*: optional
`-name` *`name`*	Specifies the name of the module being created. *Data_type*: required

Example

- The following command creates a module named`mod`:

```
create_module -name mod
```

create_module_model

```
create_module_model
[-help]
-cell cell_name
[-overwrite]
-tag tag_name
-type type
```

Creates a module model.

Parameters

`-help`	Prints a brief description for each `create_module_model` parameter. *Data_type*: optional
`-cell` *`cell_name`*	Specifies the name of the module model being created. *Data_type*: required
`-overwrite`	Specifies whether to overwrite without a user query, if a module model of the same name exists already. *Data_type*: optional
`-tag` *`tag_name`*	Specifies the tag name of the saved model. *Data_type*: required
`-type` *`type`*	Specifies the type of the module model being created. For *`type`*, you can specify: • `oio` - Equivalent to `-cell * -type pre_pnr` • `pre_pnr` - Saves specific cells based on -cell • `pnr` - For OpenAccess • `pkg` - Invokes apd to dump mcm. *Data_type*: required

Examples

- The following command exports a single die and top:
  `create_module_model -cell U1 -type pre_pnr -tag v0`
- The following command exports all dies in the design and top:
  `create_module_model -cell Design -type oio -tag v0`
- The following command exports all dies, packages and top:
  `create_module_model -cell * -type pre_pnr -tag v0`
- The following command exports an mcm design:
  `create_module_model -cell INFO -type pkg -tag v0`
- The following command exports an OpenAccess design (Note: You need to set the environment variable `OrbitIO.OAEnabled`.):
  `create_module_model -cell analog_die -type pnr -tag v0`

Related Commands

- `read_module_model`
- `set_module_model`

create_net

```
create_net
[-help]
-module module
-name name
[-bus startID:endID]
[-ground]
[-power]
```

Creates a net with the specified name and module.

Parameters

`-help`	Prints a brief description for each `create_net` parameter. *Data_type*: optional
`-bus startID:endID`	Specifies the bit range for bus nets. The `startID` and `endID` indicate the first and last bits on the bus. You must separate the start and end IDs with a colon (:). *Data_type*: optional
`-ground`	Adds a ground net. *Data_type*: optional
`-module module`	Specifies the name of the module or template for the net. *Data_type*: required
`-name name`	Specifies the name of the net to be created. *Data_type*: required
`-power`	Adds a power net. *Data_type*: optional

Examples

- The following command creates a net named `VSS` with the `HBM` module:

  ```
  create_net -module HBM -name VSS
  ```

- The following command creates a ground net named `VSS` with the `HBM` module and bus bit range of 0:9:

  ```
  create_net -module HBM -name VSS -bus 0:9 -ground
  ```

create_package

```
create_package
[-help]
[-center_column center_column]
[-center_height center_height]
[-center_pitch_x center_pitch_x]
[-center_pitch_y center_pitch_y]
[-center_ring_skip center_ring_skip]
[-center_row center_row]
[-center_shape_rotation center_shape_rotation]
[-center_spec {count_pitch | size_pitch | size_count | inherit_outer_pitch | inherit_outer | inherit_inner_pitch |
inherit_inner}]
[-center_stagger]
[-center_width center_width]
[-height height]
[-inner_column inner_column]
[-inner_height inner_height]
[-inner_pitch_x inner_pitch_x]
[-inner_pitch_y inner_pitch_y]
[-inner_ring_depth inner_ring_depth]
[-inner_ring_skip inner_ring_skip]
[-inner_row inner_row]
[-inner_shape_rotation inner_shape_rotation]
[-inner_spec {count_pitch | size_pitch | size_count | inherit_outer_pitch | inherit_outer | inherit_inner_pitch
| inherit_inner}]
[-inner_stagger]
[-inner_width inner_width]
[-module module]
-name name
[-outer_column outer_column]
[-outer_height outer_height]
[-outer_pitch_x outer_pitch_x]
[-outer_pitch_y outer_pitch_y]
[-outer_ring_depth outer_ring_depth]
[-outer_row outer_row]
[-outer_shape_rotation outer_shape_rotation]
[-outer_spec {count_pitch | size_pitch | size_count | inherit_outer_pitch | inherit_outer | inherit_inner_pitch |
inherit_inner}]
[-outer_stagger]
[-outer_width outer_width]
[-pattern_type {uniform_array | array_1ring | array_2ring | one_ring | two_ring}]
[-perimeter perimeter]
[-pin_start {sw | se | nw | ne}]
-substrate substrate
-template_name template_name
[-width width]
[[-outer_pad_template outer_pad_template
  | -outer_shape {circle | square | octagon | rect | oblong}
    -outer_shape_arg outer_shape_arg]
 [-inner_pad_template inner_pad_template
  | -inner_shape {circle | square | octagon | rect | oblong}
    -inner_shape_arg inner_shape_arg]
 [-center_pad_template center_pad_template
  | -center_shape {circle | square | octagon | rect | oblong}
    -center_shape_arg center_shape_arg]]
```

Creates a package with the specified name and substrate.

Parameters

`-help`	Prints a brief description for each `create_package` parameter. *Data_type*: optional
`-center_column` *`center_column`*	Specifies the column of the center array. *Data_type*: optional
`-center_height` *`center_height`*	Specifies the height of the center array. *Data_type*: optional
`-center_pad_template` *`center_pad_template`*	Specifies the pad template of the placed pins of the center ring. *Data_type*: optional
`-center_pitch_x` *`center_pitch_x`*	Specifies the pitch x of the center array. *Data_type*: optional
`-center_pitch_y` *`center_pitch_y`*	Specifies the pitch y of the center array. *Data_type*: optional
`-center_ring_skip` *`center_ring_skip`*	Specifies the skip ring of the center array. *Data_type*: optional
`-center_row` *`center_row`*	Specifies the row of the center array. *Data_type*: optional
`-center_shape {circle | square | octagon | rect | oblong}`	
	Specifies the shape of the center array. *Data_type*: optional
`-center_shape_arg` *`center_shape_arg`*	Specifies the shape arguments of the center array. *Data_type*: optional
`-center_shape_rotation` *`center_shape_rotation`*	Specifies the shape rotation of the center array. *Data_type*: optional
`-center_spec {count_pitch | size_pitch | size_count | inherit_outer_pitch | inherit_outer | inherit_inner_pitch | inherit_inner}`	
	Specifies the spec of the center array. *Data_type*: enum, optional
`-center_stagger`	Specifies the stagger of the center array. *Data_type*: optional
`-center_width` *`center_width`*	Specifies the width of the center array. *Data_type*: optional
`-height` *`height`*	Specifies the height of the package. *Data_type*: optional

`-inner_column` *`inner_column`*	Specifies the column of the inner ring. *Data_type*: optional
`-inner_height` *`inner_height`*	Specifies the height of the inner ring. *Data_type*: optional
`-inner_pad_template` *`inner_pad_template`*	Specifies the pad template of the placed pins of the inner ring. *Data_type*: optional
`-inner_pitch_x` *`inner_pitch_x`*	Specifies the pitch x of the inner ring. *Data_type*: optional
`-inner_pitch_y` *`inner_pitch_y`*	Specifies the pitch y of the inner ring. *Data_type*: optional
`-inner_ring_depth` *`inner_ring_depth`*	Specifies the ring depth of the inner ring. *Data_type*: optional
`-inner_ring_skip` *`inner_ring_skip`*	Specifies the skip ring of the inner ring. *Data_type*: optional
`-inner_row` *`inner_row`*	Specifies the row of the inner ring. *Data_type*: optional
`-inner_shape {circle | square | octagon | rect | oblong}`	
	Specifies the shape of the inner ring. *Data_type*: optional
`-inner_shape_arg` *`inner_shape_arg`*	Specifies the shape arguments of the inner ring. *Data_type*: optional
`-inner_shape_rotation` *`inner_shape_rotation`*	Specifies the shape rotation of the inner ring. *Data_type*: optional
`-inner_spec {count_pitch | size_pitch | size_count | inherit_outer_pitch | inherit_outer | inherit_inner_pitch | inherit_inner}`	
	Specifies the spec of the inner ring. *Data_type*: enum, optional
`-inner_stagger`	Specifies the stagger of the inner ring. *Data_type*: optional
`-inner_width` *`inner_width`*	Specifies the width of the inner ring. *Data_type*: optional
`-module` *`module`*	Specifies that the package is to be created using an existing module. *Data_type*: module, optional
`-name` *`name`*	Specifies the name of the package. *Data_type*: required

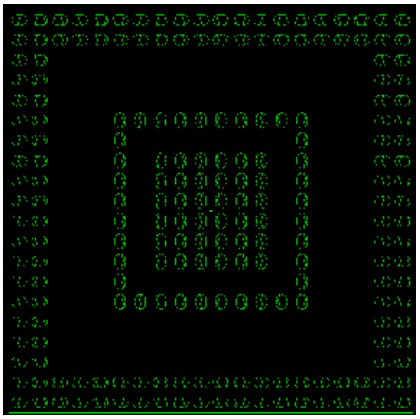
`-outer_column` *`outer_column`*	Specifies the column of the outer ring. *Data_type*: optional
`-outer_height` *`outer_height`*	Specifies the height of the outer ring. *Data_type*: optional
`-outer_pad_template` *`outer_pad_template`*	Specifies the pad template of the placed pins of the outer ring. *Data_type*: optional
`-outer_pitch_x` *`outer_pitch_x`*	Specifies the pitch x of the outer ring. *Data_type*: optional
`-outer_pitch_y` *`outer_pitch_y`*	Specifies the pitch y of the outer ring. *Data_type*: optional
`-outer_ring_depth` *`outer_ring_depth`*	Specifies the ring depth y of the outer ring. *Data_type*: optional
`-outer_row` *`outer_row`*	Specifies the row of the outer ring. *Data_type*: optional
`-outer_shape {circle | square | octagon | rect | oblong}`	
	Specifies the shape of the outer ring. *Data_type*: enum, optional
`-outer_shape_arg` *`outer_shape_arg`*	Specifies the shape arguments of the outer ring. *Data_type*: optional
`-outer_shape_rotation` *`outer_shape_rotation`*	Specifies the shape rotation of the outer ring. *Data_type*: optional
`-outer_spec {count_pitch | size_pitch | size_count | inherit_outer_pitch | inherit_outer | inherit_inner_pitch | inherit_inner}`	
	Specifies the spec of the outer ring. *Data_type*: enum, optional
`-outer_stagger`	Specifies the stagger of the outer ring. *Data_type*: optional
`-outer_width` *`outer_width`*	Specifies the width of the outer ring. *Data_type*: optional
`-pattern_type {uniform_array | array_1ring | array_2ring | one_ring | two_ring}`	
	Specifies the ring pattern of the package. Default: `array_2ring` *Data_type*: enum, optional
`-perimeter` *`perimeter`*	Specifies the perimeter of the package. *Data_type*: optional

`-pin_start {sw | se | nw | ne}`	Specifies the pin start location of the package. Default: `sw` *Data_type*: enum, optional
`-substrate` *`substrate`*	Specifies the substrate by which the package is created. *Data_type*: required
`-template_name` *`template_name`*	Specifies the name of the new template. *Data_type*: required
`-width` *`width`*	Specifies the width of the package. *Data_type*: optional

Examples

- The following command creates a package with a center array, an outer ring, and an inner ring:

```
create_package -substrate PKG \
    -name MyPackage_A \
    -template_name MyPackage \
    -pin_start sw \
    -pattern_type array_2ring \
    -outer_spec count_pitch \
    -outer_column 20 \
    -outer_row 20 \
    -outer_pitch_x 2000 \
    -outer_pitch_y 2000 \
    -outer_ring_depth 2 \
    -outer_shape oblong \
    -outer_shape_arg {1000 1500} \
    -outer_shape_rotation 90 \
    -inner_spec inherit_outer \
    -inner_ring_depth 1 \
    -inner_ring_skip 3 \
    -center_spec inherit_inner \
    -center_ring_skip 1
```



- The following command creates a package with a circular shape:

```
create_package -substrate PKG \
    -name MyPackage_A \
    -template_name MyPackage \
    -pin_start sw \
```

```
    -pattern_type array_2ring \
    -outer_spec count_pitch \
    -outer_column 20 \
    -outer_row 20 \
    -outer_pitch_x 2000 \
    -outer_pitch_y 2000 \
    -outer_ring_depth 2 \
    -outer_shape circle \
    -outer_shape_arg {1000} \ # diameter
    -outer_shape_rotation 90 \
    -inner_spec inherit_outer \
    -inner_ring_depth 1 \
    -inner_ring_skip 3 \
    -center_spec inherit_inner \
    -center_ring_skip 1
```

- The following set of commands creates a package with a pad template:

```
create_padstack -substrate PKG -name PadTemplate_MyPackage_A -pad_stack {Ball:Oblong:1000 1500}

create_package -substrate PKG \
    -name MyPackage_A \
    -template_name MyPackage \
    -pin_start sw \
    -pattern_type array_2ring \
    -outer_spec count_pitch \
    -outer_column 20 \
    -outer_row 20 \
    -outer_pitch_x 2000 \
    -outer_pitch_y 2000 \
    -outer_ring_depth 2 \
    -outer_shape_rotation 90 \
    -outer_pad_template PadTemplate_MyPackage_A \
    -inner_spec inherit_outer \
    -inner_ring_depth 1 \
    -inner_ring_skip 3 \
    -center_spec inherit_inner \
    -center_ring_skip 1
```

create_padstack

```
create_padstack
[-help]
-name name
[-pad_stack layer:type:args]
-substrate substrate
```

Creates a padstack with the specified name.

Note: You can access the functionality of the `create_padstack` command using the `create_pad_template` alias.

Parameters

`-help`	Prints a brief description for each `create_padstack` parameter. *Data_type*: optional
`-name` *`name`*	Specifies the name of the padstack. *Data_type*: required
`-pad_stack` *`layer:type:args`*	
	Defines the pad stack. The format is *`layer:type:args`*. For example: `-pad_stack {{Metal1:Circle:50}{Metal2:Rect:50 100}}` *Data_type*: optional
`-substrate` *`substrate`*	Specifies the substrate for which the padstack is being created. *Data_type*: required

Example

- The following command creates a padstack named `PKG_PAD`:

```
create_padstack -name PKG_PAD \
                -substrate PKG \
                -pad_stack {{C4Bump:Circle:50}{Ball:Oblong:50 100}}
```

Related Commands

- `delete_padstack`
- `update_padstack`

create_pin_grid

```
create_pin_grid
[-help]
-cell cell
[-layers layers]
-name name
-pitch_x pitch_x
-pitch_y pitch_y
-start_x start_x
-start_y start_y
```

Creates a pin grid as per the specified settings.

Parameters

`-help`	Prints a brief description for each `create_pin_grid` parameter. *Data_type*: optional
`-cell cell`	Specifies the cell for which the pin grid is being created. *Data_type*: required
`-layers layers`	Specifies the routable layer(s) used by the grid. Default is all layers. *Data_type*: optional
`-name name`	Specifies the name of the pin grid. *Data_type*: required
`-pitch_x pitch_x`	Specifies the x-spacing between two pins. *Data_type*: required
`-pitch_y pitch_y`	Specifies the y-spacing between two pins. *Data_type*: required
`-start_x start_x`	Specifies the reference point of pitch_x. *Data_type*: required
`-start_y start_y`	Specifies the reference point of pitch_y. *Data_type*: required

Examples

- The following command creates a pin grid, named `pin_grid`, for the `N5:FCCC_N5` cell:
  ```
  create_pin_grid -cell N5:FCCC_N5 -name pin_grid -start_x 0.3 -start_y 0.3 -pitch_x 0.5 -pitch_y 0.5
  ```
- The following command creates a pin grid, named `pin_grid_9x9`, for the `ASIC_IPG:ASIC_IPG` cell with the specified layers:
  ```
  create_pin_grid -cell ASIC_IPG:ASIC_IPG -name pin_grid_9x9 -start_x 50 -start_y 50 -pitch_x 9 -pitch_y 9 -layers {PG AP}
  ```

Related Command

- `check_pin_grid`

create_pin_matrix_pattern

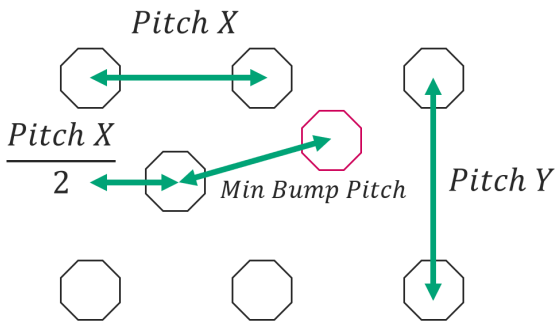
```
create_pin_matrix_pattern
[-help]
-cell owner
[-corner_keepout_type{rectangle | triangle}]
[-corner_keepout_x corner_keepout_x]
[-corner_keepout_y corner_keepout_y]
[-grid grid]
[-loc x_y]
[-loc_type {cell_center | cell_lowerleft | geometry_center | geometry_lowerleft}]
-min_bump_pitch bump_pitch
[-mirror]
[-name_prefix prefix]
[-no_cover]
[-orient {r0 | r90 | r180 | r270}]
[-pitch x_y]
-place_cell place
[-replace cover_name]
[-stagger]
[-stitch_zone stitch_zone]
-type {device | pin}
{-pattern_array row_column | -edge_spacing left_bottom_right_top | -area left_bottom_right_top}
```

Creates a pin matrix pattern as per specified settings.

Parameters

`-help`	Prints a brief description for each `create_pin_matrix_pattern` parameter. *Data_type*: optional
`-area` *`left_bottom_right_top`*	
	Specifies the area for placing bumps. *Data_type*: optional
`-cell` *`owner`*	Specifies the owner cell. *Data_type*: required
`-corner_keepout_type {rectangle | triangle}`	
	Specifies the shape type of the corner keepout. *Data_type*: enum, optional
`-corner_keepout_x` *`corner_keepout_x`*	
	Specifies the x distance of the corner keepout. *Data_type*: optional
`-corner_keepout_y` *`corner_keepout_y`*	
	Specifies the y distance of the corner keepout. *Data_type*: optional
`-edge_spacing` *`left_bottom_right_top`*	

	Specifies the minimum Die-edge clearance. *Data_type*: required
`-grid` *`grid`*	Specifies the name of the pin grid whose pitch is to be used as the default pitch for the pin matrix being created. *Data_type*: optional
`-loc` *`x_y`*	Specifies the reference location. Bumps are placed on the grid made by `-loc` and `-pitch` values. *Data_type*: optional
`-loc_type {cell_center | cell_lowerleft | geometry_center | geometry_lowerleft}`	
	Specifies the type of location for bump placement. *Default*: `geometry_center` *Data_type*: enum, optional
`-min_bump_pitch` *`bump_pitch`*	
	Specifies the minimum bump pitch. *Data_type*: required
`-mirror`	Specifies the mirror of the placement instance. *Data_type*: optional
`-name_prefix` *`prefix`*	
	Specifies the prefix name of the placement instance. *Data_type*: optional
`-no_cover`	Specifies that pin should be created without the `COVER` macro. *Data_type*: optional
`-orient {r0 | r90 | r180 | r270}`	
	Specifies the rotation orient. *Data_type*: enum, optional
`-pattern_array` *`row_column`*	
	Specifies the row and column number for placing bumps. When `-loc` is set, use that as the location of the lower-left bump.

`-pitch x_y`	Specifies the pitch of the matrix being created. This parameter must be specified if `-grid` is not set.  *Data_type*: optional
`-place_cell place`	
	Specifies the object for placement. *Data_type*: required
`-replace cover_name`	Specifies the name of an existing cover to be replaced. *Data_type*: optional
`-stagger`	Specifies that the stagger pattern of placement should be used. *Data_type*: optional
`-stitch_zone stitch_zone`	
	Specifies the name of the stitch zone. *Data_type*: optional
`-type {device | pin}`	Specifies the default bump device. *Data_type*: enum, required

Examples

- The following command creates a basic pin matrix pattern:

```
create_pin_matrix_pattern -type pin \
    -place_cell {ASIC_IPG:pad_ubump_25} \
    -cell {ASIC_IPG:ASIC} \
    -name_prefix Bump \
    -min_bump_pitch 55 \
    -pitch {55 55} \
    -edge_spacing {55 0 55 0}
```

- The following command creates a pin matrix pattern over the specified area:

```
create_pin_matrix_pattern -type pin \
    -place_cell {ASIC_IPG:pad_ubump_25} \
    -cell {ASIC_IPG:ASIC} \
    -name_prefix Bump \
```

```
    -min_bump_pitch 55 \
    -pitch {55 55} \
    -area {100 100 500 500}
```

- The following command creates a pin matrix in the specified pattern array:

```
create_pin_matrix_pattern -type pin \
    -place_cell {ASIC_IPG:pad_ubump_25} \
    -cell {ASIC_IPG:ASIC} \
    -name_prefix Bump \
    -min_bump_pitch 55 \
    -pitch {55 55} \
    -pattern_array {10 20}
```

- The following command creates a pin matrix pattern with a stitch zone:

```
create_pin_matrix_pattern -type device \
    -place_cell {INTERPOSER:C4_BUMPCELL} \
    -cell {INTERPOSER:INTERPOSER} \
    -name_prefix "Bump" \
    -min_bump_pitch 0 \
    -pitch {150 150} \
    -edge_spacing {100 100 100 100} \
    -stitch_zone C4Bump
```

- The following command creates a pin matrix pattern with a corner keepout:

```
create_pin_matrix_pattern -type device \
    -place_cell {INTERPOSER:C4_BUMPCELL} \
    -cell {INTERPOSER:INTERPOSER} \
    -name_prefix "Bump" \
    -min_bump_pitch 0 \
    -pitch {150 150} \
    -edge_spacing {100 100 100 100} \
    -corner_keepout_x 1200 \
    -corner_keepout_y 1200 \
    -corner_keepout_type triangle
```

- The following command creates a pin matrix pattern with the specified grid's pitch as reference:

```
create_pin_matrix_pattern -type device \
    -place_cell {INTERPOSER:C4_BUMPCELL} \
    -cell {INTERPOSER:INTERPOSER} \
    -name_prefix "Bump" \
    -min_bump_pitch 0 \
    -grid "pin_grid_15x15" \
    -edge_spacing {100 100 100 100} \
    -corner_keepout_x 1200 \
    -corner_keepout_y 1200 \
    -corner_keepout_type triangle
```

Related Command

- create_bump

create_place_blockage

```
create_place_blockage
[-help]
-cell cell_name
{-cover | -polygon polygon | -rects rects | [-outer_ring_by_side outer_ring_by_side
        | -outer_ring_by_edge outer_ring_by_edge] | [-inner_ring_by_side inner_ring_by_side
        | -inner_ring_by_edge inner_ring_by_edge]}
[-no_cut_by_core]
```

Creates a placement blockage.

Parameters

`-help`	Prints a brief description for each `create_place_blockage` parameter. *Data_type*: optional
`-cell` *`cell_name`*	Specifies the cell on which the placement blockage is to be applied. *Data_type*: required
`-cover`	Covers the block instance. *Data_type*: optional
`-inner_ring_by_edge` *`inner_ring_by_edge`*	
	Creates a hollow ring of placement blockage around the inside of the object(s) specified:`{edge1 edge2 edge3 ...}` *Data_type*: optional
`-inner_ring_by_side` *`inner_ring_by_side`*	
	Creates a hollow ring of placement blockage around the inside of the object(s) specified:`{x1 y1 x2 y2}` *Data_type*: optional
`-no_cut_by_core`	Specifies that placement blockages should not be truncated at the core boundary *Data_type*: optional
`-outer_ring_by_edge` *`outer_ring_by_edge`*	
	Creates a hollow ring of placement blockage around the outside of the object(s) specified:`{edge1 edge2 edge3 ...}` *Data_type*: optional
`-outer_ring_by_side` *`outer_ring_by_side`*	
	Creates a hollow ring of placement blockage around the outside of the object(s) specified: `{x1 y1 x2 y2}` *Data_type*: optional
`-polygon` *`polygon`*	Specifies the polygon vertices of the blockage, in microns:`{{ x1 y1 }{ x2 y2 }...}` *Data_type*: optional
`-rects` *`rects`*	Specifies the box vertices of the blockage, in microns:`{{lx1 ly1 ux1 uy1 }{ lx2 ly2 ux2 uy2 }...}` *Data_type*: optional

Example

- The following command creates a placement blockage for the specified cell:
  `create_place_blockage -cell DIE1 -cover`

Related Command

- `create_route_blockage`

create_projective_pins

```
create_projective_pins
[-help]
[-cell cell]
-cell_type {pin | device}
[-dont_auto_connect_net]
[-from_device device]
[-from_layer layer]
[-only_selected_pins]
[-orient {r0 | r90 | r180 | r270}]
[-pin_net_config file]
[-prefix name]
[-stitch_zone stitch_zone]
[-to_device device]
[-to_layer layer]
```

Create projective pins for the specified cell type. This command is used forcross-substrate projection. That is, you can use it to create projective pins from Interposer to InFO package or from Die to Interposer.

Parameters

`-help`	Prints a brief description for each `create_projective_pins` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell used for the project device. *Data_type*: optional
`-cell_type {pin | device}`	Specifies the kind of cell. *Data_type*: enum, required
`-dont_auto_connect_net`	Does not create net mapping between the project devices. *Data_type*: optional
`-from_device` *`device`*	Specifies the device from which the project layer is created. *Data_type*: optional
`-from_layer` *`layer`*	Specifies the layer from which the project layer is created. *Data_type*: optional
`-only_selected_pins`	Creates contacts only for the selected pins. *Data_type*: optional
`-orient {r0 | r90 | r180 | r270}`	
	Specifies the rotation of contact bumps/pins. *Data_type*: enum, optional
`-pin_net_config` *`file`*	Specifies a configuration file for the customized net of a pin. *Data_type*: file, optional
`-prefix` *`name`*	Specifies the prefix of the created project devices. *Data_type*: optional
`-stitch_zone` *`stitch_zone`*	Specifies the name of the stitch zone. *Data_type*: optional
`-to_device` *`device`*	Specifies the device to which the project layer is created. *Data_type*: optional
`-to_layer` *`layer`*	Specifies the layer to which the project layer is created. *Data_type*: optional

Example

- The following command creates projective pins for the `pin` cell type:

```
create_projective_pins -from_device /DIE1 -from_layer bump -to_device /DIE1/pa1 -to_layer m1 -cell_type pin
```

Related Commands

- `create_bump`
- `create_pin_matrix_pattern`

create_region

```
create_region
[-help]
-cell cell
[-from_partition_blocks]
[-name name]
[-rect rect]
[-selected]
[-type {none | fence | guide}]
```

Creates a region with the specified cell.

Parameters

`-help`	Prints a brief description for each `create_region` parameter. *Data_type*: optional
`-cell cell`	Specifies the cell owner. *Data_type*: required
`-from_partition_blocks`	Creates regions from partition blocks. *Data_type*: optional
`-name name`	Specifies the name of the region. *Data_type*: optional
`-rect rect`	Defines the rectangular shape of this region. *Data_type*: optional
`-selected`	Creates regions from the selected devices. *Data_type*: optional
`-type {none | fence | guide}`	
	Specifies the type of this region. *Default*: `none` *Data_type*: enum, optional

Example

- The following command creates a fence region for the `INTERPOSER` cell:

```
create_region -name CELL_FENCE -cell INTERPOSER -type fence -rect {100 200 1000 1000}
```

Related Commands

- `delete_region`

create_route_blockage

```
create_route_blockage
[-help]
-cell cell_name
{-cover | -polygon polygon | -rects rects}
-layers layer_name
```

Creates a routing blockage.

Parameters

`-help`	Prints a brief description for each `create_route_blockage` parameter. *Data_type*: optional
`-cell cell_name`	Specifies the cell on which the routing blockage is to be applied. *Data_type*: required
`-cover`	Specifies a routing blockage of the same size as the specified block instance. *Data_type*: optional
`-layers layer_name`	Specifies the layer or list of layers on which the routing blockage is to be applied. *Data_type*: required
`-polygon polygon`	Specifies the polygon vertices of the blockage, in microns:`{{ x1 y1 }{ x2 y2 }...}` *Data_type*: optional
`-rects rects`	Specifies the coordinates of a list of blockage areas, in microns:`{{lx1 ly1 ux1 uy1 }{ lx2 ly2 ux2 uy2 }...}` *Data_type*: optional

Example

- The following command creates a routing blockage for the specified cell on the specified layers:
  `create_route_blockage -cell DIE1 -cover -layers {bump m1 m2 m3}`

Related Command

- `create_place_blockage`

create_stitch_zone

```
create_stitch_zone
[-help]
[-c_height c_height]
[-c_width c_width]
-cell cell
-name name
[-x x]
[-x_width x_width]
[-y y]
[-y_width y_width]
```

Creates a stich zone with the specified name and using the specified cell.

Parameters

`-help`	Prints a brief description for each `create_stitch_zone` parameter. *Data_type*: optional
`-c_height c_height`	Specifies the height of the cross intersection. *Data_type*: optional
`-c_width c_width`	Specifies the width of the cross intersection. *Data_type*: optional
`-cell cell`	Specifies the cell owner. *Data_type*: required
`-name name`	Specifies the name of the stitch zone. *Data_type*: required
`-x x`	Specifies the x-coordinate. *Data_type*: optional
`-x_width x_width`	Specifies the width in x-coordinate. *Data_type*: optional
`-y y`	Specifies the y-coordinate. *Data_type*: optional
`-y_width y_width`	Specifies the width in y-coordinate. *Data_type*: optional

Example

- The following command creates a stitch zone named `C4Bump` using the `INTERPOSER:INTERPOSER` cell:

```
create_stitch_zone -cell INTERPOSER:INTERPOSER -name C4Bump /
                   -x 5000 -x_width 100 /
                   -y 1000 -y_width 200 /
                   -c_width 150 -c_height 250
```

Related Commands

- `delete_stitch_zone`
- `update_stitch_zone`

create_substrate

```
create_substrate
[-help]
{-layer_csv csv_file | -name name [-info_tech_file info_tech_file | -lefs lef_files | -metal_stack layer:type:thickness]}
[-type {die | spacer | pkg | board | unknown}]
```

Creates the substrate.

Parameters

`-help`	Prints a brief description for each `create_substrate` parameter. *Data_type*: optional
`-info_tech_file` *`info_tech_file`*	
	Specifies the info tech file to create the substrate. *Data_type*: optional
`-layer_csv` *`csv_file`*	
	Specifies a CSV file to create the substrate. *Data_type*: optional
`-lefs` *`lef_files`*	
	Specifies the LEF files to create the substrate. *Data_type*: optional
`-metal_stack` *`layer:type:thickness`*	
	Defines the metal stack for the substrate. The format is *`layer:type:thickness`*. For example: `-metal_stack {{Metal1:signal:50} {Metal2:signal:50}}` *Data_type*: optional
`-name` *`name`*	Specifies the substrate name. *Data_type*: optional
`-type {die | spacer | pkg | board | unknown}`	
	Specifies the default substrate type. *Data_type*: optional

Examples

- The following command creates a substrate named `PCB` with the specified metal stack:

```
create_substrate -name PCB \
                 -metal_stack {{Metal1:signal:50}{Metal2:signal:50}}
```

- The following command creates a substrate from the specified CSV file:

```
create_substrate -layer_csv layer.csv
```

- The following command creates a substrate from the specified tech file:

```
create_substrate -info_tech_file InFO_3D.tf
```

Related Command

- `delete_substrate`
- `update_substrate`

create_term

```
create_term
[-help]
[-bus startID:endID]
[-direction {input | output | inout | jumper}]
[-ground]
-module module
-name name
[-net net]
[-power]
```

Creates a term with the specified name and module.

Parameters

`-help`	Prints a brief description for each `create_term` parameter. *Data_type*: optional
`-bus` *`startID`*`:`*`endID`*	
	Specifies the bit range for bus terms. The *`startID`* and *`endID`* indicate the first and last bits on the bus. You must separate the start and end IDs with a colon (`:`). *Data_type*: optional
`-direction {input | output | inout | jumper}`	
	Specifics the direction. *Default*: `inout` *Data_type*: enum, optional
`-ground`	Adds a ground term. *Data_type*: optional
`-module` *`module`*	Specifies the name of the module or template for the term. *Data_type*: required
`-name` *`name`*	Specifies the name of the term to be created. *Data_type*: required
`-net` *`net`*	Specifies the net to be used for the term. When `-bus` is specified, `-net` specifies the name of the bus. For example, `A` for the bus `A[0:2]`. *Data_type*: optional
`-power`	Adds a power term. *Data_type*: optional

Examples

- The following command creates the ground term `VSS`:

  ```
  create_term -module Mod -name VSS -ground
  ```

- The following command creates the power bus terms `VDD[0:9]`:

  ```
  create_term -module Mod -name VDD -bus 0:9 -power
  ```

- The following command creates the term `A` with the net `B`:

  ```
  create_term -module Mod -name A -net B
  ```

- The following command creates the bus terms `C[0:4]` with the bus nets `D[0:4]`:

  ```
  create_term -module Mod -name C -net D -bus 0:4
  ```

create_via

```
create_via
[-help]
[-all_same_net_pins]
[-all_unassign]
[-direction_arg direction_arg]
[-line_width line_width]
[-min_channel_space min_channel_space]
[-offset x_y]
[-pin_via_space pin_via_space]
-source_layer layer
-via pad_template
-via_direction {n | s | e | w | ne | nw | se | sw |  inward | outward |
                inout | inpad | bga_quad_inward | bga_quad_outward |
                way2 |  way4 | way8}
```

Creates a via in the specified direction.

Parameters

`-help`	Prints a brief description for each `create_via` parameter. *Data_type*: optional
`-all_same_net_pins`	Includes all same net pins. *Data_type*: optional
`-all_unassign`	Includes unassigned pins. *Data_type*: optional
`-direction_arg` *`direction_arg`*	
	Specifies the arguments of direction, only for way2, way4 and way8. *Data_type*: optional
`-line_width` *`line_width`*	Specifies the width of the wire. *Data_type*: optional
`-min_channel_space` *`min_channel_space`*	
	Specifies the spacing between channels. *Data_type*: optional
`-offset` *`x_y`*	Specifies the offset of the created via location compared with the relative pin. *Data_type*: optional
`-pin_via_space` *`pin_via_space`*	
	Specifies the spacing between the pin and the via. *Data_type*: optional

`-source_layer` *`layer`*	Specifies the source layer of the selected pins. *Default*: The substrate is decided by the selected objects. *`substrate_name`*`:`*`layer_name`* Example: `DIE:MB` *Sugar syntax*: *`layer_name`* Example: `MB` *Data_type*: required
`-via` *`pad_template`*	Specifies the pad template. *Default*: The substrate is decided by the selected objects. *`substrate_name`*`:`*`padstack_name`* Example: `DIE:TSV` *Sugar syntax*: *`padstack_name`* Example: `TSV` *Data_type*: required
`-via_direction {n | s | e | w | ne | nw | se | sw | inward | outward | inout | inpad | bga_quad_inward | bga_quad_outward | way2 | way4 | way8}`	
	Specifies fanout via. *Data_type*: enum, required

Example

- The following command creates a via in pad:
  ```
  create_via -all_same_net_pins -source_layer MB -via TSV -via_direction inpad
  ```

delete_contact_layer

```
delete_contact_layer
[-help]
[-from_device device]
[-from_layer layer]
[-interface_name name]
[-to_device device]
[-to_layer layer]
```

Deletes the specified contact layer.

Parameters

`-help`	Prints a brief description of each `delete_contact_layer` parameter. *Data_type*: optional
`-from_device` *`device`*	Specifies the device from which the contact layer is to be deleted. *Data_type*: optional
`-from_layer` *`layer`*	Specifies the layer from which the contact layer is to be deleted. *Data_type*: optional
`-interface_name` *`name`*	Specifies the unique short name for the contact layer to be deleted. *Data_type*: optional
`-to_device` *`device`*	Specifies the device to which the contact layer is to be deleted. *Data_type*: optional
`-to_layer` *`layer`*	Specifies the layer to which the contact layer is to be deleted. *Data_type*: optional

Examples

- The following command deletes the contact layer from the `/top/ASIC_IPG` device and `AP` layer to the `/top/INTERPOSER` device and `AP` layer:

```
delete_contact_layer -from_device {/top/ASIC_IPG} \
                                  -from_layer AP \
                                  -to_device {/top/INTERPOSER} \
                                  -to_layer AP
```

- The following command deletes the contact layer with the name `inter_chip_f2f`:

```
delete_contact_layer -interface_name inter_chip_f2f
```

Related Commands

- `create_contact_layer`

For deletion of other database objects, see:

- `delete_obj`
- `delete_nets`

delete_contacts

```
delete_contacts
[-help]
[contact_layer]
[-from_device device]
[-from_layer layer]
[-pins]
[-reverse]
[-to_device device]
[-to_layer layer]
```

Deletes existing contacts.

Parameters

`-help`	Prints a brief description for each `delete_contacts` parameter. *Data_type*: optional
`contact_layer`	Specifies the contact layer object instead of a 'from-to' configuration. *Data_type*: optional
`-from_device` *`device`*	Specifies the device from which the contact layer is to be deleted. *Data_type*: optional
`-from_layer` *`layer`*	Specifies the layer from which the contact layer is to be deleted. *Data_type*: optional
`-pins`	Deletes only the contact pins, and not the contact device. If this parameter is not specified, the contact device is deleted by default. *Data_type*: optional
`-reverse`	Specifies the reverse created order of specific contact layer object. *Data_type*: optional
`-to_device` *`device`*	Specifies the device to which the contact layer is deleted. *Data_type*: optional
`-to_layer` *`layer`*	Specifies the layer to which the contact layer is deleted. *Data_type*: optional

Example

- The following command deletes only the contact pins using the 'from-to' configuration:

```
delete_contacts -from_device /Chip1 -from_layer AP \
                                -to_device /INTERPOSER -to_layer AP \
                                -pins
```

Related Commands

- `check_contacts`

- `create_contacts`

delete_current_region

```
delete_current_region
[-help]
[-all]
[-cell cell]
[-name name]
```

Deletes the specified current region.

Parameters

`-help`	Prints a brief description of each `delete_current_region` parameter. *Data_type*: optional
`-all`	Deletes all existing current regions. *Data_type*: optional
`-cell cell`	Specifies the cell owner. *Data_type*: optional
`-name name`	Specifies the name of the current region. *Data_type*: optional

Examples

- The following command deletes the current region named `regionA` on the `INTERPOSER` cell:
  ```
  delete_current_region -cell INTERPOSER -name regionA
  ```
- The following command deletes all the current regions on `INTERPOSER`:
  ```
  delete_current_region -cell INTERPOSER -all
  ```
- The following command deletes all the current regions in the database:
  ```
  delete_current_region -all
  ```

Related Commands

- `create_current_region`
- `update_current_region`

delete_external_contact_layer

```
delete_external_contact_layer
[-help]
-device device
-layer layer
```

Deletes an external contact layer object.

Parameters

`-help`	Prints a brief description for each `delete_external_contact_layer` parameter. *Data_type*: optional
`-device` *`device`*	Specifies the device from which the external contact layer is to be deleted. *Data_type*: required
`-layer` *`layer`*	Specifies the layer on which the external contact layer is to be deleted. *Data_type*: required

Example

- The following command deletes an external contact layer object from the specified device on the `UBMB` layer:
  ```
  delete_external_contact_layer -device /u_info_1 -layer UBMB
  ```

Related Command

- `create_external_contact_layer`

delete_group

```
delete_group
[-help]
-cell cell
-name name
```

Deletes the group with the specified cell and name.

Parameters

`-help`	Prints a brief description for each `delete_group` parameter. *Data_type*: optional
`-cell cell`	Specifies the cell owner. *Data_type*: required
`-name name`	Specifies the name of the group to be deleted. *Data_type*: required

Example

- The following command deletes the group named `par1` with the `DIE1` cell:

```
delete_group -name par1 -cell DIE1
```

Related Command

- `create_group`

delete_marker

```
delete_marker
[-help]
[-all]
[-area area]
[-cell cellName]
```

Deletes markers for violations in the database from the specified area, cell, or the entire design.

Parameters

`-help`	Prints a brief description for each `delete_marker` parameter. *Data_type*: optional
`-all`	Deletes markers from the entire design area. *Data_type*: optional
`-area` *`area`*	Specifies a bounding box for the area from which to delete markers. *Data_type*: optional
`-cell` *`cellName`*	Deletes markers owned by the specified cell. *Data_type*: optional

Examples

- The following command deletes all DRC markers:
  ```
  delete_marker -all
  ```
- The following command deletes the DRC markers owned by the specified cell:
  ```
  delete_marker -cell DieAbstract_2:FCCC_N10_top
  ```
- The following command deletes the DRC markers in the specified area:
  ```
  delete_marker -area {-275 1425 -220 1480}
  ```
- The following command deletes the DRC markers that are in the specified area and are also owned by the specified cell (the area is relative to the cell):
  ```
  delete_marker -cell DieAbstract_2:FCCC_N10_top -area {25 25 75 75}
  ```

Related Command

- `create_marker`

delete_nets

```
delete_nets
[-help]
name
-module module
```

Deletes the net with the specified name and module.

Parameters

`-help`	Prints a brief description of each `delete_nets` parameter. *Data_type*: optional
`name`	Specifies the name of the net to be deleted. *Data_type*: required
`-module` *`module`*	Specifies the name of the module of the net to be deleted. *Data_type*: required

Example

- The following command deletes all the net(s) with prefix `net1` in the module `INTERPOSER`:

  ```
  delete_nets net1* -module INTERPOSER
  ```

Related Commands

- `create_net`

For deletion of other database objects, see:

- `delete_contact_layer`
- `delete_obj`

delete_obj

```
delete_obj
[-help]
[any_object]
[-category {substrate | layer | cell | cell_inst | pin | pin_inst | port | pad | module
    | net | term | grid | device_grid | pin_grid | group | region | current_region | obstacle }]
[-line line]
[-name name]
[-point point]
[-rect rect]
[-selected]
[-type {pin | device | via | finger | obstacle}]
[-visible_filter]
```

Deletes one or more objects, based on the specified parameters.

Parameters

`-help`	Prints a brief description of each `delete_obj` parameter. *Data_type*: optional
`any_object`	Specifies the object or the list of objects to be deleted. *Data_type*: optional
`-category {substrate | layer | cell | cell_inst | pin | pin_inst | port | pad | module | net | term | grid | device_grid | pin_grid | group | region | current_region | obstacle}`	
	Specifies the object type of the option *`any_object`*. *Data_type*: enum, optional
`-line` *`line`*	Selects and deletes the objects through which the specified line passes. *Data_type*: optional
`-name` *`name`*	Specifies the name of the objects to be deleted. *Data_type*: optional
`-point` *`point`*	Selects and deletes the object at the specified point. *Data_type*: optional
`-rect` *`rect`*	Selects and deletes the objects falling within the specified rectangular area. *Data_type*: optional
`-selected`	Deletes the selected objects of the specified type. *Data_type*: optional
`-type {pin | device | via | finger | obstacle}`	
	Specifies the object type you want to delete. *Data_type*: enum, optional
`-visible_filter`	Enables the visible filter. *Data_type*: optional

Examples

- The following command deletes the selected objects of type `device` that meet the requirements specified by the additional parameters:
  `delete_obj -selected -visible_filter -type device -rect {0 0 10000 10000} -name bump*`
- The following command deletes the objects of type `device` that meet the requirements specified by the additional parameters:
  `delete_obj -type device -visible_filter -name HBM_38* -rect {6400 8092 8090 9640}`
- The following command deletes the objects of the `cell_inst` category from the list of objects returned by `get_db`:
  `delete_obj [lindex [get_db INTERPOSER .bumps -category cell] 0] -category cell_inst`
- The following set of commands delete all devices (bump, core, stdcell) under the specific hierarchy /HBM2:
  ```
  select_obj -select_type replace -device_path /HBM2 -type device -include_descendant
  delete_obj -selected
  ```

Related Commands

For deletion of other database objects, see:

- `delete_nets`
- `delete_contact_layer`
- `get_db`
- `select_obj`

delete_padstack

```
delete_padstack
[-help]
-name name
-substrate substrate
```

Deletes the padstack with the specified name.

Parameters

`-help`	Prints a brief description for each `delete_padstack` parameter. *Data_type*: optional
`-name` *`name`*	Specifies the name of the padstack. *Data_type*: required
`-substrate` *`substrate`*	Specifies the substrate for which the padstack is being deleted. *Data_type*: required

Example

- The following command deletes the `PKG_PAD` padstack:

  ```
  delete_padstack -name PKG_PAD -substrate PKG
  ```

Related Commands

- `create_padstack`
- `update_padstack`

delete_region

```
delete_region
[-help]
-cell cell
-name name
```

Deletes the specified region.

Parameters

`-help`	Prints a brief description for each `delete_region` parameter. *Data_type*: optional
`-cell cell`	Specifies the cell owner. *Data_type*: required
`-name name`	Specifies the name of the region. *Data_type*: required

Example

- The following command deletes the specified region:
  `delete_region -name CELL_FENCE -cell INTERPOSER`

Related Commands

- `create_region`

delete_stitch_zone

```
delete_stitch_zone
[-help]
-cell cell
-name name
```

Deletes the specified stich zone.

Parameters

`-help`	Prints a brief description for each `delete_stitch_zone` parameter. *Data_type*: optional
`-cell cell`	Specifies the cell owner. *Data_type*: required
`-name name`	Specifies the name of the stitch zone. *Data_type*: required

Example

- The following command deletes the stitch zone named `C4Bump` with the `INTERPOSER:INTERPOSER` cell:

```
delete_stitch_zone -cell INTERPOSER:INTERPOSER -name C4Bump
```

Related Commands

- `create_stitch_zone`
- `update_stitch_zone`

delete_substrate

```
delete_substrate
[-help]
-name name
```

Deletes the specified substrate.

Parameters

`-help`	Prints a brief description for each `delete_substrate` parameter. *Data_type*: optional
`-name` *`name`*	Specifies the name of the substrate that needs to be deleted. *Data_type*: required

Example

- The following command deletes the specified substrate:

```
delete_substrate -name DIE_300000_Substrate
```

Related Commands

- `create_substrate`
- `update_substrate`

deselect_obj

```
deselect_obj
[-help]
[-all]
[-area_operation {contain | touch}]
[-line line]
[-modifier modifier]
[-point point]
[-prop_name pname]
[-prop_operation {equals | not_equals}]
[-prop_values pvalues]
[-rect rect]
[-substrate substrate]
-type {net | port | pin | device | via | finger | ruler | bundle | connection | wire | obstacle | shape | term }
```

Deselects database objects of the specified type.

Parameters

`-help`	Prints a brief description for each `deselect_obj` parameter. *Data_type*: optional
`-all`	Deselects all the objects of the specified type in the design. *Data_type*: optional
`-area_operation {contain | touch}`	
	Specifies whether objects should be deselected if they are contained within or touching specified area. *Data_type*: enum, optional
`-line` *`line`*	Deselects the objects through which the specified line passes. *Data_type*: optional
`-modifier` *`modifier`*	Specifies the deselection modifier. *Data_type*: optional
`-point` *`point`*	Deselects the object at the specified point. *Data_type*: optional
`-prop_name` *`pname`*	Specifies the property to be used for deselecting objects by property. *Data_type*: optional
`-prop_operation {equals | not_equals}`	
	Specifies whether objects should be deselected if their value equals or does not equal the specified property value. *Data_type*: enum, optional
`-prop_values` *`pvalues`*	Specifies the property values to be used for deselection by property. *Data_type*: optional
`-rect` *`rect`*	Deselects the objects falling within the specified rectangular area. *Data_type*: optional
`-substrate` *`substrate`*	Specifies the substrate from which to deselect the objects. *Data_type*: optional
`-type {net | port | pin | device | via | finger | ruler | bundle | connection | wire | obstacle | shape | term}`	
	Specifies the object type you want to deselect. *Data_type*: enum, required

Example

- The following command deselects the Power and Signal pins that touch the specified rectangular area:
  ```
  deselect_obj -type pin -line {6675 7248 12797 10993} -prop_name netClass -prop_values {Power Signal} -prop_operation equals -area_operation touch
  ```

  The same action can be performed by using the equivalent `select_obj -select_type remove` command as shown below:
  ```
  select_obj -type pin -line {6675 7248 12797 10993} -select_type remove -prop_name netClass -prop_values {Power Signal} -prop_operation equals -area_operation touch
  ```

Related Commands

- clear_selection
- get_selection
- list_select_properties
- select_obj

edit_bulk_nets

```
edit_bulk_nets
[-help]
-mode {rename | create_from_term | create_to_parent}
-new_name new_name_pattern
-path path
-select select_pattern
[-select_filter select_filter]
[-select_insensitive]
```

Edits nets in bulk.

Parameters

`-help`	Prints a brief description for each `edit_bulk_nets` parameter. *Data_type*: optional
`-mode {rename | create_from_term | create_to_parent}`	
	Specifies the the type of edit operation to be performed on the nets. The following modes are allowed: • `rename` - Use this mode to rename nets. This is equivalent to the *Netlist -> Nets -> Bulk rename nets* command in the Netlist Editor. • `create_from_term` - Use this mode to create nets from terms. This is equivalent to the *Netlist -> Terms -> Bulk create nets* command in the Netlist Editor. • `create_to_parent` - Use this mode to create nets from child terms. This is equivalent to the *Netlist -> Children -> Bulk create -> To Parent Nets* command in the Netlist Editor. *Data_type*: enum, required
`-new_name` *`new_name_pattern`*	
	Specifies a pattern for renaming the nets. *Data_type*: required
`-path` *`path`*	Specifies the path of the module instance. *Data_type*: required
`-select` *`select_pattern`*	Specifies a pattern for selecting db objects. *Data_type*: required
`-select_filter` *`select_filter`*	
	Specifies a filter for selected items. See Supported Filters for details on allowed filters. *Data_type*: optional
`-select_insensitive`	Specifies insensitive mode for selecting items. *Data_type*: optional

Supported Filters

Three types of filters are supported:

Net Filters	`is_signal is_pg is_mapped is_unmapped with_a_term with_terms without_term`

Term Filters	`is_signal is_pg is_mapped is_unmapped`
Bump Filters	`is_signal is_pg with_net without_net with_term without_term`

The type of select or match filter that is applied depends on the `-mode` option specified with the `edit_bulk_nets` command:

-mode	**-select_filter**
`rename`	Net
`create_from_term`	Term
`create_to_parent`	Term

Example

- The following command renames the nets for the `/Substrate/BGA/CHIP1` device based on specified parameters:

```
edit_bulk_nets -mode create_to_parent \
    -path /Substrate/BGA/CHIP1 \
    -select {D_(.*)\[(.*)} \
    -new_name {ATest_{1}};
```

Related Command

- `edit_bulk_terms`

edit_bulk_terms

```
edit_bulk_terms
[-help]
[-match match_pattern]
[-match_filter match_filter]
[-match_insensitive]
-mode {rename | map_from_parent_net | map_from_parent_term | create_from_bump | create_from_net | create_to_parent |
       create_from_parent_net | create_from_parent_term}
[-new_name new_name_pattern]
-path path
-select select_pattern
[-select_filter select_filter]
[-select_getter {net | term | parent_net | bump | bump_net}]
[-select_insensitive]
```

Edits terms in bulk.

Parameters

`-help`	Prints a brief description for each `edit_bulk_terms` parameter. *Data_type*: optional
`-match` *`match_pattern`*	Specifies a pattern to match db objects when mapping terms. *Data_type*: optional
`-match_filter` *`match_filter`*	Specifies a filter for matched items when mapping terms. See Supported Filters for details on allowed filters. *Data_type*: optional
`-match_insensitive`	Specifies insensitive mode for matching items when mapping terms. *Data_type*: optional
`-mode {rename | map_from_parent_net | map_from_parent_term | create_from_bump | create_from_net | create_to_parent | create_from_parent_net | create_from_parent_term}`	
	Specifies the type of edit operation to be performed on the terms. See Supported Mode Values for details on allowed modes. *Data_type*: enum, required
`-new_name` *`new_name_pattern`*	Specifies a pattern for renaming the terms. *Data_type*: optional
`-path` *`path`*	Specifies the path of the module instance. *Data_type*: required
`-select` *`select_pattern`*	Specifies a pattern to select db objects. *Data_type*: required
`-select_filter` *`select_filter`*	Specifies a filter for selected items. See Supported Filters for details on allowed filters. *Data_type*: optional
`-select_getter {net | term | parent_net | bump | bump_net}`	
	Specifies a string getter for selected items: • `net` - NetStringGetter • `term` - TermStringGetter • `parent_net` - TermParentNetStringGetter • `bump` - BumpStringGetter • `bump_net` - BumpNetStringGetter *Data_type*: enum, optional
`-select_insensitive`	Specifies the insensitive mode for selecting items. *Data_type*: optional

Supported Mode Values

The table below shows the values that are supported for `-mode`:

`rename`	Renames terms. This is equivalent to the *Netlist -> Terms -> Bulk rename terms* command in the Netlist Editor.
`map_from_parent_net`	Maps terms from parent nets. This is equivalent to the *Netlist -> Children -> Bulk map -> From Parent Nets* command in the Netlist Editor.
`map_from_parent_term`	Maps terms from parent terms. This is equivalent to the *Netlist -> Children -> Bulk map -> From Parent Terms* command in the Netlist Editor.
`create_from_bump`	Creates terms from bumps.
`create_from_net`	Creates terms from nets. This is equivalent to the *Netlist -> Nets -> Bulk create terms* command in the Netlist Editor.
`create_to_parent`	Creates terms from nets. This is equivalent to the *Netlist -> Children -> Bulk create -> To Parent Terms* command in the Netlist Editor.
`create_from_parent_net`	Creates terms from nets. This is equivalent to the *Netlist -> Children -> Bulk create -> From Parent Nets* command in the Netlist Editor.
`create_from_parent_term`	Creates terms from nets. This is equivalent to the *Netlist -> Children -> Bulk create -> From Parent Terms* command in the Netlist Editor.

Supported Filters

Three types of filters are supported:

Net Filters	`is_signal is_pg is_mapped is_unmapped with_a_term with_terms without_term`
Term Filters	`is_signal is_pg is_mapped is_unmapped`
Bump Filters	`is_signal is_pg with_net without_net with_term without_term`

The type of select or match filter that is applied depends on the `-mode` option specified with the `edit_bulk_terms` command:

-mode	**-select_filter**	**-match_filter**

rename	Term	x
map_from_parent_net	Net	Term
map_from_parent_term	Term	Term
create_from_bump	Bump	x
create_from_net	Net	x
create_to_parent	Term	x
create_from_parent_net	Net	x
create_from_parent_term	Term	x

Examples

- The following command maps terms from parent nets:
```
edit_bulk_terms -mode map_from_parent_net \
     -path /Substrate/BGA/CHIP1 \
     -select {(.*)} \
     -match {D_{1}};
```
- The following command maps terms from parent terms with the insensitive option:
```
edit_bulk_terms -mode map_from_parent_term \
     -path /Substrate/BGA/CHIP2 \
     -select {(.*)} \
     -match {D_{1}} \
     -match_insensitive;
```
- The following command creates terms from the net at the top design:
```
edit_bulk_terms -mode create_from_net \
    -path / \
    -select {C4(.*)} \
    -new_name {C4{1}}
```
- The following command creates terms from nets with filter:
```
edit_bulk_terms -mode create_from_net \
     -path /Substrate \
     -select {(.*)} \
     -select_filter is_pg \
     -new_name {{1}};
```

Related Commands

- edit_bulk_nets

eval_bsh

```
eval_bsh
[-help]
java_statements
```

Evaluates the specified beanshell script

Parameters

`-help`	Prints a brief description for each `eval_bsh` parameter. *Data_type*: optional
`java_statements`	Specifies the script in the beanshell. *Data_type*: string, required

Example

- The following commands evaluate the specified Java statements:

```
eval_bsh {System.out.println("hello, java");}
====> hello, java
```

```
 ## create design by beanshell
eval_bsh {
    OrbitIO.createDesign("Design", "micron", 1000000L, 1000000L, 3, true);
}
```

Related Command

- `source_bsh`

get_db

```
get_db
[-help]
start
[chain]
[patterns]
[-category {predefine | program | attribute | substrate | layer | cell | cell_inst | pin | pin_inst
    | port | padstack | module | mod_inst | net | term | term_map | seal_ring | scribe_line | grid
    | device_grid | pin_grid | group | region | current_region | contact_layer | marker
    | external_contact_layer | device_path | mod_path}]
[-dbu]
[-expr expression]
[-if expression]
[-unique]
```

Returns objects and attributes in the database with different types of filtering.

Parameters

`-help`	Prints a brief description for each `get_db` parameter. *Data_type*: optional
start	Specifies the starting objects or root attributes. *Data_type*: required

chain	Specifies the attribute names that are valid for each level of the traversal. Check the chain Attribute Specification section for details.
	You can also use the following commands to get chain attribute help:
	`get_db start chain.help -category category`
	`get_db start chain.?h -category category`
	Data_type: optional

patterns	Specifies string pattern(s). *Data_type*: vararg, optional
`-category {predefine | program | attribute | substrate | layer | cell | cell_inst | pin | pin_inst | port | padstack | module | mod_inst | net | term | term_map | seal_ring | scribe_line | grid | device_grid | pin_grid | group | region | current_region | contact_layer | marker | external_contact_layer | device_path | mod_path}]`	
	Specifies the category of root attributes to return. If you do not specify the `-category` option, the tool determines the category automatically. If *start* is a predefined attribute belonging to a category, such as `program` or `predefine`, you do not need to specify `-category` explicitly, as shown in the examples below: • `get_db cmd_file` • `get_db tcl_cmd_file` • `get_db log_file` • `get_db program_name` • `get_db program_version` If *start* is not a recognized attribute, the tool takes `device_path` as the default category. So, for example, `get_db MyPackage .cell` will mean the same as specifying `get_db MyPackage .cell -category device_path`. *Data_type*: enum, optional
`-dbu`	Indicates that attribute values (`coord`, `pt`, `rect`, `polygon`, `area`, ...) should be returned as an `int` in database units. *Data_type*: optional
`-expr` *expression*	Returns the object if the expression is true. *Data_type*: optional
`-if` *expression*	Specifies a boolean expression with extensions to make pattern matching simpler. *Data_type*: optional
`-unique`	Removes duplicate entries from the return list. If the specified attribute is a list, this parameter returns a unique set of lists, and does not uniquifies the elements within the list. In such a case, use `lsort`. *Data_type*: optional

`start` Specification for the `predefine` Category

`current_design`	Returns the design in the database.
`cmd`	Returns the command.
`selected`	Returns the selected object set.
`attributes`	Returns all the attributes of database objects.
`substrates`	Returns all the substrates.
`cells`	Returns all the cells.
`cell_insts`	Returns all the cell instances.
`terms`	Returns all the terms.
`nets`	Returns all the nets.
`modules`	Returns all the modules.
`mod_insts`	Returns all the module instances.

chain Attribute Specification

For the `predefine` Category

Design chain attribute specification

Attribute	Description
	Returns the key string of cells.
`.design_name`	Returns the design name.
`.top_designs`	Returns the top designs in the cell_inst type.
`.substrates`	Returns substrates.
`.contact_layers`	Returns the contact layers in tuple string.
`.region_paths`	Returns the region paths.
`.external_contact_layers`	Returns the external contact layers.
`.dbu`	Returns the dbu of the current design. The unit per distance in integer type.
`.decimal_places`	Returns the decimal places of the current design.
`.history`	Returns the db history.

For the `attribute` Category

Attribute	Description
`.base_name`	Returns the base name of the attribute.
`.name`	Returns the name of the attribute with the category name.
`.category`	Returns the category of the attribute for get_db.
`.is_settable`	Returns whether or not the attribute is settable. *Data_type*: boolean

For the `cell` Category

Attribute	Description
	Returns the key string of cells.
`.type`	Returns the type of cells. Type can be: `die pkg board spacer ip_component mechanical cover ring block pad core endcap hardmacro softmacro bump stdcell contact_device bondfinger group logical unknown`
`.source_type`	Returns the source type of cells. Source type can be: `unknown lefdef lef def verilog upd apd spd orbitio_wirebonder orbitio_contact user milkyway bumpfactory spreadsheet diebgatxt dieabstractxml fileimport`
`.name`	Returns the name of the cell.
`.substrate`	Returns the name of the substrates of the cell.
`.regions`	Returns the regions for the cell. For more details, see the definition of regions in the *DEF Syntax* chapter in *LEF/DEF Language Reference*.
`.groups`	Returns the groups for the cell. For more details, see the definition of groups in the *DEF Syntax* chapter in *LEF/DEF Language Reference*.
`.module`	Returns the module of the cell.
`.bbox`	Returns the bounding box of the cell in rectangle type.
`.current_regions`	Returns the current regions of the cell owner.
`.region_paths`	Returns the region paths.
`.nets`	Returns the nets of the cell.

`.terms`	Returns the terms of the cell.
`.children`	Returns the child devices in the cell inst type.
`.bumps`	Returns the bumps of the cell in the cell inst type.
`.pins`	Returns the pins of the cell in the pin type.
`.device_grids`	Returns the device grids of the cell.
`.pin_grids`	Returns the pin grids of the cell.
`.seal_ring`	Returns the seal ring of the cell.
`.scribe_line`	Returns the scribe line of the cell.
`.insts`	Returns the cell instances which were instantiated by of the cell.
`.markers`	Returns the violation markers of the cell owner.
`.polygon`	Returns the outline shape in polygon.

For the `cell_inst` Category

Attribute	Description
	Returns the key string of cell instances.
`.name`	Returns the name of cell instances.
`.cell`	Returns the cell of instances. It can be chained with the attributes of the `cell` category.
`.parent`	Returns the parent cell who owns this instance.
`.substrate`	Returns the name of substrates of the cell inst.
`.bbox`	Returns the bounding box of the cell inst.
`.location`	Returns the location of the cell inst.
`.orientation`	Returns the orientation of the cell inst.
`.face_up`	Returns the face up of the cell inst.

`.type`	Returns the type of the cells. Type can be:
	`die pkg board spacer ip_component mechanical cover ring`
	`block pad core endcap hardmacro softmacro bump stdcell`
	`contact_device bondfinger group logical unknown`

`.nets`	Returns the nets of the cell.
`.terms`	Returns the terms of the cell.
`.children`	Returns the child devices in the cell inst type.
`.bumps`	Returns the bumps of the cell in the cell inst type.
`.connect_nets`	Returns the tuples of connected nets under the parent cell. The tuples: `{{logical_path net_name}*}` **Note**: *`logical_path`* means *`mod_path`*.
`.connect_terms`	Returns the tuples of connected terms under the parent cell. The tuples: `{{logical_path term_name}*}` **Note**: *`logical_path`* means *`mod_path`*.
`.polygon`	Returns the transformed outline polygon relative to the parent cell.
`.place_status`	Returns the placement status of cell inst.

For the `contact_layer` Category

Attribute	Description
`.contact_pin_height`	Returns the contact pin height of the contact layer
`.underfill_material`	Returns the underfill material of the contact layer
`.bump_material`	Returns the bump material of the contact layer

For the `current_region` Category

Attribute	Description
`.name`	Returns the name of the current region.
`.bbox`	Returns the bounding box of the current region.
`.total_power`	Returns the total power of the current region in W.
`.current_nets`	Returns the current net configs.

For the `device_grid` Category

Attribute	Description
`.name`	Returns the name of the device grid.
`.grid`	Returns the grid object of the device grid in the grid type.

For the `device_path` Category

Attribute	Description
	Returns the string of the absolute physical device path.
`.cell`	Returns the cell of the last cell in the path.
`.cell_inst`	Returns the cell instance of the last cell in the path.
`.module`	Returns the module of the last cell in the path.
`.mod_path`	Returns the module path representation.
`.substrate`	Returns the substrate of the device path.
`.bbox`	Returns the bounding box of this path.
`.z`	Returns the `z` location in design stacking.
`.tier`	Returns the tier number in design stacking.
`.connect_nets`	Returns the tuples of connected nets under the path root cell. The tuples:`{{logical_path net_name}*}` **Note**: *`logical_path`* means *`mod_path`*.
`.connect_terms`	Returns the tuples of connected terms under the path root cell. The tuples:`{{logical_path term_name}*}` **Note**: *`logical_path`* means *`mod_path`*.
`.location`	Returns the world location of the device's origin.

For the `external_contact_layer` Category

Attribute	Description
`.device`	Returns the device of the external contact layer.
`.layer`	Returns the layer of the external contact layer.
`.height`	Returns the height of the external contact layer.

For the `grid` Category

Attribute	Description
`.name`	Returns the name of the substrate-based grid.
`.grid`	Returns the grid object of the substrate-based grid in the grid type.

For the `group` Category

Attribute	Description
`.name`	Returns the name of the group.

For the `history` Category

Attribute	Description
`.max_history`	Returns the maximum history items for undo/redo.

For the `layer` Category

Attribute	Description
`.name`	Returns the name of the layer.
`.type`	Returns the type of the layer. The type of the layer can be: `unknown signal power mixed jumper dielectric artwork route overlap masterslice implant placebuoundtop placeboundbottom`
`.thickness`	Returns the thickness of the layer in the user unit.
`.height`	Returns the height of the layer in user unit.
`.metal_material`	Returns the metal material of the layer.
`.dielectric_material`	Returns the dielectric material of the layer.
`.metal_percentage`	Returns the metal percentage of the layer.

For the `marker` Category

Attribute	Description
`.bbox`	Returns the bounding box of the marker shape.
`.layers`	Returns the layers of the marker.
`.type`	Returns the type of the marker.
`.subtype`	Returns the subtype of the marker.
`.polygon`	Returns the polygon boundary of the marker.

For the `mod_inst` Category

Attribute	Description
`.name`	Returns the name of the module instance.
`.module`	Returns the module of the module instance.
`.nets`	Returns the nets of the module instance.
`.terms`	Returns the terms of the module instance.
`.children`	Returns the children of the module instance.

For the `mod_path` Category

Attribute	Description
`.cell`	Returns the cell of the last cell in the path.
`.cell_inst`	Returns the cell instance of the last cell in the path.
`.module`	Returns the module of the last cell in the path.
`.device_path`	Returns the device path.
`.connect_nets`	Returns the tuples of connected nets under the path root cell. The tuples:`{{logical_path net_name}*}` **Note**: *`logical_path`* means *`mod_path`*.
`.connect_terms`	Returns the tuples of connected terms under the path root cell. The tuples:`{{logical_path term_name}*}` **Note**: *`logical_path`* means *`mod_path`*.
`.substrate`	Returns the substrate of the device path.
`.bbox`	Returns the bounding box of this path.
`.z`	Returns the z location in design stacking.
`.tier`	Returns the tier number in design stacking.
`.location`	Returns the world location of the device's origin.
`.face_up`	Returns the face up of the device path.
`.orientation`	Returns the orientation of the device path.

For the `module` Category

Attribute	Description
`.name`	Returns the name of the module.
`.nets`	Returns the nets of the module.
`.terms`	Returns the terms of the module.
`.children`	Returns the children of the module inst.

For the `net` Category

Attribute	Description
`.name`	Returns the name of the net.
`.use`	Returns the use of the net. The use of the net can be: `unknown analog clock ground power reset scan signal tieoff nc`
`.module`	Returns the module of the net.
`.terms`	Returns the terms of the net.
`.child_term_maps`	Returns the term mapping objects with child module instances. See category`term_map`.
`.descendant_nets`	Return the tuples of the connected descendant nets. The tuples: `{{logical_path net_name}*}` **Note**: *`logical_path`* means *`mod_path`*.
`.descendant_pins`	Return the tuples of the connected descendant pins. The tuples: `{{physical_path net_name}*}` **Note**: *`physical_path`* means *`device_path`*.

For the `padstack` Category

Attribute	Description
`.name`	Returns the name of the padstack.
`.substrate`	Returns the substrate of the padstack.
`.layers`	Returns the used layers of the padstack.
`.stack`	Returns the shape information in the form of a string expression: `{ {layername:shape:shape_arg} * }`

For the `pin` Category

Attribute	Description
`.name`	Returns the name of the pin.
`.net`	Returns the net of the pin.
`.type`	Returns the type of the pin. The type of the pin can be: `iopad wireboundpad bumppad ballpad bondfingerpad via tsv tdv core contact mechanical unknown`
`.bbox`	Returns the bounding box of the pin (include all ports).

For the `pin_grid` Category

Attribute	Description
`.name`	Returns the name of the pin grid.
`.grid`	Returns the grid object of the pin grid in the grid type.

For the `port` Category

Attribute	Description
`.bbox`	Returns the bounding box of the port.
`.padstack`	Returns the padstack of the port.
`.layer_shapes`	Returns the layer shapes of the port.

layer shape chain Attribute Specification

Attribute	Description
`.bbox`	Returns the bounding box of the port.
`.layer`	Returns the layer of the layer shape.
`.shape`	Returns the shape information in a string expression. `{ {shape:shape_arg} * }`

For the `program` Category (`root` Attribute)

Attribute	Description
`cmd_file`	Specifies the cmd log file name of the program. Data_type: string
`tcl_cmd_file`	Specifies the tcl cmd log file name of the program. Data_type: string
`log_file`	Specifies the log file name of the program. Data_type: string
`program_major_version`	Returns the major version and one digit after the decimal point. Data_type: string
`program_name`	Specifies the name of the program. Data_type: string
`program_short_name`	Specifies the short name of the program. Data_type: string
`program_version`	Specifies the version of the program. Data_type: string

For the `region` Category

Attribute	Description
`.name`	Returns the name of the region.
`.bbox`	Returns the bounding box of the region.

For the `region_path` Category

Attribute	Description
`.name`	Returns the name of the region path.
`.nodes`	Returns the nodes of the path.

node chain Attribute Specification

Attribute	Description
`.contact_region`	Returns the contact region of the node.
`.device_path`	Returns the device path of the node.
`.restrict`	Returns whether the node is restricted.

For the `scribe_line` Category

Attribute	Description
`.name`	Returns the name of the scribe line.
`.bbox`	Returns the bounding box of the scribe line.
`.north`	Returns the north spacing of the scribe line.
`.south`	Returns the south spacing of the scribe line.
`.west`	Returns the west spacing of the scribe line.
`.east`	Returns the east spacing of the scribe line.

For the `seal_ring` Category

Attribute	Description
`.name`	Returns the name of the seal ring.
`.bbox`	Returns the bounding box of the seal ring.
`.north`	Returns the north spacing of the seal ring.
`.south`	Returns the south spacing of the seal ring.
`.west`	Returns the west spacing of the seal ring.
`.east`	Returns the east spacing of the seal ring.

For the `substrate` Category

Attribute	Description
`.name`	Returns the name of the substrate.
`.height`	Returns the height of this substrate in user units.
`.shrink_factor`	Returns the shrink factor in double type. Default:`1.0`
`.type`	Returns the substrate type. The type of the substrate can be: `die spacer pkg board unknown`
`.source_type`	Returns the source type. The source of the substrate can be: `upd apd lefdef milkway spreadsheet user orbitio_wirebonder orbitio_contact aif spb bumpfactory diebgatxt verilog dieabstractxml fileimport`
`.lefs`	Returns the LEF files of the substrate.
`.source`	Returns the source file of the substrate. If `source_type == lefdef`, the source file would be tech LEF file.
`.layers`	Returns the layers of the substrate.
`.padstacks`	Returns padstacks of substrate. See categorypadstack.
`.tech_mfg_grid`	Returns the manufacturing grid of the substrate by default.
`.cells`	Returns the cells of the substrate.

For the `term` Category

Attribute	Description
`.name`	Returns the name of the term.
`.use`	Returns the use of the term. The use of the term can be: `signal analog power ground clock reset scan tieoff`
`.net`	Returns the net of the term.
`.type`	Returns the type of the term. The type of the term can be: `in out inout jumper`
`.module`	Returns the module of the term.

For the `term_map` Category

Attribute	Description
`.mod_inst`	Returns the connected child module instance.
`.term`	Returns the connected term on the child module instance.
`.parent_net`	Returns the parent net of this mapping.

Others

- The `rect` type has built-in sub-attributes like:

Attribute	Description
`.area`	Area of rectangle
`.ll`	Lower-left point
`.ur`	Upper-right point
`.width`	dx
`.length`	dy

- A `point` has built-in sub-attributes like:

Attribute	Description
`.x`	X coordinate
`.y`	Y coordinate

- The `grid` type has built-in sub-attributes like:

Attribute	Description
`.x`	Origin X coordinate of the grid
`.y`	Origin Y coordinate of the grid
`.dx`	Space X coordinate of the grid
`.dy`	Space Y coordinate of the grid

- Hierarchical instance (`hier_inst`) attributes:

Attribute	Description
`.obj`	Returns the db object.
`.path`	Returns the physical path.
`.mod_path`	Returns the logical module path.
`.bbox`	Returns the bounding box.

- Attributes that can be used with `selected`

Attribute	Description
`.objs`	Returns the db object of hierarchical instances. (`.hier_insts.obj`)
`.hier_insts`	Returns hierarchical instances.
`.category`	Returns the category for selected based on `get_db -category`.

Examples

Examples of getting database objects

- The following command returns all the substrates in the design:
  `get_db substrates`
  The output is:
  `{PKG DIE INTERPOSER}`
- The following command returns all the cells in the design:
  `get_db cells`
  The output is:
  `{PKG:pkg DIE:DIE DIE:BUMP INTERPOSER:INTERPOSER}`
- The following command returns all the cell instances in the design:
  `get_db cell_insts`
  The output is:
  `{DIE:DIE:BUMP1 DIE:DIE:BUMP2}`

Examples of getting *`chain`* help

- The following command can be used to get help on attributes of the module of cell `MyPackageTemplate`:
  `get_db -category module [get_db -category cell MyPackageTemplate .module] .help`
  The output is:
  ```
  20:08:52 Info: Available attributes of 'Module 'PKG_MyPackageTemplate'':
  name
  nets
  terms
  ```
- The following command gets chain help on the top designs:
  ```
  get_db current_design .top_designs.?h
  14:00:14 I: Available attributes of `Device INTERPOSER (INTERPOSER)`:
  cell
  name(settable)
  substrate
  ...
  14:00:14 I: Available attributes of `Device ASIC (ASIC)`:
  cell
  name(settable)
  ...
  ```
- The following command can be used to get help on attributes of the cell of the INTERPOSER device:
  ```
  get_db INTERPOSER .cell.help
  14:03:18 I: Available attributes of `DeviceTemplate 'INTERPOSER' on [Substrate 'INTERPOSER']`:
  substrate
  module
  name(settable)
  ...
  ```

Examples of the `predefine` Category

- The following command returns the names of all public commands in the `planner::` namespace:
  `get_db cmd .names`
  The output is:
  `align_contacts align_device_placement assign_pg_bumps assign_terms check_cell_uniqueness ...`
- The following command returns all namespaces:
  `get_db cmd .namespaces`

The output is:

```
{pdn sta}
```

- The following command returns the current design name:

  ```
  get_db current_design .design_name
  ```

  Sample output:

  ```
  3DTOP
  ```

- The following command returns the names of all instances for top designs:

  ```
  get_db current_design .top_designs.name
  ```

  Sample output:

  ```
  {HBM1 HBM2 ASIC INTERPOSER PKG}
  ```

- The following command returns the names of all templates for top designs:

  ```
  get_db current_design .top_designs.cell.name
  ```

  Sample output:

  ```
  {HBM HBM ASIC INTERPOSER PKG}
  ```

- The following command returns the contact layers:

  ```
  get_db current_design .contact_layers
  ```

  Sample output:

  ```
  {{/ASIC_IPG AP /INTERPOSER AP}{/JESD2358B-HBM_Ballout Bump /INTERPOSER AP}}
  ```

Examples of the `attribute` Category

- The following command returns unique attributes:

  ```
  get_db attributes .category -unique
  ```

  Output:

  ```
  cell cell_inst ...
  ```

- The following command returns whether or not the name attribute of a cell is settable:

  ```
  get_db attribute:cell/name .is_settable -category attribute
  ```

  Output:

  ```
  true
  ```

Examples of the `cell` Category

- The following command returns the cell type:

  ```
  get_db INFO .type -category cell
  ```

  Sample output:

  ```
  {pkg}
  ```

- The following command returns the cell type:

  ```
  # $cell: the name/primary key string of cell
  get_db $cell .type -category cell
  ```

  Sample output:

  ```
  {die}
  ```

- The following command returns the cell source type:

  ```
  get_db DIE1 .source_type -category cell
  ```

  Sample output:

  ```
  {lefdef}
  ```

- The following command returns the cell substrate:

  ```
  get_db FCCC_N5 .substrate -category cell
  ```

  Sample output:

  ```
  {N5}
  ```

Examples of the `cell_inst` Category

- The following command returns the name of the cell instance:
  ```
  # $cell_inst: primary key string of Cell Instance
  get_db $cell_inst .name -category cell_inst
  ```
  Sample output:
  ```
  {HBM1}
  ```
- The following command returns the cell's name from the instance:
  ```
  # $cell_inst: primary key string of Cell Instance
  get_db $cell_inst .cell.name -category cell_inst
  ```
  Sample output:
  ```
  {HBM}
  ```
- The following command returns the cell substrate:
  ```
  get_db FCCC_N5_inst .substrate -category cell_inst
  ```
  Sample output:
  ```
  {N5}
  ```

Examples of the `device_path` Category

- The following command returns the key string of the last cell in the path:
  ```
  get_db /MyPackage .cell -category device_path
  ```
  Sample output:
  ```
  PKG:MyPackageTemplate
  ```
- The following command returns the name of the last cell in the path:
  ```
  get_db /MyPackage .cell.name -category device_path
  ```
  Sample output:
  ```
  MyPackageTemplate
  ```
- The following command returns the substrate name of the device path:
  ```
  get_db /MyPackage .substrate -category device_path
  ```
  Sample output:
  ```
  PKG
  ```

Examples of the `mod_path` Category

- The following command returns the tuples of the connected terms under the specified path root cell:
  ```
  get_db /INTERPOSER/ASIC_A1 -category mod_path .connect_terms
  ```
  Sample output:
  ```
  {{/INTERPOSER/ASIC_A1 PAD}{/INTERPOSER VSS}{/ASIC_IPG VSS}{/JESD235B-HBM_Ballout VSS}
   {/INTERPOSER/HBM_4867 PAD}{/INTERPOSER/HBM_4866 PAD} ...}
  ```
- The following command returns the tuples of the connected nets under the specified path root cell:
  ```
  get_db /INTERPOSER/ASIC_A1 -category mod_path .connect_nets
  ```

Examples of using `.child_term_maps` with the `net` Category

- The following command gets the connected child module instance name from the top design net:
  ```
  get_db Design:ASIC_PAD_ARFUa1 .child_term_maps.mod_inst.name -category net
  ```
  Output:
  ```
  {INTERPOSER ASIC_IPG}
  ```
- The following command gets the connected child term name from the top design net:
  ```
  get_db Design:ASIC_PAD_ARFUa1 .child_term_maps.term.name -category net
  ```
  Output:
  ```
  {ASIC_PAD_ARFUa1 PAD_ARFUa1}
  ```

Example of the `padstack` Category

- The following command returns the shape information of the padstack in string format:
  ```
  get_db PKG:PadTemplate_MyPackage -category padstack .stack
  ```
  Output:
  ```
  {Ball:Circle:1000}
  ```

Example of the `program` Category

- The following command returns the tcl cmd log file name of the program:
  `get_db tcl_cmd_file -category program`

Example of the `substrate` Category

- The following command returns the height of the substrate:
  `get_db INTERPOSER .height -category substrate`
  Sample output:
  `110.0000`

Examples of using `-expr`

- The following command returns the layers of the substrate for which the specified expression is true:
  `get_db PKG .layers -expr {$obj(.name) eq "Bump"} -category substrate`
- The following command returns the children in the current design for which the specified expression is true:
  `get_db current_design .children -expr {$obj(.cell.name) == "MyPackageTemplate"}`

Example of using *`pattern`* matching

- The following command returns pins with names matching the specified pattern:
  `get_db MyPackageTemplate -category cell .pins.name A? B*`
  Sample output:
  `A1 A2 B1 B12`

Examples of using `-if`

- The following commands show how to usethe `-if` option:
  - `get_db current_design .nets -if {.name == BUS[0]}`
  - `get_db PKG .layers -if {.name == B* && .name != Ba*} -category substrate`
  - `get_db PKG .layers -if {(.name == Bum* || .name == Ba*) && .type == signal} -category substrate`
  - `get_db cell_insts -if {.cell.type == pkg}`

Example of using selected

- The following command returns the db object of the hierarchical instances of the selected object set:
  `get_db selected .hier_insts.obj`

Related Commands

- `delete_obj`
- `set_db`

get_obj_in_area

```
get_obj_in_area
[-help]
[-areas areas]
[-dbu]
[-enclosed_only]
[-layers layer]
-obj_type {port | pin | inst | bump}
[-polygons polygons]
[-substrate substrate]
```

Returns a list of objects based on an area query without needing to select them. The output format is a string list of `{path obj_key}`.

Parameters

`-help`	Prints a brief description for each `get_obj_in_area` parameter. *Data_type*: optional
`-areas` *`areas`*	Specifies the search areas. *Data_type*: optional
`-dbu`	Specifies that the `-areas` values are in database units rather than microns. *Data_type*: optional
`-enclosed_only`	Returns only those objects, which are completely enclosed inside the search area. The object polygon should be fully enclosed by the search area. *Data_type*: optional
`-layers` *`layer`*	Specifies the layers from which to search the objects, using the `-substrate` definition. *Data_type*: optional
`-obj_type {port | pin | inst | bump}`	
	Specifies the list of object types to be returned that overlap the specified search area. *Data_type*: required
`-polygons` *`polygons`*	Specifies a polygon search area. *Data_type*: optional
`-substrate` *`substrate`*	
	Specifies the substrate from which to search the objects. *Data_type*: optional

Examples

- The following command searches for bumps in the specified search area:
  ```
  get_obj_in_area -obj_type bump -areas {{10503.358732 7094.568419 10583.092863 7155.717903}}
  ```
  Output:
  ```
  {/top/INTERPOSER INTERPOSER:INTERPOSER:BUMP0}{/top/INTERPOSER INTERPOSER:INTERPOSER:ASIC_A215}
  ```
- The following command searches for bumps in the specified layers in the search area:
  ```
  get_obj_in_area -obj_type bump -areas {{10503.358732 7094.568419 10583.092863 7155.717903}} -substrate INTERPOSER -layers
  {AP RV}
  ```
  Output:
  ```
  {/top/INTERPOSER INTERPOSER:INTERPOSER:ASIC_A215}
  ```

Related Commands

- `clear_selection`
- `deselect_obj`
- `select_obj`

get_property

```
get_property
[-help]
obj_list
property_name
-cell cell
-type {device | pin | net}
```

Gets the personality (user-defined properties) of nets, devices, or pins.

Parameters

`-help`	Prints a brief description for each `get_property` parameter. *Data_type*: optional
`obj_list`	Specifies the list of objects for which a property is to be retrieved. *Data_type*: required
`property_name`	Specifies the name of the property. *Data_type*: required
`-cell` *`cell`*	Specifies the owner cell of the property definition. *Data_type*: required
`-type {device | pin | net}`	
	Specifies the type of the object. *Data_type*: enum, required

Examples

- The following commands set and get the `net_group` property for the specified nets:

  ```
  set_property -cell DIE1 -type net {NET1 NET2 NET3} net_group n1
  get_property -cell DIE1 -type net {NET1 NET2 NET3} net_group
  ```

  Output: `n1 n1 n1`

- The following commands set and get the `net_group` property for the specified nets:

  ```
  set_property -cell DIE1 -type net {NET1} net_group n2
  get_property -cell DIE1 -type net {NET1 NET2 NET3} net_group
  ```

  Output: `{n1 n2} {n1} {n1}`

- The following commands set and get the `device_group` property for the `Bump_1`, `Bump_2`, and `Bump_3` devices:

  ```
  set_property -cell DIE1 -type device {Bump_1 Bump_2 Bump_3} device_group d1
  get_property -cell DIE1 -type device {Bump_1 Bump_2 Bump_3} device_group
  ```

  Output: `d1 d1 d1`

Related Commands

- `read_property`
- `set_property`

get_selection

```
get_selection
[-help]
```

Fetches the current selection of database objects.

Parameters

`-help`	Prints a brief description for each `get_selection` parameter. *Data_type*: optional

Related Commands

- `clear_selection`
- `deselect_obj`
- `select_obj`

get_transform_shapes

```
get_transform_shapes
[-help]
[-db_units]
{-global_pt points_in_global | -local_pt points_in_local}
-path device_path
```

Takes coordinates local to a cell and returns them in context with the global design space and vice versa. This command returns a string list of xy coordinates.

Parameters

`-help`	Prints a brief description for each `get_transform_shapes` parameter. *Data_type*: optional
`-db_units`	Specifies values in database units. If this option is not specified, the command specifies values in microns. *Data_type*: optional
`-global_pt` *`points_in_global`*	
	Specifies pairs of global points to be converted into local coordinates. **Note**: You must specify either this parameter or the `-local_pt` parameter. *Data_type*: optional
`-local_pt` *`points_in_local`*	
	Specifies pairs of local points to be converted into global coordinates. **Note**: You must specify either this parameter or the `-global_pt` parameter. *Data_type*: optional
`-path` *`device_path`*	Specifies the device path. *Data_type*: required

Examples

- The following command gets global points from local points:
  ```
  get_transform_shapes -path /SoC_1 -local_pt {547.5 1028.5}
  ```
  The software returns the following information:
  ```
  1222.5 403.5
  ```
- The following command gets global points in DB units from local points:
  ```
  get_transform_shapes -path /SoC_1 -local_pt {547.5 1028.5} -db_units
  ```
  The software returns the following information:
  ```
  1222500000 403500000
  ```
- The following command gets local points from global points:
  ```
  get_transform_shapes -path /SoC_1 -global_pt {1222.5 403.5}
  ```
  The software returns the following information:
  ```
  547.5 1028.5
  ```

group

```
group
[-help]
-name group_name
object+
[-type {physical | logical}]
```

Groups the listed objects under the specified group name.

Parameters

`-help`	Prints a brief description for each `group` parameter. *Data_type*: optional
`-name` *`group_name`*	Specifies the name of the new module/device that groups the specified objects. *Data_type*: required
`object+`	Specifies the device or module paths within the same hierarchy that are to be grouped. *Data_type*: required
`-type {physical | logical}`	
	Specifies the grouping type. *Default*: `logical` *Data_type*: optional

Examples

- The following command creates a logical group named `logical`:
  `group {/DIE_AC/pin_1 /DIE_AC/pin_2} -name logical`
- The following command creates a physical group named `physical`:
  group {/DIE_AC/pin_3 /DIE_AC/pin_4} -name physical -type physical

Related Commands

- `create_group`
- `delete_group`
- `ungroup`

gui_canvas

```
gui_canvas
[-help]
[-off]
[-on]
```

Enables or disables canvas drawing.

Parameters

`-help`	Prints a brief description for each `gui_canvas` parameter. *Data_type*: optional
`-off`	Disables canvas drawing. *Data_type*: optional
`-on`	Enables canvas drawing. *Data_type*: optional

Example

- The following command disables canvas drawing:
  ```
  gui_canvas -off
  ```

Related Commands

- `gui_fit`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `gui_zoom`
- `refresh_canvas`

gui_fit

```
gui_fit
[-help]
```

Fits the entire design in the viewable window.

Parameters

`-help`	Prints a brief description for each `gui_fit` parameter. *Data_type*: optional

Example

- The following command fits the design in the window:
  `gui_fit`

Related Commands

- `clear_all_rulers`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `gui_zoom`
- `refresh_canvas`

gui_hide

```
gui_hide
[-help]
```

Hides the System Planner GUI.

Parameters

`-help`	Prints a brief description for each `gui_hide` parameter. *Data_type*: optional

Example

- The following command hides the System Planner GUI:
  `gui_hide`

Related Commands

- clear_all_rulers
- gui_fit
- gui_redo
- gui_set_visible_layer
- gui_set_visible_substrate
- gui_show
- gui_undo
- gui_zoom
- refresh_canvas

gui_redo

```
gui_redo
[-help]
```

Returns the design to the state it was in immediately prior to issuing the gui_undo command.

Parameters

-help	Prints a brief description for each gui_redo parameter. *Data_type*: optional

Example

- The following command returns the design to the state it was in immediately prior to issuing the gui_undo command.

  ```
  gui_redo
  ```

Related Commands

- clear_all_rulers
- gui_fit
- gui_hide
- gui_set_visible_layer
- gui_set_visible_substrate
- gui_show
- gui_zoom
- gui_undo
- refresh_canvas

gui_set_obj_color

```
gui_set_obj_color
[-help]
[-color color_name]
-obj_type {board | pkg | interposer | die | pkg_die
       | core | pad | cover | ring | block | endcap | bump | non_substrate_inst
       | substrate_inst | pin | term | via | finger | metal | wire | obs | connection
       | current_region | region | guideline | drc_marker | seal_ring | scribe_line
       | stitch_zone | core_area | user_logic_area | ruler}
[-pattern {none | solid | check_fill | check_fill2 | diag_hatch | diamond_s
       | diamond | dotted5 | dotted10 | horiz_dash | horiz | horiz_brick
       | shingle | weave | ortho_hatch | peak | plaid | plus | slant_left
       | slant_right | vert_dash | vert | zig_zag}]
```

Sets the object color.

Parameters

`-help`	Prints a brief description for each `gui_set_obj_color` parameter. *Data_type*: optional
`-color` *`color_name`*	Specifies the color name. *Data_type*: optional
`-obj_type {board | pkg | interposer | die | pkg_die | core | pad | cover | ring | block | endcap | bump | non_substrate_inst | substrate_inst | pin | term | via | finger | metal | wire | obs | connection | current_region | region | guideline | drc_marker | seal_ring | scribe_line | stitch_zone | core_area | user_logic_area | ruler}`	
	Specifies the type of the objects for which color is being set. *Data_type*: enum, required
`-pattern {none | solid | check_fill | check_fill2 | diag_hatch | diamond_s | diamond | dotted5 | dotted10 | horiz_dash | horiz | horiz_brick | shingle | weave | ortho_hatch | peak | plaid | plus | slant_left | slant_right | vert_dash | vert | zig_zag}`	
	Specifies the pattern to be used. *Data_type*: enum, optional

Examples

- The following command sets the color of pins to`red`:
  ```
  gui_set_obj_color -obj_type pin -color red -pattern solid
  ```
- The following command colors the pins to the specified color:
  ```
  gui_set_obj_color -obj_type pin -color (0,128,0,128)
  ```
- The following command colors the pins using the specified pattern:
  ```
  gui_set_obj_color -obj_type pin -pattern dotted5
  ```

Related Commands

- clear_all_rulers
- gui_hide
- gui_fit
- gui_redo
- gui_set_visible_layer
- gui_set_visible_substrate
- gui_show
- gui_undo
- gui_zoom
- refresh_canvas

gui_set_visible_layer

```
gui_set_visible_layer
[-help]
-name name
[-off]
[-on]
-substrate substrate
```

Sets the visible layers.

Parameters

-help	Prints a brief description for each gui_set_visible_layer parameter. *Data_type*: optional
-name *name*	Specifies the name of the layer. *Data_type*: required
-off	Turns visibility off for the specified layer. *Data_type*: optional
-on	Turns visibility on for the specified layer. *Data_type*: optional
-substrate *substrate*	
	Specifies the substrate. *Data_type*: required

Examples

- The following command turns visibility on for the `AP` layer on the `DIE` substrate:

  ```
  gui_set_visible_layer -substrate DIE -name AP -on
  ```

- The following command turns visibility off for the `AP` layer on the `DIE` substrate:

  ```
  gui_set_visible_layer -substrate DIE -name AP -off
  ```

Related Commands

- clear_all_rulers
- gui_hide
- gui_fit
- gui_redo
- gui_set_visible_substrate
- gui_show
- gui_undo
- gui_zoom
- refresh_canvas

gui_set_visible_obj

```
gui_set_visible_obj
[-help]
[-all]
[-obj_type {board | pkg | interposer | die | pkg_die | core | pad | cover | ring | block | endcap | bump
 | non_substrate_inst | substrate_inst | pin | pin_ground | pin_power | pin_nc | term | via | finger
 | metal | wire | obs | connection | current_region | region | guideline | drc_marker | seal_ring
 | scribe_line | stitch_zone | core_area | user_logic_area | ruler}]
[-off]
[-on]
```

Sets the visibility of objects.

Parameters

`-help`	Prints a brief description for each `gui_set_visible_obj` parameter. *Data_type*: optional
`-obj_type {board | pkg | interposer | die | pkg_die | core | pad | cover | ring | block | endcap | bump | non_substrate_inst | substrate_inst | pin | pin_ground | pin_power | pin_nc | term | via | finger | metal | wire | obs | connection | current_region | region | guideline | drc_marker | seal_ring | scribe_line | stitch_zone | core_area | user_logic_area | ruler}`	
	Specifies the type of the objects to be made visible. *Data_type*: enum, optional
`-off`	Hides or turns off visibility for the selected objects on the canvas. *Data_type*: optional
`-on`	Turns visibility on for the selected objects on the canvas. *Data_type*: optional

Examples

- The following command hides all objects of type `core` on the canvas :
  `gui_set_visible_obj -obj_type core -off`
- The following command turns on visibility for all objects on the canvas:
  `gui_set_visible_obj -all -on`

Related Commands

- `clear_all_rulers`
- `gui_hide`
- `gui_fit`
- `gui_redo`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `gui_show`
- `gui_undo`
- `gui_zoom`
- `refresh_canvas`

gui_set_visible_substrate

```
gui_set_visible_substrate
[-help]
[-all]
-name name
[-off]
[-on]
```

Sets the visible substrates.

Parameters

`-help`	Prints a brief description for each `gui_set_visible_substrate` parameter. *Data_type*: optional
`-all`	Turns visibility on for all substrates. *Data_type*: optional
`-name name`	Specifies the name of the substrate. *Data_type*: required
`-off`	Turns visibility off for the specified substrate. *Data_type*: optional
`-on`	Turns visibility on for the specified substrate. *Data_type*: optional

Examples

- The following command turns visibility on for the `DIE` substrate:
  ```
  gui_set_visible_substrate -name DIE -on
  ```
- The following command turns visibility off for the `DIE` substrate:
  ```
  gui_set_visible_substrate -name DIE -off
  ```
- The following command turns visibility on for all substrates:
  ```
  gui_set_visible_substrate -all -on
  ```
- The following command turns visibility off for all substrates:
  ```
  gui_set_visible_substrate -all -off
  ```

Related Commands

- clear_all_rulers
- gui_hide
- gui_fit
- gui_redo
- gui_set_visible_layer
- gui_show
- gui_undo
- gui_zoom
- refresh_canvas

gui_show

```
gui_show
[-help]
```

Opens the System Planner GUI if it was previously hidden.

Parameters

-help	Prints a brief description for each gui_show parameter. *Data_type*: optional

Example

- The following command opens the System Planner GUI:
  ```
  gui_show
  ```

Related Commands

- clear_all_rulers
- gui_fit
- gui_hide
- gui_redo
- gui_set_visible_layer
- gui_set_visible_substrate
- gui_undo
- gui_zoom
- refresh_canvas

gui_undo

```
gui_undo
[-help]
```

Undoes the previous edit. You can use the `gui_redo` command to redo an edit that was reversed with `gui_undo`.

Parameters

`-help`	Prints a brief description for each `gui_undo` parameter. *Data_type*: optional

Example

- The following command undoes the previous edit:
  ```
  gui_undo
  ```

Related Commands

- `clear_all_rulers`
- `gui_fit`
- `gui_hide`
- `gui_redo`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `gui_show`
- `gui_zoom`
- `refresh_canvas`

gui_zoom

```
gui_zoom
[-help]
[-in]
[-out]
[-rect rect]
[-selected]
[-to point]
```

Zooms into or out of the viewable window as per the specified parameters.

Parameters

`-help`	Prints a brief description for each `gui_zoom` parameter. *Data_type*: optional
`-in`	Zooms into the viewable window by two times. *Data_type*: optional
`-out`	Zooms out from the viewable window by two times. *Data_type*: optional
`-rect` *`rect`*	Zooms in the viewable window to show all of the rectangular area that is defined by the specified coordinates. *Data_type*: optional
`-selected`	Zooms in the viewable window to the area that encloses the selected objects. *Data_type*: optional
`-to` *`point`*	Zooms in the viewable window to the point defined by the point coordinates. *Data_type*: optional

Examples

- The following command zooms into the design in the window:
  ```
  gui_zoom -in
  ```
- The following command zooms into the area that encloses the selected objects.:
  ```
  gui_zoom -selected
  ```
- The following command zooms into the area centered around the point with x=1000 and y=2000:
  ```
  gui_zoom -to {1000 2000}
  ```
- The following command zooms into the area enclosed by the lower-left coordinates (`1000 2000`) and the upper-right coordinates (3000 5000):
  ```
  gui_zoom -rect {1000 2000 3000 5000}
  ```

Related Commands

- `clear_all_rulers`
- `gui_fit`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `refresh_canvas`

is_top_design

```
is_top_design
[-help]
cell
```

Specifies the top design.

Parameters

`-help`	Prints a brief description for each `is_top_design` parameter. *Data_type*: optional
`cell`	Specifies the cell. *Data_type*: required

Related Commands

- `check_design`
- `close_design`
- `create_design`

list_select_properties

```
list_select_properties
[-help]
-type {net | port | pin | device | via | finger | ruler | bundle | connection | wire | obstacle | shape | term | group}
```

Lists all the properties associated with the specified object type.

Parameters

`-help`	Prints a brief description for each `list_select_properties` parameter. *Data_type*: optional
`-type {net | port | pin | device | via | finger | ruler | bundle | connection | wire | obstacle | shape | term | group}`	
	Lists the properties associated with the specified object type. *Data_type*: enum, required

Example

- The following command lists all the properties associated with an object of type pin.
  `list_select_properties -type pin`

Related Commands

- `clear_selection`
- `deselect_obj`
- `get_selection`
- `select_obj`

man

```
man
page
[-help]
```

This command is a Tcl wrapper that calls the Linux ‘man’ command, so you can see man pages for System Planner commands and messages. If the argument matches the msg_id format (that is, several letters with a dash followed by an integer like `ABC-123`), the Tcl wrapper directly returns the msg_id message you would see in the log file, along with any extended help description if it is available for that msg_id. These msg_id ‘man-pages’ are not included in the man-page directories, but are only available inside System Planner.

Parameters

`-help`	Prints a brief description for each `man` parameter. *Data_type*: optional
`page`	Specifies the command name or message ID for which help is to be provided. *Data_type*: required

Examples

- The following command returns the manual description of the command `get_db`:

  ```
  man get_db
  ```

- The following command returns the manual description of the `DB-001` message ID:

  ```
  man DB-001
  ```

  Returns:

  ```
  NAME

  DB-001

  SUMMARY

  No active database in this stage.

  DESCRIPTION

  Need to create a design first.
  ```

move_device

```
move_device
[-help]
[-align_contacts]
{-selected | -objs objs}
{-origin {center | lower_left | lower_right | upper_left | upper_right}
  | -offset offset
  | -distance distance -direction {up | down | left | right}
  | -point point}
```

Moves the selected device as specified.

Parameters

`-help`	Prints a brief description of each `move_device` parameter. *Data_type*: optional
`-align_contacts`	Aligns the contact objects. *Data_type*: optional
`-direction {up | down | left | right}`	
	Specifies the direction of movement for the objects. *Data_type*: enum, optional
`-distance` *`distance`*	Specifies the distance, in microns, that the objects will be moved in the specified direction. *Data_type*: optional
`-objs` *`objs`*	Moves the specified devices (objects). *Data_type*: optional
`-offset` *`offset`*	Specifies the offset points by which the objects will be moved. *Data_type*: optional
`-origin {center | lower_left | lower_right | upper_left | upper_right}`	
	Moves the origin of the selected devices as specified. *Data_type*: enum, optional
`-point` *`point`*	Specifies the point to which the objects will be moved. *Data_type*: optional
`-selected`	Moves the selected devices. *Data_type*: optional

Examples

- The following command moves the selected device by the specified offset points:
  ```
  move_device -selected -offset {50.0 30.0}
  ```
- The following command moves the specified devices by the specified offset points:
  ```
  move_device -offset {50.0 30.0} -objs {/INTERPOSER/B /INTERPOSER/F}
  ```
- The following command moves the selected device to the specified point:
  ```
  move_device -selected -point {0.0 0.0}
  ```
- The following command moves the selected device to the right by 50 microns:
  ```
  move_device -selected -distance 50.0 -direction right
  ```
- The following command moves the origin of the selected device:
  ```
  move_device -selected -origin lower_left
  ```

Related Commands

- `create_device`
- `move_pin`
- `update_device`

move_pin

```
move_pin
[-help]
[-align_contacts]
[-direction {up | down | left | right}]
[-distance distance]
[-offset offset]
[-point point]
-selected
```

Moves the selected pin as specified.

Parameters

`-help`	Prints a brief description of each `move_pin` parameter. *Data_type*: optional
`-align_contacts`	Aligns the contact pins. *Data_type*: optional
`-direction {up | down | left | right}`	
	Specifies the direction of movement for the pins. *Data_type*: enum, optional
`-distance distance`	
	Specifies the distance, in microns, that the pins will be moved in the specified direction. *Data_type*: optional
`-offset offset`	Specifies the offset points by which the pins will be moved. *Data_type*: optional
`-point point`	Specifies the point to which the pins will be moved. *Data_type*: optional
`-selected`	Moves the selected pins. *Data_type*: required

Examples

- The following command moves the selected pin by the specified offset points:
  `move_pin -selected -offset {50.0 30.0}`
- The following command moves the selected pin to the specified point:
  `move_pin -selected -point {0.0 0.0}`
- The following command moves the selected pin to the right by 50 microns:
  `move_pin -selected -distance 50.0 -direction right`

Related Commands

- `move_device`

place_inst

```
place_inst
[-help]
-cell cell
-inst inst_name
[-fixed | -placed]
[-placed [-location location]]
```

Places a leaf instance.

Parameters

`-help`	Prints a brief description of each `place_inst` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the owner cell. *Data_type*: cell, required
`-inst` *`inst_name`*	Specifies the leaf instance name on the owner cell. *Data_type*: required
`-fixed`	Specifies that the cell being placed is in the `fixed` status. *Default*: `true` *Data_type*: optional
`-location` *`location`*	Specifies the *x* and *y* coordinates of the new origin of the instance being placed. This parameter can be specified only if `-placed` is specified. *Data_type*: optional
`-placed`	Specifies that the cell being placed is in the `placed` status but not fixed. *Data_type*: optional

Examples

- The following command places the `F` instance of the `INTERPOSER` cell with the `placed` status, which means it can be moved in the later stages, if required:
  ```
  place_inst -cell INTERPOSER -inst F -placed
  ```
- The following command places all instances on the `INTERPOSER` cell with the `fixed` status, which means these instances cannot be moved in the later stages:
  ```
  place_inst -cell INTERPOSER -inst * -fixed
  ```

Related Commands

- `unplace_obj`

place_io_info

```
place_io_info
[-help]
-device_path device_path
[-direction {clockwise | counterclockwise}]
[-io_gap io_gap]
[-io_inset io_inset]
[-pad_gap pad_gap]
[-pad_inset pad_inset]
[-start {top_right | top_left | top | bottom | bottom_right | bottom_left | right | left}]
```

Runs placement for each die.

Parameters

`-help`	Prints a brief description for each `place_io_info` parameter. *Data_type*: optional
`-device_path` *`device_path`*	Specifies the target device path. *Data_type*: instance path, required
`-direction {clockwise | counterclockwise}`	
	Specifies the placement direction. *Default*: `counterclockwise` *Data_type*: enum, optional
`-io_gap` *`io_gap`*	Specifies the gap between IO cells. *Default*: `0` *Data_type*: optional
`-io_inset` *`io_inset`*	Specifies the inset between IO cells. *Default*: `0` *Data_type*: optional
`-pad_gap` *`pad_gap`*	Specifies the gap between Pad cells. *Default*: `0` *Data_type*: optional
`-pad_inset` *`pad_inset`*	Specifies the inset between Pad cells. *Default*: `0` *Data_type*: optional
`-start {top_right | top_left | top | bottom | bottom_right | bottom_left | right | left}`	
	Specifies the anchor point. *Default*: `bottom_left` *Data_type*: enum, optional

Example

- The following command runs placement for the specified device path:

```
place_io_info -device_path /DIE1 -direction clockwise -io_gap 20 -io_inset 40 -pad_gap 30 -pad_inset 60
```

Related Commands

- `check_io_info`
- `read_io_info`

read_allegro

```
read_allegro
[-help]
file
[-align_contacts]
[-all_layer_on]
[-cell cell]
[-import_unplaced_comps]
[-just_devices]
[-just_padstacks]
[-mode {create | merge}]
[-no_bundles]
[-no_hierarchy]
[-no_interconnect]
[-no_shapes]
```

Imports an Allegro board (BRD), System-in-Package (SiP), or Multi-Chip Module (MCM) file. This command is equivalent to the *Import -> Allegro BRD, SiP, or MCM* menu command in System Planner.

Parameters

`-help`	Prints a brief description for each `read_allegro` parameter. *Data_type*: optional
`file`	Specifies the name of a BRD, SiP, or MCM file. *Data_type*: file, required
`-align_contacts`	Aligns contact pins. This option can be used only if `-mode` is `merge`. *Data_type*: optional
`-all_layer_on`	Specifies that the substrate is visible. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell template for the `merge` mode. *Data_type*: optional
`-import_unplaced_comps`	Imports unplaced components from Quickplace files. *Data_type*: optional
`-just_devices`	Specifies that only devices should be imported. *Data_type*: optional
`-just_padstacks`	Specifies that only padstacks should be imported. *Data_type*: optional
`-mode {create | merge}`	Specifies the mode for the process file. *Default*: `create` *Data_type*: enum
`-no_bundles`	Specifies that no bundles should be imported. *Data_type*: optional
`-no_hierarchy`	Specifies that no hierarchy should be imported. *Data_type*: optional
`-no_interconnect`	Specifies that no interconnect should be imported. *Data_type*: optional
`-no_shapes`	Specifies that no shapes should be imported. *Data_type*: optional

Example

- The following command merges the specified MCM file to the `BGA_PKG` cell:

```
read_allegro BGA_PKG_routed.mcm \
             -cell BGA_PKG \
             -mode merge
```

Related Command

- `write_allegro`

read_bga_text

```
read_bga_text
[-help]
[-coord{absolute | relative}]
-file file
[-parent parent_key]
[-pin_shape {circle | square | octagon | rect | oblong}]
[-pin_shape_args width_height]
-pin_style {pin | device}
[-process_shrink scale]
[-substrate substrate]
```

Imports the specified Ball Grid Array (BGA) text file. This command corresponds to the*File -> Import -> BGA Text*menu command in the System Planner GUI.

Parameters

`-help`	Prints a brief description for each `read_bga_text` parameter. *Data_type*: optional
`-coord {absolute | relative}`	Specifies whether the coordinates are absolute or relative. *Data_type*: enum, optional
`-file file`	Specifies the file name of the BGA txt file to be read. *Data_type*: file, required
`-parent parent_key`	Specifies the parent template. *Data_type*: optional
`-pin_shape {circle | square | octagon | rect | oblong}`	
	Specifies the shape of the placed pin. *Data_type*: enum, optional
`-pin_shape_args width_height`	Specifies the width/height of the shape of the placed pin.
`-pin_style {pin | device}`	Specifies the pin style to which the property CSV file applies. *Data_type*: enum, required
`-process_shrink scale`	Specifies the shrink factor for the device. For example: 0.75 to scale to 1/0.75 * 100% *Data_type*: optional
`-substrate substrate`	Specifies the substrate for which to load the txt file. If the design name does not exist, a device will be created. *Data_type*: optional

Example

- The following command reads in the `2_csv_pin_TDV_BGA` BGA text file:

  `read_bga_text -file 2_csv_pin_TDV_BGA.txt -coord relative -pin_style pin -process_shrink 1.0 -pin_shape square -pin_shape_args 21.96`

Related Commands

- `read_die_text`
- `write_bga_text`
- `write_die_text`

read_bump_locations

```
read_bump_locations
[-help]
-cell cell
[-device device]
-file_name file_name
[-minimum_pitch_x pitch_x]
[-minimum_pitch_y pitch_y]
[-mode {merge | overwrite}]
[-origin {origin | lower_left | center}]
[-overlap_control {keep_existing | keep_new}]
-version {simple | advance}
```

Reads in bump locations from the specified file.

Parameters

`-help`	Prints a brief description of each `read_bump_locations` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the input target. *Data_type*: cell, required
`-device` *`device`*	Specifies the device name of the input target. *Data_type*: optional
`-file_name` *`file_name`*	Specifies the name of the input file. *Data_type*: file, required
`-minimum_pitch_x` *`pitch_x`*	Specifies the minimum pitch x for overlap control. *Data_type*: optional
`-minimum_pitch_y` *`pitch_y`*	Specifies the minimum pitch y for overlap control. *Data_type*: optional
`-mode {merge | overwrite}`	
	Specifies the import mode. *Default*: `overwrite` *Data_type*: enum, optional
`-origin {origin | lower_left | center}`	
	Specifies the bump location option. *Default*: `center` *Data_type*: enum, optional
`-overlap_control {keep_existing | keep_new}`	
	Enables overlap control while reading in bumps. *Data_type*: enum, optional
`-version {simple | advance}`	
	Specifies the import format version: • `simple`: Innovus bump table format • `advance`: TSMC bump location file format *Data_type*: enum, required

Example

- The following command reads bump locations from the `input.txt` file in a bump table format:

```
read_bump_locations -file_name input.txt -cell FCCC_BOT -version simple
```

Related Command

- `write_bump_locations`

read_def

```
read_def
[-help]
[-align_contacts]
file_name
[-block]
[-cell cell_name]
[-core]
[-cover_and_bump]
[-device device_name]
[-endcap]
[-mode {merge | overwrite}]
[-net]
[-no_block]
[-no_core]
[-no_cover_and_bump]
[-no_design_inst]
[-no_endcap]
[-no_net]
[-no_pad]
[-no_ring]
[-no_special_net]
[-no_special_wire]
[-no_wire]
[-pad]
[-ring]
[-special_net]
[-special_wire]
[-substrate substrate_name]
[-wire]
```

Loads the specified DEF file.

Parameters

`-help`	Prints a brief description of each `read_def` parameter. *Data_type*: optional
`-align_contacts`	Aligns contact pins. *Data_type*: optional
`file_name`	Specifies the name of the DEF file to be read. *Data_type*: required
`-block`	Reads the BLOCK section from DEF components. *Data_type*: optional
`-cell` *`cell_name`*	Specifies the design name. *Data_type*: optional

`-core`	Reads the CORE section from DEF components. *Data_type*: optional
`-cover_and_bump`	Reads the COVER section from DEF components. *Data_type*: optional
`-device` *`device_name`*	Specifies a new name for the device. As a result, the device name and device template name would be different. By default, the tool uses the DESIGN name in the DEF as both the device and device template names. *Data_type*: optional
`-endcap`	Reads the ENDCAP section from DEF components. *Data_type*: optional
`-mode {merge | overwrite}`	Specifies the read mode. *Data_type*: enum, optional
`-net`	Reads the NETS section and applies to device pins. *Data_type*: optional
`-no_block`	Specifies the BLOCK section from DEF components should not be read. *Data_type*: optional
`-no_core`	Specifies the CORE section from DEF components should not be read. *Data_type*: optional
`-no_cover_and_bump`	Specifies the COVER section from DEF components should not be read. *Data_type*: optional
`-no_design_inst`	Specifies the design instance should not be created. *Data_type*: optional
`-no_endcap`	Specifies the ENDCAP section from DEF components should not be read. *Data_type*: optional
`-no_net`	Specifies the NETS section should not be read. *Data_type*: optional
`-no_pad`	Specifies the PAD section from DEF components should not be read. *Data_type*: optional
`-no_ring`	Specifies the RING section from DEF components should not be read. *Data_type*: optional
`-no_special_net`	Specifies the SPECIALNETS section should not be read. *Data_type*: optional
`-no_special_wire`	Specifies the wires defined in SPECIALNETS should not be read. *Data_type*: optional
`-no_wire`	Specifies the wires defined in NETS should not be read. *Data_type*: optional

`-pad`	Reads PAD from DEF components. *Data_type*: optional
`-ring`	Reads the RING section from DEF components. *Data_type*: optional
`-special_net`	Read the SPECIALNETS section and applies to device pins. *Data_type*: optional
`-special_wire`	Read wires defined in the SPECIALNETS section. *Data_type*: optional
`-substrate` *substrate_name*	Specifies the substrate for which to load the DEF. If the design name does not exist in memory, a device will be created. If the design already exists, DEF will be incrementally loaded for the device. *Data_type*: optional
`-wire`	Reads wires defined in the NETS section. *Data_type*: optional

Examples

- The following command reads in the `DEF.def` file for the `substrate` substrate:
  ```
  read_def DEF.def -substrate substrate
  ```
- The following command reads in the `DEF.def` file for the specified design:
  ```
  read_def DEF.def -cell {N5_sub:N5} # sugar: -cell {N5}
  ```

Related Commands

- `write_def`
- `create_substrate -lefs`
- `update_substrate -lefs`

read_design_stack

```
read_design_stack
[-help]
xml_file
[-mode {merge | overwrite}]
```

Reads the design stack from the specified XML file.The floorplan box based on the XML file is enforced by default.

Parameters

`-help`	Prints a brief description of each `read_design_stack` parameter. *Data_type*: optional
`xml_file`	Specifies the die stacking XML file to be read. *Data_type*: required
`-mode {merge | overwrite}`	
	Specifies the import mode, which can be `merge` or `overwrite`. *Data_type*: enum, optional

Example

- The following command reads the `top.xml` die stacking file:

```
read_design_stack top.xml
```

Related Commands

- `write_design_stack`

read_device_placement

```
read_device_placement
[-help]
csv_file
[-cell template]
[-origin {lower_left | lower_right | upper_left | upper_right | center | origin }]
```

Reads the device placement information from the specified `.csv` file.

Parameters

`-help`	Prints a brief description for each `read_device_placement` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be loaded. *Data_type*: file, required
`-cell` *`template`*	Specifies the cell for which to load the CSV. You can specify a device template name or top name. *Data_type*: optional
`-origin {lower_left | lower_right | upper_left | upper_right | center | origin }`	
	Specifies the placement origin. *Data_type*: enum, optional

Examples

- The following command reads the device placement information from the specified file for the `PKG:top` cell:

```
read_device_placement {C:\Users\morrisy\Desktop\tmp\tmp.csv} \
                      -cell {PKG:top}
                      -origin center
```

- The following command reads the device placement information from the specified file for the `design` cell:

```
read_device_placement {C:\Users\morrisy\Desktop\tmp\tmp2.csv} \
                      -cell design
                      -origin center
```

Related Commands

- `align_device_placement`
- `write_device_placement`

read_die_text

```
read_die_text
[-help]
[-coord{absolute | relative}]
-file file
[-parent parent_key]
[-pin_shape {circle | square | octagon | rect | oblong}]
[-pin_shape_args width_height]
-pin_style {pin | device}
[-process_shrink scale]
[-substrate substrate]
```

Imports the specified Die text file. This command corresponds to the *File -> Import -> Die Text* menu command in the System Planner GUI.

Typically, `-pin_style pin` is used for package or standard dies, which are usually sourced from Allegro/APD. `-pin_style device` is used for co-design dies, which are usually sourced from Innovus.

Parameters

`-help`	Prints a brief description for each `read_die_text` parameter. *Data_type*: optional
`-coord {absolute | relative}`	Specifies whether the coordinates are absolute or relative. *Data_type*: enum, optional
`-file` *`file`*	Specifies the file name of the Die txt file to be read. *Data_type*: file, required
`-parent parent_key`	Specifies the parent template. *Data_type*: optional
`-pin_shape {circle | square | octagon | rect | oblong}`	
	Specifies the shape of the placed pin. *Data_type*: enum, optional
`-pin_shape_args` *`width_height`*	Specifies the width/height of the shape of the placed pin.
`-pin_style {pin | device}`	Specifies the pin style to which the property CSV file applies. *Data_type*: enum, required
`-process_shrink` *`scale`*	Specifies the shrink factor for the device. For example: 0.75 to scale to 1/0.75 * 100% *Data_type*: optional
`-substrate` *`substrate`*	Specifies the substrate for which to load the txt file. If the design name does not exist, a device will be created. *Data_type*: optional

Example

- The following command reads in the `2_csv_pin_TDV_DIE2` text file:

  ```
  read_die_text -file 2_csv_pin_TDV_DIE2.txt -coord relative -pin_style pin -parent BGA -process_shrink 1.0 -pin_shape_args
  4.5
  ```

Related Commands

- `read_bga_text`
- `write_bga_text`
- `write_die_text`

read_io_info

```
read_io_info
[-help]
-device_path device_path
file
```

Loads the IO info file.

Parameters

`-help`	Prints a brief description for each `read_io_info` parameter. *Data_type*: optional
`file`	Specifies the IO info file path to be read. *Data_type*: file, required
`-device_path` *`device_path`*	Specifies the target device path. *Data_type*: instance path, required

Example

- The following command reads in the specified IO info file:
  `read_io_info -device_path /DIE1 -file {IOinfo_v1_2020_08_18_pg.csv}`

Related Commands

- `check_io_info`
- `place_io_info`

read_lef

```
read_lef
[-help]
-files file
[-substrate substrate]
```

Reads the specified LEF files. Use this command toimport LEF to an existing or new substrate.

Parameters

`-help`	Prints a brief description for each `read_lef` parameter. *Data_type*: optional
`-files` *`file`*	Specifies the LEF files to load. *Data_type*: required
`-substrate` *`substrate`*	Specifies the target substrate name. *Data_type*: optional

Related Command

- `read_def`

read_module_model

```
read_module_model
[-help]
[cell_arg]
[-cell cell_name]
[-scale scale]
[-tag tag_name]
[-type type]
```

Restores a design from a default module model repository. You can use this command only after the default central directory name where the module model data will be saved/loaded has been specified using the `set_module_model -default_dir` command.

Parameters

`-help`	Prints a brief description for each `read_module_model` parameter. *Data_type*: optional
cell_arg	Specifies the file path directly. It can be formatted as: `/path/to/iHDB/<cell>/<tag>` *Data_type*: optional
`-cell` *cell_name*	Specifies the name of the module model being created. *Data_type*: optional
`-scale` *scale*	Specifies the scale of the module model. This parameter affects only the `pnr` type.
`-tag` *tag_name*	Specifies the tag name of the flow. *Data_type*: optional
`-type` *type*	Specifies the type of the module model. For *type*, you can specify: • `oio` - Reads the top oio and merges DEF. This is the default option. • `pnr` - Reads LEF/DEF or OpenAccess design. *Data_type*: required

Examples

- The following command reads in an `oio` type module model:
  `read_module_model -cell Design -type oio -tag v0`
- The following command reads in an oio and ECO cell:
  `read_module_model -cell Die1 -type oio -tag v0`
- The following command reads in a `pnr` cell (LEF/DEF):
  `read_module_model -cell Die1 -type pnr -tag v0`
- The following command reads in a `pnr` cell (OpenAccess). (Note: You need to set environment variable `OrbitIO.OAEnabled`).
  `read_module_model -cell analog_die -type pnr -tag v0`
- The following command reads in a module model by specifying `cell_arg`, which is a file path:

```
read_module_model DBS/Interposer/v0/pre_pnr
```

Related Commands

- `create_module_model`
- `set_module_model`

read_net_interface

```
read_net_interface
[-help]
-cell cell
file
[-assign_io_pin]
```

Reads the net interface file.

Parameters

`-help`	Prints a brief description for each `read_net_interface` parameter. *Data_type*: optional
`file`	Specifies the file to be read. *Data_type*: file, required
`-assign_io_pin`	Specifies whether or not to assign the IO pin to the floorplan. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell for loading the CSV file. *Data_type*: cell, required

Example

- The following command reads in the specified net interface file:
  ```
  read_net_interface brd_interface.csv -cell PLACED_HDL
  ```

read_net_list

```
read_net_list
[-help]
csv_file
-cell cell
[-mode {overwrite | merge}]
```

Reads the specified netlist file.

Parameters

`-help`	Prints a brief description for each `read_net_list` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be loaded. *Data_type*: file, required
`-cell` *`cell`*	Specifies the cell for which the CSV is to be loaded. *Data_type*: cell, required
`-mode {overwrite | merge`	Specifies the import mode. By default, the tool does not assignIO pins to floorplans. *Default*: `merge` *Data_type*: enum, optional

Example

- The following command reads in the specified netlist file for the `MyPackage` cell:

  ```
  read_net_list net_list.csv -cell MyPackage -mode overwrite
  ```

Related Commands

- `read_term_list`
- `read_term_map`
- `write_term_list`

read_net_map

```
read_net_map
[-help]
csv_file
[-dont_auto_create_parent_nets]
[-dont_auto_create_top_nets]
```

Imports net mapping information from a CSV file. By default, the parent and top nets are created automatically.

Parameters

`-help`	Prints a brief description for each `read_net_map` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be loaded. *Data_type*: file, required
`-dont_auto_create_parent_nets`	
	Disables the automatic creation of parent nets. *Data_type*: optional
`-dont_auto_create_top_nets`	Disables the automatic creation of top nets. *Data_type*: optional

Examples

- The following command imports net mapping information from the `net_map.csv` file with automatic creation of parent and top nets:
  `read_net_map net_map.csv`
- The following command imports net mapping information from the `net_map.csv` file, while disabling the automatic creation of parent and top nets:
  `read_net_map net_map.csv -dont_auto_create_parent_nets -dont_auto_create_top_nets`

Related Commands

- `read_net_list`

read_oio

```
read_oio
[-help]
file
```

Reads the specified `.oio` file.

Parameters

`-help`	Prints a brief description for each `read_oio` parameter. *Data_type*: optional
file	Specifies the location of the `.oio` file to be read. *Data_type*: required

Related Command

- `write_oio`

read_pin_label

```
read_pin_label
[-help]
file
-cell cell
```

Reads the pin label from the specified file.

Parameters

`-help`	Prints a brief description for each `read_pin_label` parameter. *Data_type*: optional
file	Specifies the path of the file to be read. *Data_type*: file, required
`-cell` *cell*	Specifies the cell. *Data_type*: required

Example

- The following command reads in the pin labels from the `input.csv` file:

```
read_pin_label input.csv -cell TC2_NOR_TEST_1:RF_BGA ;
```

Related Command

- `write_pin_label`

read_pin_physical_list

```
read_pin_physical_list
[-help]
csv_file
 -device_path path
[-mode {overwrite | merge}]
[-origin {origin | center}]
[-type {pin | device}]
```

Reads the pin list from the specified file.

Parameters

`-help`	Prints a brief description for each `read_pin_physical_list` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be loaded. *Data_type*: file, required
`-device_path` *`path`*	Specifies the target device path. *Data_type*: required
`-mode {overwrite | merge}`	
	Specifies the mode for reading. *Default*: `merge` *Data_type*: enum, optional
`-origin {origin | center}`	
	Specifies the placement origin. If `-origin` is set to `origin`, the placement origin is set to the cell origin. *Default*: `center` *Data_type*: enum, optional
`-type {pin | device}`	Specifies the type of pin. *Default*: `device` *Data_type*: enum, optional

Example

- The following command reads in the pin list from the `input.csv` file:

  ```
  read_pin_physical_list input.csv -device_path INTERPOSER -origin origin
  ```

Related Command

- `write_pin_physical_list`

read_power_intent

```
read_power_intent
[-help]
fileName
-1801
```

Reads in a power intent file and checks syntax.

Parameters

`-help`	Prints a brief description for each `read_power_intent` parameter. *Data_type*: optional
`fileName`	Specifies the name of the power intent file to be read. *Data_type*: file, required
`-1801`	Specifies that the IEEE 1801 power format will be used. *Data_type*: required

Example

- The following command reads in a power intent file,`power.upf`, using the 1801 format:
  ```
  read_power_intent power.upf -1801
  ```

Related Information

- `write_power_intent`

read_property

```
read_property
[-help]
csv_file
-device_path path
[-mode {overwrite | merge}]
[-refer_to_parent_net_name]
[-type {net | pin | device}]
```

Imports a CSV personalities file. This command is equivalent to the *Import -> CSV Personalities* command in the context menu of a device in the System Planner.

Parameters

`-help`	Prints a brief description for each `read_property` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be loaded. *Data_type*: file, required
`-device_path` *`path`*	Specifies the device path for which the CSV file is to be loaded. *Data_type*: required
`-mode {overwrite | merge}`	
	Specifies the read mode for import. *Default*: `overwrite` *Data_type*: enum, optional
`-refer_to_parent_net_name`	
	Classifies personalities by parent net name. *Data_type*: optional
`-type {net | pin | device}`	
	Specifies the object type to which the property CSV file applies. *Default*: `net` *Data_type*: enum, optional

Example

- The following command imports the personality file, `devicePersonality.csv`, for the `DIE` device.

  ```
  read_property devicePersonality.csv -device_path /DIE -type device
  ```

Related Commands

- `set_property`

read_systemlvs_cdl_port_config

```
read_systemlvs_cdl_port_config
[-help]
-file file
[-merge]
```

Imports the specified CDL config file.

Parameters

`-help`	Prints a brief description for each `read_systemlvs_cdl_port_config` parameter. *Data_type*: optional
`-file` *`file`*	Specifies the CDL config file to be loaded. *Data_type*: file, required
`-merge`	Merges with existing pin labels. *Data_type*: optional

Example

- The following command reads the specified CDL file and merges with existing pin labels:
  `read_systemlvs_cdl_port_config -file golden_config.csv -merge`

Related Commands

- `run_systemlvs`
- `set_systemlvs_design_check`
- `set_systemlvs_interface_check`
- `set_systemlvs_netlist_usage`
- `view_systemlvs_result`

read_term_list

```
read_term_list
[-help]
csv_file
[-cell cell]
[-module module]
[-overwrite_existent_terms]
```

Reads the specified term list file.

Parameters

`-help`	Prints a brief description for each `read_term_list` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be loaded. *Data_type*: file, required
`-cell` *`cell`*	Specifies the cell for which the CSV is to be loaded. You should not specify this parameter with the `-module` parameter. *Data_type*: cell, optional
`-module` *`module`*	Specifies the module for which the CSV is to be loaded. You should not specify this parameter with the `-cell` parameter. *Data_type*: module, optional
`-overwrite_existent_terms`	Overwrites existent terms. *Data_type*: optional

Example

- The following command reads in the specified term list file for the `RF_BGA` cell:

```
read_term_list {C:\Users\morrisy\Desktop\RegressionTest\17.2-Dev\TermList\Import\BGA_TermList.csv} \
               -cell RF_BGA
```

Related Command

- `read_term_map`
- `write_term_list`

read_term_map

```
read_term_map
[-help]
csv_file
[-cell cell]
[-ignore_non_existent_nets_and_terms]
[-module module]
[-overwrite_existent_term_map]
```

Reads the specified term map file.

Parameters

`-help`	Prints a brief description for each `read_term_map` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be loaded. *Data_type*: list of file, required
`-cell` *`cell`*	Specifies the cell for which the CSV is to be loaded. You should not specify this parameter with the `-module` parameter. *Data_type*: cell, optional
`-ignore_non_existent_nets_and_terms`	
	Does not create non-existent nets or terms. *Data_type*: optional
`-module` *`module`*	Specifies the module for which the CSV is to be loaded. You should not specify this parameter with the `-cell` parameter. *Data_type*: module, optional
`-overwrite_existent_term_map`	
	Overwrites the original term maps. *Data_type*: optional

Example

- The following command reads in the specified term map file for the `RF_BGA` cell:

```
read_term_map {<path>\termMap.csv} \
              -cell RF_BGA \
              -ignore_non_existent_nets_and_terms
```

Related Command

- `read_term_list`
- `write_term_map`

read_verilog

```
read_verilog
[-help]
-file file
-mode {overwrite | merge}
[-overwrite_pg]
[-substrate substrate]
[-top module]
[-verbose]
```

Reads in the specified Verilog file.

Parameters

`-help`	Prints a brief description for each `read_verilog` parameter. *Data_type*: optional
`-file` *`file`*	Specifies the Verilog files to be read. *Data_type*: required
`-mode {overwrite | merge}`	
	Specifies the conflict strategy. *Data_type*: enum, required
`-overwrite_pg`	Specifies that power/ground information is to be overwritten. *Data_type*: optional
`-substrate` *`substrate`*	
	Specifies the substrate name. *Data_type*: optional
`-top` *`module`*	Specifies the top module. *Data_type*: module, optional
`-verbose`	Shows details about the results. *Data_type*: optional

Example

- The following command reads in the specified Verilog files in the overwrite mode:

```
read_verilog -file {mother_die.v.gz mother_die2.v} \
             -substrate DIE \
             -mode overwrite \
             -overwrite_pg
```

Related Commands

- `write_verilog`

refresh_canvas

```
refresh_canvas
[-help]
```

Refreshes the current canvas view.

Parameters

`-help`	Prints a brief description for each `refresh_canvas` parameter. *Data_type*: optional

Related Commands

- `clear_all_rulers`
- `gui_hide`
- `gui_fit`
- `gui_redo`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `gui_show`
- `gui_undo`
- `gui_zoom`
- `refresh_pin_map`

refresh_pin_map

```
refresh_pin_map
[-help]
```

Refreshes the pin map.

Parameters

`-help`	Prints a brief description for each `refresh_pin_map` parameter. *Data_type*: optional

Related Command

- `refresh_canvas`

remove_connections

```
remove_connections
[-help]
```

Removes connections.

Parameters

`-help`	Prints a brief description for each `remove_connections` parameter. *Data_type*: optional

Related Commands

- show_connections

rename_matrix_objs

```
rename_matrix_objs
[-help]
-cell cell
-pattern {col_row | row_col | horizontal | vertical | spiral_cw | spiral_ccw | serpentine_h | serpentine_v}
-prefix_type {none | net | device | text}
-start {lower_left | upper_left | lower_right | upper_right}
[-assign_style {matrix}]
[-help]
[-prefix_text prefix_text]
[-selected]
[-type {device | pin}]
{-name_style {jedec | alpha | numeric}
  | {-row_name_style {jedec | alpha | numeric} -col_name_style {jedec | alpha | numeric}}}
```

Renames objects such as pins and bump devices.

Parameters

`-help`	Prints a brief description of each `rename_matrix_objs` parameter. *Data_type*: optional
`-assign_style {matrix}`	Specifies the assign style. *Default*: `matrix` *Data_type*: enum, optional
`-cell cell`	Specifies the owner cell as context. *Data_type*: cell, required
`-col_name_style {jedec | alpha | numeric}`	
	Specifies the column naming style. This option can be used only if `-pattern` `col_row row_col` is specified. *Data_type*: enum, optional

`-name_style {jedec | alpha | numeric}`	
	Specifies the naming style to be used for the objects being renamed. This option can be used only if `-pattern` `` `horizontal vertical spiral_cw spiral_ccw serpentine_h serpentine_v` `` is specified. *Data_type*: enum, optional
`-pattern {col_row | row_col | horizontal | vertical | spiral_cw | spiral_ccw | serpentine_h | serpentine_v}`	
	Specifies the naming rule pattern. *Data_type*: enum, required
`-prefix_text` *`prefix_text`*	
	Specifies the prefix text if `-prefix_type 'text'` is specified. *Data_type*: optional
`-prefix_type {none | net | device | text}`	
	Specifies the prefix name with associated objects. *Data_type*: enum, required
`-row_name_style {jedec | alpha | numeric}`	
	Specifies the row naming style. This option can be used only if `-pattern 'col_row row_col'` is specified. *Data_type*: enum, optional
`-selected`	Specifies that the selected objects are to be renamed. *Data_type*: optional
`-start {lower_left | upper_left | lower_right | upper_right}`	
	Specifies the start anchor point. *Data_type*: enum, required
`-type {device | pin}`	Specifies the type of the objects to be renamed. *Data_type*: enum, optional

Examples

- The following command renames all bump devices:

```
rename_matrix_objs -cell MyChip \
    -type device \
    -assign_style matrix \
    -pattern row_col \
    -start lower_right \
    -row_name_style jedec \
    -col_name_style numeric \
    -prefix_type none
```

- The following command renames selected pins:

```
select_obj -type pin -rect {2860.838601 3524.737933 3312.498988 4225.822714}
rename_matrix_objs -cell MyChip \
    -type pin \
    -selected \
    -assign_style matrix \
    -pattern row_col \
    -start upper_left \
    -row_name_style jedec \
    -col_name_style numeric \
    -prefix_type text \
    -prefix_text Sel
```

run_lec

```
run_lec
[-help]
{{-workspace workspace} | {{-run_dir run_dir} {-tag tag}}}
```

Launches the LEC flow.

Parameters

`-help`	Prints a brief description for each `run_lec` parameter. *Data_type*: optional
`-run_dir` *`run_dir`*	Specifies the workspace with *`run_dir`* and *`tag`*; this argument assigns the run directory. You cannot set this argument and `-workspace` simultaneously. *Data_type*: file, optional
`-tag` *`tag`*	Specifies the workspace with *`run_dir`* and *`tag`*; this argument assigns the tag. You cannot set this argument and `-workspace` simultaneously. *Data_type*: optional
`-workspace` *`workspace`*	Specifies the workspace directly. You cannot set this argument and`-run_dir` simultaneously. *Data_type*: file, optional

Example

- The following command runs LEC flow using the specified workspace:
  `run_lec -workspace /user/Desktop/myLecTest/`

Related Commands

- `set_lec_library`
- `set_lec_netlist`
- `set_lec_user_preference`
- `view_lec_result`

run_systemlvs

```
run_systemlvs
[-help]
[-add_timestamp]
[-run_dir run_dir]
[-show_result_gui]
[-tag tag]
[-workspace workspace]
```

Launches the System LVS flow.

Parameters

`-help`	Prints a brief description for each `run_systemlvs` parameter. *Data_type*: optional
`-add_timestamp`	Adds the timestamp to the `systemLvs.run` folder. *Data_type*: optional
`-run_dir` *`run_dir`*	Specifies the workspace with *`run_dir`* and *`tag`*; this argument assigns the run directory. You cannot set this argument and `-workspace` simultaneously. *Data_type*: file, optional
`-show_result_gui`	Show the resulting GUI after systemlvs execution *Data_type*: optional
`-tag` *`tag`*	Specifies the workspace with *`run_dir`* and *`tag`*; this argument assigns the tag. You cannot set this argument and `-workspace` simultaneously. *Data_type*: optional
`-workspace` *`workspace`*	Specifies the workspace directly. You cannot set this argument and `-run_dir` simultaneously. *Data_type*: file, optional

Example

- The following command runs System LVS using the specified workspace and shows the resulting GUI after the run:
  `run_systemlvs -file golden_config.csv -merge run_systemlvs -workspace /myLvsTest/SystemLvs.run -show_result_gui`

Related Commands

- `read_systemlvs_cdl_port_config`
- `set_systemlvs_design_check`
- `set_systemlvs_interface_check`
- `set_systemlvs_netlist_usage`
- `view_systemlvs_result`

run_thermal

```
run_thermal
[-help]
-run_dir run_dir
-tag tag
-type {run | open}
```

Specifies the thermal user preference settings.

Parameters

`-help`	Prints a brief description for each `run_thermal` parameter. *Data_type*: optional
`-run_dir` *`run_dir`*	Specifies the run directory. *Data_type*: file, required
`-tag` *`tag`*	Specifies the run tag. *Data_type*: required
`-type {run | open}`	Specifies the type. *Data_type*: enum, required

Example

- The following command runs the thermal simulation:
  `run_thermal -run_dir /Run_dir -tag thermal -type run`
- The following command transfers the design to Celsius:
  `run_thermal -run_dir /Run_dir -tag thermal -type open`

Related Commands

- `set_thermal_ambient_condition`
- `set_thermal_contact_attribute`
- `set_thermal_heat_sink`
- `set_thermal_layer_material`
- `set_thermal_molding_compound`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_tech`
- `set_thermal_test_board`
- `thermal::view_result`

select_obj

```
select_obj
[-help]
[any_object+]
[-all]
[-area_operation {contain | touch}]
[-device_path device_path]
[-grid_filter {row | col}]
[-grid_pattern grid_pattern]
[-include_descendant]
[-line line]
[-modifier modifier]
[-no_visible_filter]
[-point point]
[-polygon polygon]
[-prop_name pname]
[-prop_operation {equals | not_equals}]
[-prop_values pvalues]
[-rect rect]
[-select_type {append | replace | remove | intersect | xor}]
[-substrate substrate]
-type {net | port | pin | device | via | finger | ruler | bundle | connection | wire | obstacle | shape | term | group }
```

Selects database objects of the specified type by spatial query; the selection result can be fetched with `get_selection`.

Parameters

`-help`	Prints a brief description for each `select_obj` parameter. *Data_type*: optional
`any_object+`	Specifies the list of objects to be selected. *Data_type*: optional
`-all`	Select all the objects in the design. *Data_type*: optional
`-area_operation {contain | touch}`	
	Specifies whether objects should be selected if they are contained within or touching specified area. *Data_type*: enum, optional
`-device_path` *`device_path`*	
	Specifies the device path from which to select the objects. *Data_type*: optional
`-grid_filter {row | col}`	
	Specifies the grid filter. *Data_type*: enum, optional
`-grid_pattern` *`grid_pattern`*	Specifies the pattern for the grid filter. *Data_type*: optional

`-include_descendant`	Specifies the descendant constraint with option `-device_path`. *Data_type*: optional
`-line` *`line`*	Selects the objects through which the specified line passes. *Data_type*: optional
`-modifier` *`modifier`*	Specifies the selection modifier. *Data_type*: optional
`-no_visible_filter`	Disables the visible filter. *Data_type*: optional
`-point` *`point`*	Selects the object at the specified point. *Data_type*: optional
`-polygon` *`polygon`*	Specifies the point list of the polygon, in microns. *Data_type*: optional
`-prop_name` *`pname`*	Specifies the property to be used for selecting objects by property. *Data_type*: optional
`-prop_operation {equals | not_equals}`	
	Specifies whether objects should be selected if their value equals or does not equal the specified property value. *Data_type*: enum, optional
`-prop_values` *`pvalues`*	
	Specifies the property values to be used for selection by property. Wildcard value is supported in *`pvalues`*. *Data_type*: optional
`-rect` *`rect`*	Selects the objects falling within the specified rectangular area. *Data_type*: optional
`-select_type {append | replace | remove | intersect | xor}`	
	Specifies whether to append to, replace, or remove the current selection. *Data_type*: enum, optional
`-substrate` *`substrate`*	Specifies the substrate from which to select the objects. *Data_type*: optional
`-type {net | port | pin | device | via | finger | ruler | bundle | connection | wire | obstacle | shape | term | group}`	
	Specifies the object type you want to select. *Data_type*: enum, required

Examples

- The following command selects Power and Signal pins contained within the specified rectangular area:
```
select_obj -type pin -rect {6675 7248 12797 10993} -select_type append \
           -prop_name netClass \
           -prop_values {Power Signal} \
           -prop_operation equals \
           -area_operation contain
```
- The following command selects pins with the net property:
```
select_obj -type pin -prop_name net -prop_values {/DIE/NET*}
```
- The following command selects the specified objects:
```
select_obj -type device {{/BGA/DIE1 Bump_1} {/BGA/DIE1 Bump_2}}
```
- The following command selects pins contained within the specified polygon area:
```
select_obj -type pin -polygon {{-2000.0 2000.0}{2000.0 2000.0}{0 -2000.0}}
```
- The following commands select objects using device path filters:
```
# Device Hierarchy
# /top/SoC
# /top/HBM1/COVER
# /top/HBM2/COVER
# /top/HBM3/COVER

select_obj -type device -device_path /top/HBM1 # BUMPS, COVER, not include COVER-BUMPS, HBM1
select_obj -type device -device_path /top    # HBM*, SoC
select_obj -type device -device_path /top -prop_name name -prop_values {HBM*}      # HBM*
# ---------------------------------------------
select_obj -type pin -device_path /top/HBM1            # HBM1-pin, not include BUMPS-pin, COVER-BUMPS-pin
select_obj -type pin -device_path /top/HBM1 -include_descendant # HBM1-pin, BUMPS-pin, COVER-pin, COVER/BUMPS-pin
select_obj -type pin -device_path /top/HBM1/COVER -include_descendant # COVER/BUMPS-pin
# ---------------------------------------------
select_obj -type pin -device_path /top/HBM1/Bump_*        # BUMPS-pin
select_obj -type pin -device_path /top/HBM1/C4S*          # C4SBUMPS-pin
```
- The following commands select `BUMPPAD` pins with the name prefix `Z`:
```
select_obj -type pin -select_type replace -prop_name type -prop_values BUMPPAD -prop_operation equals
select_obj -type pin -select_type intersect -prop_name name -prop_values Z*
```
- The following commands select ground bump devices with a specific cell:
```
select_obj -type device -select_type replace -prop_name cell -prop_values INTERPOSER:FRONT_UBUMP -prop_operation equals -
area_operation contain -modifier <NONE>
select_obj -type pin -select_type intersect -prop_name netClass -prop_values Ground -prop_operation equals -area_operation
contain -modifier {and devices}
```

Related Commands

- `clear_selection`
- `deselect_obj`
- `get_selection`
- `list_select_properties`

set_connection_optimization

```
set_connection_optimization
[-help]
[-fixed fixed_source]
[-fixed_type {selection | personality | floorplan}]
[-free free_source]
[-free_type {selection | personality | floorplan}]
```

Optimizes the specified connection.

Parameters

`-help`	Prints a brief description for each `set_connection_optimization` parameter. *Data_type*: optional
`-fixed` *`fixed_source`*	Specifies the source of the fixed side of the connection. *Data_type*: optional
`-fixed_type {selection | personality | floorplan}`	
	Specifies the source type of the fixed side of the connection. *Data_type*: optional
`-free` *`free_source`*	Specifies the source of the free side of the connection. *Data_type*: optional
`-free_type {selection | personality | floorplan`	
	Specifies the source type of the free side of the connection. *Data_type*: optional

Examples

- The following set of commands sets the first selected pin as the source of the fixed sided, then sets the second selected pin as the source of the free side, and finally commits the connection:

```
select_obj -type pin -area_operation contain -rect {1777 1058 1828 1275}
set_connection_optimization -fixed_type selection

clear_selection
select_obj -type pin -area_operation contain -rect {2368 1126 2474 1299}
set_connection_optimization -free_type selection

commit_connection_optimization -swap_devices
```

- The following set of commands sets first specified floorplan as source of the fixed side, then sets the second specified floorplan as the source of the free side, and finally commits the connection:

```
set_connection_optimization -fixed_type floorplan -fixed {/BGA1/DIE1 [null]:DDR0_data3:FC_DESIGN-02_1:DIE5}
set_connection_optimization -free_type floorplan -free {/BGA1 [null]:DDR0_data3:FC_DESIGN-02_1:18X18_BGA}

commit_connection_optimization
```

- The following set of commands sets first specified personality as source of the fixed side, then sets the second specified personality as the source of the free side, and finally commits the connection:

```
set_connection_optimization -fixed_type personality -fixed { /FT-REV0-15 FT-REV0-15_1:FT-REV0-15:PORT:125S1} # full
expression for personality '125S1' under path /FT-REV0-15
set_connection_optimization -free_type personality -free {/FT-REV0-15/dut1}                                # sugar
expression
commit_connection_optimization
```

Related Commands

- commit_connection_optimization

set_db

```
set_db
[-help]
start
[chain]
value
[-category {predefine | program | substrate | layer | cell | cell_inst | pin | pin_inst
    | port | padstack | module | mod_inst | net | term | term_map | seal_ring | scribe_line | grid
    | device_grid | pin_grid | group | region | current_region | device_path | mod_path}]
```

Sets values for object attributes in the database.

Parameters

`-help`	Prints a brief description for each `set_db` parameter. *Data_type*: optional
start	Specifies the starting objects or root attributes. *Data_type*: required
chain	Specifies the attribute names that are valid for each level of the traversal. Check the "chain Attribute Specification" section in `get_db` for details. *Data_type*: optional
value	Specifies a new value for the specified object attribute elements. The value type depends on which object attribute type is being modified. *Data_type*: required
`-category {predefine | program | substrate | layer | cell | cell_inst | pin | pin_inst | port | padstack | module | mod_inst | net | term | term_map | seal_ring | scribe_line | grid | device_grid | pin_grid | group | region | current_region | device_path | mod_path}`	
	Specifies the category of root attributes. If you do not specify the `-category` option, the tool determines the category automatically. *Data_type*: enum, optional

Examples

- The following command disables undo and redo:
  `set_db current_design .history.max_history 0`
- The following command changes the decimal places of unit on the current design:
  `set_db current_design .decimal_places 4`
- The following command changes the substrate name from the specified cell instance:
  `set_db MyPackage .substrate.name MyPackage -category cell_inst`
- The following command changes the order of the specified layer:
  `set_db MyPackage layer_key_string -category layer .order new_order`

Related Commands

- `delete_obj`
- `get_db`

set_era_die_config

```
set_era_die_config
[-help]
[-bump_region file]
[-def file]
-device_path path
[-pgv directory]
[-psdl file]
[-qrc file]
[-reset]
[-tsv_layers layers]
[-tsv_subckt file]
```

Sets Early Rail Analysis (ERA) die config options.

Parameters

`-help`	Prints a brief description for each `set_era_die_config` parameter. *Data_type*: optional
`-bump_region file`	Specifies the bump region file. *Data_type*: file, optional
`-def file`	Specifies the DEF file. *Data_type*: file, optional
`-device_path path`	Specifies the target device path. *Data_type*: required
`-pgv directory`	Specifies the pgv directory. *Data_type*: file, optional
`-psdl file`	Specifies the PSDL file. *Data_type*: file, optional
`-qrc file`	Specifies the QRC tech file. *Data_type*: file, optional
`-reset`	Resets the settings. *Data_type*: optional
`-tsv_layers layers`	Specifies the TSV top and bottom layers. *Format*: `{top_layer,bottom_layer}` *Data_type*: optional
`-tsv_subckt file`	Specifies the TSV subckt file. *Data_type*: file, optional

Examples

- The following command sets the ERA die config options for the `Interposer` device path:

```
set_era_die_config \
    -device_path Interposer \
    -psdl input_data/PSDL/Interposer.psdl \
    -pgv input_data/PGV/Interposer \
    -qrc input_data/QRC/Interposer/qrcTechFile \
    -bump_region input_data/Bump_Region/Interposer.bump_region \
    -tsv_subckt input_data/CKT/Interposer.subckt \
    -tsv_layers {M1 MB}
```

Related Commands

- `d`

set_era_pg_net_config

```
set_era_pg_net_config
[-help]
-cell cell
-net net
[-reset]
[-threshold threshold]
[-voltage voltage]
```

Sets the Early Rail Analysis (ERA) PG net config options.

Parameters

`-help`	Prints a brief description for each `set_era_pg_net_config` parameter. *Data_type*: optional
`-cell cell`	Specifies the target cell. *Data_type*: cell, required
`-net net`	Specifies the target net. *Data_type*: required
`-reset`	Resets the settings. *Data_type*: optional
`-threshold threshold`	Specifies the threshold. *Data_type*: optional
`-voltage voltage`	Specifies the voltage. *Data_type*: optional

Example

- The following command sets the ERA PG net config options for the `Interposer` cell:
  ```
  set_era_pg_net_config -cell Interposer -net VSS -voltage 0.0 -threshold 0.1
  ```

Related Commands

- `set_era_die_config`

set_era_result_user_preference

```
set_era_result_user_preference
[-help]
[file_name]
[-reset]
```

Sets the ERA result user preference tcl file

Parameters

`-help`	Prints a brief description for each `set_era_result_user_preference` parameter. *Data_type*: optional
`file_name`	Specifies the ERA result user preference tcl file. *Data_type*: file, optional
`-reset`	Resets the settings. *Data_type*: optional

Example

- The following command sets `user_result.tcl` as the ERA result user preference tcl file:
  `set_era_result_user_preference input_data/User_Tcl/user_result.tcl`

Related Commands

- `set_era_die_config`
- `set_era_pg_net_config`
- `set_vsrc_layers`

set_era_user_preference

```
set_era_user_preference
[-help]
[file_name]
[-reset]
```

Sets the ERA user preference tcl file

Parameters

`-help`	Prints a brief description for each `set_era_user_preference` parameter. *Data_type*: optional
`file_name`	Specifies the ERA user preference tcl file. *Data_type*: file, optional
`-reset`	Resets the settings. *Data_type*: optional

Example

- The following command sets `user_run.tcl` as the ERA user preference tcl file:
  `set_era_user_preference input_data/User_Tcl/user_run.tcl`

Related Commands

- `set_era_die_config`
- `set_era_pg_net_config`
- `set_vsrc_layers`
- `set_era_result_user_preference`

set_hierarchy

```
set_hierarchy
[-help]
[design]
[-root]
```

Switch to specified hierarchy when working with multiple designs.

⚠ Use `setenv OIO_MULTI_TOP 1` to enable the multiple designs feature.

Parameters

`-help`	Prints a brief description for each `set_hierarchy` parameter. *Data_type*: optional
design	Specifies the name of the root design. *Data_type*: optional
`-root`	Specifies the system top. *Data_type*: optional

Examples

- The following set of commands show how you can work with multiple designs:

```
create_design -name P1
create_design -name P2 -current_db
set_hierarchy -root                # Switch to root, see the P1 and P2 in one canvas
set_hierarchy P2                   # Switch to P2, see the P2 design as cell view in one canvas.
```

Related Command

- `create_design`

set_lec_library

```
set_lec_library
[-help]
{{-golden | -revised | -both} {-files files | -reset}}
```

Sets the lec library options.

Parameters

`-help`	Prints a brief description for each `set_lec_library` parameter. *Data_type*: optional
`-both`	Specifies library for both golden and revised usage. *Data_type*: optional
`-files` *`files`*	Specifies the library files. *Data_type*: list of file, optional
`-golden`	Specifies the library for golden usage. *Data_type*: optional
`-reset`	Resets the library. *Data_type*: optional
`-revised`	Specifies the library for revised usage. *Data_type*: optional

Example

- The following specifies the library files for both golden and revised usage:
  ```
  set_lec_library -both -files {./input_data/libs/mmmc/rom_512x16A_slow_syn.lib
  ./input_data/libs/mmmc/ram_256x16A_slow_syn.lib}
  ```

Related Commands

- `run_lec`
- `set_lec_netlist`
- `set_lec_user_preference`
- `view_lec_result`

set_lec_netlist

```
set_lec_netlist
[-help]
{{-golden | -revised} {-files files | -reset}}
```

Sets the lec netlist options.

Parameters

`-help`	Prints a brief description for each `set_lec_netlist` parameter. *Data_type*: optional
`-files` *files*	Specifies the netlist files. *Data_type*: list of file, optional
`-golden`	Specifies the netlist for golden usage. *Data_type*: optional
`-reset`	Resets the netlist. *Data_type*: optional
`-revised`	Specifies the netlist for revised usage. *Data_type*: optional

Example

- The following specifies the netlist files for golden usage:
  ```
  set_lec_netlist -golden -files {./riscv_3dic_v3i/orig_netlist.v.gz}
  ```

Related Commands

- `run_lec`
- `set_lec_library`
- `set_lec_user_preference`
- `view_lec_result`

set_lec_user_preference

```
set_lec_user_preference
[-help]
{{-mode {lec | setup}} {-file file | -reset}}
```

Sets the lec user preferences.

Parameters

`-help`	Prints a brief description for each `set_lec_user_preference` parameter. *Data_type*: optional
`-file` *file*	Specifies the script. *Data_type*: list of file, optional
`-mode {lec | setup}`	Specifies the mode to which the script belongs. *Data_type*: enum, required
`-reset`	Resets the script. *Data_type*: optional

Example

- The following specifies a setup mode script:
  ```
  set_lec_user_preference -mode setup -file ./riscv_3dic_v3i/setup_mode_setting.tcl
  ```

Related Commands

- `run_lec`
- `set_lec_library`
- `set_lec_netlist`
- `view_lec_result`

set_message

```
set_message
[-help]
[-id list_of_msgIDs]
{-suppress | -unsuppress | -limit number | -no_limit}
```

Controls the messages that are displayed for a particular design.

Parameters

`-help`	Prints a brief description for each `set_message` parameter. *Data_type*: optional
`-id` *`list_of_msgIDs`*	Specifies the list of message IDs that require a change in the message severity level. If you do not specify a message ID, the change in severity level is applied to all message IDs. *Data_type*: optional
`-limit` *`number`*	Sets the message limit to *`number`* for every public command. The default limit is 20, which means any specific message will be written out 20 times for one command and then further output of that message will be disabled. An additional message is displayed to inform you that the message limit has been exceeded. If *`number`* is set to `0`, the message will be completely suppressed for the next public command that triggers the message. You can unsuppress the message by setting a number greater than `0`. *Data_type*: optional
`-no_limit`	Removes any existing limit on the selected messages. Suppressed messages are not unsuppressed. *Data_type*: optional
`-suppress`	Suppresses the message. *Data_type*: optional
`-unsuppress`	Unsuppresses the message. *Data_type*: optional

Examples

- The following command suppresses the `DB-001` message:

  ```
  set_message -id DB-001 -suppress
  ```

- The following command unsuppresses the previously suppressed `DB-001` message:

  ```
  set_message -id DB-001 -unsuppress
  ```

- The following command sets the message limit to `10` for every public command:

  ```
  set_message -limit 10
  ```

set_module_model

```
set_module_model
[-help]
-default_dir directory
```

Sets or configures specific block model(s).

Parameters

`-help`	Prints a brief description for each `set_module_model` parameter. *Data_type*: optional
`-default_dir` *`directory`*	
	Specifies the Integrity Hierarchical Database (iHDB) root directory. *Data_type*: required

Related Commands

- `create_module_model`
- `read_module_model`

set_preference

```
set_preference
[-help]
[-reset]
```

Sets or resets preferences.

Parameters

`-help`	Prints a brief description for each `set_preference` parameter. *Data_type*: optional
`-reset`	Resets all preferences. You must restart the application for the reset to take affect. *Data_type*: optional

Example

- The following command resets all preferences. You must restart the application after executing the command:
  ```
  set_preference -reset
  ```

set_property

```
set_property
[-help]
obj_list
property_name
property_value
-cell cell
-type {device | pin | net}
```

Sets the personality of nets, devices, or pins by specifying user-defined properties for them.

Parameters

`-help`	Prints a brief description for each `set_property` parameter. *Data_type*: optional
`obj_list`	Specifies the list of objects for which a property is to be set, represents the key string. *Data_type*: required
`property_name`	Specifies the name of the property. The default reserved property names are as follows: • `cell_inst_group`: For `-type cell_inst` only (Device) • `pin_group`: For `-type pin` only (PinInstance) • `net_group`: For `-type net` only (Net) *Data_type*: required
`property_value`	Specifies the value of the property. *Data_type*: required
`-cell` *`cell`*	Specifies the owner cell of the property definition. *Data_type*: required
`-type {device | pin | net}`	
	Specifies the type of the object. *Data_type*: enum, required

Examples

- The following command sets the `net_group` property for the specified nets:
  ```
  set_property -cell DIE1 -type net {NET1 NET2 NET3} net_group test
  ```
- The following command sets the `device_group` property for the `Bump_1`, `Bump_2`, and `Bump_3` devices:
  ```
  set_property -cell DIE1 -type device {Bump_1 Bump_2 Bump_3} device_group test
  ```

Related Commands

- `get_property`
- `read_property`

set_systemlvs_design_check

```
set_systemlvs_design_check
[-help]
[-bump_list bump_list]
-cell cell
[-edtext edtext]
[-layer_connect layer_connect]
[-layer_map layer_map]
[-layout layout]
[-net_short_device net_short_device]
[-net_short_map net_short_map]
[-reset]
[-rule_deck rule_deck]
[-run]
[-schematic_files schematic_files]
[-schematic_type {verilog | cdl}]
```

Sets the System LVS design checks.

Parameters

`-help`	Prints a brief description for each `set_systemlvs_design_check` parameter. *Data_type*: optional
`-bump_list` *`bump_list`*	Specifies the bump list files. *Data_type*: list of file, optional
`-cell` *`cell`*	Specifies the target cell. *Data_type*: cell, required
`-edtext` *`edtext`*	Specifies the EDTEXT file. *Data_type*: file, optional
`-layer_connect` *`layer_connect`*	Specifies the layer connect file. *Data_type*: file, optional
`-layer_map` *`layer_map`*	Specifies the layer map file. *Data_type*: file, optional
`-layout` *`layout`*	Specifies the layout file. *Data_type*: file, optional
`-net_short_device` *`net_short_device`*	Specifies the net short device file. *Data_type*: file, optional
`-net_short_map` *`net_short_map`*	Specifies the net short map file. *Data_type*: file, optional
`-reset`	Resets the settings. *Data_type*: optional
`-rule_deck` *`rule_deck`*	Specifies the rule deck files. *Data_type*: list of file, optional
`-run`	Runs design checking for this cell. *Data_type*: optional
`-schematic_files` *`schematic_files`*	Specifies the schematic files. *Data_type*: list of file, optional
`-schematic_type {verilog | cdl}`	Specifies the type of the assigned schematic files. *Data_type*: enum, optional

Example

- The following command sets the specified System LVS design checks:
  `set_systemlvs_design_check -cell IMP4330301_3DS -layout imp4330301_3ds.gds -layer_map IMP4330301_3DS_layerMap.csv -layer_connect IMP4330301_3DS_layerConns.csv -run`

Related Commands

- read_systemlvs_cdl_port_config
- run_systemlvs
- set_systemlvs_interface_check
- set_systemlvs_netlist_usage
- view_systemlvs_result

set_systemlvs_interface_check

```
set_systemlvs_interface_check
[-help]
[-erc_align erc_align]
[-erc_area erc_area]
-from_device from_device
[-from_gds_layer from_gds_layer]
-from_layer from_layer
[-idx_query]
[-reset]
[-rule_deck_files rule_deck_files]
[-run]
[-schematic_files schematic_files]
[-schematic_type {verilog | cdl}]
-to_device to_device
[-to_gds_layer to_gds_layer]
-to_layer to_layer
```

Sets the System LVS interface checks.

Parameters

`-help`	Prints a brief description for each `set_systemlvs_interface_check` parameter. *Data_type*: optional
`-erc_align` *`erc_align`*	Specifies the ERC align value (um). *Data_type*: optional
`-erc_area` *`erc_area`*	Specifies the ERC area value (%). *Data_type*: optional
`-from_device` *`from_device`*	Specifies the source device. *Data_type*: required
`-from_gds_layer` *`from_gds_layer`*	Specifies the source side GDS layer. *Data_type*: optional
`-from_layer` *`from_layer`*	Specifies the source layer name. *Data_type*: optional
`-idx_query`	Starts the inter-Die Xtalk query data generation. *Data_type*: optional
`-reset`	Resets the settings. *Data_type*: optional
`-rule_deck_files` *`rule_deck_files`*	Specifies the rule deck files. *Data_type*: list of file, optional
`-run`	Runs interface checking for this contact layer. *Data_type*: optional
`-schematic_files` *`schematic_files`*	Specifies the schematic files. *Data_type*: list of file, optional
`-schematic_type {verilog | cdl}`	Specifies the type of the schematic files. *Data_type*: enum, optional
`-to_device` *`to_device`*	Specifies the target device. *Data_type*: required
`-to_gds_layer` *`to_gds_layer`*	Specifies the target side GDS layer. *Data_type*: optional
`-to_layer` *`to_layer`*	Specifies the target layer name. *Data_type*: required

Example

- The following command sets and runs the System LVS interface checks:
  `set_systemlvs_interface_check -from_device /IMP4330401_3DS -from_layer TOP -to_device /IMP4330301_3DS -to_layer TOP -from_gds_layer TOP_PIN -to_gds_layer TOP_PIN -run`

Related Commands

- `read_systemlvs_cdl_port_config`
- `run_systemlvs`
- `set_systemlvs_design_check`
- `set_systemlvs_netlist_usage`
- `view_systemlvs_result`

set_systemlvs_netlist_usage

```
set_systemlvs_netlist_usage
[-help]
[physical_aware | logical_aware | external]
```

Sets the System LVS netlist usage.

Parameters

`-help`	Prints a brief description for each `set_systemlvs_netlist_usage` parameter. *Data_type*: optional
`[physical_aware | logical_aware | external]`	
	Sets the netlist usage. If you set `physical_aware` or `logical_aware`, the tool will export the netlist correspondingly. Meanwhile, the schematic columns in the GUI will become disabled. In Contrast, if you set as `external`, the tool will not export netlists, and the schematic columns in the GUI will be enabled to let you choose the external netlist. *Data_type*: enum, required

Example

- The following command sets the System LVS interface usage to `physical_aware`:
  `set_systemlvs_netlist_usage physical_aware`

Related Commands

- `read_systemlvs_cdl_port_config`
- `run_systemlvs`
- `set_systemlvs_design_check`
- `set_systemlvs_interface_check`
- `view_systemlvs_result`

set_thermal_ambient_condition

```
set_thermal_ambient_condition
[-help]
[-air_flow air_flow]
[-reset]
[-temperature temperature]
```

Specifies the thermal ambient condition settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_ambient_condition` parameter. *Data_type*: optional
`-air_flow air_flow`	
	Specifies the ambient condition air flow (m/s). *Data_type*: distance, optional
`-reset`	Resets the thermal ambient condition settings. *Data_type*: optional
`-temperature temperature`	
	Specifies the ambient condition temperature (°C). *Data_type*: optional

Example

- The following command sets the thermal ambient condition, as specified:
  ```
  set_thermal_ambient_condition -air_flow 0.5 -temperature 25
  ```

Related Commands

- `set_thermal_contact_attribute`
- `set_thermal_heat_sink`
- `set_thermal_layer_material`
- `set_thermal_molding_compound`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_tech`
- `set_thermal_test_board`
- `set_thermal_user_preference`

set_thermal_contact_attribute

```
set_thermal_contact_attribute
[-help]
-from_device device
-from_layer layer
-to_device device
-to_layer layer
[-pin_height height]
[-pin_material material]
[-pin_underfill_material material]
[-reset]
```

Specifies the thermal contact attribute settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_contact_attribute` parameter. *Data_type*: optional
`-from_device device`	Specifies the device from which the contact layer is created. *Data_type*: required
`-from_layer layer`	Specifies the layer from which the contact layer is created. *Data_type*: required
`-to_device device`	Specifies the device to which the contact layer is created. *Data_type*: required
`-to_layer layer`	Specifies the layer to which the contact layer is created. *Data_type*: required
`[-pin_height height]`	Specifies the contact pin height (micron). *Data_type*: optional
`[-pin_material material]`	Specifies the pin material. *Data_type*: optional
`[-pin_underfill_material material]`	Specifies the pin under-fill material. *Data_type*: optional
`-reset`	Resets the thermal contact attribute settings. *Data_type*: optional

Example

- The following command sets the thermal contact attribute, as specified:
  ```
  set_thermal_contact_attribute -from_device daughter_die -from_layer AP -to_device Interposer -to_layer AP -
  pin_underfill_material AIR -pin_material COPPER -pin_height 5
  ```

Related Commands

- set_thermal_ambient_condition
- set_thermal_heat_sink
- set_thermal_layer_material
- set_thermal_molding_compound
- set_thermal_package
- set_thermal_power
- set_thermal_tech
- set_thermal_test_board
- set_thermal_user_preference

set_thermal_heat_sink

```
set_thermal_heat_sink
[-help]
[-adhesive_thickness adhesive_thickness]
[-reset]
[-size size]
```

Specifies the thermal heat sink settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_heat_sink` parameter. *Data_type*: optional
`-adhesive_thickness` *`adhesive_thickness`*	
	Specifies adhesive thickness(mm). *Data_type*: distance, optional
`-reset`	Resets the thermal heat sink settings. *Data_type*: optional
`-size` *`size`*	Specifies the heat sink size {x(mm), y(mm), thickness(mm)}. *Data_type*: optional

Example

- The following command sets a thermal heat sink with the specified size and adhesive thickness:
  ```
  set_thermal_heat_sink -adhesive_thickness 0.1mm -size {10mm, 10mm, 1mm}
  ```

Related Commands

- `set_thermal_ambient_condition`
- `set_thermal_contact_attribute`
- `set_thermal_layer_material`
- `set_thermal_molding_compound`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_tech`
- `set_thermal_test_board`
- `set_thermal_user_preference`

set_thermal_layer_material

```
set_thermal_layer_material
[-help]
layer
[-dielectric dielectric]
[-metal metal]
[-metal_percentage metal_percentage]
[-reset]
```

Specifies the thermal layer material settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_layer_material` parameter. *Data_type*: optional
`layer`	Specifies the source layers. The format is *`substrate`*`::`*`layer`*. *Data_type*: required
`-dielectric` *`dielectric`*	Specifies the dielectric material. *Data_type*: optional
`-metal` *`metal`*	Specifies the metal material. *Data_type*: optional
`-metal_percentage` *`metal_percentage`*	
	Specifies the metal percentage, range 0-100. *Data_type*: optional
`-reset`	Resets the thermal layer material settings. *Data_type*: optional

Example

- The following command sets thermal material values for the `N5:AP` layer:

```
set_thermal_layer_material N5:AP -metal COPPER -dielectric SILLICON -metal_percentage 50
```

Related Commands

- set_thermal_ambient_condition
- set_thermal_contact_attribute
- set_thermal_heat_sink
- set_thermal_molding_compound
- set_thermal_package
- set_thermal_power
- set_thermal_tech
- set_thermal_test_board
- set_thermal_user_preference

set_thermal_molding_compound

```
set_thermal_molding_compound
[-help]
[-reset]
[-size size]
```

Specifies the thermal molding compound settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_molding_compound` parameter. *Data_type*: optional
`-reset`	Resets the thermal molding compound settings. *Data_type*: optional
`-size size`	Specifies the molding compound size {x(mm), y(mm), thickness(mm)}. *Data_type*: optional

Examples

- The following command sets thermal molding compound values:
  ```
  set_thermal_molding_compound -size {10mm, 10mm, 600um}
  ```
  OR
  ```
  set_thermal_molding_compound -size {10mm, 10mm, 0.6mm}
  ```

Related Commands

- `set_thermal_ambient_condition`
- `set_thermal_contact_attribute`
- `set_thermal_heat_sink`
- `set_thermal_layer_material`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_tech`
- `set_thermal_test_board`
- `set_thermal_user_preference`

set_thermal_package

```
set_thermal_package
[-help]
[-bump_height bump_height]
[-contact_layer contact_layer]
[-dummy_package_size dummy_package_size]
[-real_package_file real_package_file]
[-real_package_name real_package_name]
[-reset]
[-type {real | dummy}]
```

Specifies thermal package settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_package` parameter. *Data_type*: optional
`-bump_height` *`bump_height`*	Specifies the height of the dummy package bump in mm. *Data_type*: distance, optional
`-contact_layer` *`contact_layer`*	Specifies the dummy package contact layer. *Data_type*: optional
`-dummy_package_size` *`dummy_package_size`*	Specifies the dummy package size {x(mm), y(mm)}. *Data_type*: optional
`-real_package_file` *`real_package_file`*	Specifies the real package file. *Data_type*: file, optional
`-real_package_name` *`real_package_name`*	Specifies the real package name. *Data_type*: optional
`-reset`	Resets the package settings. *Data_type*: optional
`-type {real | dummy}`	Specifies the type. *Data_type*: enum, optional

Examples

- The following command sets a real package:
  `set_thermal_package -type real -real_package_file info.mcm -real_package_name INFO`
- The following command sets a dummy package:
  `set_thermal_package -type dummy -dummy_package_size {10mm, 10mm} -contact_layer AP -bump_height 5um`

Related Commands

- set_thermal_ambient_condition
- set_thermal_contact_attribute
- set_thermal_heat_sink
- set_thermal_layer_material
- set_thermal_molding_compound
- set_thermal_power
- set_thermal_tech
- set_thermal_test_board
- set_thermal_user_preference

set_thermal_power

```
set_thermal_power
[-help]
device_path
[-reset]
[-total_power total_power]
[-type {total_power | region_power | vtm}]
[-vtm VTM_file]
```

Specifies thermal power settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_power` parameter. *Data_type*: optional
`device_path`	Specifies the target device. *Data_type*: required
`-reset`	Resets the power settings. *Data_type*: optional
`-total_power` *`total_power`*	Specifies the total power. *Data_type*: optional
`-type {total_power | region_power | vtm}`	Specifies the type. *Data_type*: enum, optional
`-vtm` *`VTM_file`*	Specifies the VTM file. *Data_type*: file, optional

Examples

- The following command sets by total power volume:
  `set_thermal_power -type total -total_power 1.0`
- The following command sets by VTM power:
  `set_thermal_power -type vtm -vtm power_map.vtm`
- The following command sets by current region:
  `set_thermal_power -type region_power`

Related Commands

- set_thermal_ambient_condition
- set_thermal_contact_attribute
- set_thermal_heat_sink
- set_thermal_layer_material
- set_thermal_molding_compound
- set_thermal_package
- set_thermal_power
- set_thermal_tech
- set_thermal_test_board
- set_thermal_user_preference

set_thermal_tech

```
set_thermal_tech
[-help]
[file_name]
[-reset]
```

Specifies the thermal tech settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_tech` parameter. *Data_type*: optional
`file_name`	Specifies the thermal tech file. *Data_type*: file, optional
`-reset`	Resets the thermal tech settings. *Data_type*: optional

Example

- The following command sets the thermal tech based on the `techfile.tech` file:

  ```
  set_thermal_tech techfile.tech
  ```

Related Commands

- `set_thermal_ambient_condition`
- `set_thermal_contact_attribute`
- `set_thermal_heat_sink`
- `set_thermal_layer_material`
- `set_thermal_molding_compound`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_test_board`
- `set_thermal_user_preference`

set_thermal_test_board

```
set_thermal_test_board
[-help]
[-h_size h_size]
[-l_size l_size]
[-reset]
```

Specifies the thermal test board settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_test_board` parameter. *Data_type*: optional
`-h_size` *`h_size`*	Specifies H area size {x(mm), y(mm)}. *Data_type*: optional
`-l_size` *`l_size`*	Specifies L area size {x(mm), y(mm)}. *Data_type*: optional
`-reset`	Resets the thermal layer material settings. *Data_type*: optional

Example

- The following command sets thermal test board values as specified:
  `set_thermal_test_board -h_size {10mm, 10mm} -l_size{20mm, 20mm}`

Related Commands

- `set_thermal_ambient_condition`
- `set_thermal_contact_attribute`
- `set_thermal_heat_sink`
- `set_thermal_layer_material`
- `set_thermal_molding_compound`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_tech`
- `set_thermal_user_preference`

set_thermal_user_preference

```
set_thermal_user_preference
[-help]
[file_name]
[-reset]
```

Specifies the thermal user preference settings.

Parameters

`-help`	Prints a brief description for each `set_thermal_user_preference` parameter. *Data_type*: optional
`file_name`	Specifies the thermal user preference tcl file. *Data_type*: file, optional
`-reset`	Resets the thermal user preference settings. *Data_type*: optional

Example

- The following command sets the thermal user preferences based on the `preference.tcl` file:
  ```
  set_thermal_user_preference preference.tcl
  ```

Related Commands

- `set_thermal_ambient_condition`
- `set_thermal_contact_attribute`
- `set_thermal_heat_sink`
- `set_thermal_layer_material`
- `set_thermal_molding_compound`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_tech`
- `set_thermal_test_board`

set_vsrc_layers

```
set_vsrc_layers
[-help]
-device_path path
-layers layers
[-reset]
```

Sets the voltage source (VSRC) layers.

Parameters

`-help`	Prints a brief description for each `set_vsrc_layers` parameter. *Data_type*: optional
`-device_path` *`path`*	Specifies the target device path. *Data_type*: required
`-layers` *`layers`*	Specifies the voltage source layers. *Data_type*: required
`-reset`	Resets the settings. *Data_type*: optional

Example

- The following command sets the `AP` layer as the voltage source layer for the `Interposer` device path:

  ```
  set_vsrc_layers -layers AP -device_path Interposer
  ```

Related Commands

- `set_era_die_config`
- `set_era_pg_net_config`

show_connections

```
show_connections
[-help]
```

Displays connections.

Parameters

`-help`	Prints a brief description for each `show_connections` parameter. *Data_type*: optional

Related Commands

- remove_connections

snap_cell

```
snap_cell[[
[-help]
[-cell cell]
-device_grid device_grid
-max_delta x_y
-max_snap x_y
-pin_grid pin_grid
-refer path
[-selected]
```

Snaps the target devices to a location that has the same (or smaller than max delta difference) pin grid offset as the reference device.

Within the `-max_snap` move range restriction, `snap_cell` first snaps the target devices to the location *<sx, sy>*, which has the same offset from the specified *pin_grid* as the offset of the reference device *<rx, ry>*. If there is no such location, it will instead find a location that satisfies the following condition:

```
| sx - rx | <= max_delta_x & | xy - ry | <= max_delta_y
```

Parameters

`-help`	Prints a brief description for each `snap_cell` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies that the cell is to be snapped. Snaps all instances of the specified cell to the grid. *Data_type*: optional
`-device_grid` *`device_grid`*	Specifies the name of the device grid. *Data_type*: required
`-max_delta` *`x_y`*	Specifies the maximum pin_grid offset difference between the target devices and the reference device. *Data_type*: required
`-max_snap` *`x_y`*	Specifies the maximum move range of each device. *Data_type*: required
`-pin_grid` *`pin_grid`*	Specifies the name of the pin grid. *Data_type*: required
`-refer` *`path`*	Specifies the path of the reference device with which to align. *Data_type*: required
`-selected`	Specifies that the selected objects (or target devices) are to be snapped. *Data_type*: optional

Examples

- The following command snaps the selected devices to align with the reference device at `/FCCC_N5/FCCC_N5_PB_0`:

```
snap_cell -selected -device_grid device_grid_51x51 -max_delta {0.00012 0.00012}
          -max_snap {0.001 0.001} -pin_grid pin_grid_9x9 -refer /FCCC_N5/FCCC_N5_PB_0
```

- The following command snaps all instances of the cell `N5:FCCC_N5` to align with the reference device at `/FCCC_N5/FCCC_N5_PB_0`:

```
snap_cell -cell N5:FCCC_N5 -device_grid device_grid_51x51 -max_delta {0.00012 0.00012}
          -max_snap {0.001 0.001} -pin_grid pin_grid_9x9 -refer /FCCC_N5/FCCC_N5_PB_0
```

Related Commands

- `check_cell_uniqueness`
- `uniquify_cell`

source_bsh

```
source_bsh
[-help]
file
```

Sources the specified orbit native beanshell script, which has the `.ojs`, `.tjs`, or `.bsh` extension from the tcl interface. This command is similar to the tcl native `source` command.

Parameters

`-help`	Prints a brief description for each `source_bsh` parameter. *Data_type*: optional
file	Specifies the location of the `.ojs`, `.tjs`, or `.bsh` file in the beanshell. *Data_type*: file, required

Example

- The following command sources the `testGroupBumps.tjs` file from the specified location:

```
source_bsh RegressionTest/17.2-Dev/base/Device/GroupDevices/testGroupBumps.tjs
```

Related Command

- `eval_bsh`

space_selected

```
space_selected
[-help]
-fix_side {left | right | top | bottom}
-space value
```

Spaces objects horizontally or vertically by a specified distance value, and can evenly distribute the spacing horizontally or vertically between three or more objects. Use this command after selecting two or more objects.

Parameters

`-help`	Prints a brief description of each `space_selected` parameter. *Data_type*: optional
`-fix_side {left | right | top | bottom}`	
	Specifies the vertical and horizontal object spacing. Choose one of the following: • `left`: Specifies that the leftmost object is fixed and all other objects are shifted horizontally to the right side of the fixed object, maintaining space defined in the `-space` option. • `right`: Specifies that the rightmost object is fixed and all other objects are shifted horizontally to the left side of the fixed object, maintaining space defined in the `-space` option. • `top`: Specifies that the topmost object is fixed and all other objects are shifted vertically to the down side of the fixed object, maintaining space defined in the `-space` option. • `bottom`: Specifies that the bottom-most object is fixed and all other objects are shifted vertically to the up side of the fixed object, maintaining space defined in the `-space` option. *Data_type*: enum, required
`-space` *`value`*	Specifies the spacing value. *Data_type*: required

Example

- The following command fixes the leftmost highlighted object and moves all the other objects horizontally to its right, spacing the objects 200 micrometers apart from each other:

```
space_selected -fix_side left -space 200
```

Related Commands

- `align_selected`
- `clear_selection`
- `deselect_obj`
- `get_selection`
- `list_select_properties`
- `select_obj`

swap_pin_nets

```
swap_pin_nets
[-help]
[-selected]
[-swap_contact_pins]
```

Swaps pin nets.

Parameters

`-help`	Prints a brief description for each `swap_pin_nets` parameter. *Data_type*: optional
`-selected`	Specifies that nets should be swapped for the two selected objects. *Data_type*: optional
`-swap_contact_pins`	Specifies that contact pins are to be swapped together. *Data_type*: optional

Example

- The following set of commands selects pins in the specified rectangular area and swaps their nets. The rectangular area should be specified such that only the two required pins are selected:

```
select_obj -type pin -rect {-20066 -19779 -16022 -17958}
swap_pin_nets -selected
```

- The following set of commands selects pins for the specified device and swaps contact pins together:

```
select_obj -type pin -device_path /BGA_BOT/TOP -prop_name name -prop_values {BUMP_BOTTOM_161}
select_obj -type pin -device_path /BGA_BOT/TOP -prop_name name -prop_values {BUMP_BOTTOM_158}
swap_pin_nets -selected -swap_contact_pins
```

synthesize_boundary

```
synthesize_boundary
[-help]
[-cover path_list]
-device device_path
[-enclosure left_bottom_right_top]
[-scribe_line left_bottom_right_top]
[-seal_ring left_bottom_right_top]
```

Synthesizes boundary.

Parameters

`-help`	Prints a brief description for each `synthesize_boundary` parameter. *Data_type*: optional
`-cover` *`path_list`*	Specifies the covered device list. *Data_type*: optional
`-device` *`device_path`*	Specifies the synthesize device. *Data_type*: required
`-enclosure` *`left_bottom_right_top`*	Specifies the enclosure clearance. *Data_type*: optional
`-scribe_line` *`left_bottom_right_top`*	Specifies the scribe line remained. *Data_type*: optional
`-seal_ring` *`left_bottom_right_top`*	Specifies the seal ring. *Data_type*: optional

Example

```
synthesize_boundary -device {/top/INTERPOSER} \
    -cover {/top/ASIC_IPG /top/JESD235B-HBM_Ballout} \
    -enclosure {70 70 70 70} \
    -seal_ring {100 100 100 100} \
    -scribe_line {120 120 120 120}
```

test_same_oio

```
test_same_oio
[-help]
[-current current_file]
-golden golden
[-ignore pattern]
[-max_diff diff]
```

Tests the same OIO using the current oio and the golden oio file.

Parameters

`-help`	Prints a brief description for each `test_same_oio` parameter. *Data_type*: optional
`-current` *`current_file`*	
	Specifies the current oio file. *Data_type*: file, optional
`-golden` *`golden`*	Specifies the reference (golden) oio file. *Data_type*: file, required
`-ignore` *`pattern`*	Specifies ignore patterns. *Data_type*: optional
`-max_diff` *`diff`*	Specifies the number of diff in print. *Data_type*: optional

Related Command

- `test_same_text`

test_same_text

```
test_same_text
[-help]
[-comment c]
-current current_file
-golden golden
[-ignore_order]
[-ignore_pattern pattern]
[-ignore_size]
[-max_diff diff]
[-result output_file]
[-result_header title]
[-skip_blank_line]
[-tolerance tolerance]
```

Compares the text in the current oio file with that in the golden oio file.

Parameters

`-help`	Prints a brief description for each `test_same_text` parameter. *Data_type*: optional
`-comment` *`c`*	Specifies the character representing the comment line. *Data_type*: optional
`-current` *`current_file`*	Specifies the current oio file. *Data_type*: file, required
`-golden` *`golden`*	Specifies the reference (golden) oio file. *Data_type*: file, required
`-ignore_order`	Specifies that line order should be ignored in the comparison. *Data_type*: optional
`-ignore_pattern` *`pattern`*	Specifies ignore patterns. *Data_type*: optional
`-ignore_size`	Specifies that file size should be ignored in the comparison. *Data_type*: optional
`-max_diff` *`diff`*	Specifies the number of diff in print. *Data_type*: optional
`-result` *`output_file`*	Specifies the diff result file. *Data_type*: optional
`-result_header` *`title`*	Specifies the header of the diff result file. *Data_type*: optional
`-skip_blank_line`	Specifies that blank lines should be skipped in the comparison. *Data_type*: optional
`-tolerance` *`tolerance`*	Specifies the number of tolerance diff lines. *Data_type*: optional

Related Command

- `test_same_oio`

thermal::view_result

```
thermal::view_result
[-help]
[-result result]
-root root
-tag tag
-top top
```

Displays existing thermal analysis result.

Parameters

`-help`	Prints a brief description for each `thermal::view_result` parameter. *Data_type*: optional
`-result` *`result`*	Specifies an existing result file. *Data_type*: optional
`-root` *`root`*	Specifies the iHDB root. *Data_type*: file, required
`-tag` *`tag`*	Specifies the tag. *Data_type*: required
`-top` *`top`*	Specifies the top name. *Data_type*: required

Example

- The following command displays the specified thermal result file:
  `thermal::view_result -root Run_Dir -top riscv_top -tag 0608 -result result_Result_Files_060722_231358`

Related Commands

- `run_thermal`
- `set_thermal_ambient_condition`
- `set_thermal_contact_attribute`
- `set_thermal_heat_sink`
- `set_thermal_layer_material`
- `set_thermal_molding_compound`
- `set_thermal_package`
- `set_thermal_power`
- `set_thermal_tech`
- `set_thermal_test_board`

unassign_bumps

```
unassign_bumps
[-help]
-selected
```

Unassigns the bumps from selected pins.

Parameters

`-help`	Prints a brief description for each `unassign_bumps` parameter. *Data_type*: optional
`-selected`	Specifies only selected pins. *Data_type*: required

Example

- The following set of commands unassigns the bumps from selected pins:
```
select_obj -type pin -rect {1695.451851 2338.115741 2041.271541 2687.157354}
unassign_bumps -selected
```

Related Commands

- `assign_bumps`
- `assign_pg_bumps`

ungroup

```
ungroup
[-help]
object
```

Ungroups the specified module.

Parameters

`-help`	Prints a brief description for each `ungroup` parameter. *Data_type*: optional
`object`	Specifies the module path to be ungrouped. *Data_type*: instance path, required

Examples

- The following set of commands first creates a logical group named `logical` and then ungroups it:

  ```
  group {/DIE_AC/pin_1 /DIE_AC/pin_2} -name logical
  ungroup {/DIE_AC/logical}
  ```

- The following set of commands first creates a physical group named `physical` and then ungroups it:
  group {/DIE_AC/pin_3 /DIE_AC/pin_4} -name physical -type physical

  ```
  ungroup {/DIE_AC/physical}
  ```

Related Commands

- `create_group`
- `delete_group`
- `group`

ungroup_device

```
ungroup_device
[-help]
path
```

Ungroups the cover, user, or logic device.

Parameters

`-help`	Prints a brief description for each `ungroup_device` parameter. *Data_type*: optional
path	Specifies the referred device path. *Data_type*: required

Example

- The following command ungroups the device at the specified path:
  `ungroup_device {/top/INTERPOSER/Center};`

uniquify_cell

```
uniquify_cell
[-help]
[-cell cell]
-pin_grid grid
[-selected]
```

Uniquifies cells with the specified grid. Use this command to duplicate the template for the devices that have a different offset from the pin grid.

Example:

Device	Offset from grid	Template	Uniquified template
PB1	(0.2 0.2)	PB	PB
PB2	(0.5 0.5)	PB	PB_1
PB3	(0.5 0.3)	PB	PB_2
PB4	(0.5 0.5)	PB	PB_1

Parameters

`-help`	Prints a brief description for each `uniquify_cell` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell and uniquifies all cell instances. *Data_type*: optional
`-pin_grid` *`grid`*	Specifies the name of the pin grid to check. *Data_type*: required
`-selected`	Specifies that selected objects are to be uniquified. *Data_type*: optional

Examples

- The following command uniquifies the selected cell with the pin grid named `pin_grid_9x9`:

  ```
  uniquify_cell -selected -pin_grid pin_grid_9x9
  ```

- The following command uniquifies the specified cell, `N5:FCCC_N5`, with the pin grid named `pin_grid_9x9`:

  ```
  uniquify_cell -cell N5:FCCC_N5 -pin_grid pin_grid_9x9
  ```

Related Commands

- `check_cell_uniqueness`
- `snap_cell`

unplace_obj

```
unplace_obj
[-help]
[-all]
[-blocks]
[-bumps]
-cell cell
[-cores]
```

Unplaces objects.

Parameters

`-help`	Prints a brief description of each `unplace_obj` parameter. *Data_type*: optional
`-all`	Unplaces all the objects placed on the owner cell. *Default*: `true` *Data_type*: optional
`-blocks`	Unplaces all the cell instances that are blocks placed on the owner cell. *Data_type*: optional
`-bumps`	Unplaces all the cell instances that are bumps placed on the owner cell. *Data_type*: optional
`-cell cell`	Specifies the owner cell. *Data_type*: cell, required
`-cores`	Unplaces all the cell instances that are cores placed on the owner cell. *Data_type*: optional

Examples

- The following command unplaces all the objects placed on the `INTERPOSER` cell:

  ```
  unplace_obj -cell INTERPOSER -all
  ```

- The following command unplaces all bumps on the `INTERPOSER` cell:

  ```
  unplace_obj -cell INTERPOSER -bumps
  ```

Related Commands

- `place_inst`

update_contact_layer

```
update_contact_layer
[-help]
[-from_device device]
[-from_layer layer]
[-interface_name name]
[-new_from_layer layer]
[-new_to_layer layer]
[-to_device device]
[-to_layer layer]
```

Updates a contact layer.

Parameters

`-help`	Prints a brief description for each `update_contact_layer` parameter. *Data_type*: optional
`-from_device device`	
	Specifies the device from which the contact layer is created. *Data_type*: optional
`-from_layer layer`	Specifies the layer from which the contact layer is created. *Data_type*: optional
`-interface_name name`	
	Specifies the unique short name for the contact layer. *Data_type*: optional
`-new_from_layer layer`	
	Specifies the new from layer. *Data_type*: optional
`-new_to_layer layer`	Specifies the new to layer. *Data_type*: optional
`-to_device device`	Specifies the device to which the contact layer is created. *Data_type*: optional
`-to_layer layer`	Specifies the layer to which the contact layer is created. *Data_type*: optional

Examples

- The following command updates a contact layer with both from and to layers:

```
update_contact_layer -from_device /ASIC \
                     -from_layer AP \
                     -to_device INTERPOSER \
                     -to_layer AP \
                     -new_from_layer RV \
                     -new_to_layer VIA2
```

- The following command updates a contact layer by the interface name:

```
update_contact_layer -interface_name inter_chip_f2f \
                     -new_from_layer RV \
                     -new_to_layer VIA2
```

- The following command updates a contact layer with the from layer:

```
update_contact_layer -from_device /ASIC \
                     -from_layer AP \
                     -to_device INTERPOSER \
                     -to_layer AP \
                     -new_from_layer RV
```

- The following command updates a contact layer with the to layer:

```
update_contact_layer -from_device /ASIC \
                     -from_layer AP \
                     -to_device INTERPOSER \
                     -to_layer AP \
                     -new_to_layer VIA2
```

Related Commands

- `create_contact_layer`
- `delete_contact_layer`

update_current_region

```
update_current_region
[-help]
-cell cell
[-current current]
[-layer layer]
-name name
[-net net]
[-new_name name]
[-power power]
[-power_density power_density]
[-region region]
[-threshold threshold]
[-voltage voltage]
```

Updates the specified current region.

Parameters

`-help`	Prints a brief description for each `create_current_region` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell owner. *Data_type*: required
`-current` *`current`*	Specifies the new current level in mA. *Data_type*: optional
`-layer` *`layer`*	Specifies the new layer for the current region. *Data_type*: optional
`-name` *`name`*	Specifies the name of the current region to be updated. *Data_type*: required
`-net` *`net`*	Specifies the new net for the current region. *Data_type*: optional
`-new_name` *`name`*	Specifies the new name of the current region. *Data_type*: optional
`-power` *`power`*	Specifies the power value in W. *Data_type*: optional
`-power_density` *`power_density`*	
	Specifies the power density in W/mm^2. *Data_type*: optional
`-region` *`region`*	Specifies the new rectangular area for the current region. *Data_type*: optional
`-threshold` *`threshold`*	Specifies the threshold of nets in V. *Data_type*: optional
`-voltage` *`voltage`*	Specifies the voltage of nets in V. *Data_type*: optional

Example

- The following command changes the name of the current region from `region1` to `regionA`:

```
update_current_region -cell INTERPOSER -name region1 -new_name regionA
```

Related Commands

- `create_current_region`
- `delete_current_region`

update_device

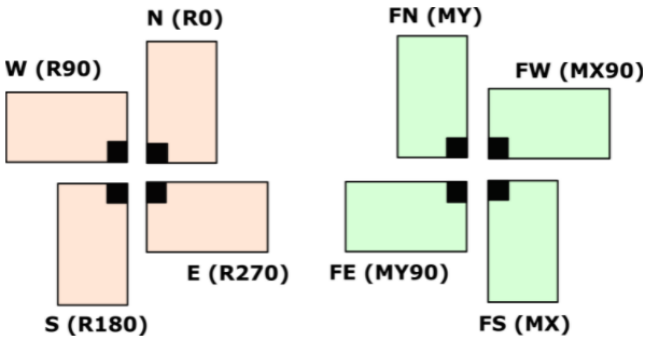
```
update_device
[-help]
[-height height]
[-name name]
[-new_device_name new_name]
[-new_template new_template]
[-new_type {die | pkg | board | cover | block | core | endcap | bump | pad
           |  ip_component | mechanical | ring | hardmacro | softmacro | stdcell
           | contact_device | bondfinger | thermal_component | unknown} ]
[-parent parent]
[-polygon polygon]
[-scribe_line left_bottom_right_top]
[-seal_ring left_bottom_right_top]
[-selected]
[-size width_height]
[-z z]
[ [[-rotation rotate] [-mirror {x | y | no}] [-flip {yes | no}]]
 |[[-orientation {r0 | r90 | r180 | r270 | my | my90 | mx | mx90}]  [-face_up {yes | no}]]]
[-location point | -llx llx -lly lly | -cx cx -cy cy]
[-llx llx -lly lly [-include_extent]]
[-distribute_height_to_layers | -distribute_height_from_layers]
```

Updates the selected or specified device.

Parameters

`-help`	Prints a brief description of each `update_device` parameter. *Data_type*: optional
`-cx cx`	Moves the center of the current device to the specific x-coordinate of the parent. *Data_type*: optional
`-cy cy`	Moves the center of the current device to the specific y-coordinate of the parent. *Data_type*: optional
`-distribute_height_from_layers`	
	Specifies that the device height should be the sum of the thickness of layers. *Data_type*: optional
`-distribute_height_to_layers`	
	Specifies that the device height should be divided uniformly into the thickness of layers. *Data_type*: optional
`-face_up {yes | no}`	Specifies the flip and mirror x of the device. *Default*: `yes` *Data_type*: enum, optional
`-flip {yes | no}`	Flips the device and inverts the layer stack. *Data_type*: enum, optional

`-height` *`height`*	Updates the height of the device. *Data_type*: optional
`-include_extent`	Specifies whether to consider the seal ring and scribe to adjust the geometry information, which affects the `-llx -lly` result. *Data_type*: enum, optional
`-llx` *`llx`*	Moves the lower-left x-coordinate of the device. *Data_type*: optional
`-lly` *`lly`*	Moves the lower-left y-coordinate of the device. *Data_type*: optional
`-location` *`point`*	Sets the new location for the device. *Data_type*: optional
`-mirror {x | y | no}`	Mirrors the device about its y axis. *Data_type*: enum, optional
`-name` *`name`*	Specifies the existing name of the device. *Data_type*: optional
`-new_device_name` *`new_name`*	
	Updates the device with the new device name. *Data_type*: optional
`-new_template` *`new_template`*	
	Replaces the existing template with the new template. *Data_type*: optional
`-new_type {die | pkg | board | cover | block | core | endcap | bump | pad | ip_component | mechanical | ring | hardmacro | softmacro | stdcell | contact_device | bondfinger | thermal_component | unknown}`	
	Updates the device with the type of the new template. *Data_type*: enum, optional
`-orientation {r0 | r90 | r180 | r270 | my | my90 | mx | mx90}`	

	Specifies the new orientation.  *Data_type*: enum, optional
`-parent` *parent*	Specifies the parent template to which the device belongs. *Data_type*: optional
`-polygon` *polygon*	Specifies the polygon boundary of device. For example: `{x1 y1 x1 y2 ... xn yn}` *Data_type*: optional
`-rotation` *rotate*	Rotates the device in degrees in a counter-clockwise/clockwise direction. For example, `90 cw` or `135 ccw` *Default*: clockwise *Data_type*: optional
`-scribe_line` *left_bottom_right_top*	
	Specifies the scribe line. *Data_type*: optional
`-seal_ring` *left_bottom_right_top*	
	Specifies the seal ring. *Data_type*: optional
`-selected`	Updates the selected devices only. You need not specify `-name` or `-parent` if a selected device is being modified. *Data_type*: optional

`-size width_height`	Updates the size of the device. *Data_type*: optional
`-z z`	Sets the z-location of the design stacking. *Data_type*: optional

Examples

- The following command updates the name of the device from `ASIC_IPG` to `TMP`:

```
update_device -name ASIC_IPG \
              -parent PKG:top \
              -new_device_name TMP
```

- The following command updates the geometry information of the device named `ASIC_IPG`:

```
update_device -name ASIC_IPG \
              -parent PKG:top \
              -face_up no \
              -llx 100 \
              -lly 500 \
              -z 500 \
              -height 10
```

- The following command updates the die extents:

```
update_device -name INTERPOSER \
              -parent PKG:top \
              -scribe_line {100 100 100 100} \
              -seal_ring {200 200 200 200}
```

- The following command replaces the existing `MyPackage_A` template with the new `MyPackageNew` template:

```
update_device -parent design \
              -name MyPackage_A \
              -new_template MyPackageNew
```

- The following command updates the polygon outline of the specified device:

```
update_device -name INTERPOSER_TEST \
    -parent {PKG:top} \
    -polygon {0 0 1000 0 1000 2000 0 2000}
```

- The following commands rotate the device as specified:
  - `update_device -name DIE -parent design -rotation {90 cw}`
  - `update_device -name DIE -parent design -rotation {90}      # = 90 cw`
  - `update_device -name DIE -parent design -rotation {90.0 ccw};`
- The following command adjusts the device height to be the sum of the thickness of layers:

```
update_device -name INTERPOSER -parent PKG:top -distribute_height_from_layers
```

- The following command divides the device height uniformly into the thickness of each layer:

```
update_device -name INTERPOSER -parent PKG:top -distribute_height_to_layers
```

Related Commands

- `create_device`
- `move_device`

update_device_template

```
update_device_template
[-help]
-name name
[-new_name new_name]
[-scribe_line left_bottom_right_top]
[-seal_ring left_bottom_right_top]
[-size width_height | -polygon polygon]
 -substrate substrate
[-type {die | pkg | board | cover | block | core | endcap | bump | group | pad | partition_block | spacer | ip_component |
mechanical | ring | hardmacro | softmacro | stdcell | contact_device | bondfinger | thermal_component | unknown}]
```

Updates the selected or specified device.

Parameters

`-help`	Prints a brief description of each `update_device_template` parameter. *Data_type*: optional
`-name` *`name`*	Specifies the name of the existing device template. *Data_type*: required
`-new_name` *`new_name`*	Specifies the new device template name. *Data_type*: optional
`-polygon` *`polygon`*	Specifies the polygon boundary of the device. For example: `{x1 y1 x1 y2 ... xn yn}` *Data_type*: optional
`-scribe_line` *`left_bottom_right_top`*	
	Specifies the scribe line. *Data_type*: optional
`-seal_ring` *`left_bottom_right_top`*	
	Specifies the seal ring. *Data_type*: optional
`-size` *`width_height`*	Specifies the size of the device. *Data_type*: optional
`-substrate` *`substrate`*	Specifies the substrate to which the device template belongs. *Data_type*: required
`-type {die | pkg | board | cover | block | core | endcap | bump | group | pad | partition_block | spacer | ip_component | mechanical | ring | hardmacro | softmacro | stdcell | contact_device | bondfinger | thermal_component | unknown}`	
	Specifies the type of the new template. *Data_type*: enum, optional

Example

- The following command updates the device template by the specified size:
  ```
  update_device_template -substrate DIE -name DIE1 -new_name DIE2 -type pkg -size {4000 6000}
  ```

Related Commands

- `create_device_template`
- `update_device`

update_padstack

```
update_padstack
[-help]
[-delete_layer layer_name]
[-layer_shape layer:type:args]
-name name
[-pad_stack layer:type:args]
-substrate substrate
```

Updates the padstack with the specified name.

Parameters

`-help`	Prints a brief description for each `update_padstack` parameter. *Data_type*: optional
`-delete_layer layer_name`	
	Removes shapes on the specified layers. *Data_type*: optional
`-layer_shape layer:type:args`	
	Updates shapes on the specified layer. The format is `layer:type:args`. For example: `-layer_shape {{Metal1:Circle:50}{Metal2:Rect:50 100}}` *Data_type*: optional
`-name name`	Specifies the name of the padstack. *Data_type*: required
`-pad_stack layer:type:args`	
	Replaces all layer shapes. The format is `layer:type:args`. For example: `-pad_stack {{Metal1:Circle:50}{Metal2:Rect:50 100}}` *Data_type*: optional
`-substrate substrate`	Specifies the substrate for which the padstack is being updated. *Data_type*: required

Examples

- The following command replaces all layer shapes:
```
update_padstack -name PKG_PAD -substrate PKG \
                -pad_stack {{C4Bump:Circle:50}{Ball:Oblong:50 100}}
```
- The following command removes the shapes on the specified layers:
```
update_padstack -name PKG_PAD -substrate PKG \
                -delete_layer {C4Bump Ball}
```
- The following command updates the shapes on the specified layer:
```
update_padstack -name PKG_PAD -substrate PKG \
                -layer_shape {C4Bump:Rect:50 60}
```

Related Commands

- `create_padstack`
- `delete_padstack`

update_shrink_factor

```
update_shrink_factor
[-help]
-shrink_factor factor
{-substrate substrate | -cell cell}
```

Updates the shrink factor of the specified substrate or cell.

Parameters

`-help`	Prints a brief description for each `update_shrink_factor` parameter. *Data_type*: optional
`-cell` *`cell`*	Specifies the cell for which shrink factor is to be updated. *Data_type*: optional
`-shrink_factor` *`factor`*	Specifies the shrink factor of the substrate. Factor of `1.0` means no shrink. *Data_type*: required
`-substrate` *`substrate`*	Specifies the substrate for which shrink factor is to be updated. *Data_type*: optional

Examples

- The following command updates the shrink factor of the `DIE` substrate to `0.8`:

  ```
  update_shrink_factor -substrate DIE -shrink_factor 0.8
  ```

- The following command updates the shrink factor of the `DIE` substrate to `1`:

  ```
  update_shrink_factor -substrate DIE -shrink_factor 1
  ```

- The following command updates the shrink factor of the `BGA_PKG` cell to `0.8`:

  ```
  update_shrink_factor -cell BGA_PKG -shrink_factor 0.8
  ```

update_stitch_zone

```
update_stitch_zone
[-help]
[-c_height c_height]
[-c_width c_width]
-cell cell
-name name
[-new_name new_name]
[-x x]
[-x_width x_width]
[-y y]
[-y_width y_width]
```

Updates the specified stich zone.

Parameters

`-help`	Prints a brief description for each `update_stitch_zone` parameter. *Data_type*: optional
`-c_height c_height`	Specifies the height of the cross intersection. *Data_type*: optional
`-c_width c_width`	Specifies the width of the cross intersection. *Data_type*: optional
`-cell cell`	Specifies the cell owner. *Data_type*: required
`-name name`	Specifies the name of the stitch zone. *Data_type*: required
`-new_name new_name`	Specifies the new name of the stitch zone. *Data_type*: required
`-x x`	Specifies the x-coordinate. *Data_type*: optional
`-x_width x_width`	Specifies the width in x-coordinate. *Data_type*: optional
`-y y`	Specifies the y-coordinate. *Data_type*: optional
`-y_width y_width`	Specifies the width in y-coordinate. *Data_type*: optional

Example

- The following command updates the `x` coordinate of the stitch zone named `C4Bump` with the `INTERPOSER:INTERPOSER` cell to `3000`:

  ```
  update_stitch_zone -cell INTERPOSER:INTERPOSER -name C4Bump -x 3000
  ```

Related Commands

- `create_stitch_zone`
- `delete_stitch_zone`

update_substrate

```
update_substrate
[-help]
[-height height]
[-info_tech_file info_tech_file | -lefs lef_files]
-name name
[-new_name new_name]
```

Updates the specified substrate as required.

Parameters

`-help`	Prints a brief description for each `update_substrate` parameter. *Data_type*: optional
`-height` *`height`*	Specifies the height of the substrate. *Data_type*: optional
`-info_tech_file` *`info_tech_file`*	
	Specifies the info tech file for updating the substrate. *Data_type*: optional
`-lefs` *`lef_files`*	Updates the LEF files for the substrate. Use this option if you want to override the existing LEF files in the design. The list of all LEF files used by this design must be given. *Data_type*: optional
`-name` *`name`*	Specifies the substrate that needs to be updated. *Data_type*: required
`-new_name` *`new_name`*	
	Renames the substrate as specified. *Data_type*: optional

Examples

- The following command changes the height of the substrate named `PKG`:

  ```
  update_substrate -name PKG -height 200
  ```

- The following command updates the LEF files for the `GPDK` substrate:

  ```
  update_substrate -name GPDK -lefs { \
  lef/bump.lef \
  lef/HB2.lef \
  lef/HB2_MACRO.lef}
  ```

Related Command

- `create_substrate`
- `delete_substrate`

view_lec_result

```
view_lec_result
[-help]
{{-workspace workspace} | {{-run_dir run_dir} {-tag tag}}}
```

Shows the LEC flow result.

Parameters

`-help`	Prints a brief description for each `view_lec_result` parameter. *Data_type*: optional
`-run_dir` *`run_dir`*	Specifies the workspace with *`run_dir`* and *`tag`*; this argument assigns the run directory. You cannot set this argument and `-workspace` simultaneously. *Data_type*: file, optional
`-tag` *`tag`*	Specifies the workspace with *`run_dir`* and *`tag`*; this argument assigns the tag. You cannot set this argument and `-workspace` simultaneously. *Data_type*: optional
`-workspace` *`workspace`*	Specifies the workspace directly. You cannot set this argument and`-run_dir` simultaneously. *Data_type*: file, optional

Example

- The following command shows the LEC flow result for the specified workspace:
  `view_lec_result -workspace /user/Desktop/myLecTest/`

Related Commands

- `run_lec`
- `set_lec_library`
- `set_lec_netlist`
- `set_lec_user_preference`

view_systemlvs_result

```
view_systemlvs_result
[-help]
[-workspace workspace]
```

Displays the System LVS run result in the GUI.

Parameters

`-help`	Prints a brief description for each `view_systemlvs_result` parameter. *Data_type*: optional
`-workspace` *`workspace`*	Specifies the workspace. *Data_type*: file, optional

Example

- The following command displays the System LVS run result:
  `view_systemlvs_result -workspace /myLvsTest/SystemLvs.run`

Related Commands

- `read_systemlvs_cdl_port_config`
- `run_systemlvs`
- `set_systemlvs_design_check`
- `set_systemlvs_interface_check`
- `set_systemlvs_netlist_usage`

write_allegro

```
write_allegro
[-help]
file
-device_path path
[-open]
```

Saves an Multi-Chip Module (MCM) file.

Parameters

`-help`	Prints a brief description for each `write_allegro` parameter. *Data_type*: optional
`file`	Specifies the name of the output file. *Data_type*: file, required
`-device_path` *`path`*	Specifies the target device path. *Data_type*: required
`-open`	Specifies whether to open the Allegro gui. *Data_type*: optional

Example

- The following command saves the MCM file `INFO.mcm`:

```
write_allegro INFO.mcm -device_path INFO
```

Related Command

- `read_allegro`

write_bga_text

```
write_bga_text
[-help]
[-coord {center | origin}]
-device_path device_path
-file file
[-format {default | rename_pin_matrix | allegro_csv | allegro_netlist_csv | rename_pin_property}]
[-pin_types pin_types]
[-process_shrink scale]
```

Exports the Ball Grid Array (BGA) information for the specified device to a BGA text file.

Parameters

`-help`	Prints a brief description for each `write_bga_text` parameter. *Data_type*: optional
`-coord {center | origin}`	Specifies the coordinates. *Data_type*: enum, optional
`-device_path` *`device_path`*	Specifies the device path for which BGA information is to be exported. *Data_type*: required
`-file` *`file`*	Specifies the BGA text output file. *Data_type*: file, required
`-format {default | rename_pin_matrix | allegro_csv | allegro_netlist_csv | rename_pin_property}`	
	Specifies the output format. *Default*: `default` *Data_type*: enum, optional
`-pin_types` *`pin_types`*	Specifies the pin types to export. *Data_type*: optional
`-process_shrink` *`scale`*	Specifies the shrink factor for the device. For example: 0.75 to scale to 1/0.75 * 100% *Data_type*: optional

Example

- The following command exports from the device path `BGA` to the specfied text file:

```
write_bga_text -device_path BGA -file $outFile -process_shrink 1.0 -pin_types {ballpad bondfingerpad contact}
```

Related Commands

- `read_bga_text`
- `read_die_text`
- `write_die_text`

write_bump_locations

```
write_bump_locations
[-help]
{-cell cell | -device device}
-file_name file_name
[-origin {origin | lower_left | center}]
[-selected]
-version {simple | advance}
```

Writes out bump locations.

Parameters

`-help`	Prints a brief description of each `write_bump_locations` parameter. *Data_type*: optional
`-cell cell`	Specifies the output target. *Data_type*: cell, optional
`-device device`	Specifies the device name of the output target. *Data_type*: optional
`-file_name file_name`	Specifies the name of the txt output file. *Data_type*: file, required
`-origin {origin | lower_left | center}`	
	Specifies the bump location option. *Default*: `center` *Data_type*: enum, optional
`-selected`	Writes only the selected bumps. *Data_type*: optional
`-version {simple | advance}`	
	Specifies the output format version: • `simple`: Innovus bump table format • `advance`: TSMC bump location file format *Data_type*: enum, required

Example

- The following command writes selected bump locations to the `out.txt` file in a bump table format:
  ```
  write_bump_locations -file_name out.txt -cell FCCC_BOT -version simple -selected
  ```

Related Command

- `read_bump_locations`

write_contact_pin_map_file

```
write_contact_pin_map_file
[-help]
-file file
```

Writes the pin map (`pinMap.csv`) for the selected contact layer cell to the specified path.

Parameters

`-help`	Prints a brief description for each `write_contact_pin_map_file` parameter. *Data_type*: optional
`-file` *`file`*	Specifies the path of the file to be written. *Data_type*: file, required

Example

- The following command writes the pin map filefor the selected contact layer cell :
  ```
  write_contact_pin_map_file -file out
  ```

write_def

```
write_def
[-help]
def_file
[-exclude_cells cellNames]
-top_module path
```

Writes out the DEF file for the specified module.

Parameters

`-help`	Prints a brief description of each `write_def` parameter. *Data_type*: optional
`-exclude_cells` *`cellNames`*	Excludes the instances of the specified cells. *Data_type*: optional
`def_file`	Specifies the DEF output file. *Data_type*: file, required
`-top_module` *`path`*	Writes out the specified module along with its submodules. *Data_type*: required

Example

- The following command writes out the `test.def` file for the specified module, while excluding the specified cells:

  ```
  read_def DEF.def -substrate substrate write_def test.def -top_module /DIE1 -exclude_cells {BONDPAD60 BONDPAD52 IORINGDO
  IORINGDI BONDPAD80}
  ```

Related Command

- `read_def`

write_design_stack

```
write_design_stack
[-help]
xml_file
[-export_each_stacking]
[-version {ver1_0 | ver2_0}]
```

Writes the design stack to the specified file.

Parameters

`-help`	Prints a brief description of each `write_design_stack` parameter. *Data_type*: optional
`xml_file`	Specifies the XML file to which the die stack will be written. *Data_type*: file, required
`-export_each_stacking`	
	Specifies whether each stacking is exported. *Data_type*: optional
`-version {ver1_0 | ver2_0}`	
	Specifies whether the export version is `ver1_0` or `ver2_0`. *Default*: `ver2_0` *Data_type*: enum, optional

Examples

- The following command writes the die stack to the `top.xml` file:

```
write_design_stack top.xml
```

Related Command

- `read_design_stack`

write_device_placement

```
write_device_placement
[-help]
csv_file
[-cell template]
[-filter_cell filter_cell]
[-origin {lower_left | lower_right | upper_left | upper_right | center | origin }]
```

Writes the device placement information to the specified CSVfile.

Parameters

`-help`	Prints a brief description for each `write_device_placement` parameter. *Data_type*: optional
csv_file	Specifies the CSV file to be written. *Data_type*: file, required
`-cell` *template*	Specifies the cell for which to write the CSV. You can specify a device template name or top name. *Data_type*: optional
`-filter_cell` *filter_cell*	
	Specifies the list of cell names. *Data_type*: optional
`-origin {lower_left | lower_right | upper_left | upper_right | center | origin }`	
	Specifies the placement origin. *Data_type*: enum, optional

Example

- The following command writes the device placement information to the specified file for the `design` cell:

```
write_device_placement 1-2-customized_placement.csv \
                       -cell design \
                       -origin lower_left \
                       -filter_cell {SOC HBM}
```

Related Commands

- `align_device_placement`
- `read_device_placement`

write_die_text

```
write_die_text
[-help]
[-coord {center | origin}]
-device_path device_path
-file file
[-format {default | io_spec | virtual_pkg | flat}]
[-pin_types pin_types]
[-process_shrink scale]
```

Exports the Die information for the specified device to a Die text file.

Parameters

`-help`	Prints a brief description for each `write_die_text` parameter. *Data_type*: optional
`-coord {center | origin}`	Specifies the coordinates. *Data_type*: enum, optional
`-device_path` *`device_path`*	Specifies the device path for which the Die information is to be exported. *Data_type*: required
`-file` *`file`*	Specifies the Die text output file. *Data_type*: file, required
`-format {default | io_spec | virtual_pkg | flat}`	
	Specifies the output format. *Default*: `default` *Data_type*: enum, optional
`-pin_types` *`pin_types`*	Specifies the pin types to export. *Data_type*: optional
`-process_shrink` *`scale`*	Specifies the shrink factor for the device. For example: 0.75 to scale to 1/0.75 * 100% *Data_type*: optional

Example

- The following command exports from the device path `BGA/DIE2` to the specified text file:

```
write_die_text -device_path BGA/DIE2 -file $outFile -process_shrink 1.0
```

Related Commands

- `read_bga_text`
- `read_die_text`
- `write_bga_text`

write_edtext_file

```
write_edtext_file
[-help]
-file file
[-include_top_pins]
-layers layers
-top_cell top_cell
```

Exports information about all external terminals to a text file with EDTEXT format.

Parameters

`-help`	Prints a brief description for each `write_edtext_file` parameter. *Data_type*: optional
`-file file`	Specifies the path of the file to be written. *Data_type*: file, required
`-include_top_pins`	Exports all pins under the top cell, regardless of their type. For example, all UNKNOWN-type pins under the top cell will also be exported when this argument is set. *Data_type*: optional
`-layers layers`	Specifies the layers to be exported. The format will be `{layerName layerName ...}`. *Data_type*: required
`-top_cell top_cell`	Specifies the top cell. *Data_type*: cell, required

Example

- The following command exports external terminals on the Metal 10 to 12 layers for the `DIE` top cell to the specified text file:

```
write_edtext_file -top_cell DIE -file ./edtext.txt -layers {Metal0 Metal1 Metal2} -include_top_pins
```

write_oio

```
write_oio
[-help]
file
```

Writes the specified `.oio` file.

Parameters

`-help`	Prints a brief description for each `write_oio` parameter. *Data_type*: optional
file	Specifies the location of the `.oio` file to be written. *Data_type*: required

Related Command

- `read_oio`

write_pin_label

```
write_pin_label
[-help]
file
-cell cell
```

Writes the pin label to the specified file.

Parameters

`-help`	Prints a brief description for each `write_pin_label` parameter. *Data_type*: optional
file	Specifies the path of the file to be written. *Data_type*: file, required
`-cell` *cell*	Specifies the cell. *Data_type*: required

Example

- The following command writes the pin labels to the `TC2_noR_test.oio` file:

```
write_pin_label TC2_noR_test.oio -cell TC2_NOR_TEST_1:RF_BGA ;
```

Related Command

- `read_pin_label`

write_pin_physical_list

```
write_pin_physical_list
[-help]
csv_file
[-coord {absolute | relative}]
-device_path path
[-origin {origin | center}]
```

Writes the physical pin list to the specified file.

Parameters

`-help`	Prints a brief description for each `write_pin_physical_list` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be written. *Data_type*: file, required
`-coord {absolute | relative}`	
	Specifies the coordinates. Default is relative to the parent device. *Default*: `relative` *Data_type*: enum, optional
`-device_path` *`path`*	Specifies the device path. *Data_type*: required
`-origin {origin | center}`	
	Specifies the placement origin. *Default*: `center` *Data_type*: enum, optional

Example

- The following command writes the pin list information to the `out.csv` file:

  ```
  write_physical_pin_list out.csv -device_path MyChip
  ```

- The following command writes the pin list information to the `out.csv` file with the placement origin as `center`:

  ```
  write_physical_pin_list out.csv -device_path MyChip -coord relative -origin center
  ```

Related Command

- `read_pin_physical_list`

write_ploc_file

```
write_ploc_file
[-help]
dir_path
[-vsrc_layer layer]
```

Writes the power pad location (PLOC) files to the specified directory.

Parameters

`-help`	Prints a brief description for each `write_ploc_file` parameter. *Data_type*: optional
`dir_path`	Specifies the directory to which to write the PLOC files. *Data_type*: file, required
`-vsrc_layer` *`layer`*	Defines the non-contacted bumps on the specified layers as the voltage source. *Data_type*: list of layer, optional

Example

- The following command writes the PLOC files to the `PLOC` directory:

```
write_ploc_file PLOC -vsrc_layer INTERPOSER:C4
```

write_power_intent

```
write_power_intent
[-help]
file_name
-1801
```

Writes out a power intent file using the the IEEE1801 standard.

Parameters

`-help`	Prints a brief description for each `write_power_intent` parameter. *Data_type*: optional
`file_name`	Specifies the name of the power intent file output directory. *Data_type*: file, required
`-1801`	Specifies that the IEEE 1801 power format will be used. *Data_type*: required

Example

- The following command writes out a power intent file, `upf_out`, using the 1801 format:

```
write_power_intent upf_out -1801
```

Related Information

- `read_power_intent`

write_term_list

```
write_term_list
[-help]
csv_file
-module module
```

Writes the specified term list file.

Parameters

`-help`	Prints a brief description for each `write_term_list` parameter. *Data_type*: optional
`csv_file`	Specifies the CSV file to be written. *Data_type*: file, required
`-module` *`module`*	Specifies the module for which the CSV is to be written. *Data_type*: module, required

Example

- The following command writes the specified term list file for the `RF_BGA` module:

```
write_term_list {C:\Users\morrisy\Desktop\RegressionTest\17.2-Dev\TermList\Export\BGA_TermList.csv} \
                                -module RF_BGA
```

Related Command

- `read_term_list`
- `read_term_map`

write_term_map

```
write_term_map
[-help]
csv_file
[-cell cell]
[-module module]
```

Writes the term map for the specified cell or module to the specified file.

Parameters

`-help`	Prints a brief description for each `write_term_map` parameter. *Data_type*: optional
csv_file	Specifies the path for the CSV file to be written. *Data_type*: file, required
`-cell` *cell*	Specifies the cell for which the CSV is to be written. *Data_type*: cell, optional
`-module` *module*	Specifies the module for which the CSV is to be written. *Data_type*: module, optional

Example

- The following command writes the term map file for the `design` cell to the `termMap.csv` file:

  ```
  write_term_map termMap.csv -cell design
  ```

Related Command

- `read_term_list`
- `read_term_map`

write_to_gif

```
write_to_gif
[-help]
[-area area]
[-height height]
-out_file name
[-width width]
```

Saves a snapshot of the current screen to a GIF file with the specified name in the current directory. You can use this command at any stage of the design flow.

Parameters

`-help`	Prints a brief description for each `write_to_gif` parameter. *Data_type*: optional
`-area` *`area`*	Specifies the design region to be captured in the snapshot. *Data_type*: optional
`-height` *`height`*	Specifies the height of the image. *Data_type*: optional
`-out_file` *`name`*	Specifies the name of the GIF snapshot file. *Data_type*: file, required
`-width` *`width`*	Specifies the width of the image. *Data_type*: optional

Example

- The following command stores the snapshot of the current canvas as a GIF format file named `canvas_snapshot`:

  ```
  write_to_gif -out_file canvas_snapshot
  ```

- The following command stores the snapshot of the specified area as a GIF format file named `demo`:

  ```
  write_to_gif -out_file demo -area {724.302171 359.014053 1493.905307 1136.125512}
  ```

- The following command stores the snapshot of the specified area as a GIF file named `demo` with the specified dimensions:

  ```
  write_to_gif -out_file demo -area {724.302171 359.014053 1493.905307 1136.125512} -width 300 -height 300
  ```

Related Commands

- `gui_fit`
- `gui_set_visible_layer`
- `gui_set_visible_substrate`
- `gui_zoom`
- `refresh_canvas`

write_verilog

```
write_verilog
[-help]
verilog_file
[-include_leaf_cell]
[-include_pg]
[-phys]
[-top_module top_module | -module module_name]
```

Writes the specified Verilog file.

Parameters

`-help`	Prints a brief description for each `write_verilog` parameter. *Data_type*: optional
`verilog_file`	Specifies the Verilog output file. *Data_type*: file, required
`-include_leaf_cell`	Includes leaf module definitions in the netlist. *Data_type*: optional
`-include_pg`	Includes PG connections in the netlist. *Data_type*: optional
`-module` *`module_name`*	
	Saves the specified module of the netlist. *Data_type*: module, optional
`-phys`	Writes out the physical cell instances, and inserts power and ground nets in the netlist. *Data_type*: optional
`-top_module` *`top_module`*	
	Saves the specified module along with its submodules. *Data_type*: module, optional

Example

- The following command writes the specified Verilog file:
  ```
  write_verilog mother_die.v -top_module mother_die -include_leaf_cell -include_pg
  ```

Related Commands

- `read_verilog`

8

Implementation Design Navigation and Management

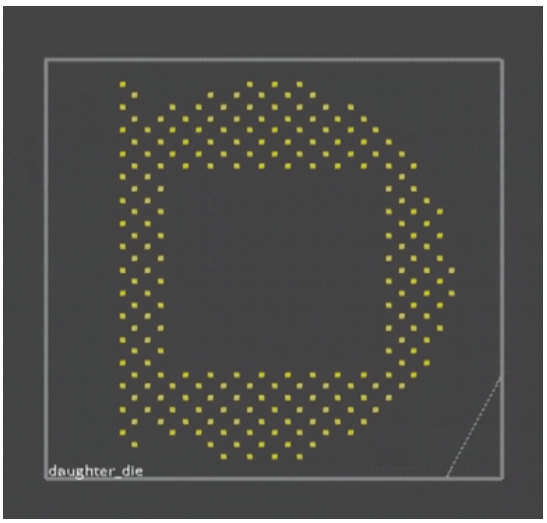
- Using the Multi-Die Physical Layout View
- Using the Color Panel in the Multi-Die Mode
- Switching between an Individual Die and 3D-IC Top

Using the Multi-Die Physical Layout View

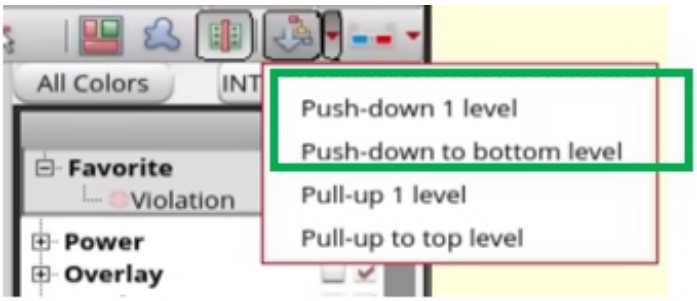
When the layout view is in the Multi-Die mode, you can view both the die instances and each die's physical layout in the layout GUI window.

To expand or collapse each die's physical layout display, use the hierarchical cell*Push-down* and *Pull-up* buttons on the toolbar.

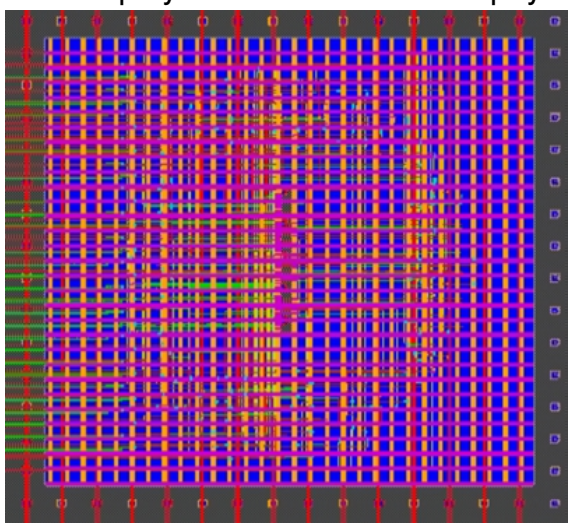
Assume that only die instance is displayed, as shown below.



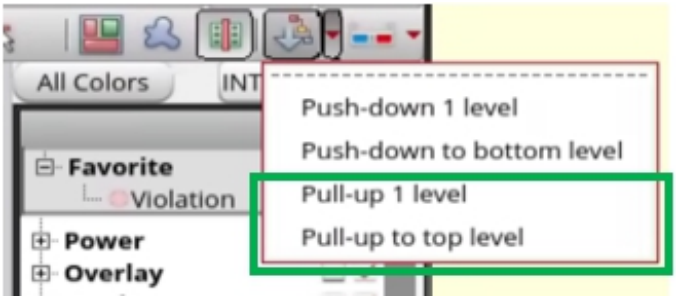
1. To expand the display, select the*Push-down 1 level* or *Push-down to bottom level* option from the pulldown menu. If the die does not have additional internal hierarchy, both these options work in the same way.



2. This displays both the die and its physical layout:



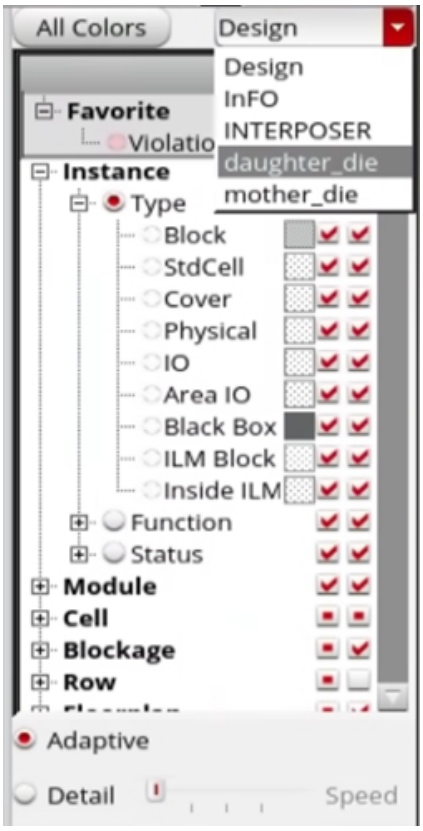
3. To collapse each die's physical layout and only display die instance, select the *Pull-up 1 level* or *Pull-up to top level* option from the pulldown menu.



Using the Color Panel in the Multi-Die Mode

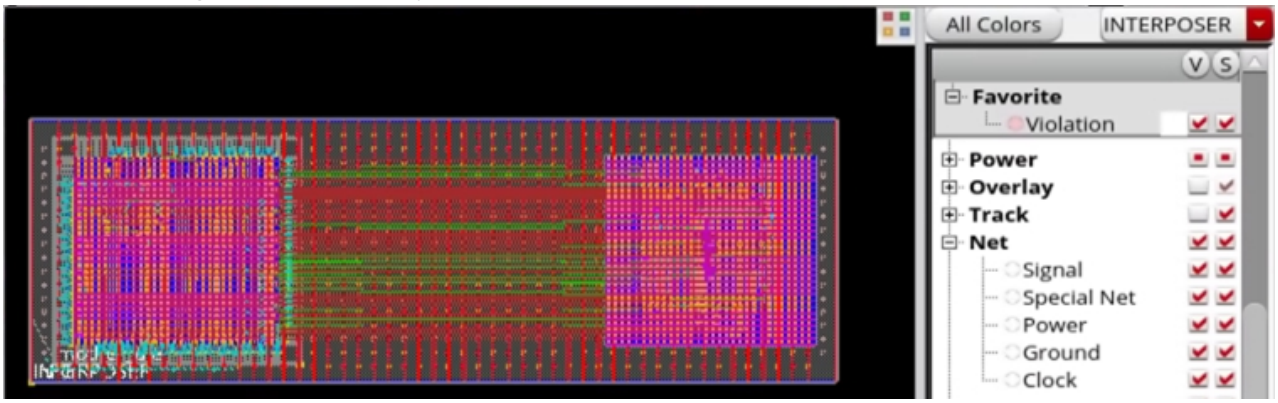
In the Multi-Die mode, use the die name pulldown list on the color panel to switch the *Visibility* and *Selectability* control among the multiple dies.

For example, to turn on or off the object visibility of a die named *daughter_die*, select it from the pulldown list and then click the visibility check boxes of the required objects.

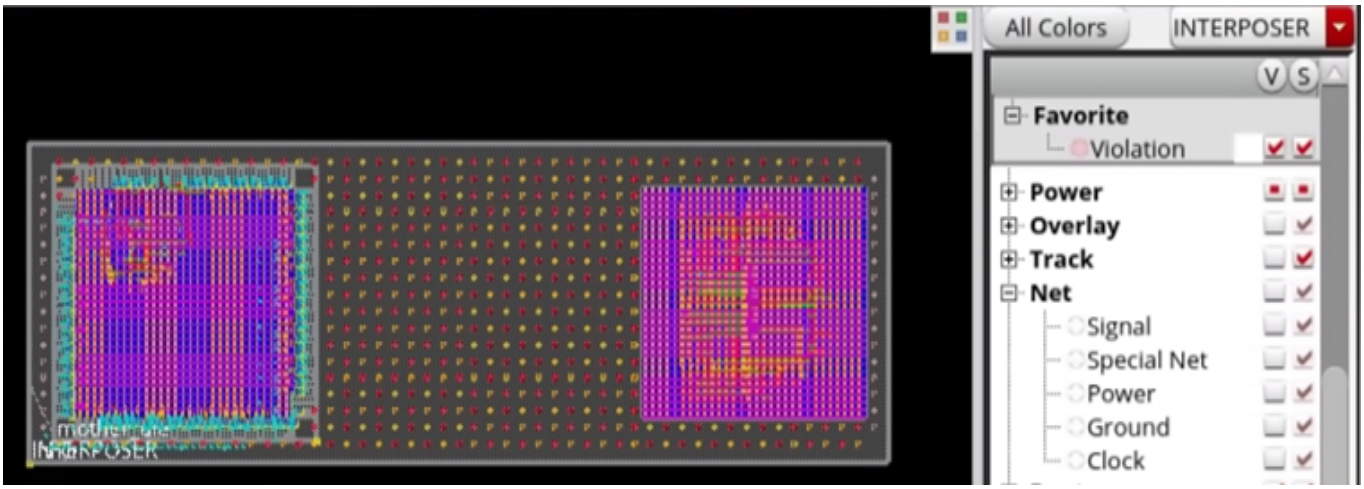


In the following example, the visibility of nets is turned off for the*INTERPOSER* die, while it is kept on for *mother_die* and *daughter_die*:

- Before turning off net visibility for *INTERPOSER*



- After turning off net visibility for *INTERPOSER*



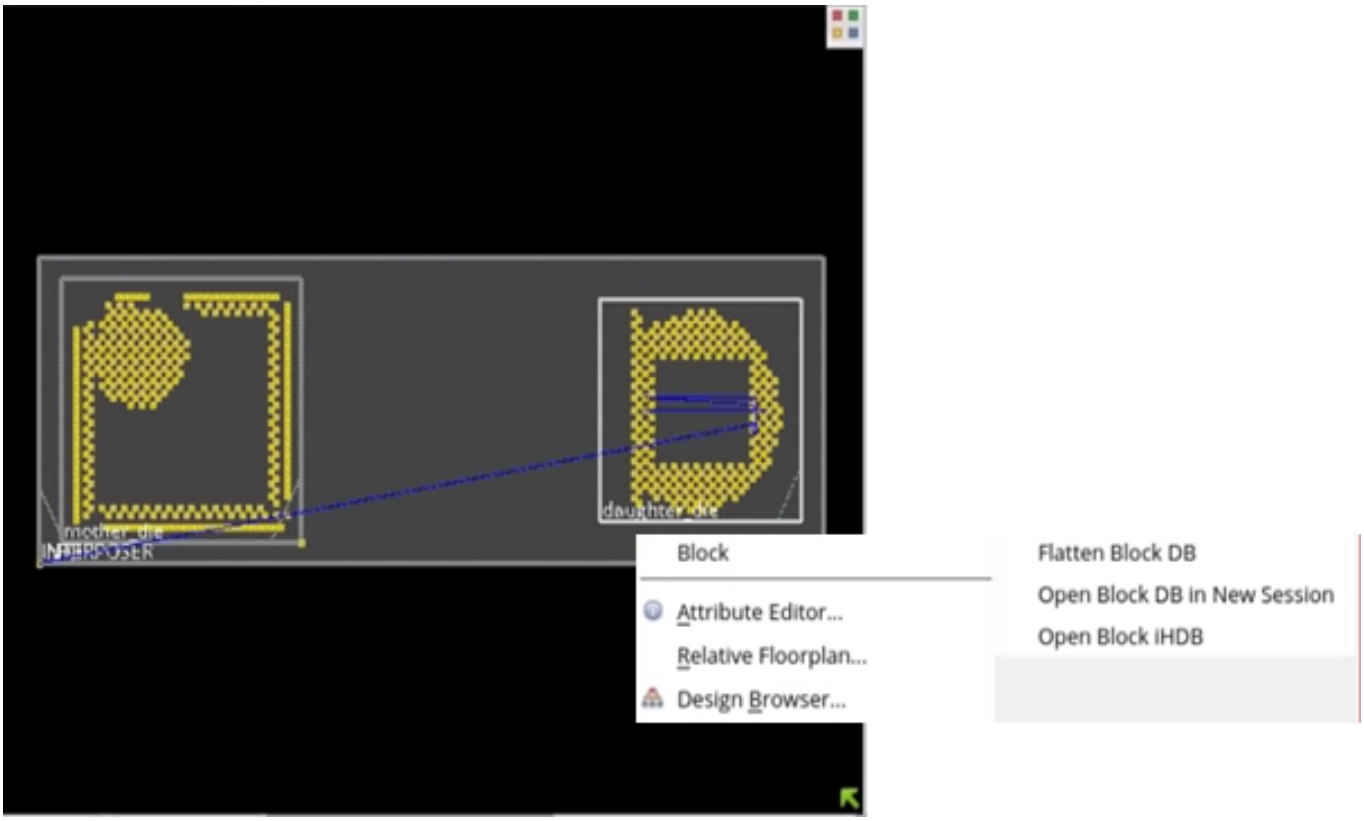
Switching between an Individual Die and 3D-IC Top

In the layout view's Multi-Die mode, you can view the 3D-IC from the top level or switch into an individual die and make it the current top cell.

Example

To switch to *daughter_die*,

1. Select the *daughter_die* instance in the 3D-IC top-level view.
2. Right click the top to open the context menu.
3. Select *Block – Open Block iHDB*.



Then the current top cell is set to*daughter_die*. The layout window will display the *daughter_die* layout in its original orientation R0 and face-up.

In the 3D-IC top-level view, when you expand the die's physical layout, the layout is displayed according to the die's location and orientation in the 3D-IC stacking. If it is face-down and rotated, Integrity 3D-IC will do the transform automatically.

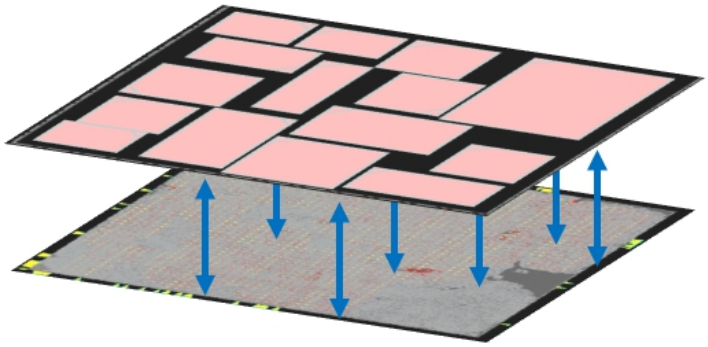
9

Memory-on-Logic Flow

- Overview of 3D-IC Memory-on-Logic Flow
-
- Recommended 3D-IC Memory-on-Logic Flow
  - Concurrent Multi-Die Placement
  - Creating Netlist Hierarchy for Memory and Logic Dies
  - Assigning Shadow Bumps
  - Inserting a Zero Delay Cell
  - Physically Unfolding the Design
  - Creating Bumps on Each Die

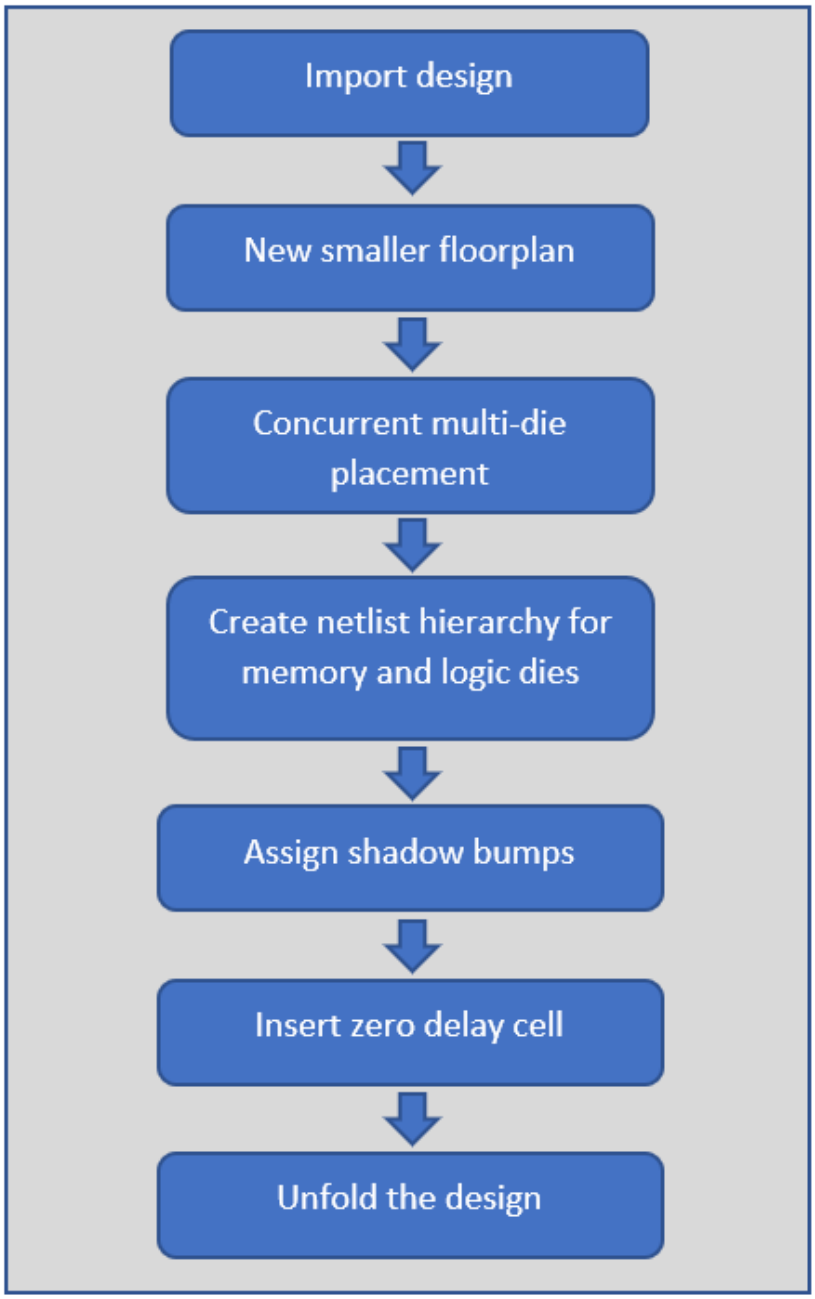
Overview of 3D-IC Memory-on-Logic Flow

The 3D-IC solution offers various benefits on optimizing the power, performance, and area (PPA) for latency, bandwidth, and power. Due to the well-known "memory wall" limitations of Moore's law, where the size and speed of on-chip memories are not able to catch up with the rate of increase of the number of transistors in a processor design, a memory die mounted on the top of a logic die is a popular 3D stacking style for improving memory latency. Integrity 3D-IC leverages a unique mixed placement technology from Innovus Implementation System to separate the memory macros from a 2D design and automatically partition and implement it into two homogenous tiers: a memory die on the top tier and a logic die with standard cell logic on the bottom tier. As some logic, such as test logic, may need to reside on the same die as the memories, the flow also supports user control by selectively assigning macros and logic partition to different tiers. The flow relies on 3D mixed placement, shadow bump assignment, and a physical unfolding of the design to achieve a fully placed, timing-aware stacked design.



Recommended 3D-IC Memory-on-Logic Flow

The diagram below shows the main steps of the 3D-IC Memory-on-Logic flow. After the*Unfold the design* step is complete, you can continue to run standard implementation steps like CTS, optimization, routing, and post-route steps as normal place-and-route flows to finish 3D implementation.



Concurrent Multi-Die Placement

The Mixed Placer is a powerful engine for placing macros and standard cells concurrently. It has been enhanced to carry out mixed placement for two dies simultaneously in three-dimensional space, driven by wire-length and timing.

For the steps for extracting the power routing density modeling information, floorplan cleanup, and setting up the constraints for macro placement, you can refer to the "Using the Mixed Placer" chapter in the *Innovus User Guide*.

To enable 3D Mixed Placer for global and detail placement in 3D space, you can use the `setPlaceMode` command. You can also use `setPlaceMode` to specify the number of dies and set the constraint for concurrent multi-die placement.

You can use `proto3d::set_obj_die_id` to specify the die id for each object. If the die id of the instance is `1`, the tool will place it on the bottom die; and if the die id of the instance is `2`, the tool will place it on the top die. Then, you can use the `place_design -concurrent_macros` command for concurrent timing-driven placement of macros and standard cells in the 3D space.
Use `refine_macro_place` to legalize the macros to honor the constraints and rules.

After the location of the macros are fixed in the floorplan, set the value of `setPlaceMode -place_opt_run_global_place` to `seed`. With the `seed` setting, you can continue to do seed placement from concurrent macro placement and do incremental standard cell placement.

Note: 3D concurrent multi-die placement support is a beta feature in this release. It is enabled by a variable specified using the `setBetaFeature` command. To use this feature, contact your Cadence representative to discuss your usage requirements. Make sure this feature meets your needs before deploying it widely.

Creating Netlist Hierarchy for Memory and Logic Dies

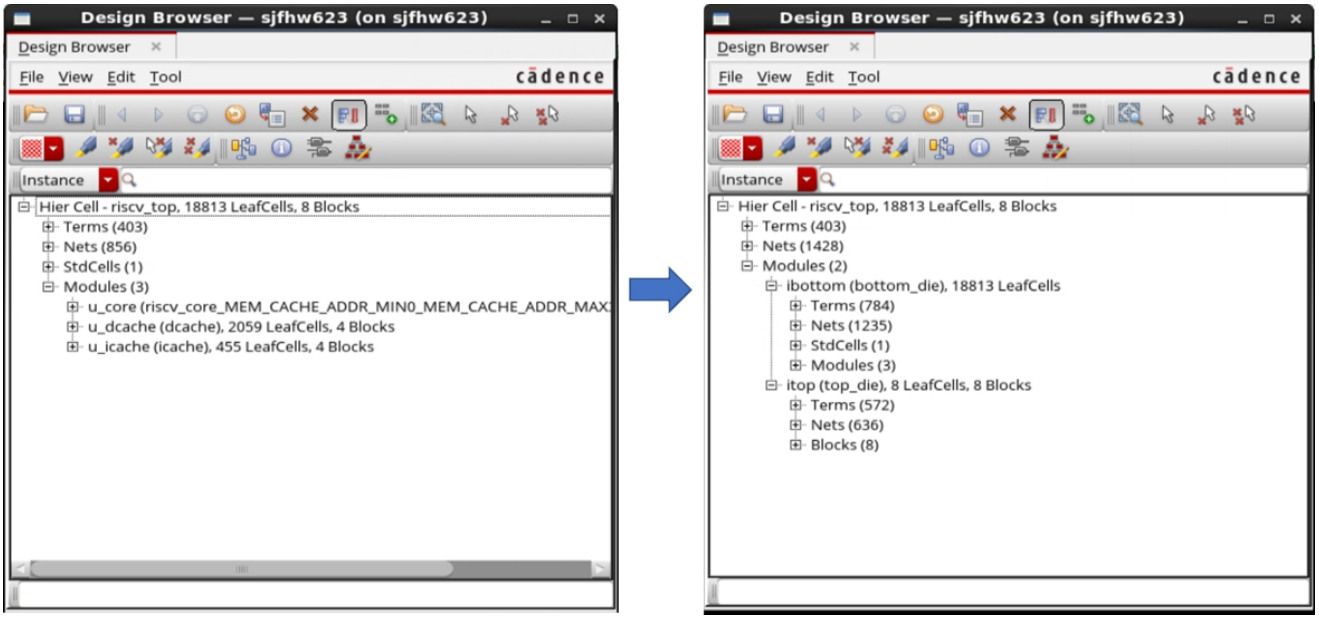
This step does the following three things:

- Reorganizes the original netlist by creating two new top-level hinsts, which represent the top and bottom dies of the design.
  You can use the `proto3d::create_module_wrapper` command to create the new netlist hierarchy. By default, the tool will put all the hard macros into the top die and put all the standard cells into the bottom die. All the primary I/Os connect to the bottom die.
- Creates feedthrough nets and ports.
  This is needed because any net connecting the top-level I/Os to the top die must cross the bottom die. The net will become a "feedthrough" net in the bottom die.

- Clones multiple fanout ports on the top die.
  If a net drives multiple macros and you need to do the DRV fixing in the bottom die, you need to make sure all the macro die input ports have only one fanout in the macro die.

For example, the following command will create an hinst `itop` with the cell name `top_die` to include all the macros and an hinst `ibottom` with the cell name `bottom_die` to include all the standard cells.

```
proto3d::create_module_wrapper -top_die_hinst itop -top_die_cell top_die -
bottom_die_hinst ibottom -bottom_die_cell bottom_die
```



You can use the option `-from_die_id` of `proto3d::create_module_wrapper` to reorganize the logic hierarchy based on the die id assignment. The instance whose die id is `1` will be put on the bottom die and the instance whose die id is `2` will be put on the top die.

Assigning Shadow Bumps

In this step, you can use `proto3d::assign_shadow_bump` to generate a bump location file that will be used to insert a zero delay cell later. The bump assignment will consider the placement of the hinsts, which is specified by `-ref_hinsts`. The clock pin will be considered as a higher priority than other signal pins so that the clock net will have a shorter distance.

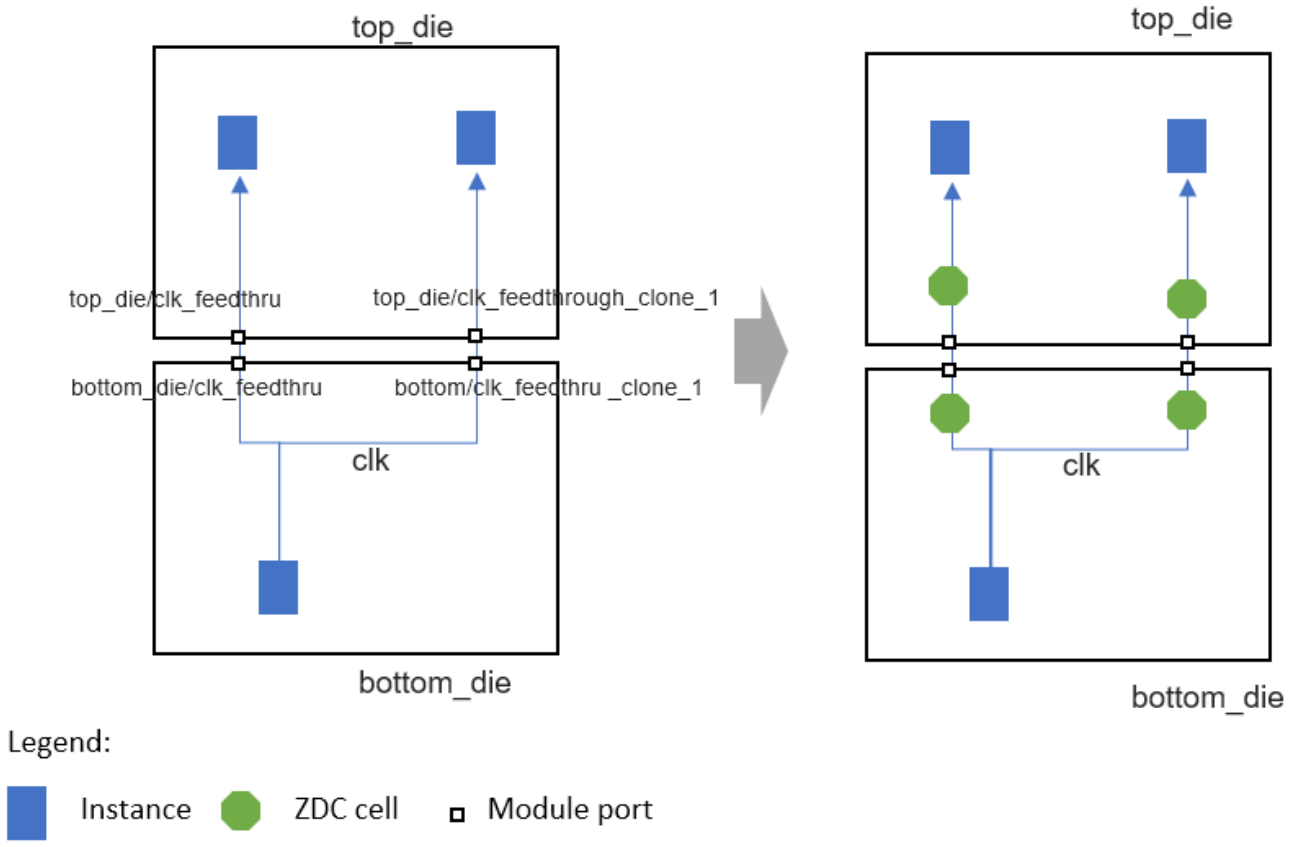
Below is an example of what you can see inside the bump location file. `{40.0 202.0}` is the location of the bump and `u_icache_data0_data_out_wl_21_r` is the pin of top die.

```
Bump: bumpName_0 BUMPCELL 40.000000 202.000000 u_icache_data0_data_out_wl_21_r
```

To reserve space for PG bumps, you can use `create_bump` to create PG bumps in the floorplan in advance, and the tool will use the unoccupied space in the floorplan for signal bump assignment.

Inserting a Zero Delay Cell

Based on the location given by the bump location file generated in the last step, `proto3d::insert_zero_delay_cell` can be used to insert a zero delay cell (ZDC) in both the top and bottom dies for bump modeling. The tool will automatically add the ZDC cells to the current design library. The size of the cell is specified by the `-zdc_size` parameter of the `proto3d::insert_zero_delay_cell` command and you can also provide a modified ZDC's library (.lef, .lib) for the tool. The cross-die connection will be represented as two ZDC cells at the bump location in each die and a dummy wire with zero net delay that connects the ZDC cells. The router will not route the nets that connect to the ZDC cells. At the end, the ZDC cells will be replaced with bumps in each die.



Physically Unfolding the Design

You can use `proto3d::unfold_floorplan` to physically unfold the design. The floorplan's size will be expanded twice vertically, with the top die unfolding to the upper half of the new floorplan, instead of overlapping the bottom die. The tool will mirror the location and flip MX for each individual instance and ZDC accordingly because the top die is facing down.

All the I/Os will stay on the bottom die. To avoid the placement and routing out of each die, a placement and route blockage band will be created between the two top-level hinsts. You can use the `-gap` parameter of the `proto3d::unfold_floorplan` command to specify the width of the band. The site rows within the top die area will be cut to prevent buffering in the top die.

After the design is unfolded, you can insert physical cells and create power stripes. For more information, see `addEndCap`, `addWellTap`, `addPowerSwitch`, `addStripe`, `editPowerVia`, and so on in the *Innovus Text Command Reference*. After the above steps are complete, you can continue to run standard implementation steps like CTS, optimization, routing, and post-route steps as normal place-and-route flows to finish 3D implementation.

Then, you can run timing budget and partition, and save the partitions to get separate design data for each die.

Creating Bumps on Each Die

In each implemented die, ZDC will be replaced with bumps at the end. To add bumps in each die, you can use the `readBumpLocation` command to read the bump location file and create bumps in the current die. The bump location file was generated in the previous shadow bump assignment step. The `readBumpLocation` command should be run after reading the stacked die config file. You can use `readTSVConfig` to import 3D-IC configuration files of the whole design, including the stack die configuration file and the top-level netlist.

After all the above steps are complete, you can assemble both the dies in the top level and do timing and power analysis.

10

3D Early Floorplan Synthesis

- Overview of 3D Early Floorplan Synthesis
  - Benefits of 3D Early Floorplan Synthesis
- General 3D-EFS Flow
- Using Proto Seeds
  - Defining Proto Seeds
  - Adding Constraints for Proto Seeds
  - Placing the Proto Seeds
  - Generating and Optimizing Partition Shapes
- Partitioning the Design
  - Logical Partitioning
  - Physical Partitioning

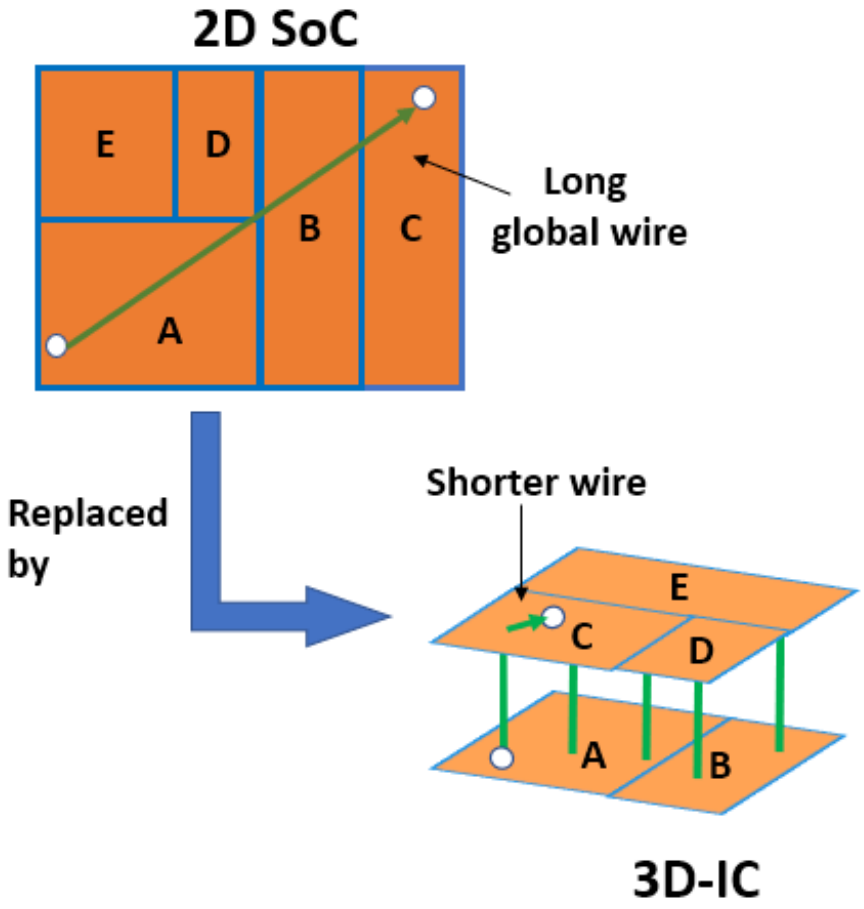
Overview of 3D Early Floorplan Synthesis

The 3D Early Floorplan Synthesis (3D-EFS) methodology is a powerful capability that enables semi-automated rapid floorplan creation, partition shape generation, and exploration for 3D-IC design from an early stage. The tool can:

- Place a module in 3D design space.
- Consider 3D wire length, number of I/O ports, and timing.
- Apply the logic restructuring feature to divide a single netlist into multi dies.

Benefits of 3D Early Floorplan Synthesis

3D-EFS provides shorter wires, a smaller profile, less power, higher bandwidth, better yield, higher performance, and better latency for critical inter-block communication.

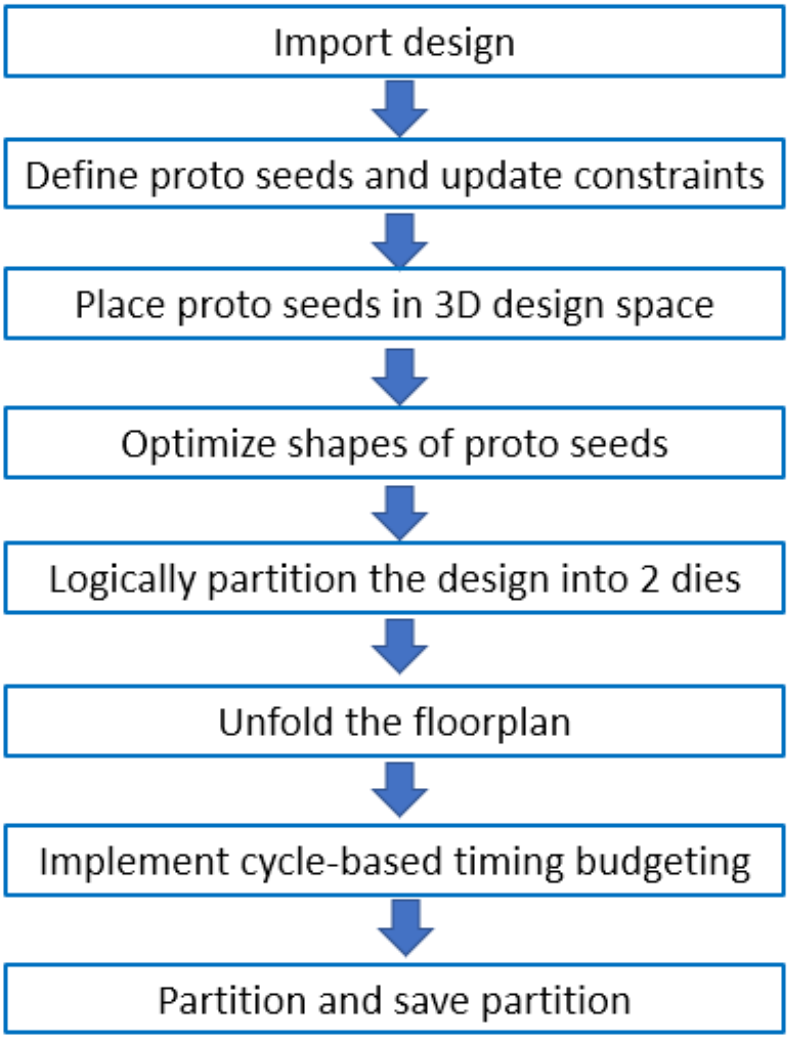


General 3D-EFS Flow

The Integrity 3D-IC release introduces the 3D-EFS flow, which provides better handling of a large-scale floorplan and critical inter-block communication, as compared to the traditional 2D floorplan flow.

The 3D-EFS flow provides you the ability to specify placement seeds easily. The general 3D-EFS flow includes proto seed implementation and design partition.

The 3DIC EFS flow is as follows:



After importing the design, you can define the floorplan size to create a new floorplan for the 3D-IC design. After the new floorplan is generated and the beta feature to enable 3D-EFS is set, you can follow the normal flow.

Using Proto Seeds

A proto seed could be any module (hInst) or instance group that needs to be shaped and placed into the floorplan. The proto seeds could have the constraints for utilization, aspect ratio range, soft fixed, and so on.

Defining Proto Seeds

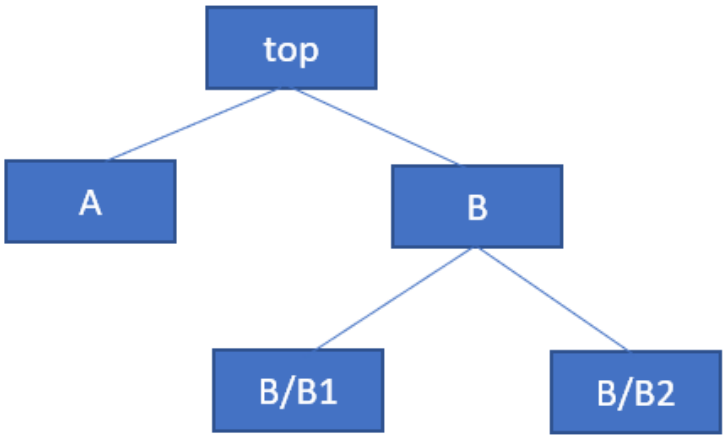
Integrity 3D-IC provides TCL commands for defining and manipulating proto seeds.

First, you can use the `create_proto_seeds` command to specify the placement seeds easily. Integrity 3D-IC supports incremental operation in which you can specify seeds with multiple `create_proto_seeds` commands. For example, the following command defines hInst `module_A` and `module_B` as proto seeds.

```
create_proto_seeds -hinsts {module_A module_B}
```

⚠ The specified seeds should not be nested.

⚠ Proto seeds should cover the whole design. As an example, consider the design hierarchy given below. You could define hInst `A` and `B` as proto seeds, or define `hInst A`, `B/B1`, `B/B2` as proto seeds.



⚠ The recommended practice is to preplace top-level macros in the floorplan.

⚠ The support for defining proto seeds is a beta feature in this release. It is enabled by a variable specified using the `setBetaFeature` command. To use this feature, contact your Cadence representative to explain your usage requirements and make sure this feature meets your needs before deploying it widely.

Adding Constraints for Proto Seeds

Use the `update_proto_seeds` command to specify constraints for proto seeds, including utilization, aspect ratio, soft fixed, and so on. The following command specifies the aspect ratio (width/height) range of proto seeds as `{0.33 3.0}` and the target utilization as `0.65`:

```
update_proto_seeds -aspect_ratio_range {0.33 3.0} -util 0.65
```

> ⚠ If there is some glue logic in the design, you can use the `-group_top_logic` parameter of the `create_proto_seeds` command to group all logic that is not part of the proto seeds into a group and define it as a proto seed. For example, the following command will create an instGroup as a proto seed for the glue logic and set the `util` of the proto seed to `0.5`.
>
> ```
> create_proto_seeds -group_top_logic -util 0.5
> ```

Placing the Proto Seeds

Cleaning up the floorplan

Invoke the `clear_floorplan_for_proto_seeds` command to clean up the floorplan for module placement and shaping. Placement and route blockages will be removed. The tool will keep fixed top-level macros in the floorplan and define them as proto seeds with `soft_fixed` constraints.

Setting the constraint for seed placement

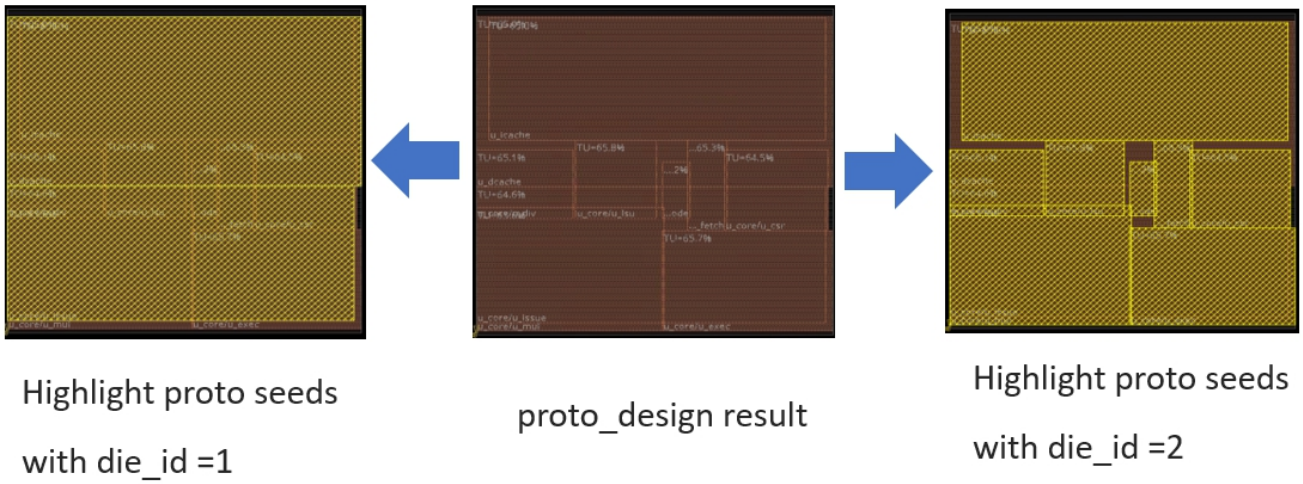
Use the `set_proto_design_mode` command to set the mode options for controlling the `proto_design` command. These include options to specify whether to remove overlaps, place macros, specify the number of dies and the cost for inter-die connection, and so on. Use the `proto3d::set_obj_die_id` command to set the die id for proto seeds, and the tool will honor the pre-defined die-id during seed placement.

Run the `proto_design` command to place the proto seeds for 3D-IC scenarios. `proto_design` will assign a die id for each proto seed, based on seeds placement, and can support pre-defined die ids and pre-fixed objects. Use the `get_proto_seeds` command to check the die id of each proto seed. For example, the following command prints out the proto seeds whose die id is `1`:

```
get_proto_seeds -name -die_id 1
```

You can use the `proto3d::get_obj_die_id` command to get each seed's die id.

The following diagram shows the proto seed placement result.

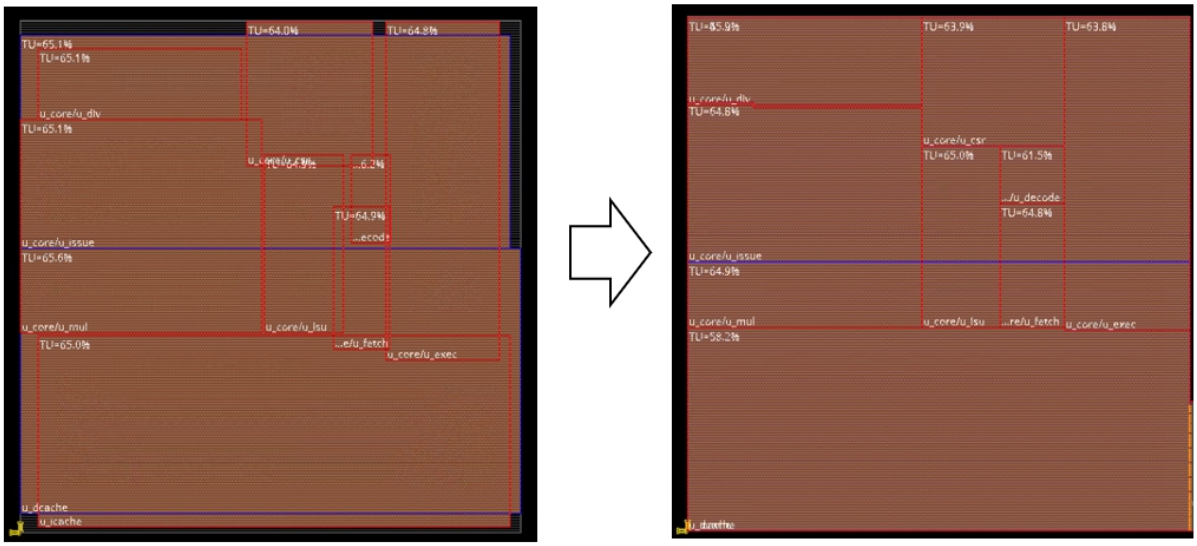


Generating and Optimizing Partition Shapes

After placing the proto seeds, you can use the `proto3d::create_3d_stacked_proto_shapes` command to generate and optimize the partition shapes of the proto seeds per die. Use the `-channel_width` option to specify a uniform channel for the design as shown in the picture below. In addition, you can also specify the maximum iteration and minimum step for proto shape optimization with the `-max_iteration` and `-min_step{x|y}` options.

Seed placement
`proto_design`

Generation/optimization of partition shapes
`proto3d::create_3d_stacked_proto_shape`



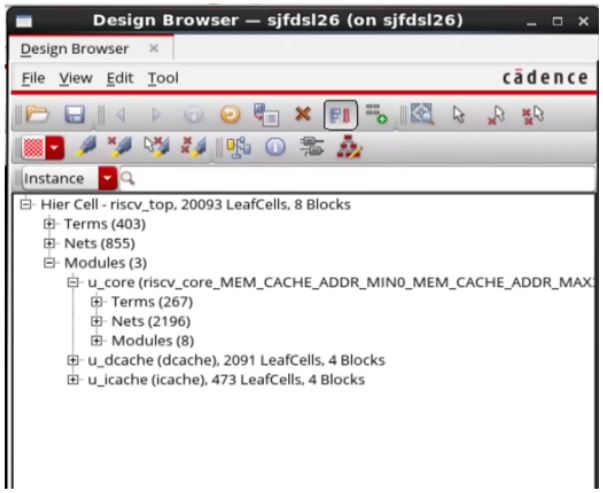
Partitioning the Design

Based on the die id assignment, the design is partitioned into two dies. This comprises two steps – logical partitioning and physical partitioning.

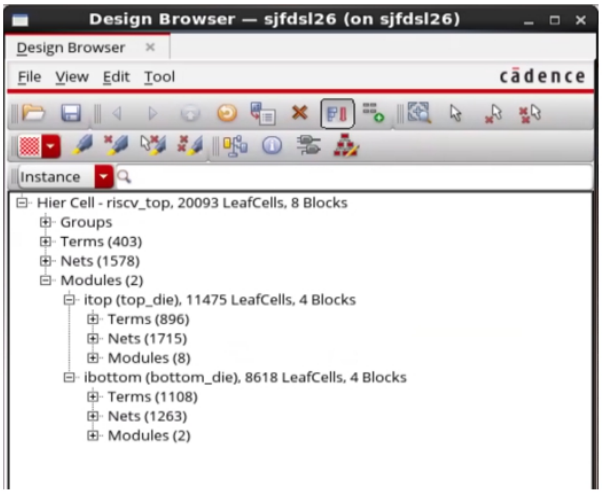
Logical Partitioning

In the first step, the design is logically partitioned into two dies based on the die id assignment. When you specify the the cell name and hinst name for each die with the `proto3d::create_module_wrapper` command, it will reorganize the original netlist by creating two new top-level hinsts that represent the top and bottom dies of the design. The proto seeds whose die id is `1` will be put into `ibottom`, and proto seeds whose die id is`2` will be put into `itop`.

The hierarchical structure of the design before and after running`proto3d::create_module_wrapper` can be seen below:



Before `proto3d::create_module_wrapper`



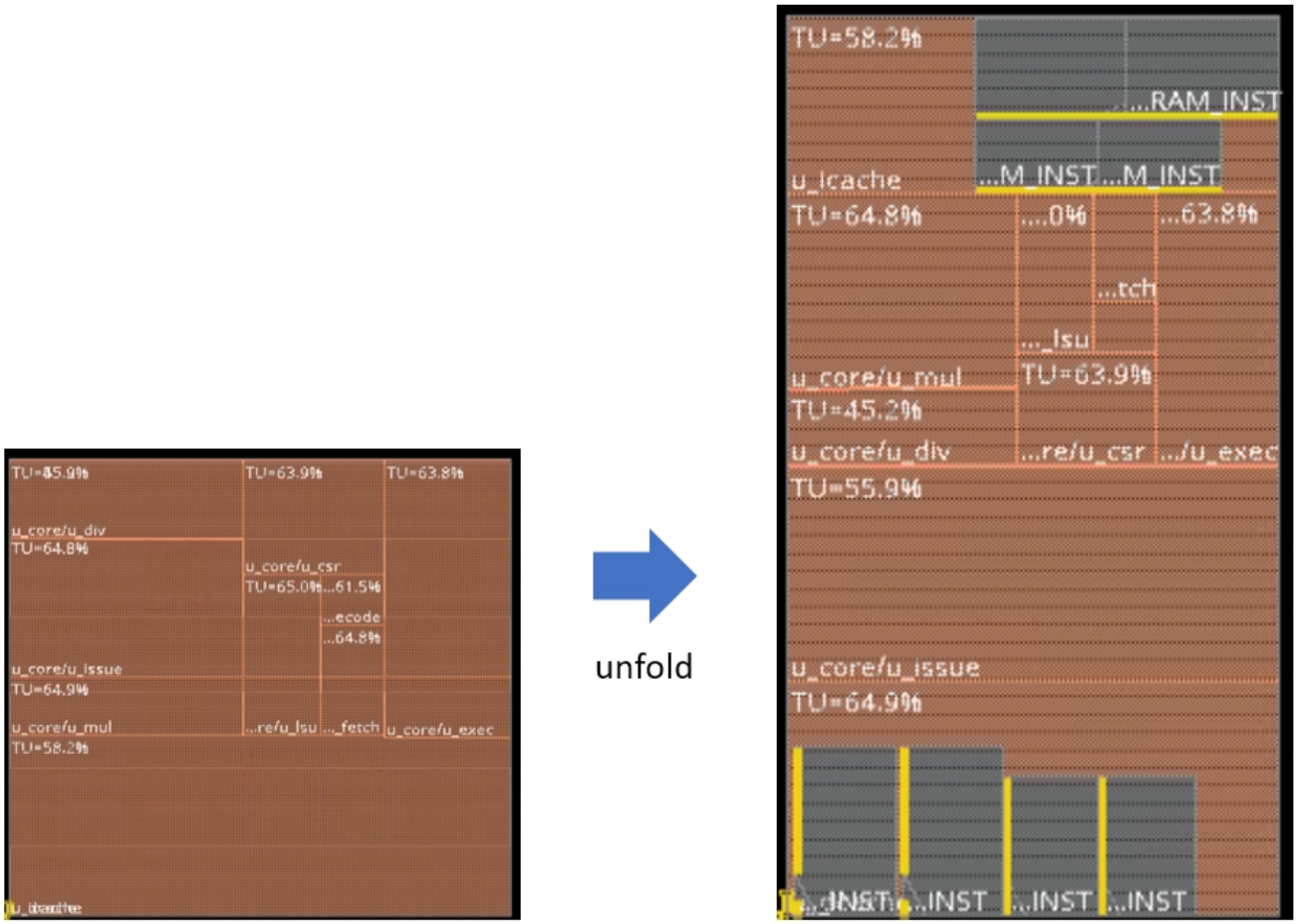
After `proto3d::create_module_wrapper`

Physical Partitioning

In the second step, the design is physically partitioned into two dies based on the die id assignment. You can use the `proto3d::unfold_floorplan` command to specify the flip die name and unfold the top die to the upper half of the new floorplan so that it no longer overlaps the bottom die. The floorplan will be expanded by 2x vertically. The tool will create a fence for the two top-level hinsts and define partitions for them. A placement and route blockage band will be created between the two top-level hinsts. I/Os are on the bottom die. You can specify the die facing type with the `-design_stack_type` option. By default, the die stack type is`face2face`, so the tool will mirror the location and flip MX each individual proto seed accordingly.

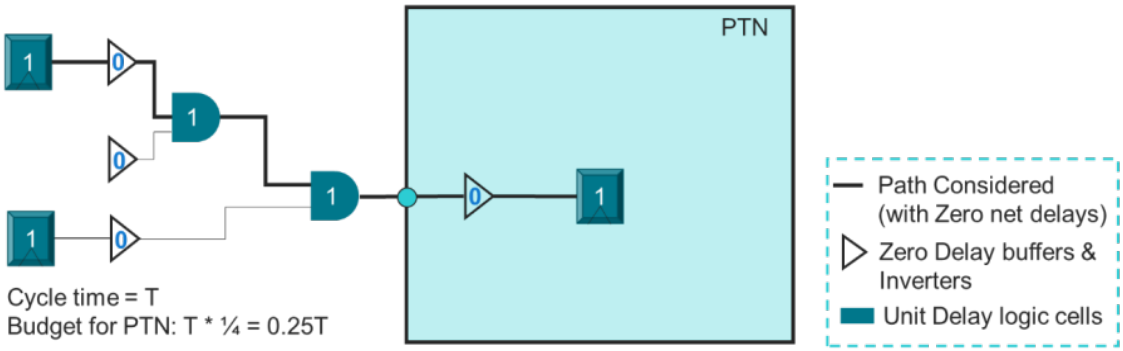
> ⚠ You can specify the width of the placement and route blockage band between the two dies. The default value is 5 or 6 times the `stdcell_row_height`, depending on the `-rowSiteHeight` constraint in `setFPlanMode`.

The unfolded floorplan can be seen in the picture below (with rough macro placement):



Before partitioning, the timing budgeting constraint based on the clock cycle and derived timing budget for each partition block in the design need to be set. For this, use the `setCycleBudgetRatio` command along with the `-cycleRatio` parameter of the `deriveTimingBudget` command.

In addition to cycle-based budgeting, stage-based budgeting is created based on a quick generated Timing Graph that has annotated unit delays for every logic cell/flip-flop, excluding buffers and inverters. This method of timing budgeting can be used for generating quick budgets in the very early stages of a design before any kind of optimization is done. Stage-based budgeting will be more accurate than cycle-based budgeting. Budgeting will take the longest logic path and proportionately divide the clock-cycle time as budgets to the blocks. The following image shows an example of stage-based budgeting. For more information, refer to the "Stage-Based Timing Budgeting" section in the Timing Budgeting chapter of the *Innovus User Guide*.



After partitions and timing constraints are created for dies, you can run the `partition` command to convert partition fences to partition blocks and push down top-level floorplanning data into partition block-level designs for accurate block implementation.

Finally, run `savePartition` to generate the data needed for top- and block-level implementation.

For more information on the EFS-related commands, see the Integrity-Specific Tcl Commands chapter.

11

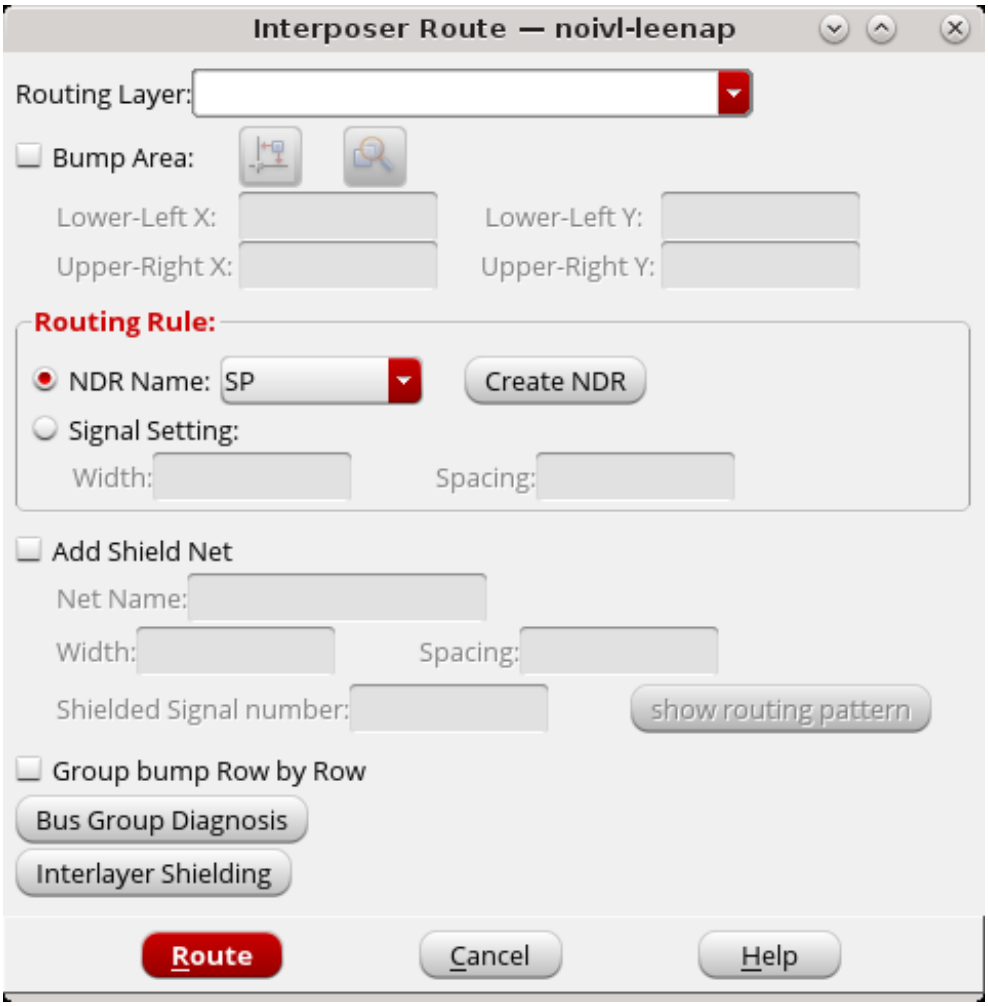
Using the Interposer Route Form

- Interposer Route
- Routing Pattern
- Bus Group Diagnosis
- Interlayer Shielding

Interposer Route

Use the Interposer Route form to perform interposer routing. To open the form:

- Choose *Tools* - *Interposer Route* in the Implementation window.



Interposer Route - Basic Fields and Options

Routing Layer	Specifies the routing layers for interposer routing. You can select one or more routing layers from the drop down list. To choose all layers, select the *All* check box. *Default*: NULL

Bump Area	Specifies the bump selection area. You can specify the area by using one of the following methods: • Click the button and draw a rectangle in the layout to specify the bump area. The software displays the upper and lower X, Y coordinates of the rectangular area in the fields below. • Click the button to retrieve the coordinates of the current viewing area in the layout. • Enter the area coordinates manually in the *Lower-Left X*, *Lower-Left Y*, *Upper-Right X*, and *Upper-Right Y* fields. *Default*: Off
Routing Rule	Specifies the routing rule to be used. You can selectan existing non-default rule (NDR) from the *NDR Name* drop-down list. Alternatively, click the *Create NDR* button to open the Create Non Default Rules form and define a new NDR. For details, see *Create Non Default Rule*. Once the new NDR is created and theCreate Non Default Rules form is closed, the software adds the new NDR to the current *NDR Name* list. Select the *Signal Setting* option to specify the width and spacing of the signal in the respective fields.

Add Shield Net	Specifies that a shield net is to be added.
	Default: Off
	Selecting the *Add Shield Net* check box enables the following fields:
	• *Net Name*: Specifies the name of the shield net.
	• *Width*: Specifies the width of the shield.
	• *Spacing*: Specifies the spacing of the shield.
	• *Shielded Signal Number*: Specifies the number of signals between shielding. If you specify:
	  ◦ 1 - Sets the shield mode to GSG
	  ◦ 2 - Sets the shield mode to GSSG
	  ◦ 3 or higher number - Sets the shield mode to GxSG
	• *show routing pattern*: Opens the Routing Pattern form, which displays the routing pattern based on the signal number specified in the *Shielded Signal Number* field. See Routing Pattern.

Group bump Row by Row	Groups bumps row by row.
Bus Group Diagnosis	Opens the Bus Group Diagnosis form. SeeBus Group Diagnosis.
Interlayer Shielding	Opens the Interlayer Shielding form. See Interlayer Shielding.
Route	Enables interposer routing to connect the macro bumps with bus routing patterns. This option is equivalent to the `routeDesign -bump` command.

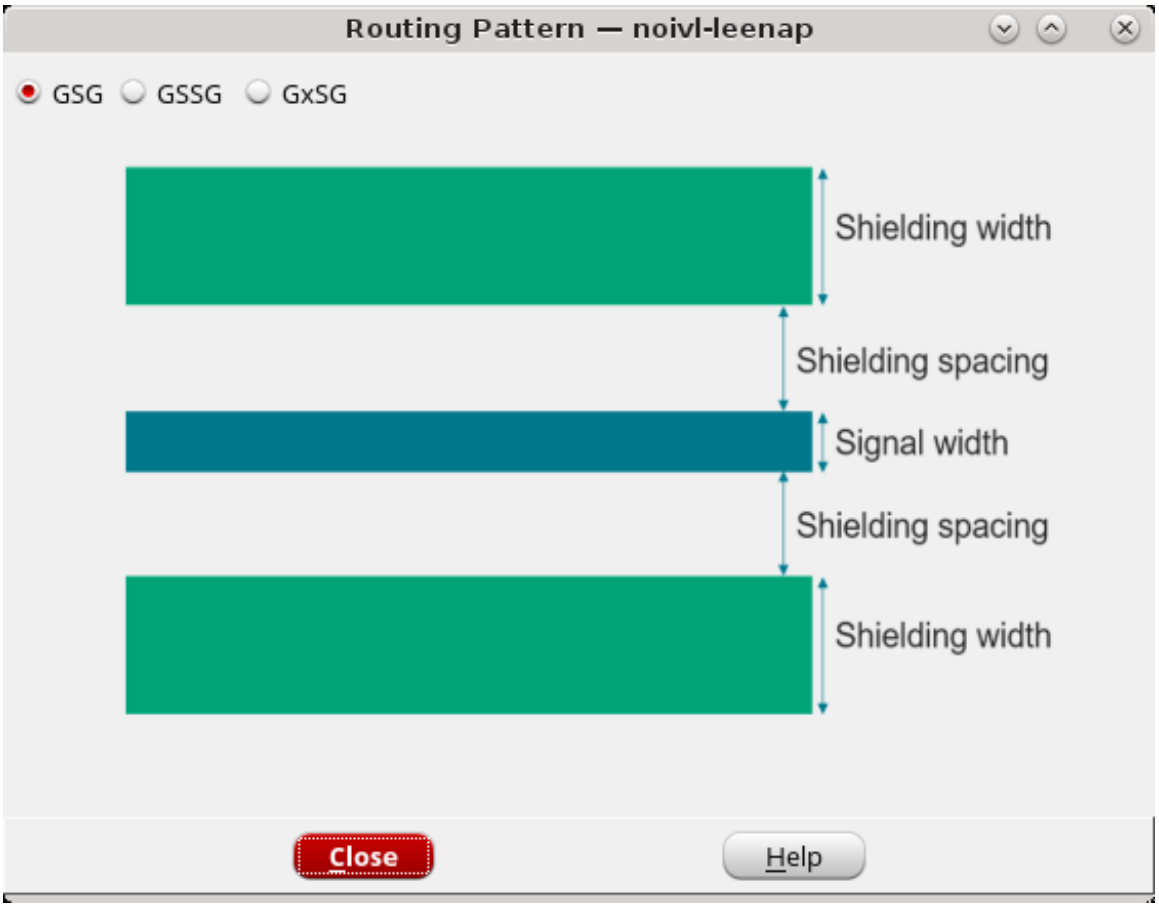
Related Commands

- `routeDesign`
- `setNanoRouteMode`

Routing Pattern

Use the Routing Pattern form to view the routing pattern based on the specified signal number. To open the form:

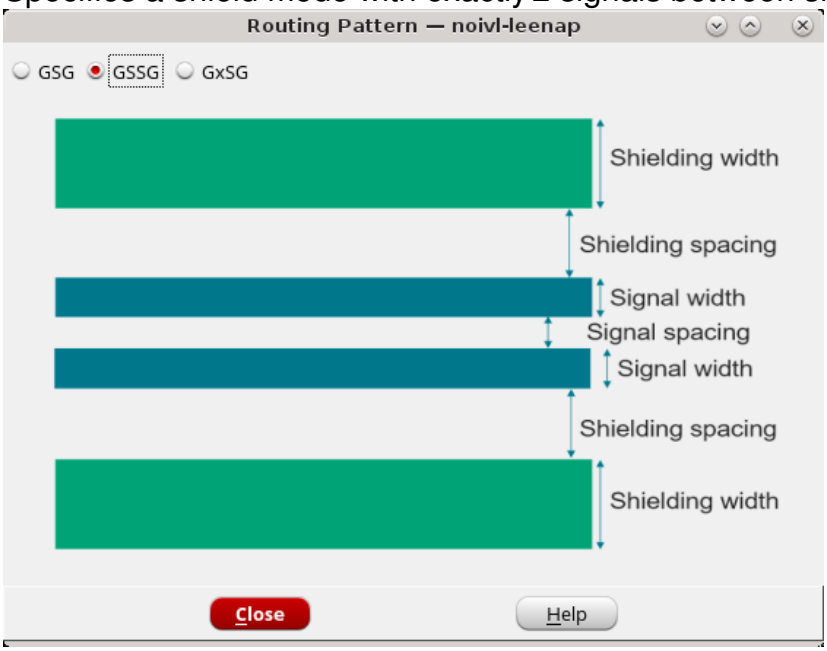
- Click the *show routing pattern* button in the Interposer Route form.

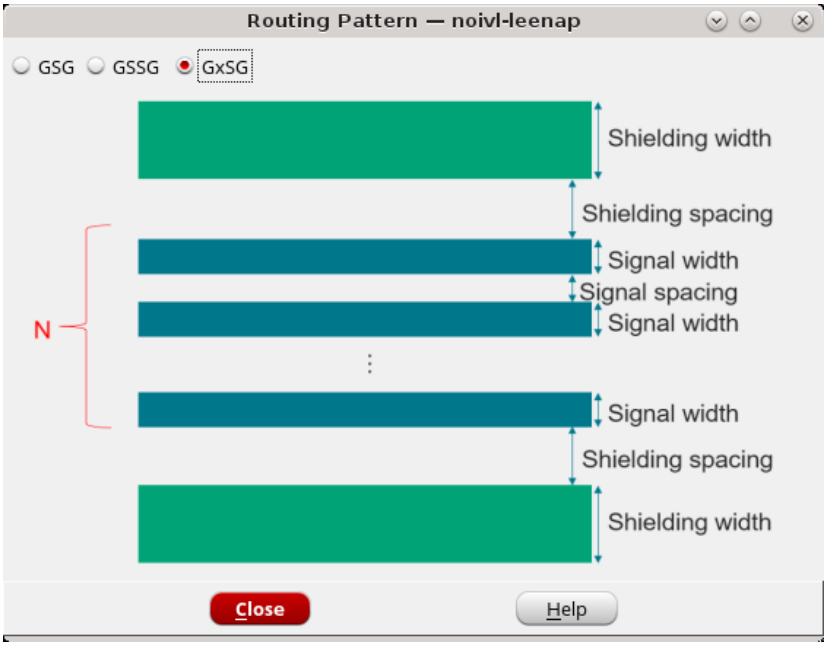


The routing pattern displayed depends on the number you specified in the *Shielded Signal Number* field in the Interposer Route form. If you specify:

- 1 - Sets the shield mode to GSG.
- 2 - Sets the shield mode to GSSG
- 3 or higher number - Sets the shield mode to GxSG

Routing Pattern - Basic Fields and Options

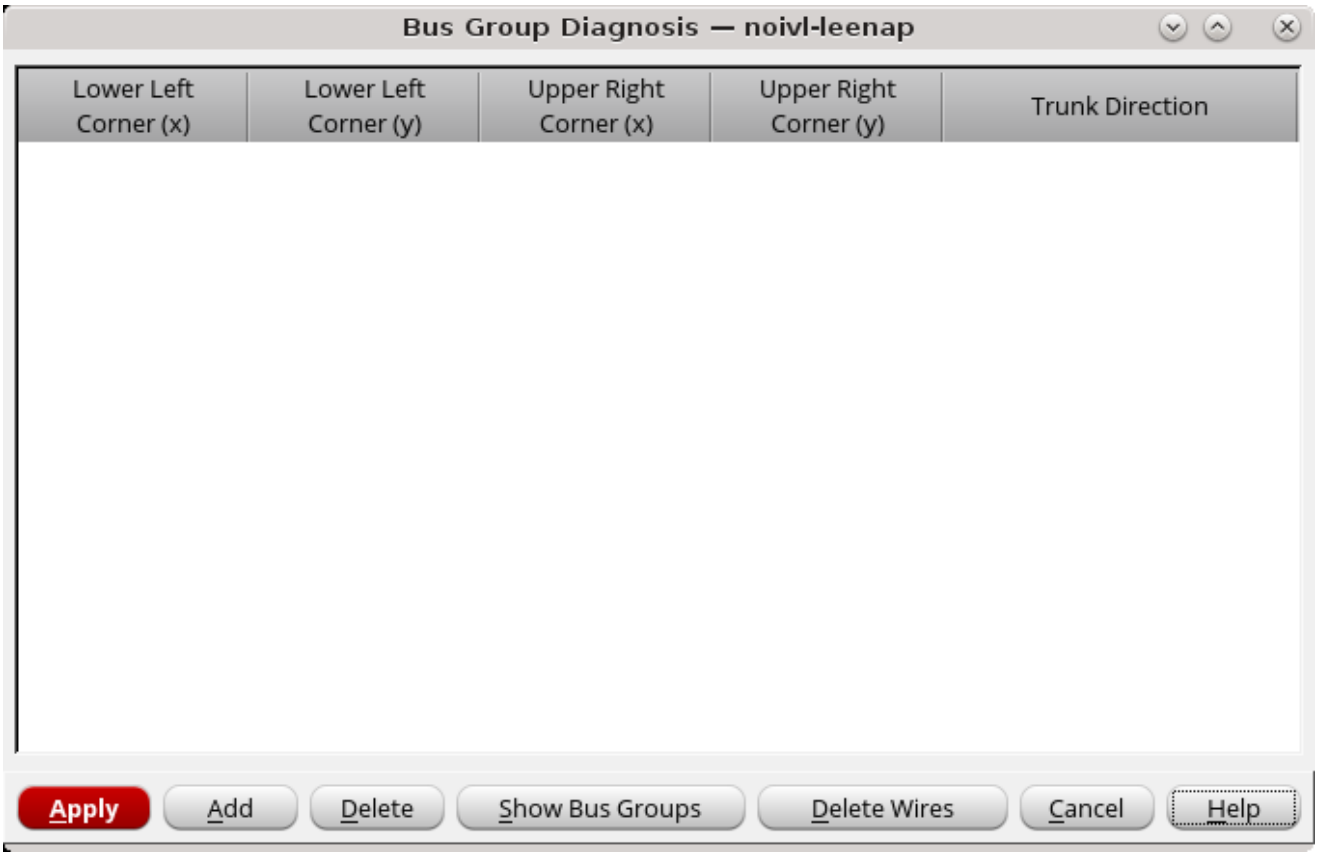
GSG	Specifies a shield mode with exactly *1* signal between shielding. This is also the default setting if no input is specified in the*Shielded Signal Number* field in the Interposer Route form *Default*: On
GSSG	Specifies a shield mode with exactly *2* signals between shielding. 

GxSG	Specifies a shield mode with *3* or more signals between shielding. 

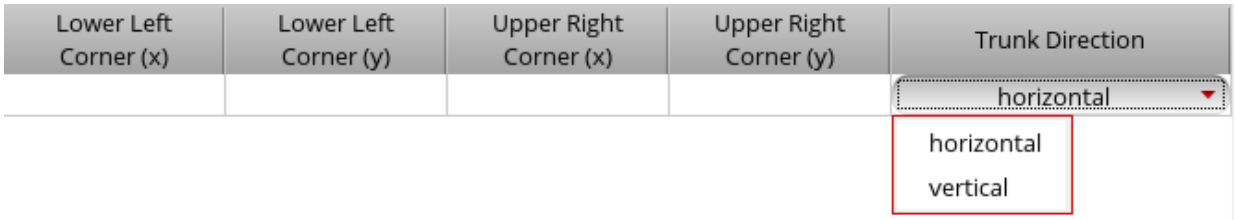
Bus Group Diagnosis

Use the Bus Group Diagnosis form to view bus group details. To open the form:

- Click the *Bus Group Diagnosis* button in the Interposer Route form.



Bus Group Diagnosis - Basic Fields and Options

Add	Adds a trunk. You can then specify its coordinates and direction. Double-click in any of the corner coordinate fields to enter the required value. 
Delete	Deletes a trunk.
Show Bus Groups	Shows the bus groups in the Implementation window. You can use the `add_gui_shape` command to draw rectangles representing routing guides.
Delete Wires	Opens the Select/Delete/Deselect Routes form, which you can use to delete a wire. See "*Select/Delete/Deselect Routes*" in the Edit Menu chapter for details.

Related Commands

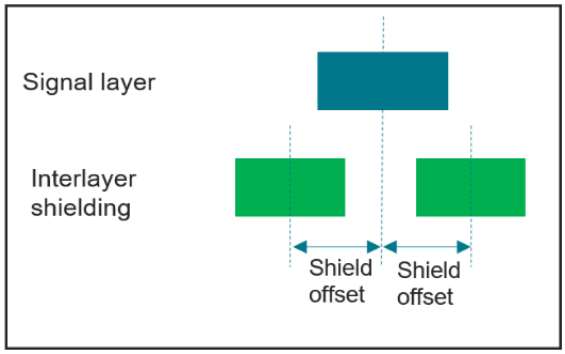
- `add_gui_shape`

Interlayer Shielding

Use the Interlayer Shielding form to view details of the shielding between layers. To open the form:

- Click the *Interlayer Shielding* button in the Interposer Route form.

Interlayer Shielding - Basic Fields and Options

Add	Adds a new row. You can then select a signal layer and specify its inter-layer shielding details. • *Signal layer*: Specifies the signal layer for which inter-layer shielding is being added. • *Shielded above/below/both*: Specifies the shielding direction for the selected signal layer. • *Shield net*: Specifies the name of the shield net. • *Shield width*: Specifies the inter-layer shielding widths. By default, all routing layers (used by current routing command) are shielded by the default width wires. • *Shield offset*: Specifies the center-to-center offset values between the shielding and signal wires. By default, all routing layers (used by current routing command) have an offset value of 0. • ?: Displays the following shield offset depiction when you hover over the '?'. 
Delete	Deletes the selected row.
Delete All	Deletes all rows.

Related Commands

The options in this form are equivalent to the following `setNanoRouteMode` options:

- `-route_interposer_interlayer_shielding_layers` *value*
- `-route_interposer_interlayer_shielding_nets` *value*
- `-route_interposer_interlayer_shielding_offsets` *value*
- `-route_interposer_interlayer_shielding_widths` *value*

12

Configuring and Launching System Level Analysis

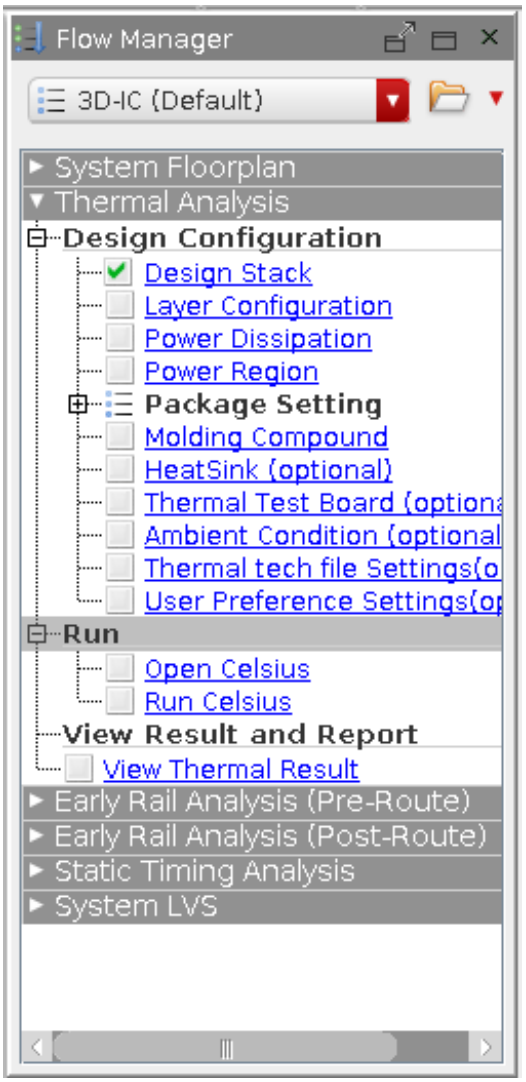
- Using The Flow Manager
  - Thermal Analysis
  - Early Rail Analysis
  - Static Timing Analysis
  - System LVS

Using The Flow Manager

The Flow Manager organizes and presents commands relevant to certain types of design objects or design tasks. It is used to simplify access to commands and flows that are commonly used. The Flow Manager provides step-by-step guidance for setting up analysis flows and launching analysis engines to do simulations and show results.

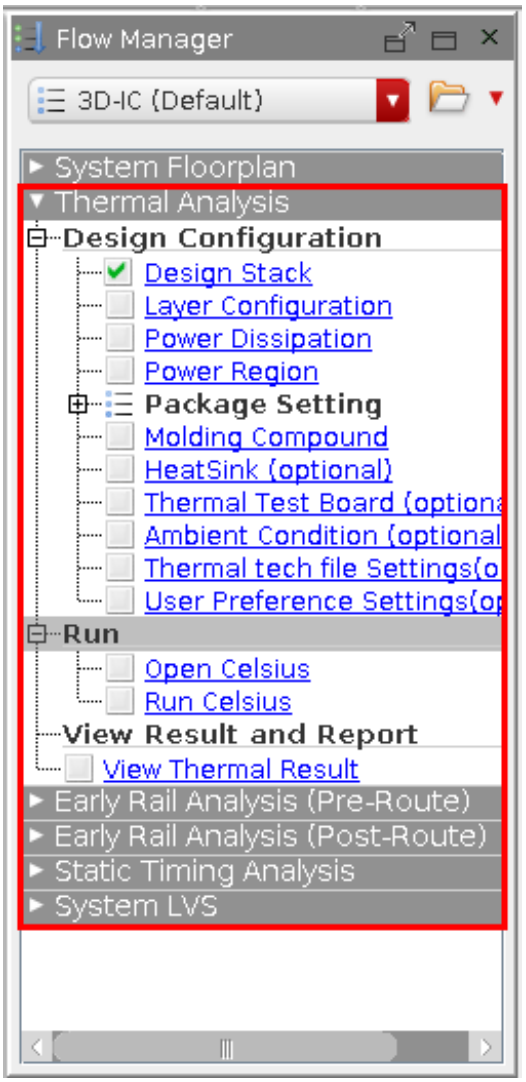
Expanding a flow menu presents the commands relevant to that flow, as shown below. Once a command or step has been executed from the Flow Manager, a green check mark is displayed in its check box. This is helpful when using Design flows with a specific set of steps. The *Reset State* command in the *View* menu can be used to clear these markers.

Double-clicking expanded menus will collapse them.



The current build of the Flow Manager has a built-in *3D-IC (Default)* flow with the following system analysis flow templates:

- Thermal Analysis
- Early Rail Analysis (Pre-Route and Post-Route)
- Static Timing Analysis
- System LVS



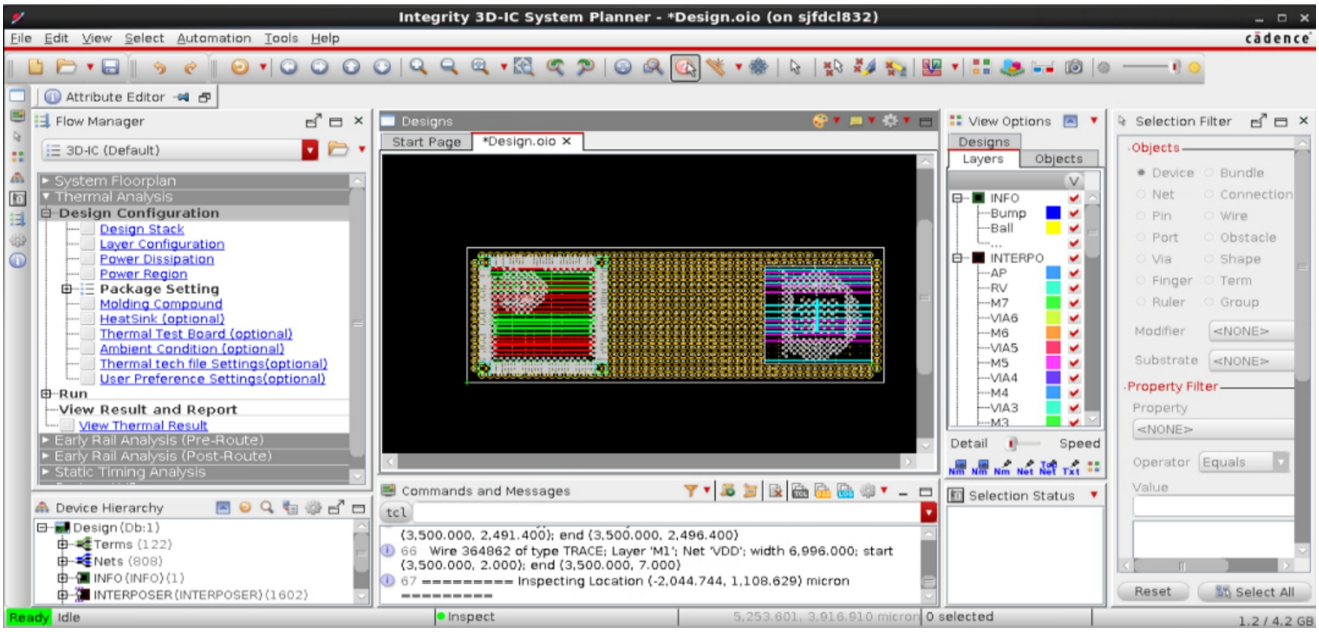
The following sections show a step-by-step example for each of the analysis flows in the default 3D-IC Flow Manager.

Thermal Analysis

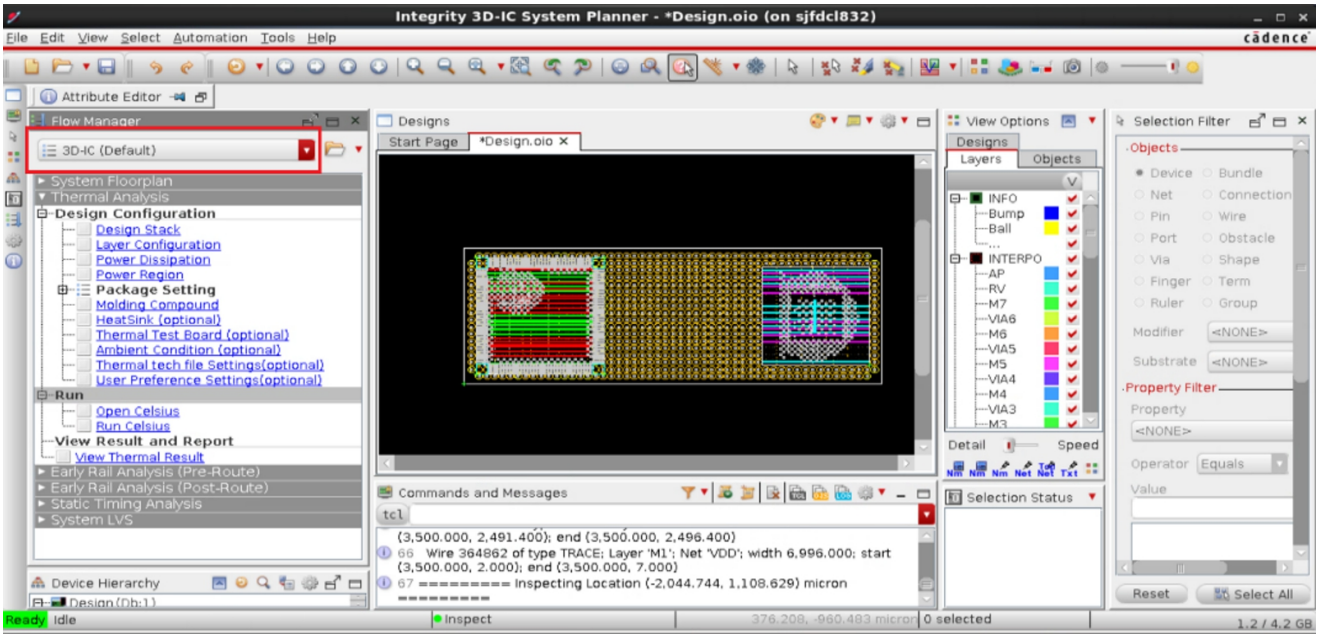
The performance of 3D-IC designs is dependent on the thermal behavior and temperature distribution among components. Integrity 3D-IC provides the capability of performing thermal analysis of 3D-IC designs.

To configure and perform Thermal Analysis from the System Planner Flow Manager, follow the steps given below:

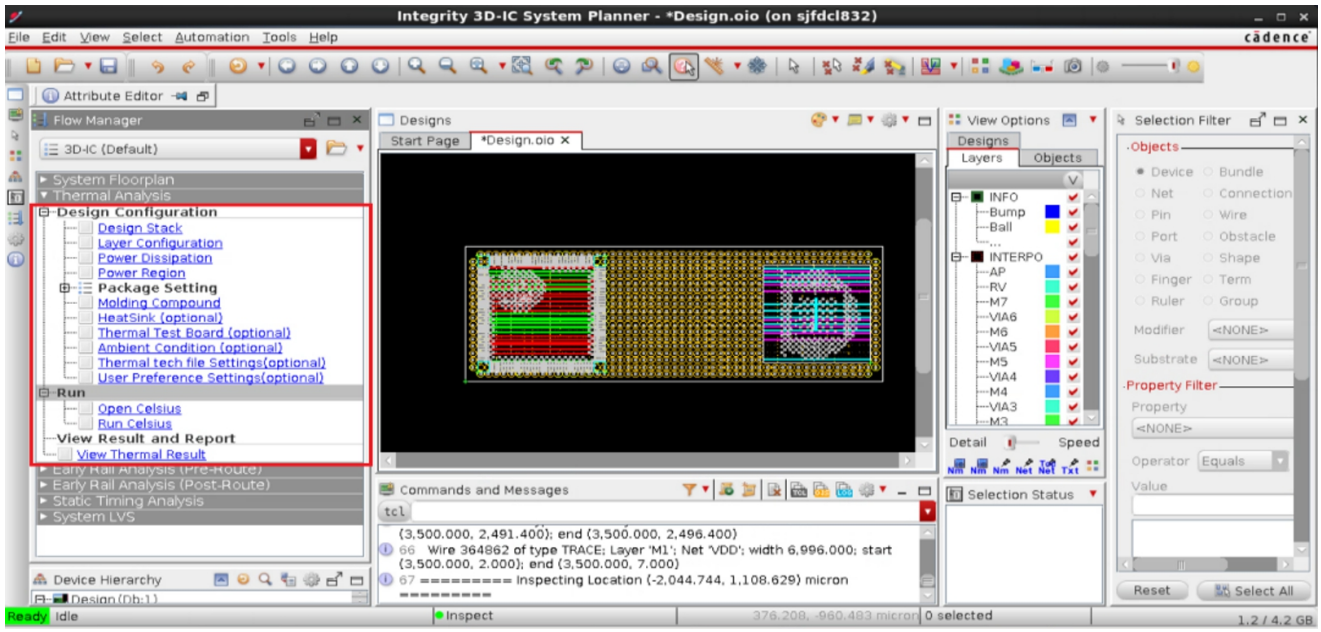
1. Load the multi-die 3D-IC design.



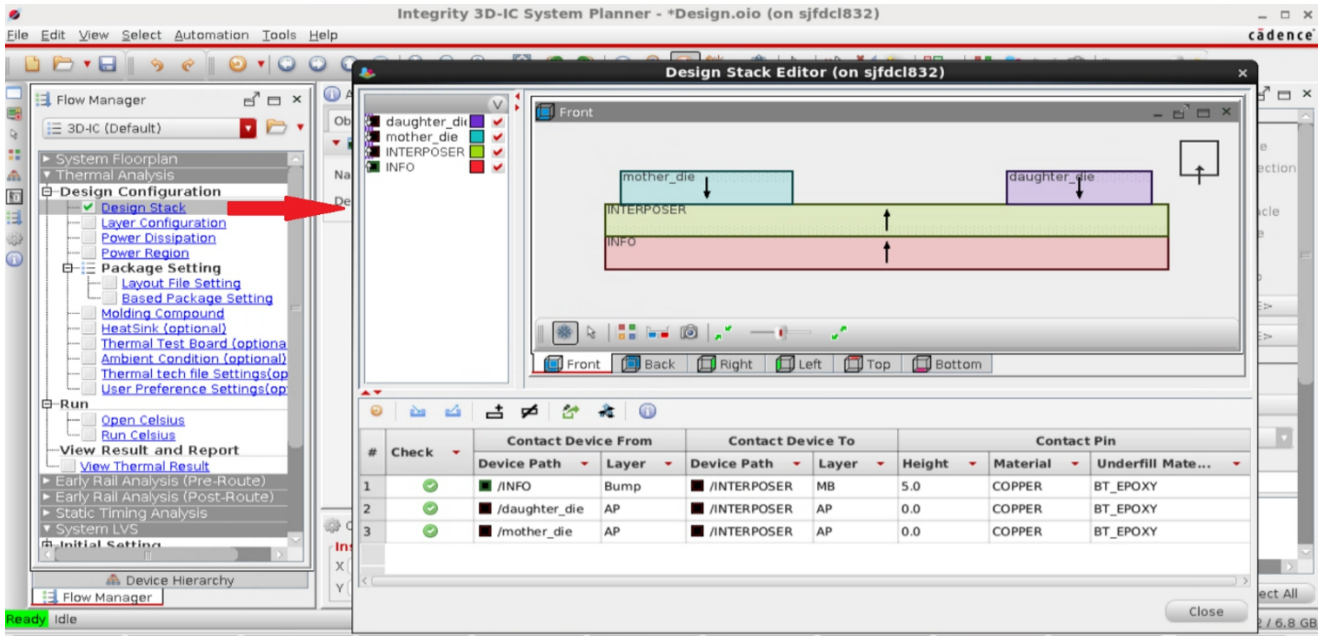
2. In the Flow Manager, ensure 3D-IC (Default) is selected.



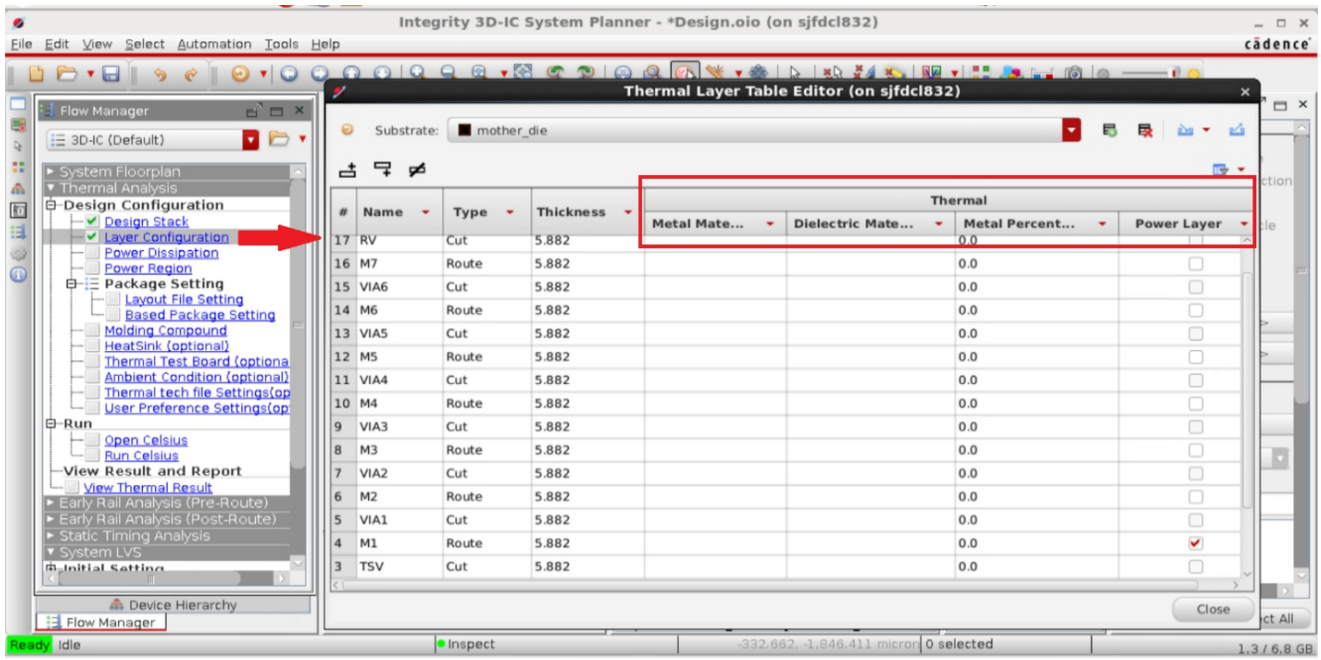
3. Double-click *Thermal Analysis*. The expanded menu is displayed, as shown below.



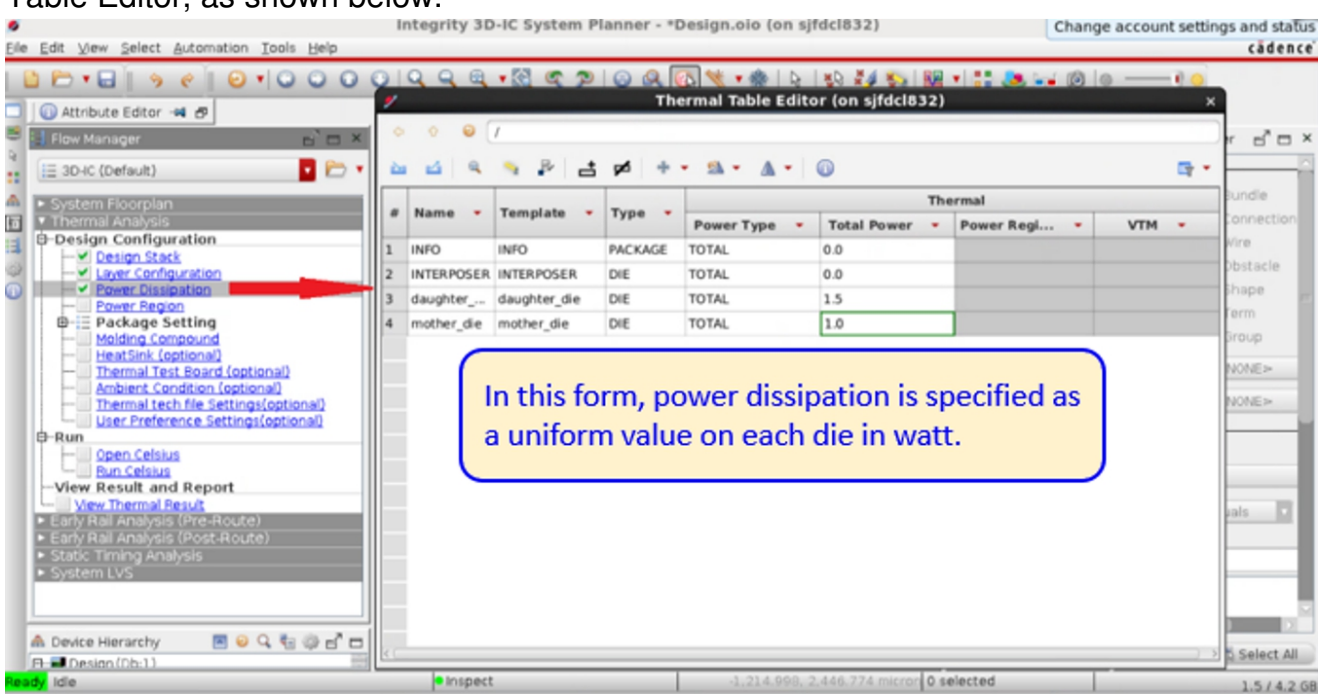
4. Under *Design Configuration*, select *Design Stack* to open the Design Stack Editor and make any changes in the design stack, if required. For instance, you can change bump heights and material properties according to the design. For details, see "Defining the Design Stackup" section in Working with Designs in System Planner.



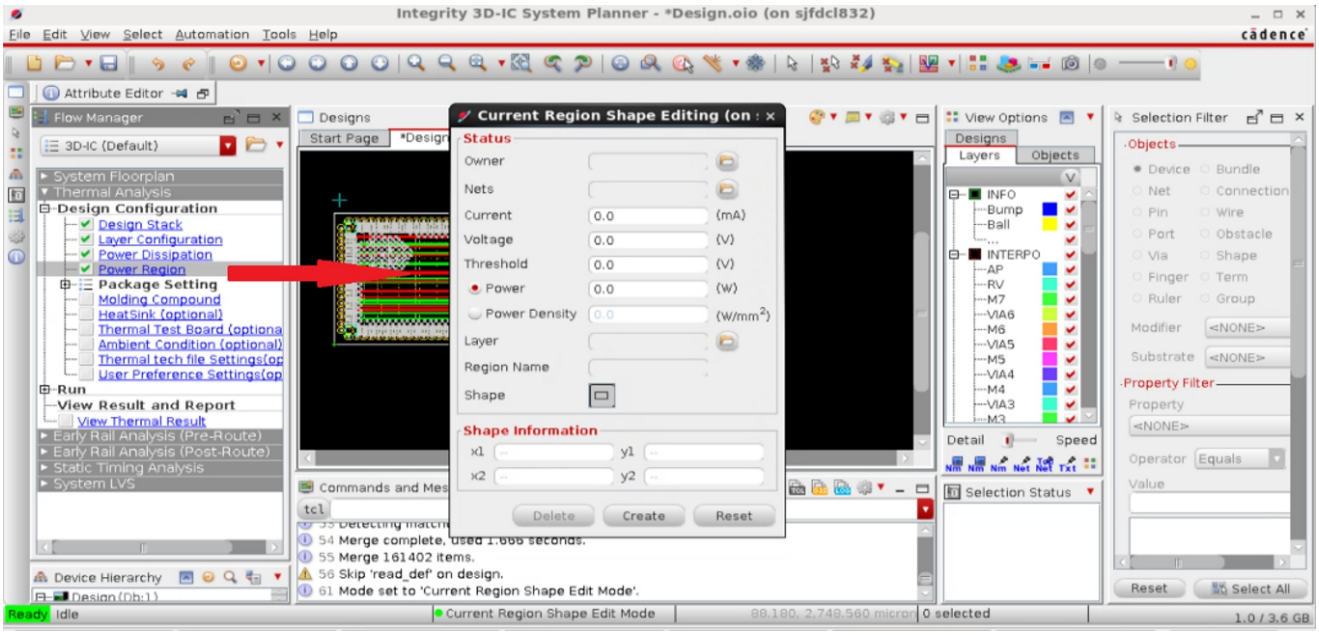
5. Next, select *Layer Configuration* to open the Thermal Layer Table Editor. Here, you can specify the thickness, metal material, dielectric material, and metal percentage for each die or retain the default values. You also need to specify one power layer for each die.



6. Next, select *Power Dissipation* to specify power dissipation values for each die in the Thermal Table Editor, as shown below.

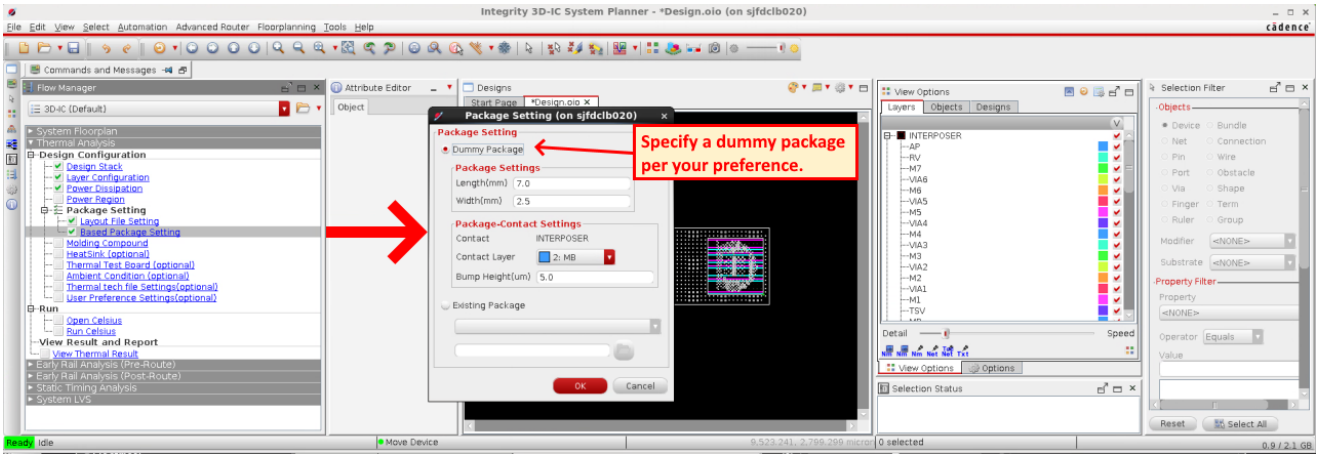


7. If you specified `POWERMAP` as *Power Type* in the previous step, you must click *Power Region* to specify the current, voltage, threshold, power and shape information for these power regions in the Current Region Shape Editing dialog.

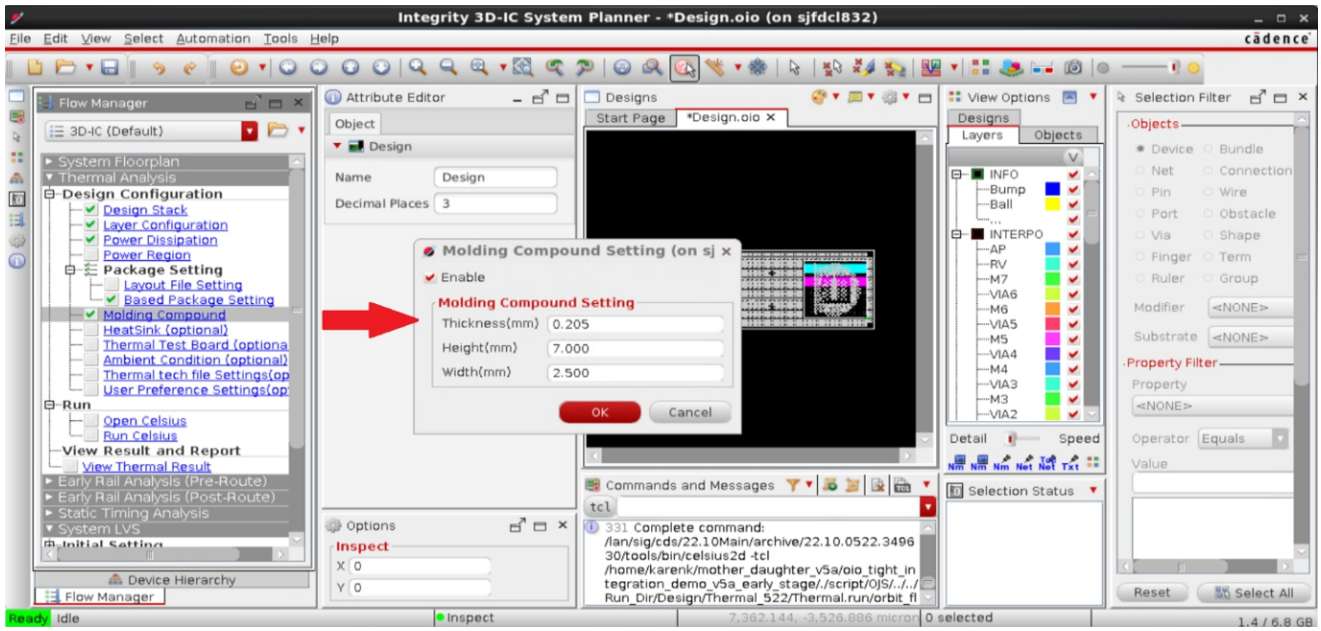


8. Now, select *Package Setting - Based Package Setting*. Select the *Dummy Package* check box and the tool will create a dummy package for you per the specified details, as shown below.

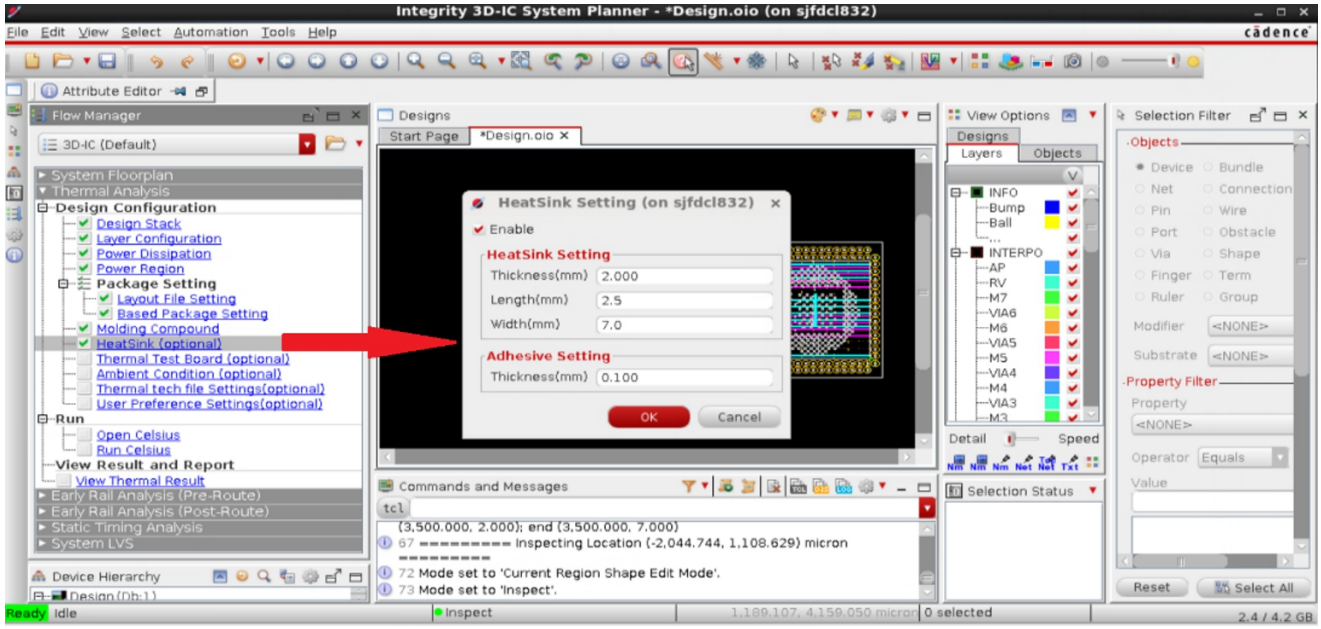
> Do not use the *Existing Package* check box as there are some issues in thermal simulation with an existing package in this release. This issue will be addressed in the next release.



9. Next, click *Molding Compound* to configure the molding compound. to enable You can specify the required thickness, height, and width in the Molding Compound Setting dialog.

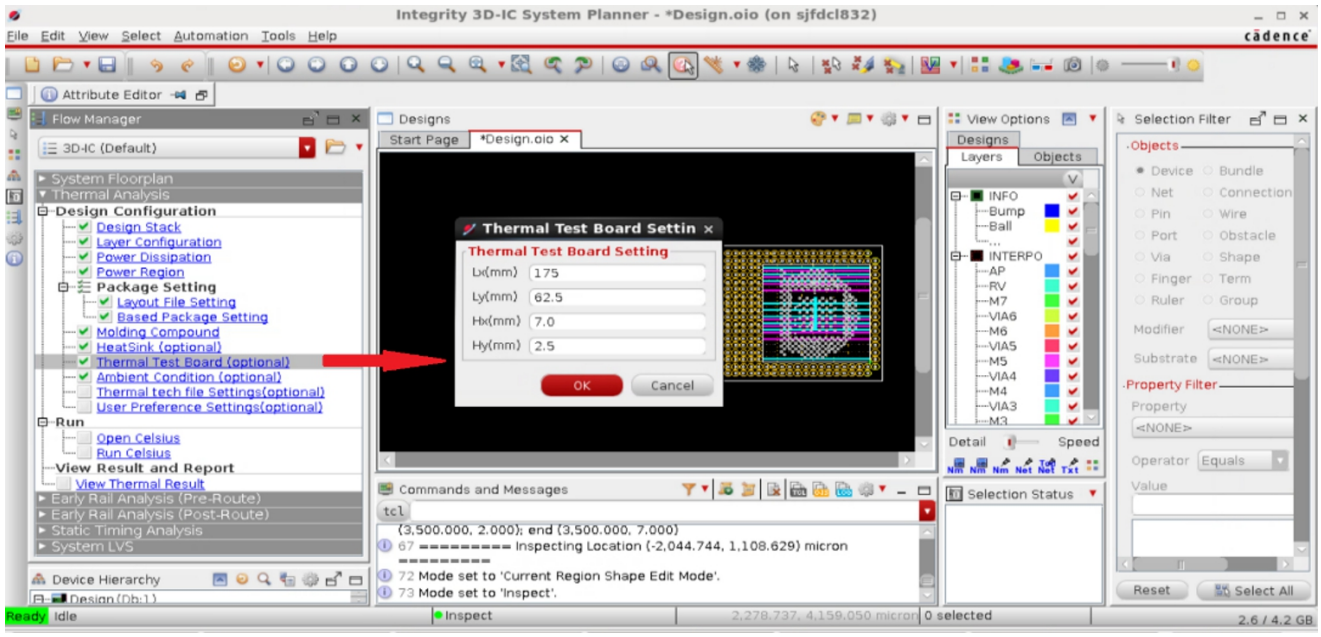


10. If applicable, set up a Heat Sink on top of the package. To do so, select *HeatSink (optional)* from the *Design Configuration* menu and specify the heat sink and adhesive settings. The heat sink shall be approximated by a piece of metal, which should enhance the heat dissipation to the environment.

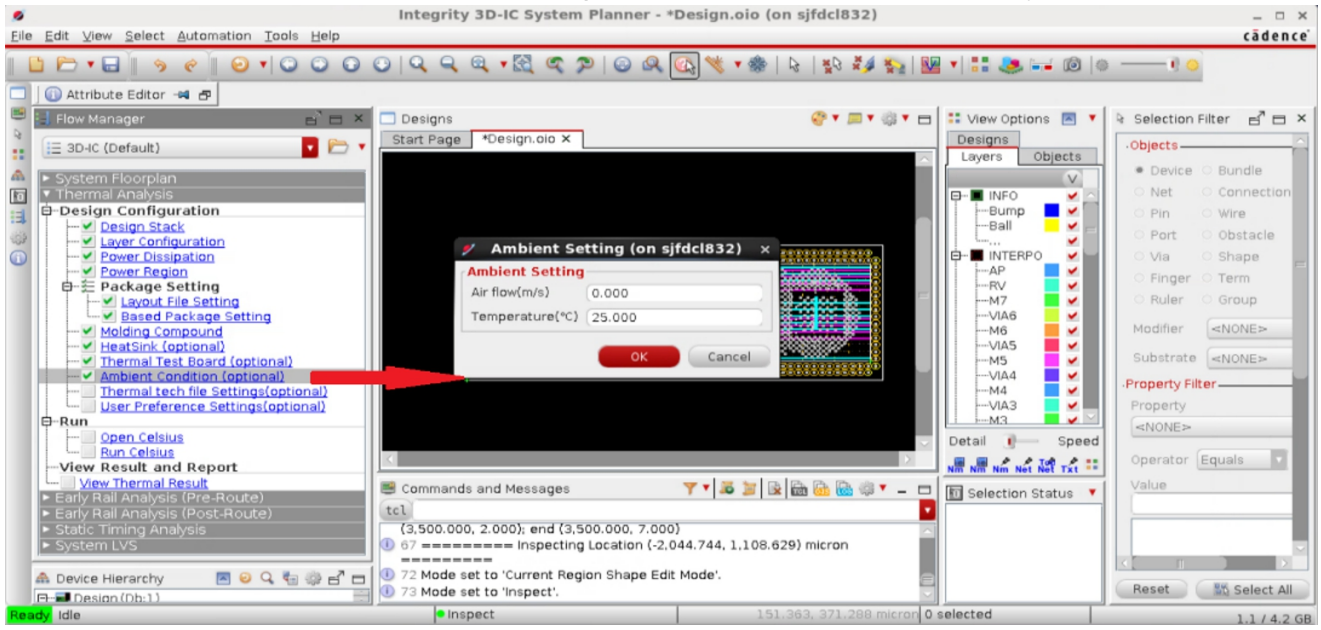


11. If applicable, set up a thermal test board. To do so, click *Thermal Test Board (optional)* and specify the board settings in the *Thermal Test Board Setting* dialog. Typically, a thermal test board should be included in the model because its effect in heat dissipation would not be negligible in most cases. The JEDEC standard thermal test boards can be taken as suitable

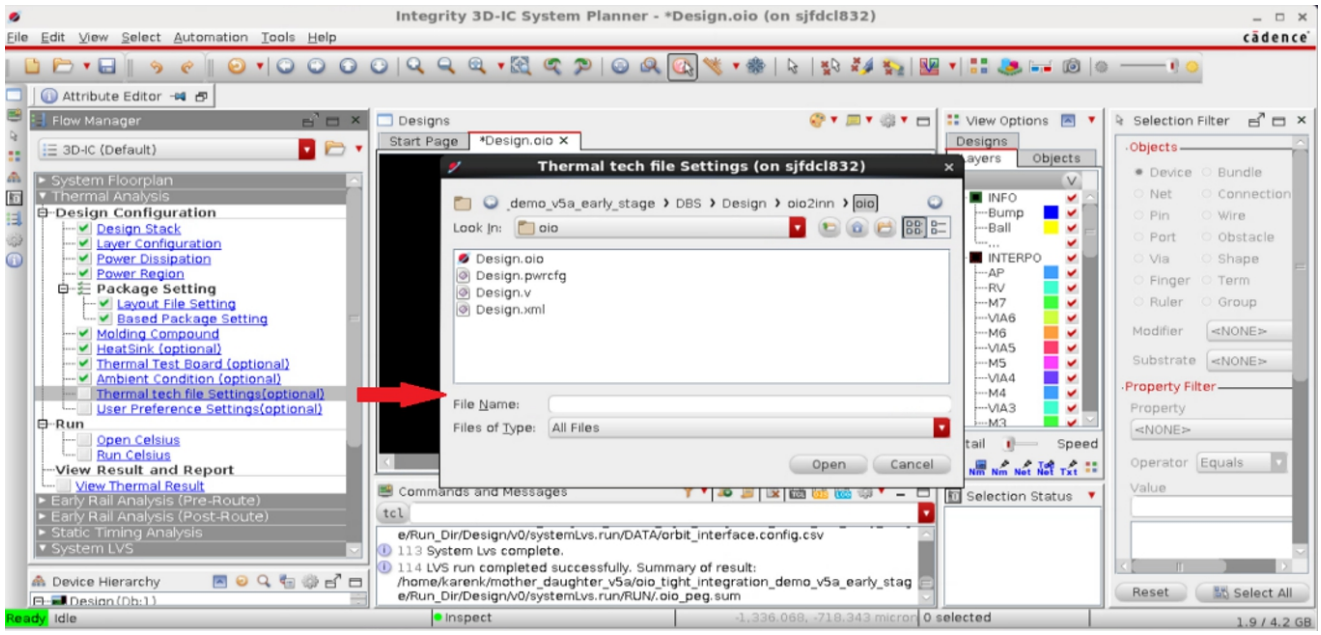
reference.



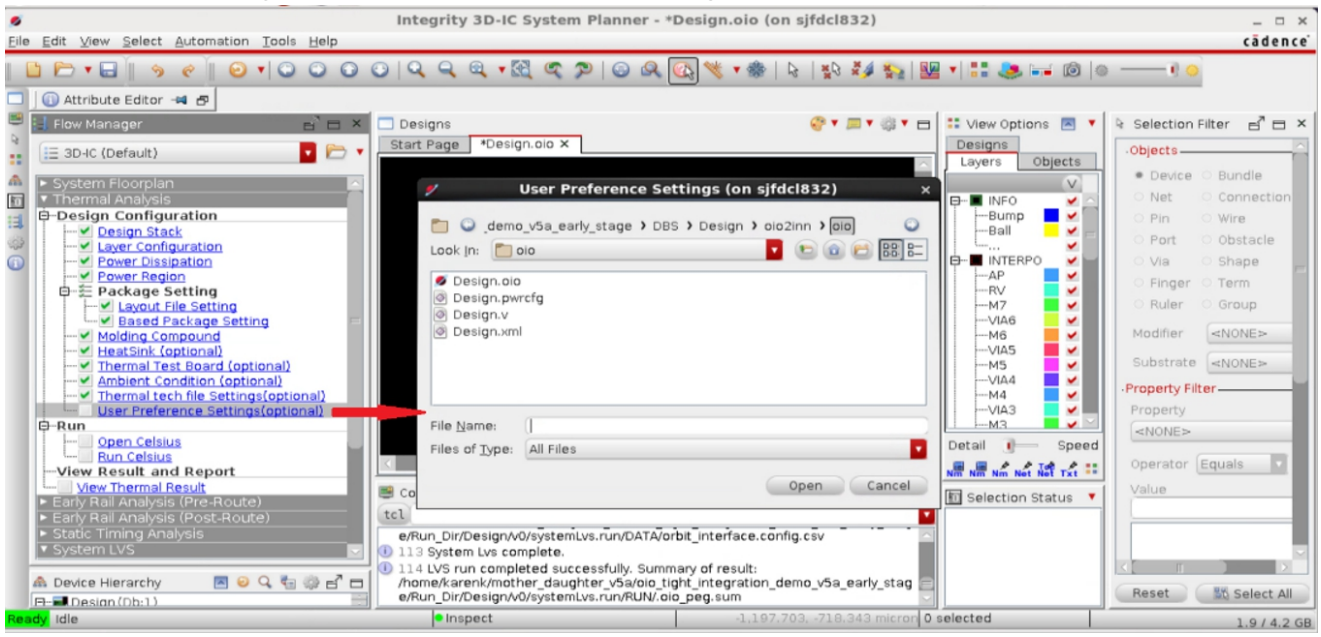
12. If applicable, specify the ambient condition settings. Select *Ambient Condition (optional)* and specify the required values in the Ambient Setting dialog. Ambient conditions in the setting include air flow spend and ambient temperature. These conditions are important for thermal simulation in most of cases. When not specified, the default values may be used.



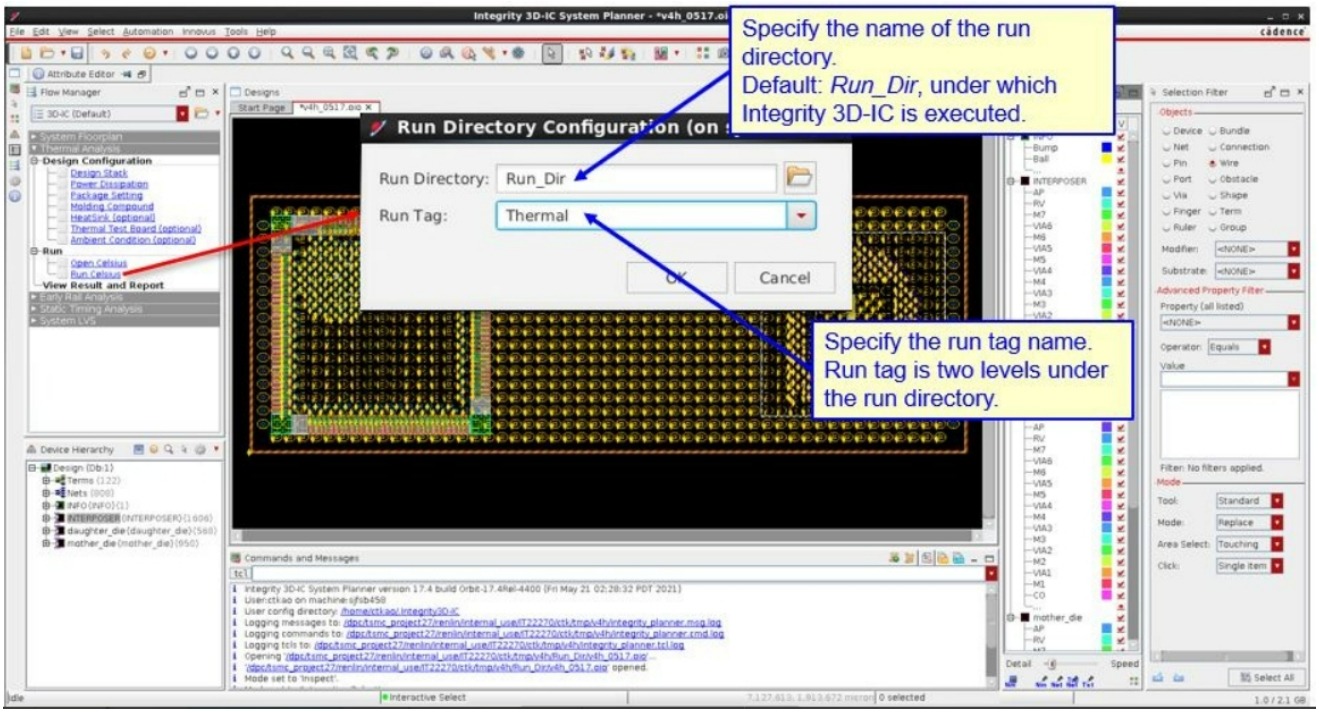
13. If an existing thermal tech file with component configurations is available, click *Thermal tech file Settings (optional)* and load that tech file.



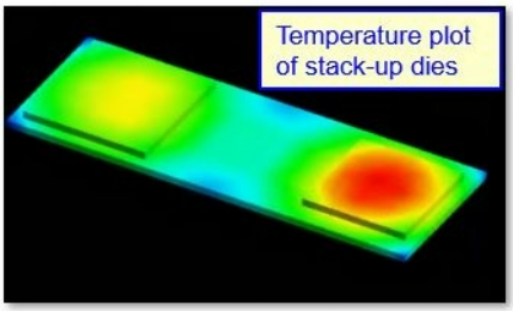
14. If a user preference settings file specifying simulation conditions is available, click*User Preference Settings (optional)* to load the existing file.

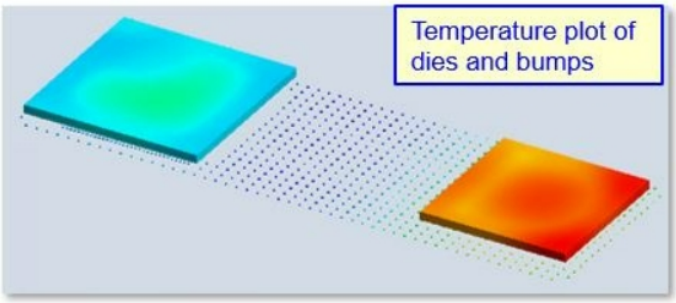


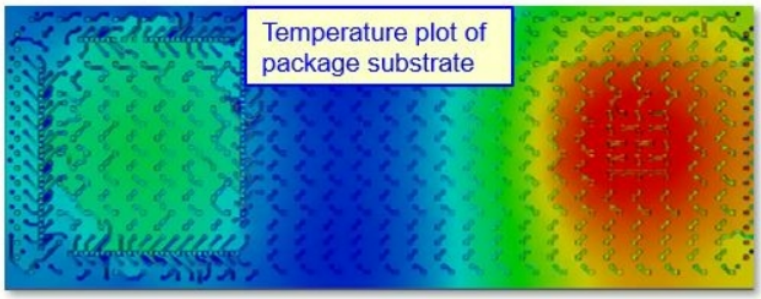
15. After completing *Design Configuration* settings, you can start the thermal analysis using Celsius™ Thermal Solver. To do so, select *Thermal Analysis – Run – Run Celsius*. In the resulting Run Directory Configuration dialog, specify the *Run Directory* and *Run Tag* settings and click *OK* to start the thermal analysis. This starts the Celsius Thermal Solver run in a separate session.



16. After the Celsius run is complete, you can view the Thermal Analysis results in the Celsius GUI. Refer to the *Celsius Thermal Solver User Guide* for post-solution operations.





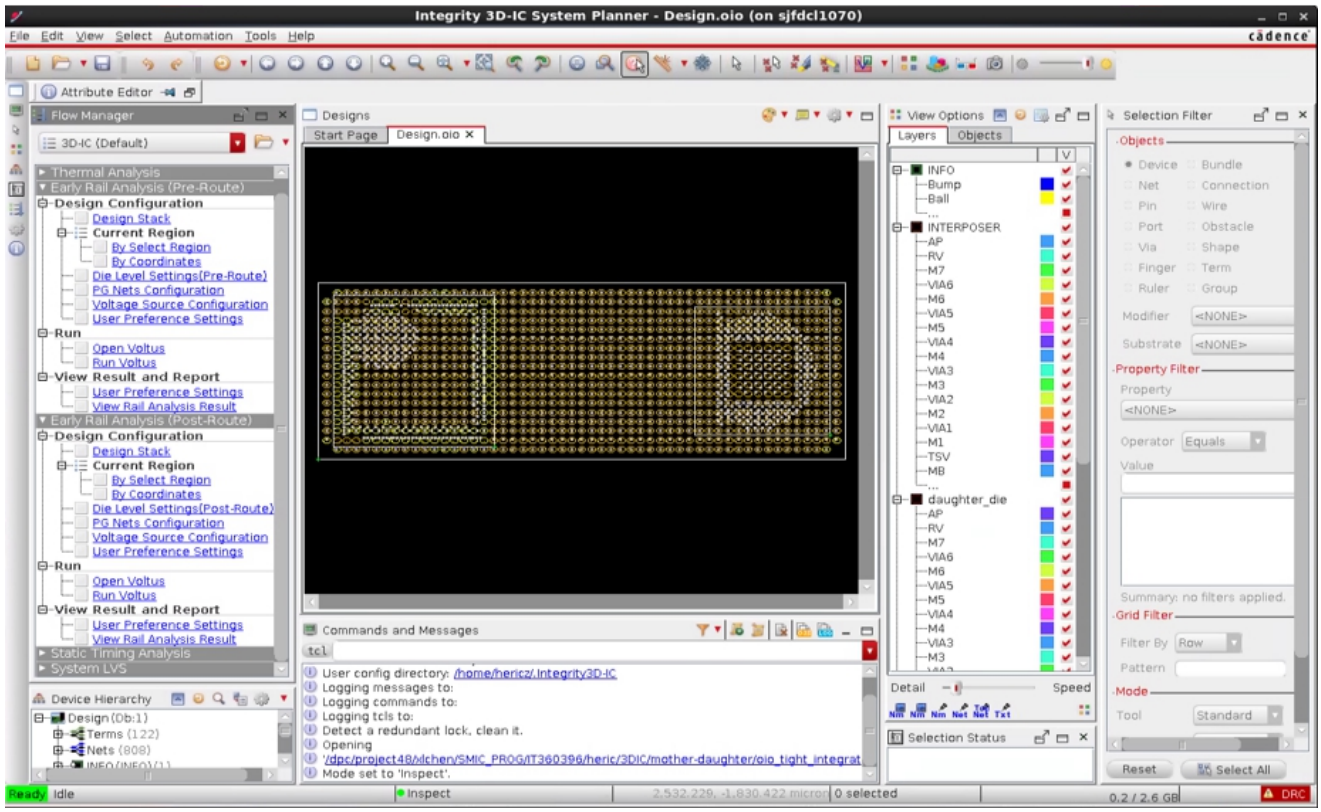


Early Rail Analysis

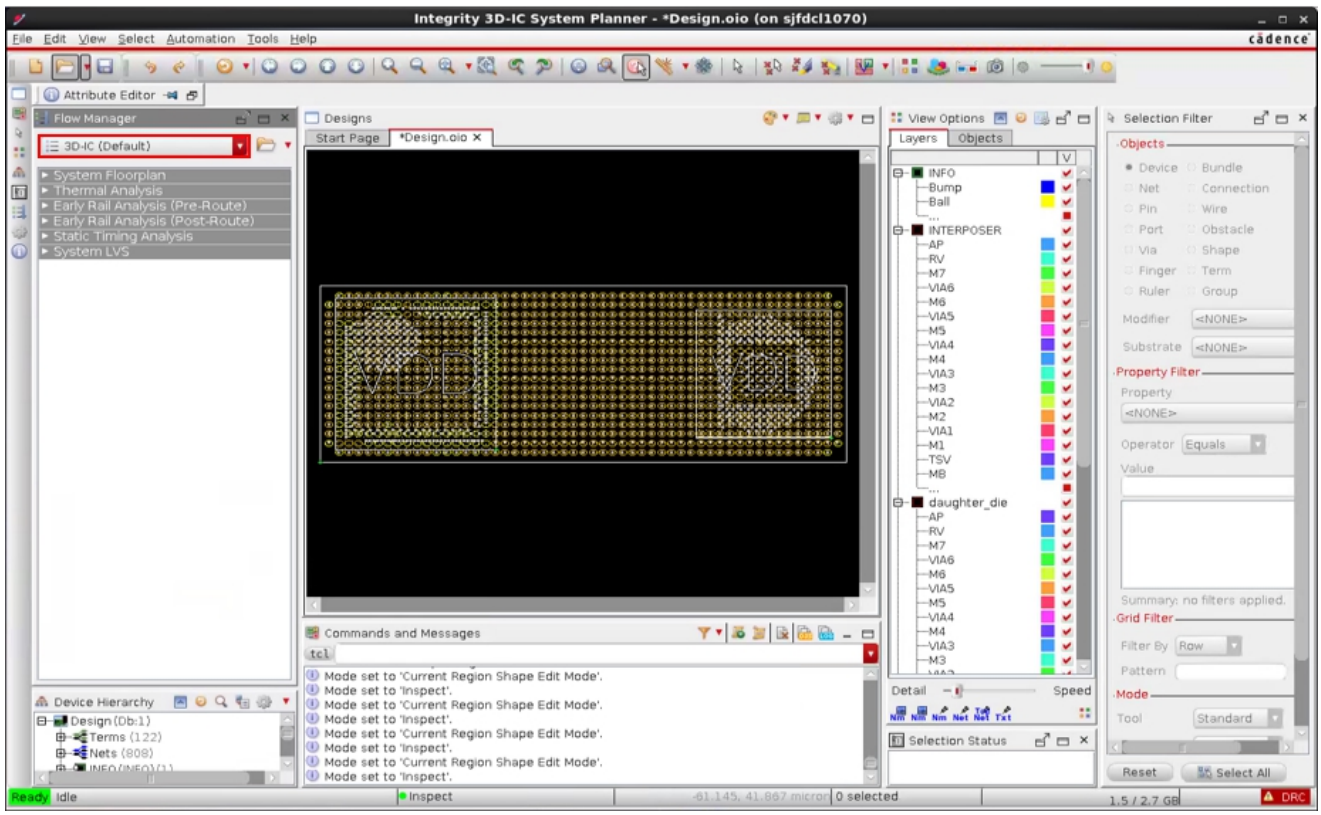
The performance of 3D-IC designs is also dependent on the power allocation/distribution and IR drop effects among multi-die configuration. Integrity 3D-IC, with its capability of performing Early Rail Analysis (ERA) of 3D-IC designs can achieve the multi-die stack-up data with detailed designs and optimized connections.

To configure and perform ERA from the System Planner Flow Manager, follow the steps given below:

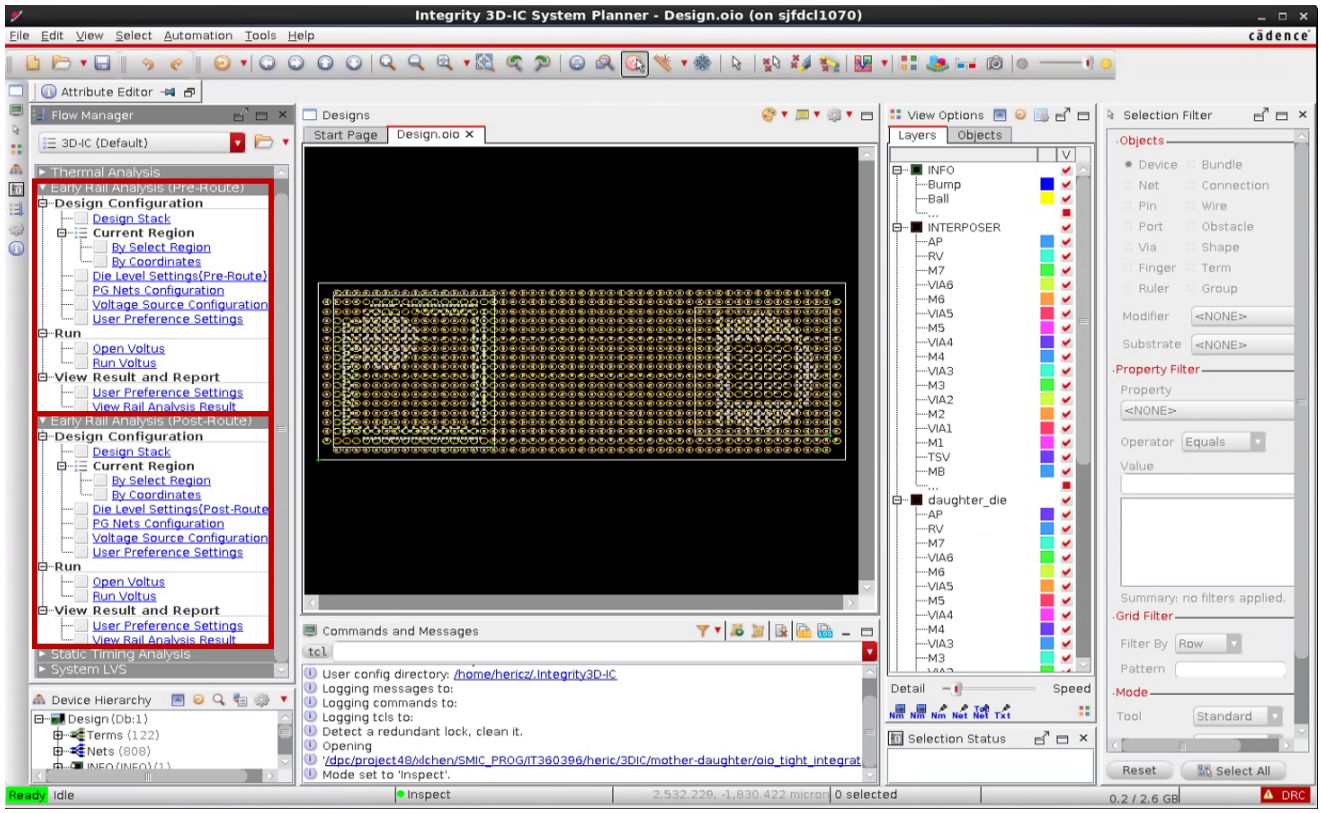
1. Load the multi-die 3D-IC design.



2. In the Flow Manager, ensure 3D-IC (Default) is selected.



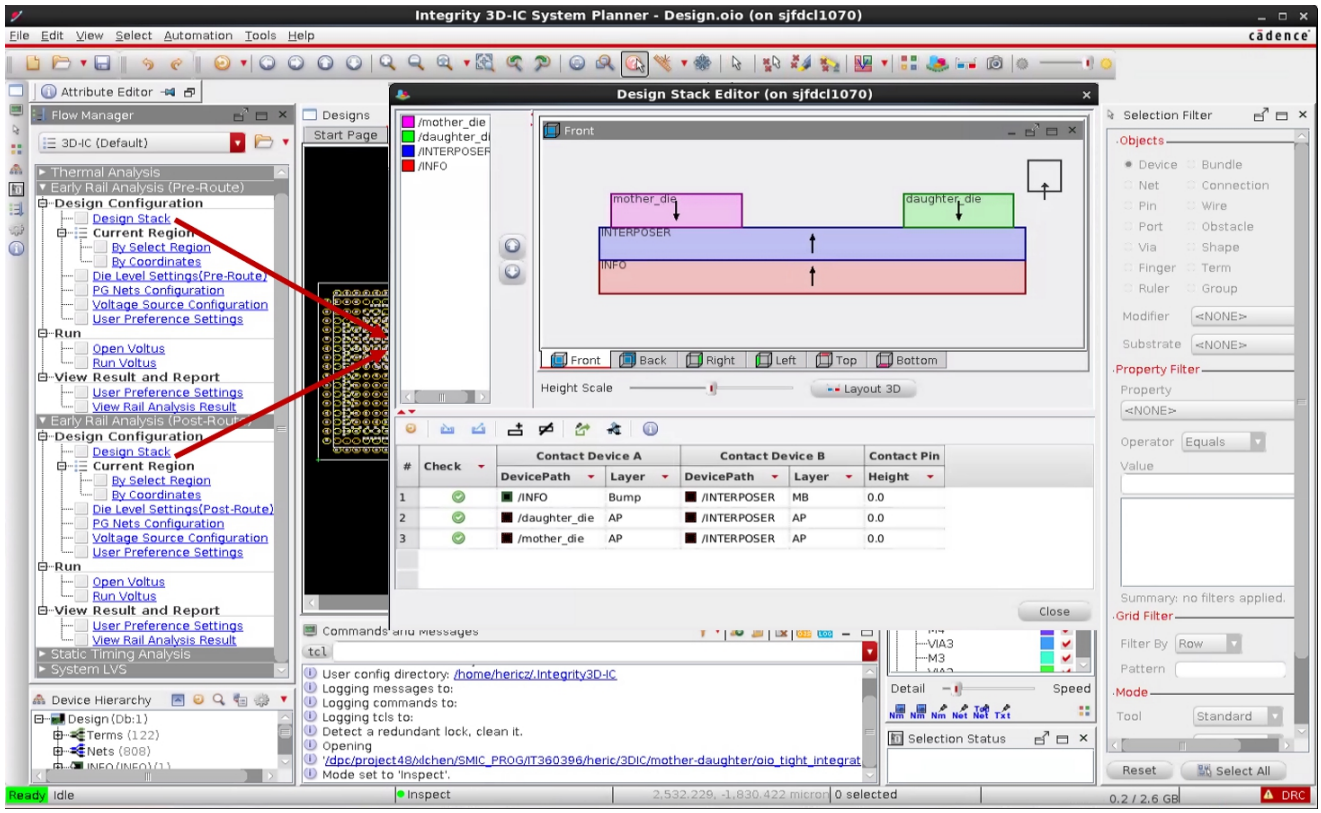
3. Double-click *Early Rail Analysis*. The expanded menu is displayed, as shown below.



Two configurations of ERA are included: Pre-Route and Post-Route.

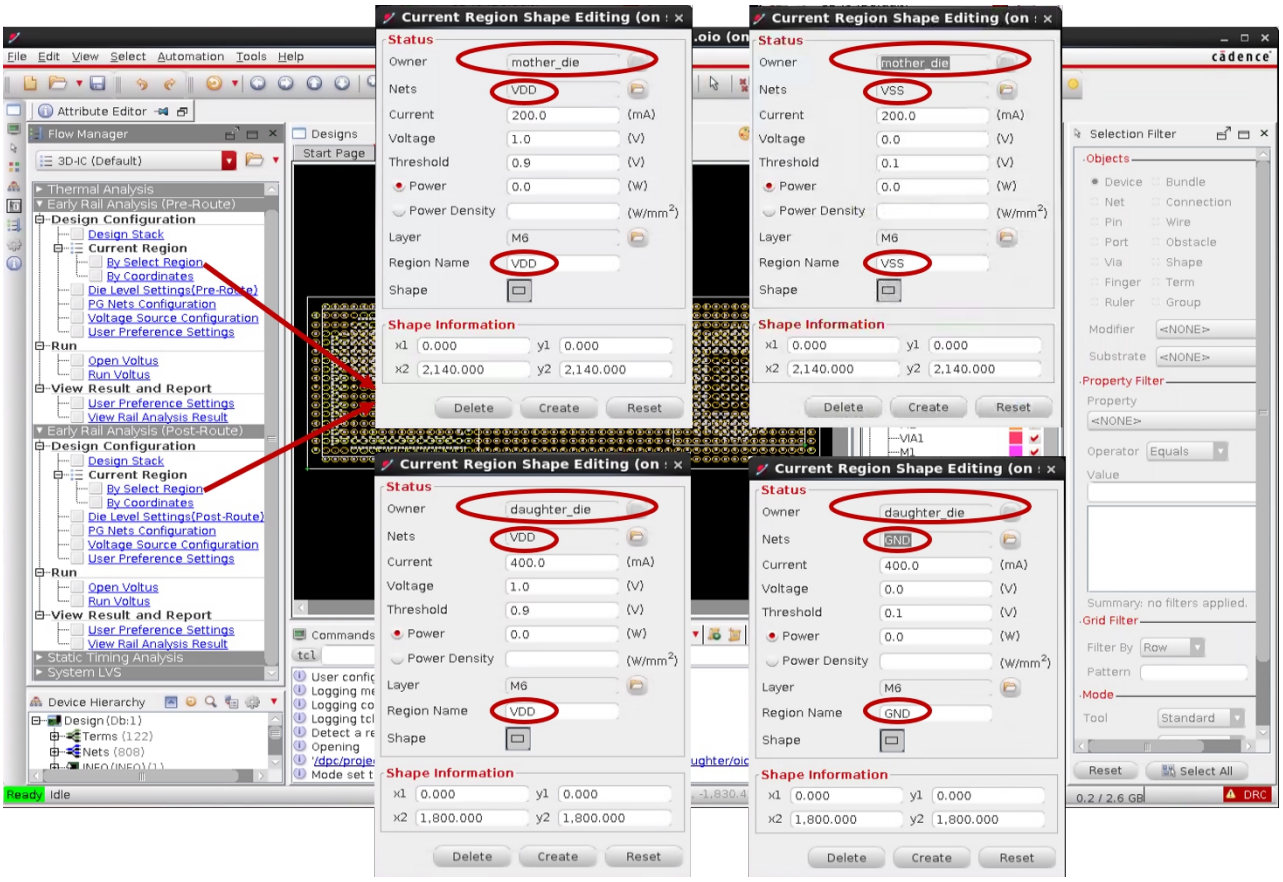
- Pre-Route: Uses a PSDL file as input data. The tool will generate a PG mesh through the PSDL file and create virtual bumps based on the `bump_region` file.
- Post-Route: Uses a P/G-finished-DEF file as input data.

4. Under *Design Configuration*, select *Design Stack* to open the Design Stack Editor and make any changes in the design stack, if required. For details, see "*Defining the Design Stackup*" section in Working with Designs in System Planner.

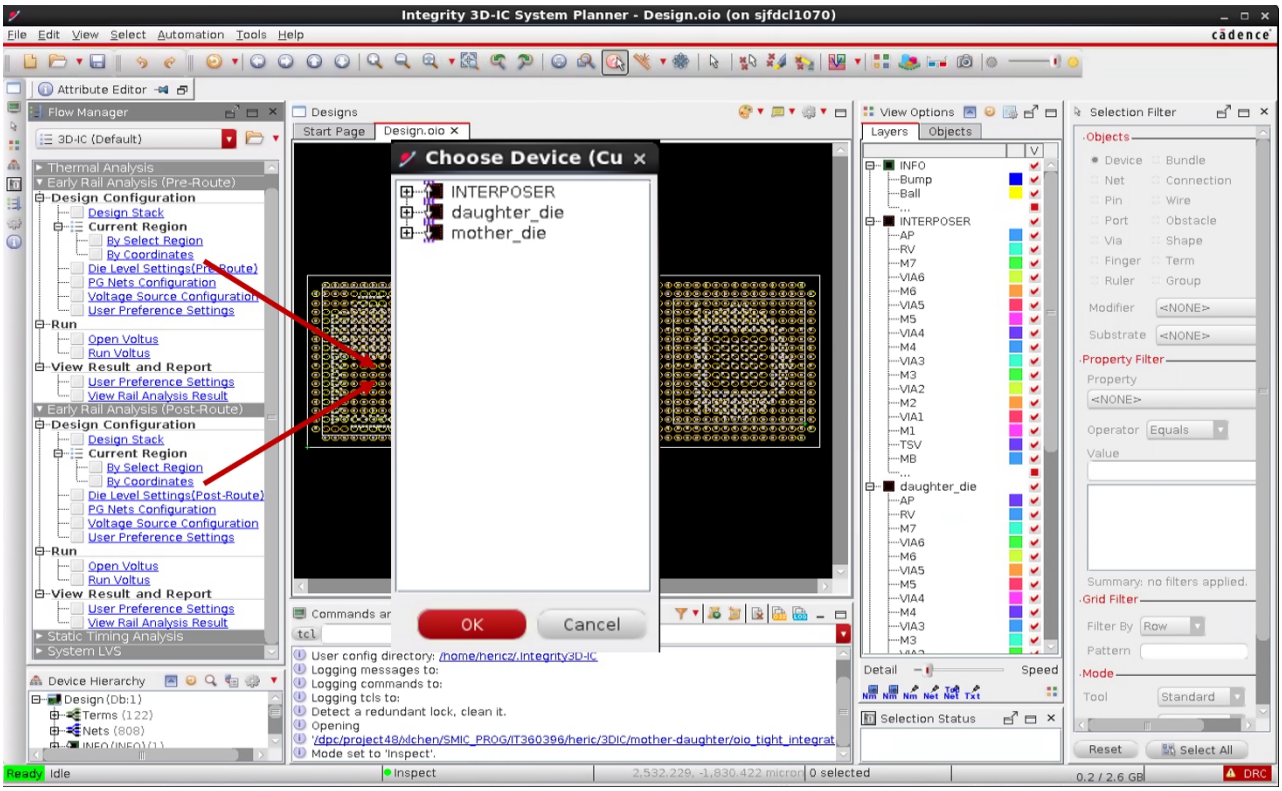


5. Next, set up the current region. To do so, expand the *Current Region* node under *Design Configuration*. Next:

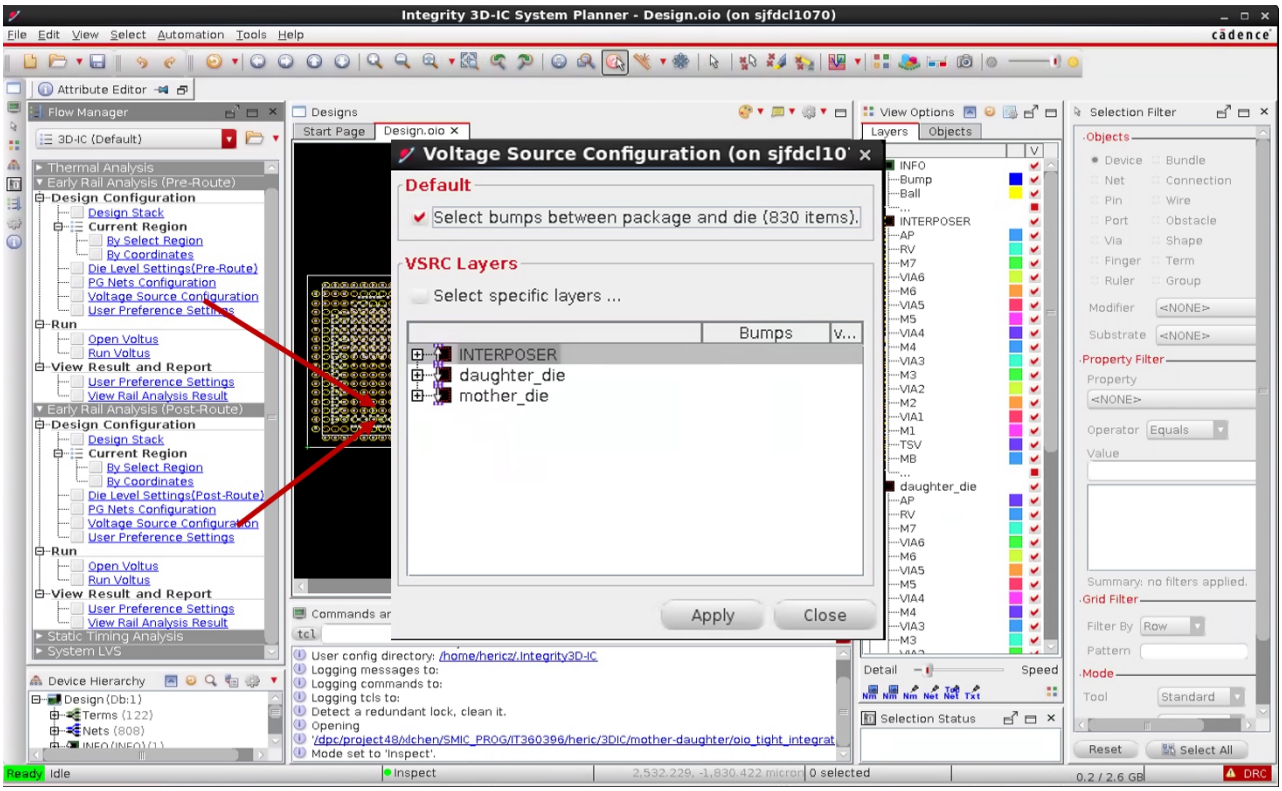
   a. Click *By Select Region*. In the resulting Current Region Shape Editing dialog, specify the current regions for each of the dies, as shown below, and create the current region by drawing a box using the mouse.



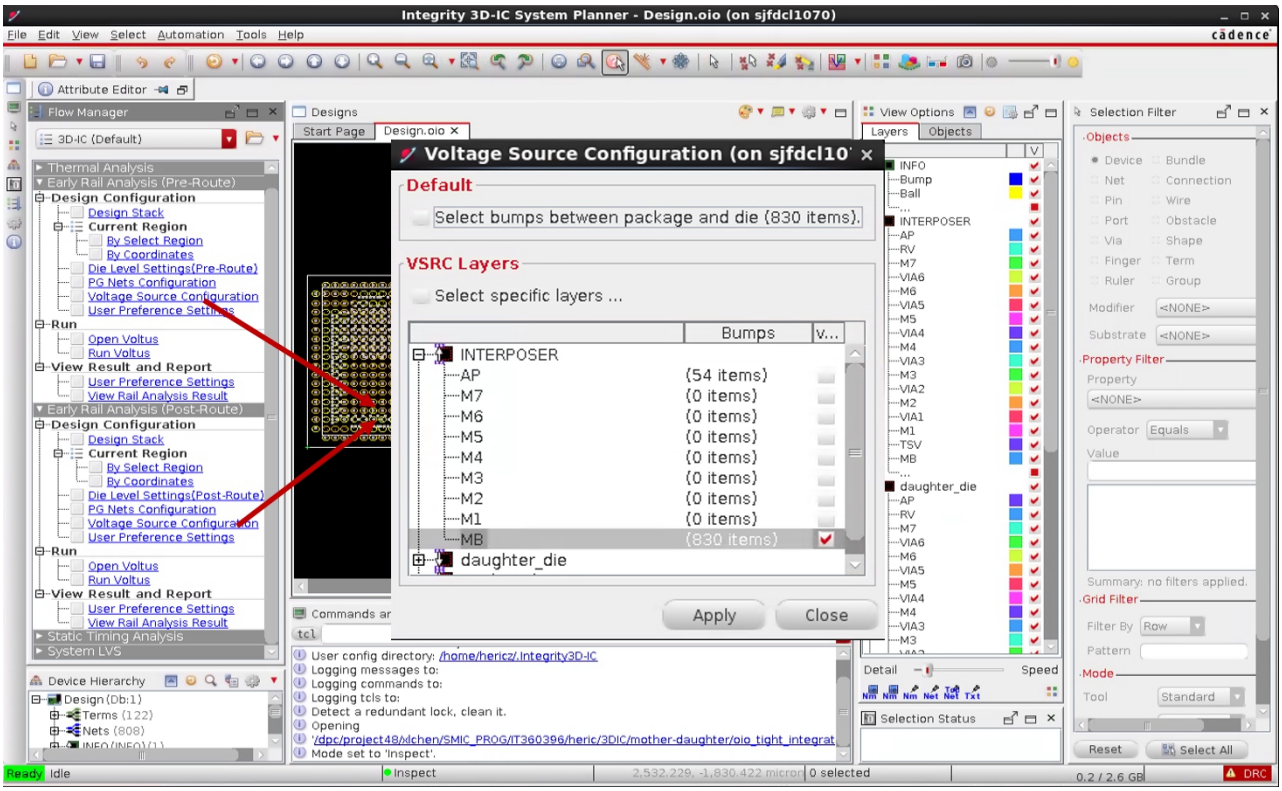
b. Click *By Coordinates* and verify the values for each of the dies in the resulting Choose Device (Current Region) dialog and modify the values, if needed.



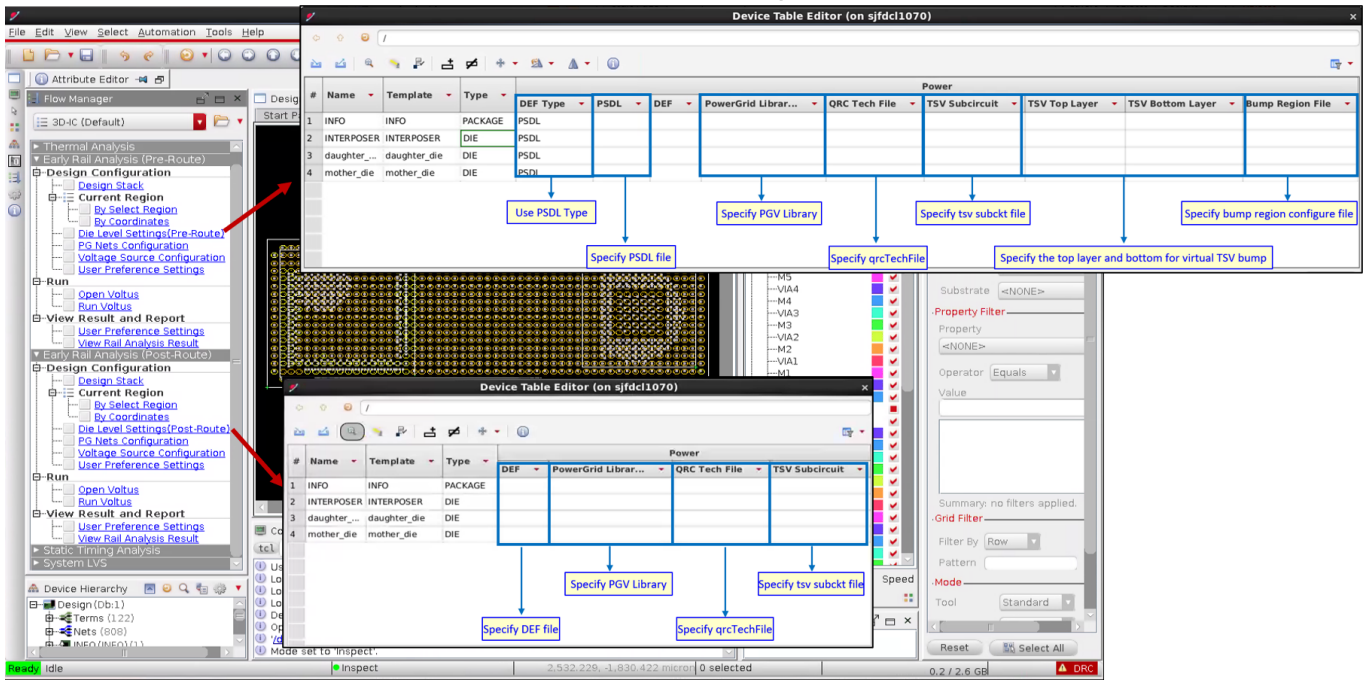
6. Now, select *Voltage Source Configuration* to set up the voltage source.
   a. If PKG Die is included in the design, select the *Select bumps between package and die* check box in the Voltage Source Configuration dialog to configure the voltage source for ERA.



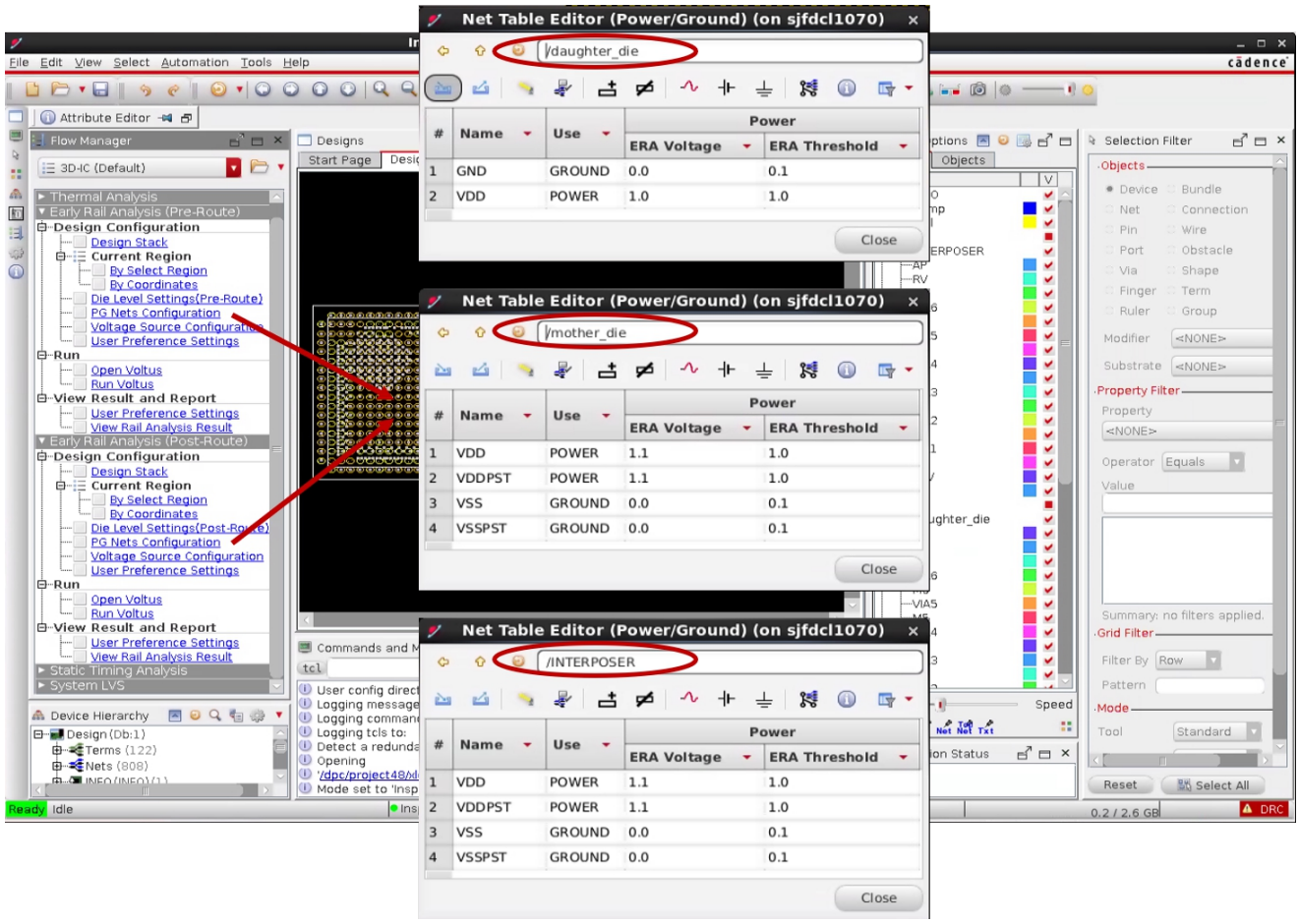
b. If PGK Die is not included in the design, the voltage source layers need to be specified manually.



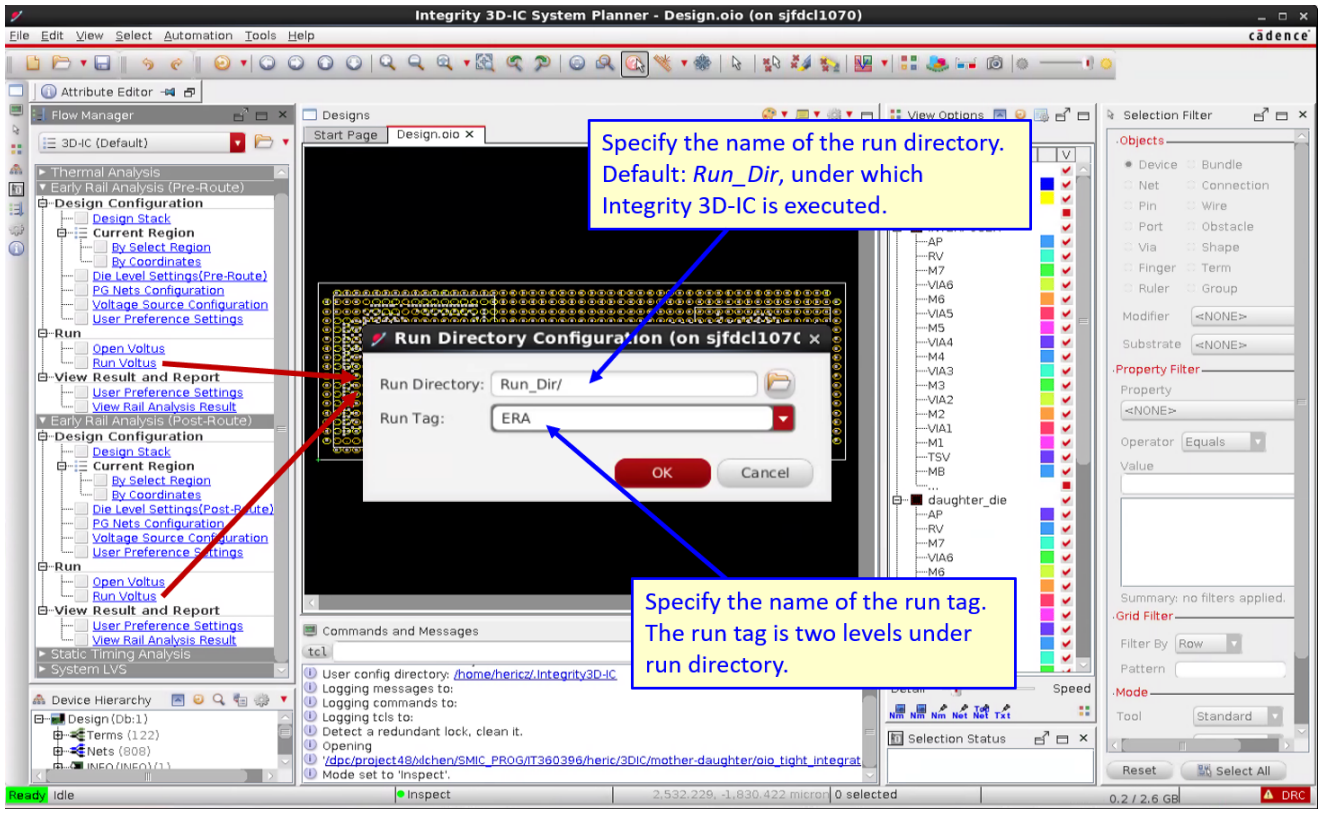
7. Next, click *Die Level Settings* to open the Device Table Editor. Here, specify the input data settings, as shown below.
   Note that Pre-Route and Post-Route need different input data.



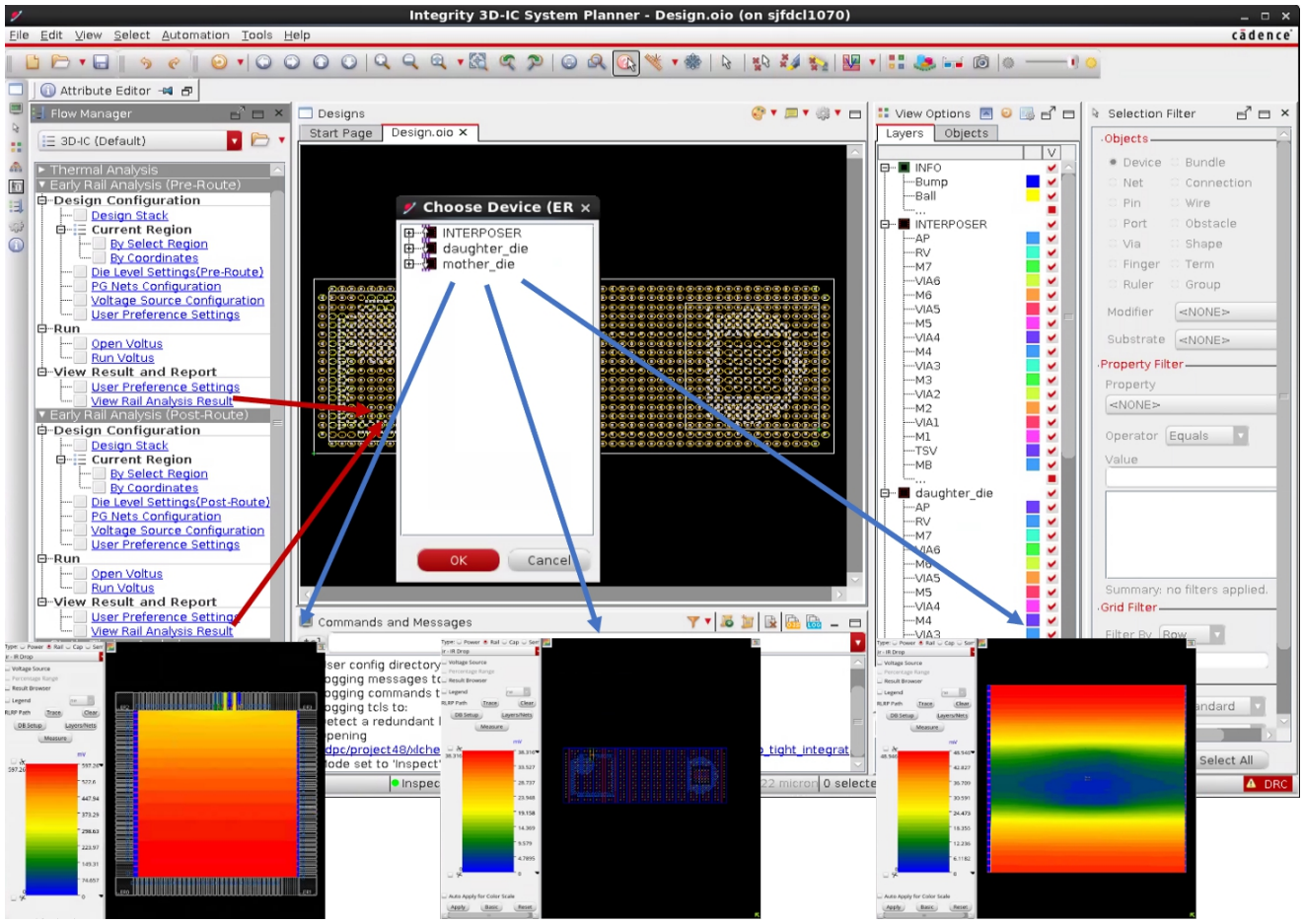
8. Now set up the PG nets for each of the die. Click*PG Nets Configuration* to open the Net Table Editor (Power/Ground) dialog. Select a die in the top field of the dialog and specify the *ERA Threshold* and *ERA Voltage* values for its PG nets. Repeat the process for the other dies.



9. You can now start the early rail analysis. To do so, select*Early Rail Analysis – Run – Run Voltus*. In the resulting Run Directory Configuration dialog, specify the*Run Directory* and *Run Tag* setting.



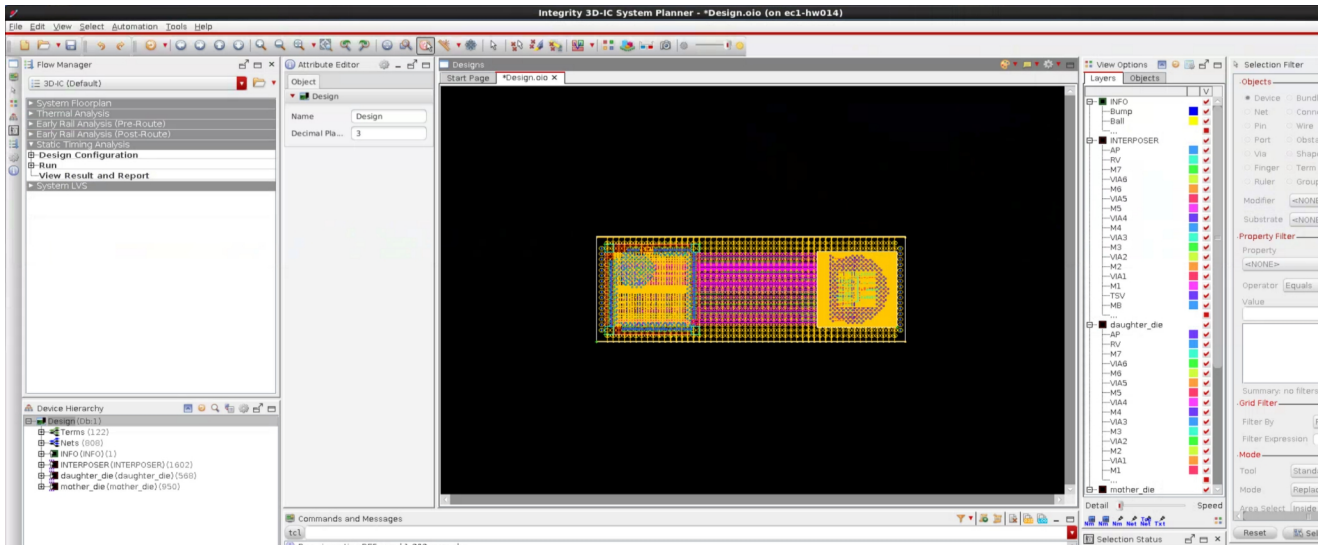
10. View the Early Rail Analysis results. Select *View Result and Report – View Rail Analysis Result*. This opens the Choose Device (ERA Result) dialog. Select a device and click*OK*. Integrity 3D-IC will invoke Voltus to show the results per die.



Static Timing Analysis

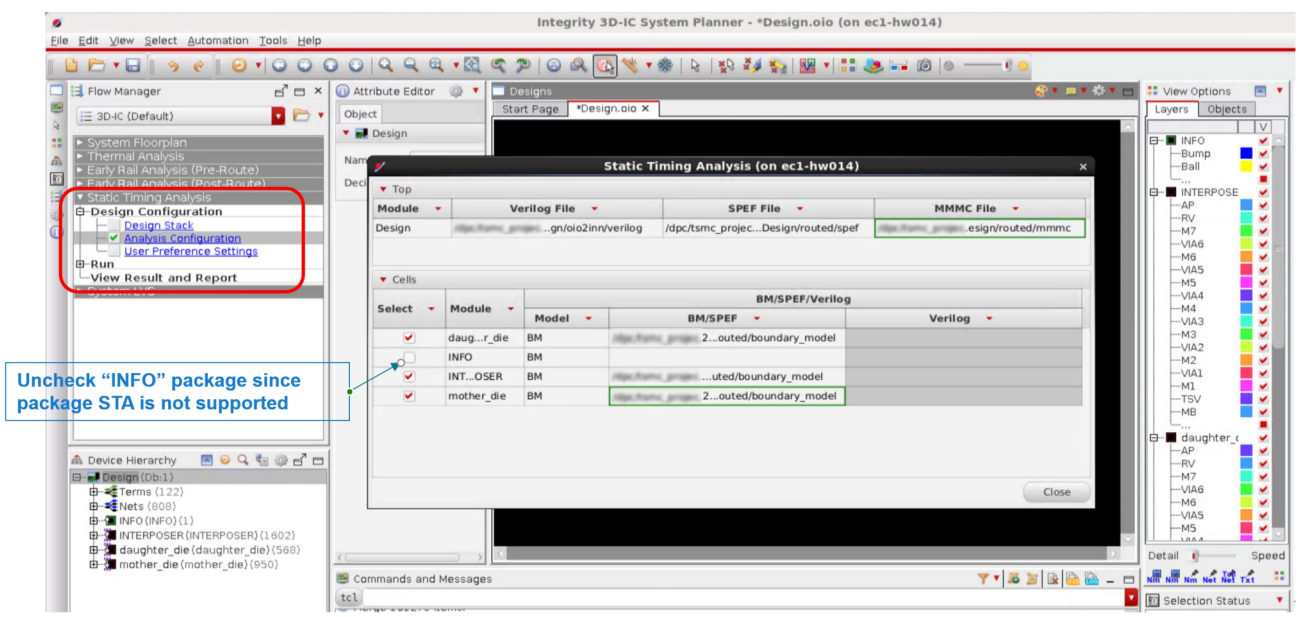
To configure and launch Tempus from the System Planner Flow Manager, follow the steps given below:

1. Load the multi-die 3D-IC design. In the Flow Manager, ensure *3D-IC (Default)* is selected.

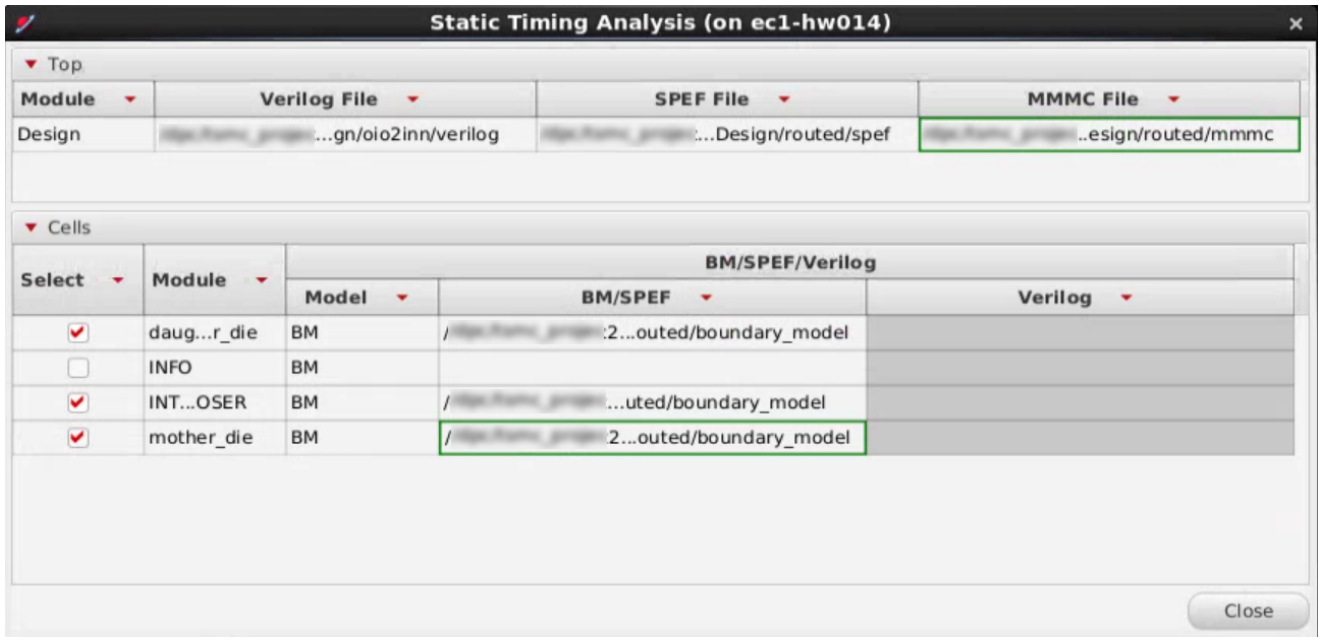


2. Select *Static Timing Analysis – Design Configuration – Analysis Configuration*. A Tempus run configure file is displayed in the Static Timing Analysis dialog, as shown below.

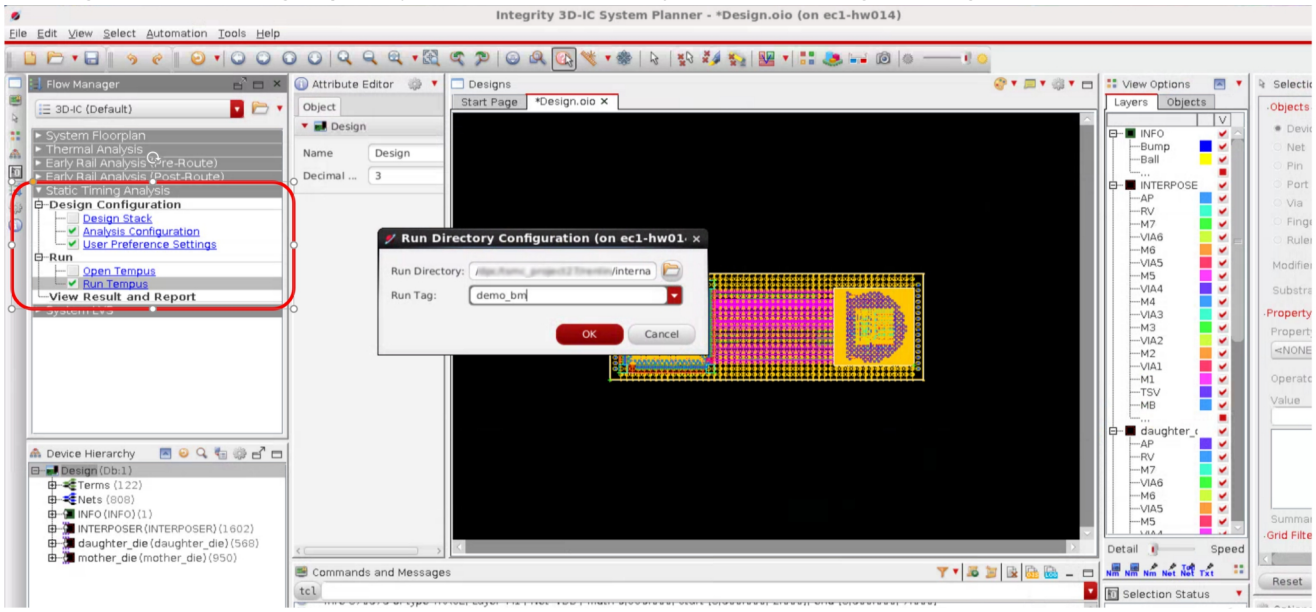
> ⚠ An existing Tempus settings tcl script can be loaded by selecting *Static Timing Analysis – Design Configuration – User Preference Settings*.



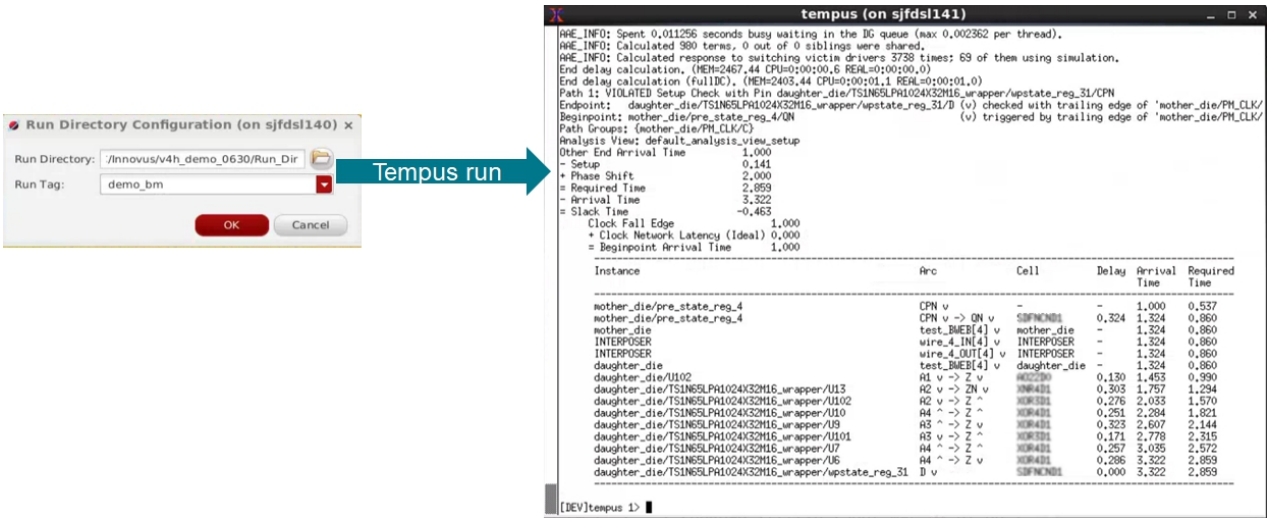
3. The design data is automatically based on the latest tag in the Integrity Hierarchical DB (iHDB). You can modify the data as needed. Both the Boundary Model (BM) and SPEF models are supported. Click *OK* after configuration.



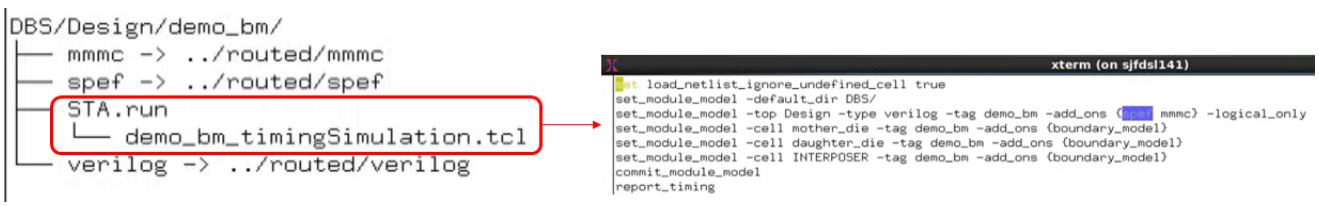
4. Now select *Static_Timing_Analysis – Run – Run Tempus*. In the resulting Run Directory Configuration dialog, specify the *Run Directory* and *Run Tag* settings.



5. Click *OK* after configuring the run directory. Tempus will be invoked in a new session.



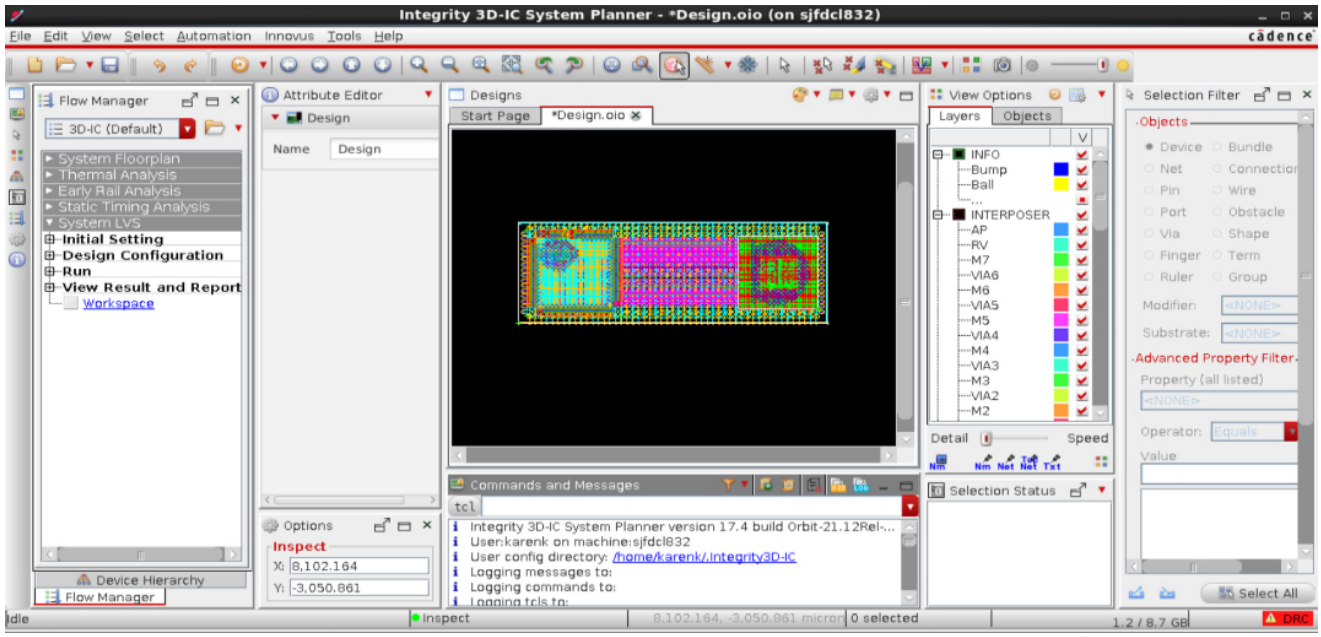
The Tempus script is generated and saved under a user-specified tag. You can, later on, modify the script and run Tempus standalone.



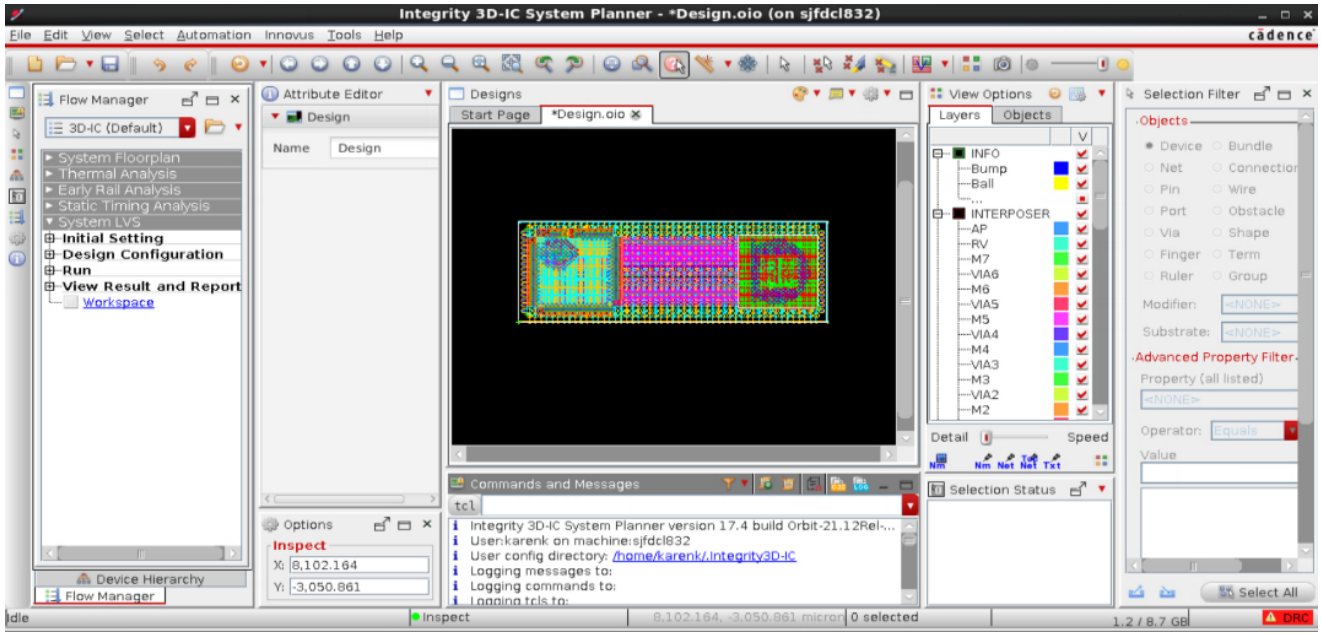
System LVS

To configure and launch a System Layout Versus Schematic (LVS) verification run from the Flow Manager, follow the steps given below:

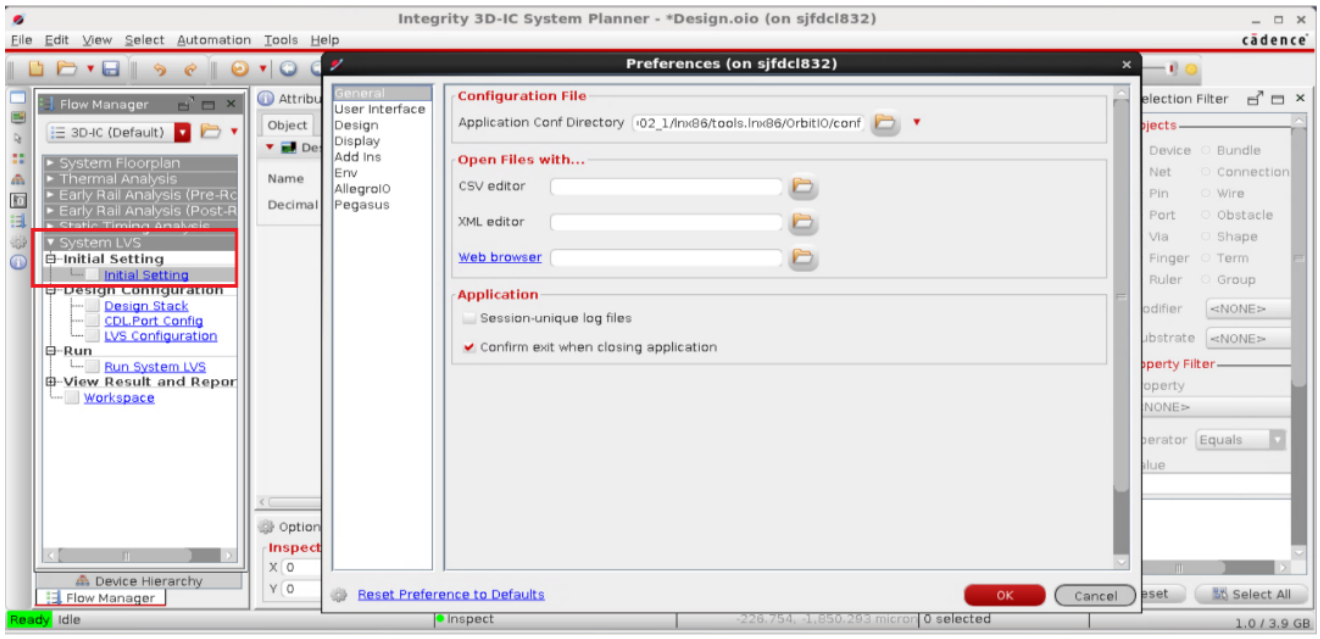
1. Load the multi-die 3D-IC design.



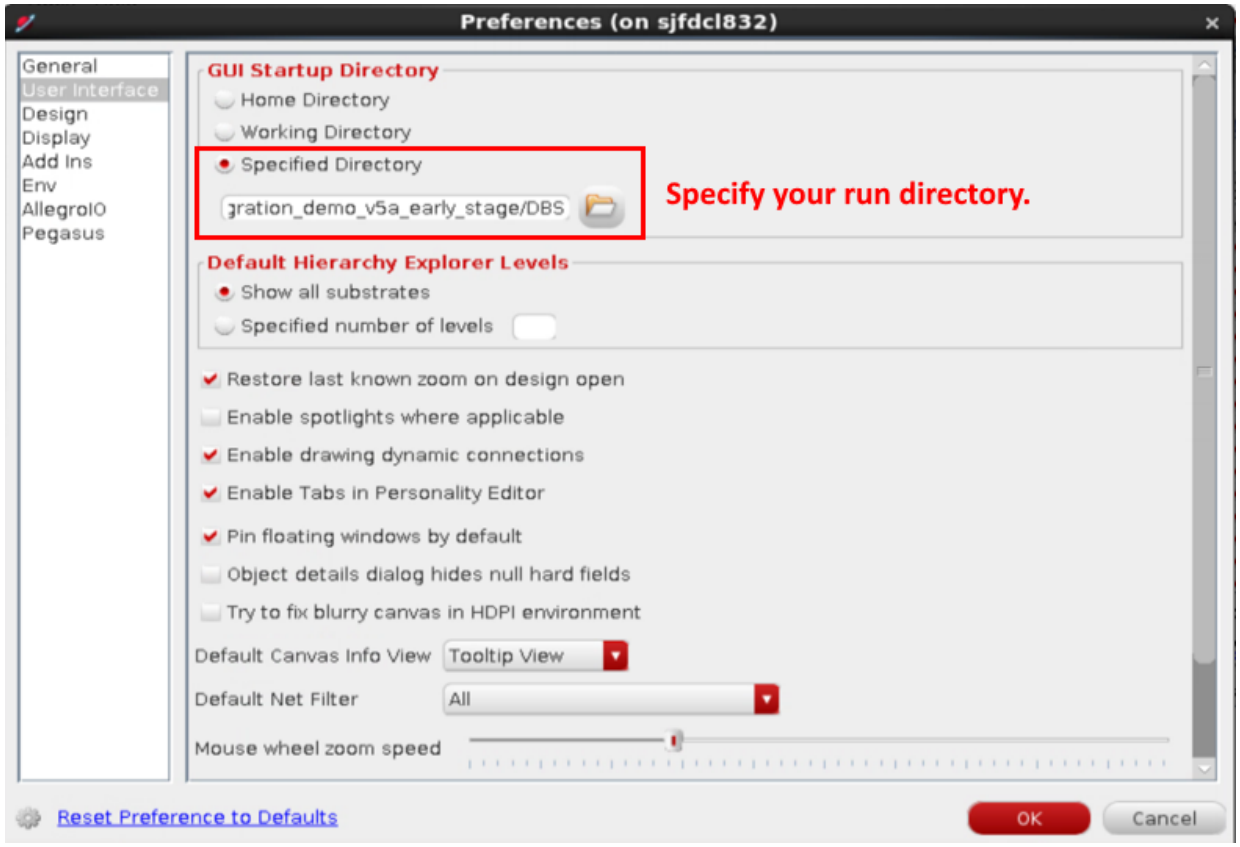
2. In the Flow Manager, ensure 3D-IC (Default) is selected.



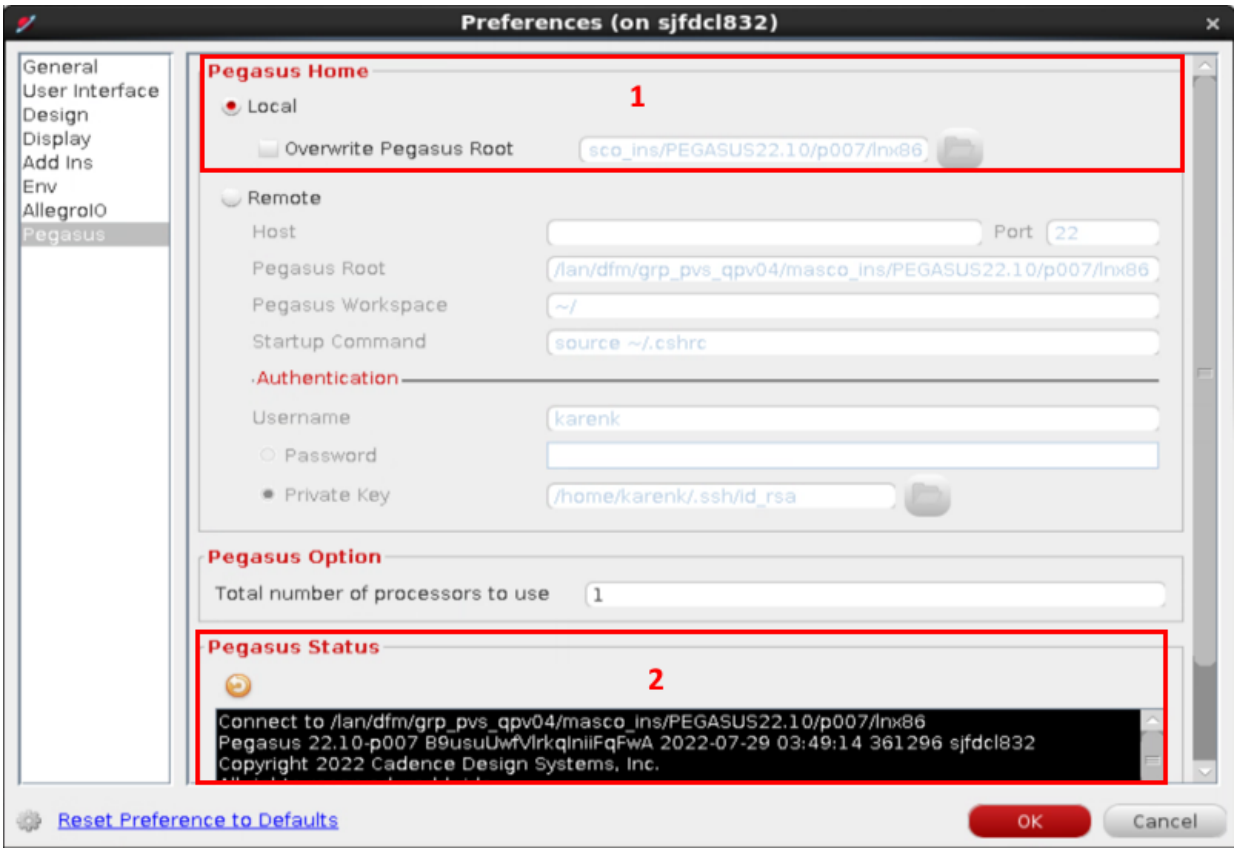
3. Select *System LVS – Initial Setting – Initial Setting*. A preference table is displayed in the Preferences dialog, as shown below.



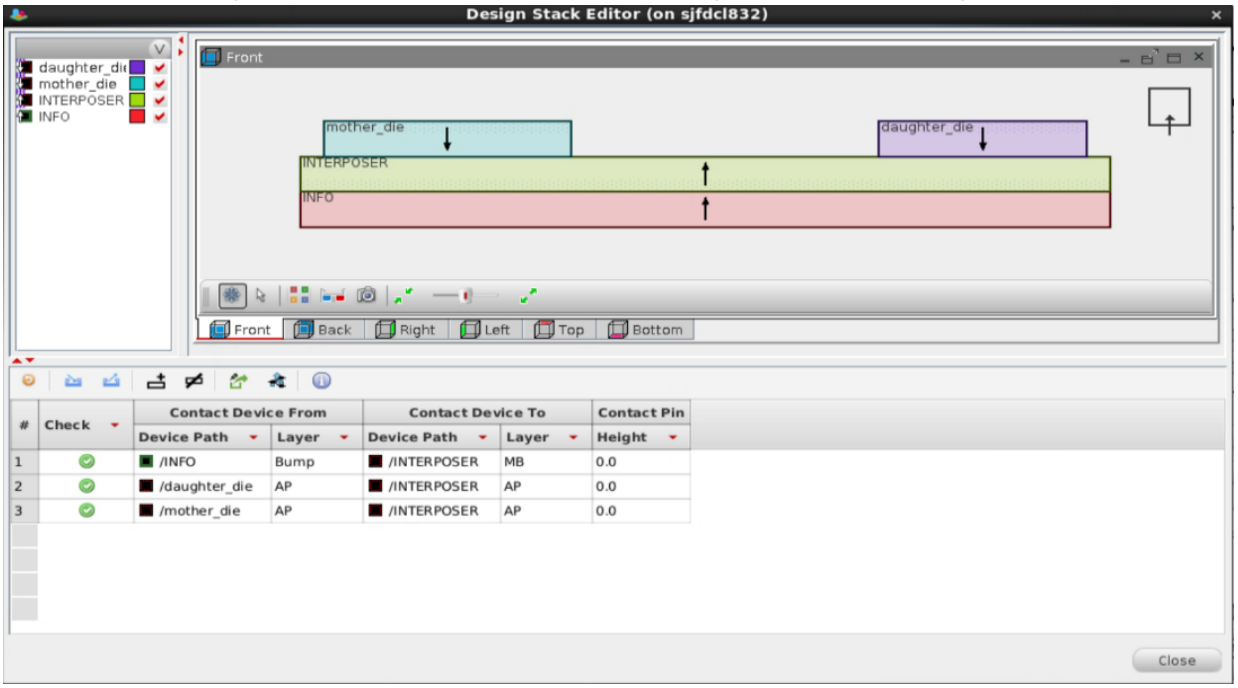
4. Click *User Interface* in the left pane of the dialog and set*Specified Directory* in the right pane.



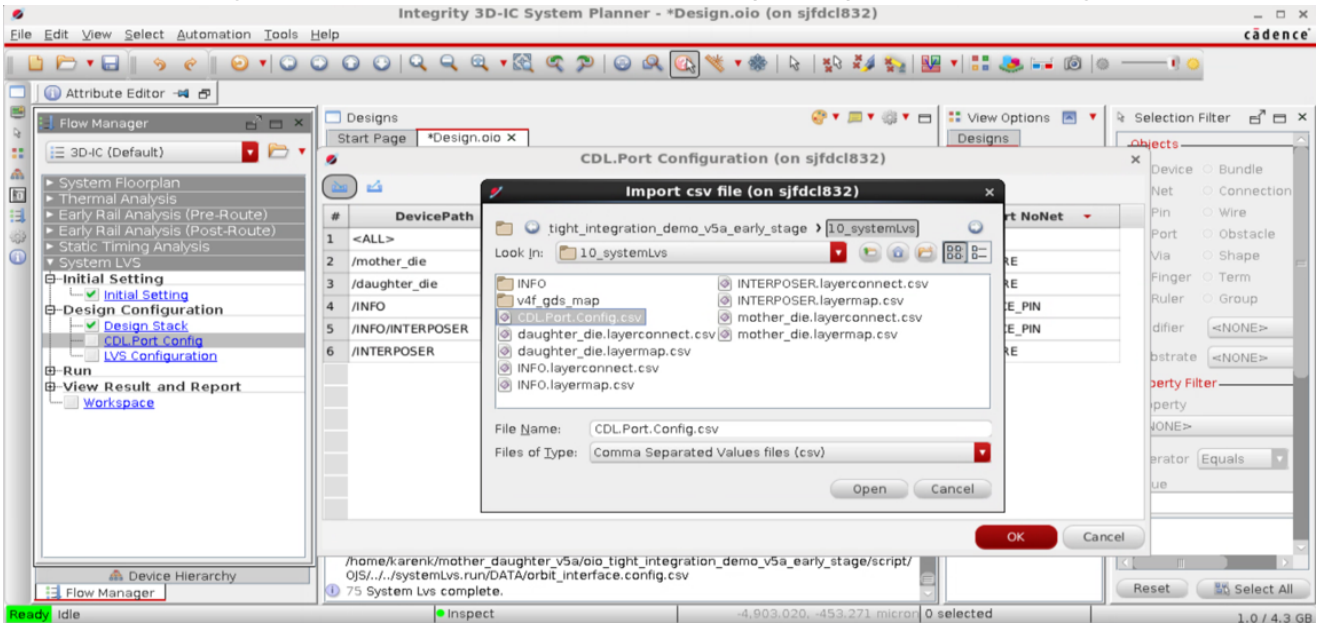
5. Next, click *Pegasus* in the left pane. In the right pane, set*Pegasus Home*, then click the *Refresh* icon in the *Pegasus Status* section and click *OK* to load the Pegasus environment.



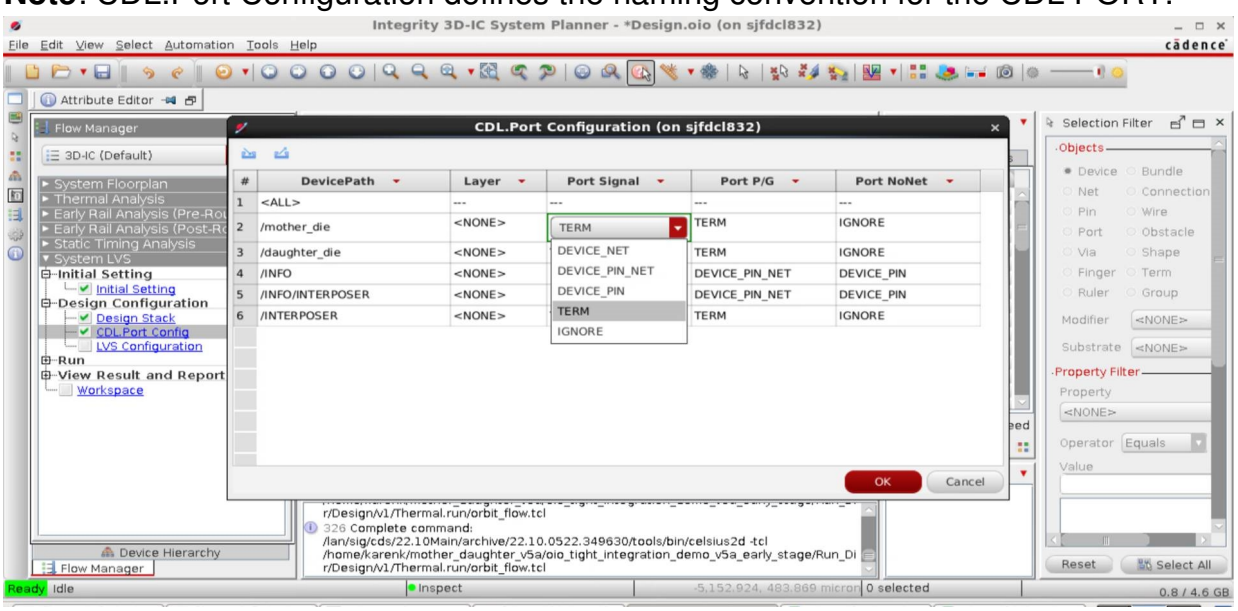
6. Now, in the Flow Manager, select *System LVS – Design Configuration – Design Stack* to review the Design Stack structure. After reviewing, close the dialog.



7. Next, select *System LVS – Design Configuration – CDL.Port Config*. This opens This opens the CDL.Port Configuration dialog, where you can define the port name with which to match the GDSII data. Here, you can either manually edit the configuration data in the table or click the *Import configuration from csv file* ( ) icon to import a predefined configuration file.



Note: CDL.Port Configuration defines the naming convention for the CDL PORT.



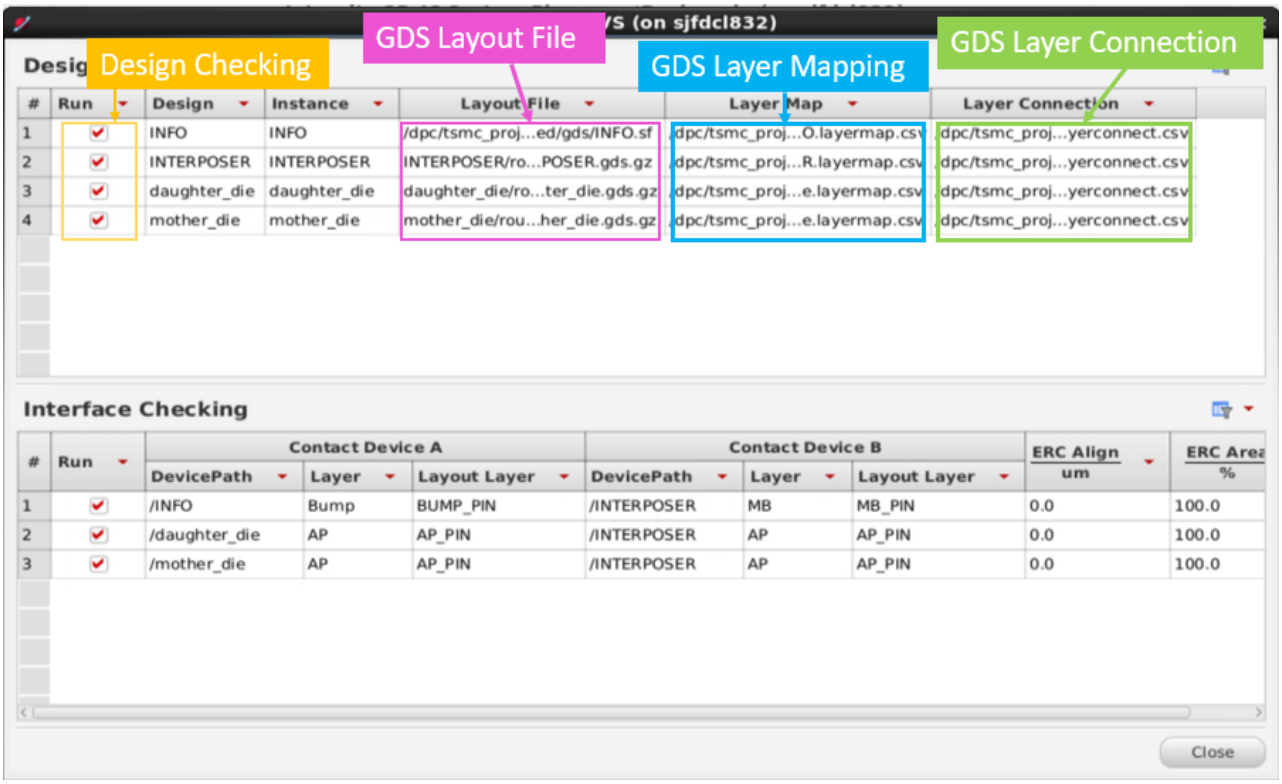
For each column of *Port Signal*, *Port P/G*, and *Port NoNet* in the CDL.Port Configuration table, you could choose:

- *IGNORE*: Port will be ignored in CDL netlist creation.
- *TERM*: Port name will be the *<TermName>* in CDL netlist creation.

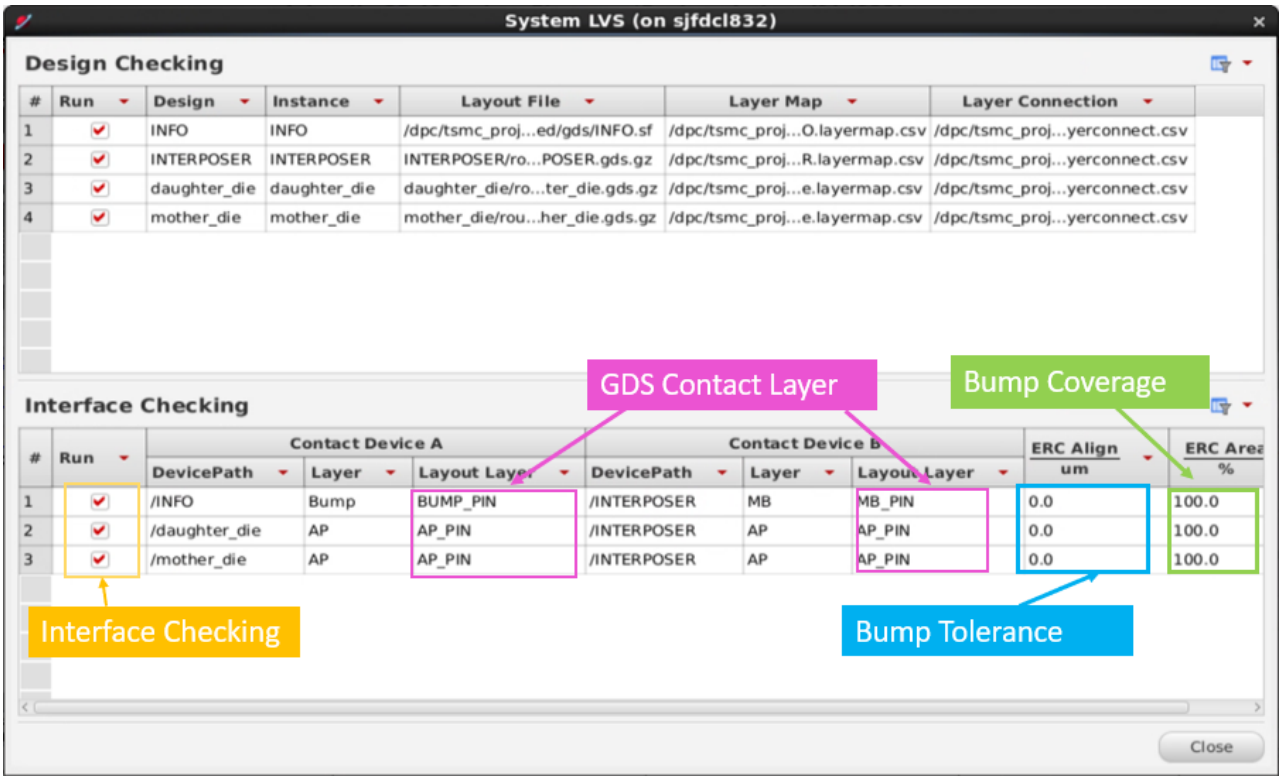
- *DEVICE_PIN*: Port name will be *<DeviceName>_<PinName>* in CDL netlist creation.
- *DEVICE_NET*: Port name will be *<DeviceName>_<NetName>* in CDL netlist creation.
- *DEVICE_PIN_NET*: Port name will be *<DeviceName>_<PinName>_<NetName>* in CDL netlist creation.

8. Next, in the Flow Manager, select *System LVS – Design Configuration – LVS Configuration* to set the LVS configuration. The System LVS dialog is displayed, as shown below.

   a. In the *Design Checking* section, select the *Run* checkboxes and specify the *Layout File*, *Layout Map* and *Layer Connection* settings for the design objects.



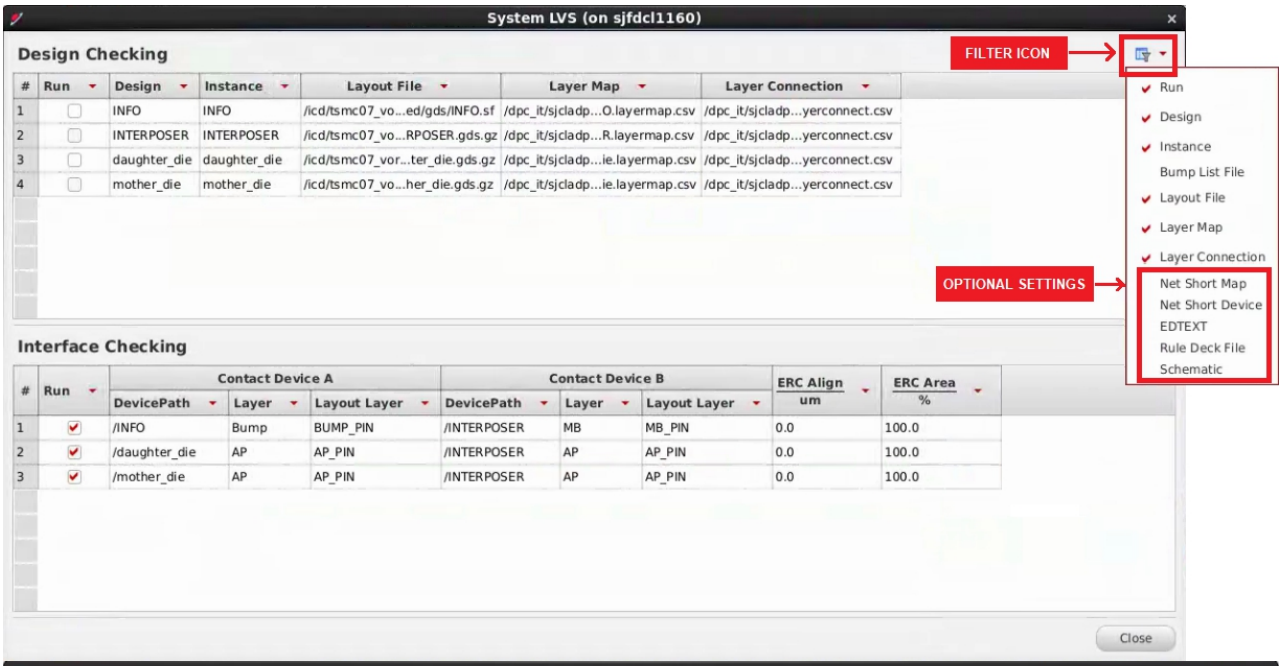
   b. In the *Interface Checking* section, select the *Run* checkboxes and specify the *Layout Layer* for *Contact Device A* and *Contact Device B* and *ERC* settings.

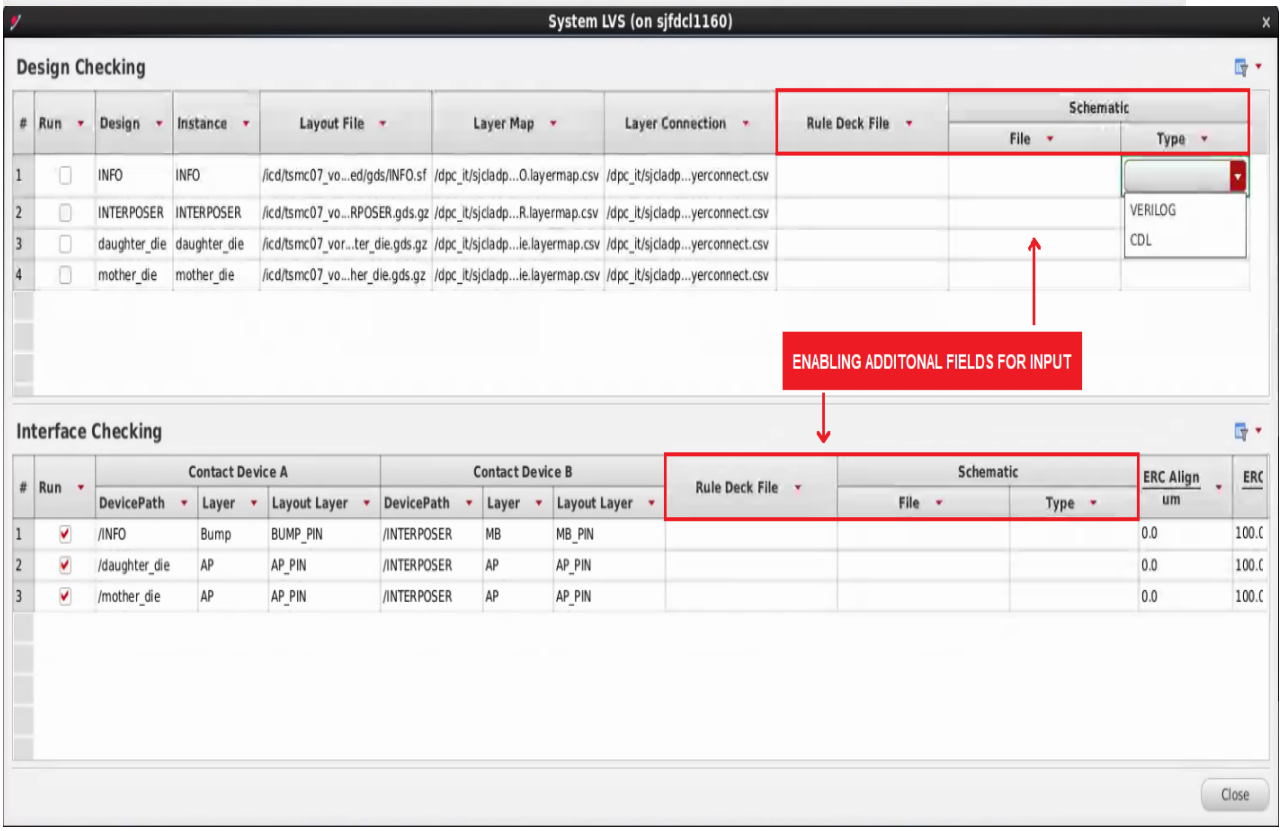


Here:

Layout Layer	Represents the connectivity between each substrate's BUMP layers (PIN Layers).
ERC Align	Specifies the tolerance. Note that: ▪ Tolerance (in um) represents the maximum difference that is allowed between the centers of connected polygons. ▪ Specifying a negative value will disable this rule from executing.
ERC Area	Represents the required minimum coverage percentage. Note that: ▪ This (percentage) represents the minimum overlap between the two pins required to PASS (at least one pin should have the minimum percentage of area covered). ▪ Specifying a negative value will disable this rule from executing.

Both the *Design Checking* and *Interface Checking* sections support additional input files. These can be enabled in the GUI by clicking on the icon displayed below and selecting the required option:



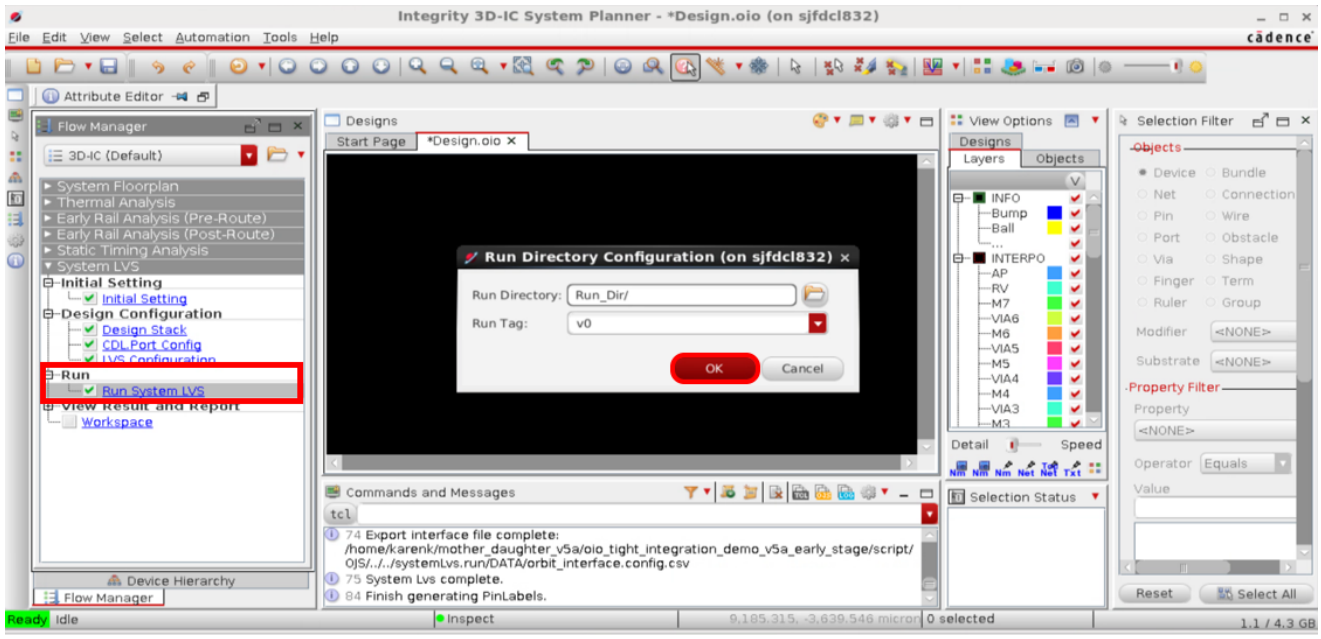


Some of the optional settings available are:

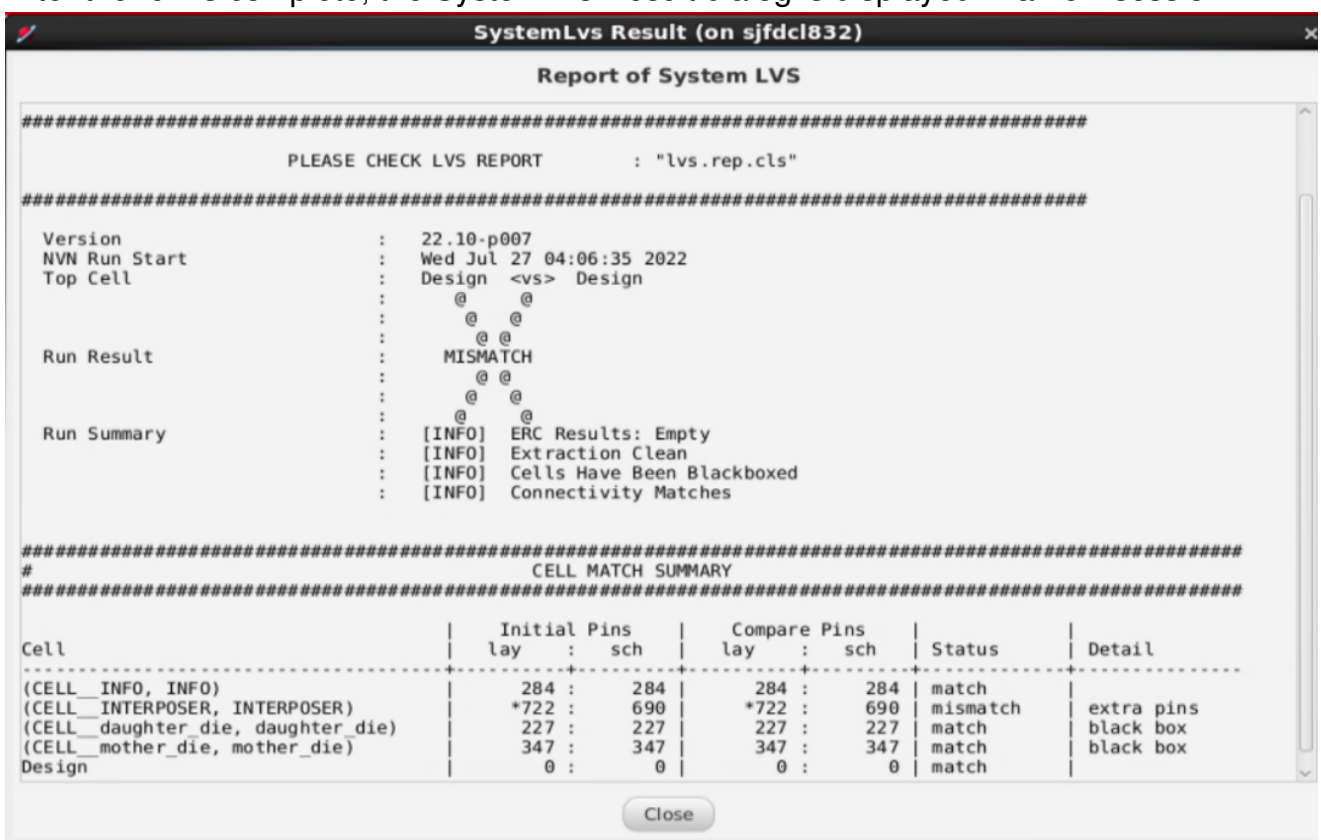
EDTEXT	Use this option to provide text information that is not physically present in the design layout.
Rule Deck File	Use this option to provide an external rule deck to launch LVS.
Schematic	Use this option to provide an external schematic to compare the result of the System LVS flow.

- If you specify only a schematic file, a regular System LVS run will be launched against the external schematic.
- If you specify both a schematic file and a rule deck, only the external rule deck is launched. There is no System LVS run.
- These fields support multiple files as input.
- Design-level schematic can be input via the *Design Checking* table and the top-level (system-level) schematic can be input via the *Interface Checking* table.
- If you have to run LVS via an existing rule deck on a single design level, specify both the rule deck and the schematic via the *Design Checking* table.
- If you have to run LVS via an existing rule deck on an interface between two or more designs, specify both the rule deck and the schematic via the *Interface Checking* table.
- Multiple entries could have same input so Pegasus will automatically sort the inputs to recognize a single run.

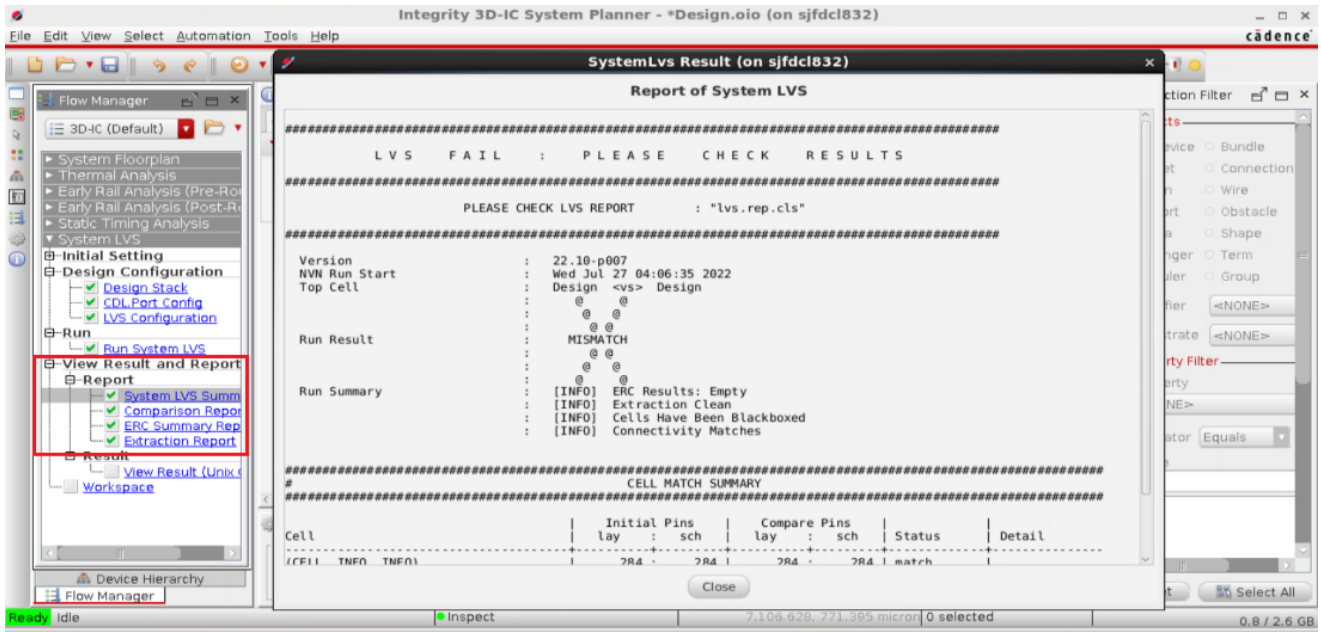
9. Next, in the Flow Manager, select *Run - Run System LVS* The RUN Directory Configuration dialog is displayed, as shown below. Set the *Run Directory* and *Run Tag* settings and then click the *OK* button to start the System LVS run.



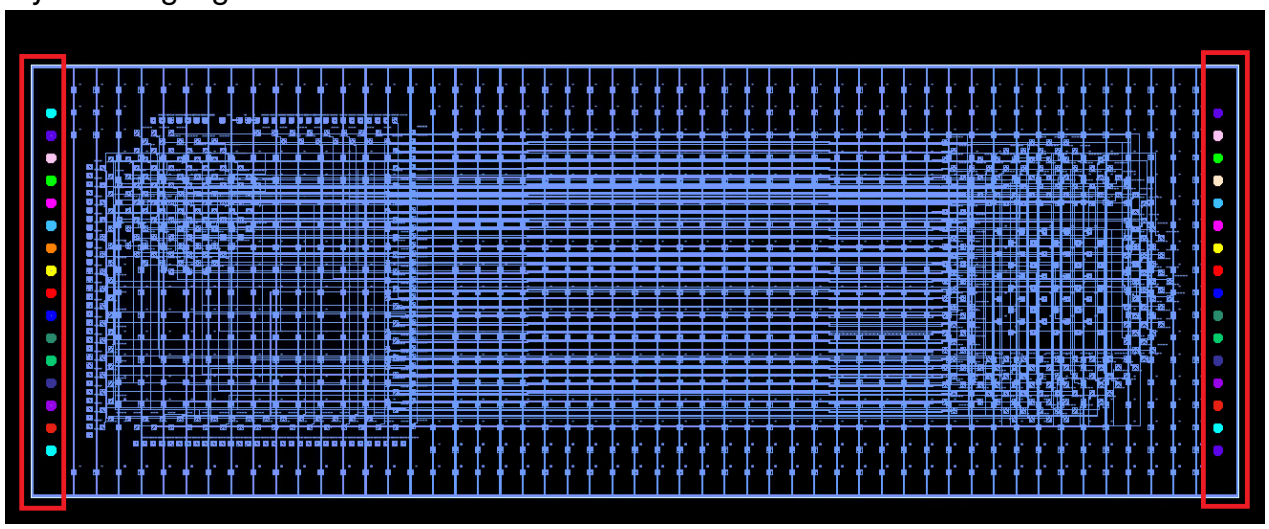
10. After the run is complete, the SystemLvs Result dialog is displayed in a new session.



11. Finally, in the Flow Manager, select *System LVS – View Result and Report – Report* to view the details of the System LVS run report.

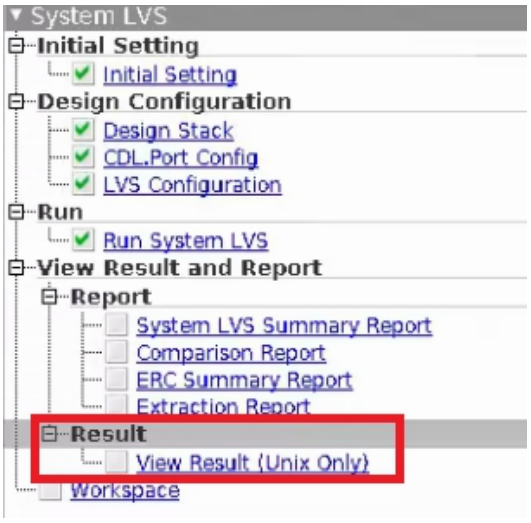


Notice that in this case, an extra pins mismatch in INTERPOSER on the 32 NC Bumps in layout is highlighted.

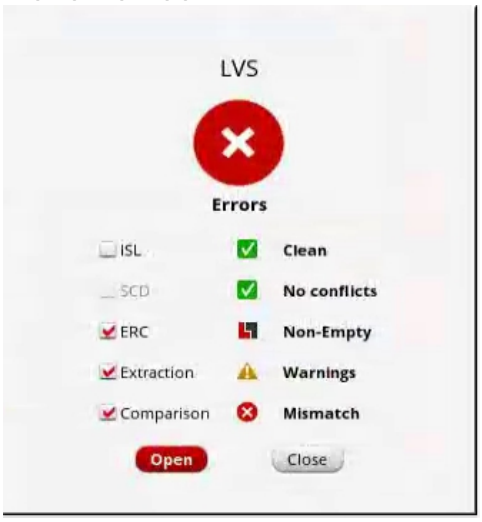


Alternatively, you can perform the following steps to view the run results:

a. Select the *View Result (Unix Only)* option to view the results in the Design Review GUI.

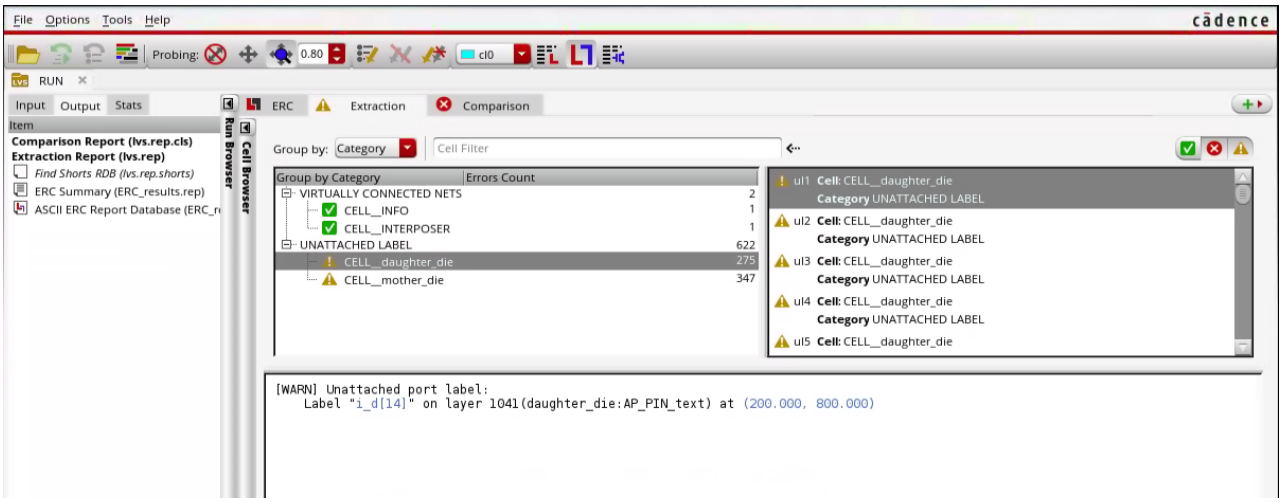


b. A pop-up appears as below:

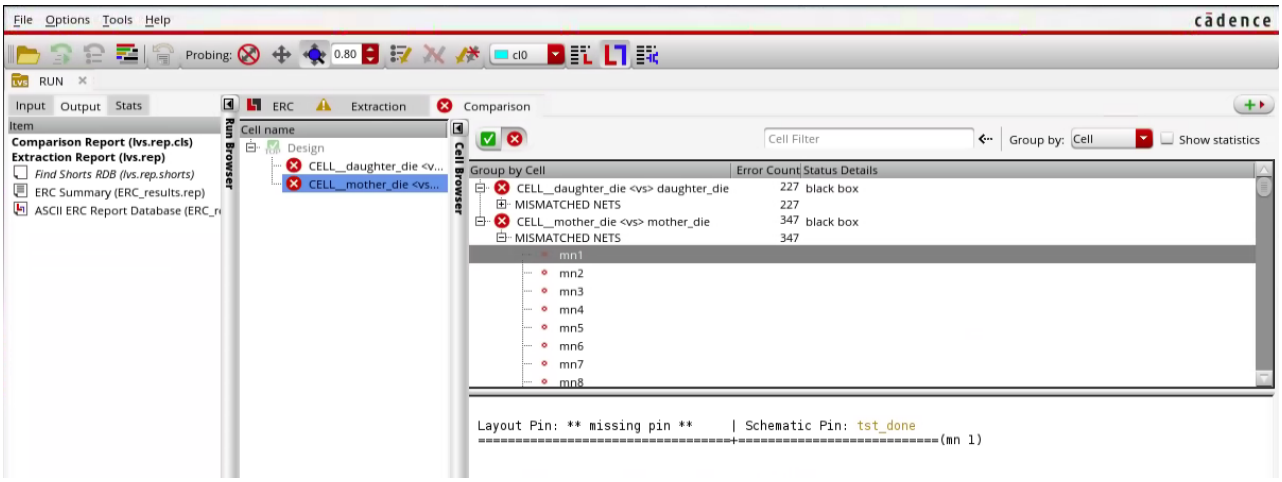


This shows the mismatches and errors that will be displayed by default in the Results Viewer. Select or deselect the relevant check boxes to view the desired results.

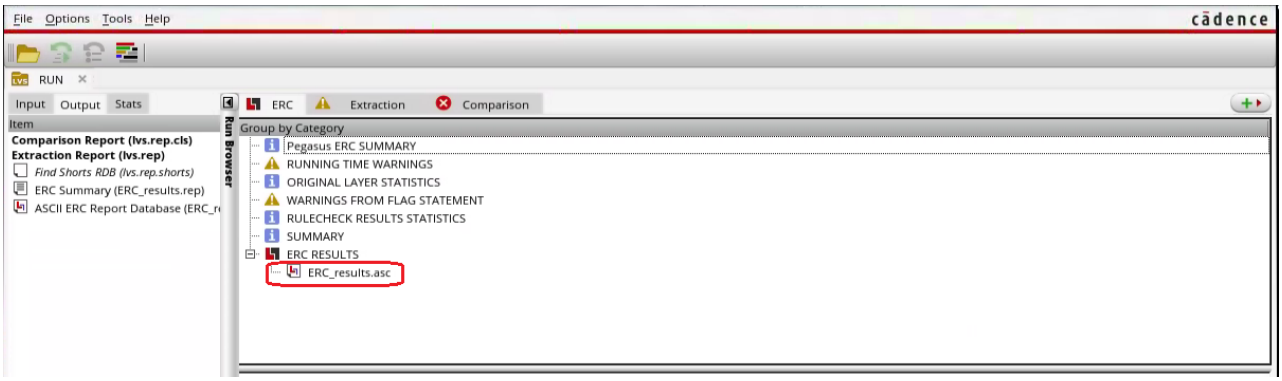
c. The images below show an example each of how failure is represented for *Extraction*, *Comparison,* and *ERC.*
Extraction Results:



Comparison Results:

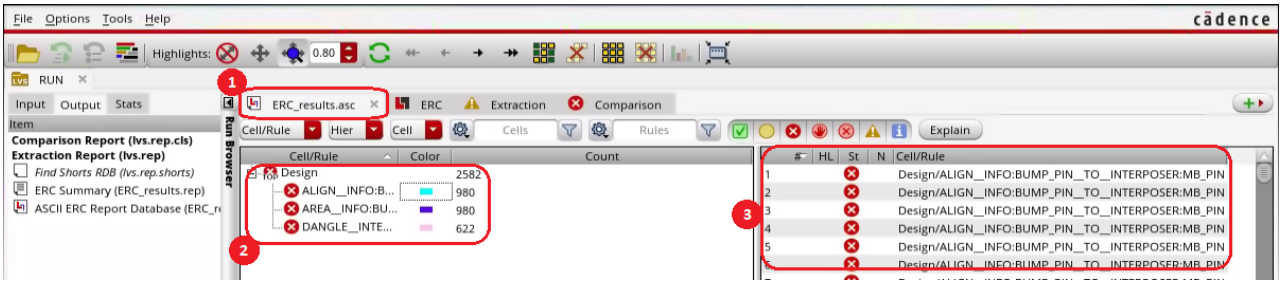


ERC Results:

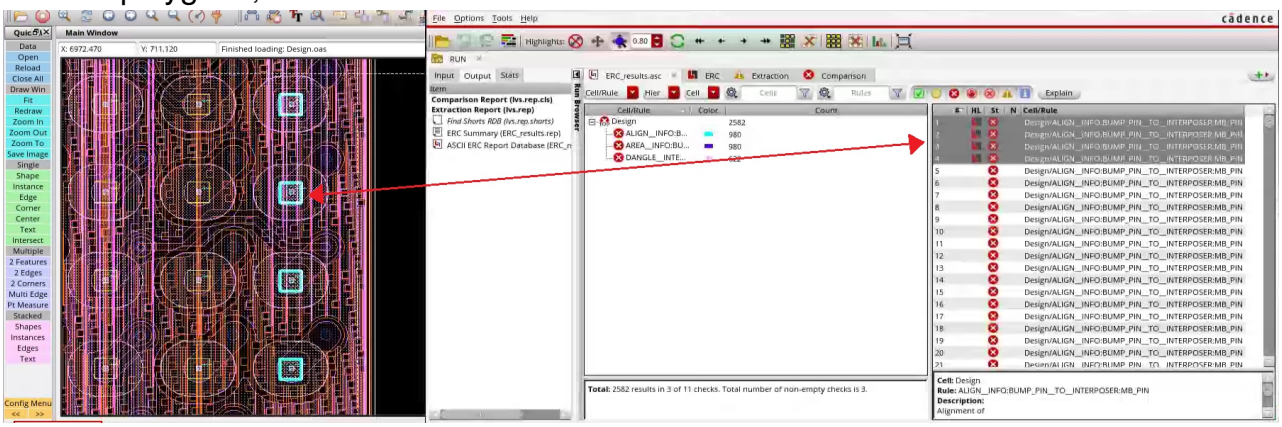


To view the ERC result details in a cross-probing form, double click the ASCII report highlighted above.

d. This will open a new tab *(#1 in image below)* in the Results Viewer with information about the ERC rules that are failing *(#2 in image)*.



e. Click any polygon information in the right *(#3 in the image above)* to view it in Design Review. To select multiple failing polygons, hold the *Ctrl* or *Shift* key while clicking the desired polygons, as shown below:

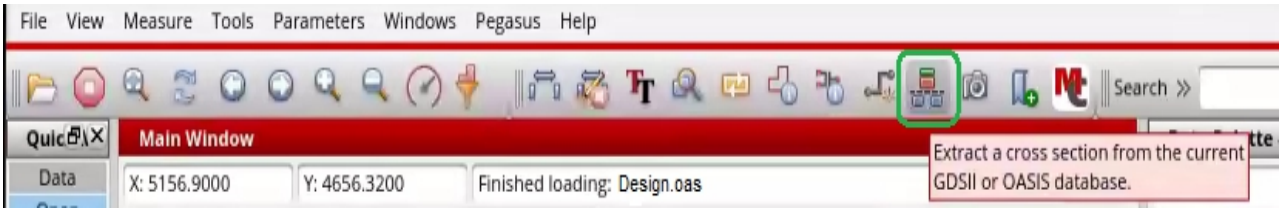


Using the Cross-Section Viewer

The cross-section viewer is is an additional functionality in Design Review to understand the 3D placement of layout layers. You can view the 3D structure in two ways.

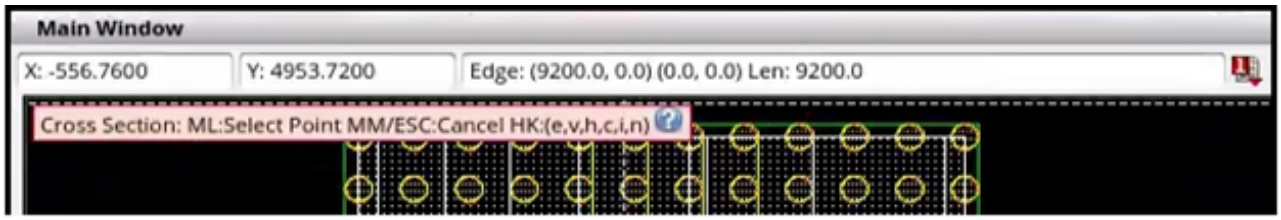
- Cross-section of a region: This allows you to select an area and view only the layers in that region. To do so:

  1. Select the icon highlighted on the toolbar below.



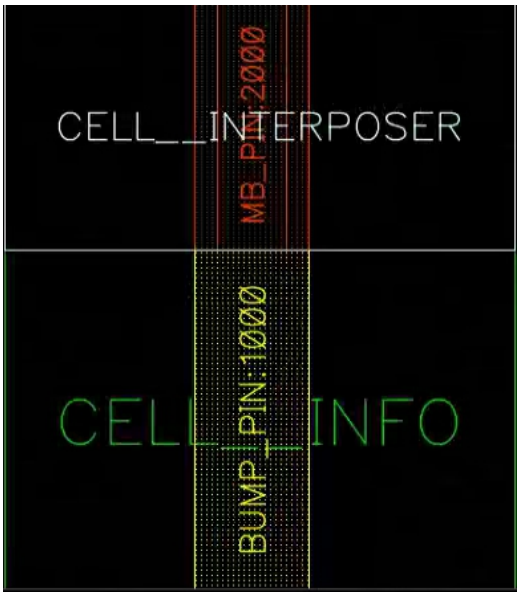
  2. Once the pink ribbon is displayed (as highlighted below), left-click and drag the mouse across the design to select the area for viewing the cross section.
     Note: The ribbon will always hint on the correct usage. Here, you can see ML (Mouse

Left) to select and MM (Middle Mouse/Scroll wheel) to cancel.



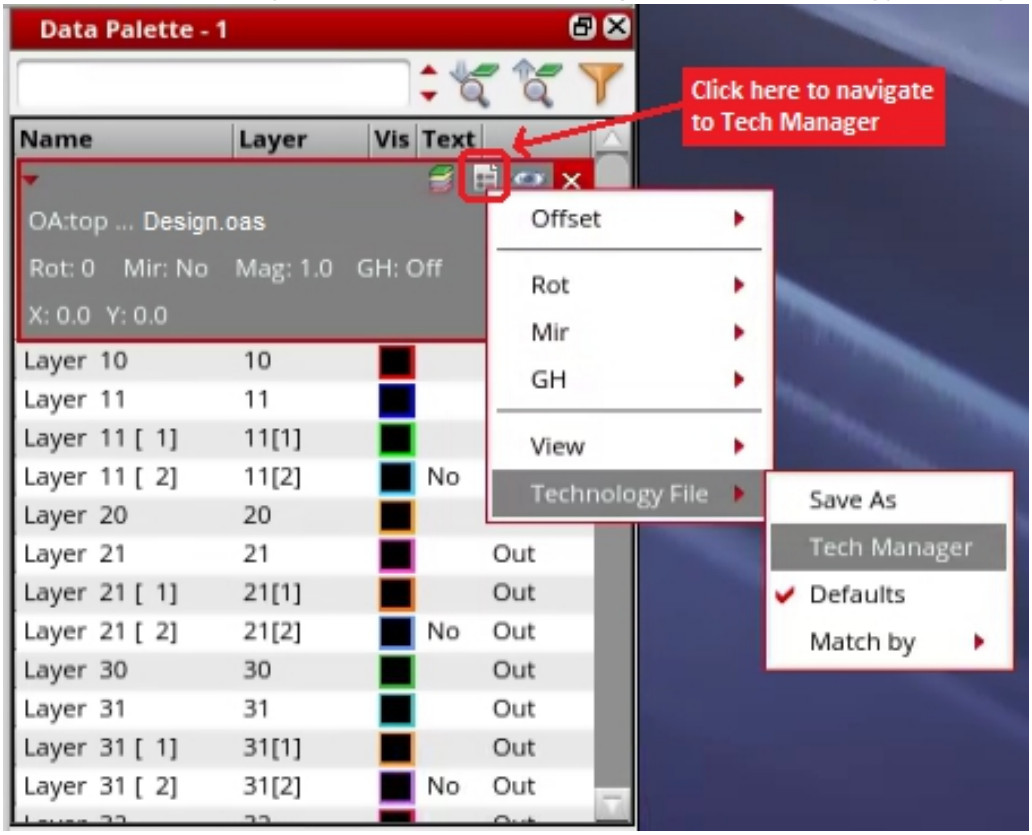
3. Once you select a region, a cross section will show the layout display, as shown below:

Zooming in further will show the names and polygons of all the layers in the selected region, as shown below.

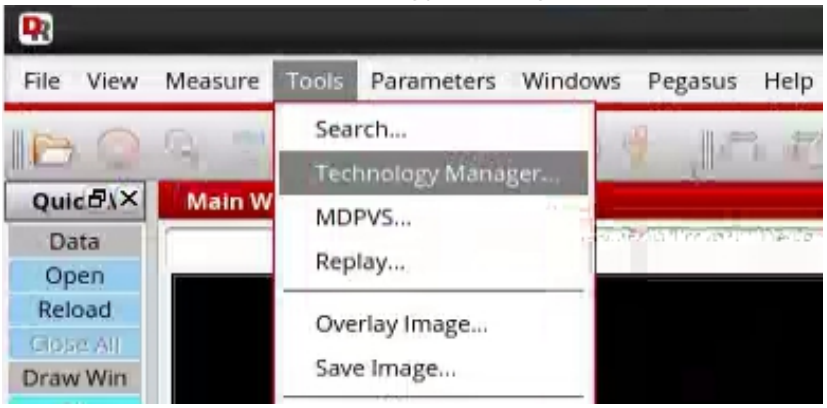


- Stacked layout (Cross-section overview): This allows you to view the overall stacking of layers in the design by height. To do so:

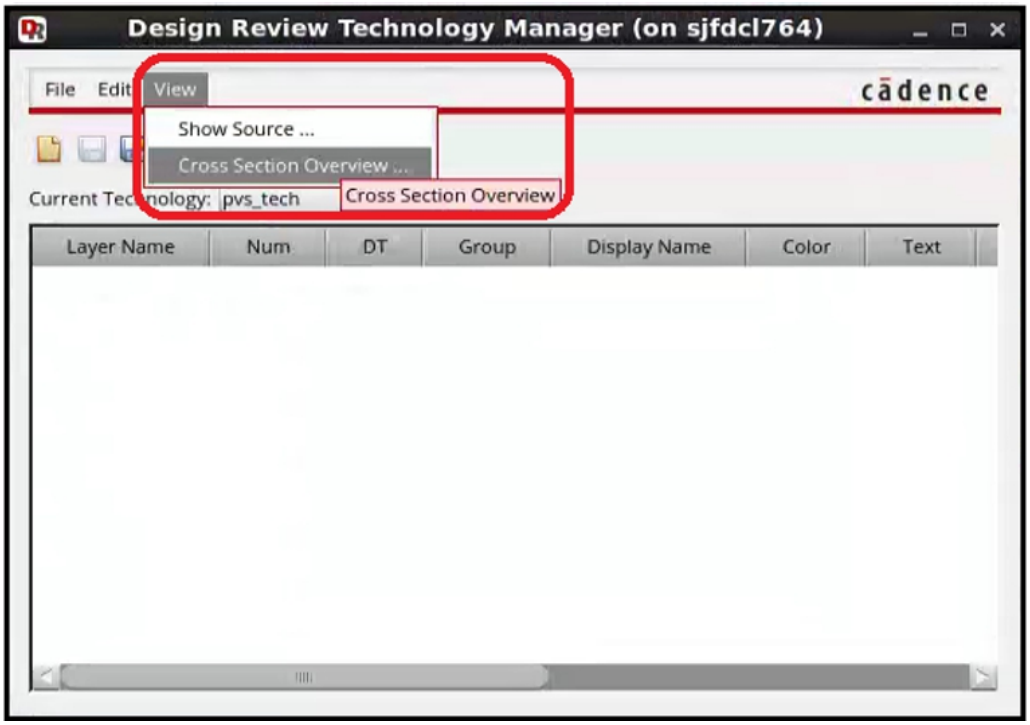
  1. Select *Tech Manager* as shown below to open the *Technology Manager* tool.



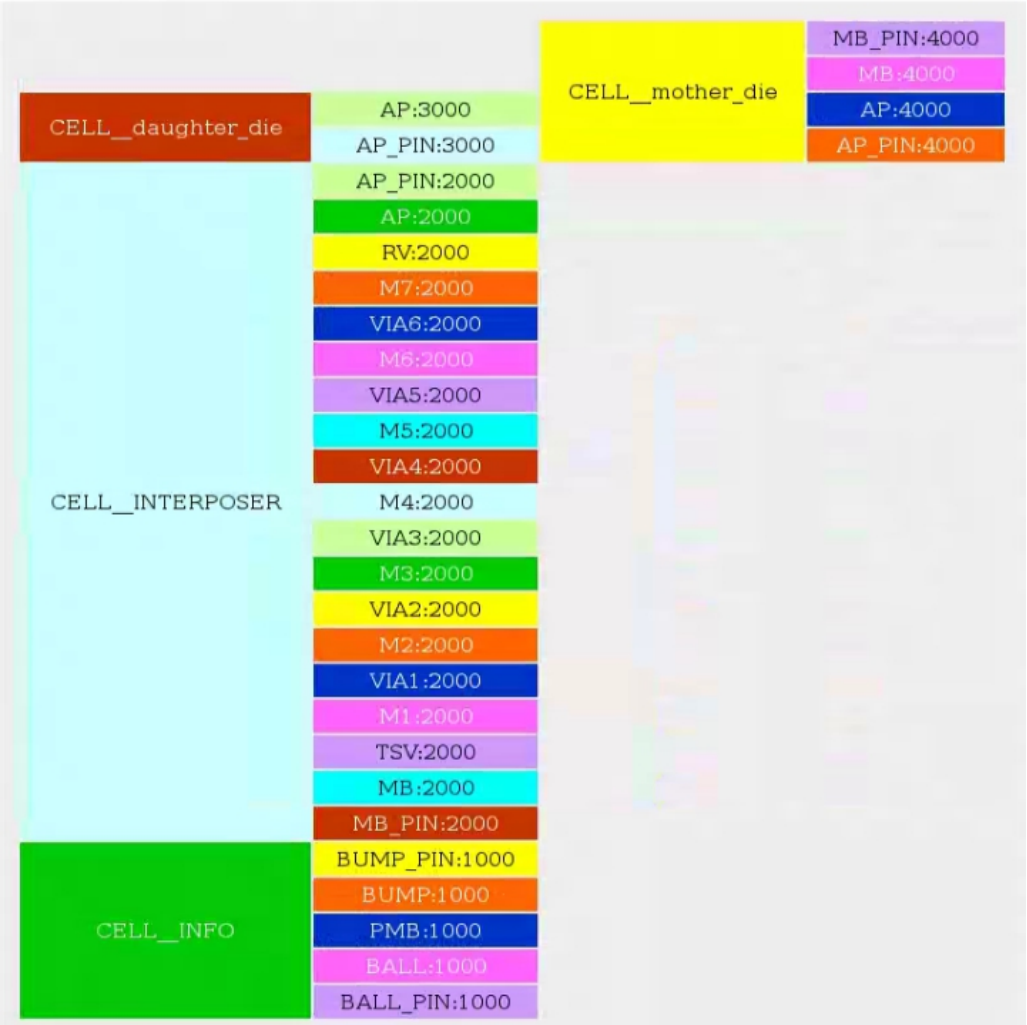
Or select *Tools* → *Technology Manager*



2. In the *Technology Manager* window, select the *Cross Section Overview* option from the *View* menu as shown below.



This will open the overall design layer stacking.



As you can see above, this shows both the cell depths and the layer depths of the overall design.