cādence®

# Virtuoso Multi-Technology User Guide

**Product Version IC23.1**
**November 2023**

© 2023 Cadence Design Systems, Inc.

Printed in the United States of America.

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

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

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

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

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

**Disclaimer:** Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

# Contents

1
Introduction to Virtuoso Multi-Technology Solution . . . . . . . . . . . 13
Virtuoso Multi-Technology Solution Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
License Requirements for Virtuoso Multi-Technology Solution . . . . . . . . . . . . . . . . . . 16

2
Unified Libraries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
Views in the Unified Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
Schematic Symbols . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
Component Footprint Views . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
Component Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
Variant Definitions in CSV Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
Component Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Padstacks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Symbol Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Components and Symbols . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
Types of Libraries for Creating Unified Libraries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
Exported Die Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
BGA Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
Tline Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
pkgLib Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
Passive Component Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
CSV Import Library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
Unified Library Validation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
Converting Allegro Libraries to Unified Libraries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
Netlisting Setup for Unified Library Components . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30


© 2023 Cadence Design Systems, Inc.

3
Import Libraries and ICs . . . . . 33
Create Technology File . . . . . 33
Import Technology File . . . . . 34
Update Technology File . . . . . 36
Constructs in the Technology File . . . . . 37
Library Import . . . . . 40
Die/TILP Instantiation . . . . . 43
Die Export Preparation . . . . . 45
Exporting Dies . . . . . 47
Edit Die Layout and Abstracts . . . . . 51
Creating and Verifying Integrity 3D-IC Compatible Die Abstracts . . . . . 53
Die Audit . . . . . 58
Creating TILPs . . . . . 61
Alternate Footprints in TILPs . . . . . 62
TILP Versions . . . . . 63
Replacing Pads . . . . . 68

4
Package Schematic Creation . . . . . 71
Instantiating SMD Instances . . . . . 72
Instantiating BGA Instances . . . . . 74
Instantiating IC Instances . . . . . 75
Creating TLines Instances . . . . . 76
Instantiating TLine Instances . . . . . 78
Creating and Saving Connectivity Information in Package Schematic . . . . . 79

5
Package Layout Creation . . . . . 81
Generate from Source . . . . . 81
Dies in Virtuoso Multi-Technology Solution . . . . . 82
Wirebonded Dies . . . . . 82
Flip Chip Dies . . . . . 83
Guides in Wirebonded Dies . . . . . 84


© 2023 Cadence Design Systems, Inc.

Creating a Guide . . . . . 86
Editing a Guide . . . . . 89
Bond Wires and Bond Fingers Creation . . . . . 91
Creating Bond Wires and Bond Fingers . . . . . 93
Creating Bond Finger Definitions . . . . . 97
Moving Bond Wires and Bond Fingers . . . . . 98
Updating the Finger Attach Point . . . . . 101
Setting a Bond Wire Profile . . . . . 102
Flip Chip Parameters . . . . . 104
Types of Bump Attachments . . . . . 107
Configuring a Module Stack . . . . . 111
Thermal Shrink Factor . . . . . 112
Creating a Curved Polygon . . . . . 120
Creating a Curved Rectangle . . . . . 121
Interactive Routing in Virtuoso RF Solution . . . . . 122
    Managing Curved Shapes in Wire Editor . . . . . 122
    Push-and-Shove Feature . . . . . 122
Fillet Creation Between Curved Path and Other Objects . . . . . 123
Void Shapes . . . . . 125
Void Shape Generation . . . . . 128
    Create Dynamic Shapes . . . . . 128
    Create Signal Shapes . . . . . 129
    Generate Void Shapes . . . . . 129
Convert Selected Dynamic Shapes . . . . . 130
    Smooth and Trim Void Shapes . . . . . 130
Package Constraints Supported by the Void Generator . . . . . 132
Dynamic Shape Priority . . . . . 135
Extracting Connectivity Information from Package Layout . . . . . 137

6
Interoperability with SiP . . . . . 139
Export Package Layout . . . . . 142
Complete the Package Layout . . . . . 143
    Manual and Automatic Placement . . . . . 143
    Placement Tasks . . . . . 143


© 2023 Cadence Design Systems, Inc.

Routing Tasks . . . . . 144
Importing the Package Layout . . . . . 145
Rebinding the Layout to a Schematic . . . . . 147

7
Verify the Package . . . . . 149
Cross-Fabric Checks Run . . . . . 149
Performing Cross-Fabric Checks . . . . . 150
Checking Layout Against Schematic . . . . . 152
Extracting the Connectivity . . . . . 153
Performing DRD Checks . . . . . 155
Supported DRD Constraints and Checks . . . . . 156

8
Edit-in-Concert . . . . . 159
Update Binding Information . . . . . 160
Viewing Die Instance Annotations . . . . . 161
Key Edit-in-Concert Views . . . . . 163
Launch Edit-in-Concert Mode . . . . . 165
Modify in Edit-in-Concert Mode . . . . . 170
Movement of Dies in Edit-in-Concert Mode . . . . . 171
Movement of Die Instances in Edit-in-Concert Mode . . . . . 174
Placement Status of IOs . . . . . 176
Changes to TILP Parameters in Edit-in-Concert mode . . . . . 178
Change from Package view to Layout View . . . . . 180
Net Tracing in Editing-In-Concert Mode . . . . . 181
Probing a Design in Edit-in-Concert Mode . . . . . 184
Checking and Fixing IO Pad Locations . . . . . 189
Running the LVA Checker . . . . . 191
Running the LVA Fixer . . . . . 193
IO Pad Connectivity Mismatch Fixes . . . . . 195
IO Pad Number Mismatch Fixes . . . . . 198
Exiting Edit-in-Concert Mode . . . . . 200
View-in-Concert Mode . . . . . 200


© 2023 Cadence Design Systems, Inc.

9
Stacked Modules Management . . . . . 203
Stacked Modules . . . . . 204
Benefits of Implementing Stacked Modules . . . . . 205
Components of a Stacked Module . . . . . 206
Stacked Module Assemblies . . . . . 208
Configuring a Stack . . . . . 210
Die Operations . . . . . 212
Creating Bumps and TSVs . . . . . 213
Assigning Connectivity between Bumps . . . . . 218
Unassigning Bump Connectivity . . . . . 220
Deleting Unassigned Bumps . . . . . 221
Moving Pins to Bumps . . . . . 222
Updating Bumps to the Abstract View . . . . . 223
Saving Bumps to File . . . . . 224
Creating Bumps from File . . . . . 225
Inter-Die Operations . . . . . 226
Propagating Bumps . . . . . 227
Fixing Bump Alignment Violations . . . . . 230

10
SiP Layout Creation . . . . . 235
Creating a SiP Layout from a Package Schematic . . . . . 235
Generating a SiP Layout from a Source Schematic . . . . . 240
Checking Against Source Schematic . . . . . 242
Creating an Extracted View . . . . . 246

11
Schematic Creation from SiP File . . . . . 249
Creating a Schematic Layout from a SiP File . . . . . 249


© 2023 Cadence Design Systems, Inc.

12
Assisted Import and Export . . . . . 253
Updating a Virtuoso Layout From a SiP File . . . . . 253
Virtuoso Layout to SiP with Assisted Export . . . . . 256
Initial SiP File to Virtuoso RF Solution Database . . . . . 257
Cell Replacement . . . . . 259

13
SiP DRC Checker . . . . . 263
Performing DRC Checks in Virtuoso RF Solution . . . . . 264

A
Virtuoso Multi-Technology Forms . . . . . 269
Allegro Design Layout Importer Form . . . . . 271
Annotate From Extracted View Form . . . . . 272
Cell Replacement Form . . . . . 273
Cell Replacement Dashboard Form . . . . . 275
Convert Allegro Libraries to Unified Libraries Form . . . . . 276
Create Extracted View Form . . . . . 277
Create MultiTech Schematic Form . . . . . 278
Export Design Update Form . . . . . 279
Pad Stack Replacement Form . . . . . 280
SiP DRC Check Form . . . . . 281
Virtuoso Multi Technology Enablement Form . . . . . 282
XOR SiP against OA Form . . . . . 284
Add Bond Finger Definition Form . . . . . 286
Add Layer Form . . . . . 286
Batch Checker Form . . . . . 287
Bind Layout Form . . . . . 287
Bump and Ball Editor Form . . . . . 288
Bump Connectivity Assignment Form . . . . . 289
Bump Propagate Form . . . . . 290
Configure Module Stack Form . . . . . 291
Create Bond Wire Form . . . . . 291


© 2023 Cadence Design Systems, Inc.

Create Bump and TSV Form . . . 293
Create Guides Form . . . 294
Create TILP Form . . . 294
Cross Fabric Check Violation Summary and Navigation Form . . . 295
Die Instance Annotations Form . . . 296
Edit Instance Properties Form (Die/Package TILP Parameters) . . . 296
Export Bump Info Form . . . 300
Export Die Form . . . 300
General Settings . . . 301
Advanced Settings . . . 302
Pin Numbering . . . 304
Area Transfer . . . 305
Import Bump Info Form . . . 307
Import Die Text File Form . . . 307
Layer Stack Editor Form . . . 308
Set Bond Wire Profile Form . . . 309
Virtuoso RF Compliance Audit Form . . . 310
Virtuoso Multi Technology Options Form . . . 311

B
Virtuoso Multi-Technology Environment Variables . . . 313
optionalPartData . . . 316
vsdpSparamCSVModelNameField . . . 317
vsdpSpiceCSVModelNameField . . . 318
areaTransferFile . . . 319
bondFingerAlignment . . . 320
bondFingerProfile . . . 321
bumpCenterMismatchCheck . . . 322
casDieLayoutVsAbstract . . . 323
cdfParamPromptLineNumber . . . 324
checkerPrecisionFactor . . . 325
createNoConn . . . 326
createWireFingerOrBoth . . . 327
csvFile . . . 328
customizePin . . . 329


© 2023 Cadence Design Systems, Inc.

deleteViewsBeforeExport . . . . . 330
dieTemplateFile . . . . . 331
distributeObjsOnGuide . . . . . 332
drdEditApkDrcBlockageOverlapCheck . . . . . 333
drdEditApkDrcComponentOverlap . . . . . 334
drdEditApkDrcShortCheck . . . . . 335
exportNoBumpTerm . . . . . 336
honorConstraints . . . . . 337
icSymbolCheck . . . . . 338
ignoreColumnNumbers . . . . . 339
ignoreLineNumbers . . . . . 340
instTermLabelCheck . . . . . 341
ioCheck . . . . . 342
layoutConnectivityFile . . . . . 343
layoutConnectivityPinNamesChkBox . . . . . 344
libraryCheck . . . . . 345
noConnCell . . . . . 346
otherChecks . . . . . 347
outputMode . . . . . 348
paramMap . . . . . 349
paramNameLineNumber . . . . . 350
partNameToImport . . . . . 351
pinNumberFile . . . . . 352
pinOrdering . . . . . 353
pinOrderingCustom . . . . . 354
sameTechnologyAbstract . . . . . 355
schematicConnectivityFile . . . . . 356
schematicConnectivityMode . . . . . 357
segSnapMode . . . . . 358
shortPinCell . . . . . 359
shortPinLib . . . . . 360
snapEndPoint . . . . . 361
snapMode . . . . . 362
termsCheck . . . . . 363
transferArea . . . . . 364
voidGeneratorTrimUnconnectedShapes . . . . . 365



© 2023 Cadence Design Systems, Inc.

C
Virtuoso Multi-Technology Solution SKILL Functions . . . . . . . 367
vmtCompareSipToOa . . . . 368
vmtcsvCreateComponentCellViewsFromCsv . . . . 371
vmtcsvInstallCsvFile . . . . 375
vmtLibImport . . . . 378
vmtValidateUnifiedLibrary . . . . 381
vrfCheckTILPVersion . . . . 383
vrfComplianceAudit . . . . 385
vrfExportLayoutSkill . . . . 387
vrfExportPackage . . . . 393
vrfLowerPriority . . . . 396
vrfHiUpdate . . . . 397
vrfRaisePriority . . . . 398
vrfSipSet . . . . 399
vrfSipGet . . . . 402
vrfTLineAbut . . . . 403
vrfUpdateTILPVersion . . . . 405


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# 1

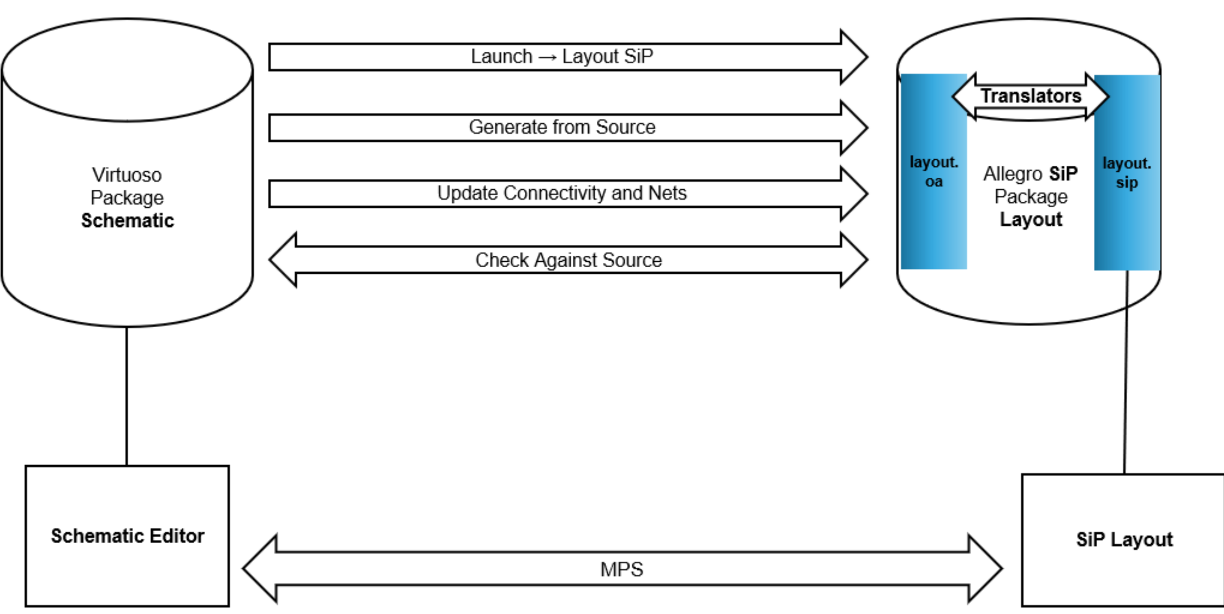
# Introduction to Virtuoso Multi-Technology Solution



Virtuoso Multi-Technology Solution enables various Virtuoso schematic–driven layout generation flows. Virtuoso Schematic Editor is the front-end application of choice for back-end tools. The basis of the flows is the use of unified libraries across systems.

The Virtuoso Multi-Technology Solution combines the benefits of Virtuoso schematic-driven Layout MXL methodology along with the features of the SiP Layout. The key features of Layout MXL, such as generate from source, check against source, update connectivity and nets, cross-selection, and hierarchical schematic are available in SiP layout. You can choose to work in the Virtuoso or Allegro environment.


© 2023 Cadence Design Systems, Inc.




The Framework provides a basic set of services that allows you to design packages, modules, and boards using a variety of heterogeneous integration styles such as die stacking, wire bonding, interposers, and wafer-level packaging (WLP). The schematic for the system might be hierarchical, consisting of ICs, packages, and boards. The flow provides a multi-technology environment for simulation and implementation.

The integrated design system consists of two design concepts, logical design and physical layout. Logical design is the process by which the design intent is specified in a schematic. The schematic can be used for ideal or pre-layout simulation to verify that the intent matches the specification. In logical design, you assign part definition names to instances in the schematic: this association is used to bind the appropriate footprint and other CDF parameters. Creating a physical layout is the process by which the design intent is implemented and expressed in terms of components and routes. The part definition names on the schematic instance imply the name of the footprint that must be used during physical layout.

A layout generation process, *Generate All From Source*, is used to create the initial layout. Edits in the schematic can be propagated to the layout through an update flow called *Update Components and Nets*. Changes made in the SiP Layout can be checked against the schematic to provide the in-design layout-versus-schematic (LVS) check that ensures that the layout never departs too far from the schematic. This process is called *Check Against*


© 2023 Cadence Design Systems, Inc.

*Source*. At any time in the physical layout process, you can cross-probe to identify corresponding parts, packages, and signals in the layout design and the schematic.

During the layout process or as a step after the layout is done, parts or all of the layout can be extracted using the high-accuracy 3D or planar extractors to create the frequency-dependent models known as `sparam`. These sparam models can be backannotated or stitched into the original schematic to create a more accurate simulation of the system. Subsequently, pre-and post-layout simulations can be compared to verify that the design specifications are met.

The Virtuoso Multi-Technology Solution environment involves various tools, such as Cadence SiP Layout Option, EMX Solver, Clarity 3D solver, Virtuoso Schematic Editor XL, Virtuoso ADE Explorer, Virtuoso ADE Assembler, and Virtuoso Visualization and Analysis XL.

***Related Topics***

*Virtuoso Schematic Editor User Guide*

*Virtuoso Layout Viewer User Guide*

*Virtuoso Layout Suite XL: Basic Editing User Guide*

*Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*

*Virtuoso ADE Explorer User Guide*

*Virtuoso ADE Assembler User Guide*

*Virtuoso Visualization and Analysis XL User Guide*

# Virtuoso Multi-Technology Solution Setup

Virtuoso Multi-Technology Solution requires Allegro® Package Designer Plus requires SiP Layout option. Allegro databases must be upgraded to version 17.4 or later for use in Virtuoso Multi-Technology Solution. The databases with `.mcm` and `.sip` file extensions can be used in the flows.

The values for bump parameters changed from `Conductivity` to `Material` when in SPB 17.4 and 22.1. However, `Conductivity` cannot be automatically mapped to `Material`. It is recommended to manually change value for bump parameters in the Allegro Package Designer Plus database."


© 2023 Cadence Design Systems, Inc.

*Important*

It is recommended to set up the PCB hierarchy path before IC and Sigrity hierarchy paths to ensure that the Virtuoso MultiTech flow works smoothly.

```
set path = ( $MMSIMHIER/tools.lnx86/bin $PCBHIER/tools/bin $ICHIER/tools/bin
$ICHIER/tools/dfII/bin $SIGRITY_EDA_DIR/tools/bin )
```

## License Requirements for Virtuoso Multi-Technology Solution

The following are the licensed products needed for Virtuoso Multi-Technology Solution:

**Virtuoso Electromagnetic Solver Assistant**

Virtuoso Layout Suite MXL

EMX_Solver

Clarity 3D Solver

**Virtuoso System Design Solution - RF Module Layout**

Virtuoso Schematic Editor XL Plus

**(Option1) For SiP Layout editing with Virtuoso Layout Suite**

Virtuoso Layout Suite MXL

OR

Virtuoso Layout Suite EXL + Virtuoso_MultiTech_Framework (95022)

**(Option2) For SiP Layout editing with Allegro Package Designer Plus**

Allegro Package Designer Plus and SiP Layout Option

***Related Topic***

*Virtuoso Software Licensing and Configuration Guide*


© 2023 Cadence Design Systems, Inc.

# 2

# Unified Libraries

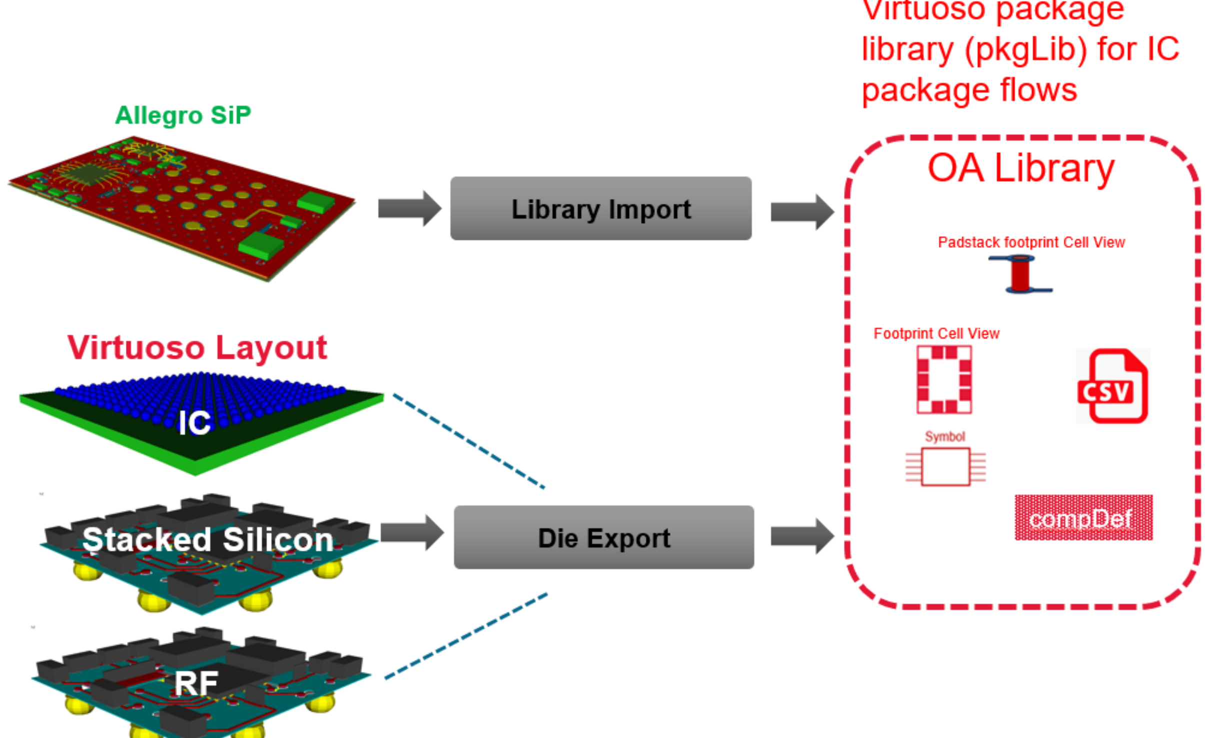
Unified libraries refer to the consistent organization of design data libraries. They provide a single point of entry for all Virtuoso-driven multiple technology flows. The flows include:

- Virtuoso Multi-Technology Solution: a Virtuoso schematic-driven Allegro Package Designer Plus or Layout MXL implementation flow
- Virtuoso RF Solution: a Virtuoso schematic-driven Virtuoso layout implementation flow
- Virtuoso EM Flow: a verification flow that allows the extraction of parts of an entire system that can be stitched into a schematic for simulation

Libraries contain schematic symbols, footprints, component definition mapping between schematic symbol and footprints, TILP supermasters, part variance CSV files, and simulation models. The mapping between schematic and layout components is implemented through standard parameters specified on schematic instances. The interpretation of these parameters is uniform across packages, dies, embedded components, and surface-mounted devices. These libraries can be used for the implementation and verification of hierarchical


© 2023 Cadence Design Systems, Inc.

schematics and layouts that span across boards, modules, packages, and ICs in an intelligent system design.

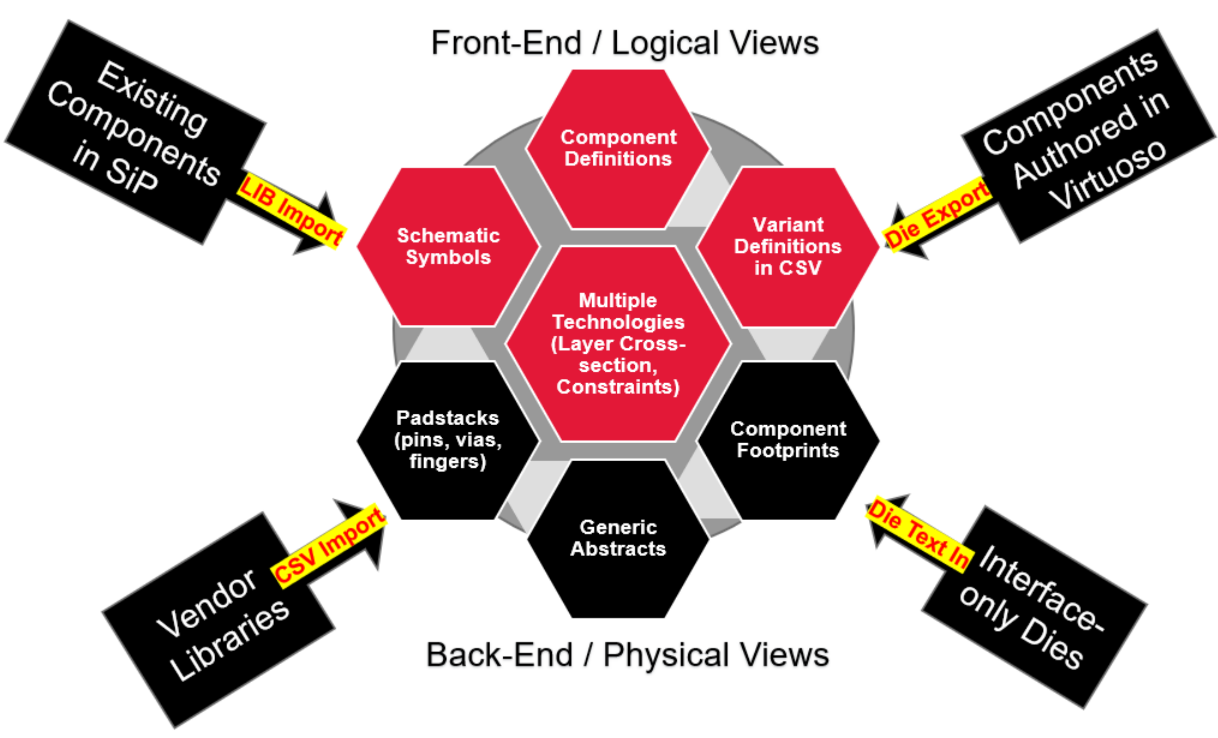




© 2023 Cadence Design Systems, Inc.

# Views in the Unified Library

The views in a unified library are shown in the illustration below.




The following views are a part of unified libraries:

- Schematic Symbols
- Component Footprint Views
- Component Definitions
- Variant Definitions in CSV Files

## Schematic Symbols

Schematic symbols are automatically created during the `libImport` or die export process. The terminal names used for these symbols must be available when the symbols are created. When Allegro views are used, the terminal name information is provided by component


© 2023 Cadence Design Systems, Inc.

definitions. When the views are created by exporting a die, the information is extracted from an IC layout. When the symbols are created by the CSV import flow, the information is provided through a mapping file provided by the customer.

Schematic symbols can be split to enhance the readability of schematics. You can manually split the symbols in Virtuoso Schematic Editor, or they can be automatically split during the import process.

## Component Footprint Views

Footprint views are also automatically created during the `libImport` or die export process. A footprint represents the physical view of the instance that is used while generating a layout. The footprint terminal names must be available when the footprint is created. When Allegro views are used, this information is provided by DRA views or symbol definitions in a SiP file. When the views are created by exporting a die, you can provide the terminal names through properties placed on the bump instances in the IC layout.

## Component Definitions

Component definitions provide mappings between schematic symbols and footprint views.

The attributes of a component definition are:

- Device type: a unique string that identifies the component definition.
- Component class: IC (Die), IO (BGA), or discrete (SMDs).
- Package: a unique string that identifies the package used by the `compDef` library.

The component definition also provides a list of `compDefPin` objects. These represent the terminals on the footprint view and are referred to as pin numbers. Each `compDefPin` can be mapped to one or more `funcDefPin`, referred to as pin names.

In Allegro, component definitions also contain properties that can be associated with either the component or specific pins on the component. These properties may be used as constraints or for downstream flows, such as simulation, verification, and so on. In addition, there are specific properties such as `PARENT_PPT` or `PARENT_PART_TYPE` that identify a generic name or the family of the component definition. All similar components belong to the same family and have the same `PARENT_PPT` or `PARENT_PART_TYPE` property.

In Virtuoso, a single component definition is created for all Allegro component definitions that share the same `PARENT_PPT` or `PARENT_PART_TYPE` property. These definitions are


© 2023 Cadence Design Systems, Inc.

generic because the properties that make them unique are not stored in the definition itself. The properties are stored in the Variant Definitions in CSV Files.

The component definitions in a library can be queried in SKILL by using the VRF Package Infrastructure Functions.

## Variant Definitions in CSV Files

Variant or Part Definition CSV files contain the properties that are present in the Allegro component definition. In addition, the import process adds some mandatory properties that facilitate the mapping between the schematic symbol and the footprint view. These properties are `footprintLibNames`, `footprintCellName`, `footprintViewName`, and `altFootprintCellNames`. The mapping process goes through the libraries specified in the `footprintLibNames` property and tries to find the cell and view described by the `footprintCellName` and `footprintViewName` properties. The first cellview that is found is used as the footprint cell when the layout is generated for a schematic that contains an instance of the generic schematic symbol.

You can import the Part Definition CSV using the Part Data form, which displays information from a `part.csv` file for the selected library or cell. Each row in the Part Data form corresponds to a row in the `part.csv` file. The first row is the filter row using which you can filter the available parts based on specific criteria. For example, if you want to view only those parts of the `Pack_Type` is `01005`, you can type `01005` in the filter column for `Pack_Type`. You can also sort the data in the form by clicking on the header of a particular column.

### *Related Topics*

vmtcsvCreateComponentCellViewsFromCsv

vmtcsvInstallCsvFile

vmtLibImport


© 2023 Cadence Design Systems, Inc.

# Component Modeling

This topic describes modeling of padstacks, components, and symbols from the Allegro platform to Virtuoso Studio for creating views in unified libraries.

## Padstacks

In Allegro, any connection point requires a padstack. Connection points are:

- Pins on dies, packages, and discretes
- Vias connecting clines and shapes on different layers
- Bondwire fingers connecting bond wires to a substrate

A padstack is a collection of pads on multiple layers, such as conductor layers, mask layers, or keep-out layers of a package or board. A padstack spans multiple conducting layers including single or multiple drill hole definitions.

In Virtuoso, padstacks are presented as TILP instances of pins for packages, dies, or SMDs, and pinFig shapes for embedded components or Tlines, and instances of vias and fingers. All pads and drill holes are defined as curved polygons. Only regular pads are recognized in Virtuoso. In addition, all padstack attributes, such as plating, backdrilling, and multi-drill, are not recognized in Virtuoso. However, some of the attributes are stored as properties without interpretation in Virtuoso; for example, multi-drill patterns are represented in Virtuoso as drill shapes.

When a SiP file is imported or a directory of PSM files is imported into Virtuoso, all implied padstacks are imported into a unified library. A footprint view, in addition to a TILP layout view, is created for each padstack. The TILP views of padstacks are used in the footprint view of a die or package component.

## Symbol Definitions

In Allegro, a symbol definition is used to represent the footprint of a component. This symbol definition contains pins that have references to their padstacks and shapes that are visible to the package layout. In Virtuoso, the symbol definition is modeled as a footprint view with instances of padstack TILPs. These padstack instances identify if the pins are front or back pins. Annotations and other shapes of the instances are mapped from generic layers in Virtuoso to layers in the SiP layout. The properties are stored on the shapes to indicate the class and sub-class to be used in the SiP layout.

**Note:** The PLATING_BAR class symbols are ignored by the vmtLibImport SKILL function.


© 2023 Cadence Design Systems, Inc.

## Components and Symbols

In Allegro, connectivity and physical information is presented through two separate abstracts. The connectivity view is presented through the component, which is associated with a component definition. The physical view is presented through the symbol, which is associated with a symbol definition.

In Virtuoso, the connectivity and physical information is provided through a single abstraction, `oaInst`. The `oaInst` has a property called DEVICE_TYPE that identifies the component definition that must be used. It also contains the `footprintLibNames`, `footprintCellName`, `footprintViewName`, and `altFootprintCellNames` properties, which are used to identify the footprint view.

The properties created on a component help determine the mapping to the associated schematic instances and nets to enable cross-probing. The top-level nets are created and connected to the pins on the component. A symbol is associated with a symbol definition and contains the physical information, such as location, orientation, mirror, and so on, in a SiP layout. The same symbol is created and moved to the right layer based on the TILP parameters, such as flipped, mirrored, order, and offset in Virtuoso.

During the component definition assignment for an instance in Virtuoso, the CDF parameters associated with each instance are used. The component definition also specifies the list of pins and functions associated with the instance.


© 2023 Cadence Design Systems, Inc.

# Types of Libraries for Creating Unified Libraries

You can use unified libraries to enable all the Virtuoso multiple technology flows. Design data libraries from various sources are imported or created as unified libraries in the following ways:

- Allegro views and self-contained SiP files are imported by the `libImport` utility to create abstracts for padstacks, LGA/BGA packages, discrete devices, and dies.
- Virtuoso IC abstracts are created by Die Export.
- Vendor libraries of discrete devices are imported through CSV Import.

The following types of design data libraries can be made as unified libraries:

- Exported Die Library
- BGA Library
- Tline Library
- pkgLib Library
- Passive Component Library
- CSV Import Library

## Exported Die Library

Exporting a die from a Virtuoso layout generates the following data:

- Abstract: This contains the instances of the padstack that correspond to the IO cell or shape-based interfaces in the source IC layout. The terminal names in the abstract view correspond to the physical domain names.
- Symbol: This corresponds to the logical interface of the physical footprint of the exported die. The terminal names in the symbol view correspond to the logical domain names.
- Schematic: This is the mapping schematic between the source IC symbol view and the exported symbol view. The duplicate pins in the source IC layout are made unique. Therefore, this mapping schematic has connectivity from the unique interface of the exported die to the generic interface of the source IC.
- Layout: This is the placeholder die TILP cellview.

Exporting the die also creates one cell for each unique padstack used in the die abstract. Consequently, the following views are created for the padstack:


© 2023 Cadence Design Systems, Inc.

- Abstract: base cellview of the padstack
- Layout: placeholder padStack TILP cellview

### Preparing for Die Export

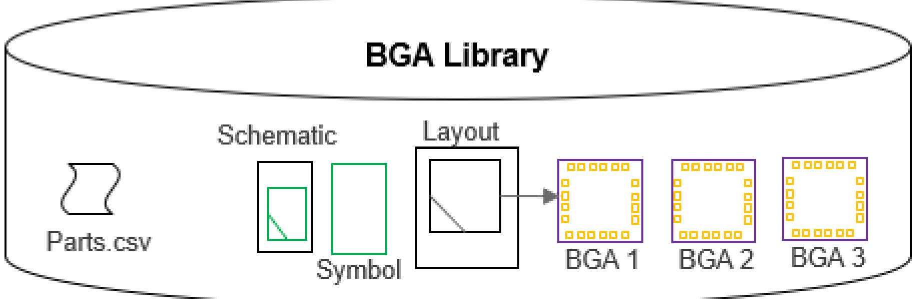
To ensure that the die abstract is accurate before using it in the package, you need to prepare the die. There are three important requirements. For details, refer to Preparing for Die Export.

### Exporting Dies

This is the step in the flow where the IC/die footprint is handed over to the package designer. By exporting the die, you can create technology-independent abstraction, which enables Edit-in-Concert, layout versus abstract (LVA) checks, and cross-fabric simulation using die schematic and model-based simulation for dies. For details, refer to Exporting Dies.

## BGA Library

Ball grid array (BGA) is a type of die component whose pins are solder balls arranged in a grid pattern. For an effective use model, group multiple BGAs within a part table.

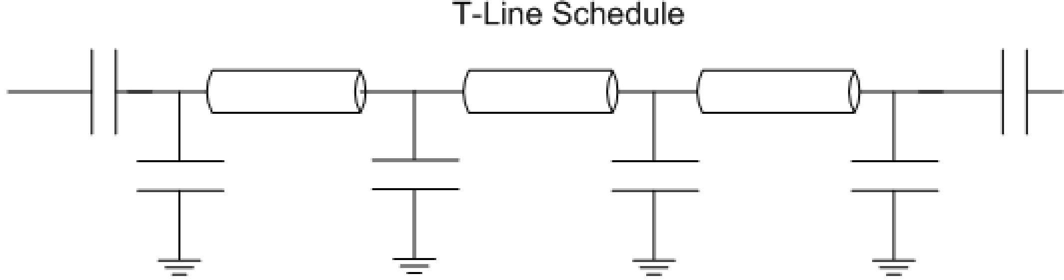



## Tline Library

The Tline library contains the symbol view, simulation view, and OpenAccess layout view for TLines. You can instantiate the symbol views and capture connectivity in the package


© 2023 Cadence Design Systems, Inc.

schematic. Tline components are derived from `rfTlineLib`, which is a library of wideband-accurate transmission line models in multi-conductor microstrip and stripline configurations.




### pkgLib Library

The pkgLib library contains wirebonds and fingers. These are stored as TILPs in the library for use in the Virtuoso Multi-Technology Solution. The mapping from Virtuoso instance parameters to Allegro symbol parameters is provided in the TILP parameter mapping.

### Passive Component Library

The passive library contains different topologies of inductors to be implemented as generic TILPs in Virtuoso.

### CSV Import Library

The CSV library describes the part variants as CDF parameters in Virtuoso. There is a need to support `.csv` import functionality for vendor-provided SMD libraries. The component definitions are created with the name-to-number mapping for pins. Subsequently, you can create the base cellview and the TILP that can be instantiated in the layout views.

## Unified Library Validation

Unified libraries must be validated to avoid common cross-view errors that happen in the libraries and designs in various flows. You can start the validator by using the `vmtValidateUnifiedLibrary` SKILL function. This function can be run on a library or a cellview.


© 2023 Cadence Design Systems, Inc.

When run on a library, it validates the following for errors:

- All the schematic symbols stored in the library.
- All the component definitions stored in the library.
- All the footprints referenced in the `part.csv` of the cell associated with the component definition to check the pin names and numbers against a symbol and footprint.

When run on a cellview, it validates the following for errors:

- All the instances and vias in the cellview.
- Corresponding component definitions.
- All the schematic symbol views referenced in the design.
- All the footprints referenced in the `part.csv` of the cell associated with an instance or via to check the pin names and numbers against the symbol and footprint,
- The parameters, such as orientation, flipped, mirrored, angle, and rotation, are consistent.

The following table illustrates all the error conditions reported by the library validator in CIW and the corrective actions that you can take to fix the errors.

| Error Condition | Corrective Action |
|---|---|
| Missing footprint library from instance with a master cellview | Ensure that the specified footprint library is defined in the `cds.lib` and is accessible. |
| Missing footprint cell in library from instance with a master cellview | Ensure that the specified footprint cell is defined in the footprint library and is accessible. |
| Missing footprint view for cell in library from instance with a master cellview | Ensure that the specified footprint view is present in the footprint cell in the footprint library and is accessible |
| The orientation, flipped, mirrored, angle, and rotation values of an instance in a cellview do not match the expected orientation and rotation | A script is provided by Cadence to fix the instance CDF parameters that have been changed inconsistently. Contact Cadence customer support for assistance. |
| Cannot find library in the list of libraries | Ensure that the library is defined in the `cds.lib` and is accessible. |
| Missing cell in the library | Ensure that the specified cell is present in the library and is accessible. |


© 2023 Cadence Design Systems, Inc.

| Error Condition | Corrective Action |
|---|---|
| Missing TILP for a cell | Rerun `vmtLibImport`. |
| Terminal differences between the component definition of a device and its symbol view | Rerun `vmtLibImport`. |
| Missing symbol view for a cell in a library | Rerun `vmtLibImport`. |
| Missing part.csv for a cell | Rerun `vmtLibImport`. |
| Terminal differences between the component definition of a device and its footprint view | Rerun `vmtLibImport`. |
| Footprint referenced in a cell cannot be found | Rerun `vmtLibImport`. |

***Related Topics***

Unified Libraries

vmtValidateUnifiedLibrary

vmtLibImport

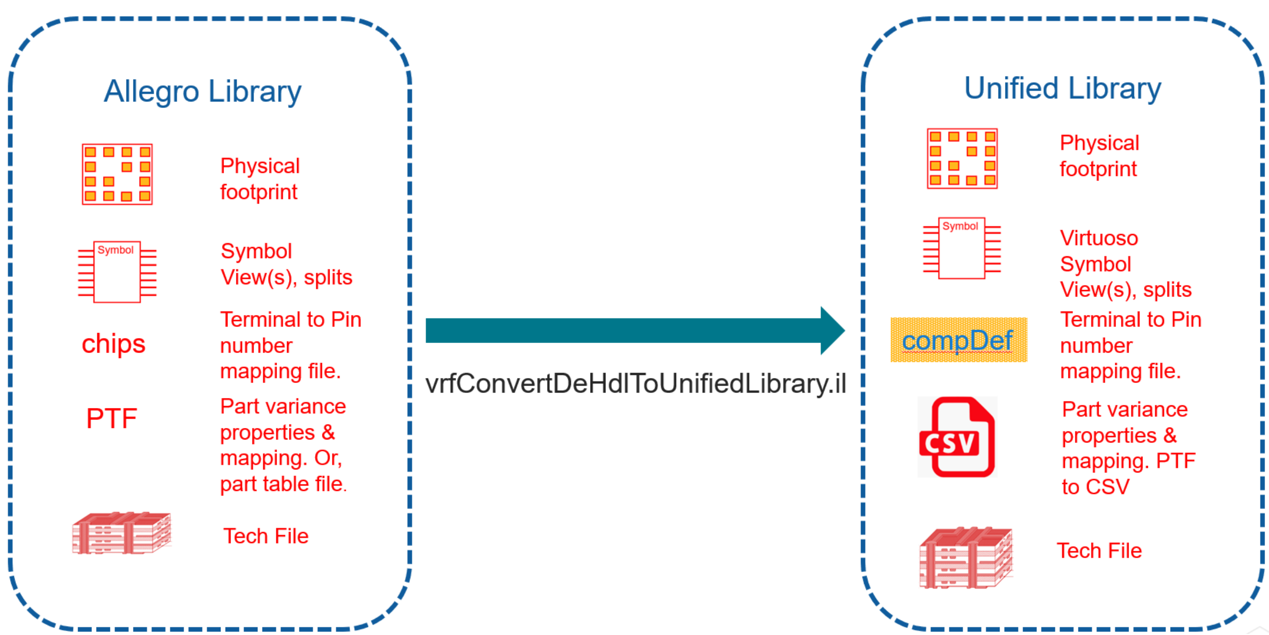
# Converting Allegro Libraries to Unified Libraries

During conversion, symbol cells are created as a part of the unified library that both Allegro Package Designer Plus & Virtuoso Schematic Editor can use in the SiP Layout Option to Virtuoso Schematic Editor flow. The Convert Allegro Libraries to Unified Libraries form lets you convert native libraries into multi-platform unified library components across the platforms.

Unified libraries are crucial in heterogeneous integration designs to support interoperability across tool boundaries. The data sharing and design exchange for advanced package design flows is simplified through conversion. A single symbol in Allegro platform with a vectorized

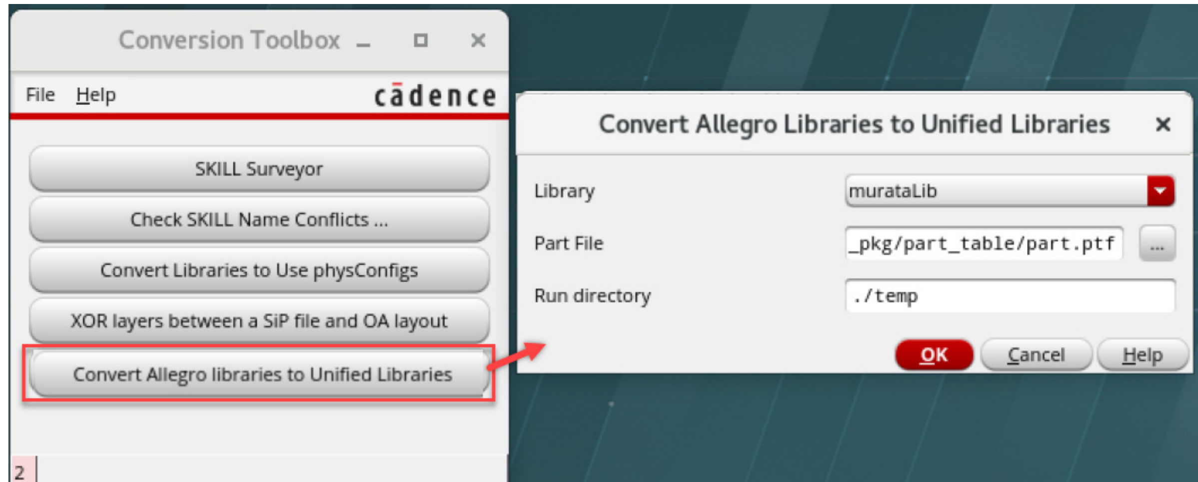

© 2023 Cadence Design Systems, Inc.

terminal name `A<0>` that has 100 pin numbers is transformed to 100 symbols in Virtuoso Studio to maintain traceability.




To convert Allegro libraries to unified libraries:

1. Click *Tools – Conversion Toolbox – Convert Allegro libraries to Unified Libraries* in the CIW. The Convert Allegro Libraries to Unified Libraries form opens.

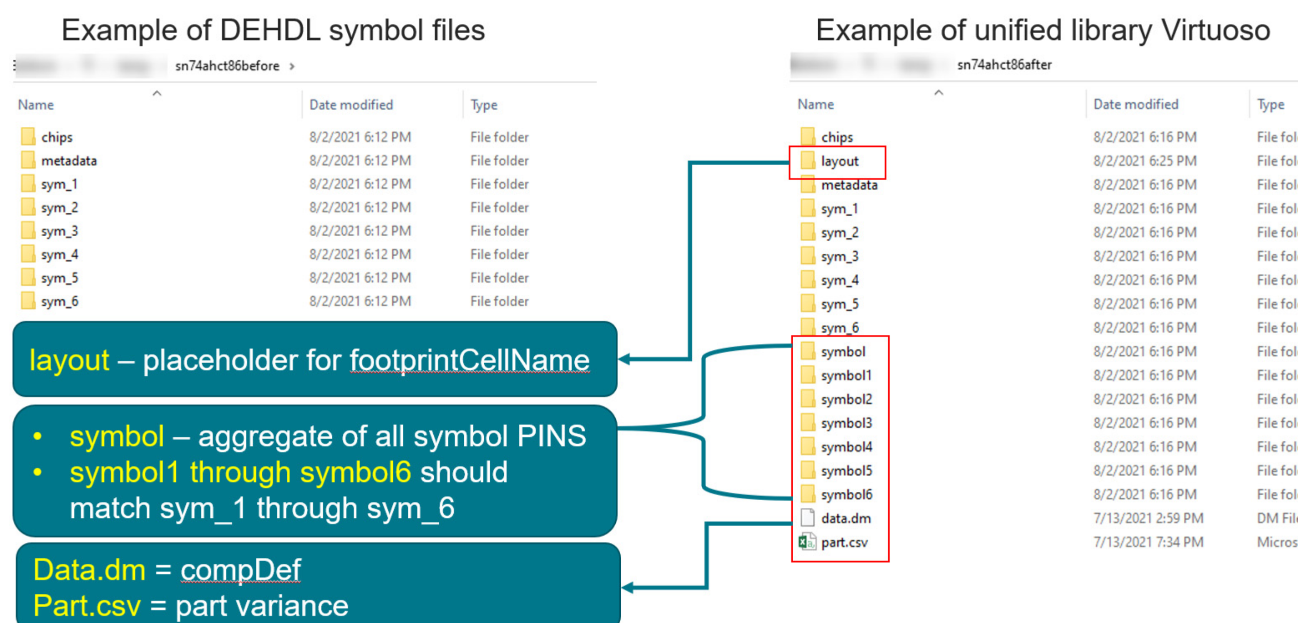



2. Specify the library containing the part file to convert the symbols and the path to the part file.

3. Click *OK*. The message about the symbol from Allegro being split is displayed in the CIW.


© 2023 Cadence Design Systems, Inc.

This is the before and after example of how DE-HDL library is augmented with additional views to support Virtuoso Studio.




However, there are the following limitations for conversion:

- The terminal names that are different across a single-cell-symbol are not supported.
- The `DE-HDL chips.prt` attributes, such as `POWER_PINS`, `DEFAULT_SIGNAL_MODEL`, and `JEDEC_TYPE` are not supported.

***Related Topics***

Unified Libraries

Convert Allegro Libraries to Unified Libraries Form

# Netlisting Setup for Unified Library Components

A customized netlisting procedure and setup can be used for including footprint-specific sparam model or SPICE model files.

To create an sparam or a SPICE model view,

1. Copy a symbol view to an sparam model or a SPICE model view.


© 2023 Cadence Design Systems, Inc.

2. Add footprint-specific `.s2p` files to the sparam model or SPICE model cellview directory.
3. In the `part.csv` file, the `ASI_MODEL` column should identify the correct sparam or SPICE model filename prefix to use.
4. Click *Tools – CDF – Edit* to add simulation CDF.
5. Click *OK*.

### *Related Topics*

vsdpSparamCSVModelNameField

vsdpSpiceCSVModelNameField


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

3

# Import Libraries and ICs

The OA technology model has been enhanced to support package layouts. This technology data can either be imported directly from Allegro or loaded through the Virtuoso technology file. In addition, vendor and customer component libraries can be imported and used in Virtuoso RF while creating and verifying the package schematic and layout.

## Create Technology File

The Virtuoso technology database is provided by the foundry (for IC fabrics) or the Outsourced Semiconductor Assembly and Test (OSAT) for package fabrics. Typically, the technology rules provided by OSAT houses are in the form of manuals that must be transcribed to Virtuoso technology file constructs that are subsequently loaded into Virtuoso.

The Virtuoso technology model has been extended and adapted to cover new constructs that are specific to the packaging world. Examples of the packaging concepts that are captured in the Virtuoso technology file are:

- An attribute stating that the corresponding technology represents a package fabric. This attribute is used heavily in Virtuoso to provide specialized operations for package layouts.
- The layer cross-section of the packaging substrate. Unlike IC fabrics, it is important for the cross section to include detailed electrical and physical information for both the conducting and dielectric layers. This information is used by the 3D extractor to accurately present the mesh that needs to be solved to create realistic S-Parameter models that can be integrated with the rest of the system for realistic simulation.
- The physical constraints define the spacing, width, and clearance rules between different objects in the layout. Unlike IC fabrics where spacing rules are provided for a specific layer, in package fabrics different spacing rules might apply for different types of objects on the same layer. A mitigating factor is that unlike IC fabrics, where a constraint can be a complicated 3D table, a package fabric has scalar values.
- The Boolean operations describing the relationship between voiding layers and conducting layers to create the effective conducting shapes. Unlike IC fabrics where a


© 2023 Cadence Design Systems, Inc.

shape on a layer associated with a trim operation splits a single shape into different conducting sub-shapes, the package fabric uses the trim operation to etch away portions of a dynamic shape that could cause an electrical short with another shape.

- Unlike IC fabrics where via definitions include metal shapes and cut shapes that represent the signal propagation from one layer to another, package fabric vias are typically represented with a mechanically drilled hole where there is an implicit assumption of conductivity through the range of layers covered by the via.
- The specification of 3D bond wire profiles that are used in a bond wire-based attachment of dies. This is unique to package fabrics.

**Note:** The fabric type of the design has to be set to "package" or "module" in the technology file for the package design capabilities to be available. Following message is issued if the appropriate fabric type has not been specified:

```
*WARNING* (OA-37002): Cannot set SIP Design type, because fabric type
'unspecified' of the attached technology database is not 'package',
'board' or 'module'.
```

The technology file must be created manually if the alternative is not available. A sample technology file is provided in the appendix of the Interoperability document (to be provided).

***Related Topics***

Import Technology File

Update Technology File

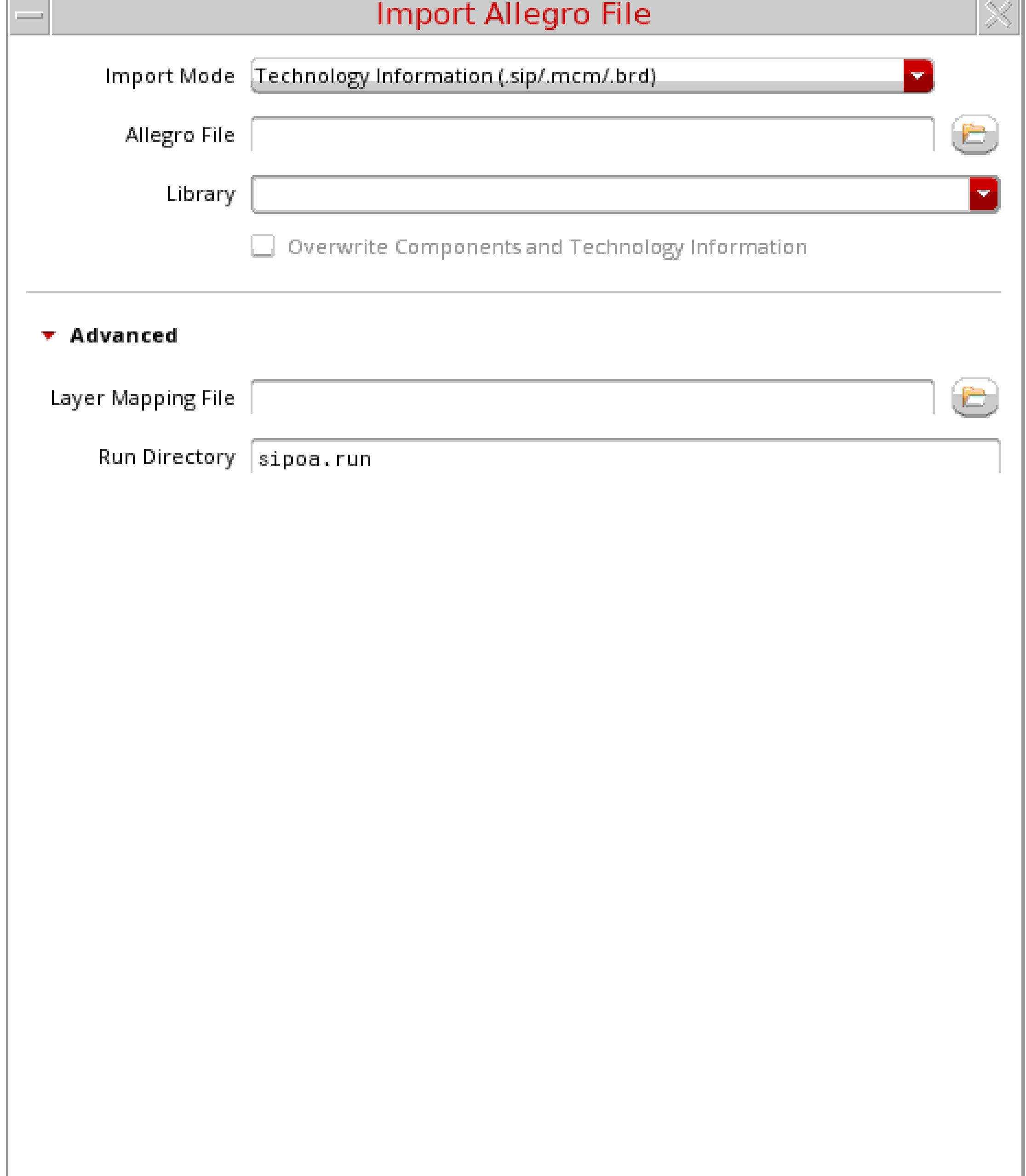
Constructs in the Technology File

# Import Technology File

A simpler alternative to create a technology database using a hand-written technology file is to import the technology from an existing Allegro `.sip` file that has been built using the same or a similar technology. For similar technologies, the technology file must be manually edited with the correct values for the target technology. After completing these edits, load the technology file into Virtuoso using the usual "Technology Manager" options.


© 2023 Cadence Design Systems, Inc.

You can import only technology information from Allegro using the translators by specifying the `Technology Information (sip/mcm/brd)` option in the *Import Mode* field.




The import of technology file includes mapping the following in SiP and OA:

- Padstack and symbol definitions of the SiP objects that map to OA cellviews including renaming to avoid same name cells in a library.


© 2023 Cadence Design Systems, Inc.

- Variants for symbols and padstacks that were modified at the instance level.
  - Translations, rotations, mirroring, and deletion of objects inside symbols
  - Padstack references (vertical offset in the layer stack), and unused pad suppression
- The symbols and padstacks with differing sets of parameters in SiP and OA
  - Top-level vias in OA that do not have mirrored parameters
- Derivation of cell names using signatures
- Layer mapping
  - Cross-section layers are mapped to layers with functions metal, dielectric, diestack
  - Mapping of non-cross-section layers with correspondence in cdsDefTechLib
  - Mapping of non-cross-section layers without correspondence in cdsDefTechLib

***Related Topics***

Create Technology File

Update Technology File

Constructs in the Technology File

## Update Technology File

When importing information into a library where data exists, you have the following options for updating the technology database while importing information from an existing `.sip` file:

- Overwrite the top-level layout, import new components, and technology objects.

  The *Layout (.sip/.mcm/.brd)* is the default *Import Mode* while running *File – Import – From Allegro*. All the information present in the `.sip` file is imported into the target library. If the target library is already present, you can optionally overwrite existing components or skip existing components.

  **Note:** When you make simple edits to the package layout, there is no need to recreate the techDB and all the components. Creating the techDB requires a lot of information and most of it is mapped to SiP. Therefore, recreating the techDB could result in loss of information.

- Overwrite only the technology information in the target library.


© 2023 Cadence Design Systems, Inc.

In this mode, the *Import Mode* field is set to `Technology information (.sip/.mcm/.brd)`. The top-level layout is not modified, but the technology and component information is updated.

- Import components represented as Allegro `.dra` files into the target library.

  A directory containing DRA files can be imported into the target library. In this mode, the *Import Mode* field is set to `DRA files`. After import, the components are available as 5.X cellviews in the target library and can be referenced in the package layout.

***Related Topics***

Create Technology File

Import Technology File

Constructs in the Technology File

# Constructs in the Technology File

The sections in the technology file that are relevant for packaging are best demonstrated through a sample technology file. Constructs that are relevant to packaging are explicitly described in the following section. Look at the sample technology file.

| | |
|---|---|
| `Controls section` | The new construct is `fabricType`. This indicates that the library in which this file is loaded contains package fabric cellviews. Note that it is important for libraries containing package cellviews to contain this construct. Many Virtuoso RF Solution features depend on this designation in the technology database. |

```
layerDefinitions section
```


© 2023 Cadence Design Systems, Inc.

| | |
|---|---|
| `techLayers sub-section` | There are two changes in this section. For every layer in the layer cross section, a VOID layer is added to allow the voiding of shapes on that layer. For details, refer to. Conceptually, it can be viewed as a Boolean operation that creates a hole in the layer it voids. This allows shapes on other nets to be created in the hole without creating an electrical short with the net on the layer that is being voided. See<br><br>The new built-in layers that are added for the Virtuoso RF Solution are highlighted in this section. These layers are used to build Technology Independent Layout Pcells (TILP) that are one of the foundations of the Virtuoso RF Solution. Technology independence allows components to be created without being aware of the target technology. When a technology independent Pcell is instantiated in a package layout (`parentCell`), the library containing the parent cell (`parentCellLibName`) is used to find the technology database associated with this instantiation. Shapes on generic layers (using the System-reserved layers highlighted in this section) are mapped to the appropriate layer in the package substrate described in the technology database. For example, shapes on the `beginGeneric` layer are mapped to either the top or the bottom of the package substrate depending on whether the component is connected to the top of the substrate or the bottom of the substrate. |
| `techLayerPurposePriorities sub-section` | New layer purposes are provided for each layer and purpose combination within the layer cross section. A new built-in purpose dynamic allows the voiding of planes. In the below-mentioned `techDerivedLayers` section, a boolean operation is specified to select shapes on the dynamic purpose and remove (NOT operation) the corresponding VOID shapes to create the effective conducting plane for that net. |
| `techDerivedLayers sub-section` | In this section, the boolean operations governing the behavior of voiding are provided. For example, in the operation below, M1_PLANE selects all the shapes on M1/dynamic and, M1_SHAPE subtracts the shapes on M1_VOID from M1_PLANE to generate the effective conducting shape. |

```
( M1_PLANE                549       ( M1        'select
dynamic   ))
  ( M1_SHAPE               550       ( M1_PLANE   'not
M1_VOID   ))
```


© 2023 Cadence Design Systems, Inc.

| layerRules section | |
|---|---|
| functions sub-section | The voiding layer purpose pairs are explicitly identified as containing a function of `trims()` indicating that they perform a negative operation on some other layer purpose pair. Dielectric layers are also designated a function of "dielectric" indicating that they are not conducting layers. |
| analysisAttributes sub-section | The `analysisAttribute` section is used to describe each layer of the cross-section in sufficient detail to allow accurate 3D extraction. The new attributes that are available are:<br>■ `materialName`<br>■ `thickness`<br>■ `conductivity`<br>■ `permittivity`<br>■ `lossTangent`<br>For details, refer to Navigating through a Technology File. |
| **constraintGroups section** | |
| constraintGroups section | The `constraintGroups` section has been enhanced to describe packaging constraints. Width, spacing, and hole spacing constraints can be defined for each layer. New parameters are provided on these constraints that allow the specification of spacing values. Spacing check is performed on these values based on the object types. For example, different spacing values can be provided between a<br>■ M1 shape in a via and a M1 shape represented as a line (path).<br>■ M1 shape in a bond finger cell and M1 shape represented as a shape (polygon).<br>The complete list of these types are: `line`, `pin`, `thruPin`, `smdPin`, `testPin`, `thruVia`, `bbVia`, `testVia`, `microVia`, `shape`, `bondFinger`, and `via`. |

***Related Topics***

Create Technology File


© 2023 Cadence Design Systems, Inc.

Import Technology File

Update Technology File

# Library Import

The table below shows how different components can be imported into the Virtuoso RF Solution.

The following types of libraries can be imported:

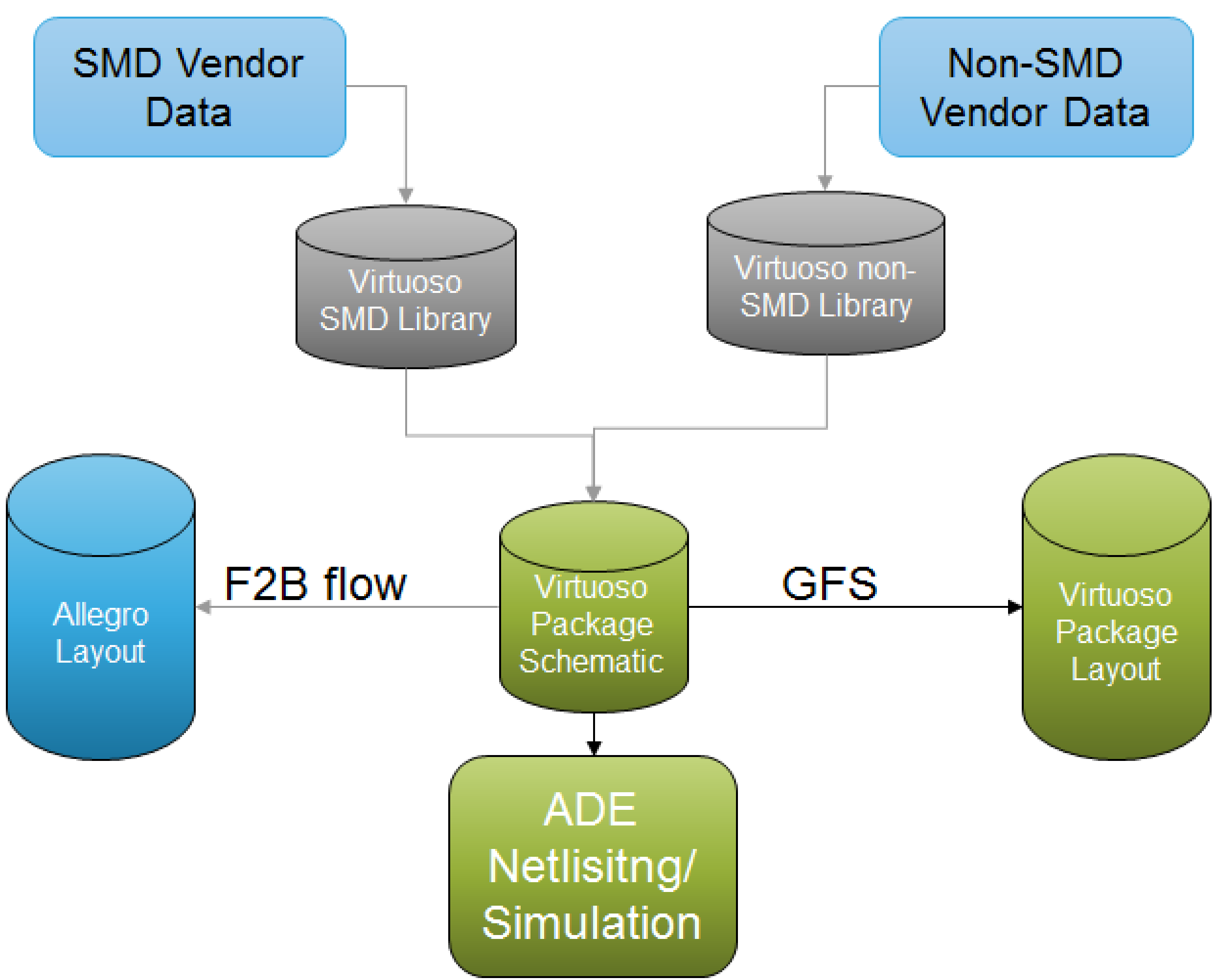
| | |
|---|---|
| `Components available in Allegro as .dra files.` | Use *File – Import – From Allegro* and set the *Import Mode* to *DRA files*. |
| `SMD Components provided in a vendor library, such as Murata` | These libraries are special Pcell/TILP components, bound to package technology layers. SMD jedecLib is imported from Allegro. It contains the physical layout with the Top and Bottom as generic layers, which inherits the top-most and bottom-most layer defined in the package layer stack-up. |
| `Die or IC` | Use the Die Export feature to create an abstract and a symbol for the die that can be used in the package schematic and layout.<br>These components are available as TILP cells to allow generic shapes in the base cellview of the component to be mapped to the right layer depending on its orientation and whether it is connected to the top or bottom of the substrate. Die libraries are not limited to only one technology file. They can be reused for different types of packages and technology files. |
| `Embedded components to be used in the package layout` | Draw the shapes in the embedded component using Virtuoso and create a component that is usable in the package layout.<br>These libraries are special Pcell/TILP components that are bound to the technology layers in such a way that the EM structures/coils are routed with the package metal layers. Embedded components cannot be technology independent because they are embedded into the package substrate and are aware of the specific layer cross-section where they are placed. |


© 2023 Cadence Design Systems, Inc.

| | |
|---|---|
| `Package connectors, such as BGA or LGA that are available as an Allegro file` | Import the connector as a cellview using *File – Import – From Allegro* and set the *Import Mode* to `Layout (.sip/.mcm/.brd)`. The form allows you to also create a symbol for the connector that can be used in a schematic. |

To ease the transition from an Allegro-based flow to the Virtuoso RF Solution, *File – Import – From Allegro* is used to import a library of existing component footprints into a 5.X library.

To enable you to use the raw component data from vendors, such as simulation/packaging combinations and simulation models, it can be imported as libraries.




Surface Mounted Device (SMD) components are resistor, capacitor, and inductor. SMD data includes the following:

- CSV File


© 2023 Cadence Design Systems, Inc.

- ❑ Packaging/Physical Data Association
- ❑ Simulation Data Association

```
PART_NO,TYPE,JEDEC_TYPE,VALUE,TOLERANCE,,VOLTAGE,TEMP_CHAR,,FREQ_RANGE,
"Part Number","Type","JEDEC Type","Capacitance [pF]",Tolerance,,"Rated Voltage [V]","Temp. Characteristics (EIA)",,"Frequency Range",
GRM0335C1E3R3CA01,General,0201,3.3,+/-0.25[pF],,25,C0G,,100[MHz]-8.5[GHz],
GRM1555C1HR10WA01,General,0402,0.1,+/-0.05[pF],,50,C0
GRM1555C2AR10WA01,General,0402,0.1,+/-0.05[pF],,100,C0G,,100[MHz]-20[GHz],
GRM1555C1HR20BA01,General,0402,0.2,+/-0.1[pF],,50,C0G,,100[MHz]-20[GHz],
GRM1555C1HR20WA01,General,0402,0.2,+/-0.05[pF],,50,C0G,,100[MHz]-20[GHz],
GRM1555C2AR20BA01,General,0402,0.2,+/-0.1[pF],,100,C0G,,100[MHz]-20[GHz],
GRM1555C2AR20WA01,General,0402,0.2,+/-0.05[pF],,100,C0G,,100[MHz]-20[GHz],
GRM1555C1HR30BA01,General,0402,0.3,+/-0.1[pF],,50,C0G,,100[MHz]-20[GHz],
GRM1555C1HR30WA01,General,0402,0.3,+/-0.05[pF],,50,C0G,,100[MHz]-20[GHz],
GRM1555C2AR30BA01,General,0402,0.3,+/-0.1[pF],,100,C0G,,100[MHz]-20[GHz],
GRM1555C2AR30WA01,General,0402,0.3,+/-0.05[pF],,100,C0G,,100[MHz]-20[GHz],
.....
.....
....
```

Param Name

Param Prompt

Part Specifications

- ■ Model Files
  - ❑ Spice (.ckt)
  - ❑ Sparam (.s2p)

When the SMDs are being translated, the CDF properties are generated from PTF. Once SMD components are imported into the schematic, they are translated as TILPs in the layout.

### *Related Topics*

Create Technology File

Import Technology File

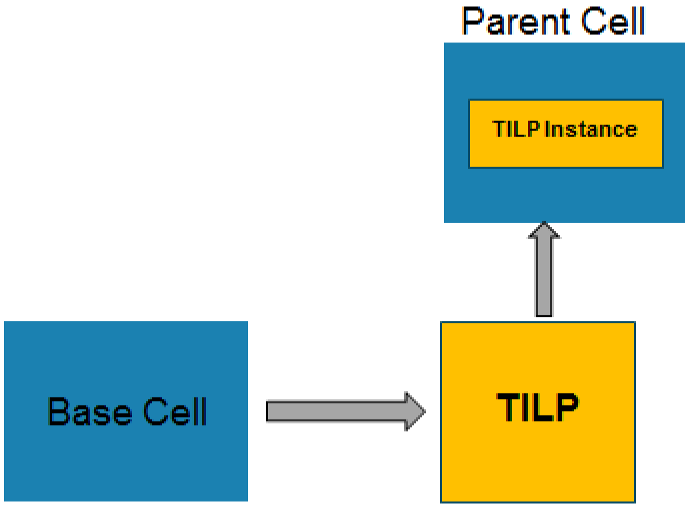
Update Technology File


© 2023 Cadence Design Systems, Inc.

# Die/TILP Instantiation

You can instantiate die footprints of ICs in the package layout from package library. IC/die footprint is the cellview in Virtuoso representing an IC design that includes the layout, schematic, and symbol. Exported die is the cellview that represents the footprint of the Die/ IC/package, which will be instantiated in the package design. It contains an abstract, TILP, schematic, and symbol view. IO cell is also instantiated in the die/IC layout to represent its external connectivity.

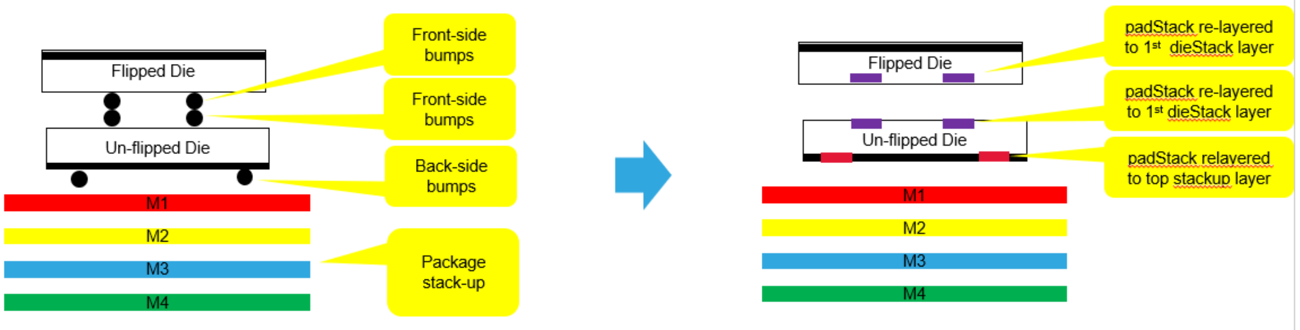
The package layout cells (TILP footprints) that can be designed, much before the layers are identified, by using the SMDs or exported dies through a technology file created from a `.sip` file. TILPs are created from the die symbols or package symbols and are added to the libraries as die TILP, SMD TILP, package TILP, and so on. The die or package symbols are created from the footprint terminals of a base layout cell. Subsequently, the TILP is instantiated in a parent cell. It can be imported into the parent cell through an export die library. Before creating TILPs, ensure that the die abstract that is used at the package level is accurate.





© 2023 Cadence Design Systems, Inc.

Die TILPs support rotation and mirroring. You can attach die TILP on the package layer in various positions. It could be unflipped, flipped, or mirrored. They can be added on the package substrate at an angle.

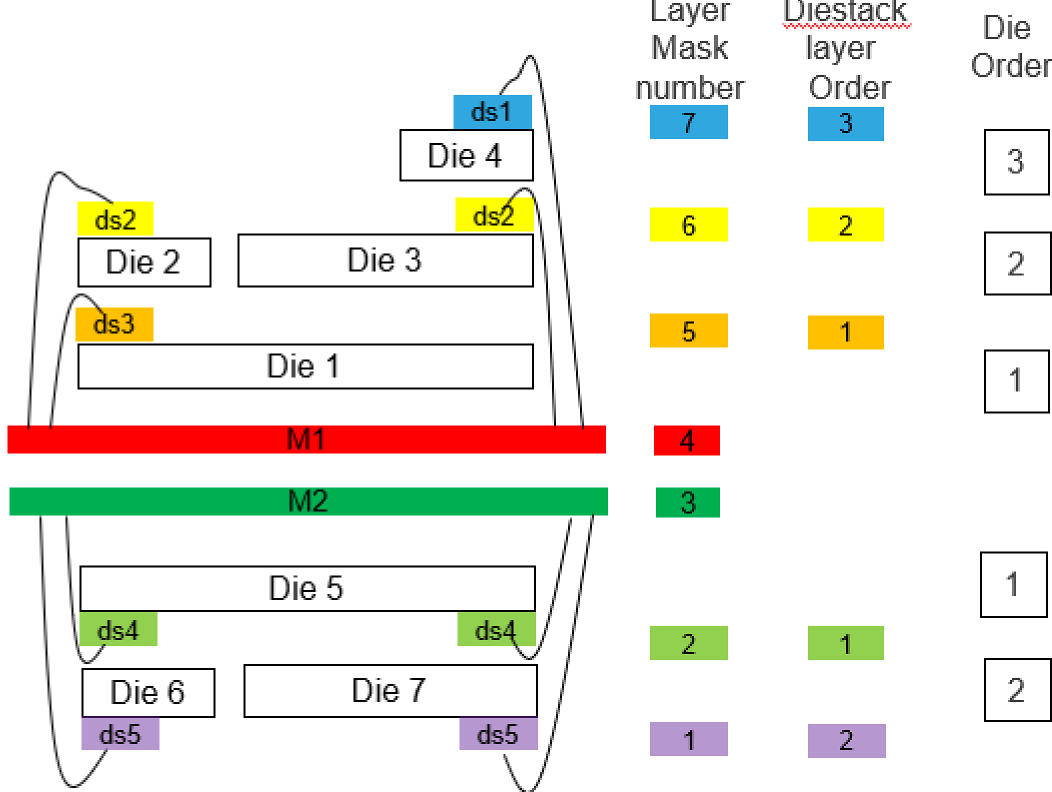



There is also a possibility of applying a shrinking factor to reduce boundary and transform the padStack. Following are some terms associated with placing multiple dies on the package substrate:

- Die order- It is the order of a die in a die stack on the package substrate. It could be a stack of package too.
- Die stack layer- It prevents 2D connectivity extractor to report false short between overlapping padStacks.


© 2023 Cadence Design Systems, Inc.

- Die stack layer order- Based on the layer mask number, the diestack layer order is the position of the diestack layer from the metal layer. The diestack layer order is not saved in a techFile but automatically derived from the layer mask number.




Exporting the die is an essential step before creating TILPs to be used in the package layouts but you need to prepare the die data before die export.

***Related Topics***

Die Export Preparation

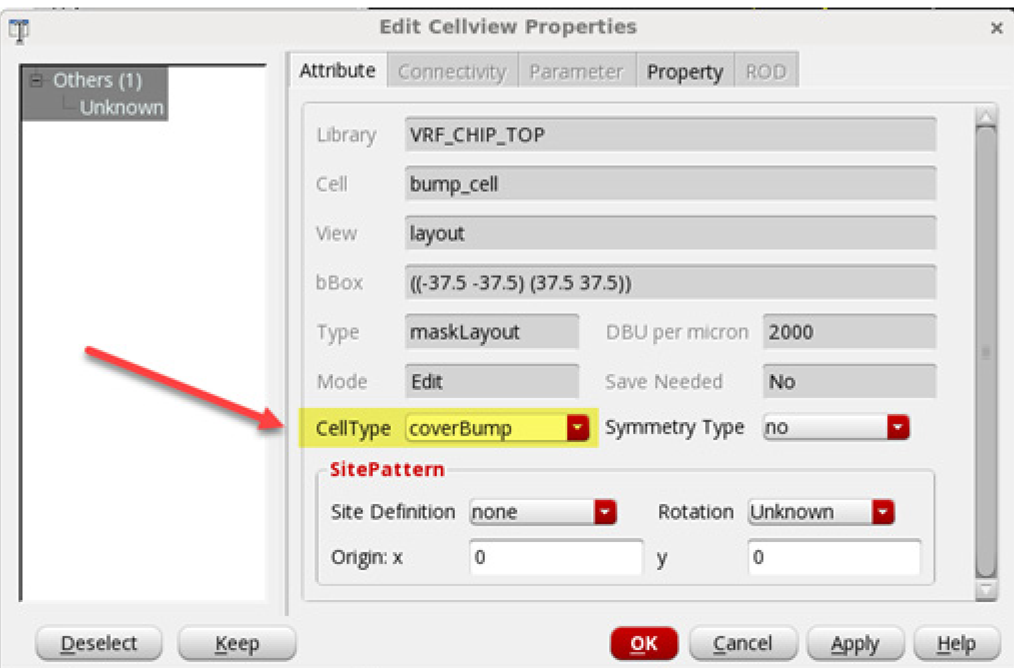
Exporting Dies

# Die Export Preparation

To ensure that the die abstract is accurate before using it in the package, you need to prepare the die. There are three important requirements:

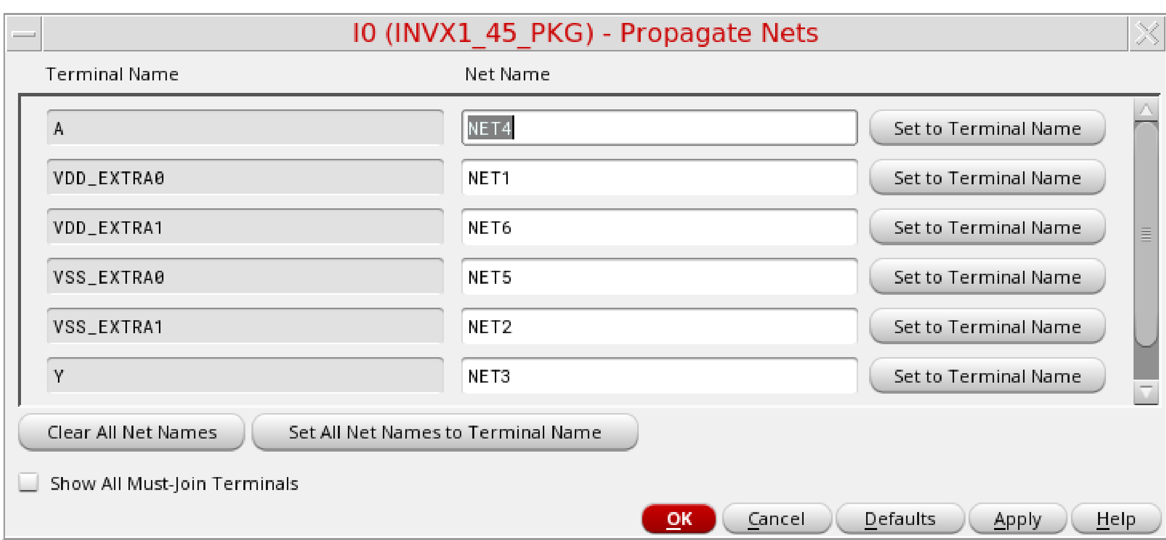

© 2023 Cadence Design Systems, Inc.

- Set the *CellType* to `coverBump` for the cells that contain the bump instances.




**Note:** For level 1 bump instances, the attributes, parameters, and properties that can change the geometry or placement of the instance are disabled.

- Update the connectivity information for the bump cells.




- Define `prBoundary` at the top level of the design.

Cells that are not defined as `coverBump` are not extracted while exporting the die.


© 2023 Cadence Design Systems, Inc.

***Related Topics***

Die/TILP Instantiation

Exporting Dies

# Exporting Dies

Video

To find out more about the process of creating TILPs and the various types of views that are generated, see Creating a TILP by Exporting the Die.

Video

To find out more about the process of exporting a shape-based die, see Exporting the Shape-Based Die.

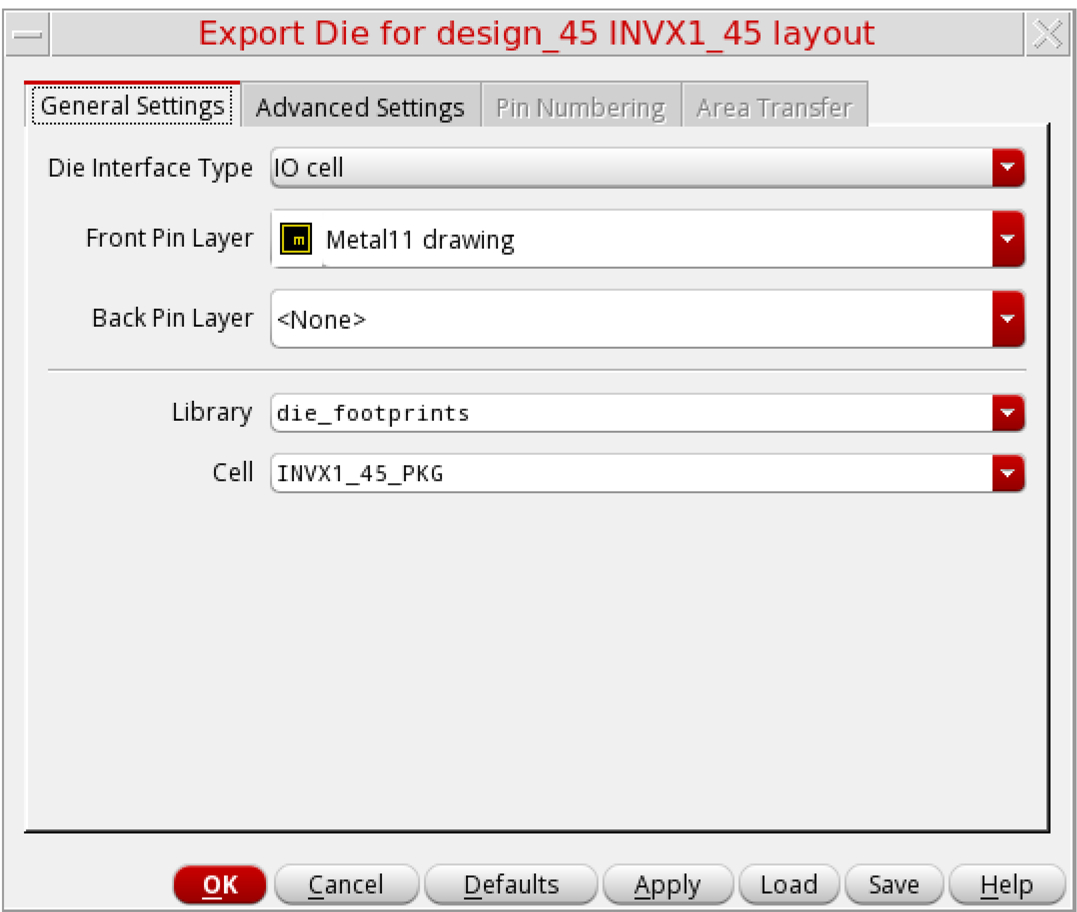
This is the step in the flow where the IC/die footprint is handed over to the package designer. By exporting the die, you can create technology-independent abstraction, which enables Edit-in-Concert, layout versus abstract (LVA) checks, and cross-fabric simulation using die schematic and model-based simulation for dies.

To export a die:

1. Open the IC/die layout in Layout MXL.


© 2023 Cadence Design Systems, Inc.

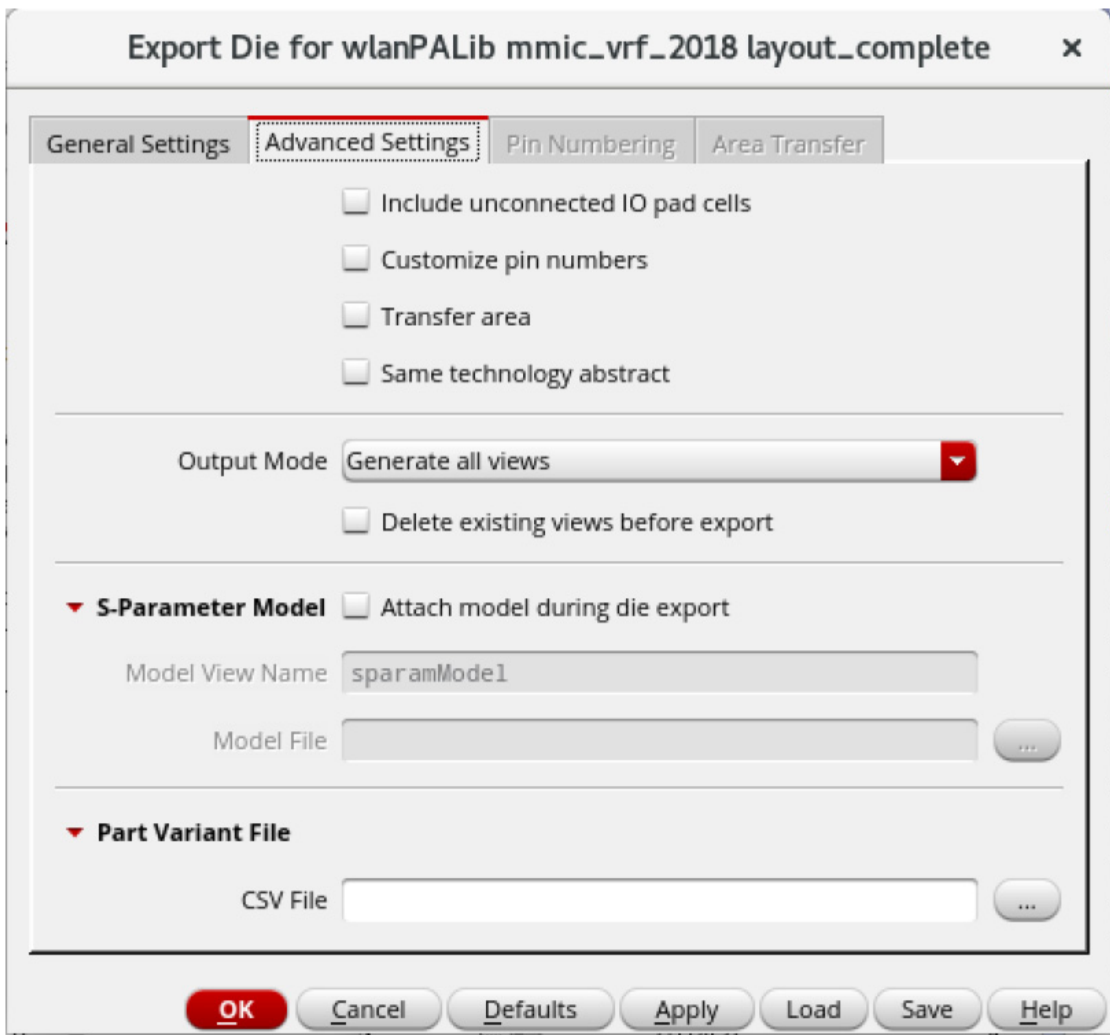
**2.** Click *Module – Export Die*. The Export Die Form is displayed.




**3.** On the *General Settings* tab, specify the *Front Pin Layer* while defining the inputs for exporting the die. Most of the options are already set by default in the form.


© 2023 Cadence Design Systems, Inc.

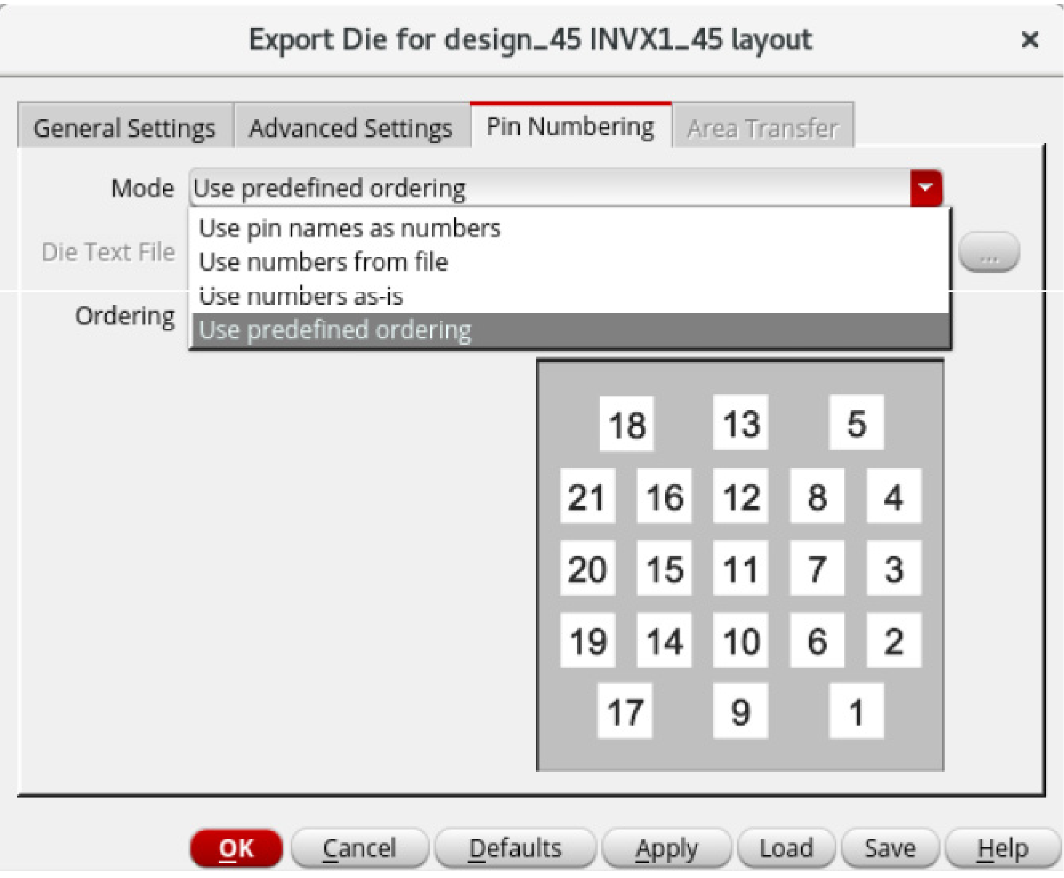
4. On the *Advanced Settings* tab, specify the options for the die abstract.




5. Select *Customize pin numbers* to enable the *Pin Numbering* tab.
6. Select *Transfer area* to enable the *Area Transfer* tab.


© 2023 Cadence Design Systems, Inc.

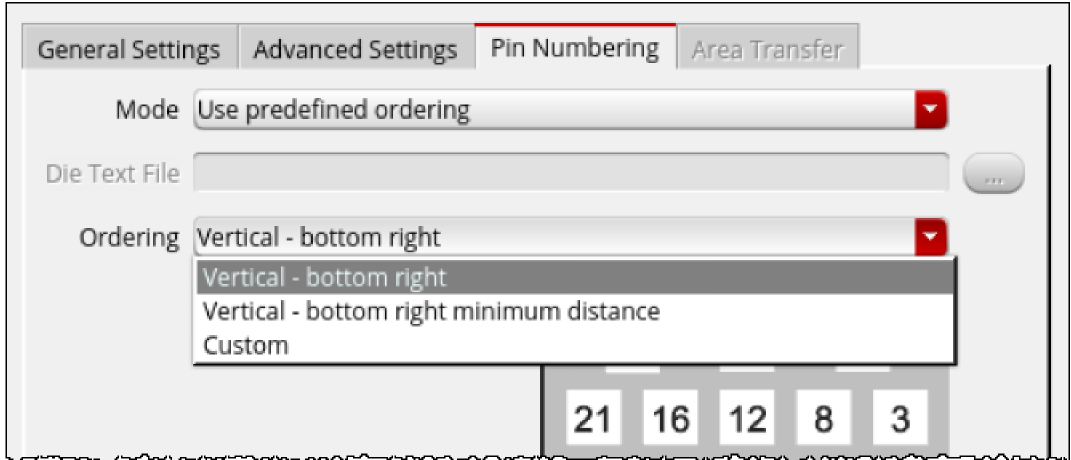
7. On the *Pin Numbering* tab, select the required option from the *Mode* drop-down list to generate the pin numbers for a die. The pin names in a footprint view are usually numbers, therefore, they are referred to as pin numbers.




8. Specify a value for *Die Text File* if the *Use numbers from file* option is selected for *Mode*.


© 2023 Cadence Design Systems, Inc.

9. Select the required option from the *Ordering* drop-down list if the *Use predefined ordering* option is selected for *Mode*.




10. On the *Area Transfer* tab, specify an *Area Transfer File* that includes rules to add some additional shapes, such as logos, in an abstract view.

11. Click *OK* to export the die.

## Edit Die Layout and Abstracts

After exporting the die, you can edit the die layouts and die abstract TILPs that are part of the same package by using the Edit-in-Concert feature. The signal types are also propagated from an IC layout to the die abstract during the die export.

Die export includes the following tasks when `IO cell` is chosen as the *Die Interface Type* in the Export Die form:

- Extracts the IO Cells (bump/pads) from a given die/IC layout hierarchy and ensures that any duplicate pin names are made unique. For example, if there are two pins named as `VSS<1>`, they are mapped as `VSS_EXTRA0<1>` and `VSS_EXTRA1<1>`.


© 2023 Cadence Design Systems, Inc.

- Connects the front-end hierarchy of the exported die to the front end hierarchy of the other die. You can to access the IC schematic hierarchy when the die is instantiated in a package.

The summary report generated after die export contains information about whether the IO cells are connected, unconnected, or invalid based on the *Front Pin Layer* value. It also warns about any IO Cells that contain layers different from the *Front Pin Layer* value.

Die export includes the following tasks when `Shape with overlapping label` is chosen as the *Die Interface Type* in the Export Die form:

- Exports die for shape-based layouts.
- Accepts pin layer purpose pair and label layer purpose pair only for the front and back side.
- Extracts pins based on the shapes. In shape-based layouts, a shape is exported as a pin with a name the same as the text of the overlapping label. For a shape to be exported as a pin,
    - It must be a rectangle, polygon, or ellipse.
    - The origin of the label must overlap with the shape to be exported.

After exporting the die, a summary report is printed, which describes the number of labels found with and without overlapping shapes for the front side and back side layers.

```
Labels with overlapping shapes:

                Front Side: 3

                Back Side: 2

Labels without overlapping shapes:

                Front Side: 1

                Back Side: 0
```

**Note:** LVA, Edit-in-Concert, and fixer do not work with die abstracts that have been exported using *Export Die* for the shape-based IC layout.

#### *Related Topic*

Export Die Form


© 2023 Cadence Design Systems, Inc.

vrfExportLayoutSkill

Die/TILP Instantiation

Die Export Preparation

## Creating and Verifying Integrity 3D-IC Compatible Die Abstracts

The die abstract needed for the Virtuoso Integrity 3D-IC flow is similar to the die abstract created by the usual Export Die command because it contains all the required bump information.

A die abstract for the Virtuoso Integrity 3D-IC flow has the following features.

- Uses the same technology information as an analog IC, which means that the bump master from the IC layout can be used to create bump instances in the die abstract.
- Bumps are connected to the top-level nets or terminals through instance-terminal-based connectivity.
- There are physical-only bumps and one top-level net can be connected to multiple bumps. Bus and terminal definitions are transferred to the die abstract.
- Die abstract follows the centering of the IC layout and is not necessarily centered at the origin.
- All instances and terminals follow the same naming convention as in the IC layout.

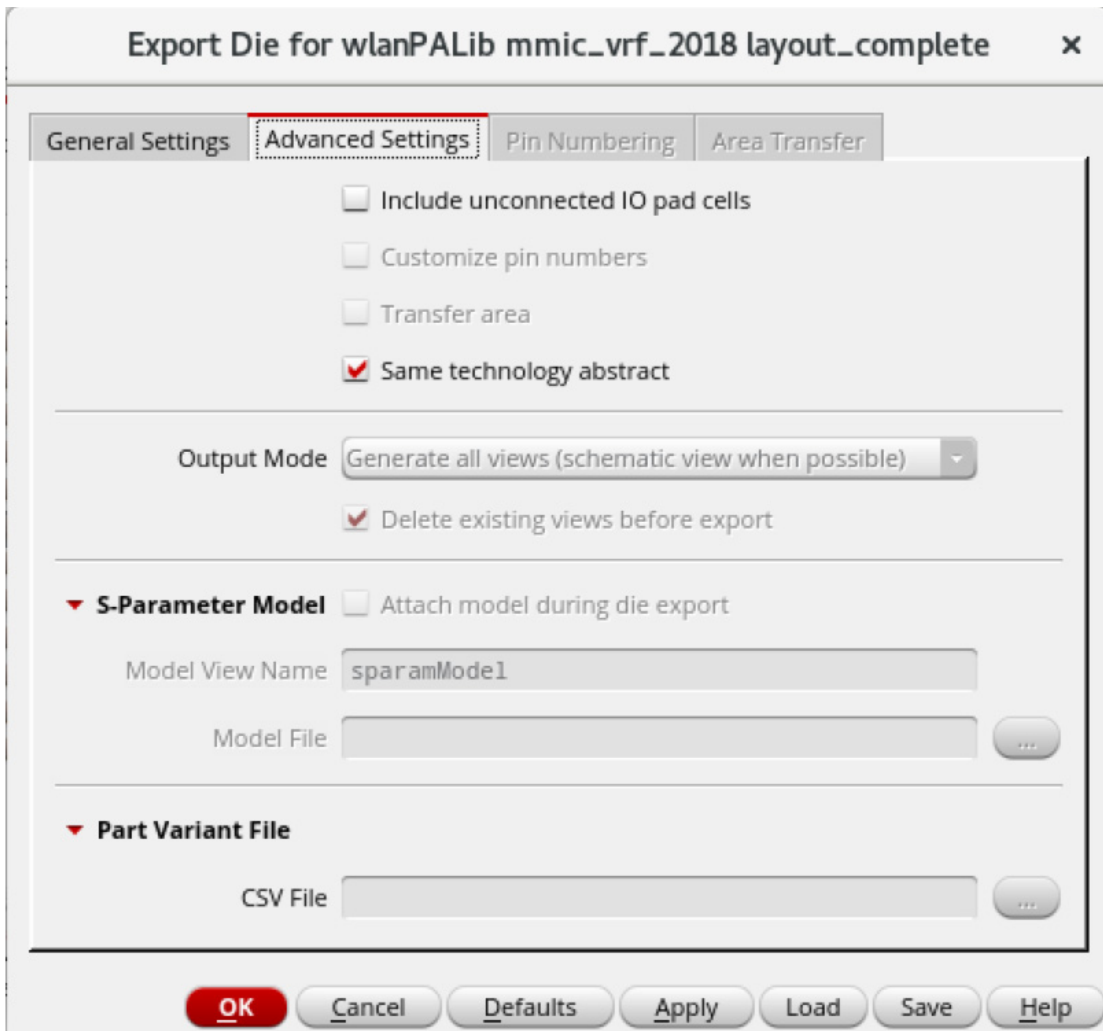
To create a die abstract:

1. Open the IC/die layout in Layout MXL.
2. Click *Module – Export Die*. The Export Die Form is displayed.

   Ensure that the *Cell* name for the die abstract is same as the IC layout cell name.


© 2023 Cadence Design Systems, Inc.

3. On the *Advanced Settings* tab, select the *Same technology abstract* option for the die abstract.




Most of the other options on the tab for creating a TILP are disabled because selecting the option creates only the die abstract view. The TILP super-master, schematic, and symbol views are not created because the abstract will not be used for package instantiation in Virtuoso.

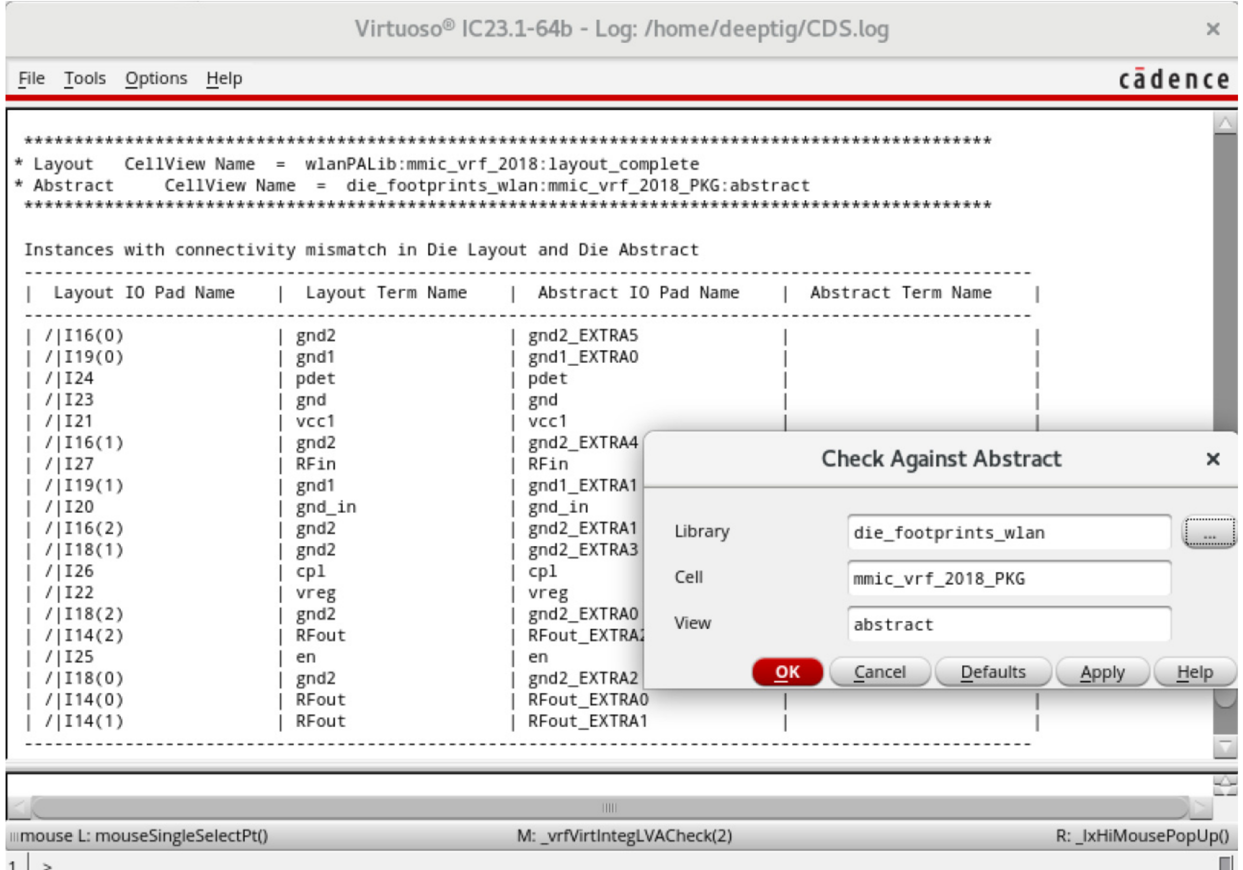
4. Click *OK* to export the die.

To check and update the IC layout with the updated die abstract from iHDB:

1. Click *Module – Compare Layout With Abstract – Check*.


© 2023 Cadence Design Systems, Inc.

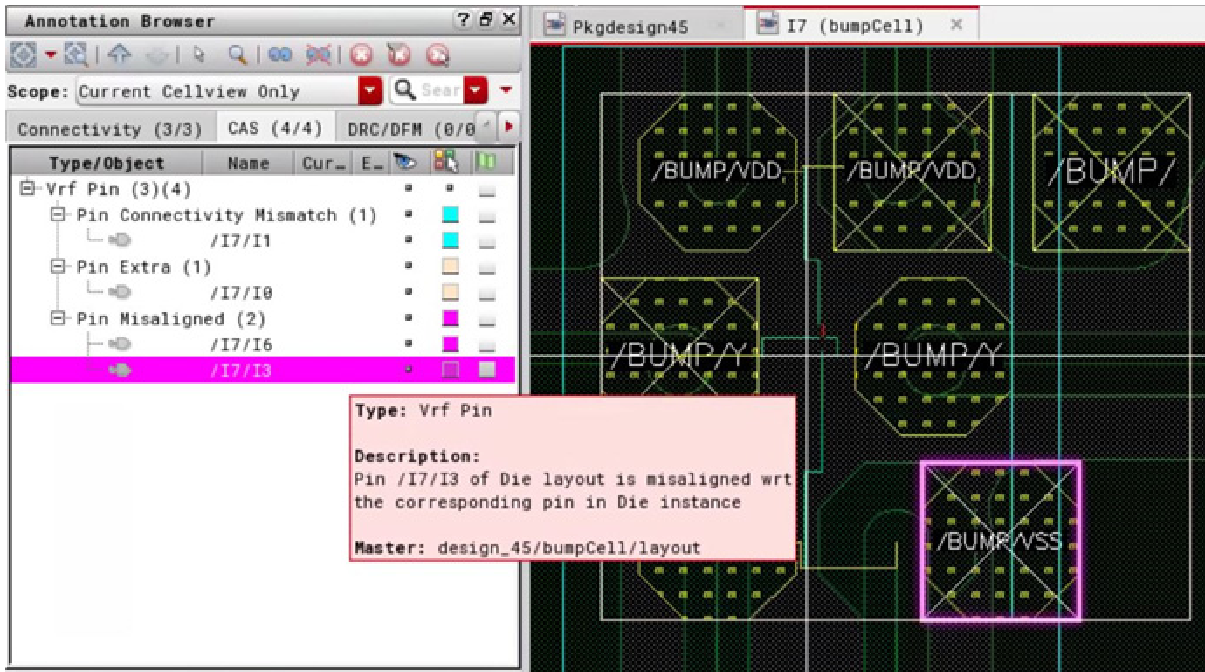
The Layout Versus Abstract (LVA) checker compares the IC layout with the die abstract and generates markers showing the differences. You can run the checker on all bumps or only the selected bumps in the IC layout.





© 2023 Cadence Design Systems, Inc.

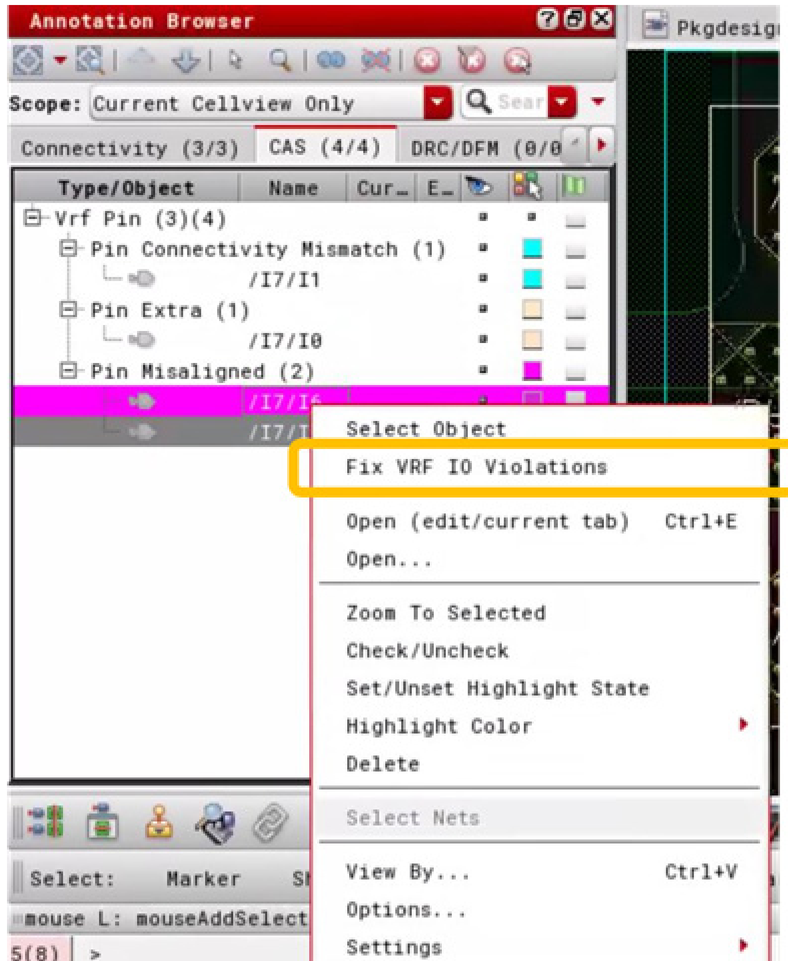
Markers show the positions of violations, with more complete information provided in the Annotation Browser.





© 2023 Cadence Design Systems, Inc.

2. Click *Module – Compare Layout With Abstract – Fix*. Alternatively, click *Fix Virtuoso RF IO Violations* from the shortcut menu in the Annotation Browser.




The command fixes the violations reported by the *Check* command by modifying the bumps in the IC layout as per bumps in the die abstract. This command is also selection based, which means violations specific to the selected bumps are fixed and specific bump modifications are brought into the IC layout. If no bumps are selected, all violations are fixed and all changes from the die abstract are brought into the IC layout.

### *Related Topic*

Virtuoso Integrity 3D-IC Flow

sameTechnologyAbstract

Export Die Form

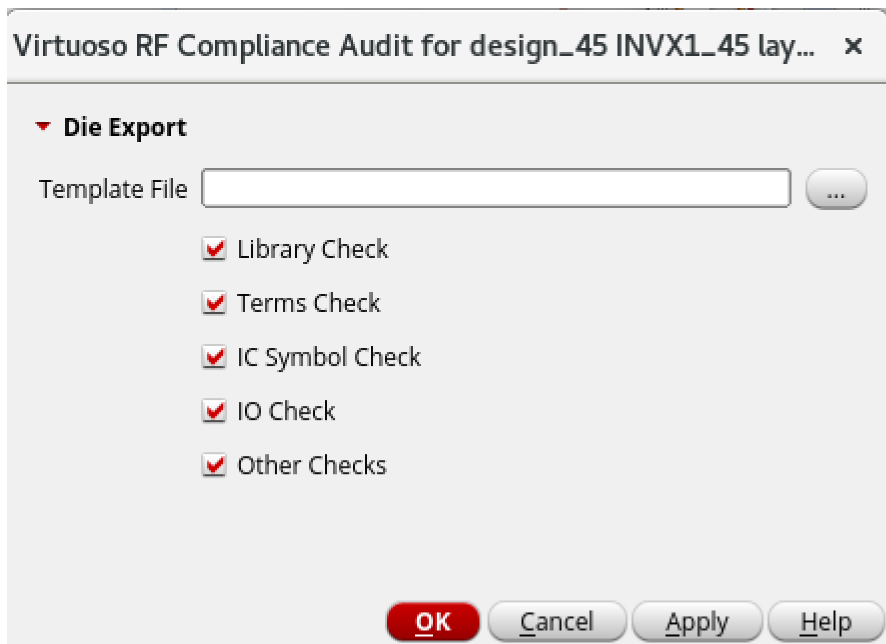

© 2023 Cadence Design Systems, Inc.

vrfExportLayoutSkill

Die/TILP Instantiation

# Die Audit

When exporting a die for the first time on an IC layout, it is not uncommon to find errors in your design. One of the reasons might be that the design has not been set up correctly for the Virtuoso RF Solution requirements. The audit functionality is a checker utility that reports all errors or warnings for an export die function on an IC layout before actually exporting a die. You can click *Module – Compliance Audit* to open the Virtuoso RF Compliance Audit form.During the audit, you are provided the same environment or settings as while running actual die export functionality to ensure that the same issues can be found and fixed earlier in the cycle. You can specify the template file that is exported from the Export Die form to give the desired settings for die audit.




The template file must include the following to perform an audit:

- Die interface type
- Front label and back label layer-purpose pairs (if shape-based)
- Front and back layer-purpose pairs
- Pin numbering-related values


© 2023 Cadence Design Systems, Inc.

- Output mode values (for partial die export)

There can be various violations reported during abstract or schematic creation, such as:

**Terminals-related**

- Top-level terminals that are not connected to any instance.
- Top-level pin or terminal direction is not input, output, or inputOutput.
- Top-level terminals are connected to an instance but the instance master does not have `coverBump` or `pad` specified as wirebond cell type
- Terminal is connected to a pad instance that has master with `coverBump` or `pad` cell type specified, however, the shape inside the pad is not on the pin layer specified in the Export Die form.

**Library-related**

- No cells of type `coverBump` or `pad` have been found in the library.

**Other Checks**

- Top cell does not have a PR boundary.
- Die export needs the design to be marker free but the design has markers in the layout.

**IC Symbol-related**

- IC symbol is missing.

**IO Check-related**

- Design has overlapping pads.
- Mismatched terminals found between a die symbol and die Layout.
- The BUMP_TO_PIN property is found.
- Target label does not match any terminal in die symbol and vice versa.
- No valid target label is found in a design.
- Multiple shapes on a specified layer overlap with a label.
- Multiple different labels on a specified layer overlap with a shape.


© 2023 Cadence Design Systems, Inc.

You can view the following report that is created in the working directory after the die audit has been performed. It reports all bumps cells even if Unconnected pins are set to off.

DieAudit_IOReport.txt - Notepad

File Edit Format View Help

```
Die Audit IO Report
-------------------
Cell View: wlanPALib mmic_vrf_2018
Date: Apr 28 10:18:15 2023


IO Cells
--------
Total: 2
Cell 'gpdksige50:bump2_shielded:layout' has type as 'coverBump'
Cell 'gpdksige50:bump2:layout' has type as 'coverBump'



Hierarchy Tree
--------------
Total 19
```

### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit

dieTemplateFile

icSymbolCheck

ioCheck

libraryCheck

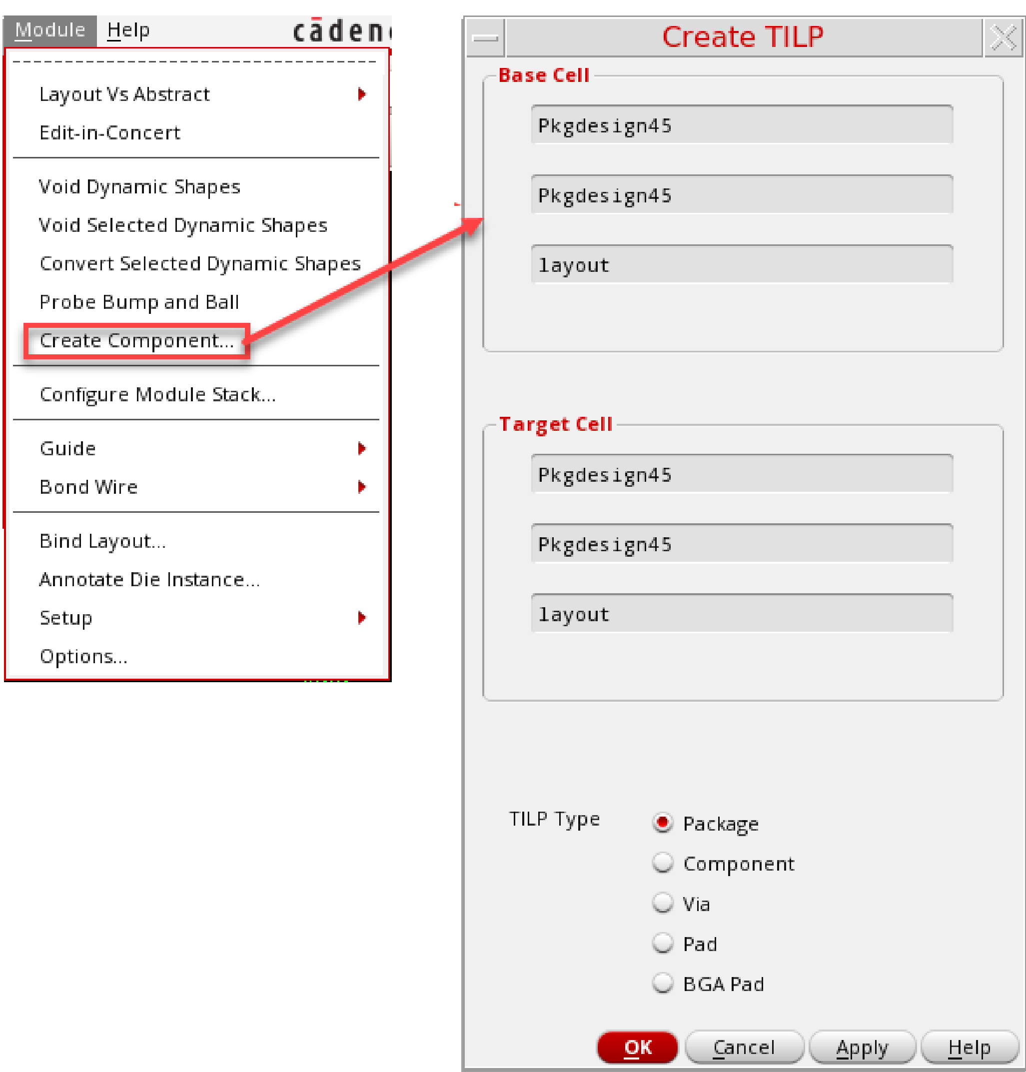
otherChecks


© 2023 Cadence Design Systems, Inc.

# Creating TILPs

You can create padstack vias, surface-mounted devices (SMD) pads, embedded components, package TILPS, or die TILPs from a package layout by using the GUI option in the Virtuoso RF Solution. It instantiates the base cell that you want to use as the default footprint cell for the TILP. The base cellview of a component in the layout is created and thereafter, the corresponding TILP view is created by using the *Module* menu as shown in the following steps.

1. Click *Module – Create Component* to open the Create TILP form. If the TILP view already exists, it is overwritten.




2. Place the resulting component in the package layout similar to a scenario when the component is being imported from Allegro.


© 2023 Cadence Design Systems, Inc.

*Caution*

***The TILPs created in the new version of Virtuoso Studio are not backward compatible, that is, they cannot be used in the older versions of Virtuoso Studio.***

***Related Topic***

Create TILP Form

## Alternate Footprints in TILPs

The cell names of footprints in a TILP can be specified as not allowed, must be included, or alternate footprints based on a specific syntax. You do this either in the `altFootprintCellNames` column in a part.csv file or in the *footprintALTCellName* text field in the Edit Instance Properties form.

Use the following guidelines to specify placement of certain footprints or parts for use in TILPs:

- The `altFootprintCellNames` column in a part.csv can specify that certain parts are not allowed in particular placements by specifying the placement as `NONE`. For example, `(TOP:CAP1005N;BOTTOM:NONE)` where `BOTTOM:NONE` means that this part is not valid on the `BOTTOM` placement.
- There is no change in the placement of a footprint cell when `altFootprintCellNames` is not formated as `(TOP:1005;BOTTOM:SPDT;EMBEDDED:LGA)`.
- If `altFootprintCellNames` is shown as `(TOP:1005;BOTTOM:SPDT;EMBEDDED:LGA)` and the `Stackup Offset` parameter is not `0`, the cell name specified by the `EMBEDDED` keyword is used as the footprint cell. For example, `LGA` is used as the cell name in this example.
- If `altFootprintCellNames` is shown as `(TOP:1005;BOTTOM:SPDT;EMBEDDED:LGA)` and the `Mirrored` parameter is `nil`, the cell name specified by the `TOP` keyword is used as the footprint cell. For example, `1005` is used as the cell name in this example.

  If the `Mirrored` parameter is `t`, the cell name specified by the `BOTTOM` keyword is used as the footprint cell. For example, `SPDT` is used as the cell name in this example.
- If a specific keyword is not specified at the instance place, that is, if `altFootprintCellNames` is shown as `(TOP:1005;BOTTOM:SPDT)` and the


© 2023 Cadence Design Systems, Inc.

`Stackup Offset` parameter is `2`, the `footprintCellName` parameter is used as the footprint cell.

If the given cell name does not allow retrieval of an existing cell in a TILP, the Pcell evaluation fails and a marker is dropped in the sub-master cellview. The given cell name for placement overrides the existing `footprintCellName` parameter.

# TILP Versions

The version of a TILP for a component is important to understand the associated parameters and whether it is compatible with the given version of the tool. TILPs are backward compatible, that is, a design with an old version of TILP can be opened without any error. TILP evaluation is preserved by ensuring same geometry and connectivity is being generated in each TILP master along the new version of the tool.

You can upgrade the TILP version by regenerating the TILP to the most recent version using vrfUpdateTILPVersion. Use the `vrfCheckTILPVersion` and `vrfUpdateTILPVersion` SKILL functions to check and update the TILP versions.


© 2023 Cadence Design Systems, Inc.

Here is the table that shows the TILP versions, types of TILPs, and related parameters.

| TILP Versions | TILP Types | Related Parameters |
|---|---|---|
| **Version 1** | Die | mirrored |
| | | order |
| | | offset |
| | | shrinkFactor |
| | | x_thermalShrinkFactor |
| | | y_thermalShrinkFactor |
| | | rotation |
| | | baseCellCellName |
| | | baseCellViewName |
| | | parentCellLibName |
| | Package | rotation |
| | | mirrored |
| | | baseCellLibName |
| | | baseCellCellName |
| | | baseCellViewName |
| | SMD | JEDEC_TYPE |
| | | jedecLibNames |
| | | ALT_SYMBOLS |
| | | mirrored |
| | | offset |
| | | rotation |


© 2023 Cadence Design Systems, Inc.

| **TILP Versions** | **TILP Types** | **Related Parameters** |
| --- | --- | --- |
| | OtherPadStack | mirrored |
| | | backSide |
| | | flipped |
| | | order |
| | | offset |
| | | shrinkFactor |
| | | rotation |
| | | baseCellViewName |
| | | parentCellType |
| | | parentCellLibName |
| | ViaPadStack | rotation |
| | | baseCellViewName |
| **Version 2 (ICADVM18.1 ISR5 onwards)** | Package | Order |
| | | Offset |


© 2023 Cadence Design Systems, Inc.

| TILP Versions | TILP Types | Related Parameters |
|---|---|---|
| **Version 2 (ICADVM18.1 ISR10 onwards)** | Die | footprintLibNames |
| | | footprintCellName |
| | | FootprintViewName |
| | | altFootprintCellNames |
| | | Removed baseCellCellName |
| | | Removed baseCellViewName |
| | | Removed parentCellLibName |
| | Package | Removed baseCellCellName |
| | | Removed baseCellViewName |
| | | Removed baseCellLibName |
| | SMD | footprintLibNames |
| | | footprintCellName |
| | | FootprintViewName |
| | | altFootprintCellNames |
| | | jedecLibNames |
| | | RemovedJEDEC_TYPE |
| | OtherPadStack | footprintLibNames |
| | | footprintCellName |
| | | FootprintViewName |
| | | altFootprintCellNames |
| | | Removed baseCellViewName |
| | ViaPadStack | footprintLibNames |
| | | footprintCellName |
| | | FootprintViewName |
| | | altFootprintCellNames |
| | | Removed baseCellViewName |


© 2023 Cadence Design Systems, Inc.

| TILP Versions | TILP Types | Related Parameters |
|---|---|---|
| **Version 2 (ICADVM20.1 onwards)** | All TILP types markers are propagated from the footprint cell to the sub-master0 | |
| **Version 3 (ICADVM20.1 ISR15 onwards)** | Die | Marker propagation from footprint cell to instance master |
| | | insTerms |
| | | scribeNorth |
| | | scribeSouth |
| | | scribeEast |
| | | scribeWest |
| | Package | footprintLibNames |
| | | footprintCellName |
| | | FootprintViewName |
| | | altFootprintCellNames |
| | | insTerms |
| | | Order/Offset |
| | SMD | insTerms |
| | | Marker propagation from footprint cell to instance master |
| **Version 4** | Package | Marker propagation from footprint cell to instance master |

### *Related Topics*

vrfCheckTILPVersion

vrfUpdateTILPVersion


© 2023 Cadence Design Systems, Inc.

# Replacing Pads

When creating a die abstract by exporting a die in Virtuoso Studio, the pads are created based on the IC layout shapes. However, these pads do not represent soldering at assembly level. It is necessary to replace these pads by the real pads at the package-design level, either in Virtuoso Multi-Technology or Allegro platform.

Before recreating pad stacks, it is recommended to review the shrink factor of the die. When a die has a shrink factor, the pads are also shrunk with the die size. However, the pads used for replacement exist in their manufacturing size. Applying the shrink factor to these pads leads to smaller pads than expected. Therefore, apply the shrink factor to padstacks only if they are on a die stack layer. If they are on a substrate layer, do not apply the shrink factor.

You can replace a padstack cell in a footprint with another padstack cell from the same or another library.

To replace a padstack cell:

1. Click *Module – Replace PadStack* to open the Pad Stack Replacement form.

2. In the *Source PadStack* drop-down list, select the padstack cell to be replaced.
3. Select from the *From Current Footprint Library* or *Copy From Another Library* check boxes to specify the target padstack cell.
4. Click *OK*.

   The padstacks are replaced in the footprint of the selected instance. If *Copy From Another Library* is selected, the padstack cell is also copied to the footprint library.


© 2023 Cadence Design Systems, Inc.

### *Related Topics*

Pad Stack Replacement Form


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# 4

# Package Schematic Creation

The package schematic contains the symbols of SMDs from SiP and the die symbol obtained from die export. It can contain more components, such as LGA, embedded components, and Transmission Lines (TLines). The package schematic contains the IC and package portions of the design that are represented, designed, and verified within a single environment. It eliminates the tedious and error prone process of maintaining multiple schematic databases. It lets you Edit-in-Concert the package and IC in the design flow. For example, you can explore the possibility of implementing a selected passive component of the IC, verify the performance by running a simulation, and decide about how best to implement the component. In some cases, a network of passive components may be better implemented across the boundary of the die placed on a package. This is true for RF modules, where the optimum design of filtering and matching networks often leverages components on both the die and in the package substrate.


© 2023 Cadence Design Systems, Inc.

# Instantiating SMD Instances

To ensure that the top-level package design is derived from a package technology, create a package schematic inside a library where the technology library has the `package`, `module`, or `board` fabric. For example, a library created by importing a design from Allegro.

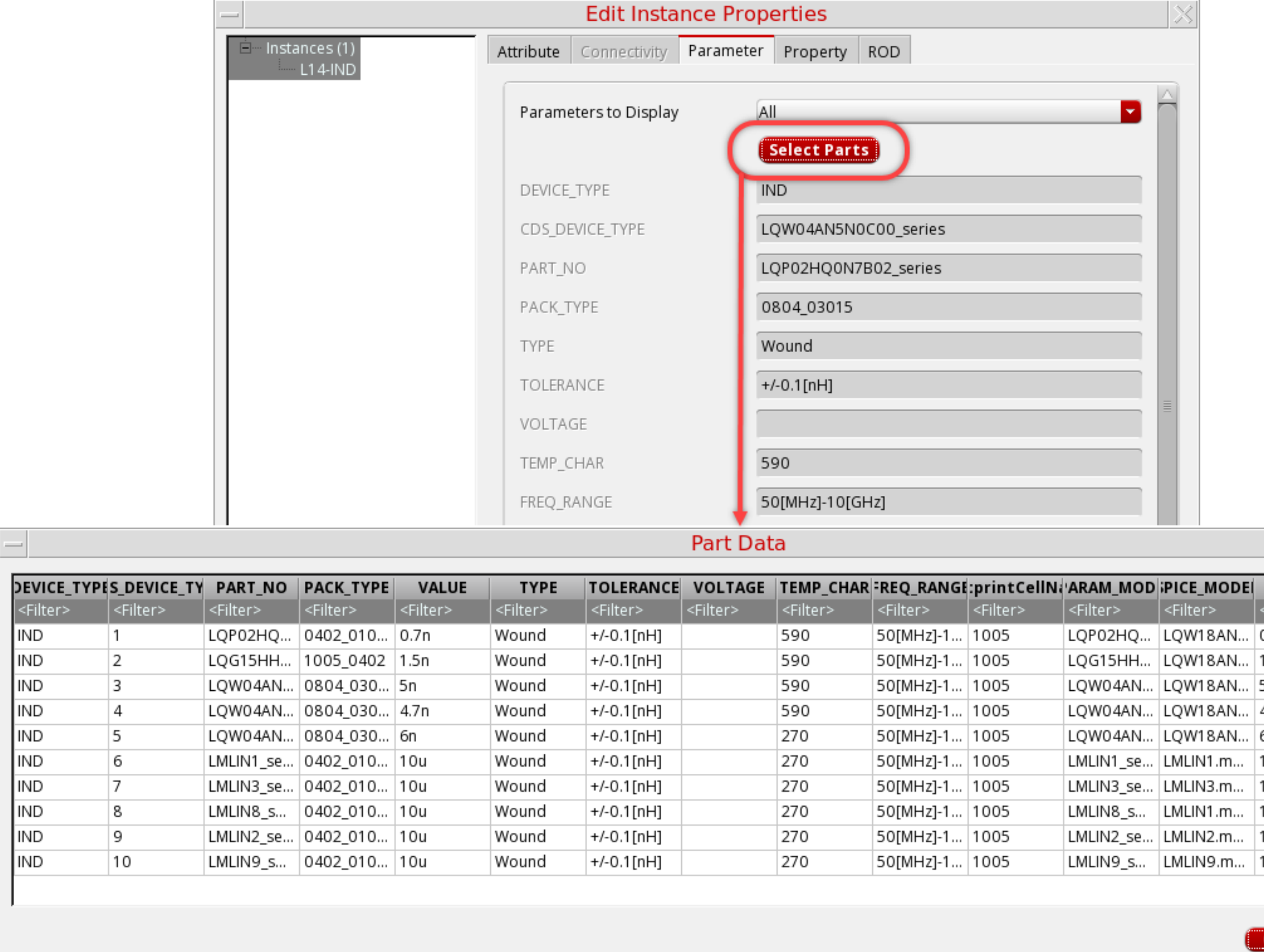
SMD components are found in the `smdLib`, a library created by importing SMD components from a CSV file. It is designed to be used in conjunction with a part table file (`*.ptf`) that lists the available components and their associated electrical characteristics and other useful information provided by the component manufacturer. The *Select Parts* button in the Edit Object Properties form displays the most useful electrical information of each individual SMD footprints. For example, for an inductor, it displays the inductance, Q or quality factor, self-resonance frequency, and tolerance. In addition, an associated S-parameter model file to be used for simulation is mentioned. Models are obtained from the manufacturer and their names are stored in the PTF.

To instantiate SMD instances:

1. Open the Edit Object Properties form for an SMD instance.
2. Click *Select Parts* in the Edit Object Properties form.


© 2023 Cadence Design Systems, Inc.

The Part Data dialog box appears. It displays the information from the Physical Part Table file (`.ptf` file). Each row in the Part Data dialog box corresponds to a physical component variant associated with the logical part you have selected in the schematic.




3. Apply filters to select the part variant.
4. Click *OK*. The instance can be added to the schematic.

### *Related Topics*

Instantiating BGA Instances

Instantiating IC Instances

Creating TLines Instances

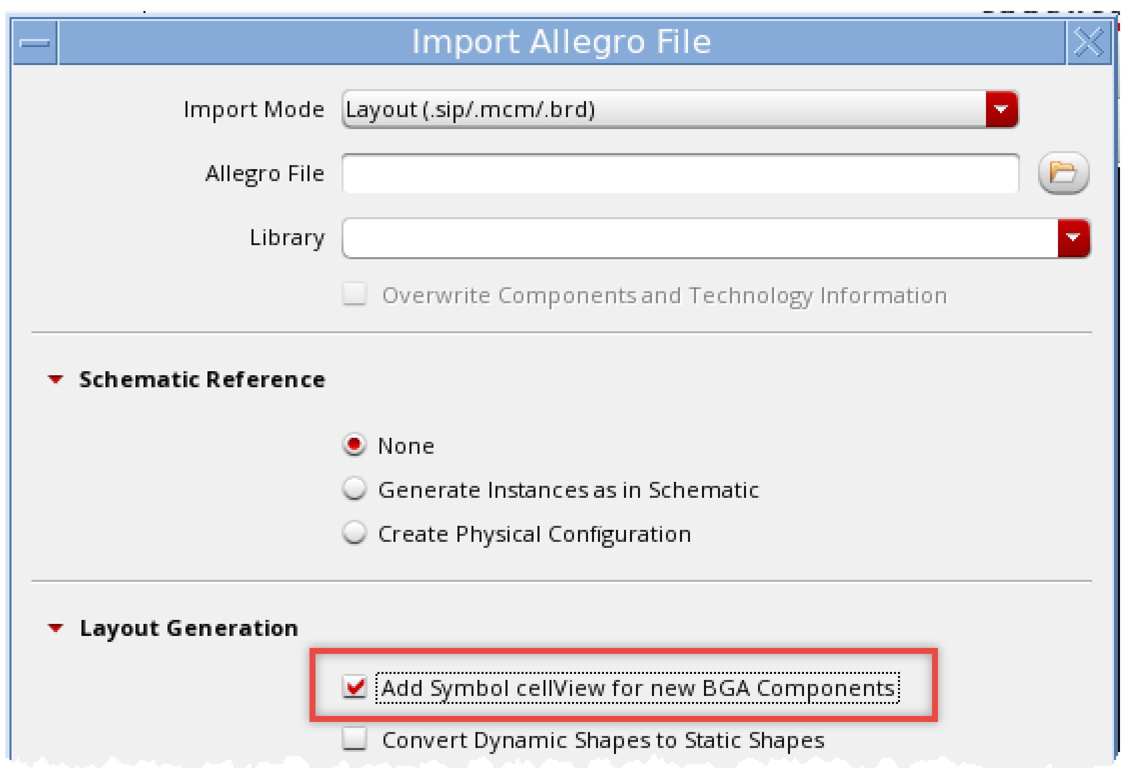

© 2023 Cadence Design Systems, Inc.

Instantiating TLine Instances

# Instantiating BGA Instances

Ball grid array (BGA) instances are a type of package instance. To instantiate BGA instances in a schematic:

1. Click *CIW – File – Import – From Allegro.* The Import Allegro File form opens.
2. Select the *Add Symbol cellView for new BGA Components* in the Import Allegro File form for creating the symbol while importing from Allegro.




3. Click *OK*. It creates a schematic symbol of the BGA, which can be instantiated in the schematic.

### *Related Topic*

Instantiating SMD Instances

Instantiating IC Instances


© 2023 Cadence Design Systems, Inc.

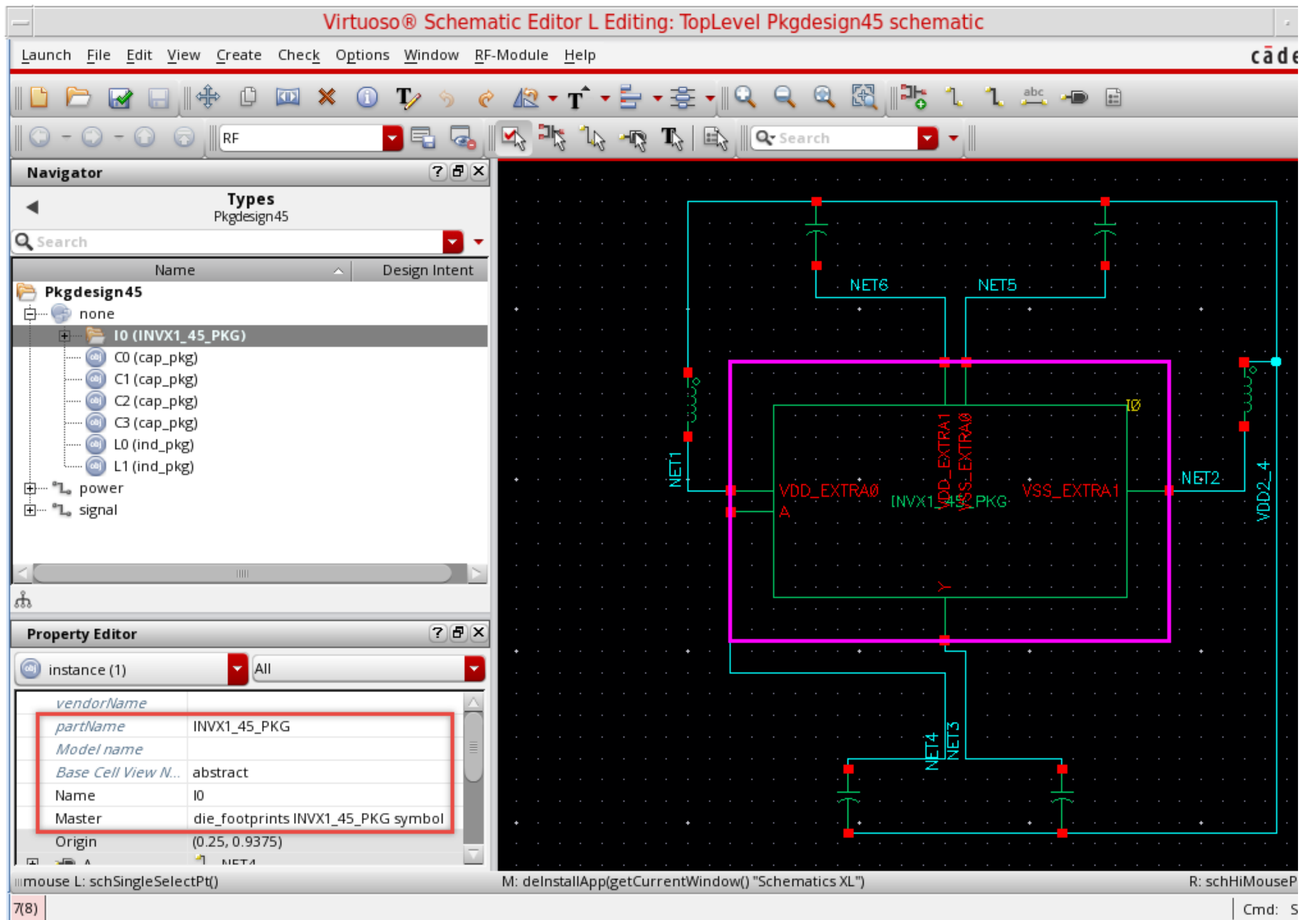
[Creating TLines Instances]

[Instantiating TLine Instances]

# Instantiating IC Instances

Update the package schematic in the package library by instantiating the die footprint generated while exporting the die. To instantiate IC instances:

1. Create a new cellview in the package library.
2. Instantiate the die symbol by using the Add Instance form.




3. Create connectivity for the instance.
4. Check and save the package schematic. The die symbol appears in the schematic as an instance.


© 2023 Cadence Design Systems, Inc.

***Related Topic***

Instantiating SMD Instances

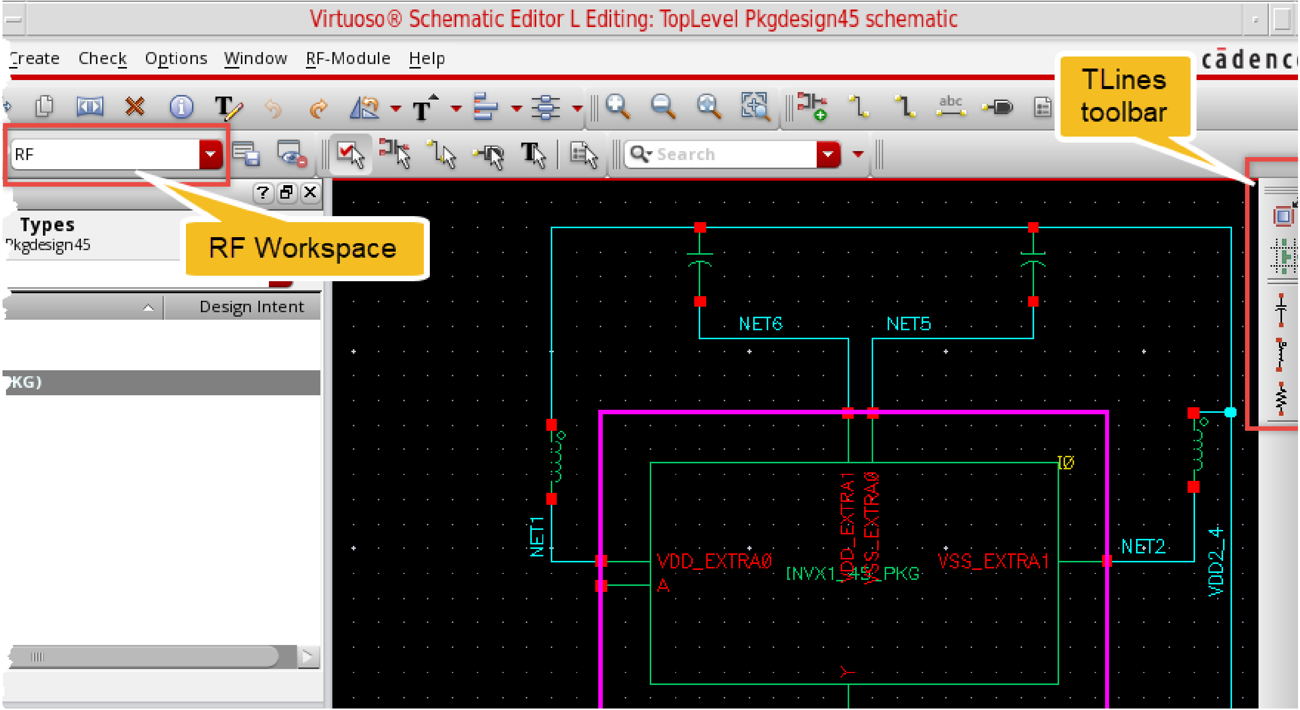
Instantiating BGA Instances

Creating TLines Instances

Instantiating TLine Instances

# Creating TLines Instances

The Virtuoso RF Solution provides a library that contains the symbol view, simulation view, and OA layout view for TLines. You can instantiate the symbol views and capture connectivity in the package schematic. Tline components are derived from rfTlineLib, which is a library of wideband-accurate transmission line models in multi-conductor microstrip and stripline configurations.




TLine instances can be added to the package schematic by using the RF toolbar available in the RF workspace. The RF toolbar lets you access the TLine components that can be placed in the schematic.


© 2023 Cadence Design Systems, Inc.

Each button on the toolbar opens the Add Instance form to enable you place the components on the canvas. All TLine components in a schematic can be found by using the Search toolbar or assistant with the keyword “tlines”. This is useful for cross selecting multiple instances to compare width and length parameters in the Property Editor.

Based on the frequency-dependent per-unit-length parameters calculated by a 2D quasi-static electromagnetic solver, the models are integrated in Virtuoso Analog Design Environment and accessible from standalone Spectre netlists.

To create stackup and net topology:

1. Inflate the existing schematic to preserve the design for accommodating space for transmission line insertion.
2. Create stackup to represent a partial slice through the technology stack and store material properties and cross section information.
3. Create net topology to provide a fast error-free method of creating and editing net topologies.

***Related Topic***

Create Stackup

Create Net Topology

Inflate

Instantiating SMD Instances

Instantiating BGA Instances

Instantiating TLine Instances


© 2023 Cadence Design Systems, Inc.

# Instantiating TLine Instances

`rfTlineLib` includes an interactive graphical stack-up editor for storing the substrate geometry and material properties. The TLine topologies that are used as TILPs in the Virtuoso RF Solution are shown in this illustration.

To instantiate TLines instances:

1. Instantiate the TLines TILP from `rfTlineLib`.The library contains the symbol, simulation, and OA layout view.
2. Specify TLine parameters, such as length and width.
3. Simulate the package schematic using the Tline simulation model.
4. Generate the package layout by using Layout Suite MXL.

### *Related Topic*

Instantiating SMD Instances


© 2023 Cadence Design Systems, Inc.

Instantiating BGA Instances

Creating TLines Instances

Instantiating TLine Instances

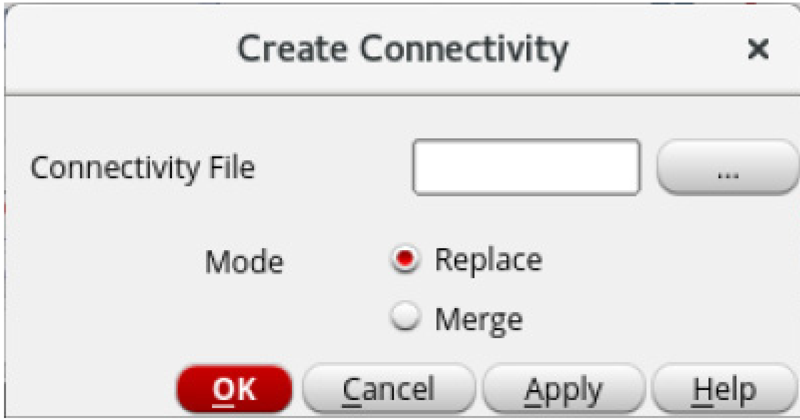
## Creating and Saving Connectivity Information in Package Schematic

You can create and save connectivity information for the package schematic in your production design flow. It allows to save the connectivity information from another schematic view and restore it to a new schematic view. It also enables you to do some connectivity ECO using the Text Editor by creating the connectivity file, modifying it, and finally reloading it. The connectivity information, such as net creation and instance pin connections to the net, is saved and used from a text file. There are many ways to create and save the connectivity information. For example, you can create a schematic from scratch by using a connectivity file, redo the connectivity of an existing schematic by using a connectivity file, or create a connectivity file, modify, and reload the file for updated connectivity information in the schematic. You can create a connectivity file in any of the following ways:

- Create from a Virtuoso schematic
- Create in Microsoft Excel or Text Editor

To save the connectivity information from the schematic into the specified file:

1. Choose *Module – Connectivity – Create Connectivity*. The Create Connectivity form opens to input the connectivity file name.




2. Specify a connectivity file.


© 2023 Cadence Design Systems, Inc.

3. Select one of the following values of *Mode* for creating the connectivity file:
   - *Replace*: The entire connectivity is deleted before creating the a new one based on the input file.
   - *Merge*: Only the connectivity specified in the input connectivity file is appended to the existing connectivity.
4. Click OK.

   The connectivity file contains data about connectivity in the following format:

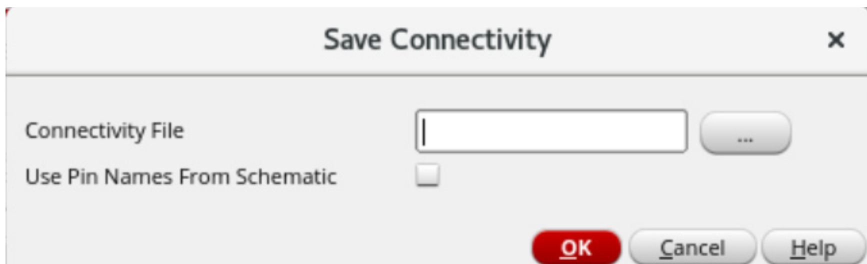
   ```
   Net_Name<space>Instance_Name1/Pin_Name<space>Instance_Name2/
   Pin_Name…
   ```

   Example: `REF i1/OUT i2/IN i3/IN i4/IN`

Connectivity information extraction follows a label-based approach. A label of the specified net is created on wire stubs. Before extracting connectivity, the tool checks for the presence of wire stubs for all instance terminals in the package layout and creates missing wire stubs where required.

To save the connectivity information from the schematic into the specified file:

1. Choose *Module – Connectivity – Save Connectivity*. The Save Connectivity form is displayed.




2. Specify the connectivity file.
3. Click OK.

### *Related Topic*

Extracting Connectivity Information from Package Layout

schematicConnectivityFile

schematicConnectivityMode


© 2023 Cadence Design Systems, Inc.

# 5

# Package Layout Creation

Package layout is generated from the package schematic for editing the package layout. To complete the physical layout, use GFS to generate the layout. If there are flip chip attachments, use the appropriate instance properties to place the components on the top or bottom of the substrate with the correct orientation. Connect balls to IO pads through curved paths by using flight lines and connectivity/DRC markers in the Annotation Browser. You need to fix shorts on signals with power and ground planes by creating dynamic shapes and performing voiding. The synchronized edits in the die abstract and layout are covered in the Edit-in-Concert topic.

**Note:** Module fabrics are treated the same way as package fabrics. Therefore, the commands in the *Module* menu are available to cellviews with package and module fabrics.

## Generate from Source

The package layout is generated from the package schematic using the existing Generate from Source (GFS) functionality to place the components appropriately.

During GFS, each layout instance, net, and terminal is created. Individual components in the schematic are represented as the TILP instances that contain type specific information of their function and parameters, such as rotation, mirrored, flipped, and so on, which control the re-layering and geometry of the shapes contained within the components.

The GFS process creates a single level of layout hierarchy. This layout hierarchy may bind to a different schematic hierarchy - the binder component in the Virtuoso RF Solution maintains the correspondence between objects in the schematic and layout hierarchies.

***Related Topics***

Generating a Layout

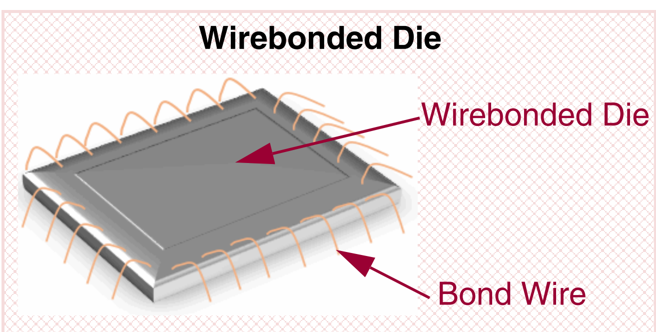
Update Binding Information

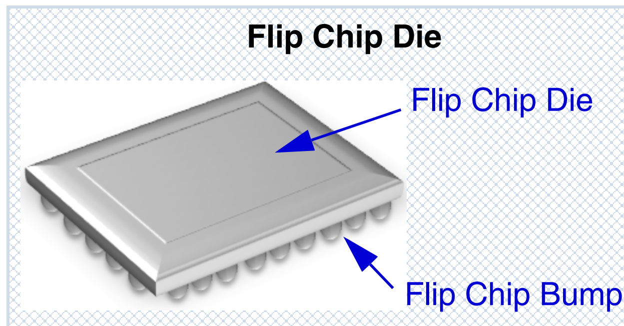

© 2023 Cadence Design Systems, Inc.

# Dies in Virtuoso Multi-Technology Solution

Depending on the mode of attachment, the dies in the Virtuoso RF solution can be classified as wirebonded and flip chip dies, as described below.

- **Wirebonded die:** Uses bond wires to establish connections. Bond wires are connected to the die IO pads on one side and the bond wire endpoints or bond fingers on the other.
- **Flip chip die:** Uses solder balls to establish connections. To connect the dies, solder balls are placed on the top metal layer or on the metal layer at the back. These solder balls and the metal pads beneath them are called bumps. Bumps that lie between a die and the package substrate are called flip chip bumps.

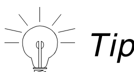



## Wirebonded Dies

The key components of wirebonded dies are the following:

- **Guides:** Paths drawn around dies to define valid placement locations for bond wire endpoints and bond fingers.
- **Bond wires:** Wires, usually made of gold, that connect the die pads either to bond fingers on the component substrate or to die pads on another die.
- **Bond fingers:** A metal pad on the outer layer of the component substrate to which a wire bond is attached. Bond fingers help establish electrical connections between the component substrate and the die.

 *Tip*

Virtuoso supports the free placement of bond wire endpoints and bond fingers. Therefore, you can place bond wires and bond fingers without creating guides.


© 2023 Cadence Design Systems, Inc.

## Flip Chip Dies

For flip chip attachments, die instances on the package are flipped if the IO pads on the die are on the front-side of the IC and connected to the top of the package substrate. In addition, if they are being mirrored, they are connected to the bottom of the package substrate. If the die instance attaches to an internal layer in the package substrate (attachment in a cavity for instance), the layer offset must be specified.

***Related Topics***

Dies in Virtuoso Multi-Technology Solution

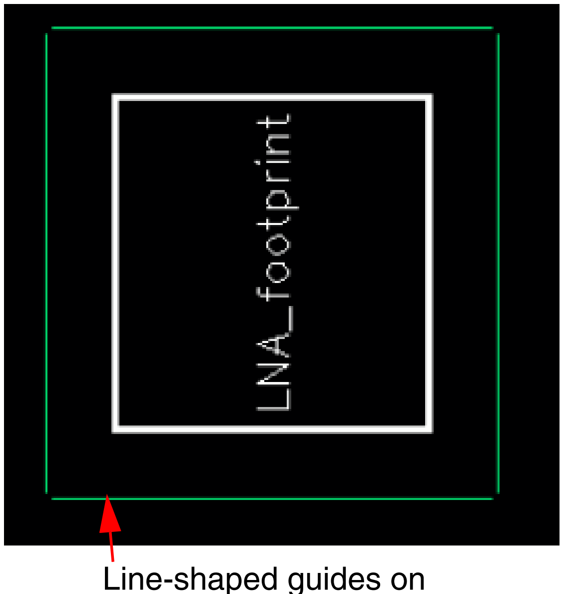
Package Layout Creation

Flip Chip Parameters

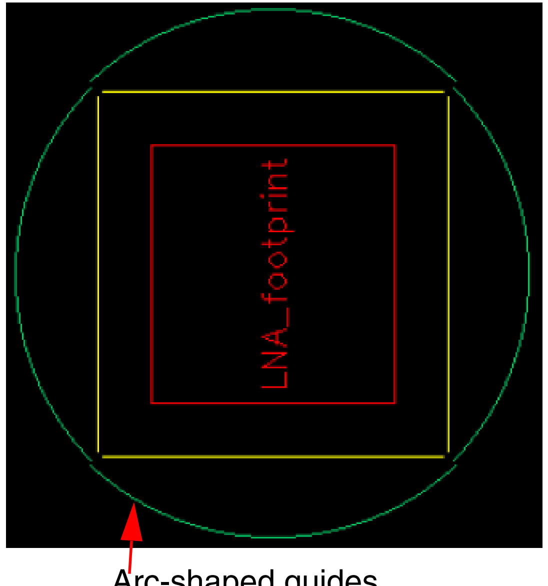

© 2023 Cadence Design Systems, Inc.

# Guides in Wirebonded Dies

You can create one or more line-shaped or arc-shaped guides around dies.

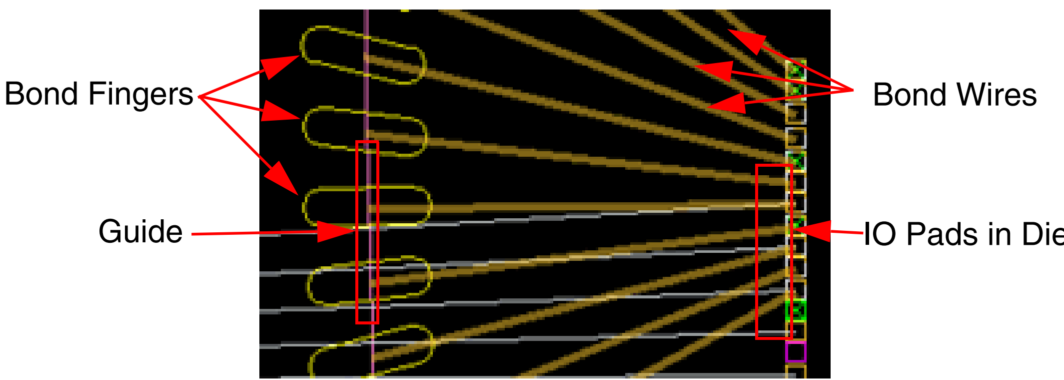



Line-shaped guides on
all four sides of the die




Arc-shaped guides
outside the line-shaped guides
on all four sides of the die

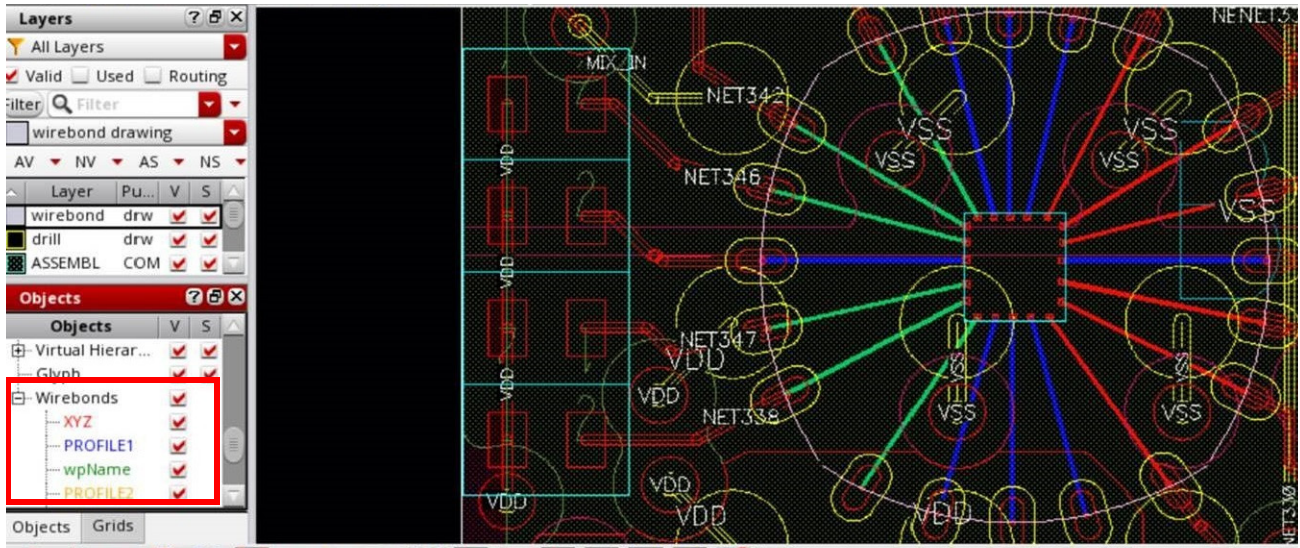
The following diagram shows the connections in a wirebonded die, where the bond fingers are placed on a guide.





© 2023 Cadence Design Systems, Inc.

You can view wirebonds in the *Objects* panel of the *Palette* assistant. You can change the Visibility (*V*) settings of wirebonds through the *Palette*. You can also use the environment variable, `pteInfraWirebondProfileColors`, to assign colors to wirebond profiles.




You can use the SKILL functions, `leGetWirebondProfileVisible` and `leSetWirebondProfileVisible` to view or set the visibility of wirebond profiles, respectively.

### *Related Topics*

Creating a Guide

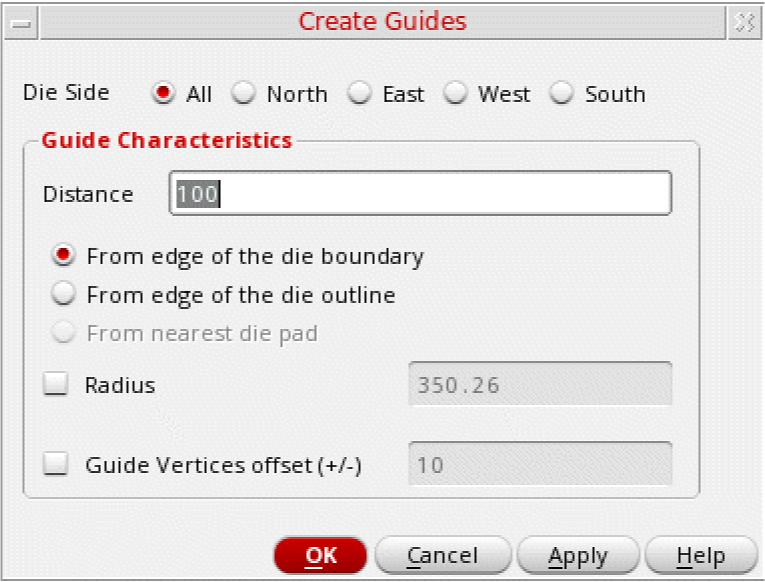
Editing a Guide


© 2023 Cadence Design Systems, Inc.

# Creating a Guide

To create a guide:

1. Select the die for which you would want to create a guide.
2. Choose *Module — Guide — Create* to open the Create Guides form.




The Create Guides form is not displayed when:

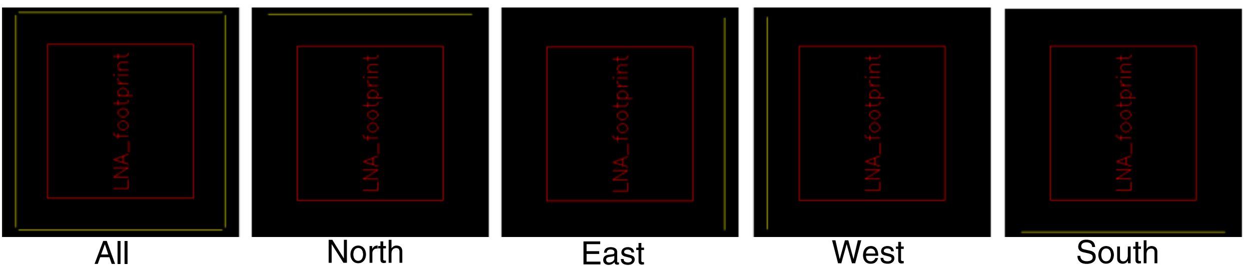
- No die is selected. You are prompted to select a die. The Create Guides form is displayed only after you select a die.
- The selected die is a flip chip. Flip chip dies use solder bumps (not bond wires) to establish connections. Therefore, they do not require guides or bond wires.

**Note:** If multiple dies are selected before invoking the command, only the first die is considered for creating guides.

3. Select a *Die Side* value to specify the side along which one or more guides are to be created: *All*, *North*, *East*, *West*, or *South*.


© 2023 Cadence Design Systems, Inc.

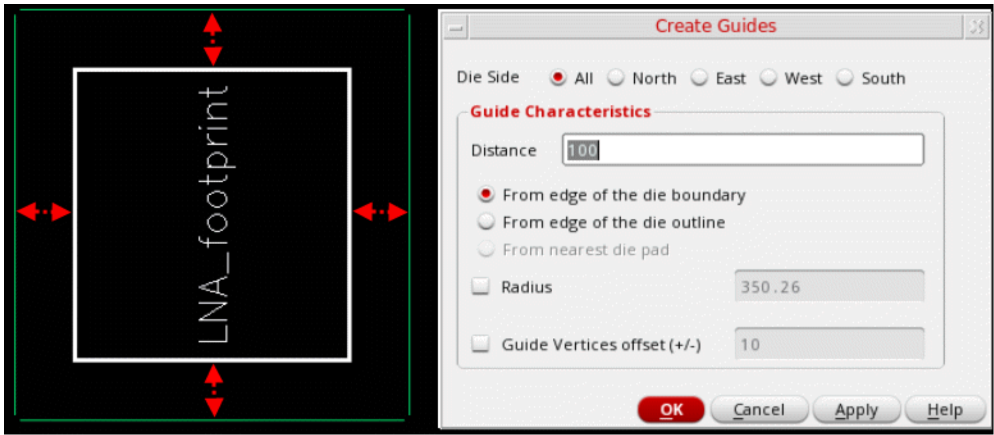
The default value is *All*. Line-shaped guides are created on all four sides of the die. The following images depict die sides.




**Note:** Line-shaped guides are created by default. To create arc-shaped guides, specify a *Radius* value. Arc-shaped guides support the equidistant placement of bond fingers from the IO pads in the die.

4. Specify the distance of the guide in the *Distance* field and the reference from which the distance is to be calculated.

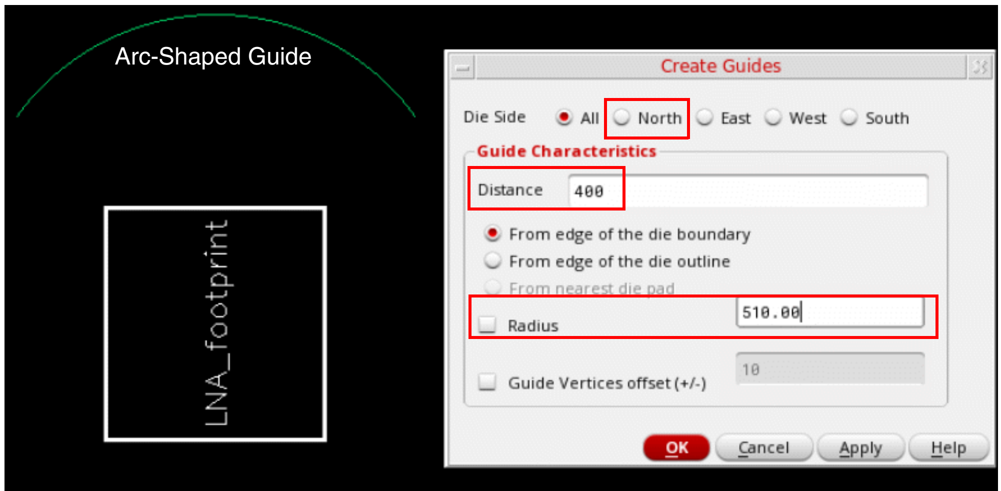
5. Select *From edge to the die boundary* to specify that the distance is to be calculate from the PR boundary of the die. For example, if Die Side is set to North, the distance is calculated from top edge of the PR boundary of the die. This option is selected by default. Use this option to ignore all text labels outside the die PR boundary while creating guides.



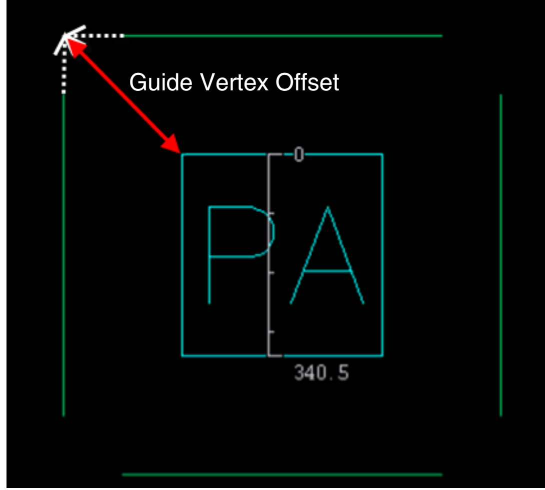

6. Select *From edge to the die outline* to specify that the distance is to be calculated from the die outline that includes all objects lying outside the PR boundary of the die, for example the labels located outside the die PR boundary. If the Die Side is set to North, the distance is calculated from the top of the die outline, which could be larger than the die PR boundary.


© 2023 Cadence Design Systems, Inc.

7. (Optional) Select and specify a *Radius* value to create arc-shaped guides. The curvature of the guide is calculated based on the specified radius.




8. (Optional) Select *Guide Vertices offset* and specify the distance of the diagonal line drawn from the two vertexes of the guide to the center of the die.




9. Click *Apply* or *OK* to create the guide.

   You can now create bond wires with their endpoints on the guides.

***Related Topics***

Guides in Wirebonded Dies

Editing a Guide

Create Guides Form


© 2023 Cadence Design Systems, Inc.

# Editing a Guide

The *Module – Guide* menu includes the *Stretch* and *Move* commands that let you edit guides.

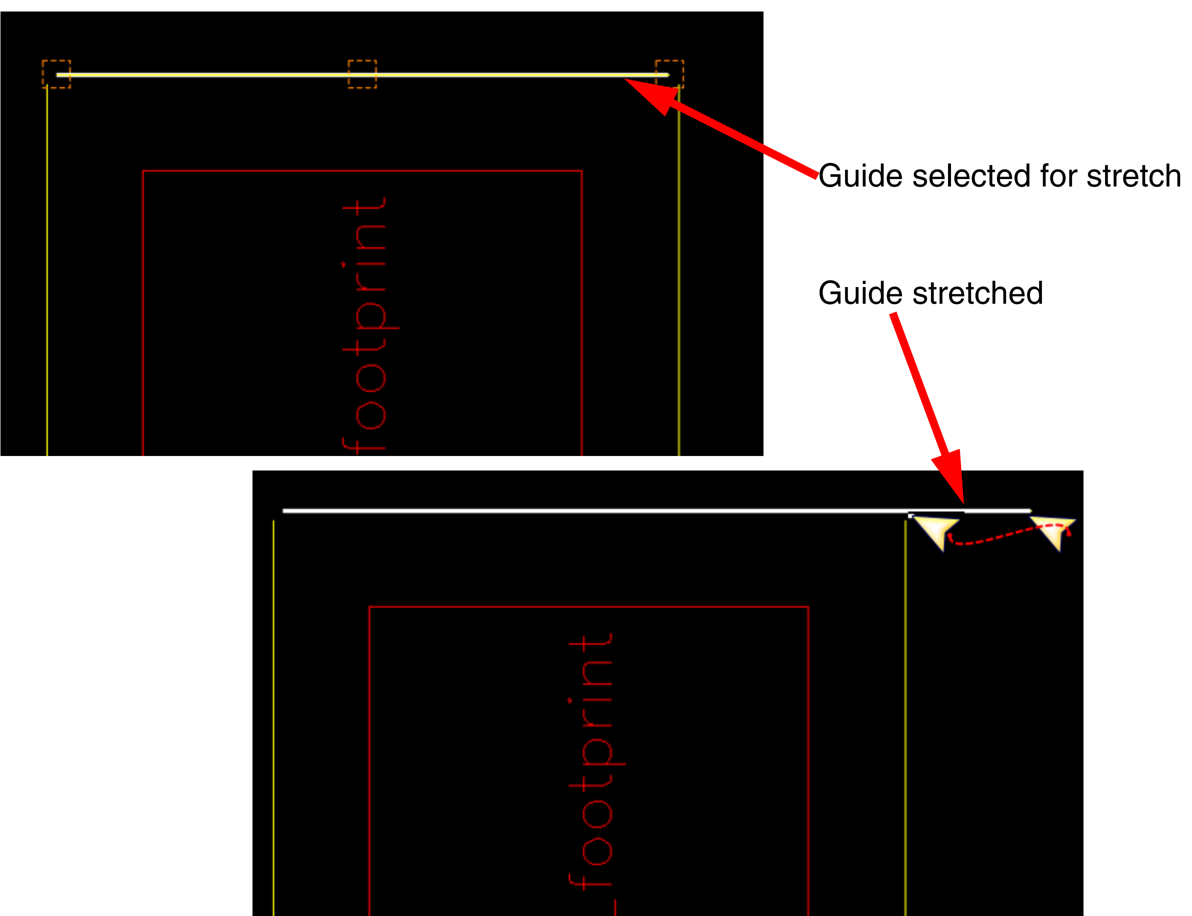
To stretch a guide:

1. Choose *Module — Stretch*.
2. Select the guide to be stretched. Drag lines are displayed on the guide.

   **Note:** If the guide is not selectable, enable the selection setting of the corresponding layer-purpose pair in the Palette assistant.

   Three boxes are displayed at either ends and in the middle of the selected guide. Hover the cursor over a box to stretch the guide from that position.
3. Click and drag the guide to stretch it in the required direction. Release the pointer.

   The guide is stretched. Any existing bond fingers or wires are re-positioned accordingly.



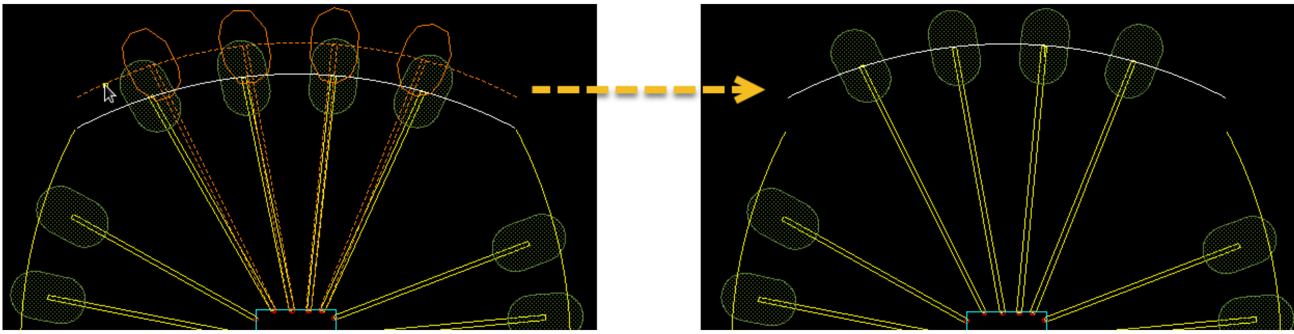


© 2023 Cadence Design Systems, Inc.

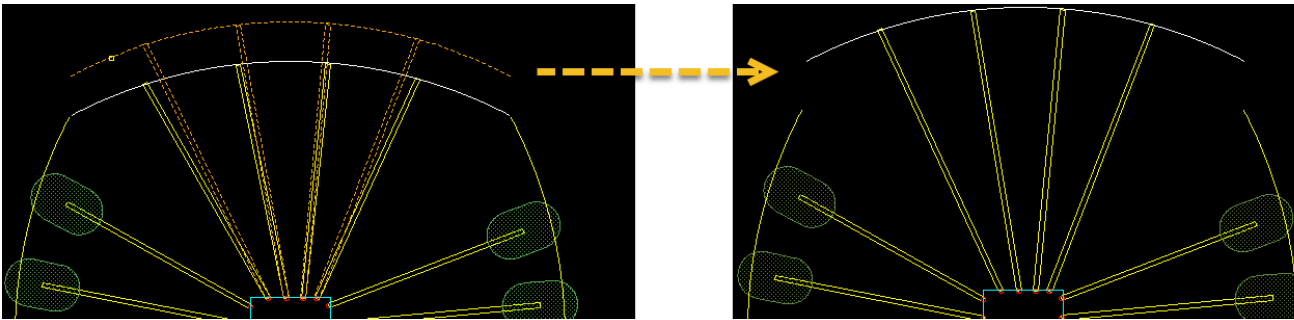
To move a guide:

1. Choose *Module – Move*. Alternatively, you can use the shortcut key `M`.
2. Select the guide to be moved.
3. Move the pointer to a new location.
4. Click where you want to place the guide. The guide is moved to the new location. Any existing bond fingers or wires are moved accordingly.

The following images depict the movement of guides with bond fingers.



The following images depict the movement of guides without bond fingers.



***Related Topics***

[Guides in Wirebonded Dies]

[Creating a Guide]


© 2023 Cadence Design Systems, Inc.

# Bond Wires and Bond Fingers Creation

Bond wires are wires of very fine diameter, typically made of aluminum, copper, silver, or gold, that connect the die IO pads to the package substrate. Virtuoso provides options to create and edit bond wires.

Virtuoso supports both freestyle and guide-based placement of bond wires. Bond wires can originate from level-1 pins or die IO pads and end in a guide or selected points in the canvas. You can also create bond wires from bond fingers to guides. Before creating bond wires and bond fingers, ensure the following:

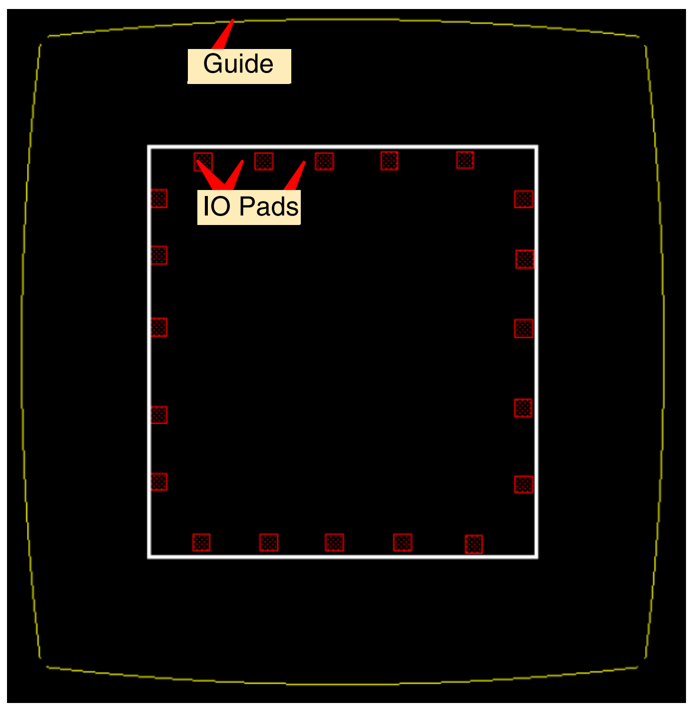
- The wire profiles are defined in the technology file using the `techCreateWireProfile` SKILL function:

```
techCreateWireProfile( tech "BW_PROFILE1" "forward"
'(
    ( type "point" horizontal (type "length" value 0.0)    vertical (type "percent" value 25.0))
    ( type "point" horizontal (type "percent" value 25.0)  vertical (type "angle" value 10.0))
 )
)

techCreateWireProfile( tech "BW_PROFILE2" "forward"
'(
    ( type "point" horizontal (type "length" value 0.0)    vertical (type "percent" value 50.0))
    ( type "point" horizontal (type "percent" value 25.0)  vertical (type "angle" value 20.0))
 )
)
```


© 2023 Cadence Design Systems, Inc.

- For guide-based placement, the guides are in place.




***Related Topics***

Create Bond Wire Form

Creating Bond Wires and Bond Fingers

Setting a Bond Wire Profile

Creating Bond Finger Definitions

Updating the Finger Attach Point

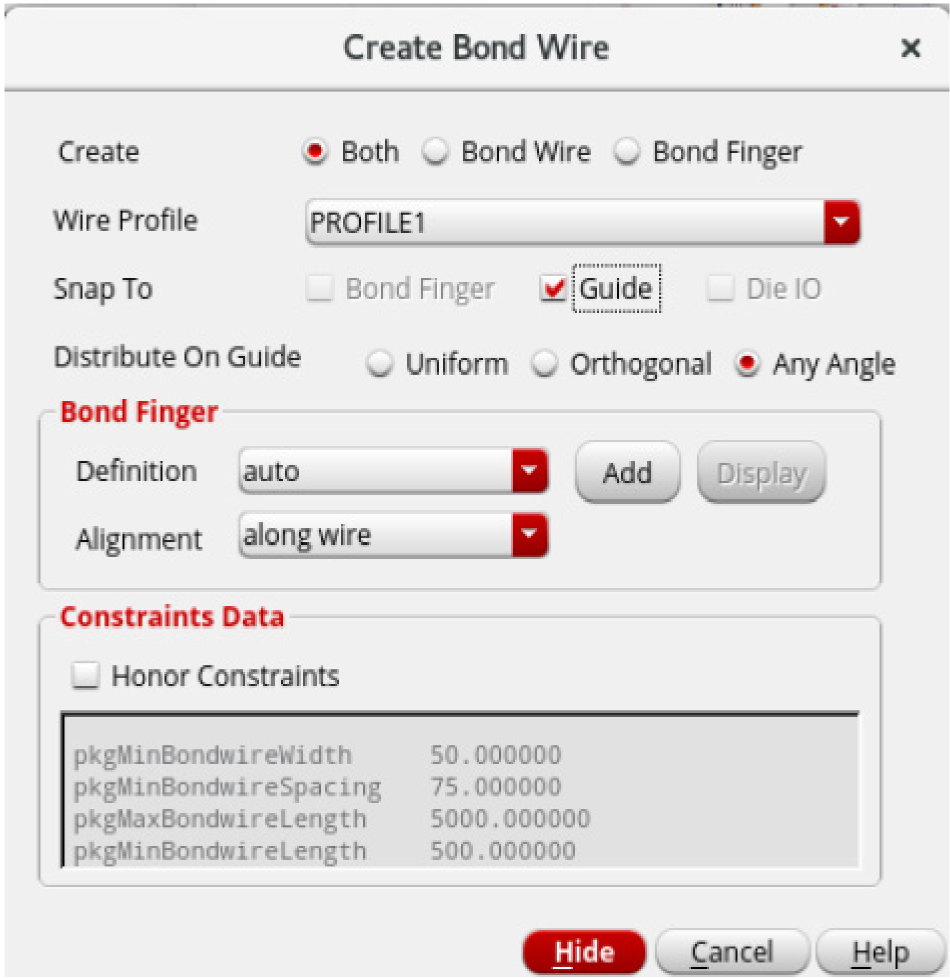
techCreateWireProfile


© 2023 Cadence Design Systems, Inc.

# Creating Bond Wires and Bond Fingers

To create bond wires:

1. Select the level-1 pins, IO pads, or bond fingers for which bond wires are to be created. It is recommended that you create bond wires for only one guide at a time. For example, choose all the IO pads for which bond wires are to be created in the guide in the top direction.
2. Choose *Module – Bond Wire – Create.*
3. Move the pointer over the required guide to see the valid placement locations for bond wire endpoints or fingers.
4. Press `F3` to display the Create Bond Wire form.




5. Select one of the following options to specify what you want to create:


© 2023 Cadence Design Systems, Inc.

- ❑ *Both*: Use this option to create both a bond wire and a bond finger.
- ❑ *Bond Wire*: Use this option to create only a bond wire without adding a bond finger, for example to re-bond dies after replacing or swapping them.
- ❑ *Bond Finger*: Use this option to create a bond finger.

Options in the *Create* section are not available when bond fingers are selected as the source.

6. Select a *Wire Profile* value.

   **Note:** *Wire Profile* is available only when *Create* is set to *Both* or *Bond Wire*.

7. Set *Snap To* to specify whether the created components are to automatically snap to *Bond Finger*, *Guide*, or *Die IO*.
   - ❑ In the *Bond Wire* mode, all *Snap To* options are available.
   - ❑ In the *Both* and *Bond Finger* modes, only *Guide* is available.

   In the *Bond Wire* mode, the tool supports snapping of bond wires to existing embedded bond fingers at level-1 in design, if available.

   For bond wires that originate from bond fingers, *Guide* is the only available snapping option.

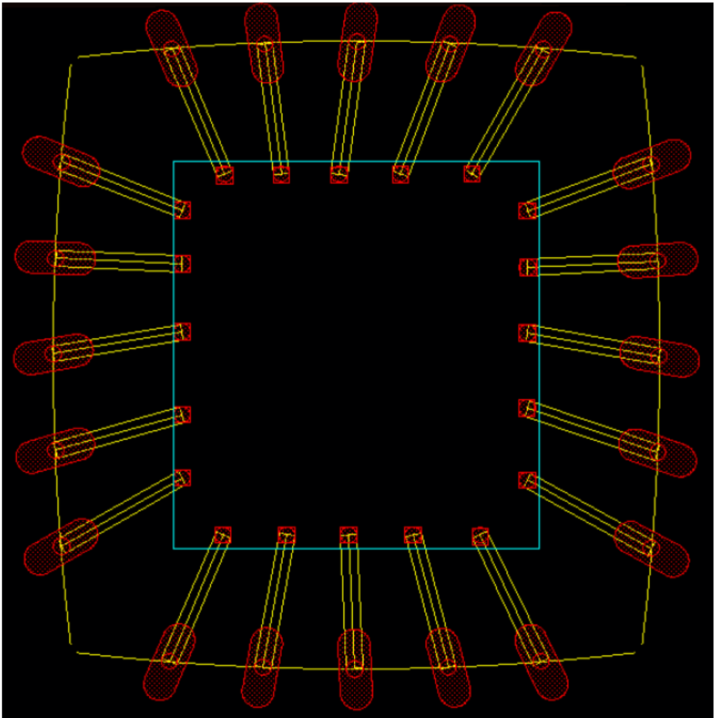
8. Select *Distribute On Guide* to one of the following values to specify how bond fingers are to be distributed on the guide. The available options are: *Uniform*, *Orthogonal,* and *Any Angle*.
9. Specify the *Bond Finger* parameters. The *Bond Finger* section is available for edit only when *Create* mode is set to *Bond Finger*.
10. Select a bond finger definition from the *Definition* list.
    - ❑ Click *Display* to view the settings in the Add Bond Finger Definition form. You can edit the settings as per your requirements.
11. Click *Add* to create a new bond finger definition. The Add Bond Finger Definition form is displayed.
12. Click *OK*. The new bond finger definition is automatically selected in the *Definition* list. Select one of the following *Alignment* types for bond fingers:
    - ❑ *along wire*: Aligns bond fingers in the same direction as the bond wires.
    - ❑ *orthogonal to die*: Determines the die edge that is closest to IO pads and aligns the bond fingers orthogonal (perpendicular) to this edge. The direction of the bond finger does not change even when it is moved to another die edge.Click *Hide*.


© 2023 Cadence Design Systems, Inc.

13. Click the *Honor Constraints* check box to set a higher priority for constraints over the bond wire profile. By default, it is deselected to ensure that bond wire profile has a higher priority than the constraints.

14. Click to place the bond wires and bond fingers at the required locations.

15. Follow steps 1 through 6 to create bond wires (and bond fingers) in all directions, as shown in the following image.



Bond wires and bond fingers are created as per your specifications. The bond wire end points are centered to the bond finger and die pins. A single net connects each IO pad to its corresponding bond wire and bond finger.

**Note:** The `pkgMinBondwireWidth` constraint is honored during bond wire creation.

*Important*

The bond wire settings are valid for both mirrored and non-mirrored dies.

The *Check Against Source* command does not report mismatches between wire bond and finger instances.


© 2023 Cadence Design Systems, Inc.

### *Related Topics*

Create Bond Wire Form

Bond Wires and Bond Fingers Creation

Creating Bond Finger Definitions

pteInfraWirebondProfileColors

leGetWirebondProfileVisible

leSetWirebondProfileVisible

pkgMinBondwireSpacing

pkgMinBondwireWidth

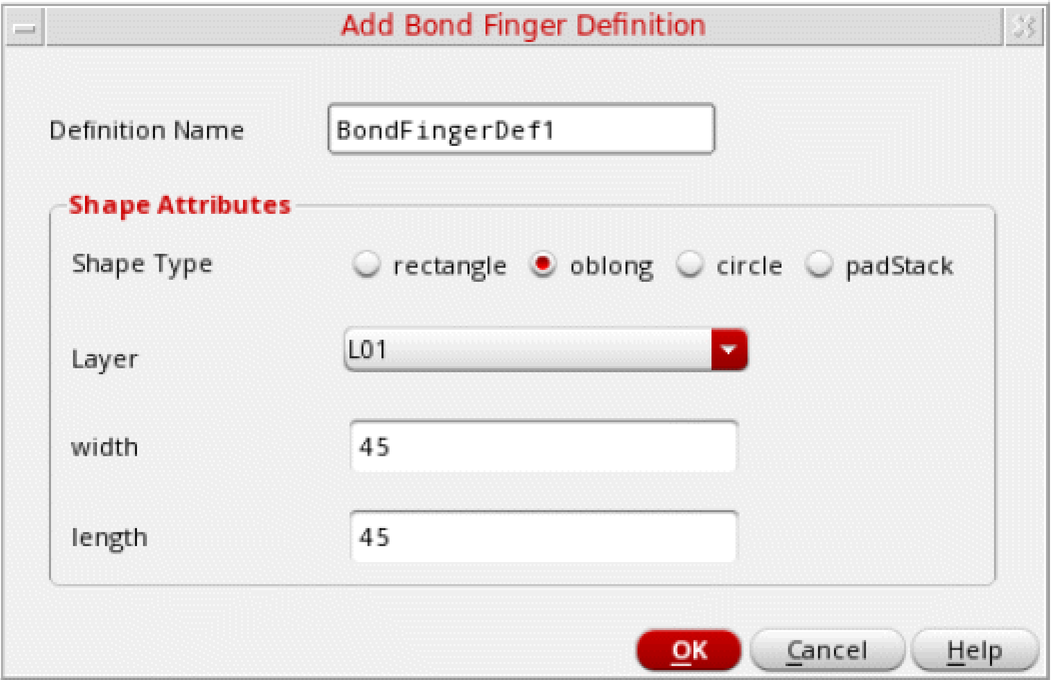
honorConstraints


© 2023 Cadence Design Systems, Inc.

# Creating Bond Finger Definitions

The Virtuoso RF solution supports the creation and reuse of bond fingers. When creating bond wires, instead of specifying bond finger parameters each time, you can choose from a list of predefined bond fingers. To define bond fingers:

1. Open the Create Bond Wire form.
2. Click *Add* next to the *Definition* option in the *Bond Finger* section. The Add Bond Finger Definition form is displayed.




3. Specify a unique value for *Definition Name*.
4. Select a shape for the bond wire: *rectangle*, *oblong*, *circle*, or *padStack*. Depending on the selected shape, the remaining options vary.
   - For *rectangle* and *oblong*, specify the *Layer*, *width*, and *length* of the bond fingers.
   - For *circle*, specify values in the *Layer* and *diameter* fields.
   - For *padStack*, specify values in the *Library Name*, *Cell Name*, and *View Name* fields. You can click *Browse* to choose values from the Library Browser.
5. Click *OK*.


© 2023 Cadence Design Systems, Inc.

***Related Topics***

Create Bond Wire Form

Bond Wires and Bond Fingers Creation

Padstacks

Creating Bond Wires and Bond Fingers

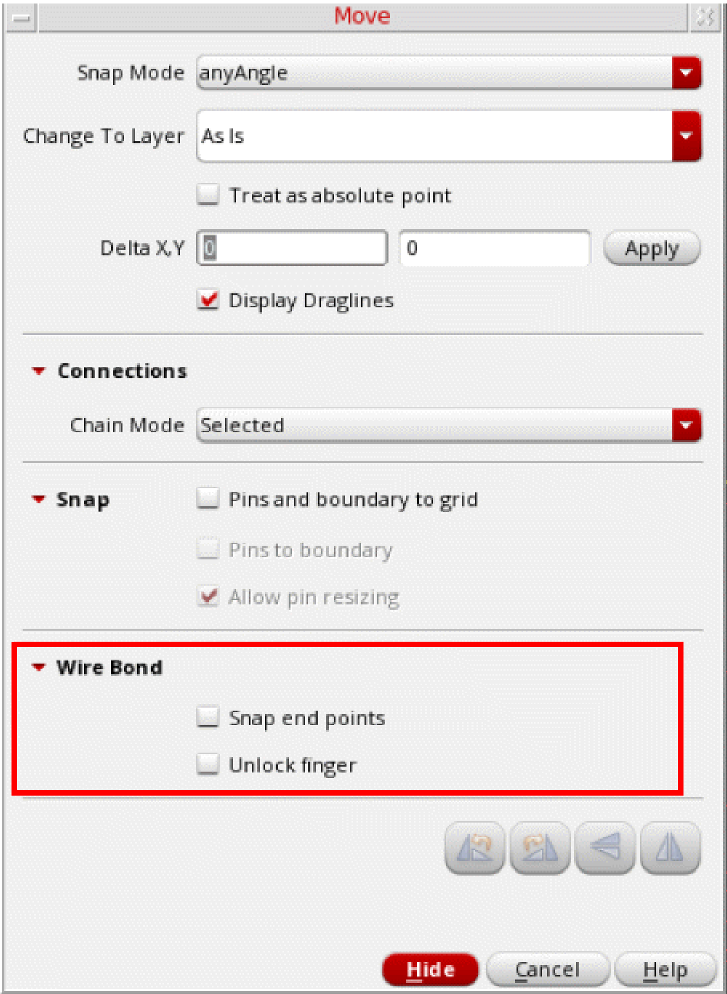
# Moving Bond Wires and Bond Fingers

The *Move wire or finger command* lets you move bond wires and fingers individually to a new position on the guide. To move a bond wire:

1. Choose *Module – Bond Wire – Move wire or finger*. Alternatively, you can use the shortcut key `M`.
2. Select the bond wire to be repositioned by selecting either the complete instance or the level-1 path of the bond wire instance.
3. Press `F3` to display the Move form. In addition to the standard Move options, the form includes a *Wire Bond* section. Select *Snap end points* to specify the snapping behavior of bond wires. The snapping behavior differs based on the selection.
   - If the IO side of a bond wire is selected, the endpoint is snapped to the same net IOs.
   - If the finger side is selected, the snapping behavior depends on the *Unlock finger* setting.
     - **OFF (default):** Endpoints of locked fingers are snapped to the nearest guide.


© 2023 Cadence Design Systems, Inc.

- **ON**: Endpoints are snapped to the nearest guide, finger of same net, or IO of same net.



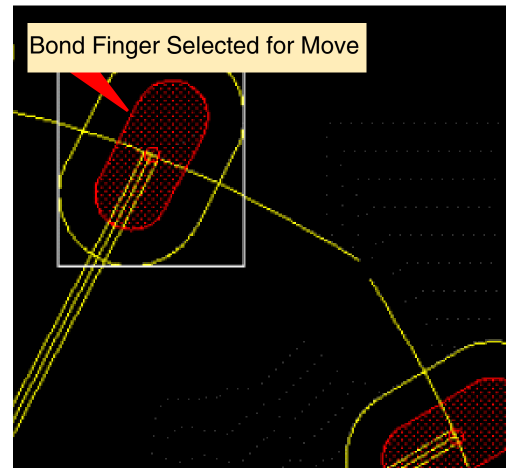

**a.** Select the required options in the Move form.

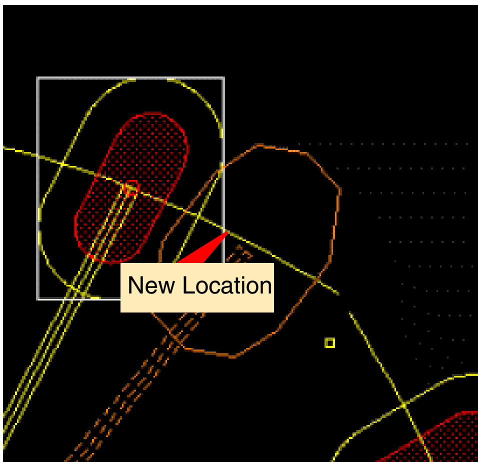
**b.** Click *Hide*.

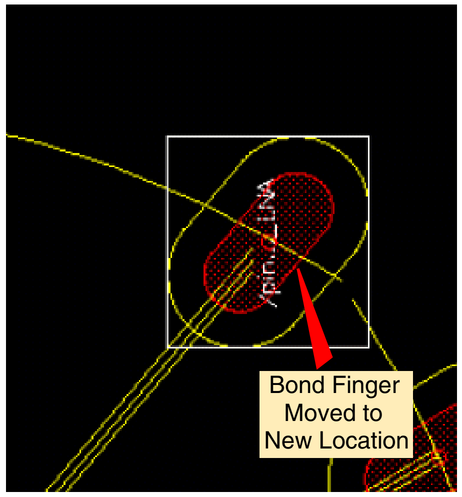
**c.** Move the pointer to the required location. Click to place the bond wire.


© 2023 Cadence Design Systems, Inc.

The bond finger and bond wire are repositioned as follows:




### *Related Topic*

Updating the Finger Attach Point

Setting a Bond Wire Profile

Move Form

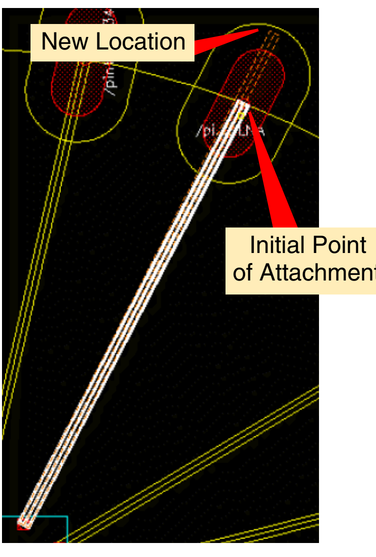

© 2023 Cadence Design Systems, Inc.

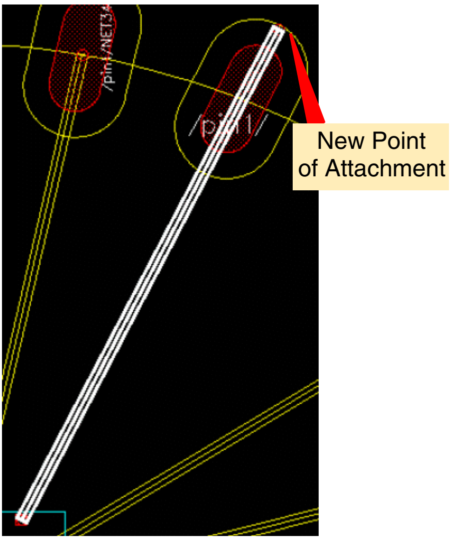
# Updating the Finger Attach Point

The Update finger attach point command lets you alter the point where a bond wire is attached to its bond finger.

1. Choose *Module – Bond Wire – Update finger attach point.*
2. Click the bond finger for which the point of attachment is to be changed.
3. Move the pointer to the required location.
4. Click to indicate the new attachment point.

   The bond finger attachment point is updated as shown in the following images.




***Related Topic***

Moving Bond Wires and Bond Fingers

Setting a Bond Wire Profile


© 2023 Cadence Design Systems, Inc.

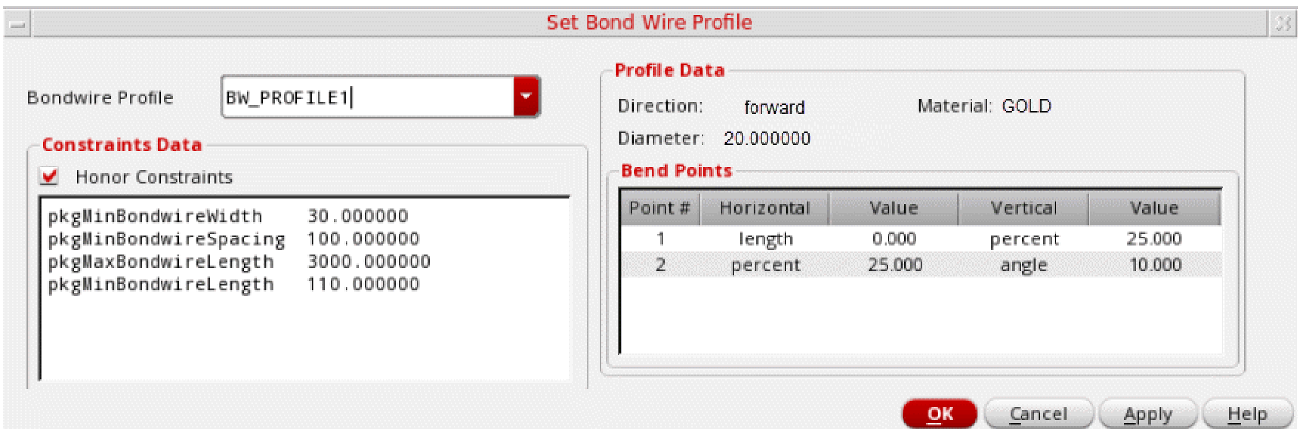
# Setting a Bond Wire Profile

A bond wire profile defines the properties and curvature of a bond wire. Bond wire profiles can be applied to existing bond wires to update their structures and properties.

In the Virtuoso RF Solution, bond wire profiles are defined in the technology file. You can create bond wire profiles that match your specific design needs.

To apply a bond wire profile to a bond wire:

1. Select the required bond wires in the layout canvas.
2. Choose *Module – Bond Wire – Set Bond Wire Profile*. The Set Bond Wire Profile form appears.




3. Select the required profile from the *Bondwire Profile* list.
4. Select *Honor Constraints* to specify that the bond wires are to honor the listed constraints. To deviate from the specified constraints, turn off *Honor Constraints*.

   For example, in the profile shown above, the wire profile *Diameter* is set to `20.00` and `pkgMinBondWireWidth` constraint is `30.00`. To override the profile-specified values over constraints, turn off Honor Constraints. In the above example, the correct value is `30.00`. However, to assign the value as `20.00`, turn off *Honor Constraints*.

5. Verify the following settings in the *Profile Data* section:
   - *Direction*: Specifies the direction of the bond wire.
   - *Material*: Specifies the material with which the bond wire is made.
   - *Diameter*: Specifies the diameter of the bond wire. The valid values are positive, non-zero numbers.


© 2023 Cadence Design Systems, Inc.

6. Verify the settings in the *Bond Points* section.

   **Note:** You cannot edit the above values in the *Set Bond Wire Profile* form. Update the bond wire profile definitions in the technology file.

Click *OK* to apply the bond wire profile to the selected bond wire.

Alternatively, to set the bond wire profile quickly without opening the form, select the required bond wire, right-click, and select a bond wire profile from the Set Bond Wire Profile command in the shortcut menu.

### *Related Topics*

Moving Bond Wires and Bond Fingers

Updating the Finger Attach Point

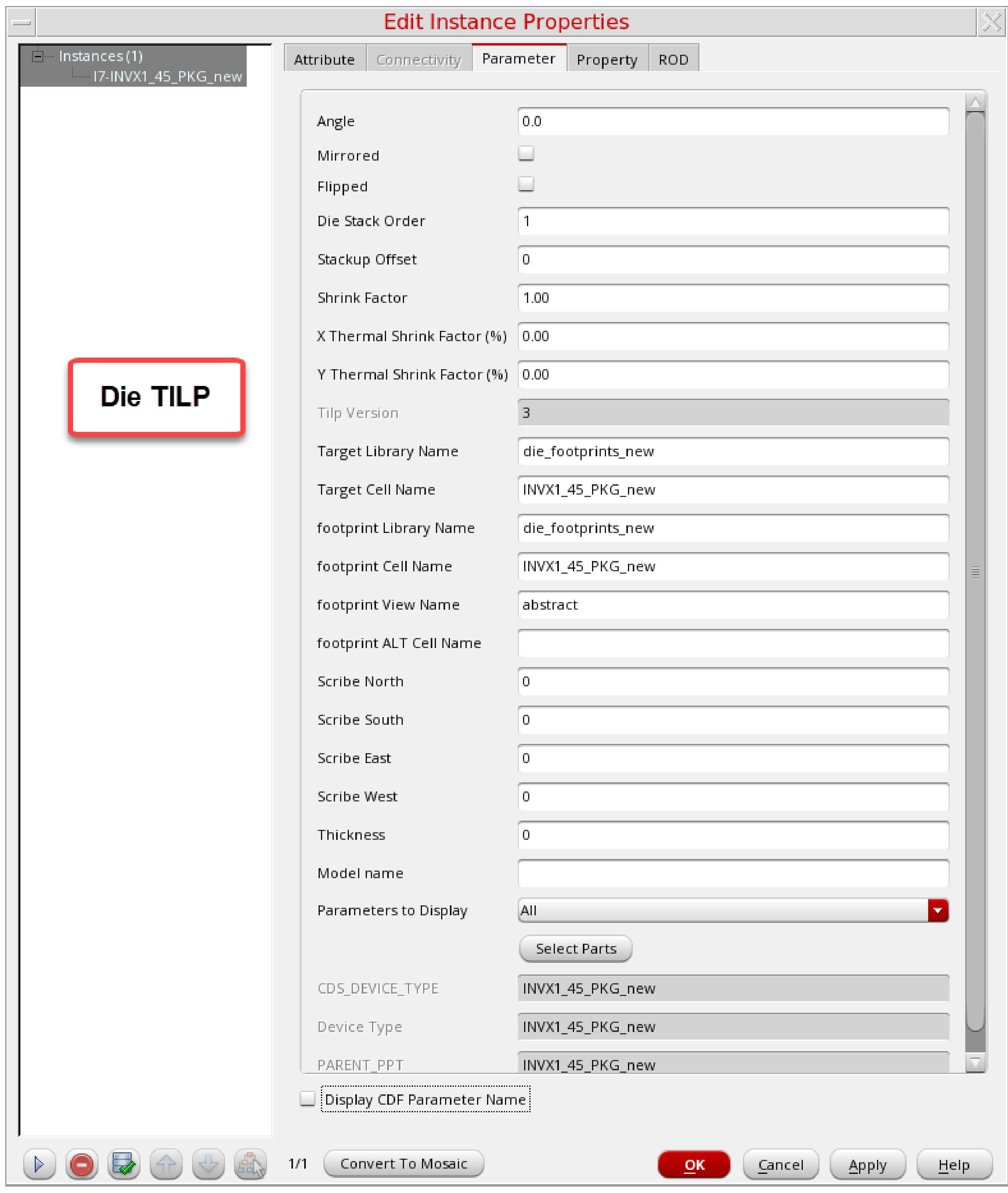
Set Bond Wire Profile Form


© 2023 Cadence Design Systems, Inc.

# Flip Chip Parameters

The image below shows the parameters to be set when attaching a die or package TILP to the appropriate side of the substrate. Select the die instance in the package layout and right-click to view the *Parameter* tab of the Edit Instance Properties form.

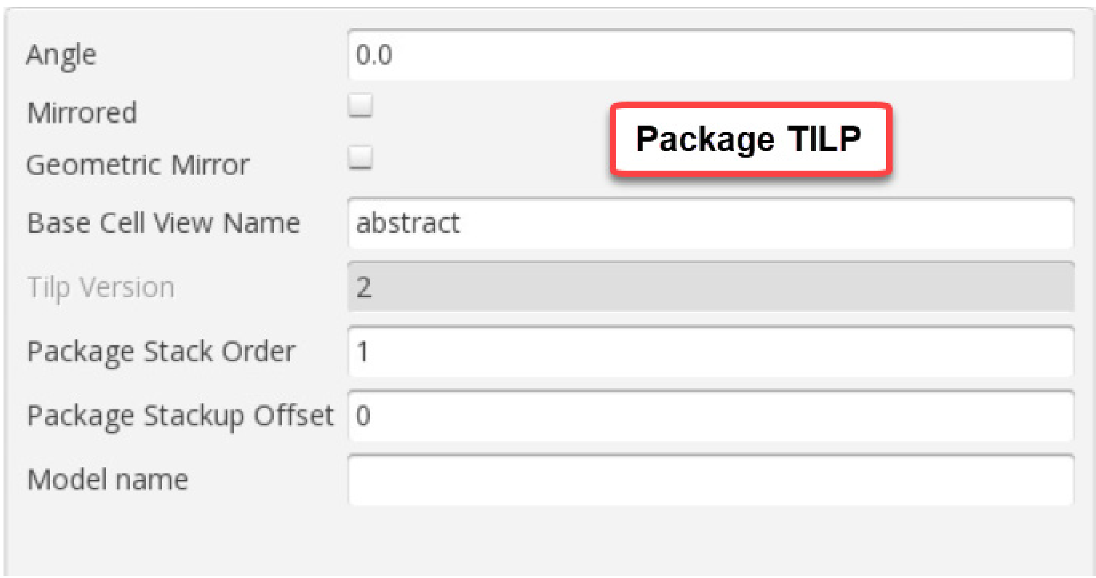

© 2023 Cadence Design Systems, Inc.



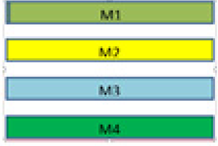


© 2023 Cadence Design Systems, Inc.

This image shows the parameters for a package TILP.




Refer to the following illustration when choosing the parameters. A sample package stackup is shown with four layers (intervening dielectrics are not shown).




***Related Topics***

Dies in Virtuoso Multi-Technology Solution

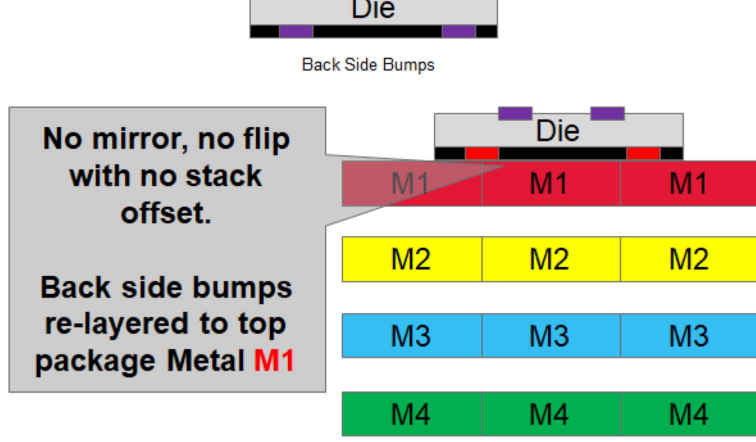
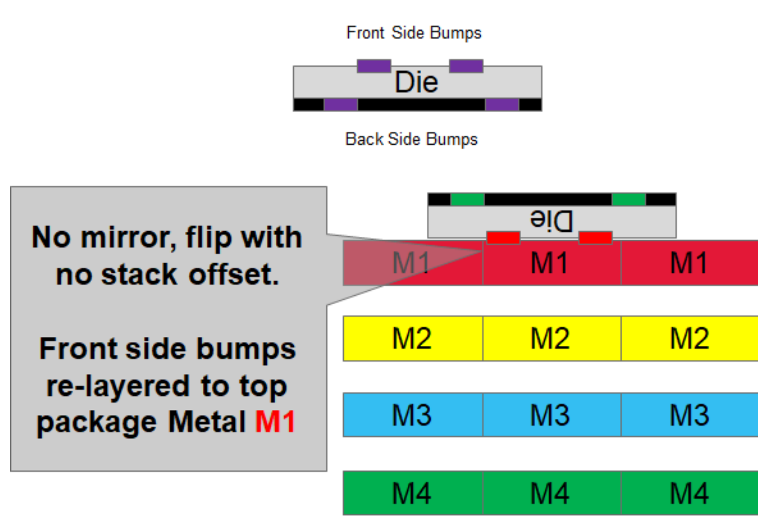
Types of Bump Attachments

Edit Instance Properties Form (Die/Package TILP Parameters)

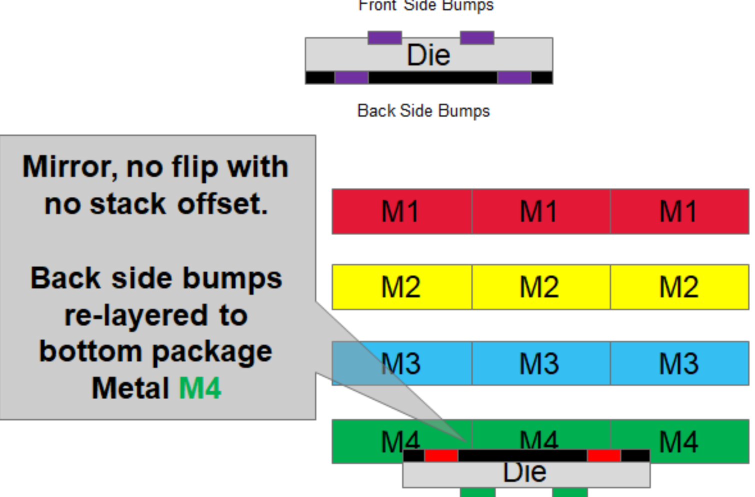
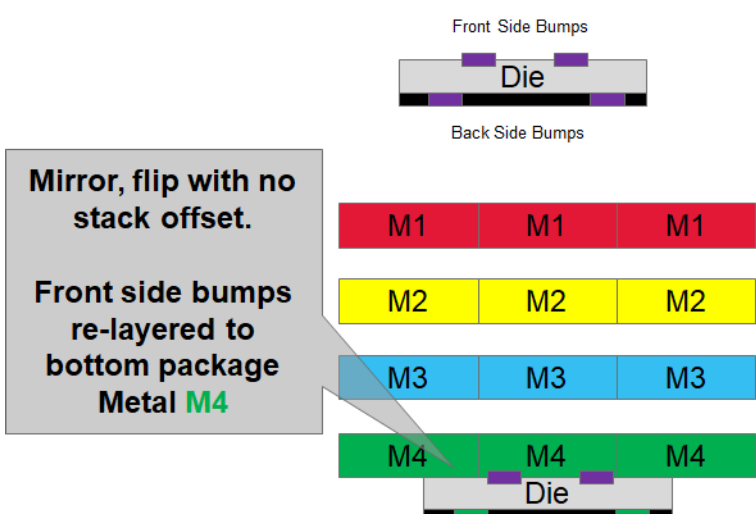

© 2023 Cadence Design Systems, Inc.

# Types of Bump Attachments

Here are the various options of bump attachments.

| Type of Bump Attachment | Illustrations |
|---|---|
| `No mirror, No Flip, Stack Offset = 0` |  |
| `No mirror, Flip, Stack Offset = 0` |  |

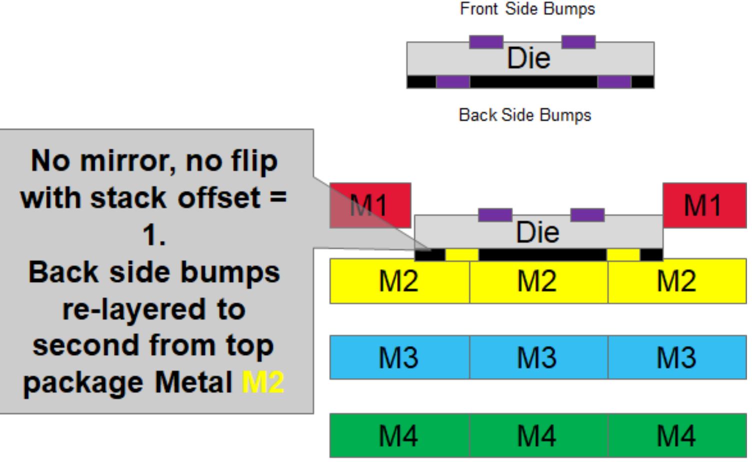
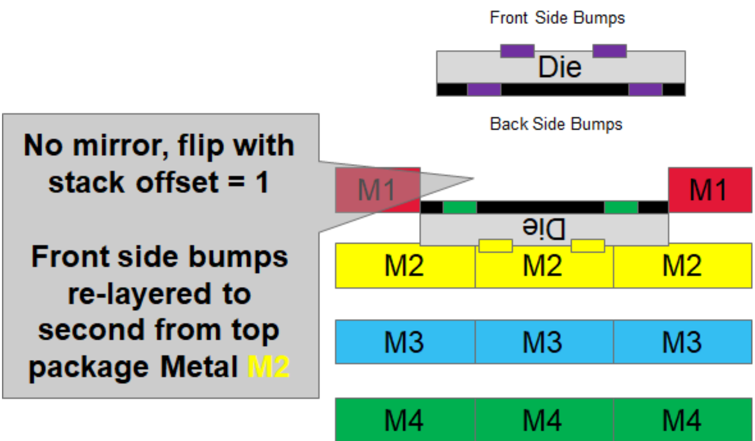

© 2023 Cadence Design Systems, Inc.

| Type of Bump Attachment | Illustrations |
|---|---|
| `Mirror, No Flip, Stack Offset = 0` |  |
| `Mirror, Flip, Stack Offset = 0` |  |





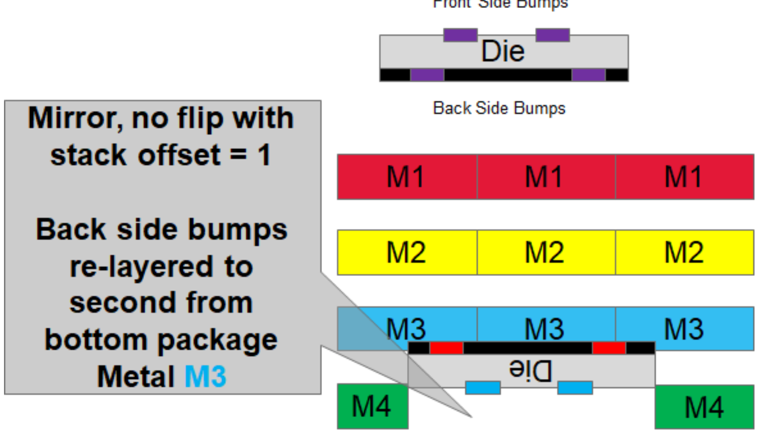
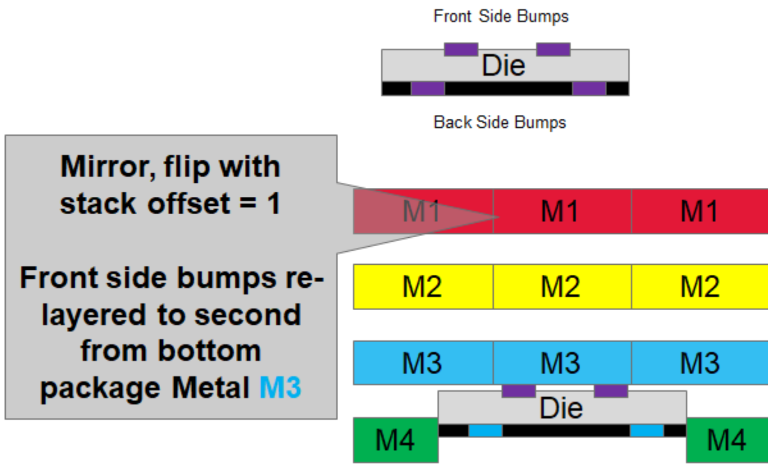

© 2023 Cadence Design Systems, Inc.

| Type of Bump Attachment | Illustrations |
|---|---|
| `No mirror, No Flip, Stack Offset = 1` |  |
| `No mirror, Flip, Stack Offset = 1` |  |






© 2023 Cadence Design Systems, Inc.

| Type of Bump Attachment | Illustrations |
| --- | --- |
| `Mirror, No Flip, Stack Offset = 1` |   |
| `Mirror, Flip, Stack Offset = 1` |   |

***Related Topics***

IxGenerationOrientation

Flip Chip Parameters

Dies in Virtuoso Multi-Technology Solution

TILP Versions


© 2023 Cadence Design Systems, Inc.

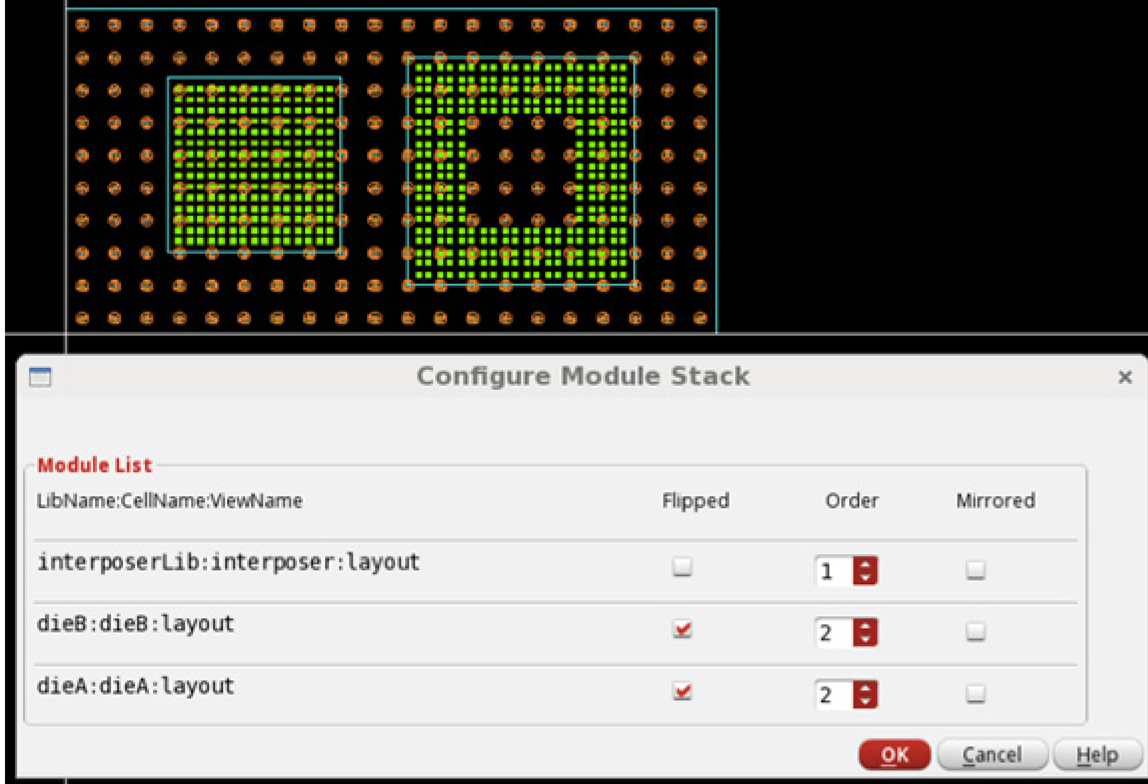
# Configuring a Module Stack

In a multi-chip module or stacked IC, to configure a module stack:

1. Click *Module – Configure Module Stack* to define the arrangement of dies. The Configure Module Stack form is displayed.

   The arrangement definition in the form includes the following information:

   - ❑ Type of dies - wirebonded or flipchip
   - ❑ Position of dies in the stack
   - ❑ Mirrored or not, that is, if they are placed at the front or at the rear side of an interposer




2. Select *Flipped* to specify whether a die is a flipchip.
3. Use the *Order* drop-down list to specify the order of a die or object in a stack.
4. Select *Mirrored* to specify whether a die is mirrored.
5. Click *OK*.


© 2023 Cadence Design Systems, Inc.

***Related Topic***

Package Layout Creation

Types of Bump Attachments

# Thermal Shrink Factor

The Virtuoso RF Solution lets you encapsulate dies of different semiconductor materials into a single package. Within a package, flip chip die-to-package assembly is done by soldering the IO pads in the die to the package substrate by heating both the die and the package. The bump flux in the flip chip die melts and fuses with the package substrate. Such connections

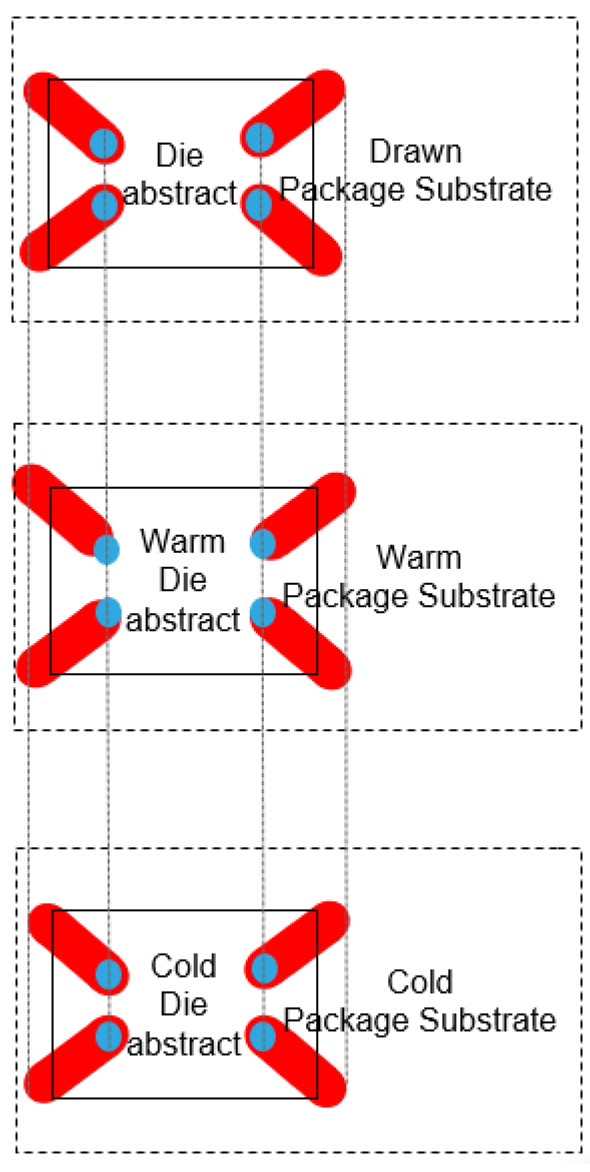

© 2023 Cadence Design Systems, Inc.

might be impacted by the differences in the coefficients of thermal expansion (CTEs) of the semiconductor materials that are used in the package. Consider the following example:

**Initial state:**
A flip chip die abstract is placed on a die package substrate. The IO pads in the die abstract (blue circles) overlap the connections in the package substrate (red lines).

**The die abstract and package substrate are heated:**
Their CTEs are different. In this example, the package expands more than the die abstract. The package traces spread and their endpoints do not coincide with the die bump center. The electrical and net connectivities between the die abstract and package substrate are broken.

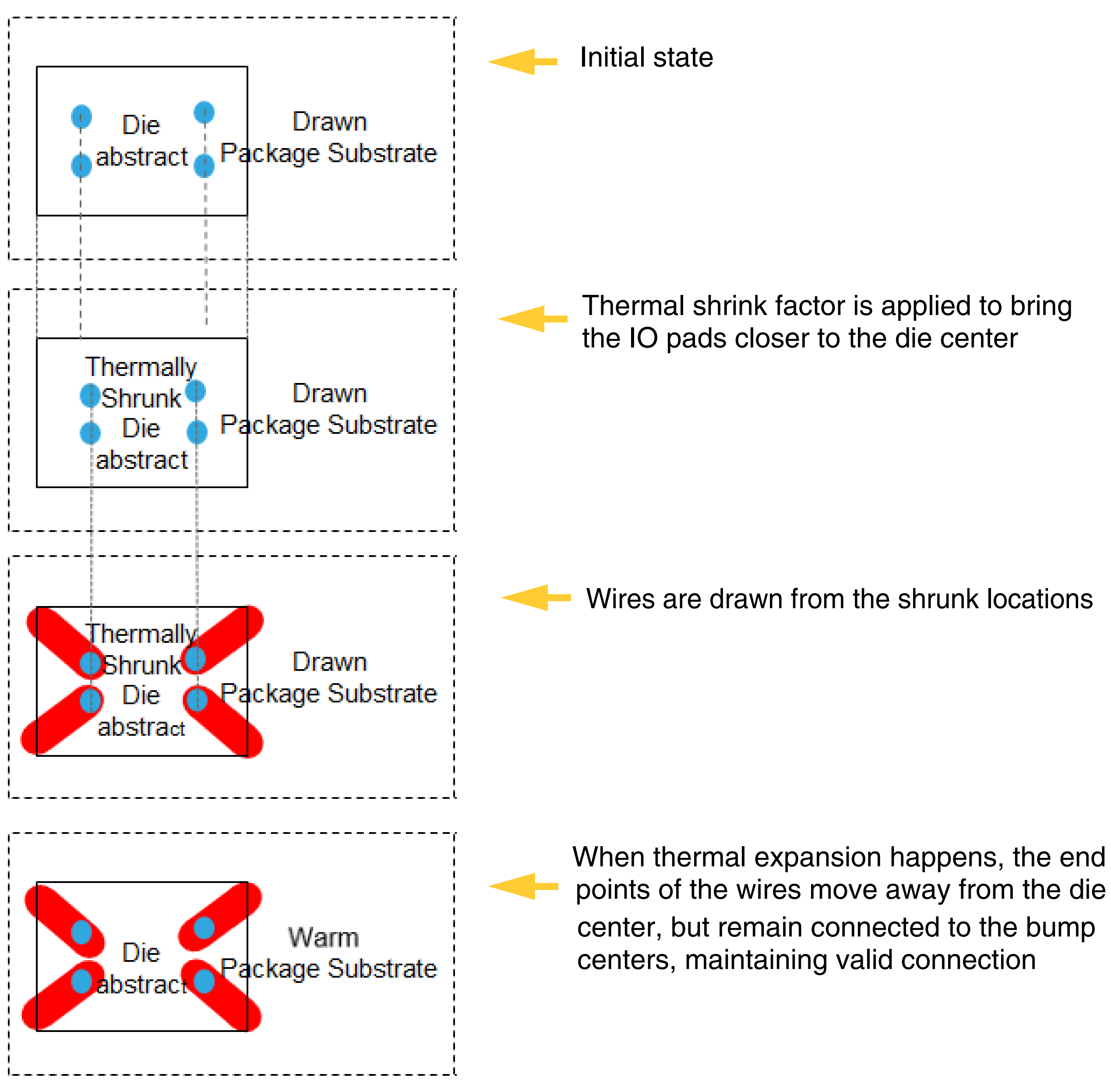
**The system cools down (post manufacture):**
The package traces return to their original locations. The IO pads are placed over the package traces (routes) spatially, however their underlying connections are broken. The damage was done when the die-package assembly was heated. The melted bump flux could not make a good electrical connection because the IO pads were not overlapping the package traces when the system was heated.




To overcome such connectivity issues due to differences in CTEs, Virtuoso lets you set the thermal shrink factor for such dies. This option adjusts the initial positions of IO pads such


© 2023 Cadence Design Systems, Inc.

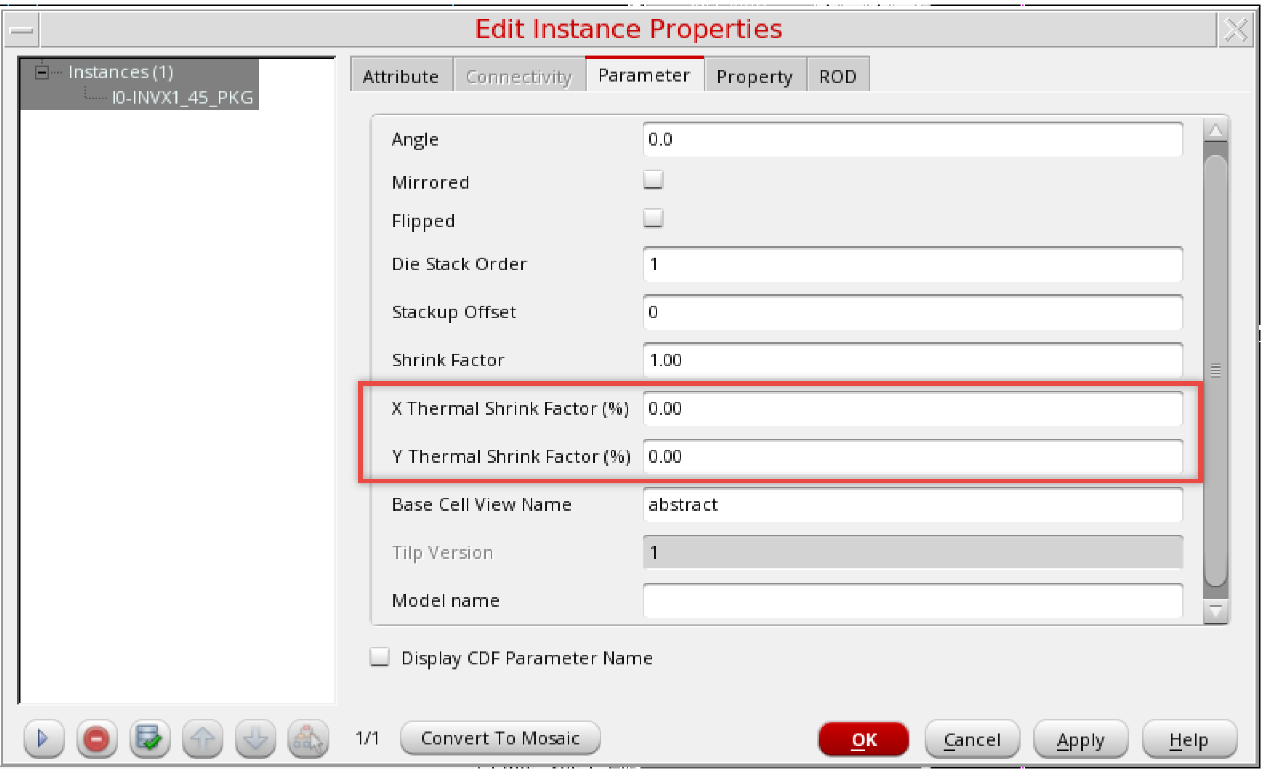
that connectivity is not broken when the dies are heated. It is recommended that you apply the thermal shrink factor before defining connectivity, as shown in the following example:





© 2023 Cadence Design Systems, Inc.

Thermal shrink factor is defined as a parameter of the TILP instance. *X Thermal Shrink Factor* and *Y Thermal Shrink Factor* are percentage values for expansion or contraction along the x-axis and y-axis, respectively.



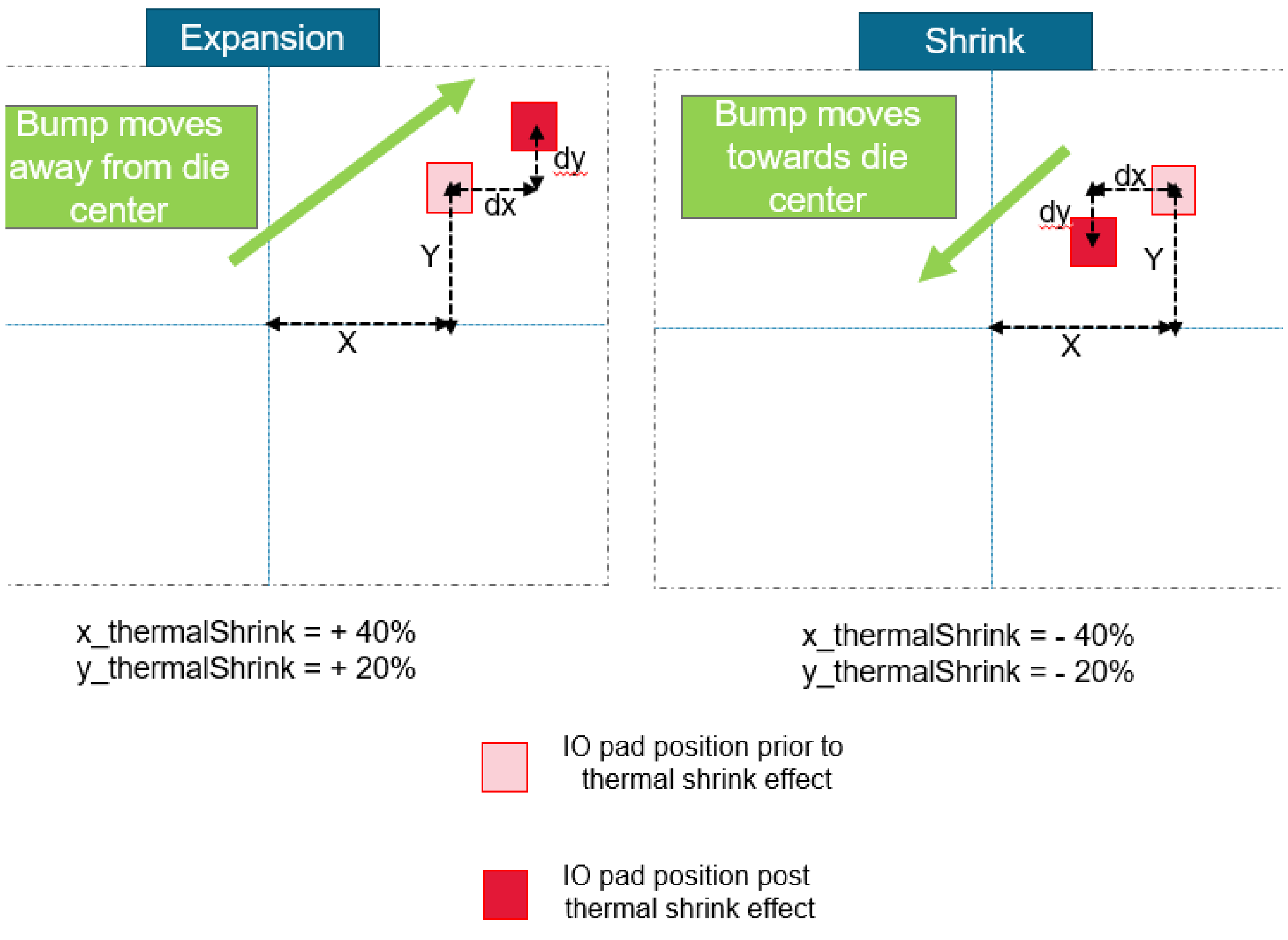

- **Negative Thermal Shrink Factor** (thermal shrink) is applicable when the package expands more than the die abstract.
- **Positive Thermal Shrink Factor** (thermal expansion) is applicable when the abstract die expands more than the package.

Thermal shrink is applied on IO pads as a factor of the distance of IO pads from center of the die.

```
dx = X * x_thermalShrink
```


© 2023 Cadence Design Systems, Inc.

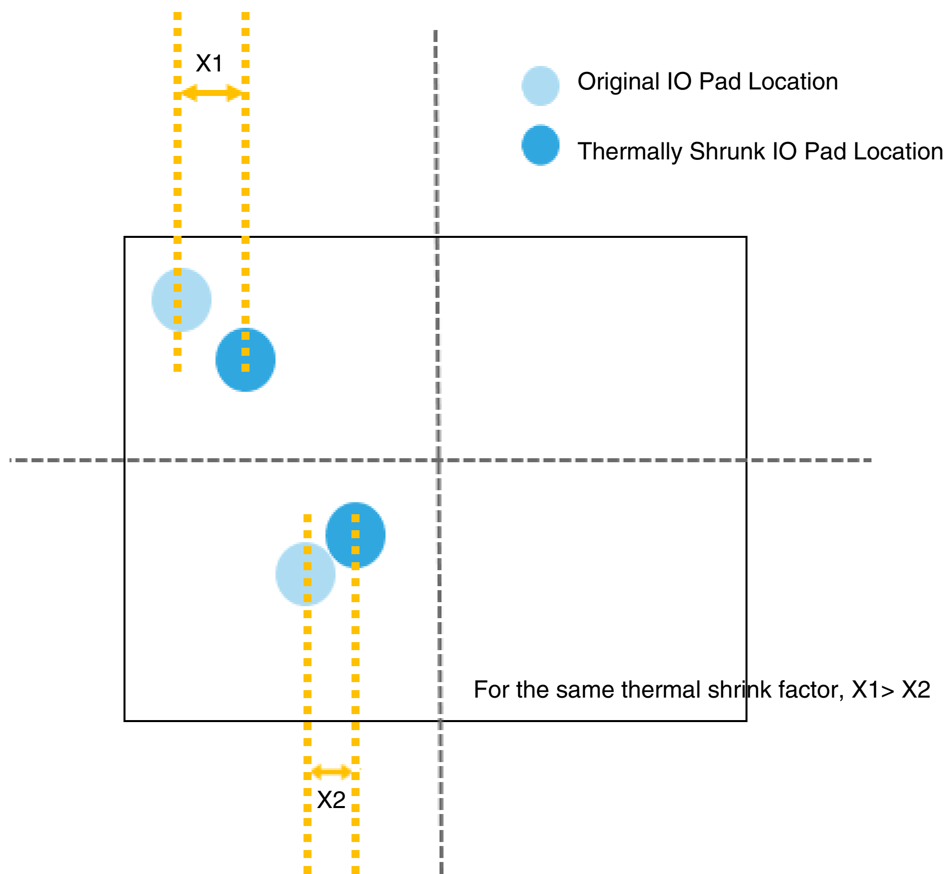
```
dy = Y * y_thermalShrink
```





© 2023 Cadence Design Systems, Inc.

Closer the IO pad is to center of the die, less is the effect of thermal shrink factor. Farther the IO pad is from the center of the die, greater is the effect of thermal shrink factor.



*Important*

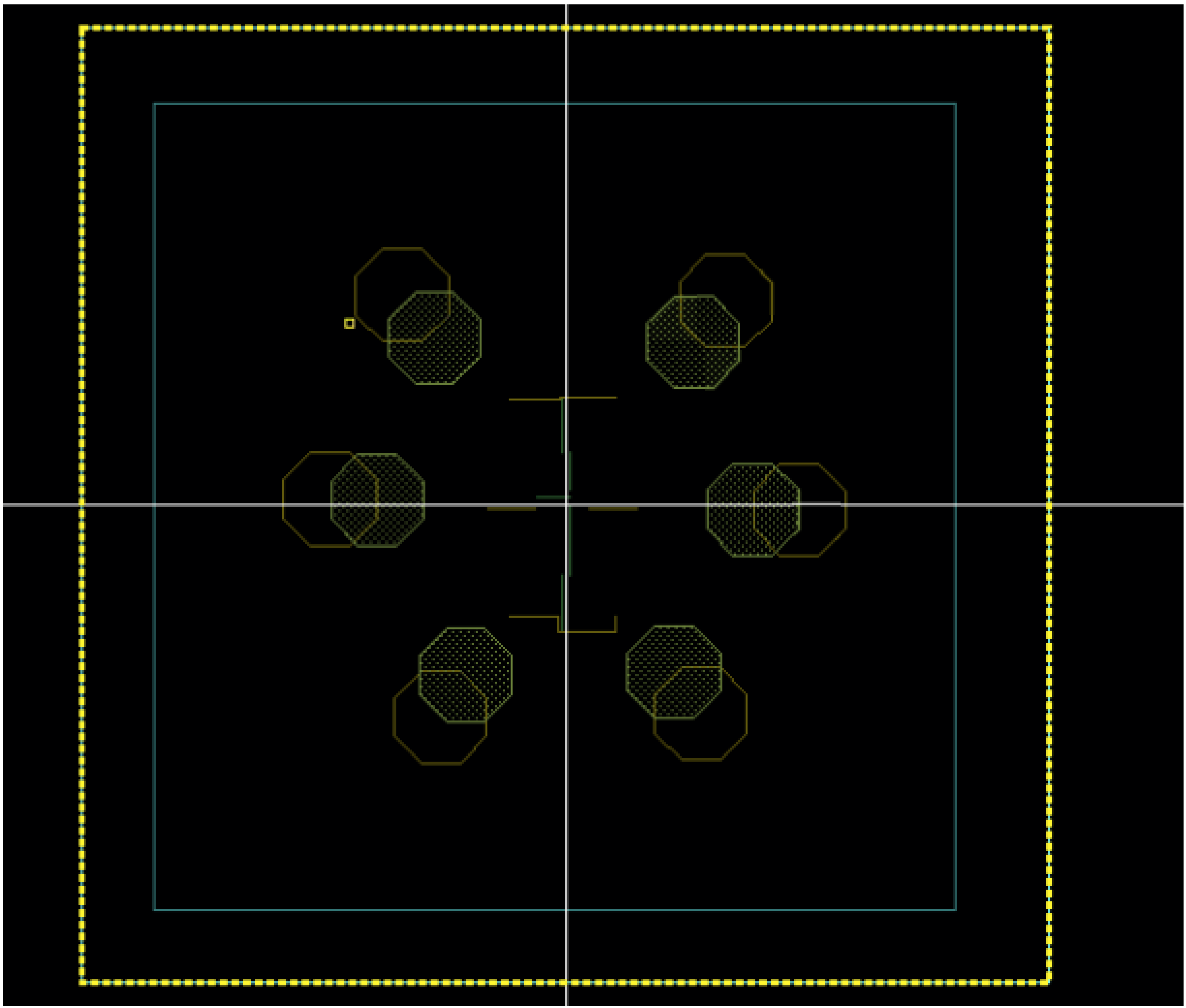
If the IO pad is present at the exact center of the die, there is no effect of thermal shrink factor on that specific IO pad.

### Impact of Applying Thermal Shrink Factor

- **Edit-in-Concert:** When thermal shrink is applied to a TILP die, the IO pads in the die are adjusted, but the IO pads in the corresponding die layout view remain at their actual positions. This may lead to visual misalignments in the Edit-in-Concert mode. When you


© 2023 Cadence Design Systems, Inc.

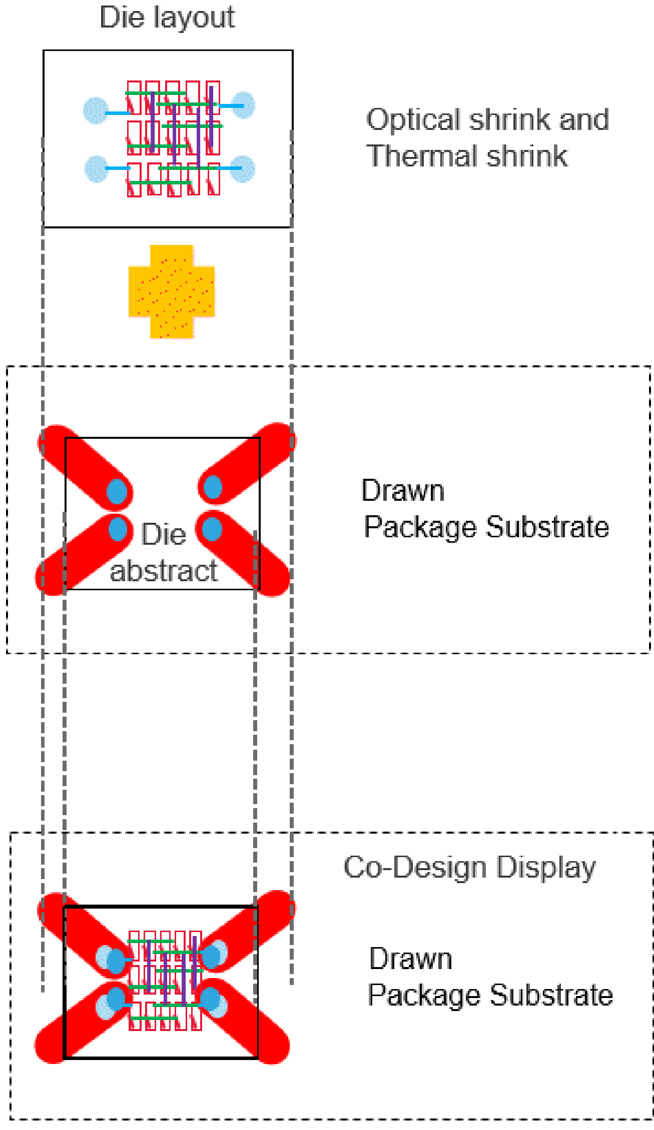
launch the Edit-in-Concert mode, a warning message indicating the cause of these misalignments is displayed.



- **LVA Checker and Fixer:** When thermal shrink is applied, despite visual misalignments in the Edit-in-Concert mode, the LVA checker does not report any misalignment violations and the Annotation Browser does not display any violation markers.
- **Optical Shrink:** Optical shrink factor reduces the size of the entire die, whereas thermal shrink factor only shrinks (or expands) the IO pads. When both optical and thermal shrink


© 2023 Cadence Design Systems, Inc.

factors are specified, the optical shrink factor is applied first. Thermal shrink factor is then applied to the optically shrunk die.




### *Related Topics*

Thermal Shrink Factor

Key Edit-in-Concert Views


© 2023 Cadence Design Systems, Inc.

Checking and Fixing IO Pad Locations

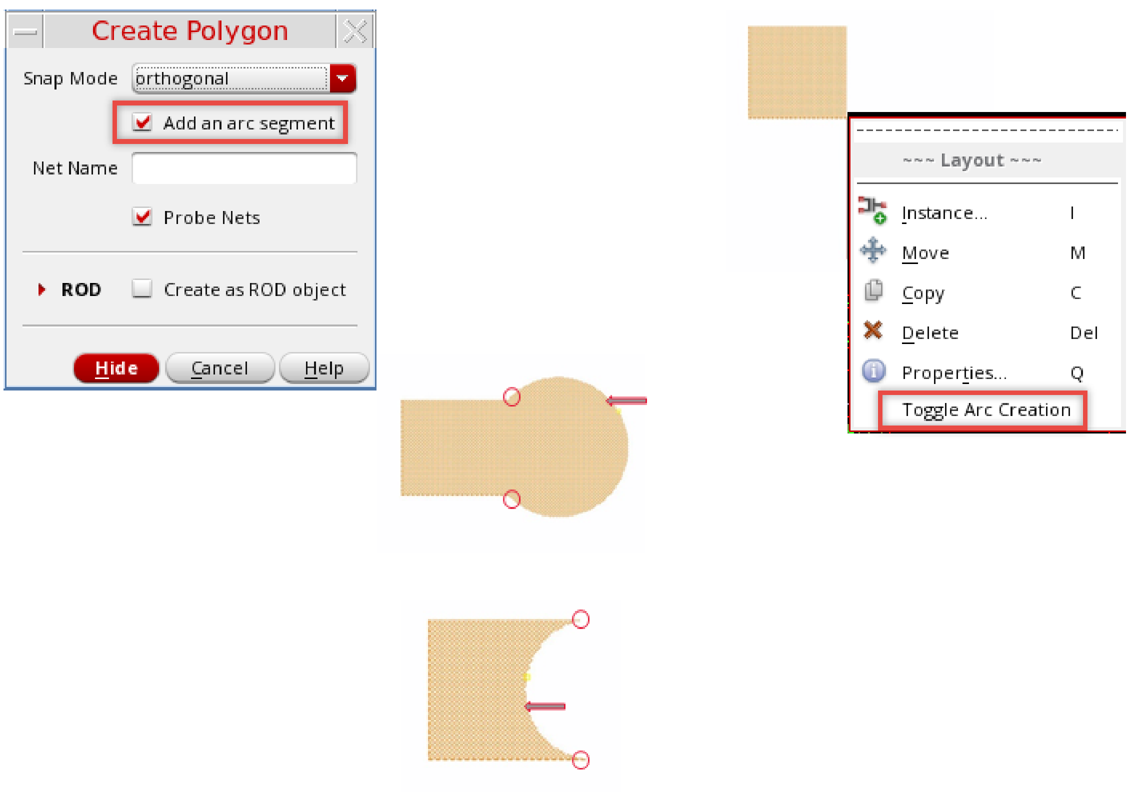
Edit Instance Properties Form (Die/Package TILP Parameters)

# Creating a Curved Polygon

The Virtuoso Layout recognizes the curved shapes, polygons, rectangle, and paths. You can edit the curved shapes in various ways, such as updating the shape, rotating, or stretching. Additionally, you can modify a regular shape to curved shape and edit the curved shape to add new arcs.

To create a curved polygon in a package layout:

1. Click *Create – Shape – Polygon*. The Create Arc Polygon form appears.
2. Add an arc segment by selecting *Add an arc segment*. The curved polygon and the arc segment are created using the LPP selected in the Palette assistant.
3. Create a polygon on the canvas. You can right-click and select *Toggle Arc Creation* to complete the arc.





© 2023 Cadence Design Systems, Inc.

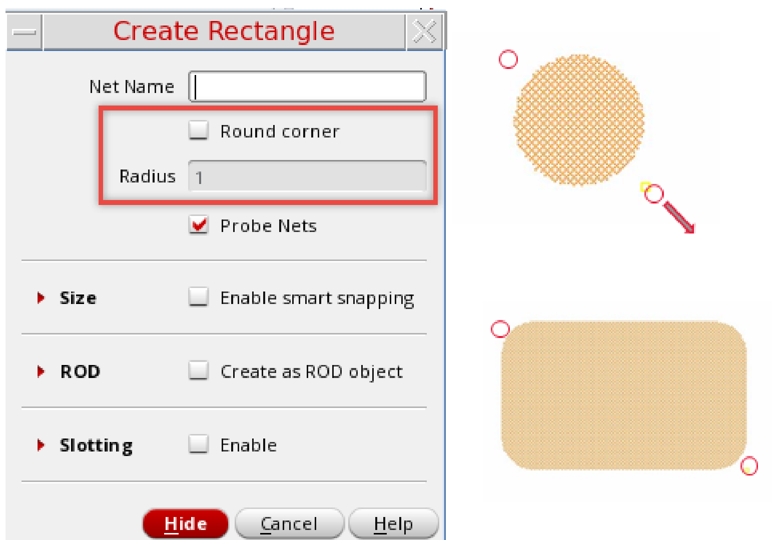
***Related Topics***

Creating a Curved Rectangle

# Creating a Curved Rectangle

To create a curved rectangle in a package layout:

1. Click *Create – Shape – Rectangle*.
2. Select the *Round corner* check box.
3. Specify the value for the *Radius* field. The smallest possible shape is a circle of the given radius.




***Related Topics***

Creating a Curved Polygon


© 2023 Cadence Design Systems, Inc.

# Interactive Routing in Virtuoso RF Solution

The interactive and assisted routing capability of the Virtuoso RF Solution enables you to route connections interactively within the Virtuoso environment. It provides efficient ways to interactively and automatically route connections in order to meet critical design constraints and rules. Interactive routing lets you do the following:

- Complete critical nets before automatic routing.
- Complete incomplete nets left after automatic routing.

Apart from the usual capability of routing, interactive routing has been enhanced to support curved paths that are available in cellviews along with package constraints. The bindkey `P` that is used for the Create Wire command is defined to create a curved path when you have a package design.

## Managing Curved Shapes in Wire Editor

The Wire Editor in the Virtuoso RF Solution has been enhanced to create curved paths instead of providing the choice between normal OA paths and pathsegs. Curved paths are OA paths with **curved** extensions. The Virtuoso RF Solution enhancements include additional snap modes (L45LongFirst and L45AngleFirst), padstack exit from intermediate layers, support for instances used as pins, padstack and via center snapping.

## Push-and-Shove Feature

The push-and-shove feature applies to routing objects, such as wires and vias. Static objects, such as pins, bumps, and bond fingers, are not considered routing objects. This capability lets you push wires out of the way by either automatically adjusting the new wire or by pushing an existing wire.

Enabling this feature enhances correct-by-construction design and makes the design DRC- and shorts-aware.

### *Related Topic*

Pushing and Shoving in Interactive Routing

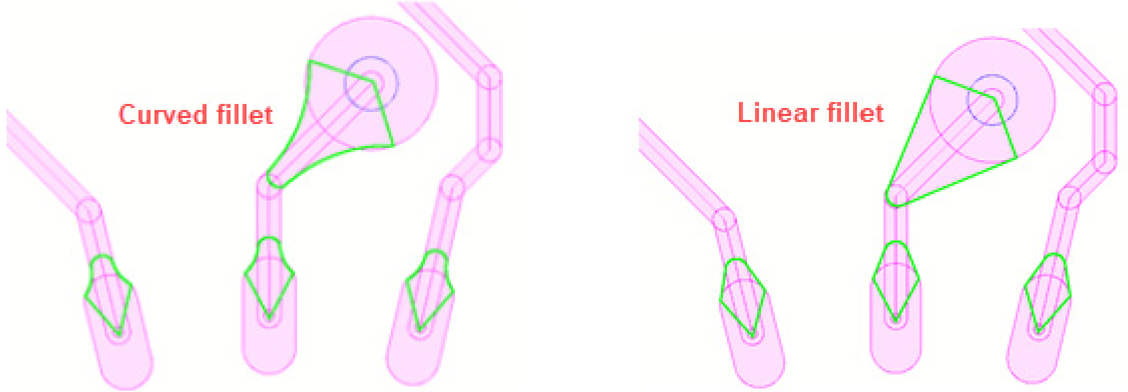
Supporting Curved Paths


© 2023 Cadence Design Systems, Inc.

# Fillet Creation Between Curved Path and Other Objects

A fillet is an extra etch added to improve wire to pad or wire to via connection. It is created on analog and high-speed circuits, or areas of a design where shock and vibration to the design might disrupt connections. In Virtuoso, a fillet is created as a curved polygon.

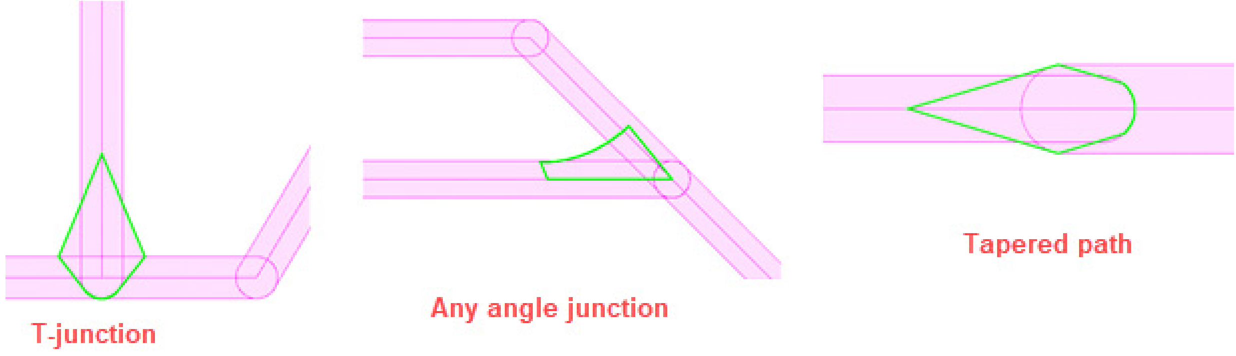
A fillet is created from the center of the shape to the intersection of the shape and can be either curved or linear.




A fillet is inserted between a curved path and other objects, such as other curved paths, pins, vias, and bond fingers.

- Other Curved Paths

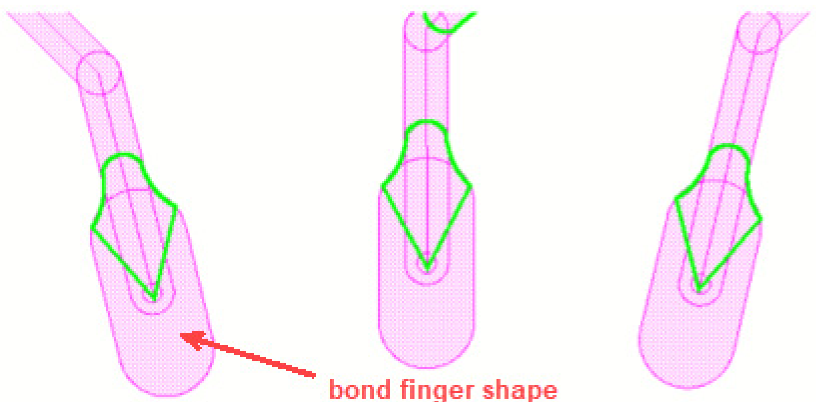
  The following figure shows fillet creation between a curved path and other curved paths, including T-junctions, any angle crossing, or intersections. Tapered paths are also considered.




- Bond Fingers

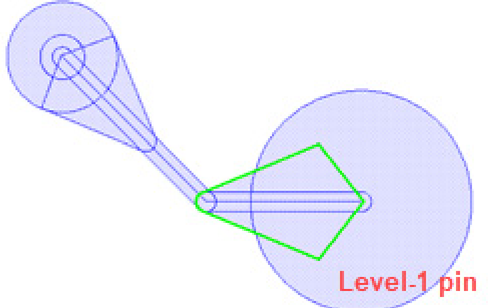

© 2023 Cadence Design Systems, Inc.

The following figure shows fillets created between a curved path and bond fingers.



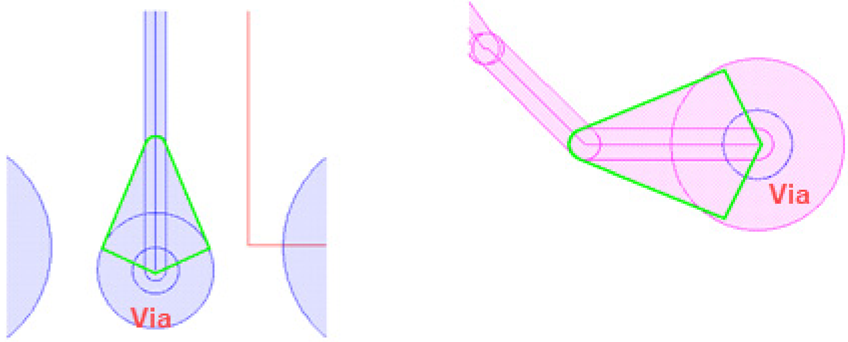

- Pins

  The following figure shows a fillet created between a curved path and a pin.




- Vias

  The following figure shows a fillet created between a curved path and a via.





© 2023 Cadence Design Systems, Inc.

Fillets are created using the *Fillet* icon on the Metal Density toolbar and the scope and parameters are defined using the *Fillet* tab on the Metal Density Options form.

***Related Topics***

Create Fillet

Fillet Creation

Fillet tab

# Void Shapes

Voiding is an important step in creating the package layout that lets you route on ground/power planes by automatically insulating signal nets.

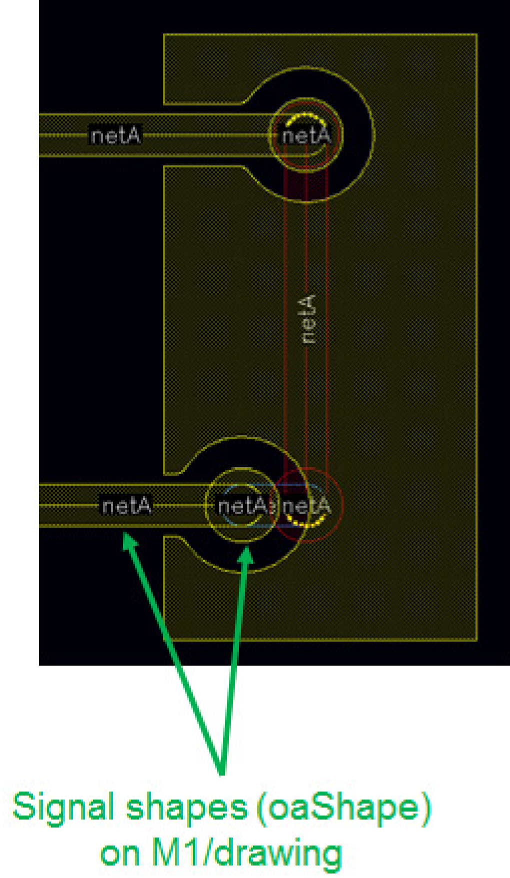
Ground/power planes are typically used in packages to reduce the impedance of the power/ground supply network by lowering ohmic resistance and shielding nearby components from EM noise. If routing for other signals cannot be completed on the layers other than the ones devoted to power and ground, they must be completed on power and ground planes.

The Virtuoso RF solution includes a void generator that lets you generate void shapes in package layout designs. Void shapes are automatic shapes that insulate signal shapes (routing) from dynamic shapes such as power or ground planes. The power and ground planes are created on a dynamic purpose and the void shapes cut holes on the dynamic shape around the signal shapes. The effective conducting shape on power or ground is the Boolean subtraction of the void shapes from the dynamic shape.

Voiding helps designers route on the ground/power planes by automatically insulating signal nets. Void the shapes by using *Module – Void Dynamic Shapes*. It deletes the existing void shapes and creates voids to insulate signal shapes. By insulating the signal shapes from the ground/power planes, it eliminates shorts between shapes on signal and power/ground nets.


© 2023 Cadence Design Systems, Inc.

You can also click *Module – Void Selected Dynamic Shapes* to void a few selected shapes.

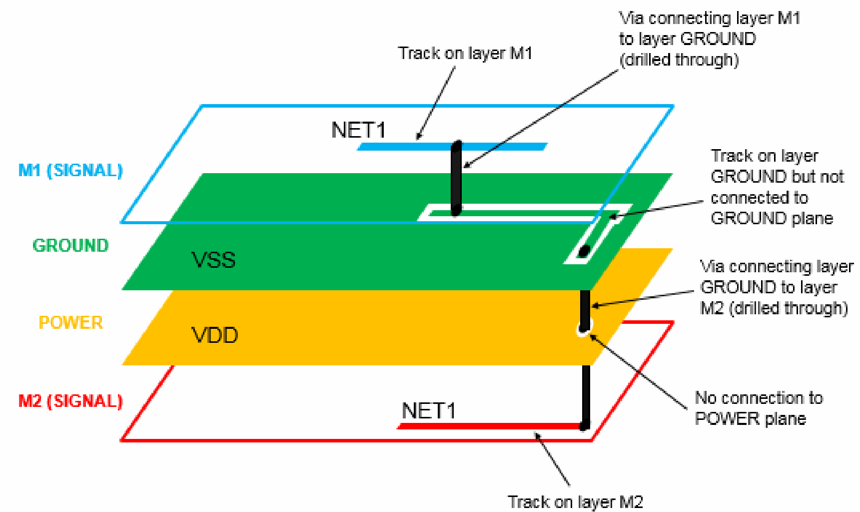



Before generating void shapes, you must create the required dynamic shapes. The technology file must define the dynamic purposes before dynamic shapes can be created.

**Note:** Voiding of shapes is done only when the fabric type is specified as package, board, or multi-chip module (MCM) in the technology file.


© 2023 Cadence Design Systems, Inc.

After creating the dynamic shapes and signal shapes, you can generate void shapes. Void shapes are generated to insulate signal paths or vias that have a different connectivity from the plane they are going through, as shown in the following figure:




In this example, a via connects layers M1 and GROUND; a second via connects layers GROUND, POWER, and M2. Voids are generated around the vias and paths in layer GROUND and around the via on layer POWER to insulate the NET1 signal from the respective planes.

**Note:** Power and ground planes are used as reference planes to reduce the impedance of the power and ground supply and EM noise (shielding). You can use dynamic shapes for shielding components or to fill entire layers with conductor as voltage distribution (embedded) planes.

### *Related Topic*

Import Libraries and ICs

Void Shape Generation

Convert Selected Dynamic Shapes

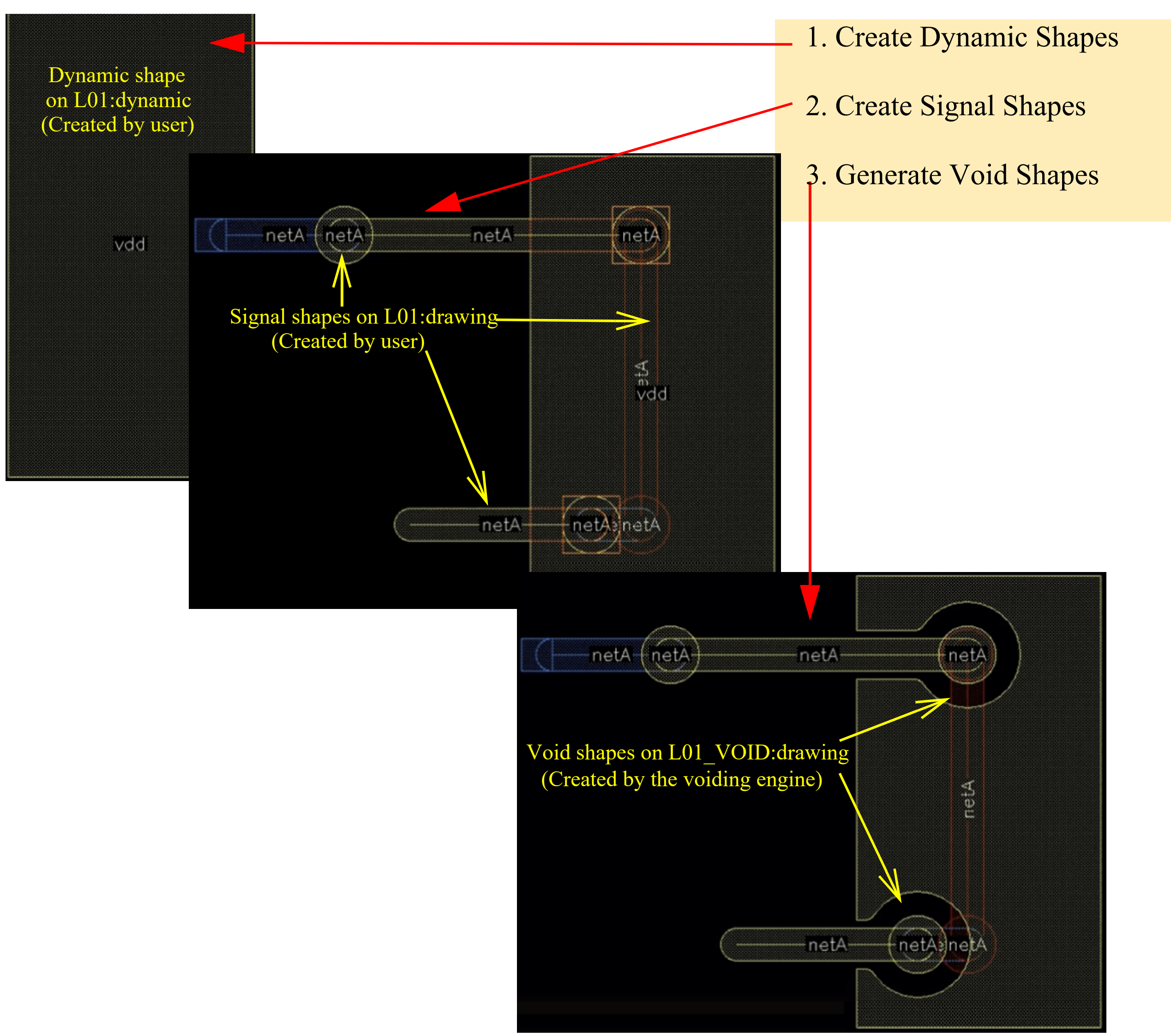
Package Constraints Supported by the Void Generator

Dynamic Shape Priority


© 2023 Cadence Design Systems, Inc.

# Void Shape Generation

The following diagrams demonstrate the flow for generating voids:




## Create Dynamic Shapes

Dynamic shapes are the shapes on purpose `dynamic`, for example `(L01 dynamic)`. Unlike signal shapes, dynamic shapes retain their connectivity, irrespective of the `lxStickyNet` property.


© 2023 Cadence Design Systems, Inc.

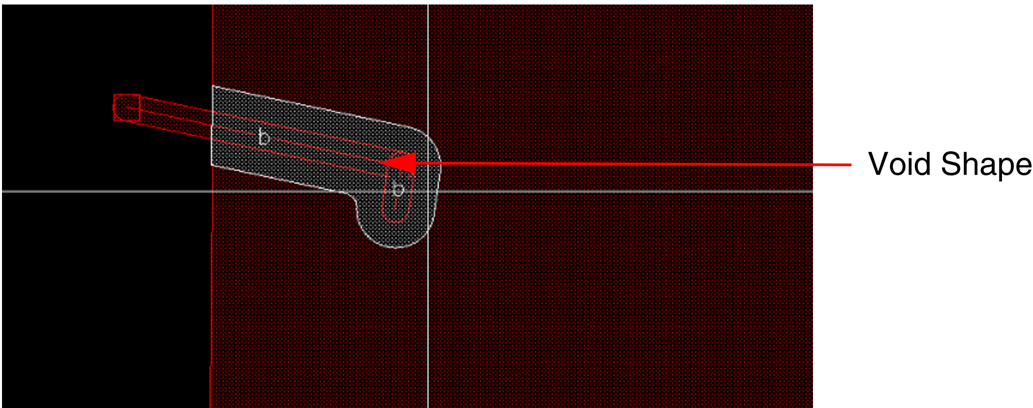
## Create Signal Shapes

Signal shapes are the shapes on purpose `drawing`, for example `(L01 drawing)`. Signal shapes are routing shapes such as pins, paths, and vias.

Example: Dynamic shape on `(L01 dynamic)` on net A and signal shapes on `(L01 drawing)` on net B.

## Generate Void Shapes

Choose *Module – Void Dynamic Shapes* to generate void shapes. Void shapes are shapes created by the void generator on a specific layer, for example `(L01_VOID drawing)` for a dynamic shape on `(L01 dynamic)`. Void shapes are stored in the database.




For each dynamic shape on `(layer dynamic)`, the void generator searches for overlapping signal shapes in `(layer drawing)`. The generator creates void shapes on `(<layer>_VOID drawing)` for shapes on different nets. In other words, if the nets of the dynamic and signal shapes are different, the void generator effectively removes parts of metal around the signal shape. The conducting shapes, which are the metal shapes left after removing the void shapes from the dynamic shapes, are the effective metal shapes that will be manufactured.

*Important*

The *Module – Void Dynamic Shapes* command is a batch command. Therefore, void shapes will be out-of-date after making any change to the shapes, such as moving or resizing them, and must be regenerated manually.


© 2023 Cadence Design Systems, Inc.

***Related Topic***

Void Shapes

Convert Selected Dynamic Shapes

Package Constraints Supported by the Void Generator

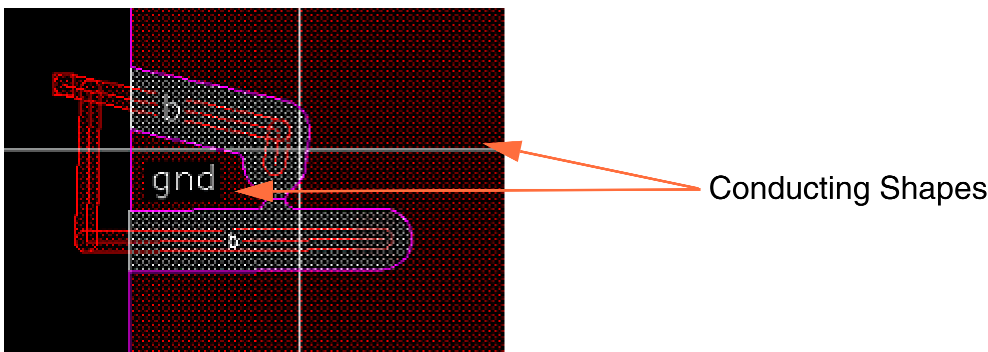
Dynamic Shape Priority

# Convert Selected Dynamic Shapes

Choose *Module – Void Selected Dynamic Shapes* to convert dynamic shapes to static shapes.

## Smooth and Trim Void Shapes

The void generator smooths the conducting shapes that result from voiding. This smoothing is required for manufacturing ability of the design. Sharp corners (of angles lower than 90 degrees) are rounded and the `pkgMinAperture` constraint is enforced. Void generation might split a dynamic shape into multiple conducting shapes or “islands,” as shown in the following image:

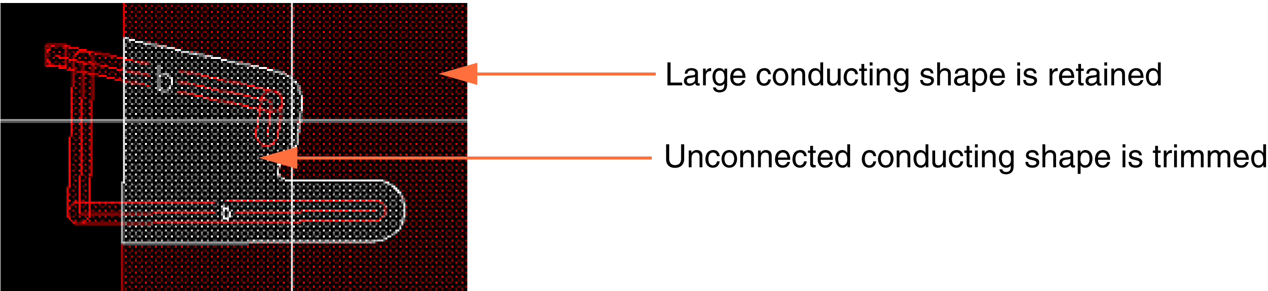


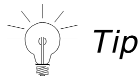

When a `pkgMinArea` constraint is present, the void generator trims the conducting shapes that are smaller than `pkgMinArea` value.

Use the `voidGeneratorTrimUnconnectedShapes` environment variable to control trimming of unconnected conducting shapes. The default value is `t`. In this state, the unconnected conducting shapes are trimmed, as shown below. The small conducting shape


© 2023 Cadence Design Systems, Inc.

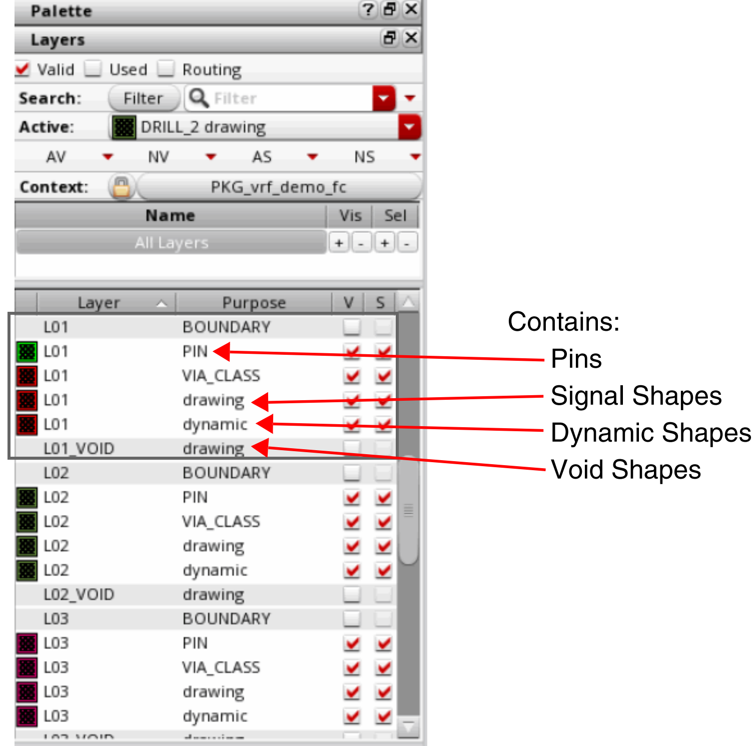
is trimmed out because it is not connected to any pin. The large conducting shape is retained because either it is connected to a pin or it is the largest conducting shape.



*Tip*

The following image shows the various layers in the Palette assistant.




### *Related Topic*

[leConvertSelectedDynamicShapes]

[Void Shapes]


© 2023 Cadence Design Systems, Inc.

[Void Shape Generation](#)

[Package Constraints Supported by the Void Generator](#)

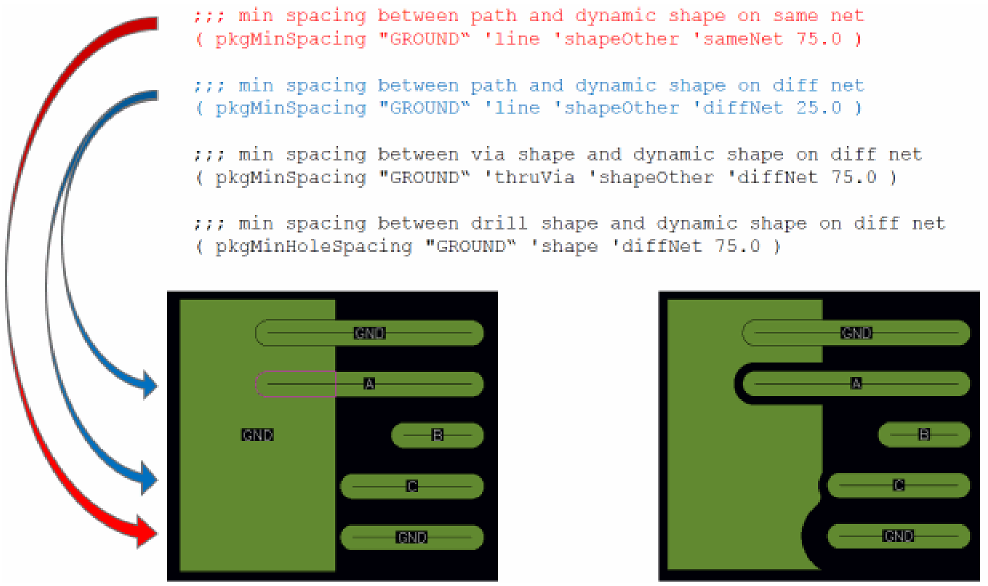
[Dynamic Shape Priority](#)

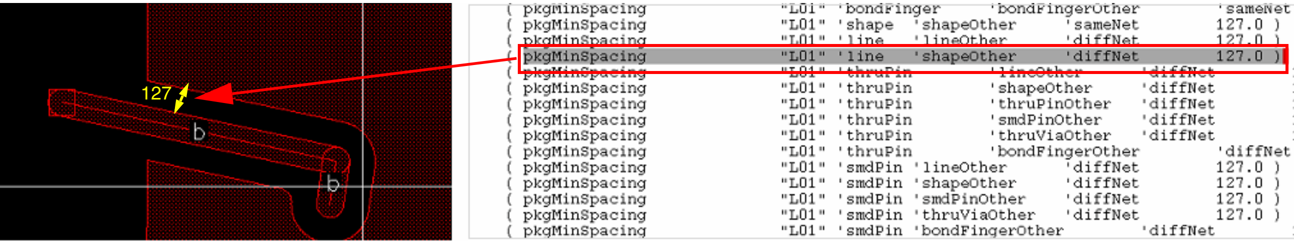
[voidGeneratorTrimUnconnectedShapes](#)

# Package Constraints Supported by the Void Generator

The void generator honors the following constraints:

- **pkgMinSpacing:** Specifies the minimum spacing between a signal shape and a dynamic shape. Therefore, this value indicates the width of voids. The `pkgMinSpacing` values are defined in the technology file. The values differ for different types of signal object and nets:

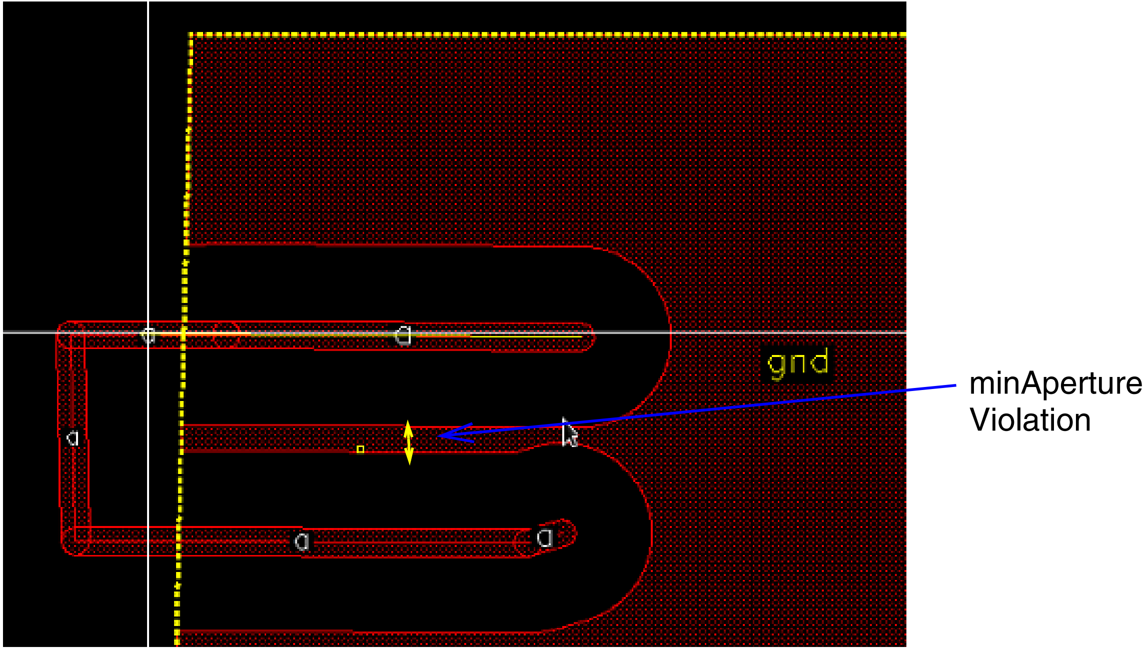


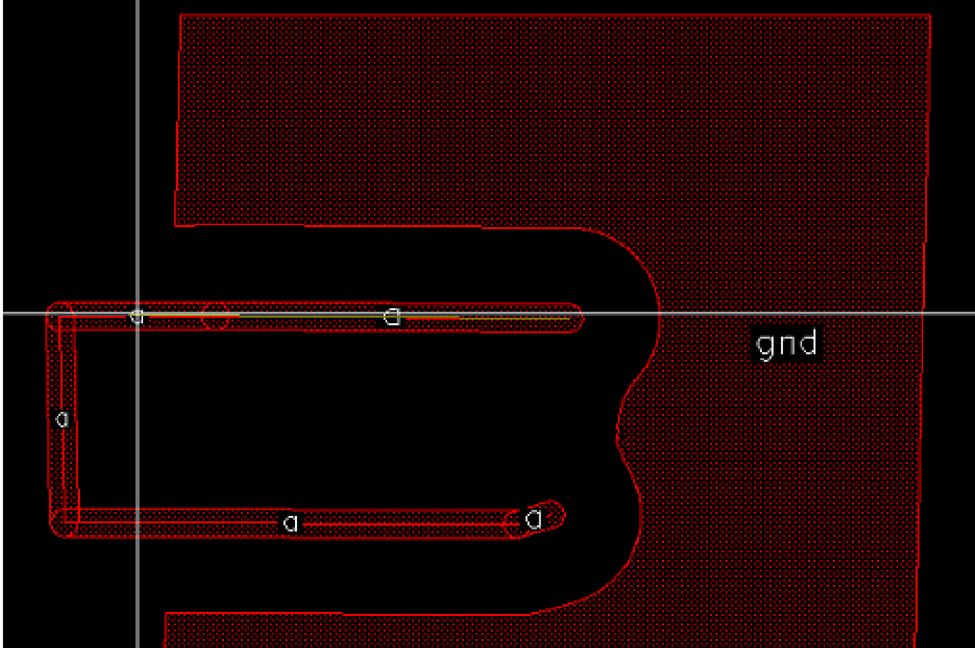


© 2023 Cadence Design Systems, Inc.

- **pkgMinAperture:** Specifies the minimum width of a piece of metal. Such pieces can be created by voiding a dynamic shape.



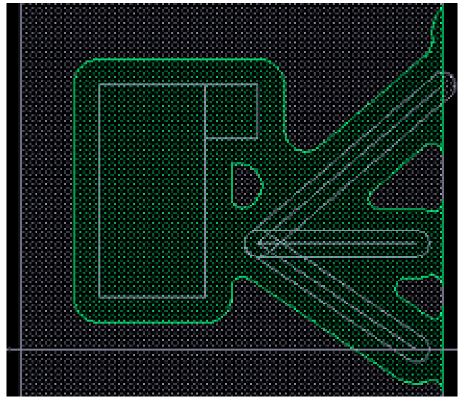

If a part of conducting shape is too narrow, the void generator removes it so that there is no violation of the `minAperture` constraint, as shown in the following image:

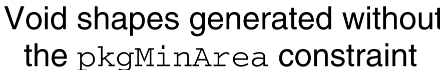


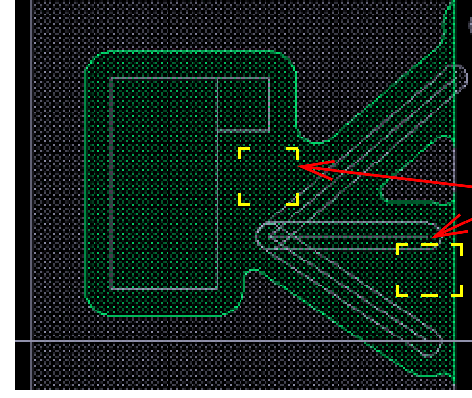


© 2023 Cadence Design Systems, Inc.

- **pkgMinArea:** Specifies the square root of the minimum area of a metal shape. This constraint applies to the conducting shapes resulting from voiding dynamic shapes. The void generator generates void shapes so that there is no violation of the `pkgMinArea` constraint. The `pkgMinArea` is applicable to individual layers. To apply the constraint to all layers, use `pkgMinAreaDefault`.




Void shapes generated without the `pkgMinArea` constraint


Voids shapes generated with a `pkgMinArea` constraint

The voiding engine does not generate conducting shapes that violate the `pkgMinArea` constraint

**Note:** To match the use model in the Allegro platform, all conducting shapes smaller than `minArea` are voided, even if there are no remaining parts of the dynamic shape.

### *Related Topic*

pkgMinSpacing

pkgMinAperture

pkgMinArea

Void Shapes

Void Shape Generation

Convert Selected Dynamic Shapes

Dynamic Shape Priority

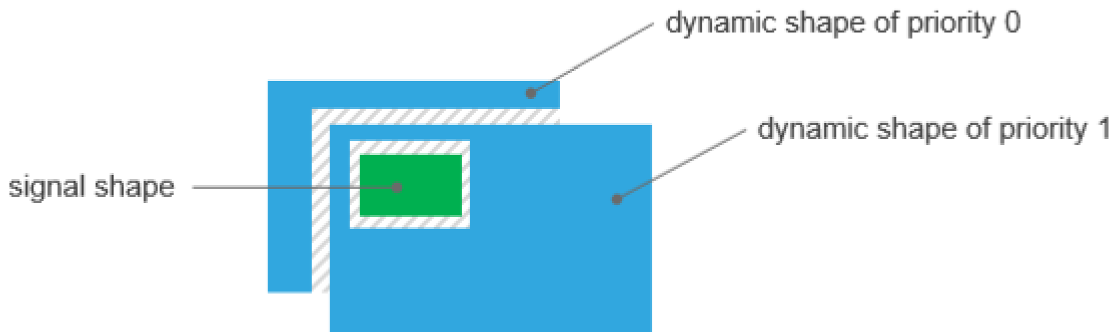

© 2023 Cadence Design Systems, Inc.

# Dynamic Shape Priority

Dynamic shapes have a priority attribute that determines the order in which the overlapping dynamic shapes are voided.

The dynamic shape priority is associated with each dynamic shape for a given metal layer in a given layout. A dynamic shape with a higher priority is cut out from the dynamic shape with a lower priority.

The following figure represents a simple overlap scenario where a dynamic shape is being overlapped by a signal shape and the shape, in turn, overlaps another dynamic shape of lower shape priority.




To insulate the overlapping dynamic shapes, void shapes are created. These void shapes:

- Insulate the dynamic shapes from all the overlapping signal shapes.
- Insulate a dynamic shape from all the overlapping dynamic shapes that have higher priorities.

***Related Topics***

Void Shapes

Void Shape Generation

Package Constraints Supported by the Void Generator

Dynamic Shape Priority

Convert Selected Dynamic Shapes

vrfHiUpdate


© 2023 Cadence Design Systems, Inc.

vrfRaisePriority


© 2023 Cadence Design Systems, Inc.

# Extracting Connectivity Information from Package Layout

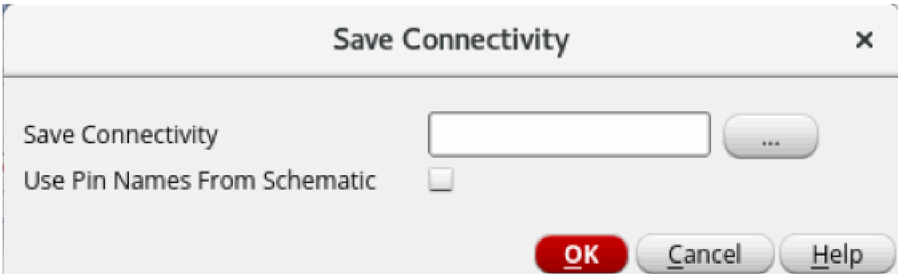
The Virtuoso RF Solution lets you extract connectivity information from package layouts. This information is saved to a text file, which can later be applied to other package schematics or layouts. A connectivity file stores connectivity information in the following format:

```
Net_Name<space>Instance_Name1/Pin_Name<space>Instance_Name2/
Pin_Name
```

For example, `REF i1/OUT i2/IN i3/IN i4/IN?`

To extract connectivity information from the current package layout:

1. Choose *Module – Connectivity – Save Connectivity* to display the Save Connectivity form.




2. Specify a name and the location to save the connectivity file in the *Save Connectivity* field. You can also click the browse button, select the required folder, and then specify a a file name.
3. Select *Use Pin Names From Schematic* to use pin names from schematic instead of pin numbers in an instance.
4. Click *OK*.


© 2023 Cadence Design Systems, Inc.

Connectivity information from the current package layout is saved to the specified text file. The following image shows a connectivity file.

```
## File automatically by Virtuoso.
## Source: PKG_golden_bk pkg_sch_golden layout
## Date: Feb 18 16:08:58 2021

VBIAS_PA BGA/D6 I9/PIN1_EXTRA1 I9/PIN1_EXTRA2 I9/PIN1_EXTRA0 C5/PLUS
FINGER_36/pin1 FINGER_35/pin1 FINGER_34/pin1 BONDWIRE_32/pin1 BONDWIRE_31/pin1
BONDWIRE_30/pin1 BGA/D7
VDD L23/PLUS L21/PLUS R1/PLUS R4/MINUS BGA/C1 BGA/D1
VSS_PA BGA/D5 C4/MINUS I9/VSS_EXTRA0 I9/VSS_EXTRA1 I9/VSS_EXTRA2 I9/VSS_EXTRA3
C5/MINUS C8/MINUS L5/MINUS L7/MINUS C1/MINUS FINGER_20/pin1 FINGER_19/pin1
FINGER_18/pin1 FINGER_17/pin1 BONDWIRE_24/pin1 BONDWIRE_23/pin1
BONDWIRE_22/pin1 BONDWIRE_21/pin1 BGA/B5 BGA/B7 BGA/C5 BGA/C6 BGA/C7
VSS BGA/D4 BGA/D3 BGA/D2 I1/VSS_EXTRA0 I1/VSS_EXTRA1 I1/VSS_EXTRA2
I1/VSS_EXTRA3 L4/MINUS [illegible]MINUS FINGER_5/pin1 FINGER_4/pin1 FINGER_3/pin1
FINGER_2/pin1 BONDWIRE_5/pin1 BONDWIRE_4/pin1 BONDWIRE_3/pin1 BONDWIRE_2/pin1
BGA/A6 BGA/B2 BGA/B3 BGA/B4 BGA/C2 BGA/C3 BGA/[illegible]
MIX_IN K1/unbal BGA/B1
RF_OUT_PA I9/RF_OUT L7/PLUS FINGER_33/pin1 BONDWIRE_29/pin1 BGA/A7
VDD_PA BGA/B6 I9/VDD_EXTRA3 I9/VDD_EXTRA2 I9/VDD_EXTRA1 I9/VDD_EXTRA0 C1/PLUS
FINGER_32/pin1 FINGER_31/pin1 FINGER_30/pin1 FINGER_29/pin1 BONDWIRE_36/pin1
BONDWIRE_35/pin1 BONDWIRE_34/pin1 BONDWIRE_33/pin1
VBIAS_LNA R5/PLUS I1/V_bias I1/res_bias R3/PLUS FINGER_24/pin1 FINGER_9/pin1
BONDWIRE_13/pin1 BONDWIRE_11/pin1 BGA/A3
GND K0/unbal_gnd K1/unbal_gnd BGA/A1 BGA/A2
NET338 L23/MINUS I1/res_m3 FINGER_6/pin1 BONDWIRE_7/pin1
NET347 I1/res_gatem2 R4/PLUS FINGER_7/pin1 BONDWIRE_18/pin1
NET348 I1/resgate_m3 R1/MINUS FINGER_14/pin1 BONDWIRE_19/pin1
NET346 L21/MINUS I1/res_m4 FINGER_15/pin1 BONDWIRE_20/pin1
NET335 L2/MINUS I1/inputm1_scs FINGER_16/pin1 BONDWIRE_6/pin1
NET344 R3/MINUS L2/PLUS I1/input_m1 FINGER_22/pin1 BONDWIRE_9/pin1
NET337 I1/source_m1 L6/PLUS FINGER_23/pin1 BONDWIRE_10/pin1
```

Net_Name Instance_Name1/Pin_Name Instance_Name2/Pin_Name

You can manually edit or create a new connectivity files as per your requirements.

### *Related Topic*

Creating and Saving Connectivity Information in Package Schematic

layoutConnectivityFile

layoutConnectivityPinNamesChkBox


© 2023 Cadence Design Systems, Inc.

# 6

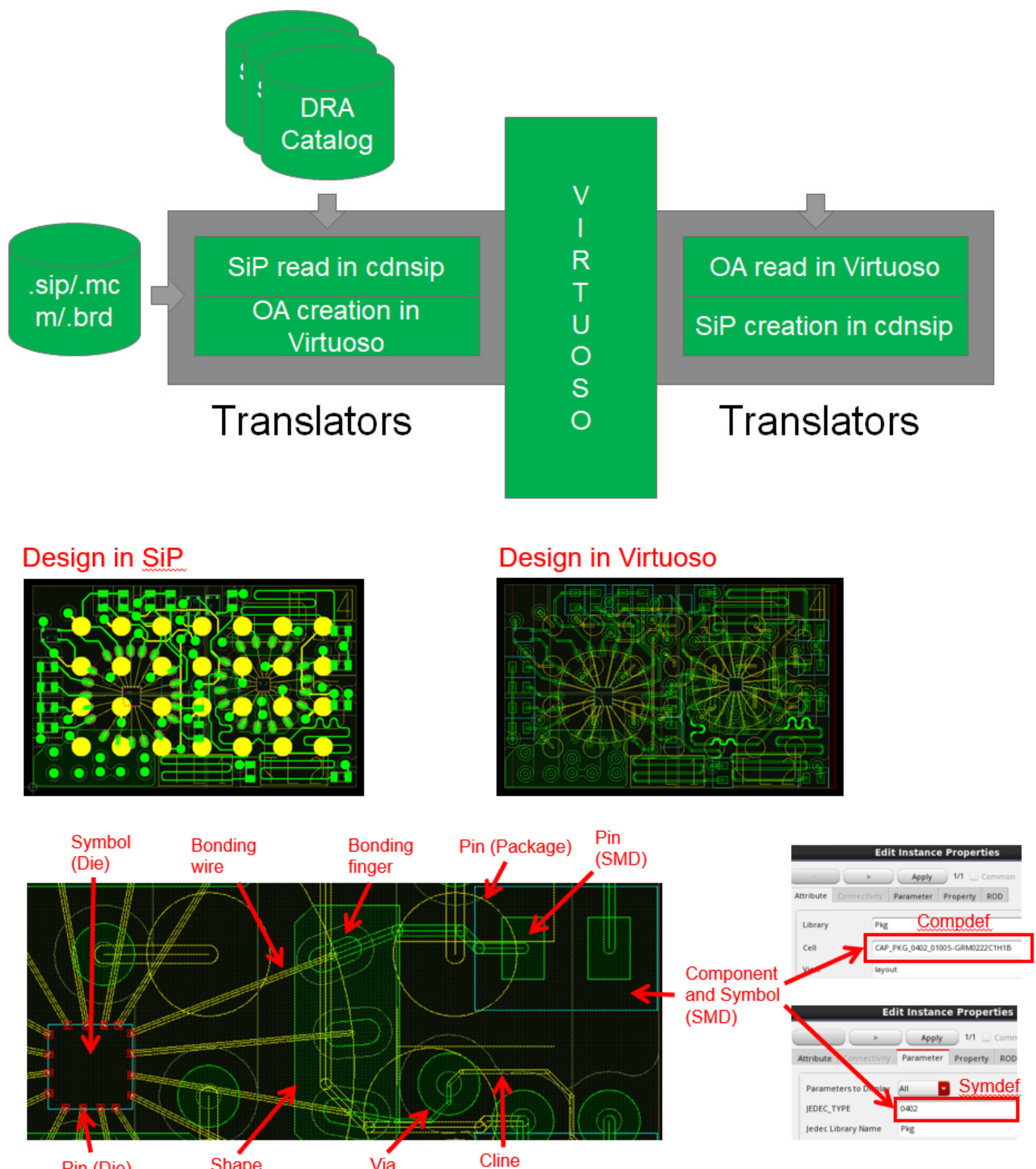
# Interoperability with SiP

The Virtuoso RF Solution is a schematic driven package layout flow in Virtuoso that provides the capability to finish the package in Allegro. Therefore, interoperability with Allegro for package design should be robust to support seamless data transfer. The data models in Allegro and Virtuoso have significant differences and the gap needs to be bridged. The Allegro translators are invoked from Virtuoso and require a license for the SPB tools based on the type of the input or output database.

Due to the differences in the underlying SiP and OA databases, the object representations in SiP and OA are sometimes not entirely equivalent. Once the object has been translated from SiP to OA, it might not restore to its original SiP representation from the OA representation. Consequently, the translation of a SiP design to OA and back to SiP might be different from


© 2023 Cadence Design Systems, Inc.

the original one. However, the physical layouts created from the two design representations are equivalent from the manufacturing and analysis point of view up to a certain tolerance.




The following SPB layout design tools are involved in the export and import of the layout information:


© 2023 Cadence Design Systems, Inc.

- `.sip` file from Cadence SiP Layout (SiP)
- `.mcm` file from Allegro Package Designer (APD)
- `.brd` file from Allegro PCB Designer (Allegro)

### *Related Topic*

Allegro Translators

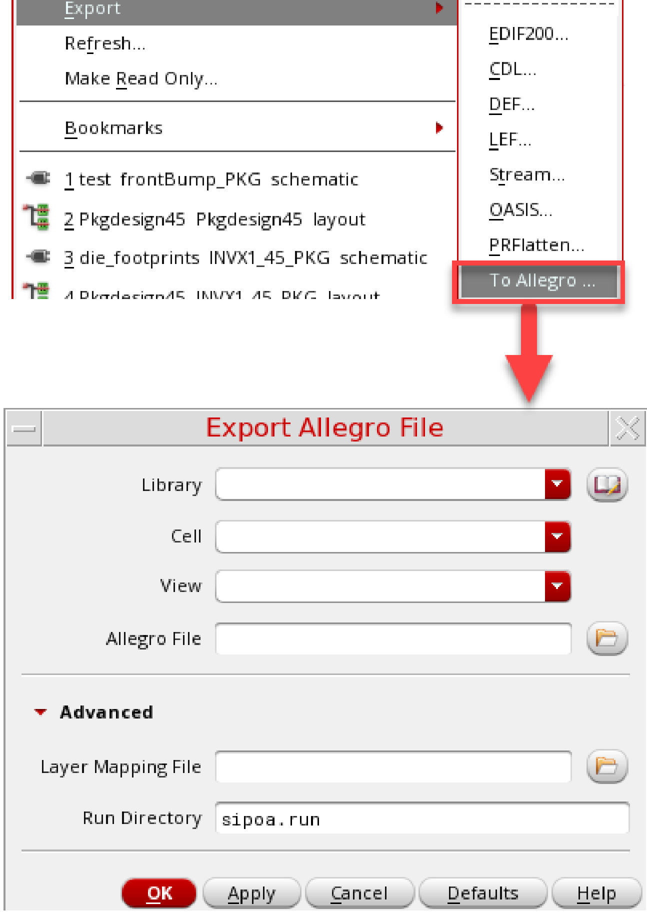

© 2023 Cadence Design Systems, Inc.

# Export Package Layout

When exporting the package layout, the Allegro translator creates a `.sip` file from the OA library (specified cellview) that has been created or modified to ensure it has the same structure.

During export to Allegro, the data is translated in the following two steps:

- Opening the OA design and writing its content or just the updating content to an intermediate file.
- Reading the intermediate file in the SPB tool and creating or updating the `.sip` design from its contents.




***Related Topic***

Interoperability with SiP


© 2023 Cadence Design Systems, Inc.

Importing the Package Layout

# Complete the Package Layout

After translation, launch SiP Layout Option to view and modify the package layout as needed.

Allegro layout editors provide manual and automatic tools for placing components, routing, and wire bonding.

## Manual and Automatic Placement

With manual placement, you can place elements individually or place elements of the same type during one pass.

In automatic placement, the layout editor places elements based on placement properties you assign that restrict or influence component positioning and part packaging.

In either manual or automatic placement, you can:

- Selectively identify certain parts for placement by attaching a placement “tag” to them.
- Optionally create a floor plan, which is a block diagram of rooms in a design. Rooms are rectangular areas that you create.

  You can use rooms to keep specific logical functions or related elements together in specific areas.

You may want to alternate manual placement with automatic placement. You can preplace sensitive or fixed parts manually, run automatic placement, and then rearrange some autoplaced parts. You could finish by optimizing the overall placement of a design with manual placement.

## Placement Tasks

You can use any combination of placement and swap tools. For example, component placement typically involves the following activities:

- Determining design requirements.
- Creating the items required for placement processing (such as grids or package keepin and keepout areas).


© 2023 Cadence Design Systems, Inc.

- Setting basic placement controls, such as package keepout and keepin areas, placement properties, and automatic placement parameters.
- Running manual placement alternately with automatic placement. You can run different iterations of automatic placement and view the placement results.
- Reviewing placement status and automatic placement results.

## Routing Tasks

The PCB layout editor provides tools that help you perform the following when routing physical designs:

- Interactive routing
- Automatic routing with Allegro PCB Router
- Glossing to improve the appearance and manufacturability of a physical design

The following steps describe the basic routing process. The basic routing flow—assuming automatic routing—is the following:

1. Prepare for routing:
    - Check layers to see that layer types and photo film types are correct.
    - Create internal shapes on planes.
    - Create blind and buried vias.
    - Set routing controls (routing areas, constraints and properties, grids, nets, and so on).
2. Manually route critical nets.
3. Define routing parameters in Allegro PCB Router to control how automatic routing functions.
4. Run Allegro PCB Router.
5. Review routing results.
6. Interactively finish or correct etch.
7. Gloss the design.
8. Optionally analyze the design for signal integrity or EMI.


© 2023 Cadence Design Systems, Inc.

***Related Topic***

Placing the Elements

Routing the Design

# Importing the Package Layout

When importing the package layout, the Allegro translator can be run in two modes, non-incremental and incremental mode.

In the non-incremental mode, it imports a `.sip` file to the OA library that has been created or modified to ensure it has the same structure as created during the export.

While importing, the translator creates an OA library file from an input database created by one of the SPB layout design tools. It translates the data in two steps:

- Open the design in the SPB tool and write its content (in incremental mode) to an intermediate file.
- Read the intermediate file in Virtuoso and create the OA data from its content (in incremental mode).

You can import a `.sip`, `.mcm`,`.brd`, or a catalog of `.dra` files. During import, the translators create a layer stack up and constraints in techdb and TILPs for SMDs or dies.

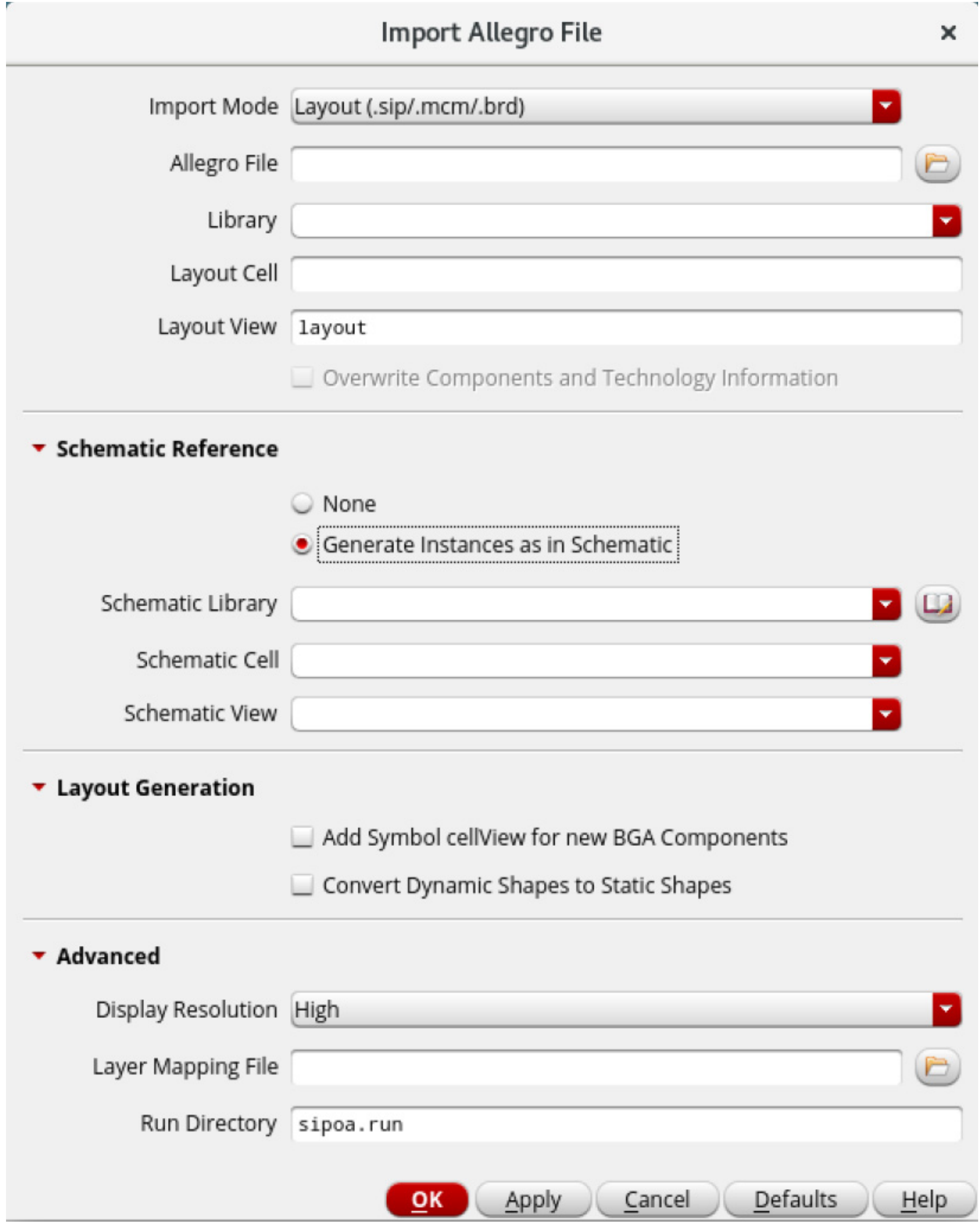
When you translate the data from SiP into Virtuoso, the relevant file is created by a package designer. It contains the BGA cell, the vias to be used for package routing, and the technology file used in SIP. This technology file is compiled into the Virtuoso tech.db that is in the package library. The package technology file is not the same as the IC technology file. In contrast, Edit-in-Concert in the Virtuoso RF Solution lets you seamlessly switch between the different technology files. You can also import the locations of IO pads on a die. This information is stored in a bump location file, which can be used later to create and align IO pads in an adjacent die.

To import the SiP file to Virtuoso, which contains the BGA and package techfile:

1. Click *CIW – File – Import – From Allegro…*


© 2023 Cadence Design Systems, Inc.

The Import Allegro File form opens.




2. Specify *Import Mode, Allegro File,* and *Library*. The *Layout Cell* and *Layout View* fields get populated automatically.

3. Select *Add Symbol cellview for new BGA Components*.

4. Click OK.


© 2023 Cadence Design Systems, Inc.

5. Check Library Manager to ensure that the new cells have been added appropriately.

When importing a layout from the Allegro platform to Virtuoso Studio, the ratio of DataBase Unit Per User Unit remains 1:1. This means that the resolution of a layout remains same during the import.

***Related Topic***

Interoperability with SiP

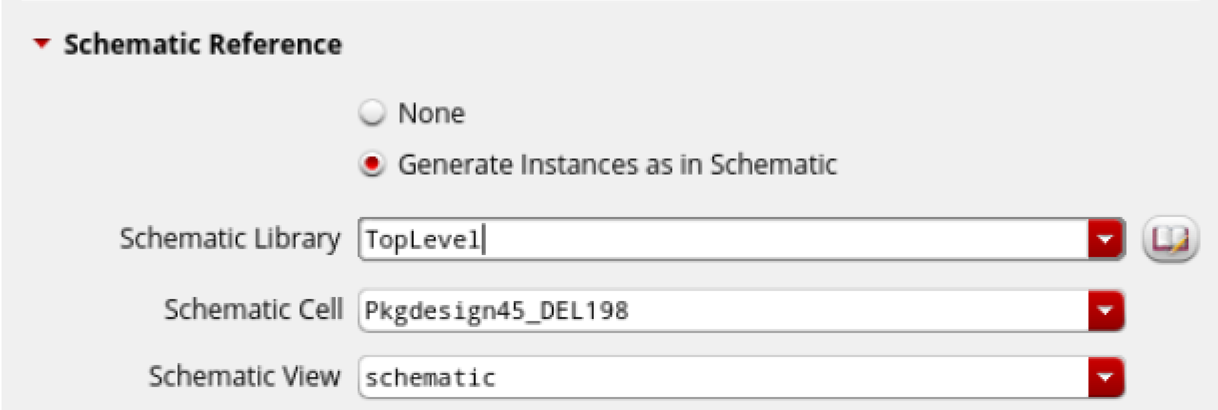
Export Package Layout

# Rebinding the Layout to a Schematic

This section covers rebinding the updated layout to package schematic.

1. Select *Generate Instances as in Schematic* in the *Schematic Reference* section of the Import Allegro File form.
2. Specify the schematic cellview that will be used for rebinding.

   This sets the reference for the top-level layout to the specified package schematic. The package schematic and top-level layout are bound to each other.




3. Click OK.
4. Check Library Manager to ensure that the new cells have been added appropriately.


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# 7

# Verify the Package

The connectivity, LVS, and DRD checks are being done on the layout. When the package layout is modified, you need to verify that the connectivity of the physical implementation is valid. Markers are created to indicate opens and shorts present in the layout. Batch and interactive DRD checks can work with both curvilinear and regular shapes in a package layout. The DRD checker can be used to verify for the constraint violations in a package layout. Both connectivity extractor and DRD checker take into account void shapes.

Verifying that a design is correct is often the most difficult and yet also the most important aspect of designing a package layout. Additionally, routing paths should be adjusted iteratively until the package is optimized for all constraints. The number of layers depends on power levels and complexity.

***Related Topic***

Checking Layout

## Cross-Fabric Checks Run

The *Cross-Fabric Check* command lets you compare connectivity across fabrics and display results in a system check report. Instead of running connectivity checks for each die in a design, you can run the *Cross-Fabric Check* command at the board level. The command extracts connectivity information and runs the required connectivity checks hierarchically in the following sequence:

1. Connectivity information is extracted from the board layout and connectivity checks are run to ensure that the connections are valid.
2. Check Against Source (CAS) checks are run on the board layout.
3. Layout vs Abstract (LVA) checks are run on all package footprints.
4. Connectivity information is extracted for each package layout and connectivity checks are run to ensure that the connections are valid.


© 2023 Cadence Design Systems, Inc.

5. CAS checks are run on package layouts.
6. LVA checks are run on all die footprints.
7. Connectivity information is extracted for IC layouts and connectivity checks are run to ensure that the connections are valid.
8. CAS checks (only top-level) are run on IC layouts.

Violations are displayed in a system check report and in the Annotation Browser.

***Related Topic***

Performing Cross-Fabric Checks

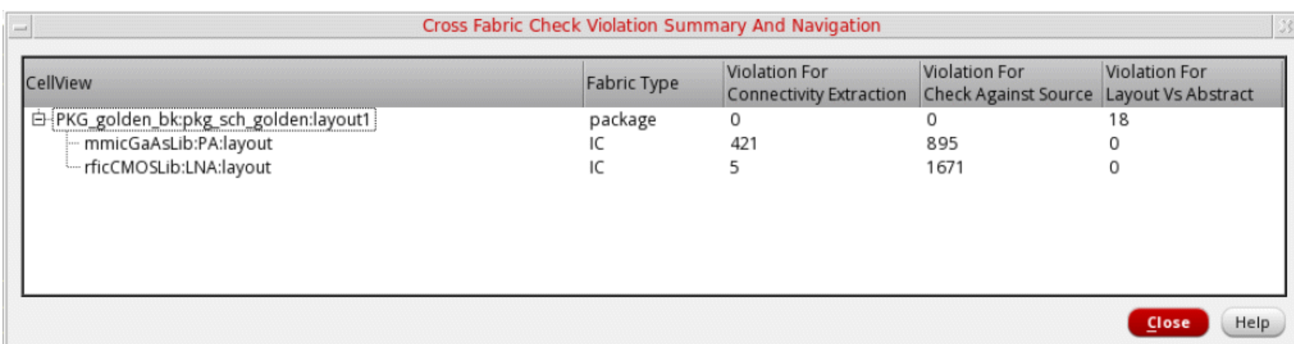
# Performing Cross-Fabric Checks

To run a cross-fabric check:

1. Choose *Module – Cross Fabric Check*.

   The design opens in Edit-in-Concert mode and cross-fabric checks are run.

2. Violations are reported in the Cross Fabric Check Violation Summary and Navigation form and on the *Connectivity* and *CAS* tabs of Annotation Browser.




3. Expand the package view in the *CellView* column to view a list of cellviews in which violations are reported. *Fabric Type* specifies the fabric type of the cellviews.

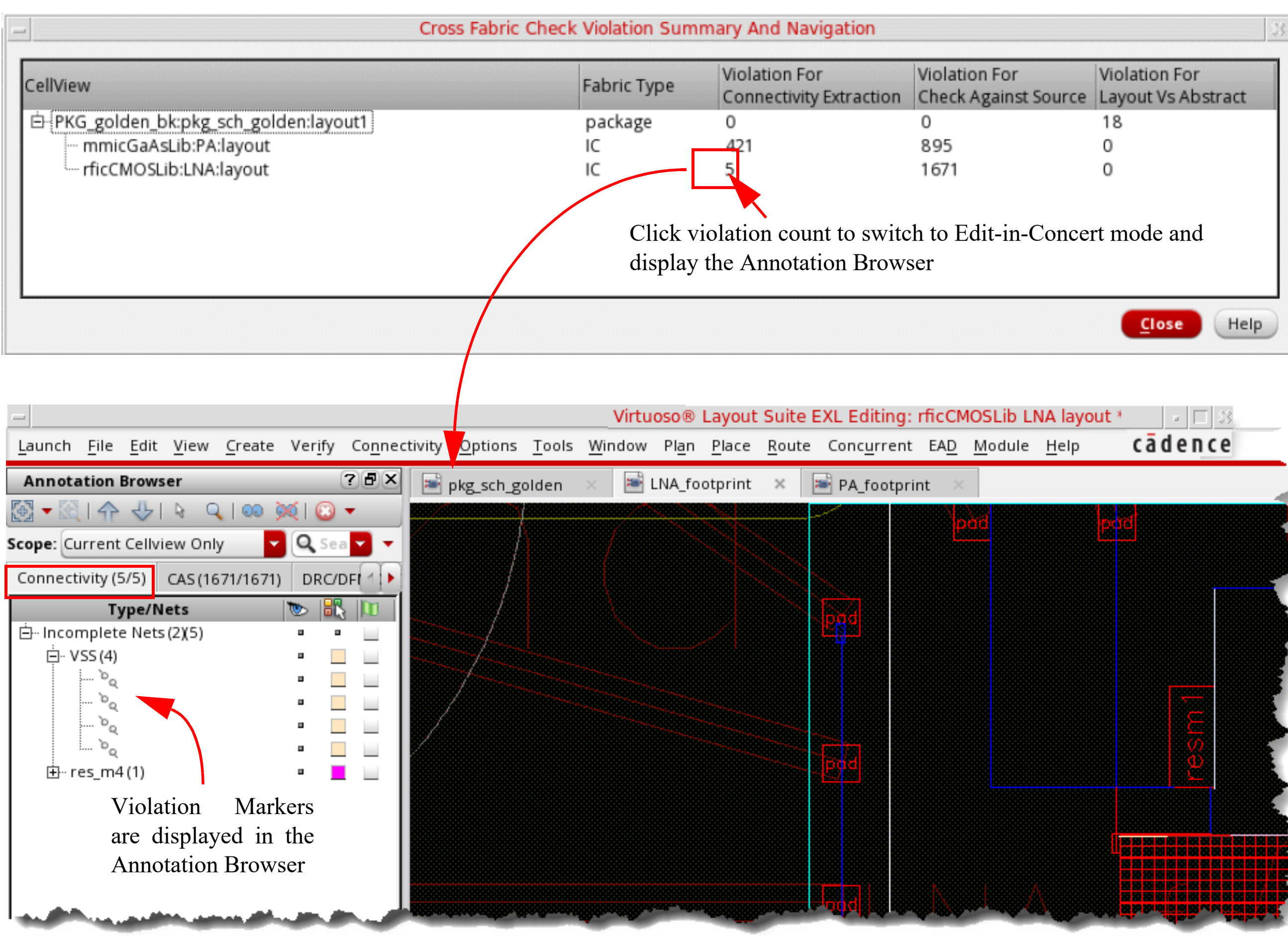
   Cross-fabric check violations are listed under the following columns:

   - *Violation for Connectivity Extraction* lists the violations reported during connectivity extraction.


© 2023 Cadence Design Systems, Inc.

- ❑ *Violation for Check Against Source* lists the number of violations reported by the CAS checker.
- ❑ *Violation for Layout Vs Abstract* lists the number of violations reported by the LVA checker.

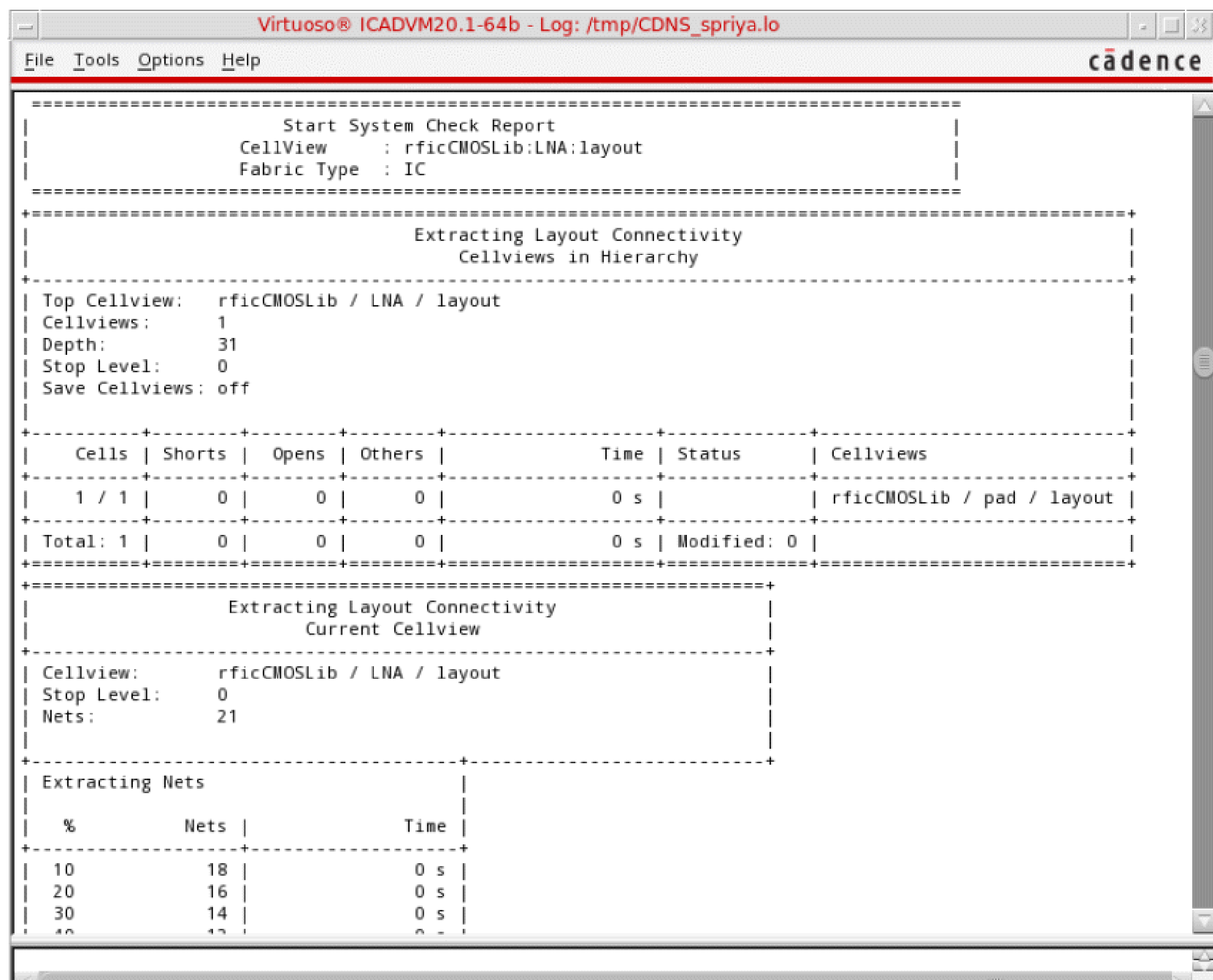
**4.** Click a violation count to open the corresponding die in the foreground in Edit-in-Concert mode. The Annotation Browser is displayed with the corresponding tab open. For example, if you click the *Violation for Connectivity Extraction* count, the *Connectivity* tab is displayed.





© 2023 Cadence Design Systems, Inc.

A detailed system check report is displayed in the CIW.




### *Related Topic*

Cross Fabric Check Violation Summary and Navigation Form

Cross-Fabric Checks Run

# Checking Layout Against Schematic

The development of any design involves an iterative process of synchronizing the differences between the schematic and layout. Changes, especially caused by Engineering Change Orders (ECOs), are made in the schematic and need to be updated in the layout. Similarly,


© 2023 Cadence Design Systems, Inc.

changes in the layout, during interoperability with Cadence SiP Layout, require updating the schematic.

***Related Topic***

Checking a Layout Against a Schematic

# Extracting the Connectivity

Connectivity extraction is needed to update the connectivity in the layout and check that the physical connectivity of a layout matches its logical connectivity. During extraction, markers are created to represent the shorts, opens, and invalid connections. When the layout is modified, interactive extraction is done automatically on modified objects and areas.

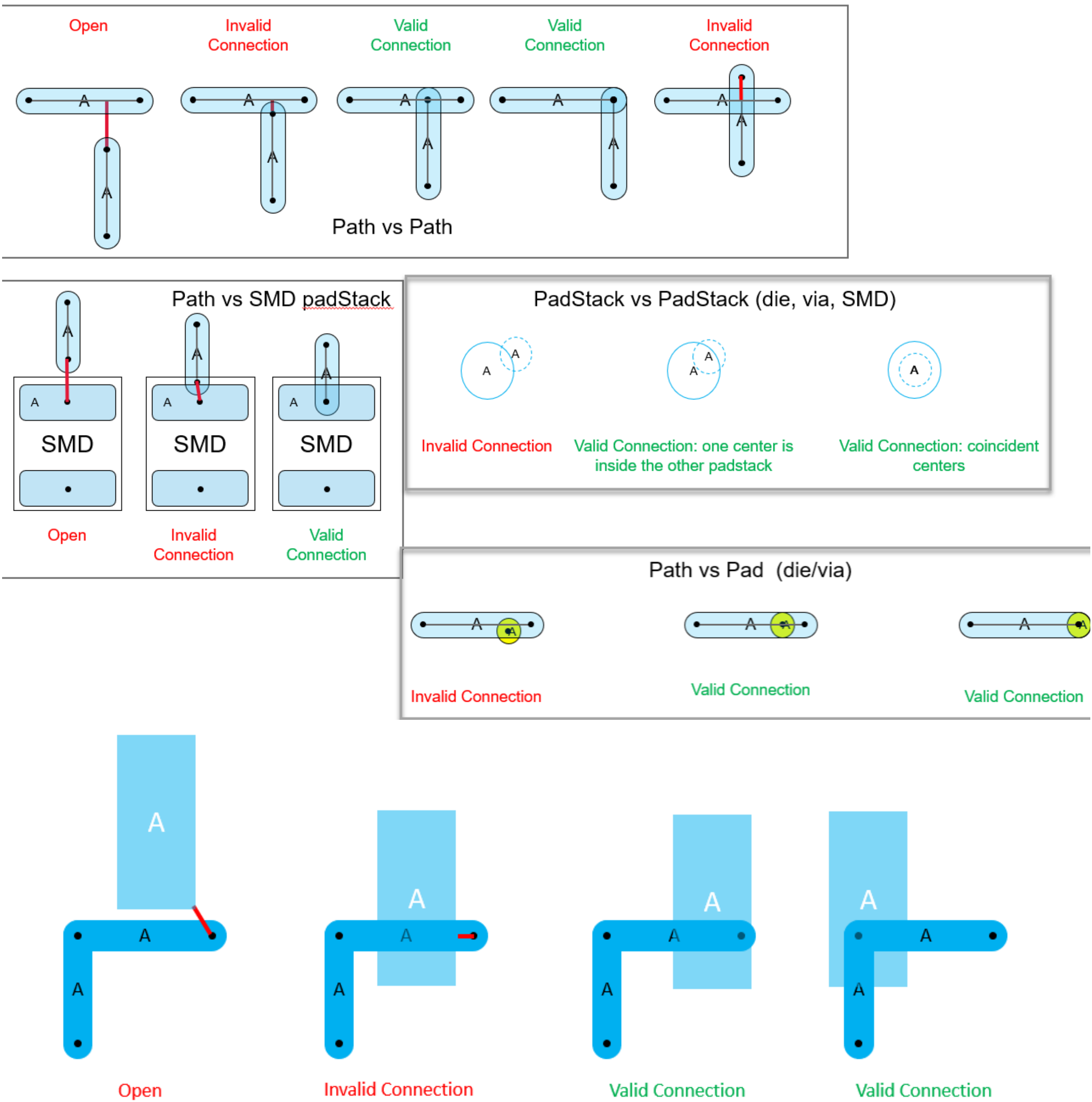
Cadence SiP layout requires a valid connections while importing package designs from Virtuoso. For example, a connection between two paths is valid if the end point of one path is overlapping the center line of the other path.

To extract connectivity:

1. Run the extractor explicitly using the *Connectivity – Extract Layout* command to update and check the connectivity for the entire package layout for invalid connections.
2. View and fix the violation markers in the *Connectivity* tab of the Annotation Browser assistant.


© 2023 Cadence Design Systems, Inc.

The following images show examples of open connections, invalid connections, and valid connections.





© 2023 Cadence Design Systems, Inc.

***Related Topic***

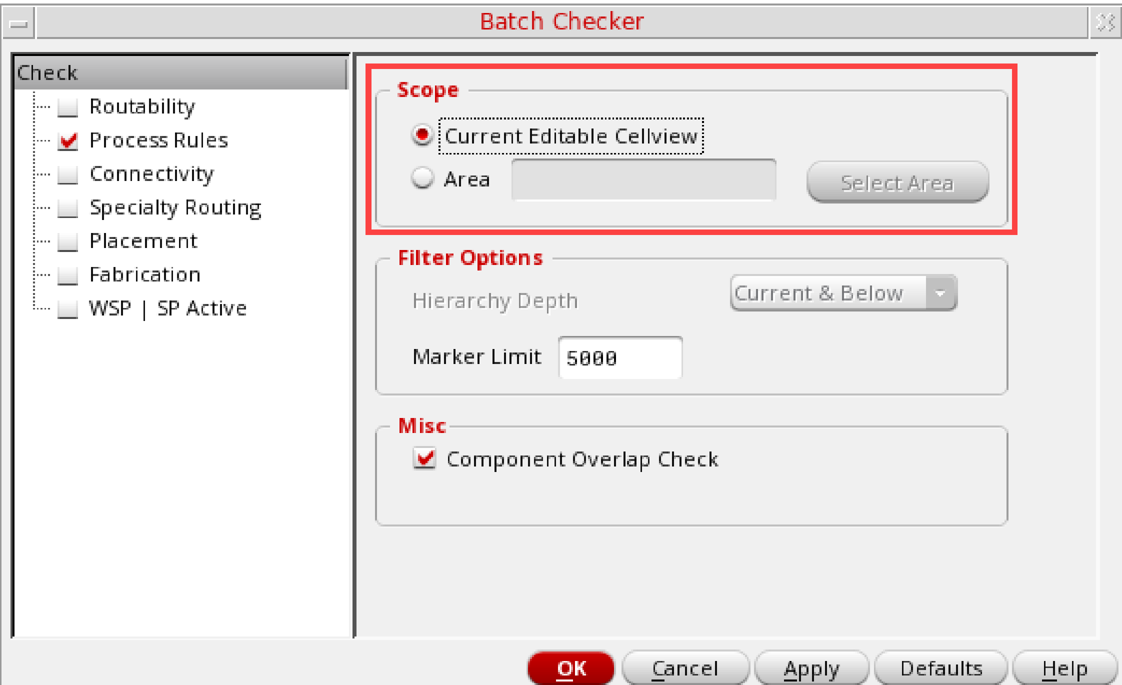
Connectivity Extraction

# Performing DRD Checks

After a layout is imported from Allegro, DRD checks the layout against design rules defined as constraints. DRD performs the checks in Post-Edit and Batch-Check modes.

To perform DRD checks, do the following:

1. Click the *Post-Edit* icon on the *Options* toolbar to turn on the Post-Edit mode.
2. Choose *Verify – Design*.
3. In the *Scope* section of the Batch Checker form, define the scope of checking in batch mode.




4. In the *Marker Limit* field, specify the maximum number of post-edit markers to be created per call for the Post-Edit mode.
5. Click *OK*.


© 2023 Cadence Design Systems, Inc.

Alternatively, you can use the *DRD* toolbar to set up the various batch mode options as follows:

1. Click the *Verify Design* button .
2. Select an option from the drop-down list.

   *Current Cellview* (default) checks all shapes in the current editable cellview.

   *Select Area* checks all shapes in the specified area of the design.

   In this case, specify the lower-left and upper-right coordinates for the region in this format: `((xlower ylower) (xupper yupper))`, for example, `((5.2 3.3) (5.7 3.4))`, in the text box provided on the *DRD* toolbar. Alternatively, you can select the area on the canvas.

# Supported DRD Constraints and Checks

DRD supports the following constraints for verifying a layout.

- `pkgMaxBondwireLength`
- `pkgMinBondwireLength`
- `pkgMinBondwireSpacing`
- `pkgMinBondwireWidth`
- `pkgMinHoleSpacing`
- `pkgMaxLineWidth`
- `pkgMinLineWidth`
- `pkgMinSpacing`

**Note:** The DRD checker ignores constraints with negative spacing values.

The following DRD checks are performed for verifying a layout.

### Overlap Checking

DRD checks if the PR boundaries of two SMD or die instances placed next to each other overlap. If the Pcell parameter of the instances, `flipOverY`, is set to `nil`, they are placed on the top of the board. If the parameter is set to `t`, the instances are placed on the bottom of the board. In either case, the PR boundaries of the instances must be non-overlapping.


© 2023 Cadence Design Systems, Inc.

To perform overlap checking, select the *Component Overlap Check* check box on the *Process Rules* page of the Batch Checker form. Alternatively, set the `drdEditApkDrcComponentOverlap` environment variable to `t` in your `.cdsinit` or `.cdsenv` file.

### Supported Net Overrides

DRD supports the following net overrides for verifying a layout:

- Net: Set of constraints applied to a net
- Net Group: Set of constraints applied to a group of nets
- Net Class: Set of constraints applied to a mix of nets and net groups
- Net Class-Class: Set of constraints applied between two net classes

For more information on net overrides, see *Allegro® Platform Constraints Reference* and *Allegro® Constraint Manager User Guide*.

### DRD Region Overrides

DRD supports region overrides for verifying a layout. A layout can have multiple regions, with each region having a separate set of region-specific constraints. All shapes in a region are checked against their own region-specific constraints.

Region-specific constraints override net, design, and foundry constraints. When regions are hierarchical, the constraints of an inner region override the constraints of outer regions.

For more information on net overrides, see *Allegro® Platform Constraints Reference* and *Allegro® Constraint Manager User Guide*.

### Reporting minSpacing and Short Violations

DRD reports `minSpacing` and short violations between shapes on top-level nets and shapes on local nets, which are shapes in the hierarchy with no top-level net. However, DRD does not report such violations for a shape on the top-level net in the `minSpacing` halo around the same top-level pin.

For example, consider an inductor that has pins on nets A and B and its coil shape over net C. DRD reports short violations between the inductor coil shape and all nets at the top level, except when a shape on the top level of net B is within the `minSpacing` value for pin B. This


© 2023 Cadence Design Systems, Inc.

allows the connection to pin B without DRC violations. DRD reports shorts violations for shapes on net A that are near pin B and for shapes on net B that are near pin A.


© 2023 Cadence Design Systems, Inc.

# 8

# Edit-in-Concert

The Edit-in-Concert feature lets you view and edit die packages and their corresponding die layouts synchronously. It enables the IC designers and package designers, who are cross-geographically located and work on different platforms, update the designs (die or package) using the abstract files that do not require synchronous or concurrent operation of the tools. For example, when a layout engineer creates a die layout and a package engineer creates the corresponding die package.

- In die layouts, device placement is driven by factors such as the internal blocks of the design, routing of the layout, and the power requirements of the layout.
- In die packages, device placement is determined by factors such as the final alignment with the package BGA, the minimum bond wire lengths, and the usage of passive elements in the package.

You can use Edit-in-Concert to view and edit the board, package, and layout synchronously. Edit-in-Concert lets you:

- Visualize the IC in the context of the package and see where a change to one die impacts the design of the other.
- Ensure that the combination of die layout and the die package meets all design requirements, while minimizing the overall cost of production.

In Edit-in-Concert mode, the dies in a package are superimposed. The current die is displayed in the foreground, available for edit. The dies in the background serve as a template to guide the design.

Video

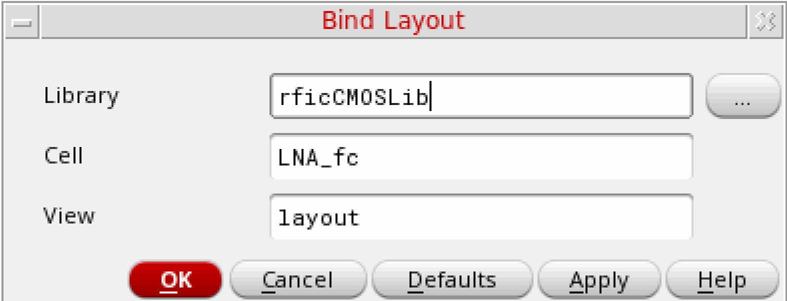
To know more about this feature, watch the Virtuoso RF Solution: Using the Edit-in-Concert Mode to Edit Die Packages and Layouts video available on Cadence Support portal.


© 2023 Cadence Design Systems, Inc.

# Update Binding Information

When a package layout is launched in Layout MXL, its persistent bindings are read and stored in memory. To view these bindings, choose *Module – Bind Layout*. The Bind Layout form is displayed, which provides information about the *Library*, *Cell*, and *View* of the layout to which the current package die is bound. The default values are the properties set by die export on the corresponding TILP. The Bind Layout form lets you bind die footprint instances to die layouts and package footprint instances to package layouts.

To maintain the binding between the schematic die symbol and the die layout in package, if there is any mismatch in the binding between the schematic and layout instances, you should update the physical binding.




In certain situations, for example, when dies of different technologies are brought together using TILPs, the binding information might be lost during the process. In such situations, use the Bind Layout form to update the binding information.


© 2023 Cadence Design Systems, Inc.

# Viewing Die Instance Annotations

In very complex package designs, it might be difficult to visually distinguish the die instances from other die components. The *Module* menu includes the *Die Instance Annotations* command, which lets you highlight the die instances and display annotations that provide basic information about the die instances.

Die instance annotations display the following information:

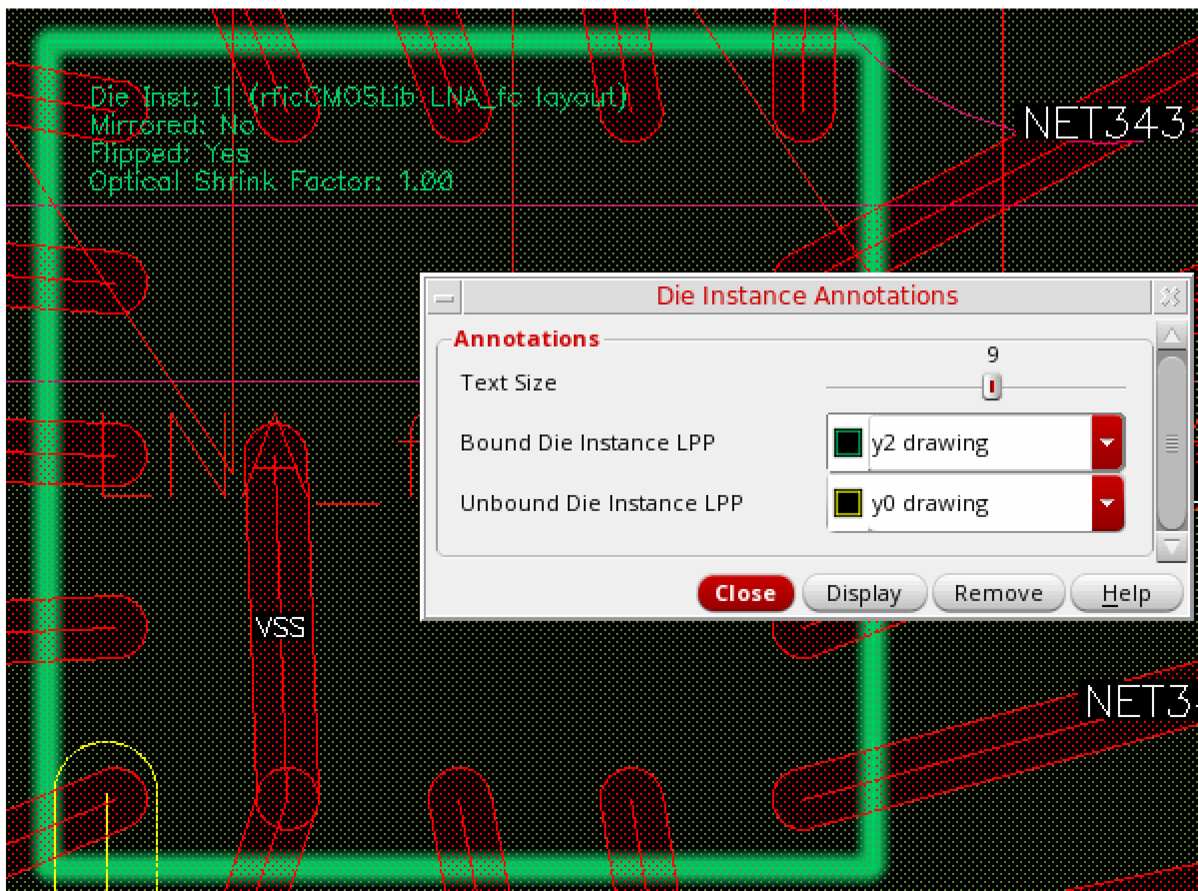
- Name of the die instance. The instance to which it is bound is displayed within brackets.
- Mirroring state of the die instance.
- Flipped state of the die instance.
- Optical shrink factor of the die instance.

**Note:** Use the Edit Instance Properties form to change the above values.

To view die instance annotations:


© 2023 Cadence Design Systems, Inc.

1. Choose *Module – Die Instance Annotations*. The Die Instance Annotations form appears. Use the options in this form to customize the way in which die instance annotations are displayed.




2. Drag the *Text Size* slider to adjust the size of annotation text.
3. Use the *Bound Die Instance LPP* and *Unbound Die Instance LPP* options to assign different highlight colors to bound and unbound die instance LPPs to facilitate easy visual distinction in complex designs.
4. Click *Display* to view the annotations on the canvas.

**Note:** Click *Remove* to hide the die instance annotations.


© 2023 Cadence Design Systems, Inc.

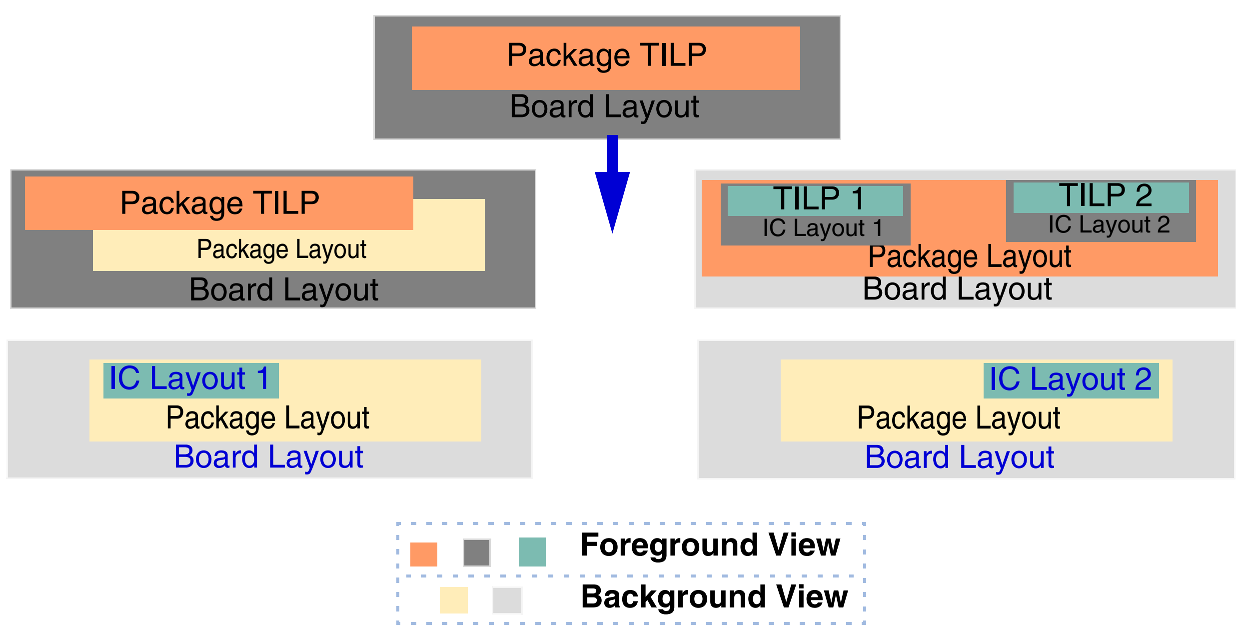
# Key Edit-in-Concert Views

The following table shows the key views in Edit-in-Concert mode and their descriptions.

| Views | Description |
|---|---|
| `Die layout` | Layout view of an unpackaged chip |
| `Die symbol` | Symbol view of an unpackaged chip |
| `Die schematic` | Schematic view of an unpackaged chip |
| `Package schematic` | Schematic view of the package<br>A package contains the padstacks, labels, outline, TILPs, and so on. It visually represents the component in Layout MXL. Note that a single package may consist of a symbol or multiple logical symbols. |
| `Package layout` | Layout view of the non-IC fabric |
| `Die abstract` | One of the results of exporting the die along with a TILP, schematic, and symbol view for instantiation in a package. |
| `TILP` | Technology Independent Layout Pcells (TILPs) are created from the die symbols, which have been created from exporting the die. TILPs are added to the libraries from importing into the package layout. |
| `Die abstract symbol` | Symbol view of the die abstract created while exporting the die |
| `Die abstract Schematic` | Schematic view of the die abstract created while exporting the die |

The *Edit-in-Concert* command processes the source layout for more than one hierarchal level. In the following example, the launch pad for the command is a board layout that includes a package TILP. The *Edit-in-Concert* command processes the corresponding package layout and opens separate tabs for all the constituent IC layouts, with the package and board layouts in the background view.


© 2023 Cadence Design Systems, Inc.




| Launch Pad | | Package Tab | | Package Layout Tab | | IC Layout Tab | | IC Layout Tab | |
|---|---|---|---|---|---|---|---|---|---|
| **BG View** | **FG View** | **BG View** | **FG View** | **BG View** | **FG View** | **BG View** | **FG View** | **BG View** | **FG View** |
| | Board Layout | | Board Layout | Board Layout | Package Layout | Board Layout | | Board Layout | |
| | | Package Layout | | IC Layout1 | TILP1 | | IC Layout1 | | IC Layout2 |
| | Package TILP | | Package TILP | IC Layout2 | TILP2 | Package Layout | | Package Layout | |

***Related Topic***

Packaging-Related Terminology

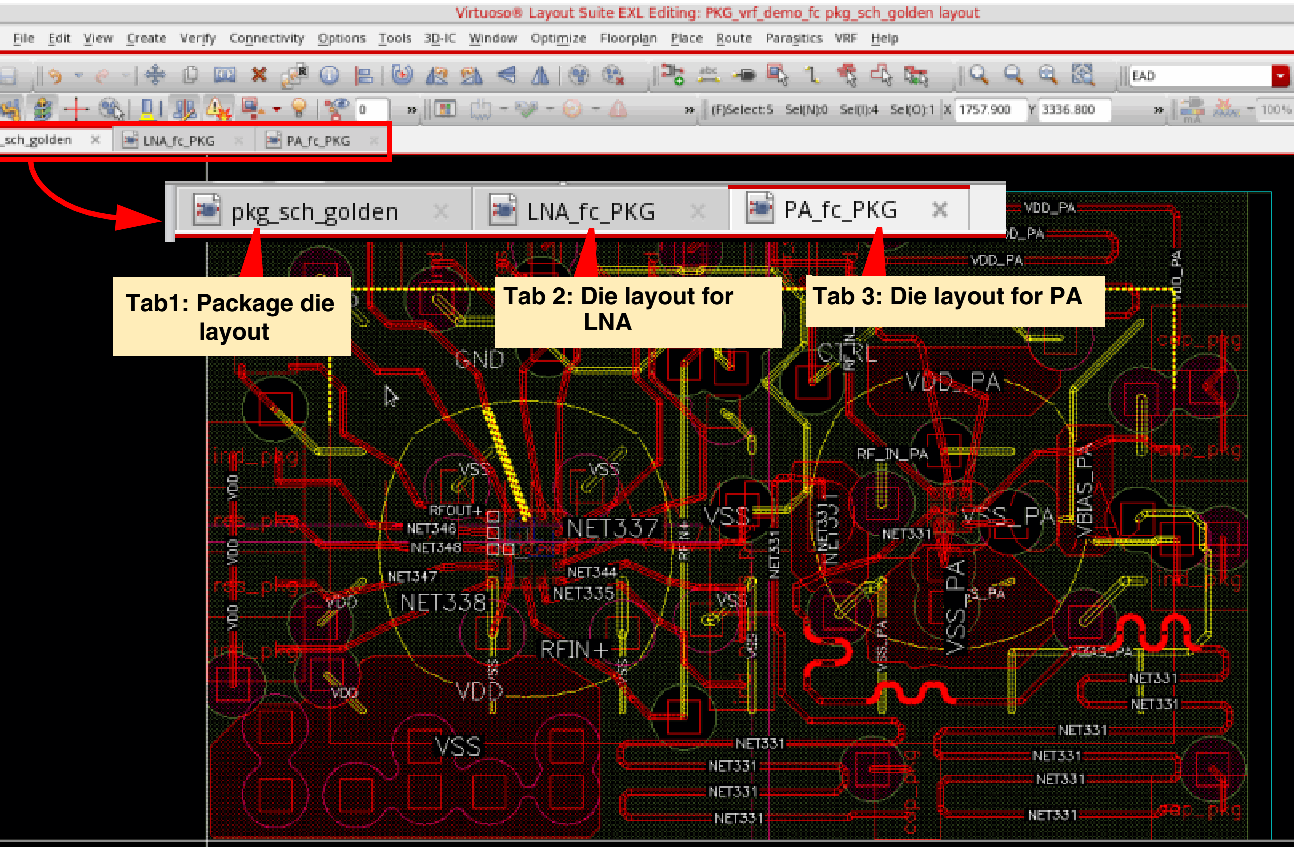
Die/TILP Instantiation


© 2023 Cadence Design Systems, Inc.

# Launch Edit-in-Concert Mode

Edit-in-Concert mode can be launched from package, module, and board layouts. To switch to Edit-in-Concert mode, with a package or board die open, choose *Module – Edit-in-Concert*. The package and die layouts associated with the die footprint instances in the board or package die are displayed on subsequent tabs.

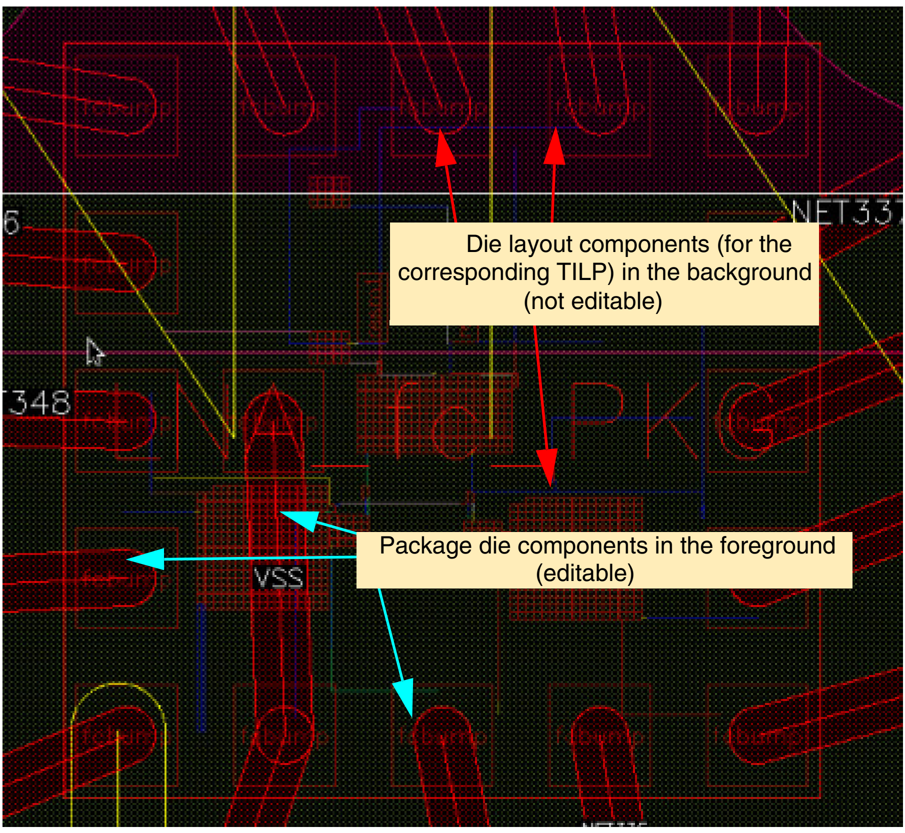
In the following example, the given package die includes two TILPs, LNA and PA. In Edit-in-Concert mode, the package die layout is displayed on the first tab and the die layouts for the two TILPs are displayed on separate tabs.





© 2023 Cadence Design Systems, Inc.

Here is zoomed-in image of the above design, which shows the package die in the foreground (editable) and the die layout corresponding to the TILP in the background (not editable). The zoom settings are propagated across the dies open in Edit-in-Concert mode.




In Edit-in-Concert mode, the display levels of the background dies on the package tab is determined by the display levels of the die layout cellviews on their respective tabs.

Similarly, on each die tab, the package cellview in the background has the same view level as set in the package cellview window. You can control different view levels for different dies in the background in the package tab. Also, you can view the complete hierarchy of the package cellview without opening the complete hierarchy of the die layout cellviews in Edit-in-Concert mode.

**Note:** On the die layout tab, the die layout components are displayed in the foreground and the package die components are displayed in the background.

**Note:** Module fabrics are treated the same way as package fabrics. The commands in the *Module* menu are available to cellviews with package and module fabrics.


© 2023 Cadence Design Systems, Inc.

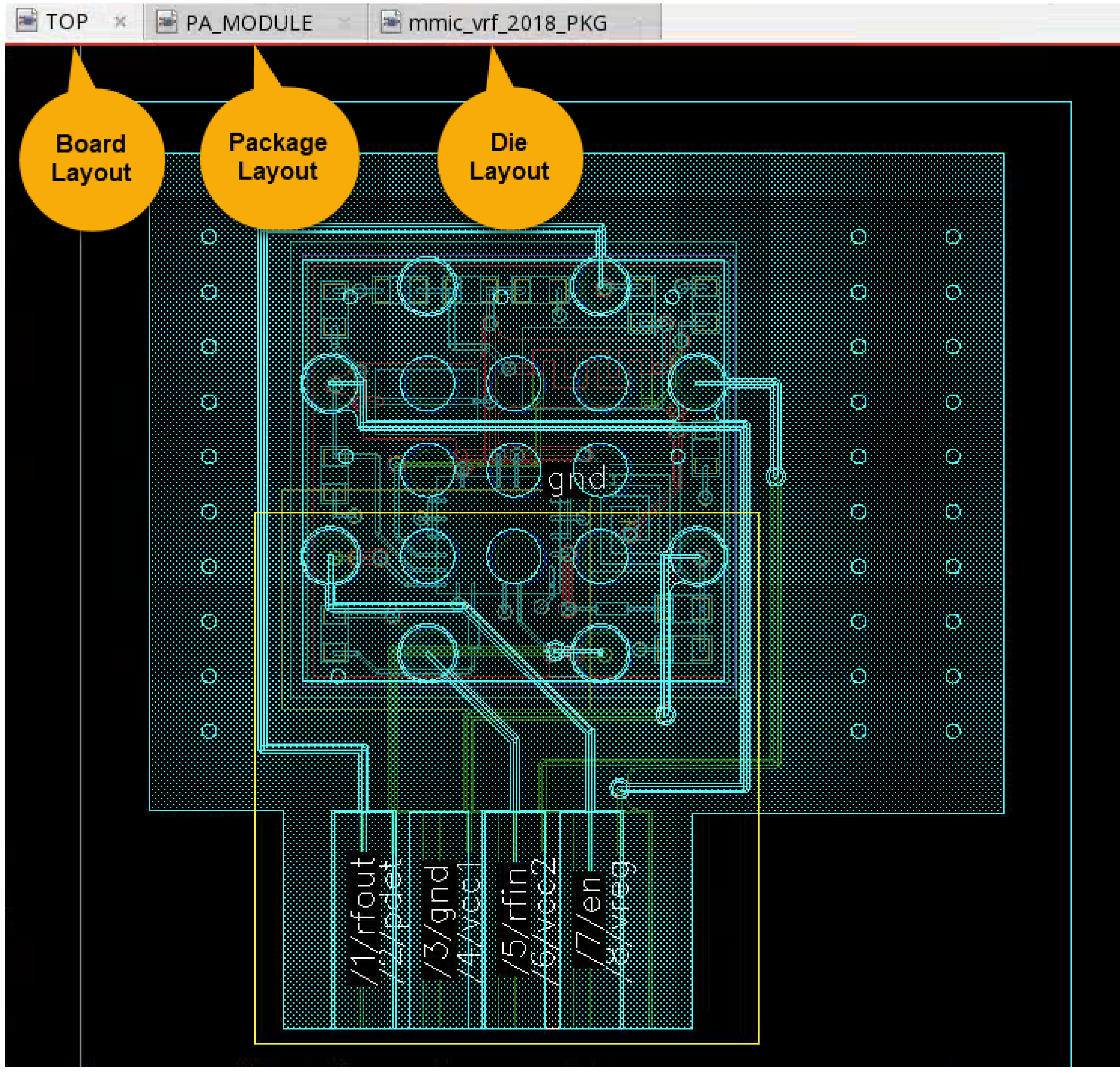
### Launch Edit-in-Concert from Board Layout

To open Edit-in-Concert mode from a board layout, with the board layout open, choose *Module – Edit-in-Concert.*

- The board design is displayed on the first tab and it contains the package TILP.
- The second tab contains the package layout corresponding to the TILP in the board. This tab also contains the TILPS corresponding to die layouts.


© 2023 Cadence Design Systems, Inc.

- The third tab contains the IC die layout corresponding to the TILP in package layout, which is displayed on the second tab.




**Note:** You can use the Palette assistant to control the visibility of layers across all dies that are open in Edit-in-Concert mode. For example, if you hide the visibility of layers A and B in the foreground view, these layers are automatically hidden in the background cellviews. This feature is useful for designs with a large number of shapes in multiple layers, where understanding the layers and shapes in the background cellviews is difficult. Enabling selective visibility of layers in the background cellview is useful in such scenarios.


© 2023 Cadence Design Systems, Inc.

Edit-in-Concert mode supports packages with multiple instances of the same die that are bound to the same layout and die footprint. The die selected before launching Edit-in-Concert is considered the active die and is displayed on a separate tab in Edit-in-Concert mode. The changes made to this die are automatically propagated to other instances of the die. The updates made by the LVA fixer are also propagated to the remaining dies.

In addition to single packages, Edit-in-Concert supports package-on-package designs, where there can be more than one package TILP in a module or board layout.

Edit-in-Concert mode does not support die instances that are bound to the same layout but have different die footprints.

#### *Related Topics*

Edit-in-Concert

Key Edit-in-Concert Views

Launch Edit-in-Concert Mode


© 2023 Cadence Design Systems, Inc.

# Modify in Edit-in-Concert Mode

Edit-in-Concert supports dynamic modification of location and contents of the board, package, and IC dies. Therefore, these dies are always synchronized. Any modification you make to a die is auto-updated on the other tabs in the background. These modifications are immediately visible on all the relevant Edit-in-Concert tabs.

For example, when a bump cell containing bumps has been instantiated multiple times in a layout, moving a bump in Edit-in-Concert mode either at the top-level or level-1 results in the movement of the bump at multiple locations. The bump is moved at all locations in which the master bump cell has been instantiated.

The Virtuoso RF solution supports the following tasks in the Edit-in-Concert (in-sync) mode:

- Zooming into dies
- Move dies
- Move die instances
- Run cross-fabric check
- Run cross-fabric net probe
- Change TILP parameters
- Run the Layout vs Abstract (LVS) checker and fixer
- Run Net Tracer
- Probe nets, bump instances, and Ball Grid Array (BGA) balls

***Related Topics***

Movement of Dies in Edit-in-Concert Mode

Movement of Die Instances in Edit-in-Concert Mode

Changes to TILP Parameters in Edit-in-Concert mode

Change from Package view to Layout View

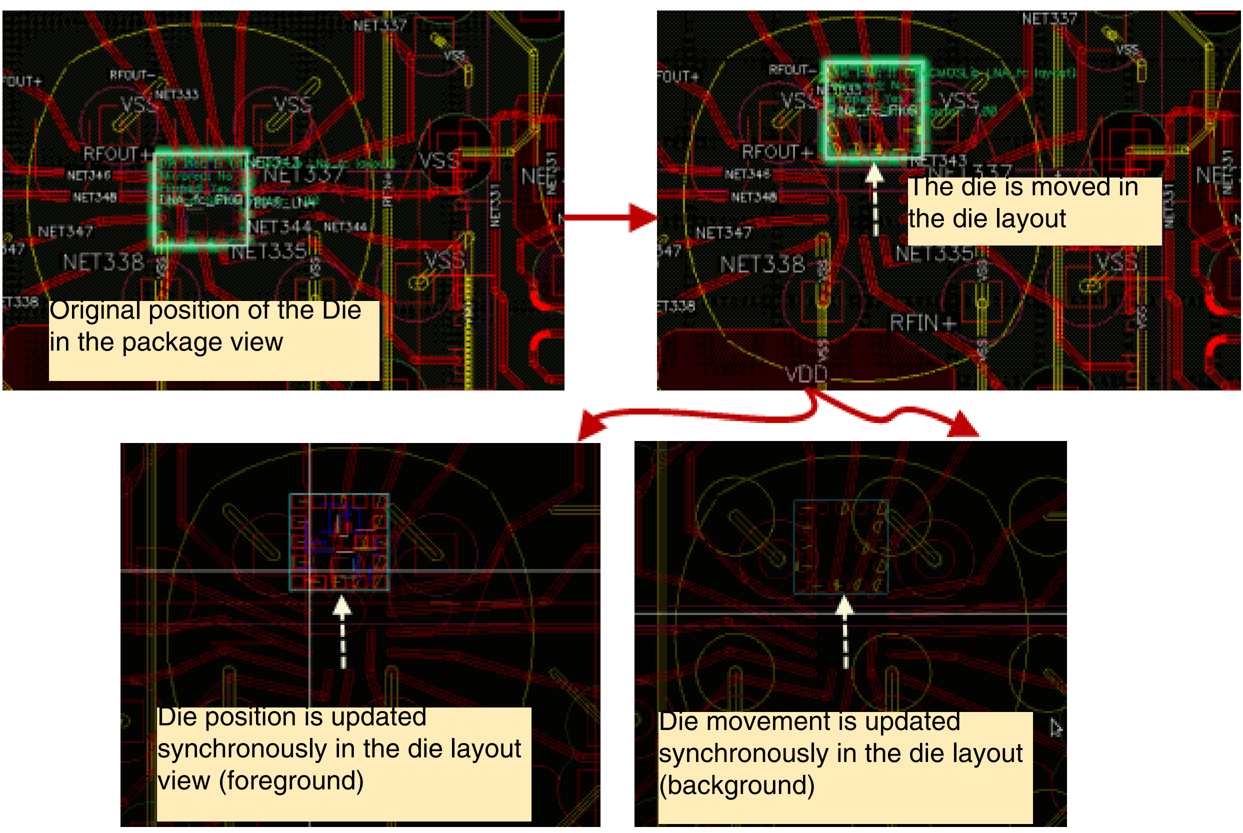
Net Tracing in Editing-In-Concert Mode

Probing a Design in Edit-in-Concert Mode


© 2023 Cadence Design Systems, Inc.

## Movement of Dies in Edit-in-Concert Mode

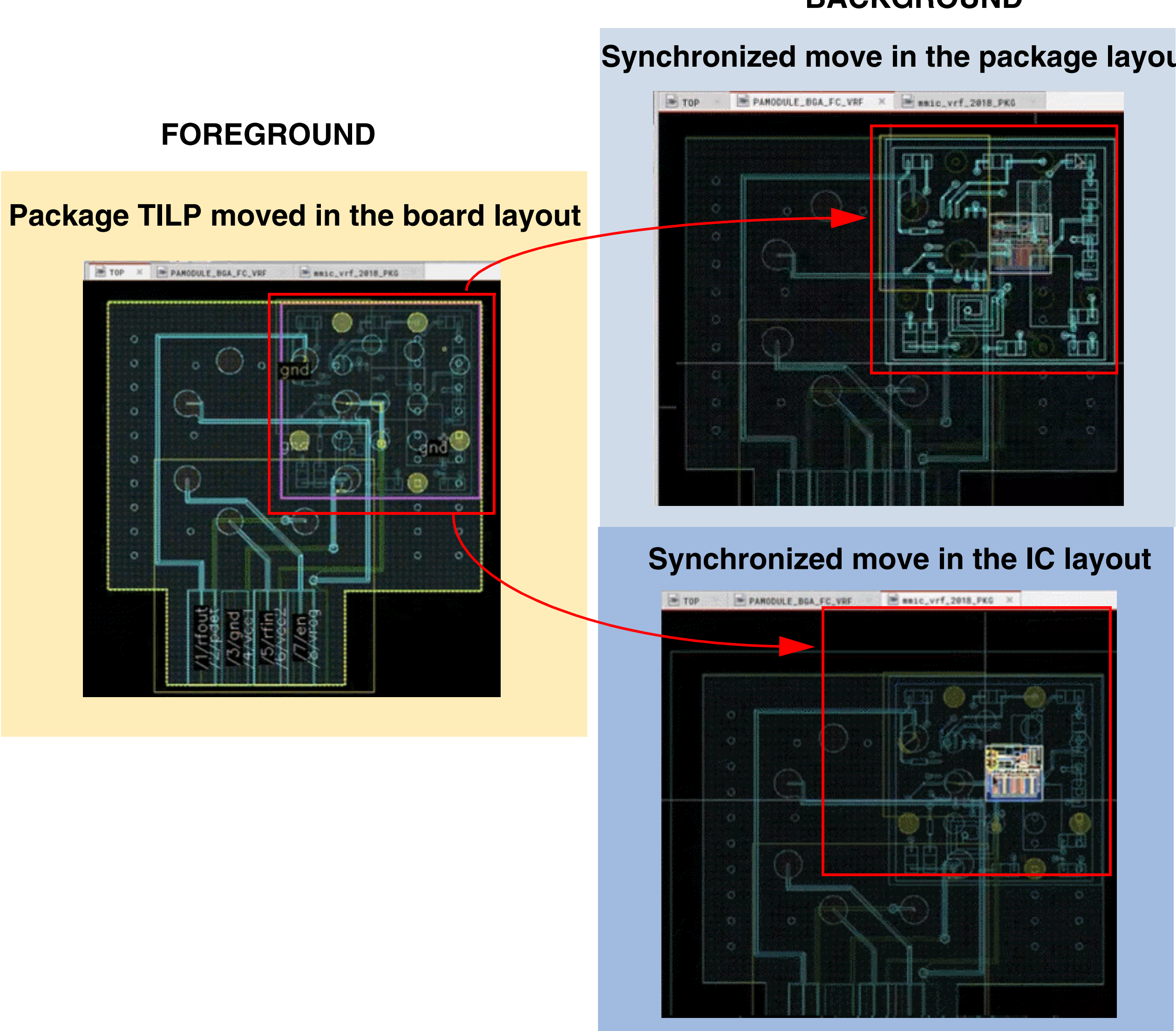
When you move a die in the package view, the die layout in the background is moved synchronously on the other tabs.




Similar synchronized movement is observed of package TILPs. When you move a package TILP, the position of the TILP is automatically updated in the background view of all the other tabs that are open in Edit-in-Concert mode.


© 2023 Cadence Design Systems, Inc.

In the following example, a package TILP is moved in the board die. Synchronous movement of the die is observed on the other tabs.




***Related Topics***

Modify in Edit-in-Concert Mode

Movement of Die Instances in Edit-in-Concert Mode

Changes to TILP Parameters in Edit-in-Concert mode


© 2023 Cadence Design Systems, Inc.

Change from Package view to Layout View

Net Tracing in Editing-In-Concert Mode

Probing a Design in Edit-in-Concert Mode

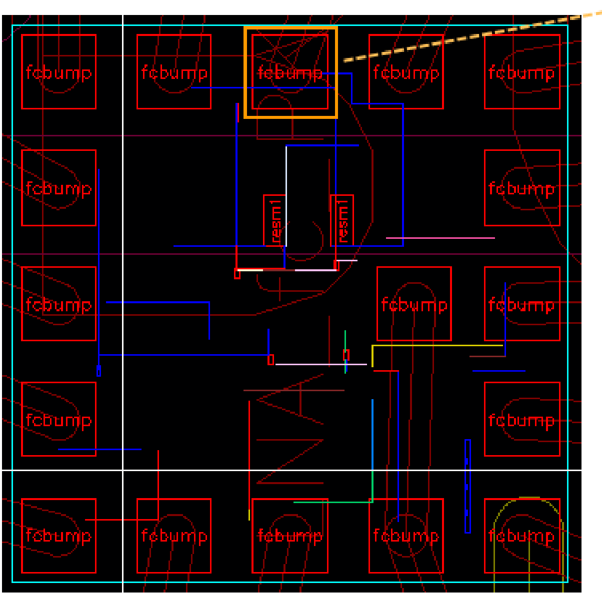

© 2023 Cadence Design Systems, Inc.

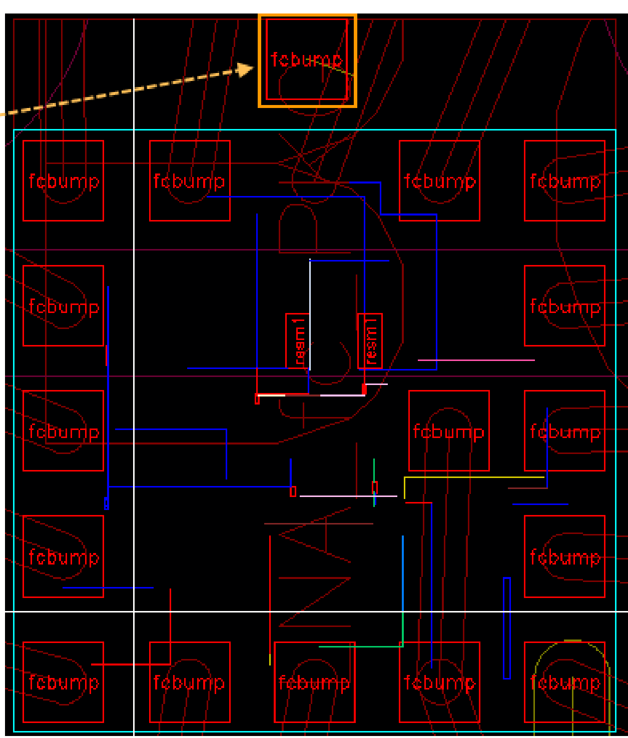
## Movement of Die Instances in Edit-in-Concert Mode

Edit-in-Concert supports the following in-sync die movements:

- IO pad movement between board, package, and IC fabrics. The following images depict the synchronous movement of an IO pad in the die layout view and its corresponding package layout view Edit-in-Concert tabs:

**TILP View - An I/O pad is moved from its original position**

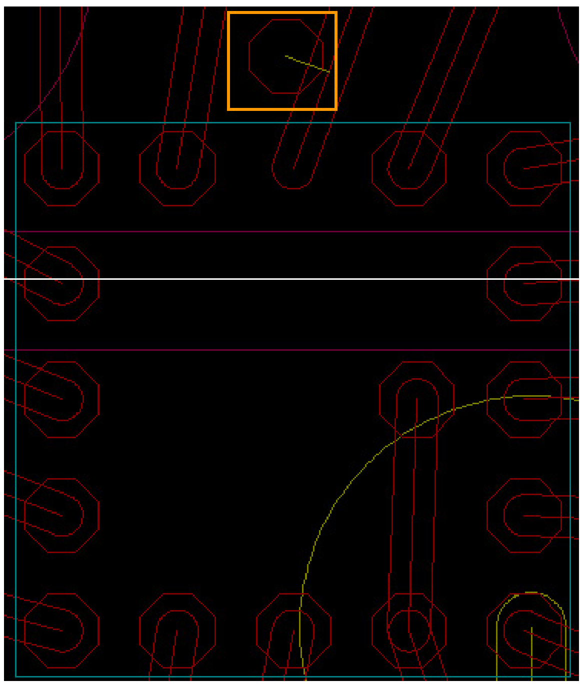


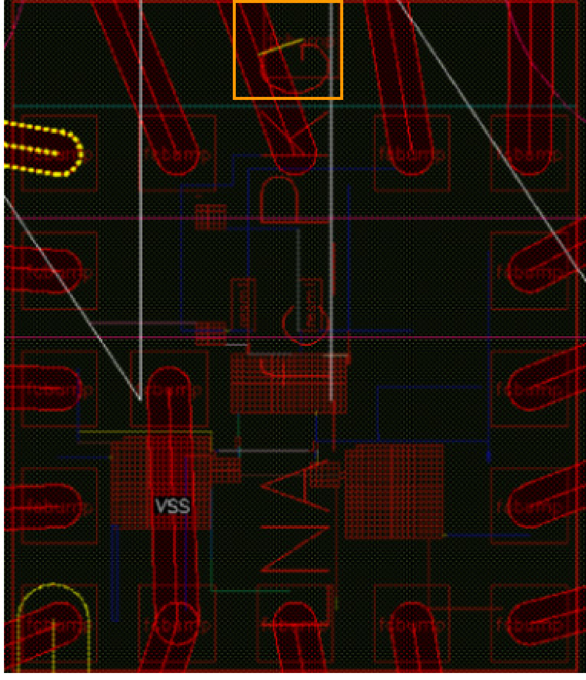





© 2023 Cadence Design Systems, Inc.

**Die Layout View and Package Layout View - Position of the I/O pad is dynamically updated**






*Important*

Use the *Edit - Hierarchy - Edit In Place* command to edit placed instances that are located at lower levels. Using the *Edit - Hierarchy - Descend* command is not recommended.

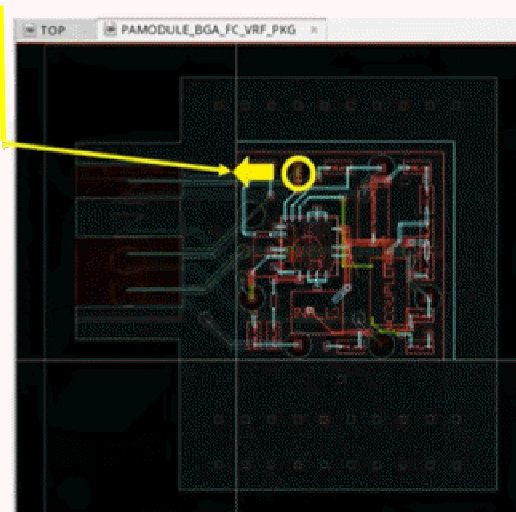
■ Ball grid array (BGA) and land grid array (LGA) IO movement between package footprint and package layout.

In the following example, a board design containing a package TILP instance and its corresponding package layout are open in Edit-in-Concert mode. An IO is moved in the


© 2023 Cadence Design Systems, Inc.

package TILP instance. The corresponding IO in the package layout BGA instance is also moved.

**In-Sync IO movement**




Board with Package Layout in the Background

Package with Board Layout in the Background

***Related Topics***

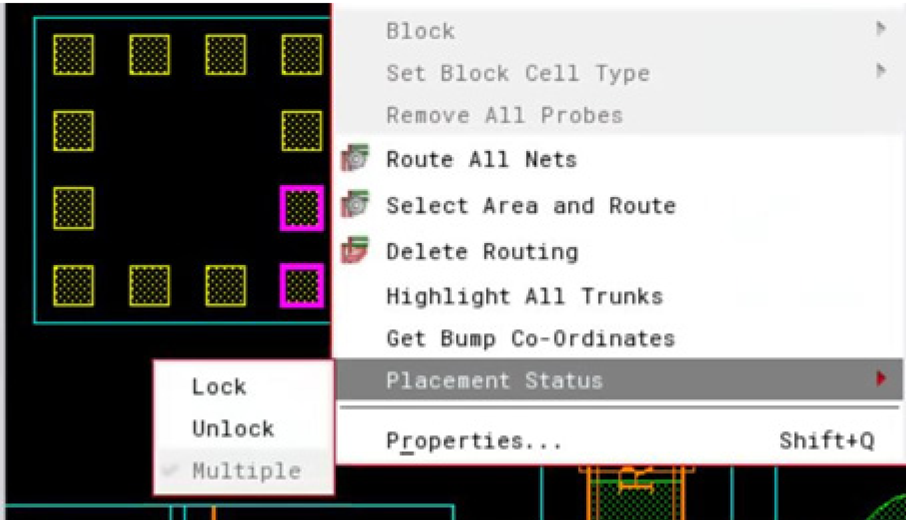
- Modify in Edit-in-Concert Mode
- Movement of Dies in Edit-in-Concert Mode
- Changes to TILP Parameters in Edit-in-Concert mode
- Change from Package view to Layout View
- Net Tracing in Editing-In-Concert Mode
- Probing a Design in Edit-in-Concert Mode

## Placement Status of IOs

You can view or set the placement status of IOs as *Lock* or *Unlock* using the *Placement Status* menu item. The menu item appears in the context menu when you select an IO or an instance. If the selected IO is locked, the *Lock* option appears selected in the sub-menu. Similarly, the *Unlock* option appears selected if the selected IO is unlocked. If both locked


© 2023 Cadence Design Systems, Inc.

and unlocked IOs are selected in a layout when selecting an instance, the *Multiple* option appears selected but this is a view-only option. You cannot set this option for IOs.

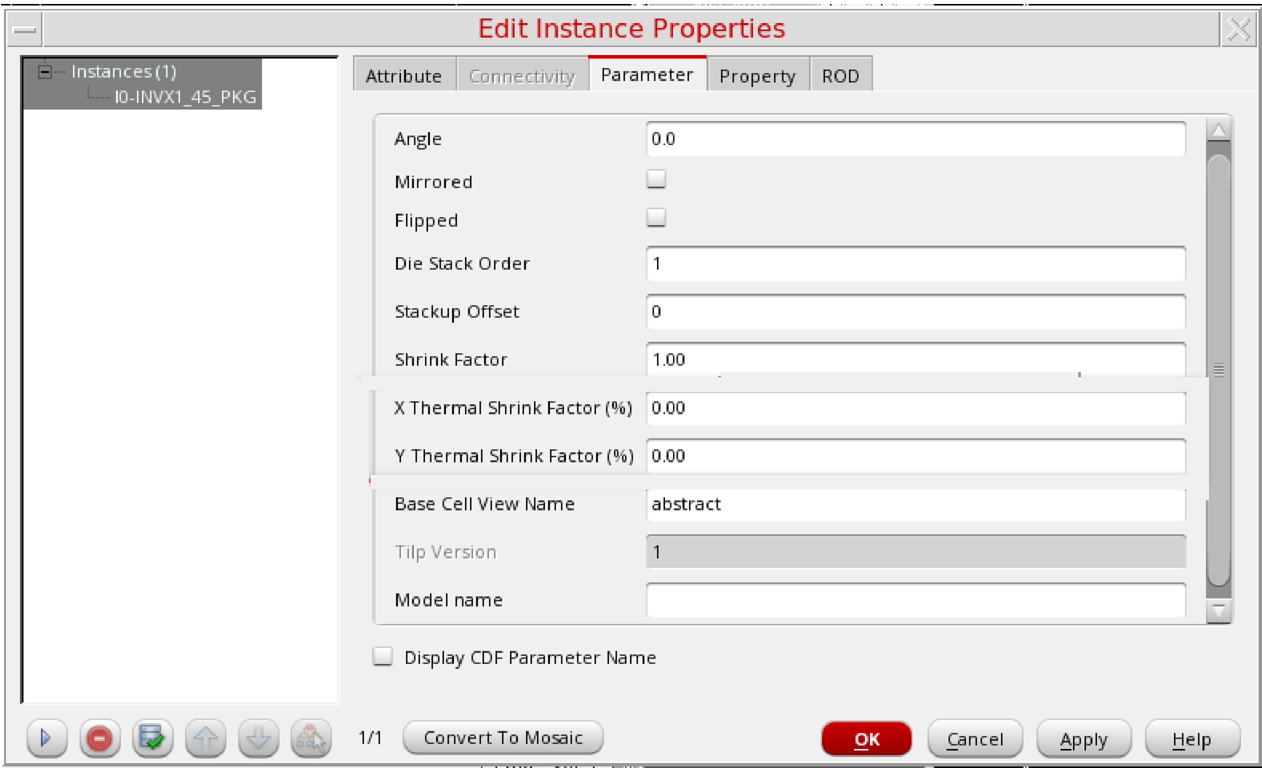




© 2023 Cadence Design Systems, Inc.

## Changes to TILP Parameters in Edit-in-Concert mode

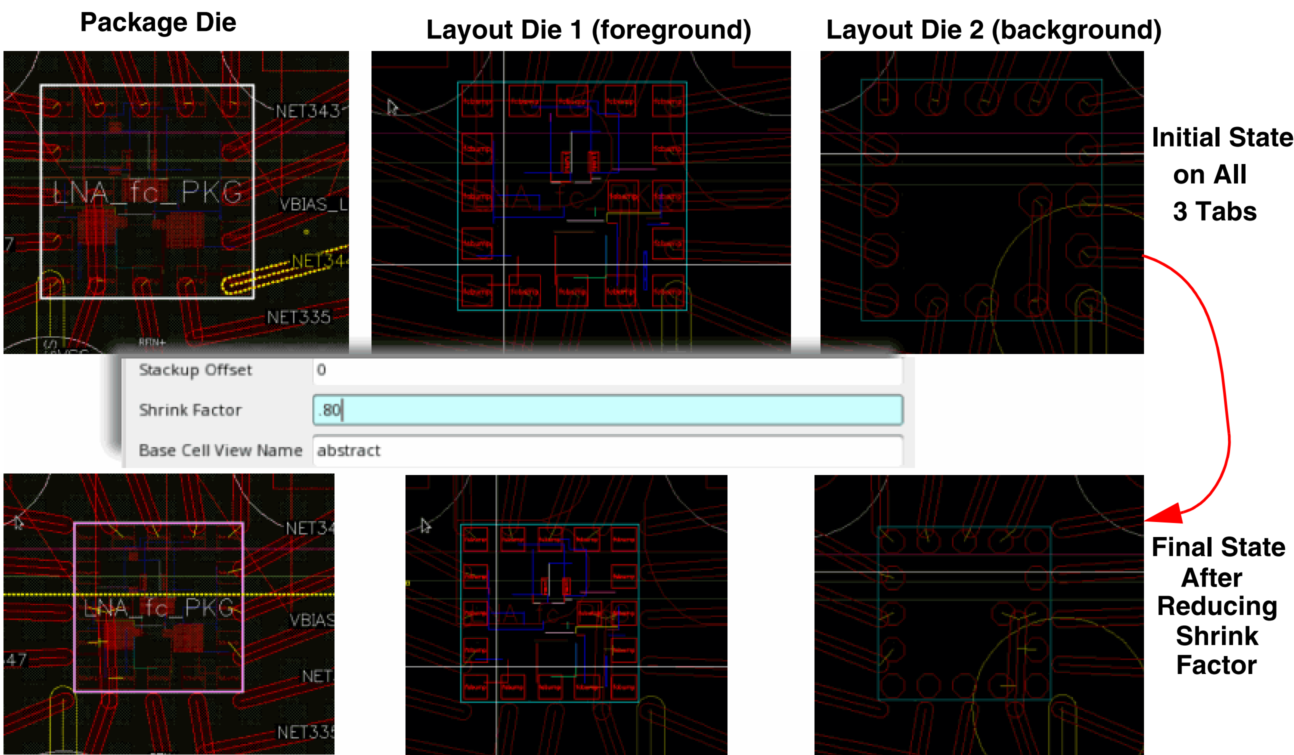
In Edit-in-Concert mode, any changes you make to the *Mirrored*, *Flipped*, and *Shrink Factor* TILP parameters are propagated automatically to the other tabs.





© 2023 Cadence Design Systems, Inc.

In the following example, in Edit-in-Concert mode, the *Shrink Factor* of a TILP is reduced in the package die. The change is propagated to all other dies that are open in Edit-in-Concert.



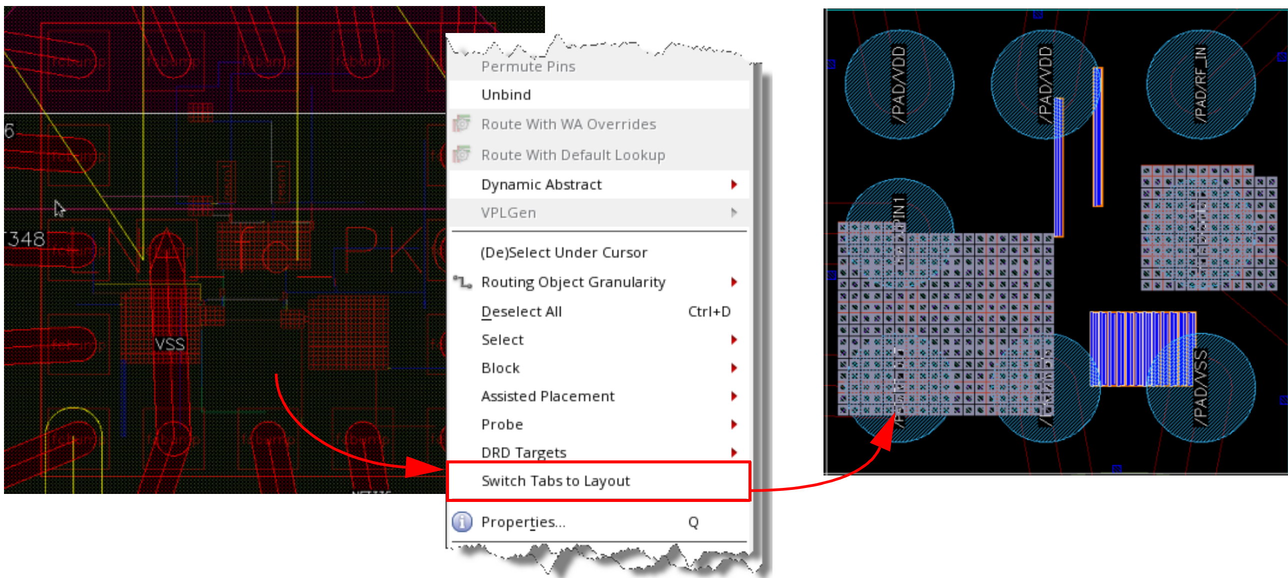

### *Related Topics*

- Editing TILP parameters
- Edit-in-Concert
- Modify in Edit-in-Concert Mode
- Movement of Dies in Edit-in-Concert Mode
- Change from Package view to Layout View
- Net Tracing in Editing-In-Concert Mode
- Probing a Design in Edit-in-Concert Mode


© 2023 Cadence Design Systems, Inc.

## Change from Package view to Layout View

In Edit-in-Concert mode, at any point, you can switch from the package view to the corresponding layout view. To do this, right-click the canvas and choose *Switch Tabs to Layout* from the shortcut menu, as shown below:




### *Related Topics*

- Editing TILP parameters
- Edit-in-Concert
- Modify in Edit-in-Concert Mode
- Movement of Dies in Edit-in-Concert Mode
- Changes to TILP Parameters in Edit-in-Concert mode
- Net Tracing in Editing-In-Concert Mode
- Probing a Design in Edit-in-Concert Mode


© 2023 Cadence Design Systems, Inc.

## Net Tracing in Editing-In-Concert Mode

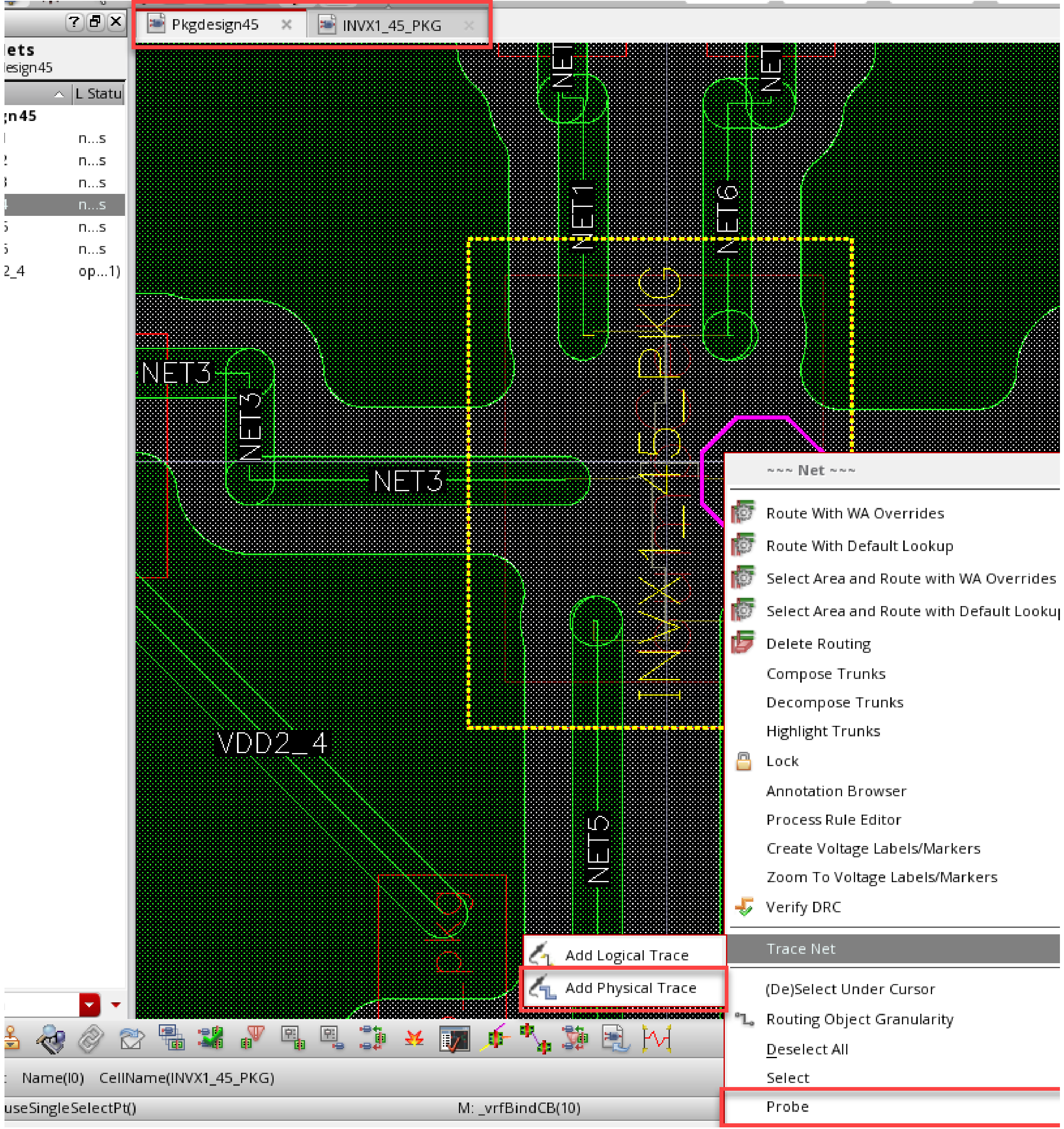
Multi-technology fabrics, such as dies and packages, can be traced for nets while editing-in-concert. Multiple traces are created from layers in a package to layers in a die.

To trace nets across multi-technology fabrics:

1. Open a design in Edit-In-Concert mode.
2. Select a net using the Navigator assistant.


© 2023 Cadence Design Systems, Inc.

3. Click *Trace Net – Add Physical Trace* in the context menu to trace through a die and package. Alternatively, you can use the `IntAddTrace` SKILL function to create a trace starting from the specified figure or net name in the specified window.





© 2023 Cadence Design Systems, Inc.

4. Click *Probe – Add* in the context menu to make a net visible through color tracing. You can use *Probe – Remove* and *Probe – Remove All* to remove color tracing.

***Related Topics***

- IntAddTrace
- Edit-in-Concert
- Modify in Edit-in-Concert Mode
- Movement of Dies in Edit-in-Concert Mode
- Changes to TILP Parameters in Edit-in-Concert mode
- Change from Package view to Layout View
- Probing a Design in Edit-in-Concert Mode


© 2023 Cadence Design Systems, Inc.

## Probing a Design in Edit-in-Concert Mode

Probing lets you interactively explore the design and locate the counterpart of a selected object in different windows on different fabrics. In Edit-in-Concert mode, you can probe nets, bump instances, and Ball Grid Array (BGA) balls. Multi-fabric net probe lets you probe nets from a package or module to a die or from a board to a package, and then to a die. When an object, for example, a net, is probed in the package die tab, it is highlighted in the corresponding layout tabs. This section covers the procedures for:

- Probing a Net
- Probing Bump or Ball

### Probing a Net

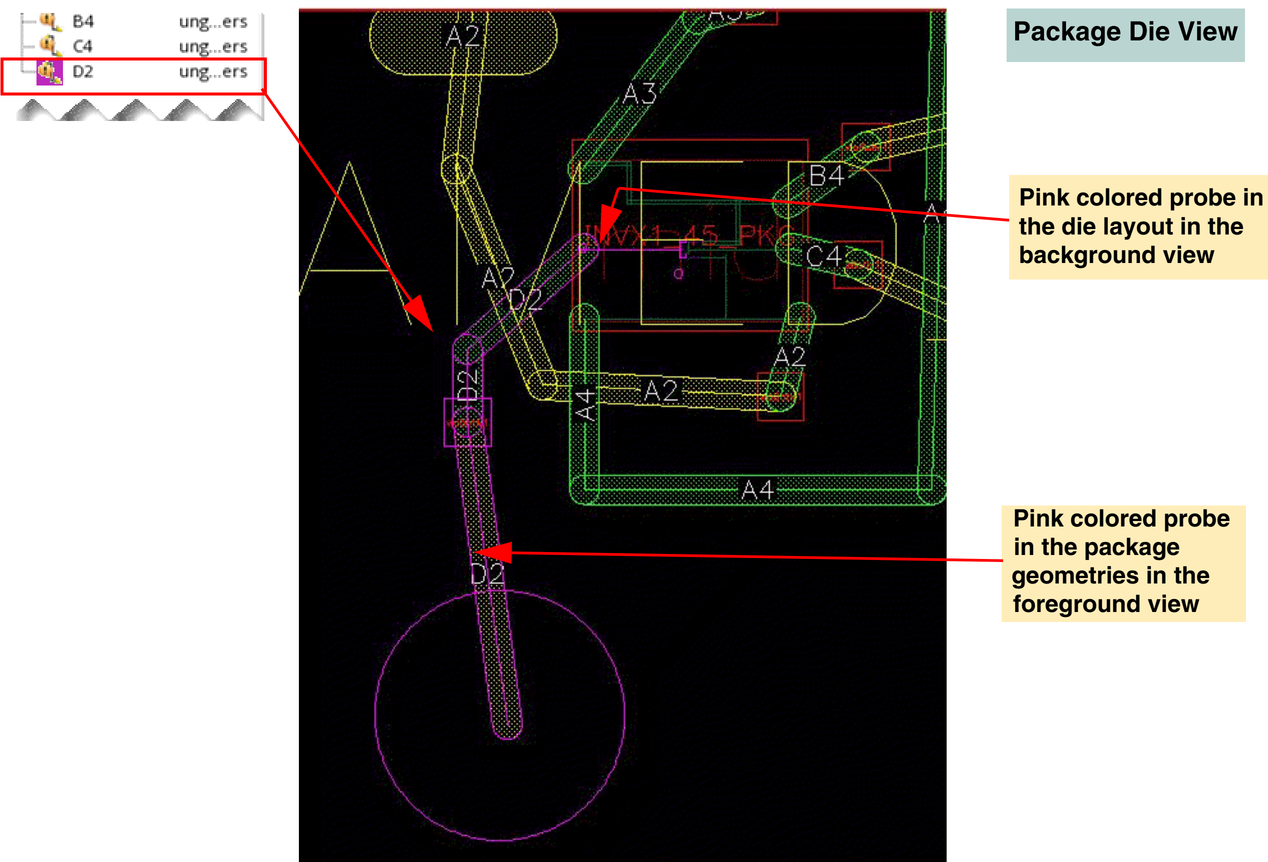
To probe a net:

1. Launch Edit-in-Concert mode from a package die of the required die instance.
2. Click either the package tab or the die layout tab.
3. Right-click the required net in the Navigator assistant.
4. Choose *Probe* and select a highlight color from the shortcut menu.

The top-level net geometries (in the foreground) and the net geometries inside the die layout (in the background) are highlighted on the package tab. Similar highlights are seen in the die layout tab.


© 2023 Cadence Design Systems, Inc.

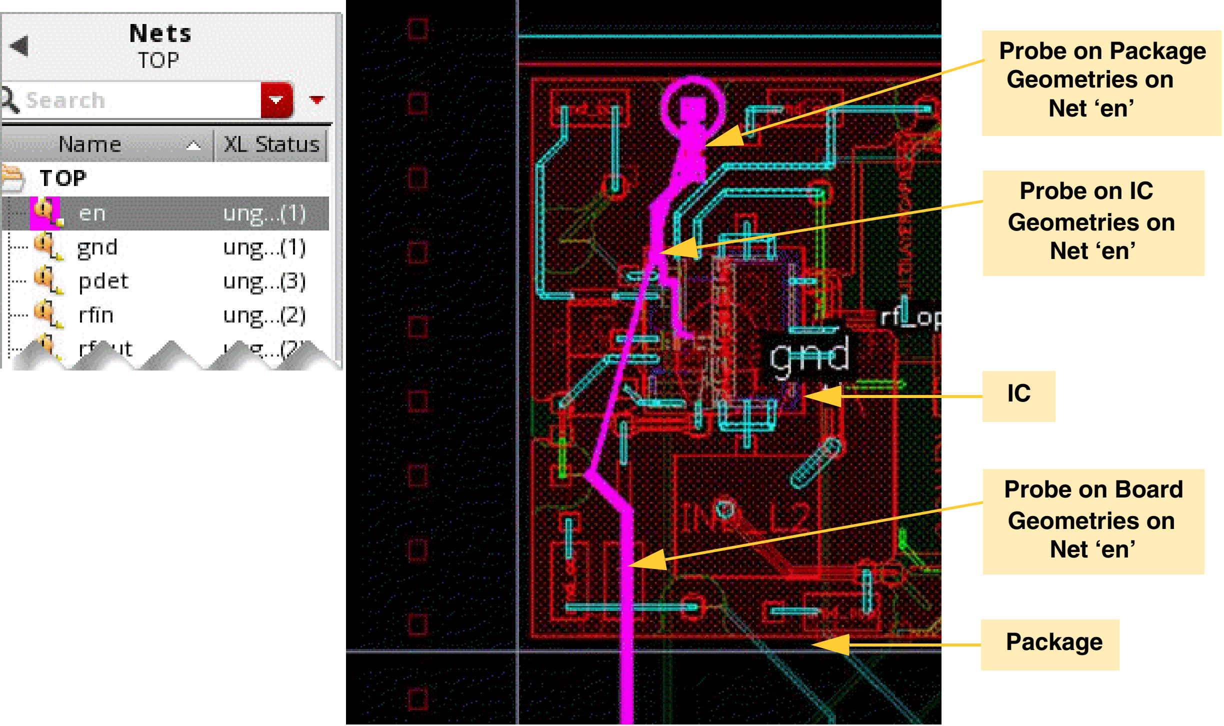
**Example1:** In the first image, net `D2` is probed (with a pink-colored highlight) in the package die.




The second image depicts the same probe, when viewed from the die layout.


© 2023 Cadence Design Systems, Inc.

**Example 2:** In the following example, a net is probed from a board fabric. The probe highlights the net object in the board, package, and die.



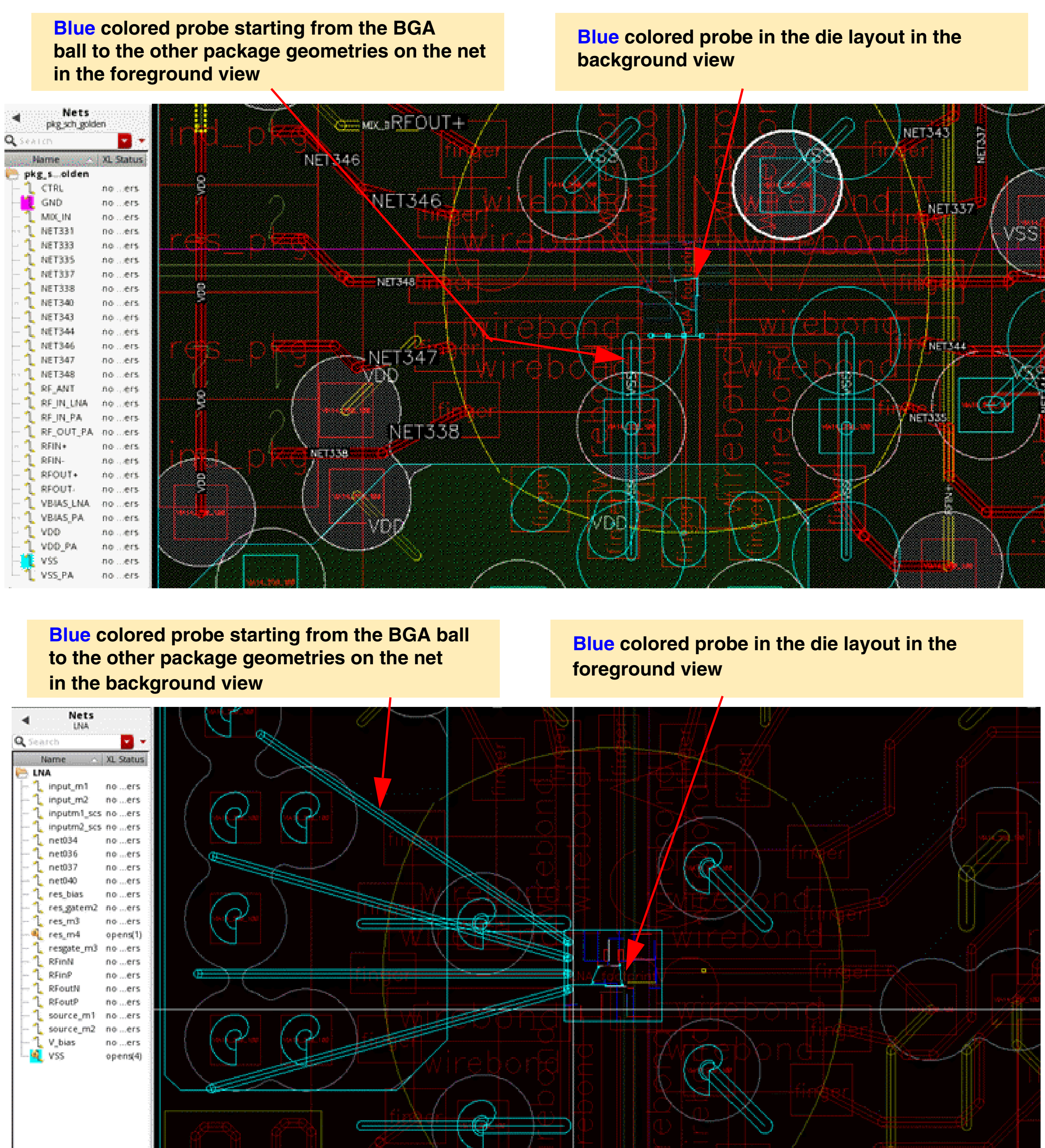

### Probing Bump or Ball

To probe a bump or ball:

1. Launch Edit-in-Concert mode from a package die of the required die instance.
2. Move to the package tab.
3. Choose *Module – Probe Bump and Ball*.
4. Click the BGA ball or bump instance to be probed.


© 2023 Cadence Design Systems, Inc.

The net connecting the selected bump or BGA ball is highlighted hierarchically across the top level and die layout. In the following examples, the selected net is highlighted hierarchically starting from the selected BGA ball.





© 2023 Cadence Design Systems, Inc.

To select balls and bumps in a design, use *Pins* in the *Objects* panel of the Palette assistant.

### *Related Topics*

- IntAddTrace
- Edit-in-Concert
- Modify in Edit-in-Concert Mode
- Movement of Dies in Edit-in-Concert Mode
- Changes to TILP Parameters in Edit-in-Concert mode
- Change from Package view to Layout View
- Net Tracing in Editing-In-Concert Mode


© 2023 Cadence Design Systems, Inc.

# Checking and Fixing IO Pad Locations

Components of the die layouts and die packages that are not edited in Edit-in-Concert mode might not be synchronized with each other. Use the *Module – Layout Vs Abstract* submenu to run the following LVA commands to check for and fix mismatches between a die package and its constituent die layouts:

- Running the LVA Checker
- Running the LVA Fixer

**Note:** The *Check IO Pad Locations*, *Fix IO Pad Locations* (and *Options*) commands for IC fabrics is only available in Edit-in-Concert mode.

You can run these commands on all or selected die instances or IO pads.

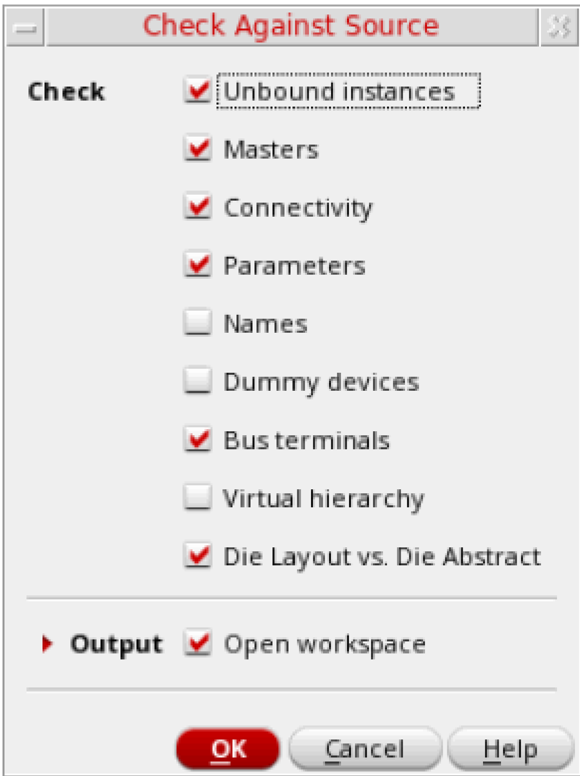
*Important*

The *Check IO Pad locations*, *Fix IO Pad locations*, and *Edit-in-Concert* commands do not run on instances that are bound to a layout with shape-based IO pads. A warning message is displayed listing all the instances that have been ignored.


© 2023 Cadence Design Systems, Inc.

*Important*

Alternatively, you can use the *Connectivity – Check – Against Source* (CAS) command to check the differences between the schematic and the layout. The Check Against Source form includes the *Die Layout vs. Die Abstract* check box that lets you check for mismatches between a die package and its constituent die layouts. Violations are listed on the *CAS* tab of the Annotation Browser.




***Related Topics***

Edit-in-Concert

Modify in Edit-in-Concert Mode

Running the LVA Checker

Running the LVA Fixer

IO Pad Connectivity Mismatch Fixes

IO Pad Number Mismatch Fixes


© 2023 Cadence Design Systems, Inc.

## Running the LVA Checker

LVA checker can be run from the board, package, module, or die layout.

- If run from board layout, checking is done between package layout and package abstract.
- If run from package layout, checking is done between die layout and die abstract. Module fabrics are treated the same way as package fabrics.

  When a package layout and a board layout are open in Edit-in-Concert mode, you can run the LVA checker between the package layout and the package footprint in the board by selecting the ball grid array (BGA) or land grid array (LGA) instance in the package layout.

To run the LVA checker:

1. Select the checks that the LVA checker must run in the Virtuoso RF Options form.
2. Select the required dies or instances on which the LVA checker must run.

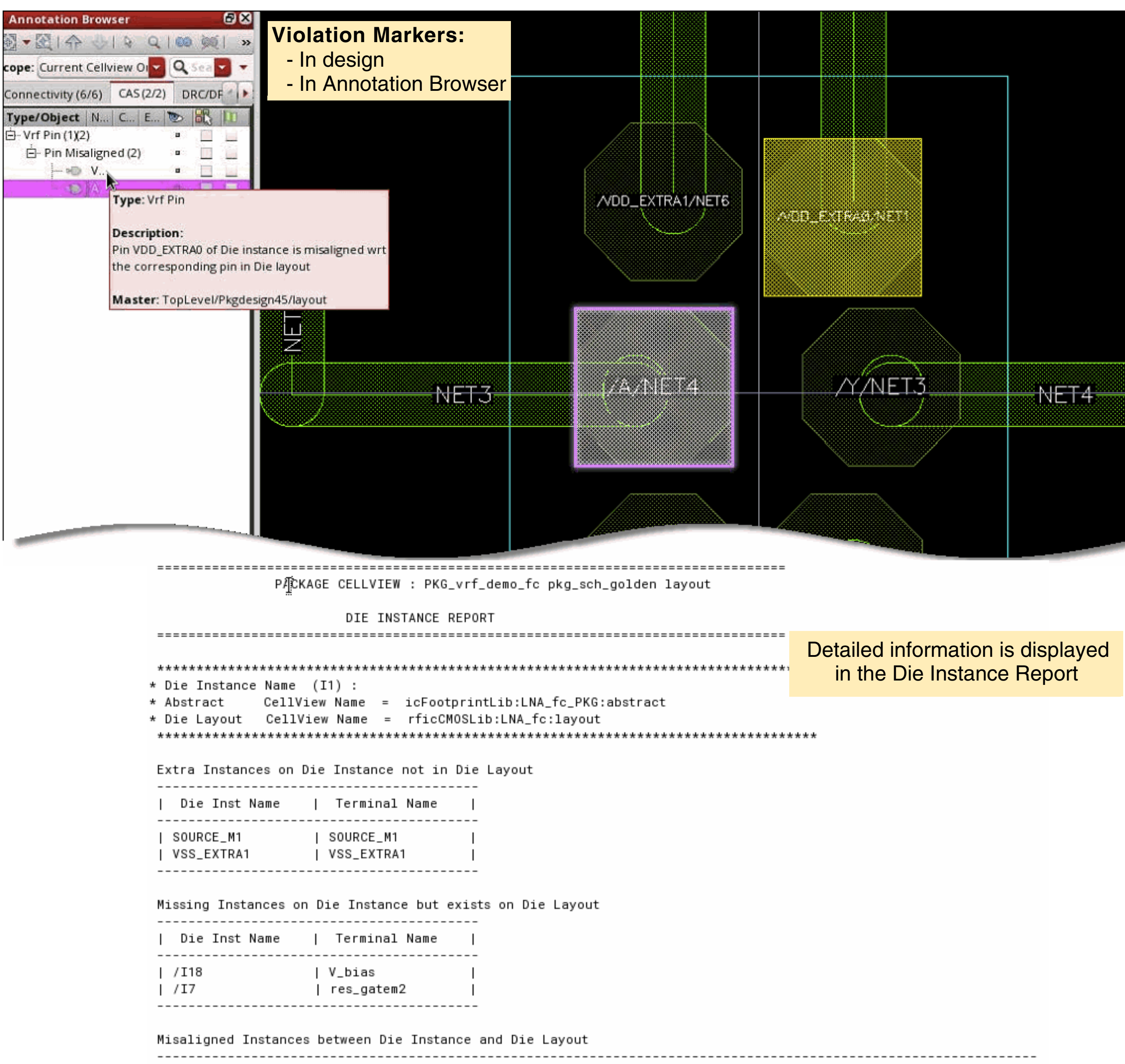
   The command runs on one or more selected dies and instances. If there is nothing selected, it runs on all the dies and instances in the design.
3. Choose *Module – Layout Vs Abstract – Check IO Pad locations* from either the package layout or the die layout to start the LVA checker.

Violation markers are created on the CAS tab of the Annotations Browser to indicate the following types of violations:

- Connectivity mismatches
- Missing IO pads
- Extra IO pads
- Misaligned IO pads


© 2023 Cadence Design Systems, Inc.

A detailed DIE INSTANCE REPORT is displayed in the CIW.




## Related Topics

Checking and Fixing IO Pad Locations

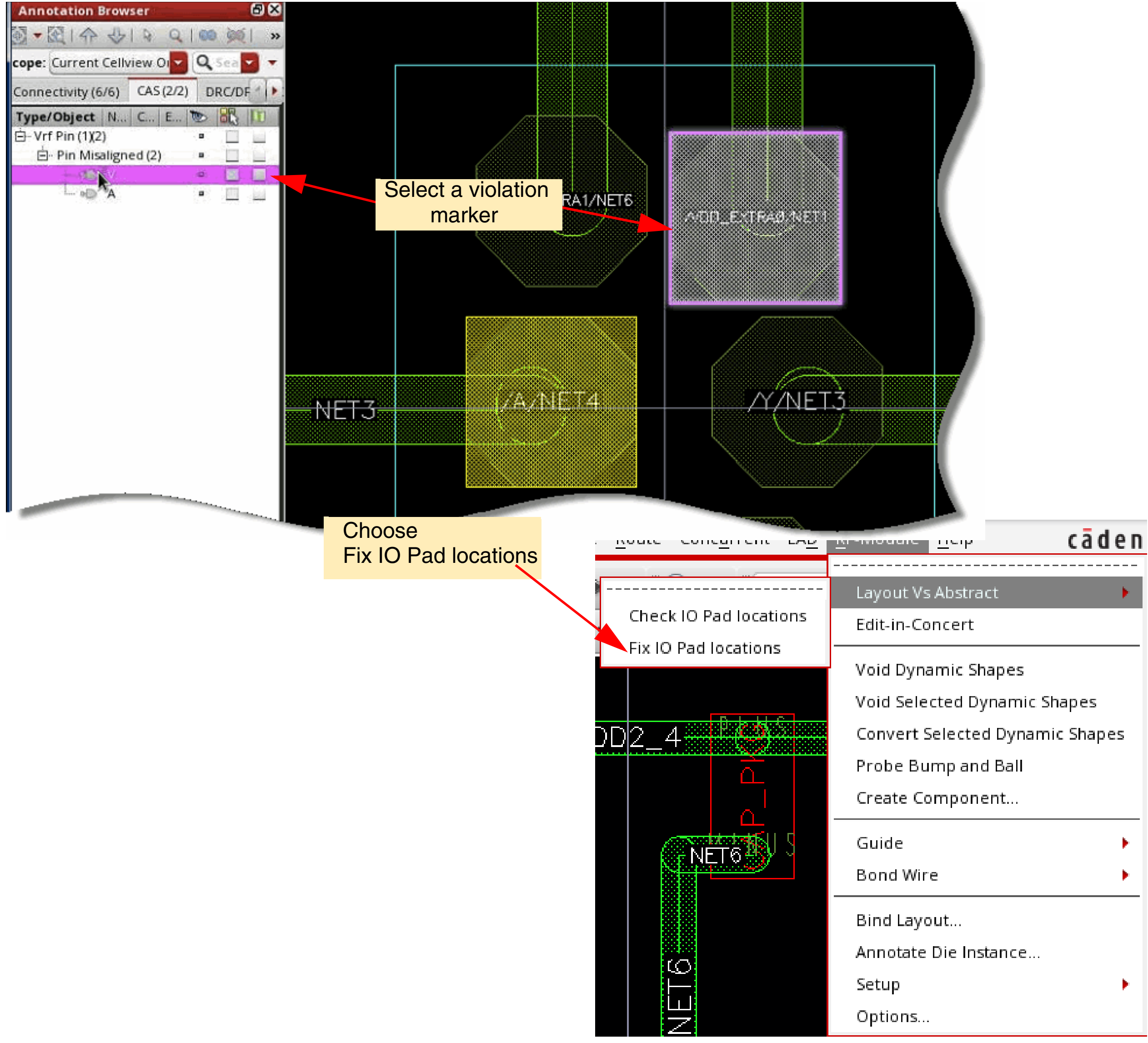
Running the LVA Fixer

casDieLayoutVsAbstract


© 2023 Cadence Design Systems, Inc.

## Running the LVA Fixer

To fix the violation markers created by the LVA checker, select the marker in the Annotation Browser and then choose *Module – Layout Vs Abstract – Fix IO Pad locations*.




The LVA fixer makes the required changes to fix the violation:

- Fixes connectivity mismatches.
- Fixes the differences in the number of IO pads by doing one of the following:
    - Creating missing IO pads


© 2023 Cadence Design Systems, Inc.

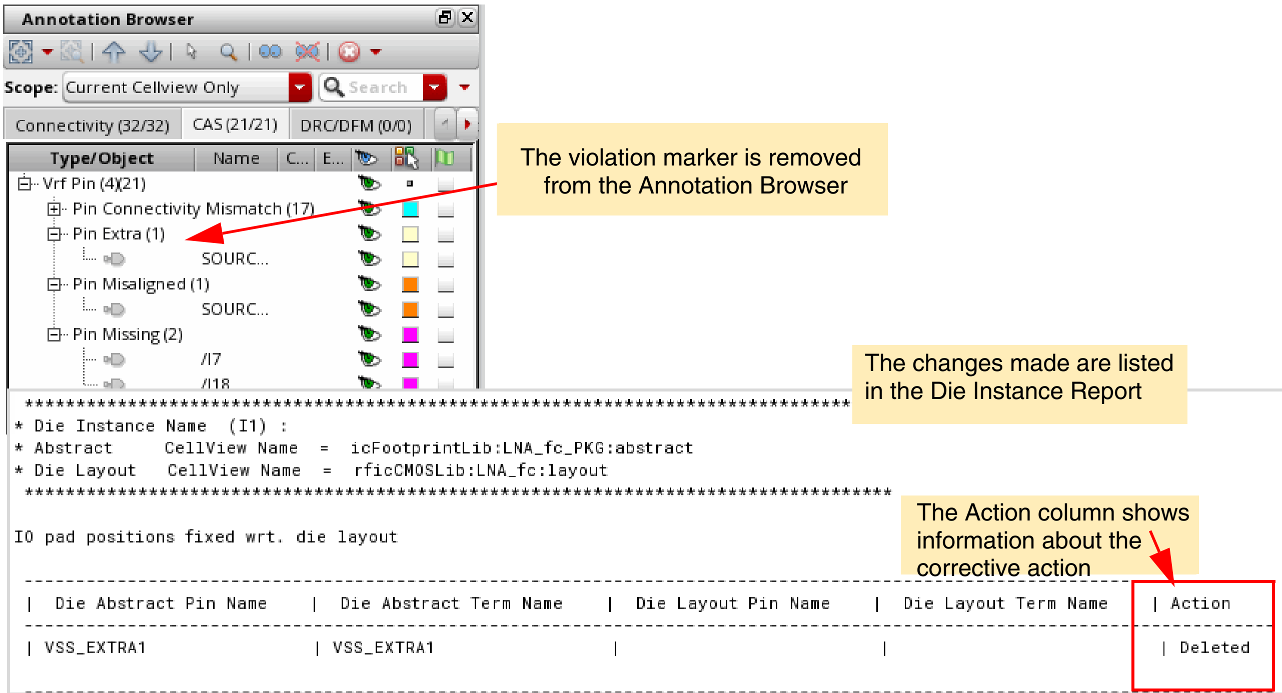
- ❑ Deleting extra IO pads

  Then, it updates die abstract's schematic and symbol accordingly.

- ■ Aligns IO pads

  Use the Virtuoso RF Options form to select the violations that the LVA checker must fix.

The markers are cleared from the design and the Annotation Browser. The changes made are listed in the Die Instance Report.




## *Related Topics*

IO Pad Connectivity Mismatch Fixes

IO Pad Number Mismatch Fixes


© 2023 Cadence Design Systems, Inc.

## IO Pad Connectivity Mismatch Fixes

The LVA fixer addresses the connectivity mismatches that the LVA checker reports for the die abstract, and propagates the changes to the corresponding symbol and schematic views of the die abstract. Therefore, the connectivity in all associated views are synchronized.


© 2023 Cadence Design Systems, Inc.

| Fabric | Type of Mismatch | Resolution |
|---|---|---|
| Non-IC Fabric<br>**Note:** The die layout view in the background is the golden view. | IO pads have connectivity in the die layout, but have no connectivity (unconnected IO pads) in the die abstract | The LVA fixer refers to the bump connectivity information in the die layout and establishes similar connectivity in the die abstract. It also propagates the changes to the corresponding die abstract's symbol and schematic views.<br>**Note:** If the IO pad is connected to a net that has multiple IO pads/pins, for example, VDD has three IO pads where the connectivity of any bump is changed to VDD, no update is needed in die abstract's symbol because the symbol has only one VDD irrespective of the fact that how many IO pads of VDD exist in the die abstract.<br>**Note:** If the IO pad is connected to a net that has no existing IO pads in the die abstract, for example, if the IO pad connectivity is changed to pin A that has no existing IO pads, the die abstract's symbol needs to be updated. |
| | IO pads have connectivity in the die abstract, but have no connectivity (unconnected IO pads) in the die layout | The LVA fixer removes connectivity from the IO pads in the die abstract. It also propagates the changes to the corresponding die abstract's symbol and schematic views. |
| | IO pads with different connectivities in the die layout and the die abstract | The LVA fixer establishes connectivity in the die abstract as per the die layout and propagates the changes to the corresponding die abstract's symbol and schematic views. |


© 2023 Cadence Design Systems, Inc.

| Fabric | Type of Mismatch | Resolution |
|---|---|---|
| IC Fabric<br>**Note:** The package view in the background is the golden view. | IO pads with different connectivities in the die layout and the die abstract | ■ The LVA fixer establishes connectivity in the die layout as per the die abstract.<br>■ Else, the LVA fixer issues a message indicating that the die layout is not in sync with the die abstract's schematic or symbol . |

***Related Topics***

Edit-in-Concert

Modify in Edit-in-Concert Mode

Running the LVA Checker

Running the LVA Fixer

IO Pad Number Mismatch Fixes


© 2023 Cadence Design Systems, Inc.

## IO Pad Number Mismatch Fixes

The LVA fixer updates the die abstract or die layout by adding/deleting the IO pads while fixing in Edit-in-Concert mode. The following table represents various scenarios for updating the symbol and schematic of the die abstract.

| Fabric | Type of Mismatch | Resolution |
| --- | --- | --- |
| Non-IC Fabric<br>**Note:** The die layout view in the background is the golden view. | More IO pads on the die abstract compared to the die layout | ■ LVA fixer deletes IO pads from the die abstract but it updates the die abstract's symbol based on the following conditions only:<br>❑ If the IO pad is connected to a net that has multiple IO pads/ pins, for example, VDD has three IO pads where one is deleted and two remain, no update is needed in the die abstract's symbol. The reason being that the symbol has only one VDD irrespective of the number of IO pads of VDD that exist in the die abstract.<br>❑ If the IO pad is connected to a net that has only one IO pad/ pin, for example, pin A has one IO pad that is being deleted, the die abstract's symbol needs to be updated. |
| | Less IO pads on the die abstract compared to the die layout | LVA fixer adds IO pads to the die abstract and updates the die abstract's schematic and symbol. |


© 2023 Cadence Design Systems, Inc.

| Fabric | Type of Mismatch | Resolution |
|---|---|---|
| IC Fabric<br>**Note:** The package view in the background is the golden view. | More IO pads on the die layout compared to the die abstract | ■ LVA fixer deletes IO pads from the die layout.<br>■ The LVA fixer issues a message indicating that the die layout is not in sync with the die layout's schematic or symbol. |
| | Less IO pads on the die layout compared to the die abstract | ■ LVA fixer adds IO pads in the die layout.<br>■ The LVA fixer issues a message indicating that the die layout is not in sync with the die abstract's schematic or symbol. |

***Related Topics***

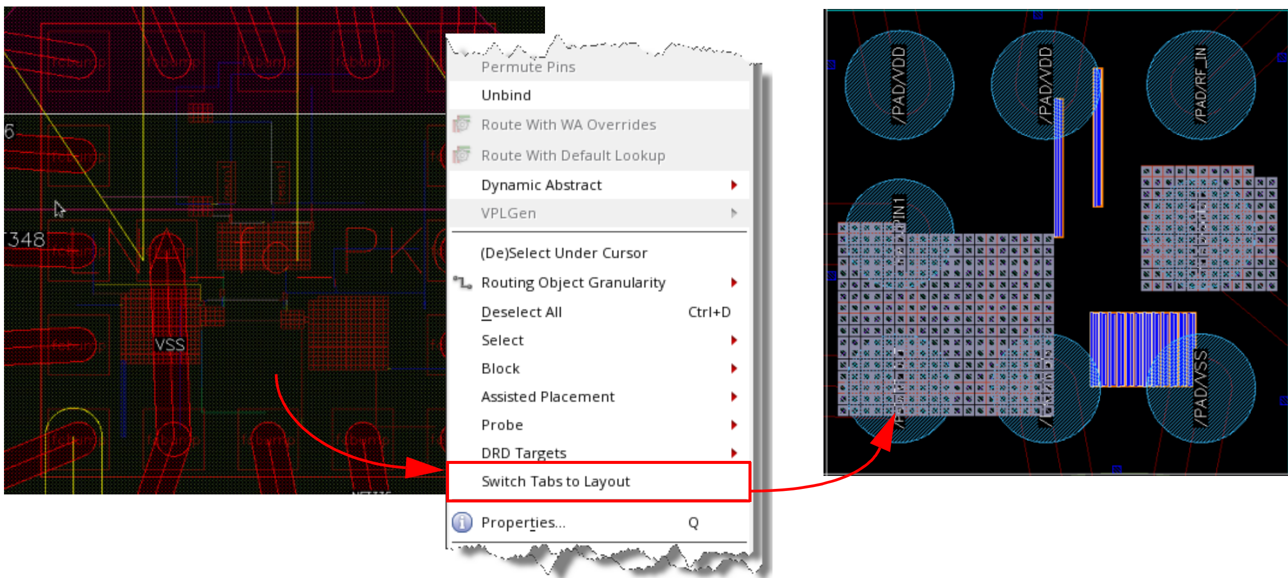
- Edit-in-Concert
- Modify in Edit-in-Concert Mode
- Running the LVA Checker
- Running the LVA Fixer
- IO Pad Connectivity Mismatch Fixes


© 2023 Cadence Design Systems, Inc.

# Exiting Edit-in-Concert Mode

To exit Edit-in-Concert mode, close one or more tabs. A message prompting you to save the changes is displayed. Click *Yes* to retain the changes. If you click *No*, the changes made in Edit-in-Concert mode are lost. As a result, the dies might be asynchronous.

To check for such violations, use the layout versus abstract (LVA) checker. Use the LVA fixer to fix such violations.




### *Related Topics*

[Edit-in-Concert]

[Launch Edit-in-Concert Mode]

[Modify in Edit-in-Concert Mode]

# View-in-Concert Mode

View-in-concert mode allows designers with limited permissions to view package and IC representations alongside each other, similar to edit-in-concert mode. This functionality is used by designers to analyze their focus sections of an IC in the context of the package and view the package routing near it. The other items in the *Module* menu are enabled or disabled based on the read or write permissions of a cellview. The *View-In-Concert* menu item is visible only when the cellview is opened in read-only mode.


© 2023 Cadence Design Systems, Inc.

You can trace and probe nets across fabrics in view-in-concert mode. The probing of bump and ball is also enabled.

The following edit-in-concert features are partially available in view-in-concert mode.

- For read-only cellviews, you can view dies in the Virtuoso 3D viewer but are not able to configure die stacks because changes to die parameters are not allowed. All editable fields in the Configure Module Stack form are disabled.
- You can view violations or markers though the Layout vs Abstract (LVS) checker and fixer. You are also able to run the checker on read-only package cellviews but you cannot save or fix markers in the Annotation Browser.
- You can run cross-fabric checks on board, module, or package cellview. It generates the complete report, markers are created, and the Cross Fabric Check Violation Summary and Navigation form is launched, but you cannot save or fix the markers.
- You can view-in-concert shape-based abstract cellviews but cannot perform LVA check or trace nets.

The following edit-in-concert features are not available in view-in-concert mode.

- Any movement in the IO pad position in a die layout is not replicated on the die footprint in package cellview.
- Bind Layout binds a die layout with a die instance by adding properties on die instance or abstract cellview. Therefore, it requires a package to be editable.
- Bump management commands are not supported.

### *Related Topics*

Edit-in-Concert


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# 9

# Stacked Modules Management

Stacked modules are the basic building blocks of modern micro-electronic systems. Miniaturization of devices, for example cellular devices and medical devices, led to an increased demand for compact integrated circuit (IC) integration without any impact on device performance. To achieve compaction, IC dies are stacked and bonded. Stacking dies increases the available surface area on a package. Also, the electrical connections between stacked ICs are stronger than those on regular ICs.

***Related Topics***

Stacked Modules

Benefits of Implementing Stacked Modules

Components of a Stacked Module

Stacked Module Assemblies

Virtuoso Stacked Silicon Solution Flow

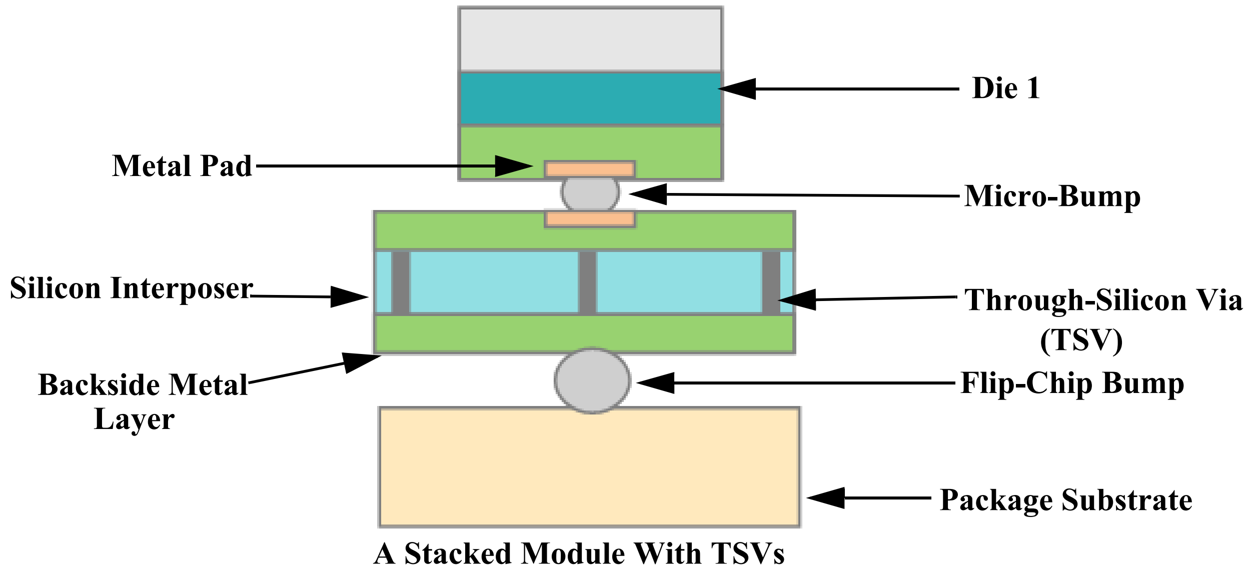
Configuring a Stack


© 2023 Cadence Design Systems, Inc.

# Stacked Modules

A stacked module is a circuit that is integrated vertically with two or more dies that are stacked, aligned, and bonded using bumps. In conventional ICs, dies are connected using bumps or bonding pads on one side of the chip. In stacked modules, re-distributed layers (RDLs) are generated at the back of the chip. Bumps can be placed on both sides of the chip. Through-silicon vias (TSVs), through-package vias (TPVs), bumps, and silicon interposers help establish interconnections between dies.

The following diagram depicts a stacked module.




A Stacked Module With TSVs

In stacked modules, dies can be stacked to create several tiers, with multiple dies on each tier. Each die can be flipped or rotated in the package.

Virtuoso Stacked Silicon solution enables you to specify the die configuration, interconnection between dies, and the relative positions of dies. You can also manipulate TSVs and bumps and synchronously edit dies.


© 2023 Cadence Design Systems, Inc.

# Benefits of Implementing Stacked Modules

- **Heterogeneous Dies**

  An IC package can accommodate multiple heterogeneous dies, such as logic, memory, analog, RF, and micro-electrical mechanical systems (MEMS) at different process nodes, such as 28nm for high-speed logic and 130nm for analog. Multiple dies that are manufactured separately using different process technologies can be stacked and bonded into a single package.

- **Isolated Digital and Analog Substrates**

  Using the Virtuoso Stacked Silicon solution ensures the isolation of digital substrates from analog substrates, thus avoiding any digital-to-analog or analog-to-digital substrate noise propagation.

- **Reduced Form Factor**

  Stacking dies allows miniaturization, thus saving space on the board and in the end product. This solution is ideal for extremely compact mobile devices.

- **Low Power Consumption**

  Using stacked devices can reduce power because big drivers are no longer needed. A stack can use small I/O drivers with lower power. Reduced RLC (which stands for resistance-inductance-capacitance) helps reduce power. The interconnect between packages is also reduced, leading to better performance and power profiles.

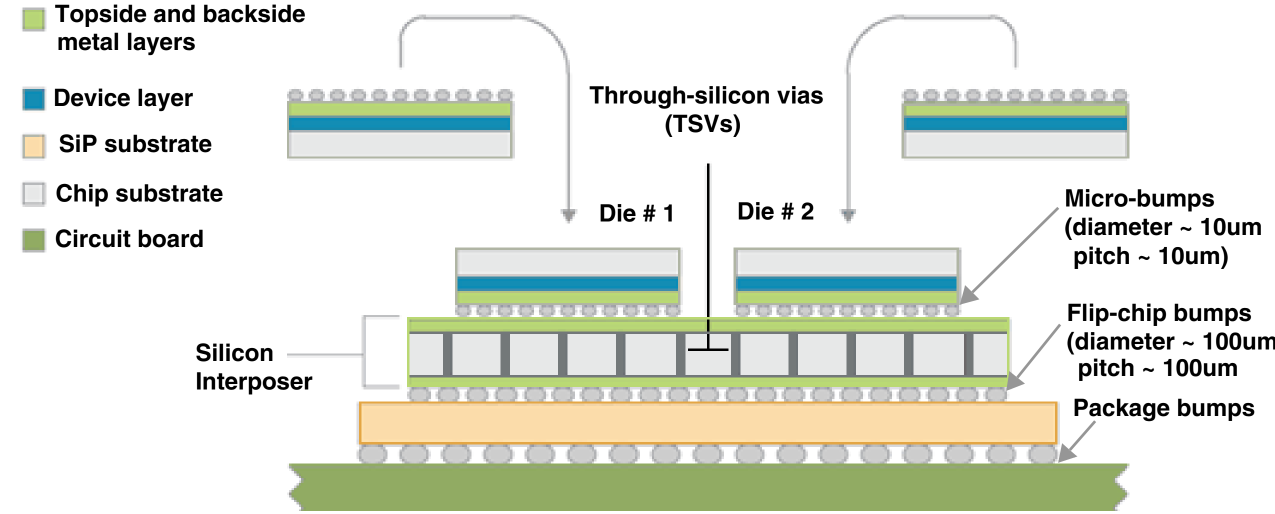
### *Related Topics*

- Stacked Modules


© 2023 Cadence Design Systems, Inc.

# Components of a Stacked Module

The following diagram depicts a stacked module.




The key components of a stacked module are:

- **Die**

  A die is an unpackaged chip.

- **Die Pad**

  A die pad is a metal contact on a die that is used to make electrical connections between the die and the component. It is also called an IO pad or a die pin. On flip-chip dies, it is called a solder bump, whereas for wire-bound ICs it might be called bondwire pads.

- **Die Stack**

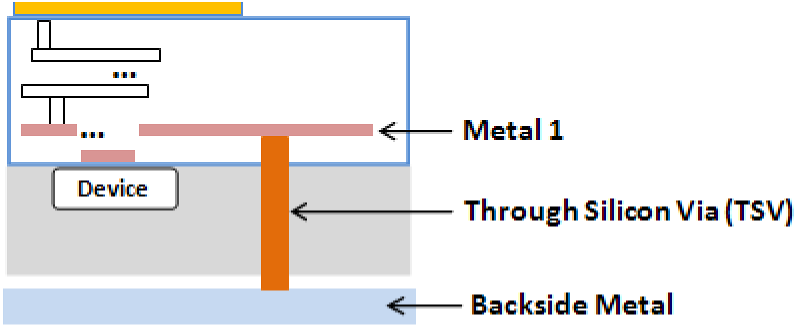
  A die stack is a vertical stack of one or more dies, spacers, and interposers.

- **Through-Silicon Via (TSV)**

  TSVs are copper vias with diameters ranging between 1 and 30 microns that pass through a silicon substrate. The top cap layer of a TSV is the first normal routing layer, Metal1, and the bottom cap layer is the backside metal layer. Therefore, TSVs enable


© 2023 Cadence Design Systems, Inc.

signal propagation and power delivery between the top metal layer and the backside metal layer.




**TSV Cross-Section**

- **Bump**

  A bump includes a solder ball that is placed on the top metal layer or backside metal layer of a die and the metal pad beneath it. Aligned bumps between dies are called micro-bumps or landing pads.

  Cross-die signals and power travel to adjacent dies through micro-bumps. Bumps between a die and the package substrate are called flip-chip bumps.

- **Silicon Interposer**

  A silicon interposer is an electrical routing channel that helps establish much finer die-to-die interconnections, thereby increasing the performance and reducing the power consumption. Silicon interposers are dies that can include TSVs to provide connections from the upper metal layers to additional backside metal layers.

### *Related Topics*

- Stacked Modules

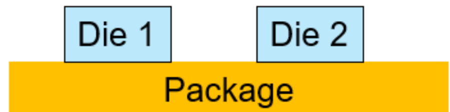

© 2023 Cadence Design Systems, Inc.

# Stacked Module Assemblies

Virtuoso Stacked Silicon solution supports the following types of stacked module assemblies:

- **Package -> ICs**

  In this type of assembly, individual dies are stacked on a package. Dies can be stacked horizontally or vertically.

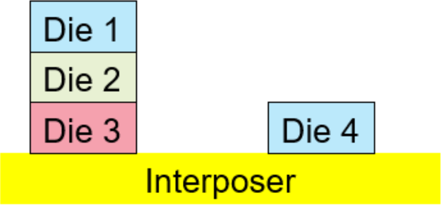


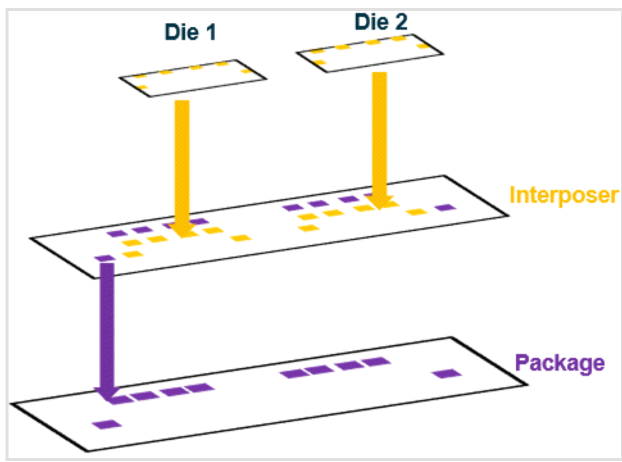

- **Generic Container -> Stacked ICs**

  In the absence of a package, a generic container is used to instantiate a Technology Independent Layout Pcells (TILP). Dies are stacked vertically on a generic container.

- **Generic Container or Package -> Interposer -> Stacked ICs**

  In such an assembly, dies are stacked on an interposer. The interposer is stacked on a package or generic container. The interposer acts as a routing channel to establish die-to-die and die-to-package connections using TSV.

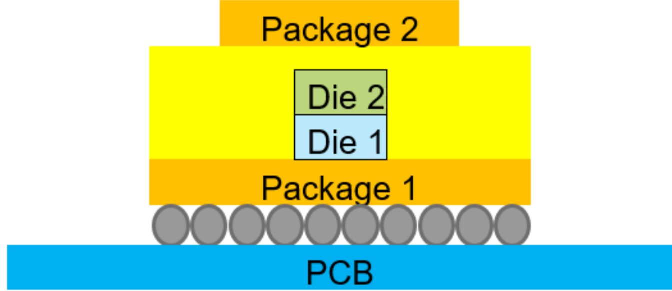



- **Board -> Package -> IC**


© 2023 Cadence Design Systems, Inc.

In such an assembly, a package is mounted vertically on a PCB. The assembly could have a combination of dies, stacked dies, packages, or stacked packages mounted on the board.




***Related Topics***

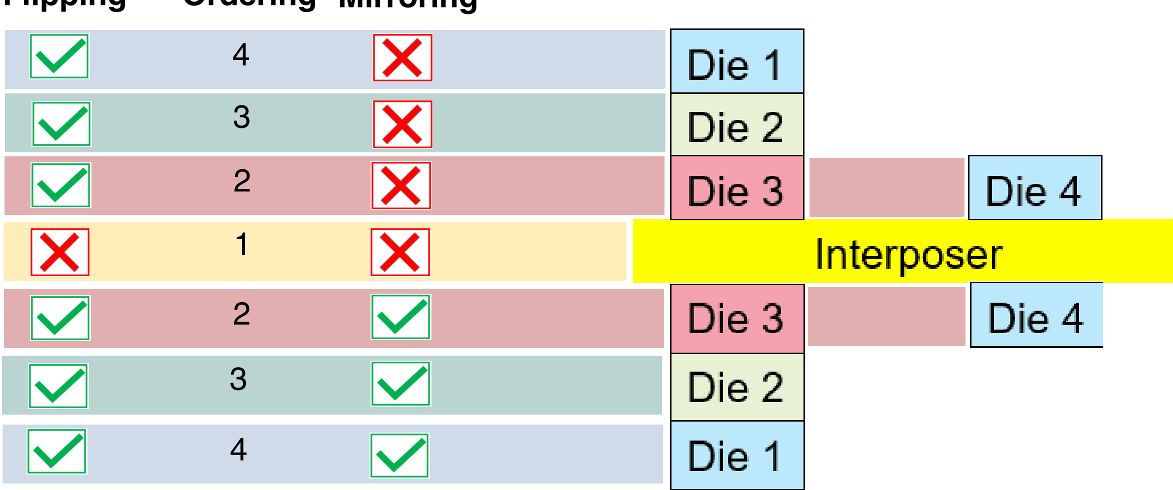
- Stacked Modules
- Components of a Stacked Module
- Configuring a Stack


© 2023 Cadence Design Systems, Inc.

# Configuring a Stack

Before you start with bump management tasks, it is important to configure a stack. The following diagram illustrates a stack with dies vertically mounted on either sides of an interposer. To configure a die stack, you define the following die properties:

- **Flipping:** Specifies whether the dies are to be used as they are or must be flipped. In the following example, the interposer is used as it is, whereas the dies mounted above and below the interposer are flipped.
- **Ordering:** Specifies the sequence in which dies are to be stacked. The die order is assigned from the center. Therefore, the interposer is considered the first die (order 1), followed by the subsequent dies.
- **Mirroring:** Specifies the orientation of dies. In the following illustration, the flipped dies that are stacked above the interposer are not mirrored, whereas the ones stacked below the interposer are.

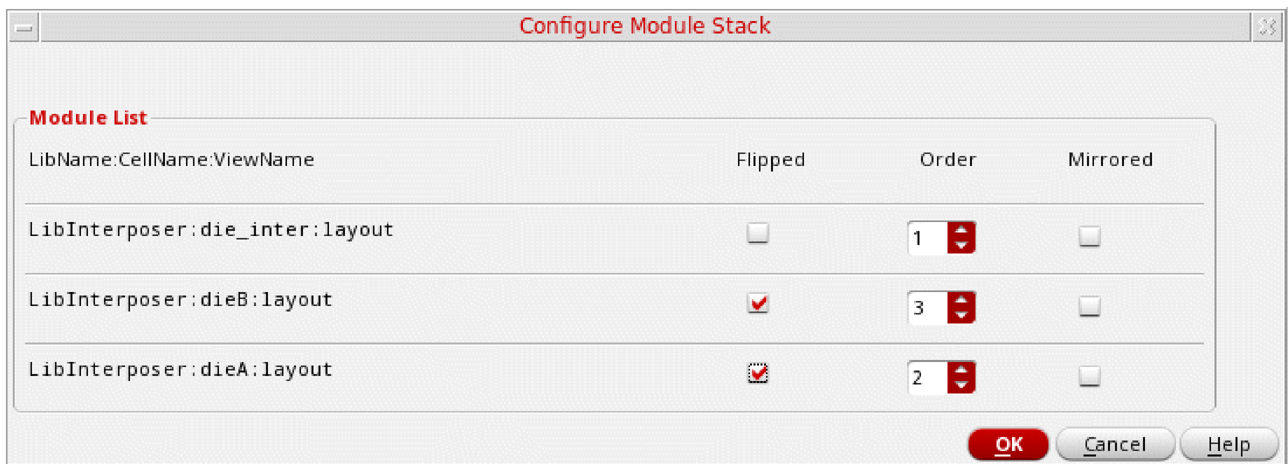



To configure a stack:

1. Add the required dies to the design.


© 2023 Cadence Design Systems, Inc.

2. Click *Module – Configure Module Stack* to display the Configure Module Stack form.



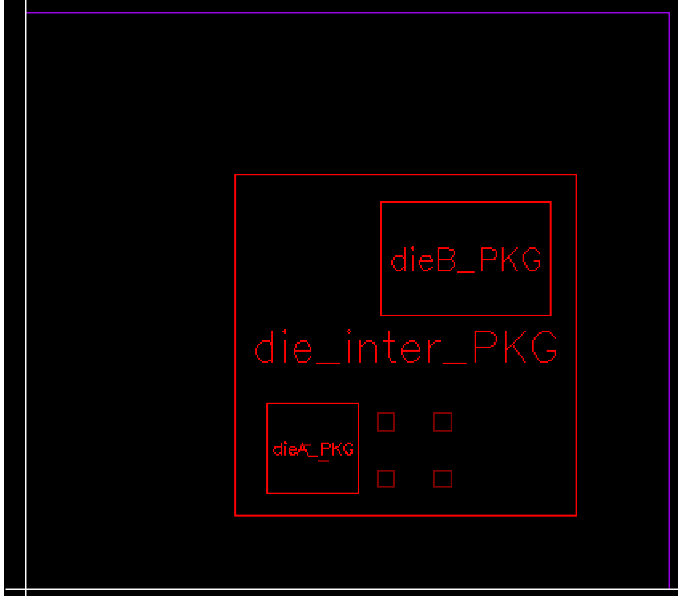

All dies in the design are listed in the *Module List*.

3. Select *Flipped* for dies to be flipped in the stacked assembly. For example, for a design where two dies are mounted on an interposer, select *Flipped* for the two dies.

4. Use *Order* to specify the sequence in which the dies are to be stacked.

5. Select *Mirrored* for dies to be mirrored.

6. Click *OK* to apply the settings.





© 2023 Cadence Design Systems, Inc.

***Related Topics***

- Stacked Modules
- Components of a Stacked Module

# Die Operations

A stacked module comprises dies that are integrated vertically, aligned, and bonded using bumps. Bump management tasks are an integral part of the Virtuoso Stacked Silicon solution. Bump management tasks can be categorized into die operations and inter-die operations. Die operations are performed on individual dies and inter-die operations impact two or more dies.

In Virtuoso, bump management tasks can be run only when the design is open in Edit-In-Concert mode. In this mode, die commands are available from the tabs corresponding to individual dies and inter-die commands are available from the package or container tab.

***Related Topics***

Creating Bumps and TSVs

Assigning Connectivity between Bumps

Unassigning Bump Connectivity

Deleting Unassigned Bumps

Moving Pins to Bumps

Updating Bumps to the Abstract View

Saving Bumps to File

Creating Bumps from File


© 2023 Cadence Design Systems, Inc.

# Creating Bumps and TSVs

In flip-chip dies, bumps and TSVs are used to establish connections between dies. Bumps connect individual dies, whereas TSVs cut through the silicon substrate to connect the top metal layer to the backside metal layer.

Bumps and TSVs are created at the die level. Before generating bumps and TSVs, ensure that the following definitions are in place:

- **Bump cellview:** Create the required bump shape in a cellview. Set the cellview type as `coverBump`.
- **Vias:** Define the required standard vias in the `viaDefs` section of the technology file. For more information, see Specifying Via Definitions.

Using this information, you can generate bump arrays and TSVs on the required dies.

To create bumps:

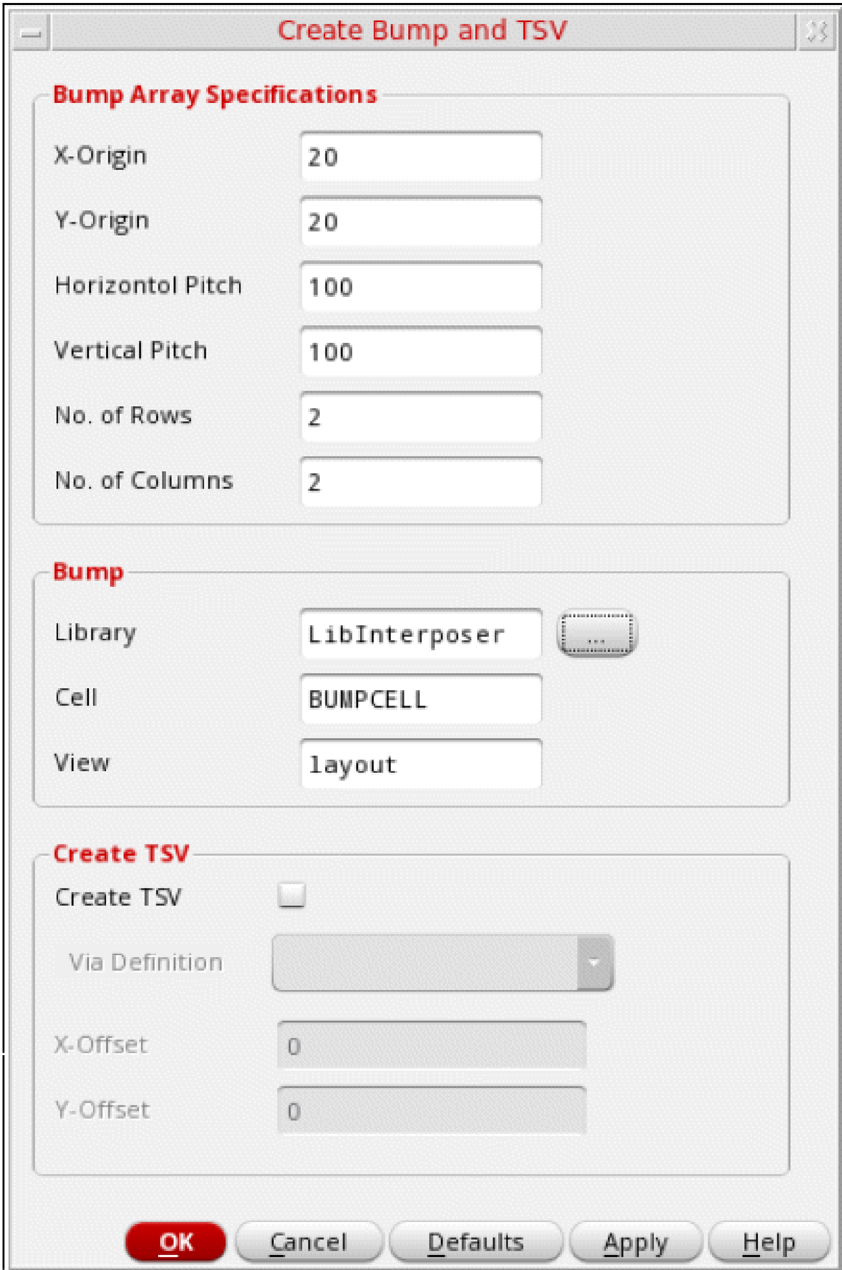
1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In-Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.

2. Click the tab corresponding the die on which you want to create the bumps and TSVs.


© 2023 Cadence Design Systems, Inc.

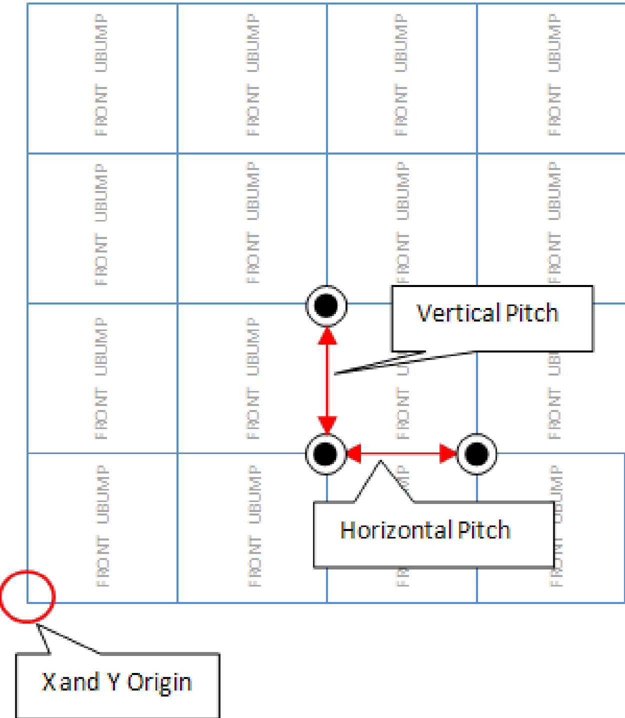
3. Select *Module – Bump Management – Create Bumps – Array* to open the Create Bump and TSV form.





© 2023 Cadence Design Systems, Inc.

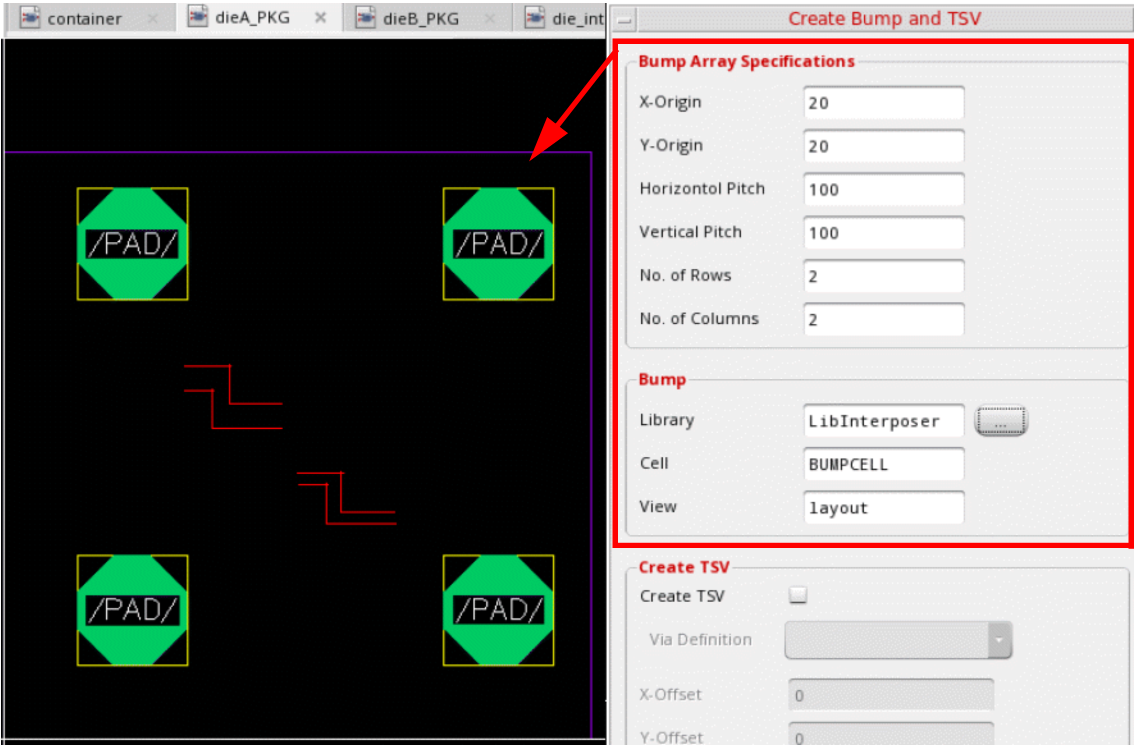
4. Specify the X and Y coordinates of the first bump of the array in the *X-Origin* and *Y-Origin* fields, respectively. The following image depicts how the pitch and origin values are applied.




5. Specify the horizontal distance between the bump edges in the *Horizontal Pitch* field.
6. Specify the vertical distance between the bump edges in the *Vertical Pitch* field.
7. Specify the number of rows to be generated in the array in the *No. of Rows* field.
8. Specify the number of columns to be generated in the array in the *No. of Columns* field.
9. Specify the reference bump cell in the *Library*, *Cell*, and *View* fields in the *Bump* group box. Optionally, click *Browse* and select the cellview from the *Library Browser*.
10. Click *Apply*.


© 2023 Cadence Design Systems, Inc.

A bump array as per your specifications is generated in the layout canvas.



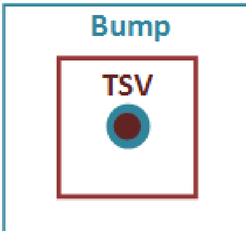

To create a TSV:

1. Select *Create TSV*.
2. Select a via from the *Via Definition* cyclic field. The *Via Definition* cyclic field lists all the standard and custom vias that are available in the technology file.

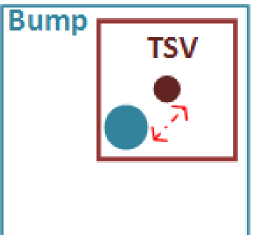
   **Note:** Via definitions are listed in the order in which they are defined in the `validVias` section in the technology file.


© 2023 Cadence Design Systems, Inc.

3. Specify the X and Y offsets for the TSVs. These values define the offset from the center of each TSV to the center of the corresponding bump.




The center of the TSV is aligned with the center of the bump.
Here:
X-Offset = 0
Y-Offset = 0




The center of the TSV is not aligned with the center of the bump.
Here:
X-Offset = 4
Y-Offset = 4

4. Click *OK* or *Apply*.

TSVs as per your specifications are created on the active die.

### *Related Topics*

- Create Bump and TSV Form (form reference)


© 2023 Cadence Design Systems, Inc.

# Assigning Connectivity between Bumps

Bumps created on dies have to be connected to nets. Bumps are assigned to nets based on their proximities.

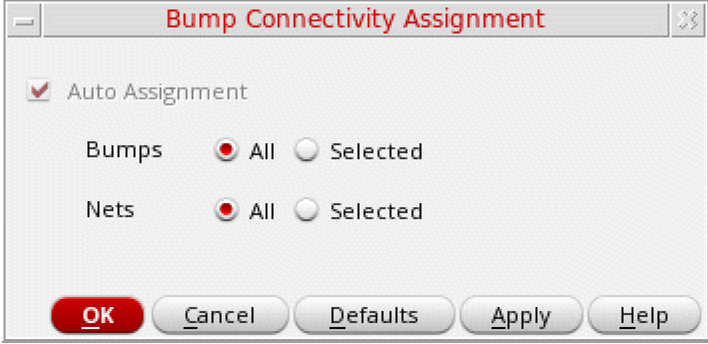
To assign connectivity to bumps:

1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In-Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.

2. Click the tab corresponding the die on which you want to assign connectivity to bumps.

3. Choose *Module – Bump Management – Define Bump Connectivity – Assign Connectivity to Bumps* to display the Bump Connectivity Assignment form.



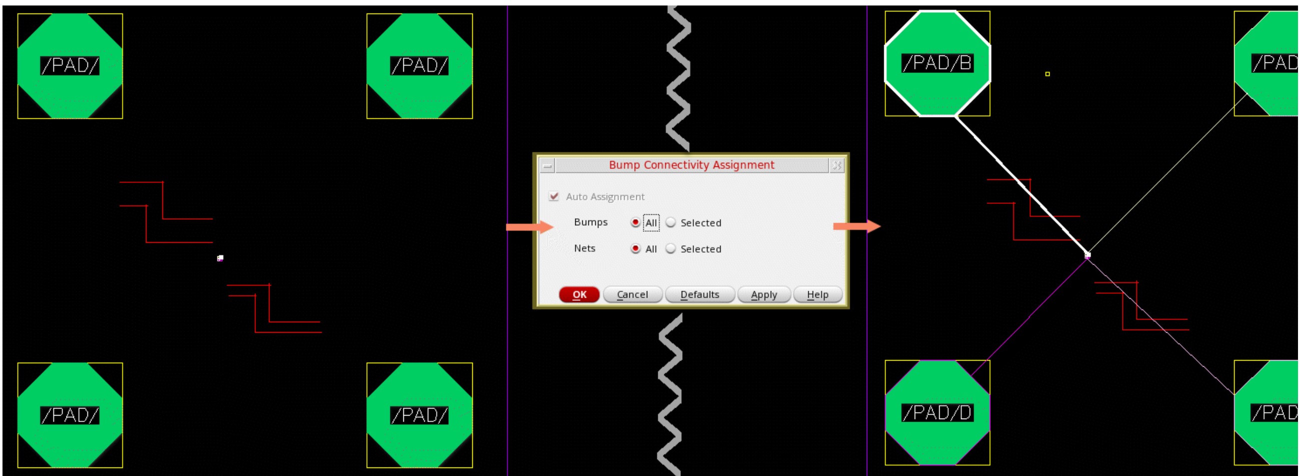

4. Specify the bumps for which you want to establish connectivity. The available options are:
   - *All*: All bumps in the active die are assigned to their closest nets.
   - *Selected*: Select the required bumps in the layout canvas.

5. Specify the nets to which the bumps are to be connected. The available options are:
   - *All*: All nets in the current design are considered.
   - *Selected*: Select the required nets in either the layout canvas or the Navigator assistant. The net is connected to the closest bump.

6. Click *OK*.


© 2023 Cadence Design Systems, Inc.

Bumps are connected to nets as per your specifications.




### *Related Topics*

- Bump Connectivity Assignment Form (form reference)
- Creating Bumps and TSVs
- Unassigning Bump Connectivity


© 2023 Cadence Design Systems, Inc.

# Unassigning Bump Connectivity

At any point, for example, when the automatic assignments do not match your requirements, can choose to unassign connectivity for one or more bumps. You can then manually assign connectivity for these bumps.

To unassign connectivity for bumps:

1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In-Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.
2. Click the tab that contains the required bumps.
3. Choose *Module – Bump Management – Define Bump Connectivity – Unassign Bump Connectivity*.

Connectivity for the specified bumps is unassigned.

***Related Topics***

- Creating Bumps and TSVs
- Assigning Connectivity between Bumps


© 2023 Cadence Design Systems, Inc.

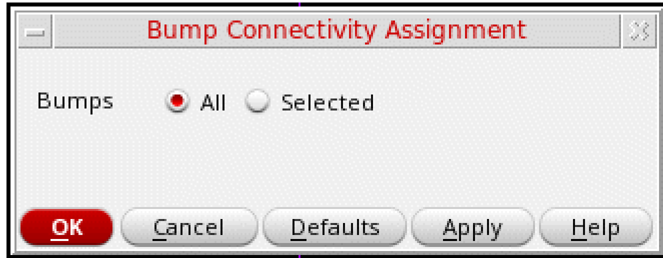
# Deleting Unassigned Bumps

You can delete from the design all bumps that are not connected to any nets. To delete unassigned bumps:

1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In-Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.

2. Click the tab that contains the unassigned bumps.
3. Choose *Module – Bump Management – Define Bump Connectivity – Delete Unassigned Bumps*. The Bump Connectivity Assignment form is displayed.




4. Select *All* or *Selected* to specify the bumps.
5. Click *OK*.

Connectivity for the specified bumps is unassigned.

### *Related Topics*

- Bump Connectivity Assignment Form (form reference)
- Assigning Connectivity between Bumps
- Unassigning Bump Connectivity


© 2023 Cadence Design Systems, Inc.

# Moving Pins to Bumps

Typically, when a layout design is generated using either *Floorplan – Generate Physical Hierarchy* or *Connectivity – Generate All From Source*, all generated instances and pins are placed below the PR boundary, at the same relative positions as in the schematic.

In the Virtuoso Stacked Silicon solution, after defining connectivity for bumps and TSVs, the next step is to move pins to the corresponding bumps based on their connectivity information.

Virtuoso Stacked Silicon solution provides the following commands to move pins to bumps:

- *Module – Bump Management – Define Bump Connectivity – Move Pins to Bumps – Move*: Moves pins to the corresponding bumps.
- *Module – Bump Management – Define Bump Connectivity – Move Pins to Bumps – Relayer and Move*: Moves pins to the corresponding bumps and re-layers them to the bump layer. Choosing this option helps avoid the task of changing the layer-purpose pair (LPP) of pins to match the LPP of bump instances.


© 2023 Cadence Design Systems, Inc.

# Updating Bumps to the Abstract View

In the Edit-In-Concert mode, the bumps that you add to a die layout are not dynamically updated on the abstract views instantiated in the package. Therefore, after finalizing bumps on the die layout, update the die representation in the package die. The die information is updated in the associated die abstract, die footprint, and TILP instance.

To update bumps to the corresponding abstract view:

1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In-Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.

2. Click the tab that contains the required bumps.

3. Choose *Module – Bump Management – Update Bumps to Abstract.*

The bumps are now propagated to the abstract and are visible and selectable in the package layout in the Edit-In-Concert mode.


© 2023 Cadence Design Systems, Inc.

# Saving Bumps to File

Virtuoso Stacked Silicon solution provides options to save bumps to a file. The file contains detailed information about the bumps, including their locations and connectivity information.

You can save bumps in the current die to a text file. You can later reuse this file to create similar bumps on other dies.

To save bumps to a file:

1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In_Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.

2. Click the tab that contains the bumps.

3. Choose *Module – Bump Management – Save Bumps to File*. The Export Bump Info form is displayed.

4. Specify a file name and the location in the *Bump File Path* field. You can click *Browse* to select a location.

5. Click *OK*.

### *Related Topics*

- Export Bump Info Form (form reference)


© 2023 Cadence Design Systems, Inc.

# Creating Bumps from File

Virtuoso Stacked Silicon solution supports the creation of reusable bump files. You can use these bump files, which are in text format, to generate similar bumps in different dies. Bump files follow a specific format. The following image displays a sample bump file.

```
DiePad1 LibInterposer/BUMPCELL/layout
PinNumber      PadStack       XCoord       YCoord       Rotation       PinUse       NetName
frontBump_4   DiePad1   120.000000   120.000000   0 BI   A
frontBump_3   DiePad1   20.000000   120.000000   0 BI   B
frontBump_2   DiePad1   120.000000   20.000000   0 BI   C
frontBump_1   DiePad1   20.000000   20.000000   0 BI   D
```

To create bumps from a bump file:

1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In-Concert.*

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.

2. Click the tab on which bumps are to be generated.

3. Choose *Module – Bump Management – Create Bumps – From File*. The Import Bump Info form is displayed.

4. Specify the path to the bump file in the *Bump File Path* field or click *Browse* to select the file.

5. Select *Create TSV* to create a TSV.

6. Select a via from the *Via Definition* cyclic field, which lists all the standard and custom vias that are defined in the technology file.

   **Note:** The via definitions are listed in the order in which they are defined in the `validVias` list in the technology file.

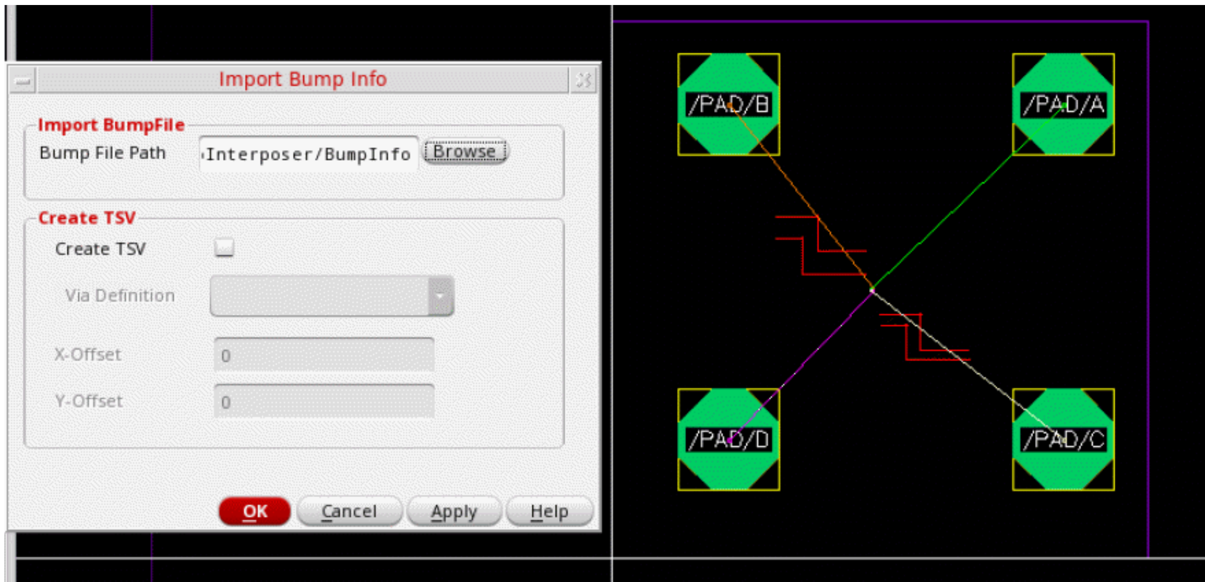
7. Specify the X and Y offsets for the TSVs. These values define the offset from the center of each TSV to the center of the corresponding bump.

8. Click *OK*.

Bumps and TSVs as per your specifications are created.


© 2023 Cadence Design Systems, Inc.

**Note:** If the net names are listed in the bump file, the created bumps are also assigned appropriate connectivity.




### *Related Topics*

- Import Bump Info Form (form reference)

# Inter-Die Operations

After you finalize the bumps for a die layout, switch to the package or container tab in the Edit-In-Concert mode to run the following inter-die commands:

- Propagating Bumps
- Fixing Bump Alignment Violations

### *Related Topics*

Creating Bumps and TSVs

Fixing Bump Alignment Violations


© 2023 Cadence Design Systems, Inc.

# Propagating Bumps

Propagating bumps involves transferring information about bump locations and assignments from a source die to a target die. These are the two dies to be vertically stacked. Propagating bumps helps create aligned bumps in the two stacked dies.

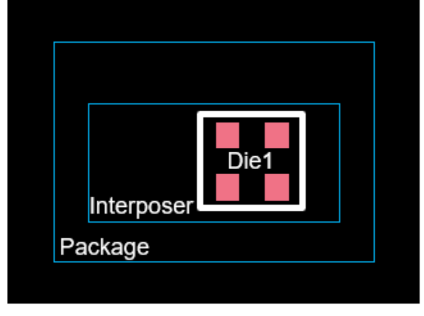
In the following example, die1 is the source die from which bumps are propagated to the target die, an interposer. In the final design, die1 is stacked on the interposer.

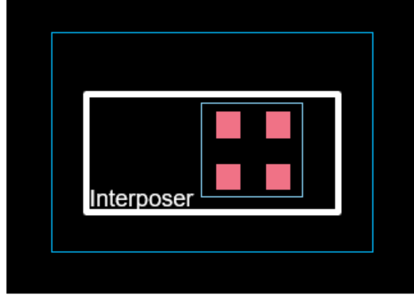
**Bumps are Propagated from Source to Target Die**

The bumps in the target die are aligned to the bumps in the source die and are assigned the same connectivity based on the top level.

**Note:** If the target die is a TILP, then in addition to the TILP, the bumps are automatically propagated to the corresponding die layout.

In the following example, the `Die1` TILP is the source and the `Interposer` TILP is the target. All bumps are propagated from the source to the target and then to the interposer layout.




Propagation of bumps is an inter-die operation. Before you launch this command, move to the package or container tab in the Edit-In-Concert mode. Ensure that the following prerequisites are met:

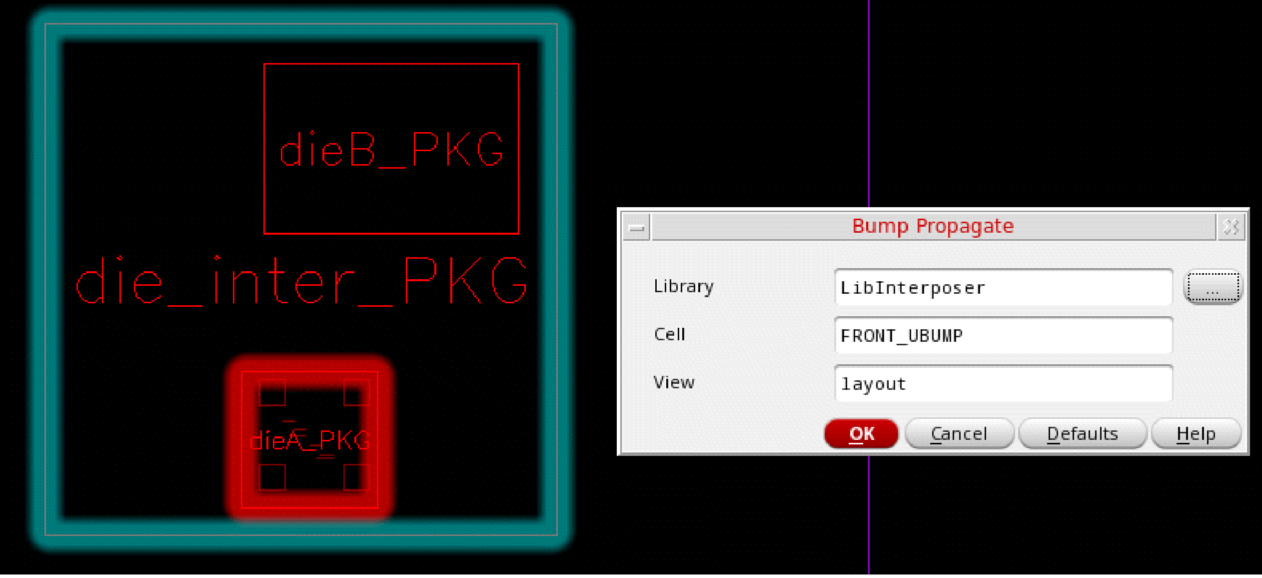

© 2023 Cadence Design Systems, Inc.

- The selected source and target instances are directly stacked on each other. If not, use the Configure Module Stack form to update the ordering of the dies.
- All the source bumps are entirely within the boundary of the target design.

To propagate bumps:

1. With the container or package layout open, enable Edit-In-Concert mode by selecting *Module – Edit-In-Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.
2. Click the container or package tab.
3. Choose *Module – Bump Management – Propagate Bumps.*
4. Select the source die instance.
5. Select the target die instance. The Bump Propagate form is displayed.




6. Specify the *Library*, *Cell*, and *View* of the bump cellview to be used to create bumps in the target instance.

All bumps are propagated from the source to the target cellview. The connectivity of the propagated bumps are automatically assigned as per the source. For bumps that do not have any connectivity in the source, no connectivity assignments are done in the target.

If the target is a TILP, in addition to the TILP, the bumps are automatically propagated to corresponding layout.


© 2023 Cadence Design Systems, Inc.

### *Related Topics*

- Bump Propagate Form (form reference)


© 2023 Cadence Design Systems, Inc.

# Fixing Bump Alignment Violations

The proper alignment of bumps in the source and target instances is necessary for proper routing. Bump propagation creates bumps in the target instance that are aligned with the bumps in the source instance. However, when making manual adjustments to the design, you might accidentally move a bump. The final task in the bump management flow is to verify the alignment of bumps in dies that are stacked one over the other and fix alignment issues, if any.

To fix bump alignment violations:

1. With the container or package layout open, launch the Edit-In-Concert mode by selecting *Module – Edit-In-Concert*.

   The package design is displayed on the first tab, and the layouts of the die instances in the package are displayed on separate tabs.

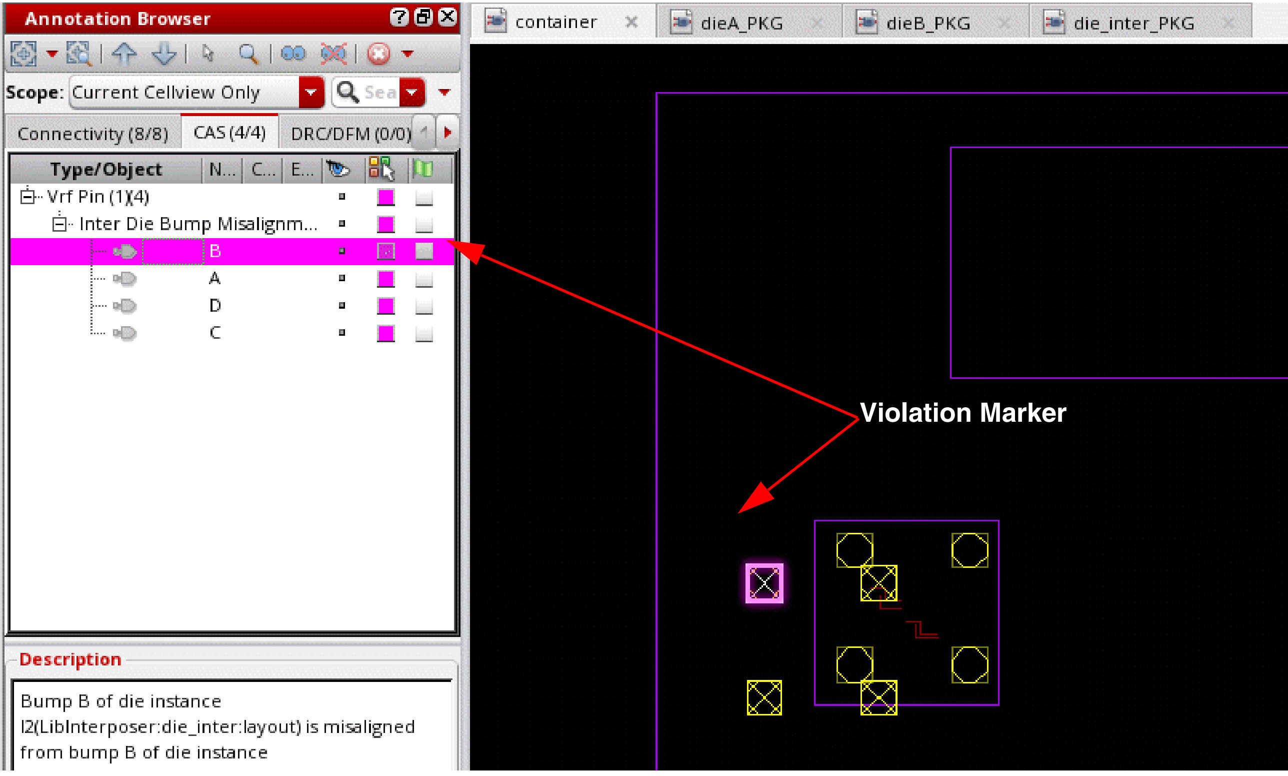
2. Click the container or package tab.
3. Choose *Module – Bump Management – Check Bump Alignment.*
4. Select the source cellview.
5. Select the destination cellview.

   Bump alignment mismatches are flagged with violation markers on the *CAS* tab of the Annotation Browser assistant.

   **Note:** To access the Annotation Browser assistant, choose *Window – Assistants –*


© 2023 Cadence Design Systems, Inc.

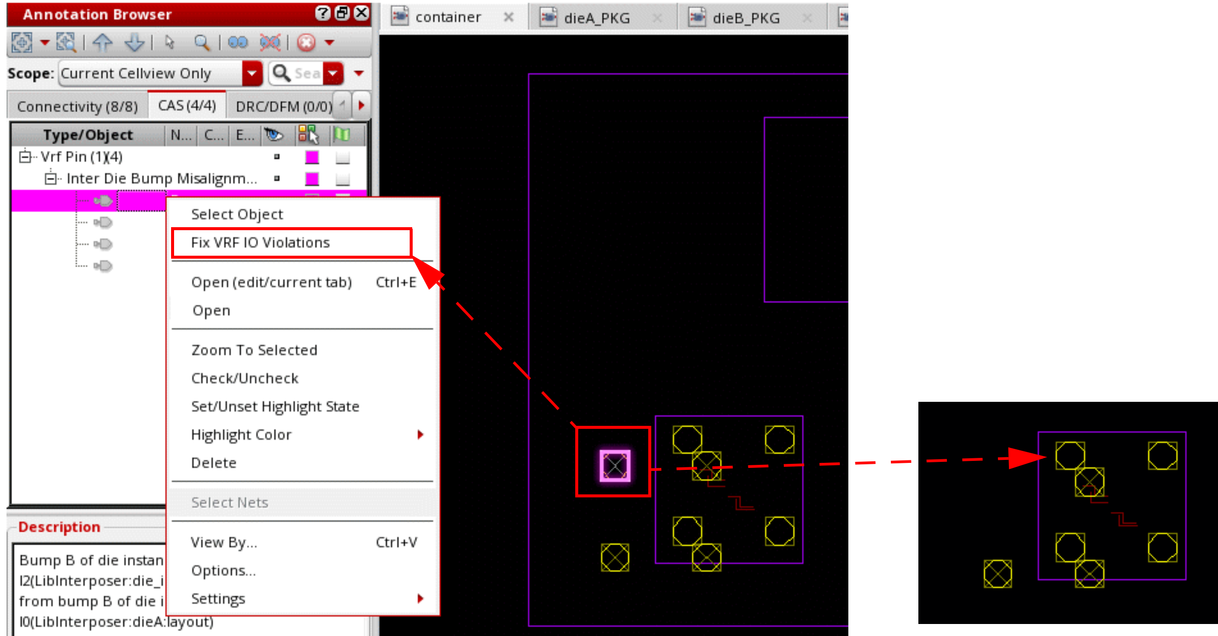
*Annotation Browser*.




**6.** Click each marker in the Annotation Browser assistant to view the location and a short description of the violation.

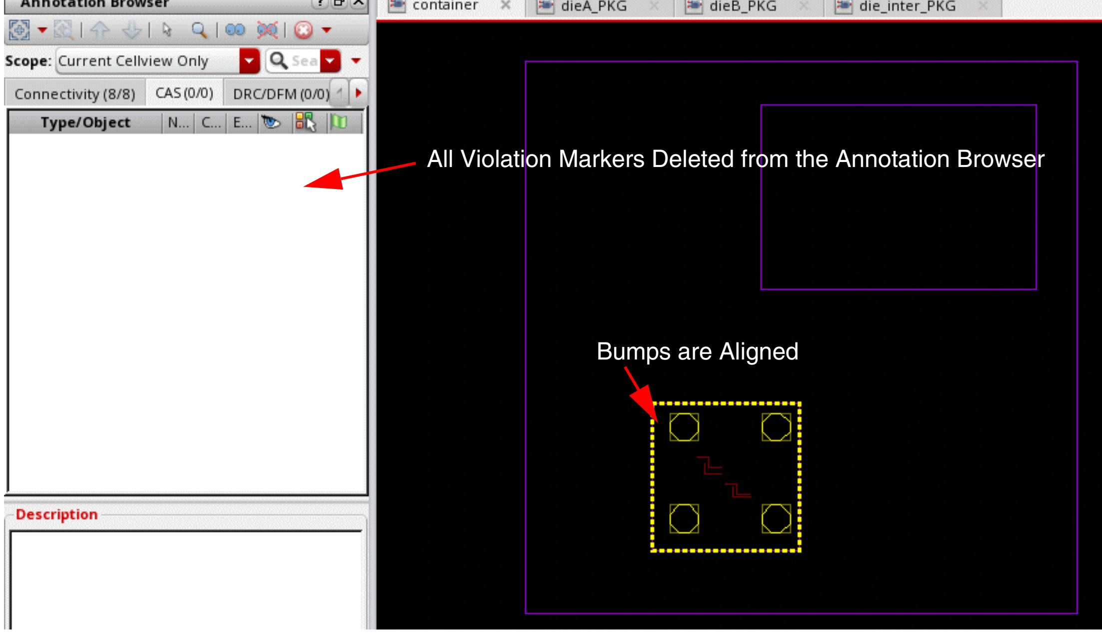

© 2023 Cadence Design Systems, Inc.

7. To fix a violation, right-click the violation marker and choose *Fix VRF IO Violations*.




8. To fix all violations, choose *Module – Bump Management – Fix Bump Alignment.*

The bumps in the target instance are fixed as per their counterparts in the source instance.




The results are reported in the CIW.


© 2023 Cadence Design Systems, Inc.

### *Related Topics*

- Propagating Bumps
- Creating Bumps from File


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# 10

# SiP Layout Creation

The development of any design involves an iterative process of synchronizing the differences between the schematic and the SiP layout. Changes, especially caused by Engineering Change Orders (ECOs), are made in the package schematic and need to be updated in the SiP layout. Similarly, changes in the SiP layout, such as reference designator changes and section and pin swaps, require updating the package schematic.

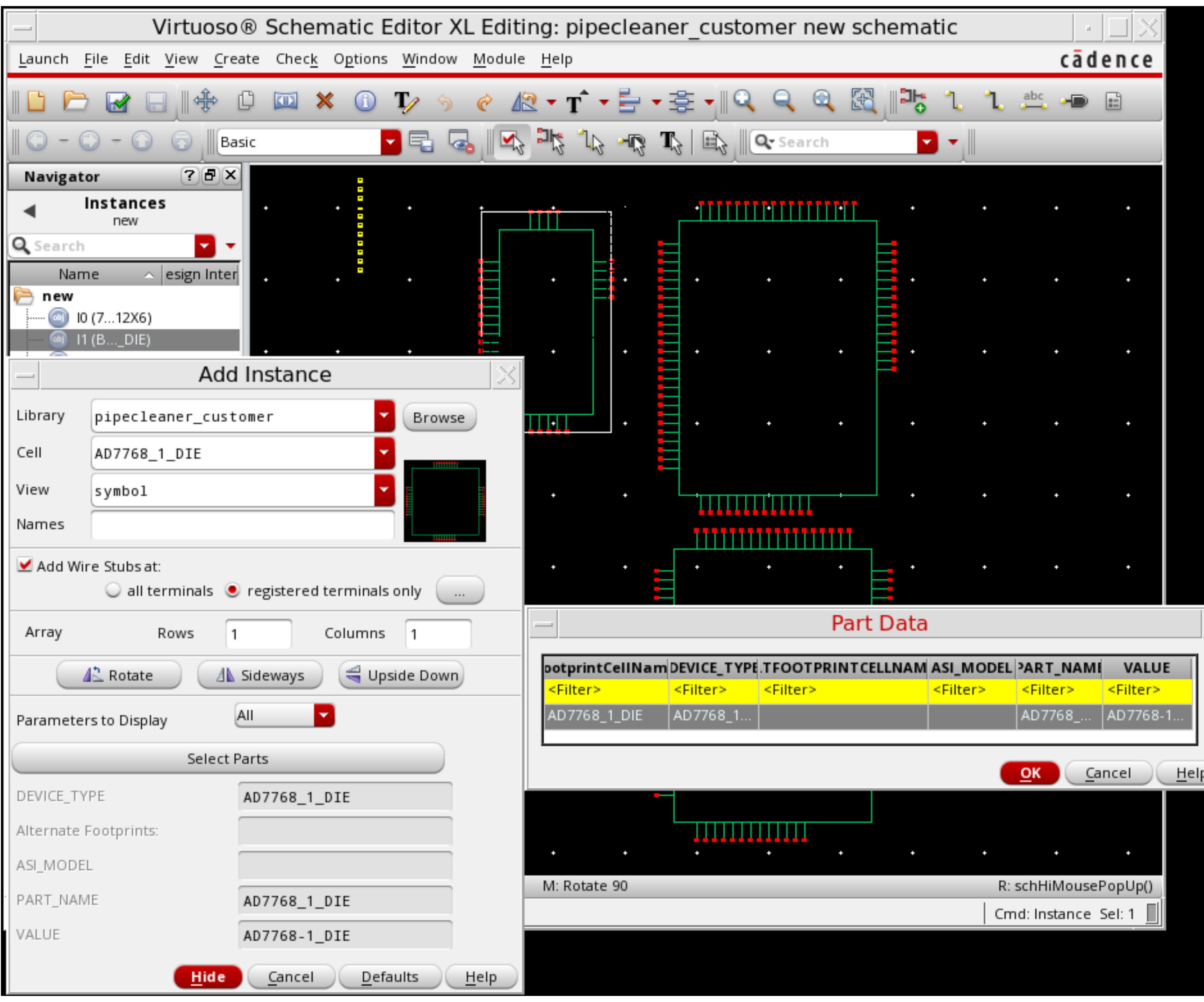
## Creating a SiP Layout from a Package Schematic

Before, you create a SiP layout, ensure that the IC, PCB, and Sigrity hierarchy paths have been set for a seamless flow of tasks in the Virtuoso Multi-Technology Solution environment.

To create the SiP layout:


© 2023 Cadence Design Systems, Inc.

1. Create a package schematic by selecting parts and adding instances in the canvas.

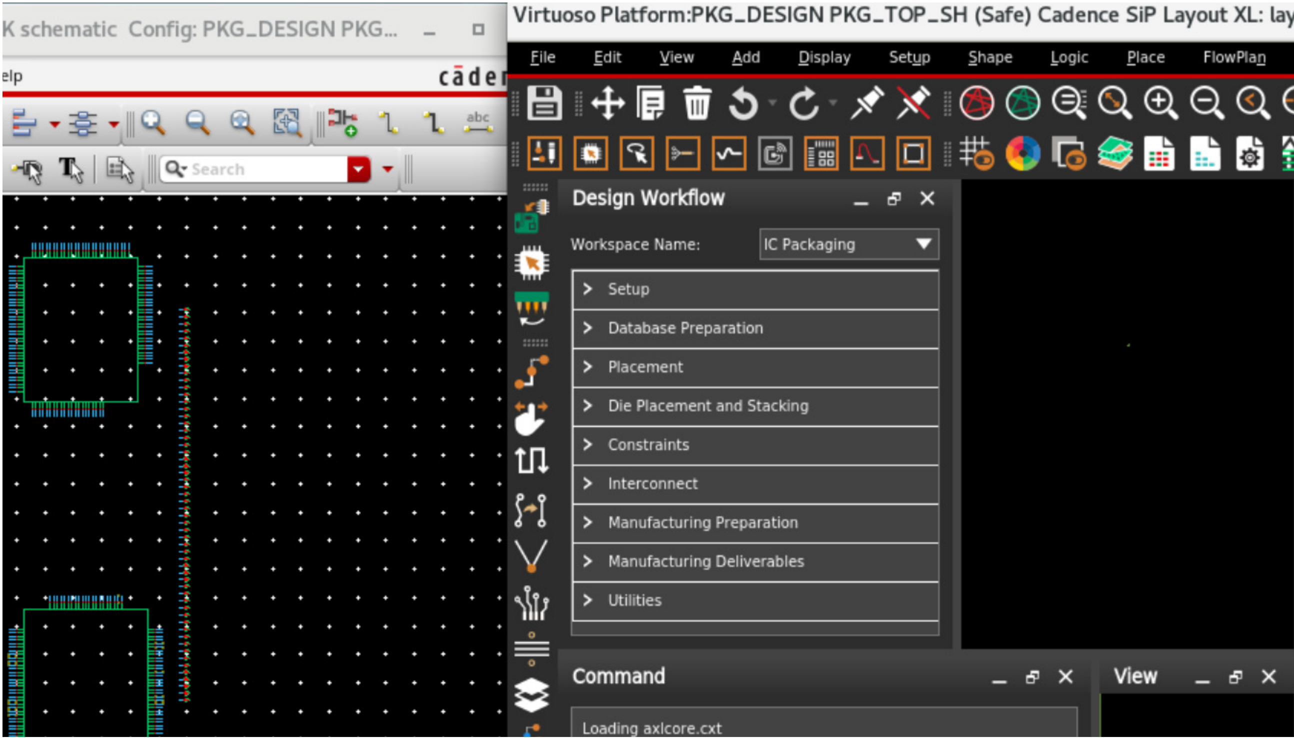



2. Choose *Launch – Layout SiP*.

3. Create a new layout by using the New File form. The Cadence SiP Layout opens.


© 2023 Cadence Design Systems, Inc.

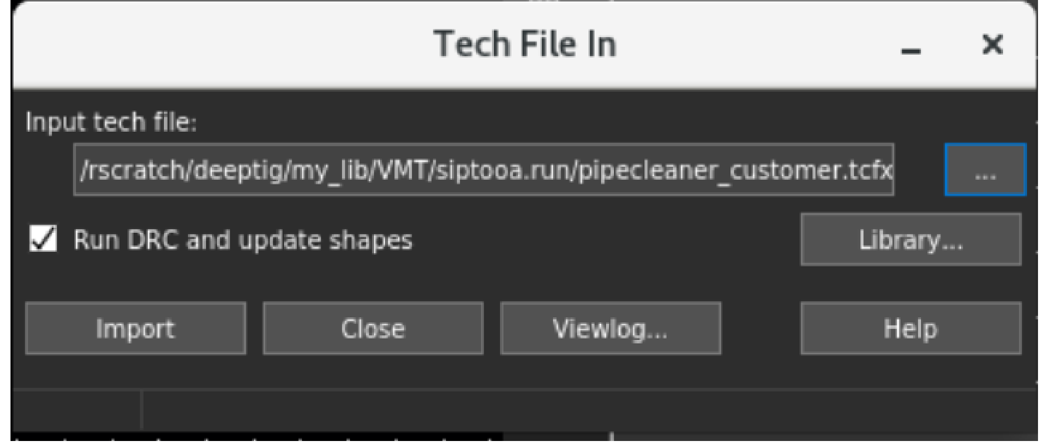
4. Arrange Cadence Virtuoso Schematic Editor and Cadence SiP Layout windows alongside.




5. Import the technology and parameter files.

   a. Choose *File – Import – Techfile*.

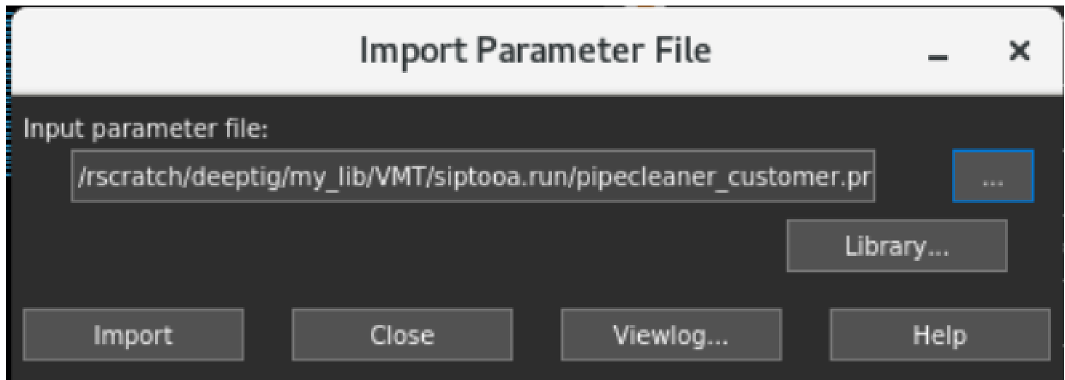
   b. Specify the file in the *Input tech file* field of the Tech File In form.




   c. Choose *File – Import – Parameters*.


© 2023 Cadence Design Systems, Inc.

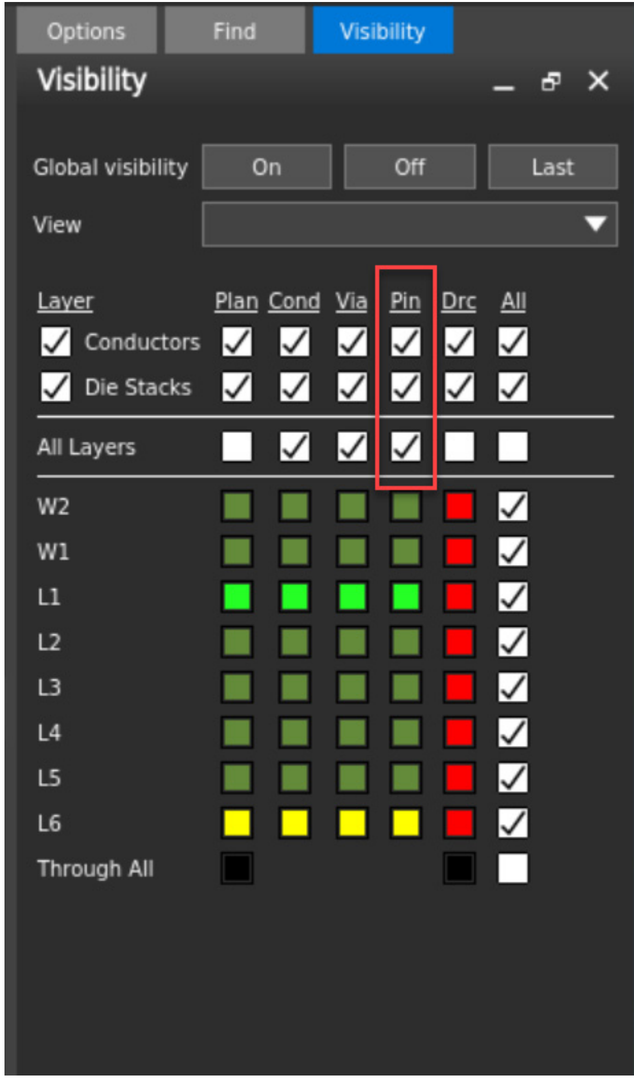
**d.** Specify the file in the *Input parameter file* field the Import Parameter File form.





© 2023 Cadence Design Systems, Inc.

6. In the *Visibility* tab on the canvas, enable *Pin* for *Layer Conductors*. A blank layout with complete technology information is created.




### *Related Topics*

Generating a SiP Layout from a Source Schematic

Checking Against Source Schematic

Creating an Extracted View

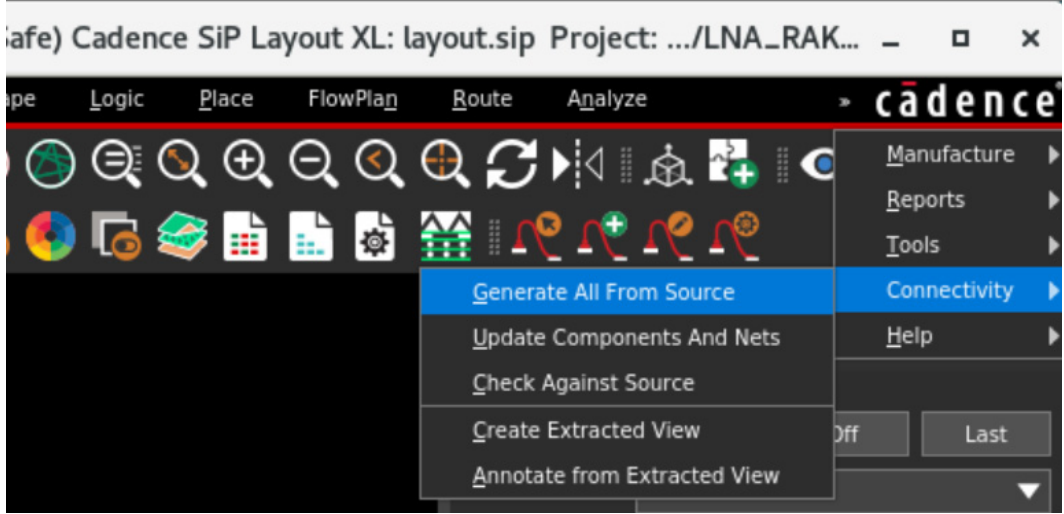

© 2023 Cadence Design Systems, Inc.

# Generating a SiP Layout from a Source Schematic

This topic lists the steps to generate a SiP layout from a package schematic in Virtuoso. You can use the *Connectivity* menu available in the SiP layout to create the physical layout in SiP. The capabilities include cross-selecting components in the schematic and highlighting them in the layout and vice versa.

To generate from source:

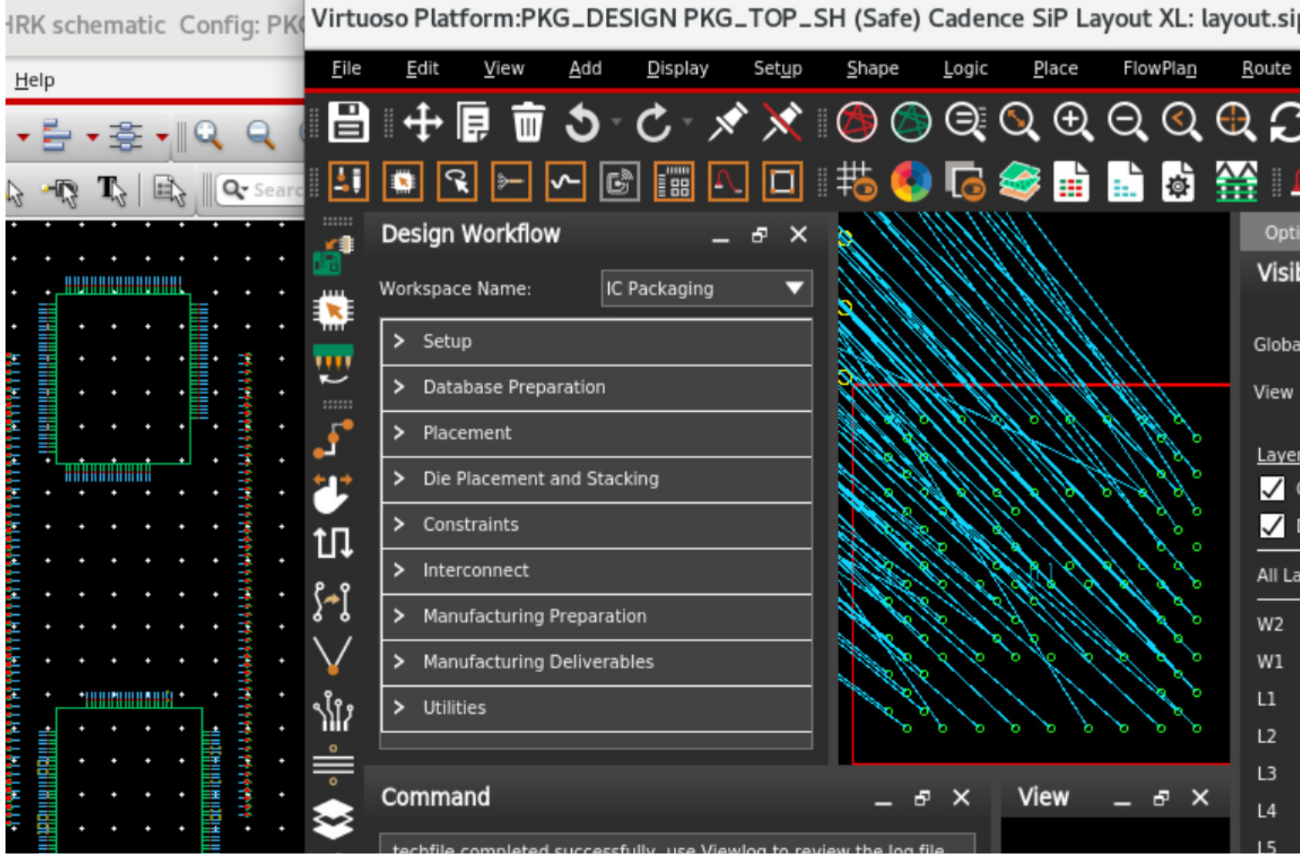
1. Choose *Connectivity – Generate All from Source*.





© 2023 Cadence Design Systems, Inc.

2. Click various instances in Virtuoso Schematic Editor and see how the corresponding symbol in Cadence SiP Layout is zoomed-in, selected, and highlighted.




**Note:** You can use the Scribe Lines feature that shows the physical extents of the actual manufactured die. This includes the scribe area outside the design extents that is part of the wafer scribe or sawing process.

### *Related Topics*

[Scribe Lines Feature]

[Creating a SiP Layout from a Package Schematic]

[Checking Against Source Schematic]

[Creating an Extracted View]


© 2023 Cadence Design Systems, Inc.

# Checking Against Source Schematic

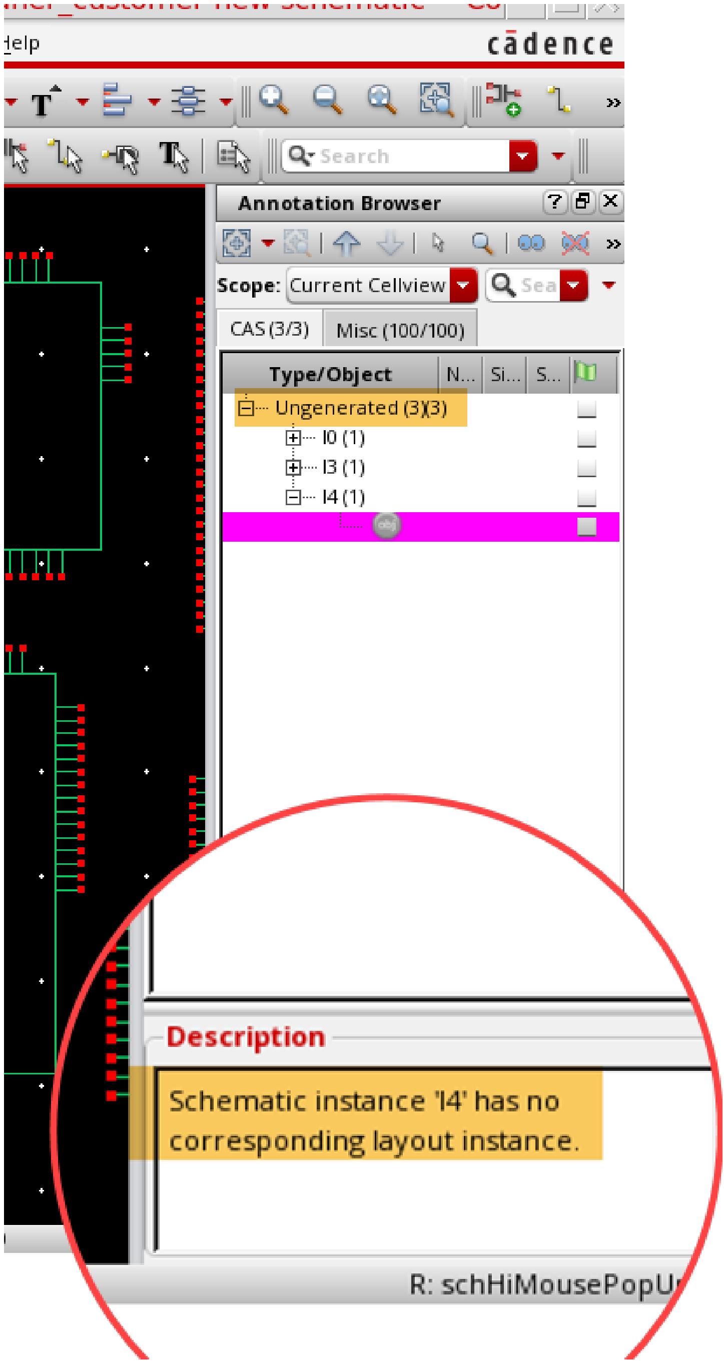
To check against the source schematic:

1. Click a blank portion of canvas to undo any selection in Virtuoso Schematic Editor.
2. Choose *Edit – Copy*.
3. In the canvas, select an instance and copy.
4. Choose *Connectivity – Check Against Source* in the SiP layout.

   The Annotation Browser opens in the Virtuoso Schematic Editor.


© 2023 Cadence Design Systems, Inc.

5. In the Annotation Browser, expand the *Ungenerated* list. Note the *Description* field.

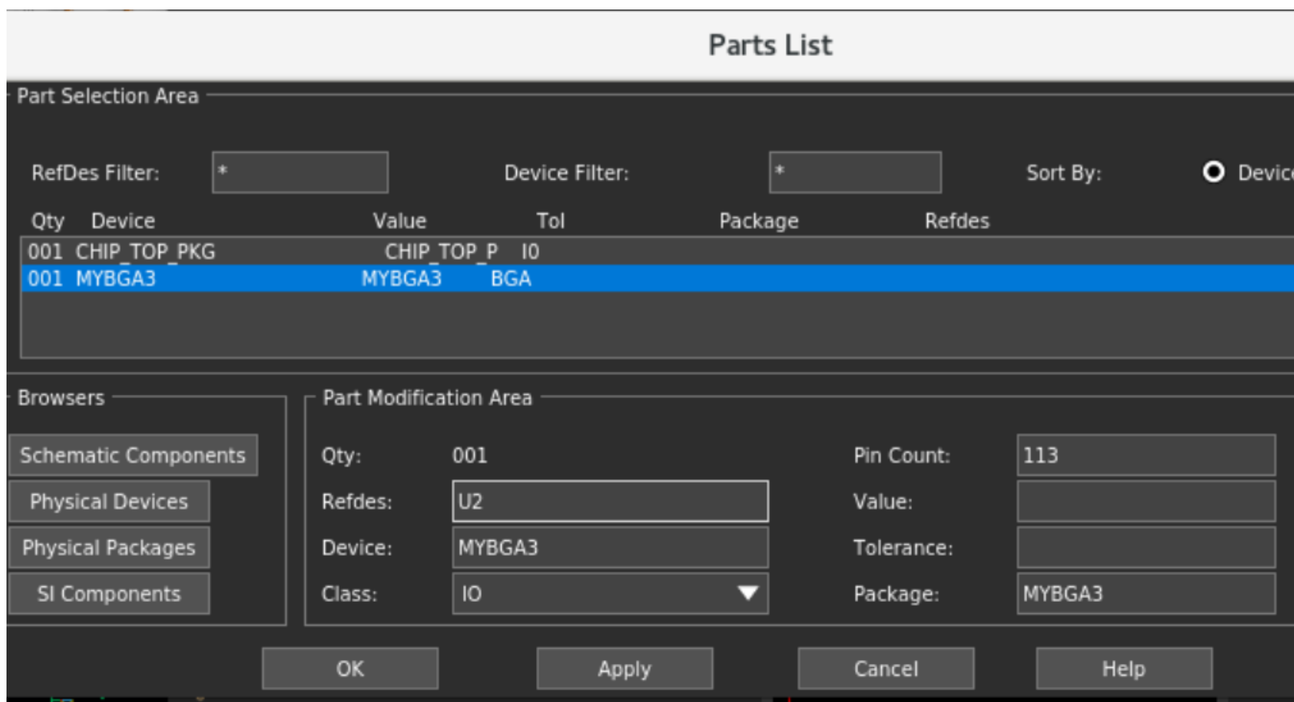



6. Click *Connectivity – Update Components and Nets* in the SiP layout. The Annotation browser removes the mismatched instances.

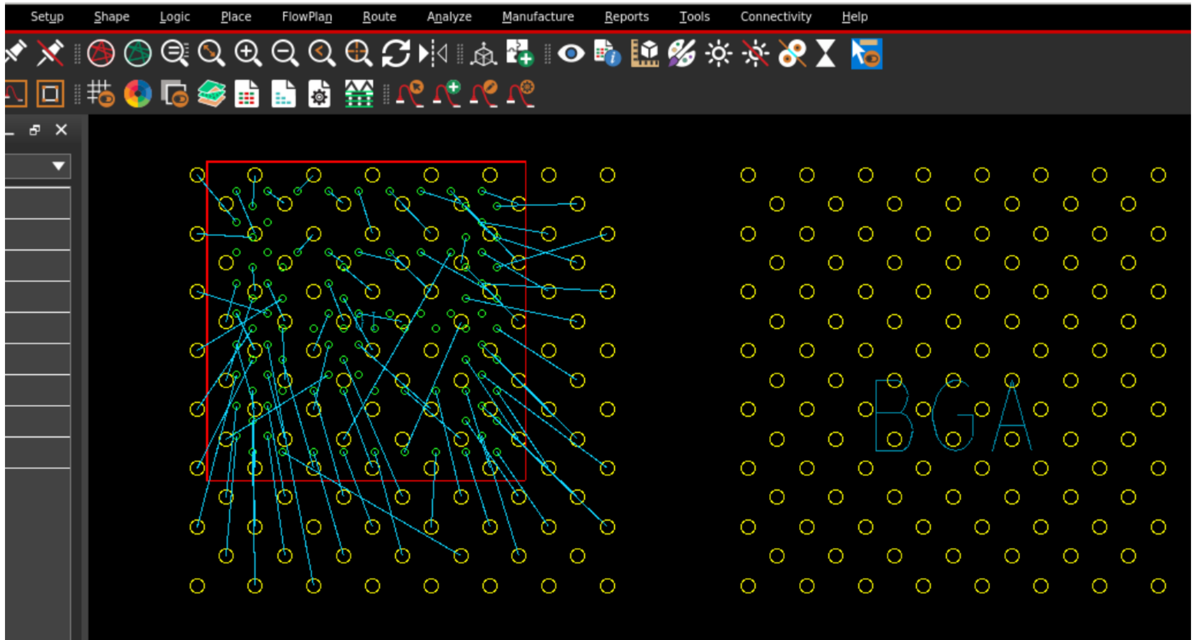
7. Choose *Logic – Edit Parts List* in the SiP layout.


© 2023 Cadence Design Systems, Inc.

8. In the Parts List form, click a device and update its `Refdes` value to `U2`.



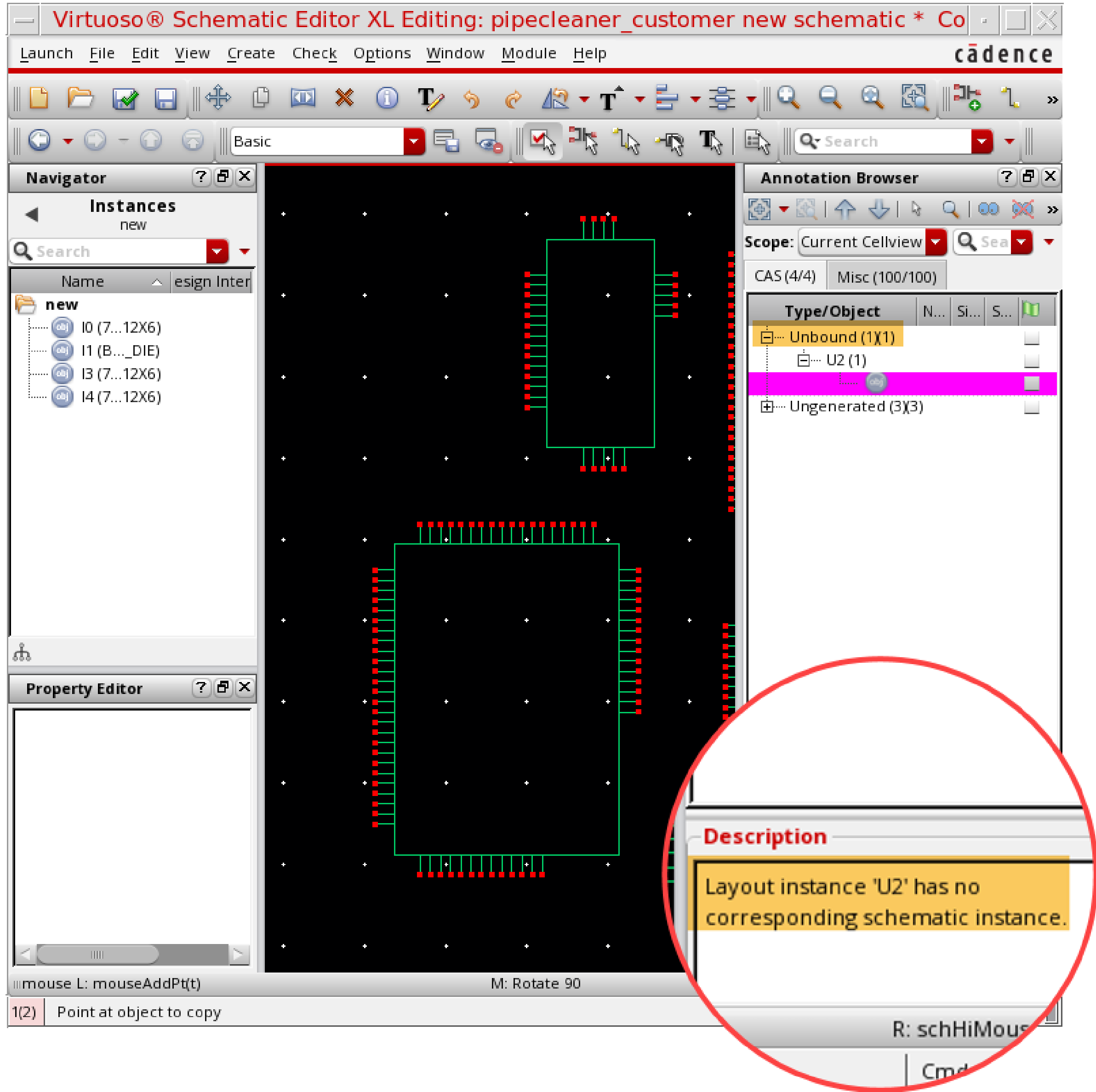

9. Click *Place – Manually* and place `U2`.




10. Click *Connectivity – Check Against Source* in the SiP layout. The Annotation Browser opens in the Virtuoso Schematic Editor.


© 2023 Cadence Design Systems, Inc.

**11.** In the Annotation Browser, expand the *Unbound* list. Note the *Description* field. `U2` is not recognized in the corresponding schematic and an error is reported.




## *Related Topics*

Creating a SiP Layout from a Package Schematic

Generating a SiP Layout from a Source Schematic

Creating an Extracted View

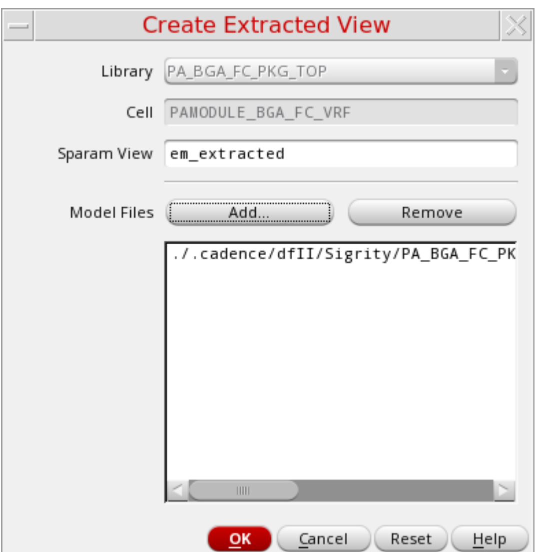

© 2023 Cadence Design Systems, Inc.

# Creating an Extracted View

To create an extracted view of a package schematic and backannotate parasitic models to the golden schematic:

1. Choose *Connectivity – Create Extracted View* in SiP Layout Option to create an extracted view from selected model files.

   The Create Extracted View form opens.



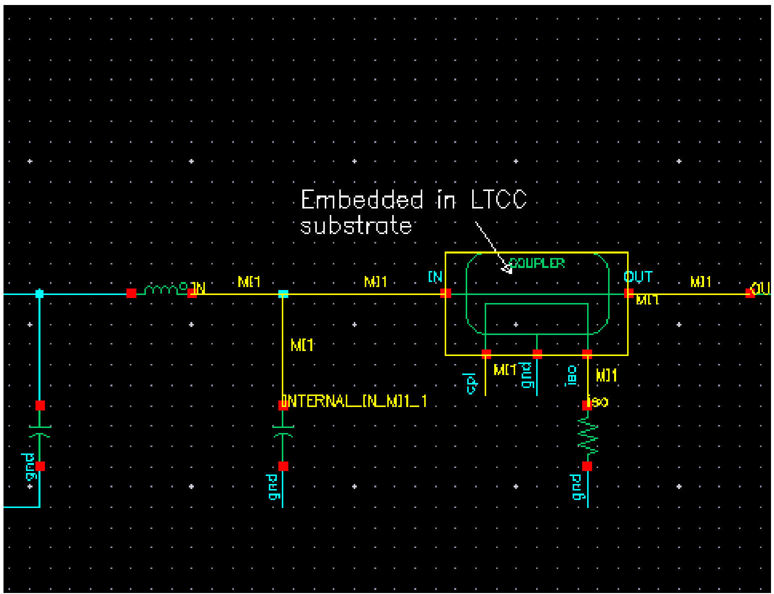

*Important*

Only SiP files generated using the latest Virtuoso Multi-Technology Solution and model files created using Sigrity 2019 or later can be used to create an extracted view.

2. Add the model files in the *Model Files* field and click *OK*. An extracted cellview contains an n-port instance where the n-port instance refers to an S-parameter file. This S-parameter file is stored inside the extracted cellview folder. This ensures that extracted view and S-parameter file are always synchronized.


© 2023 Cadence Design Systems, Inc.

3. Choose *Connectivity – Annotate from Extracted View* to annotate the parasitic models from the extracted view on the master schematic. The Annotate From Extracted View form opens, with models highlighted in the master schematic as shown below.




For IC schematics, you can also use Quantus QRC-based Smart Views to create extracted views after EM simulation.

***Related Topics***

Create Extracted View Form

Annotate From Extracted View Form

Running a Clarity Simulation

Creating an Extracted View from Smart View

Creating Extracted Views from Models


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# 11

# Schematic Creation from SiP File

Creating a multi-technology schematic is an enablement step. It creates a schematic view based on the connectivity information defined in a SiP file. This schematic enables the connectivity-driven flow in the Virtuoso RF Solution and it allows running EM extraction and simulation. The inputs are a SiP file, a unified library containing the components, and the mapping information.

***Related Topics***

Creating a Schematic Layout from a SiP File

Create MultiTech Schematic Form

SiP Layout Option to Virtuoso Schematic Editor Flow

## Creating a Schematic Layout from a SiP File

The *Create MultiTech Schematic* option in the Virtuoso Schematic Editor can be used to import the instances and the connectivity information from a SiP file to create the initial Virtuoso schematic view. Before you create a schematic, ensure that the IC, PCB, and Sigrity hierarchy paths have been set for a seamless flow of tasks in the Virtuoso Multi-Technology Solution environment.

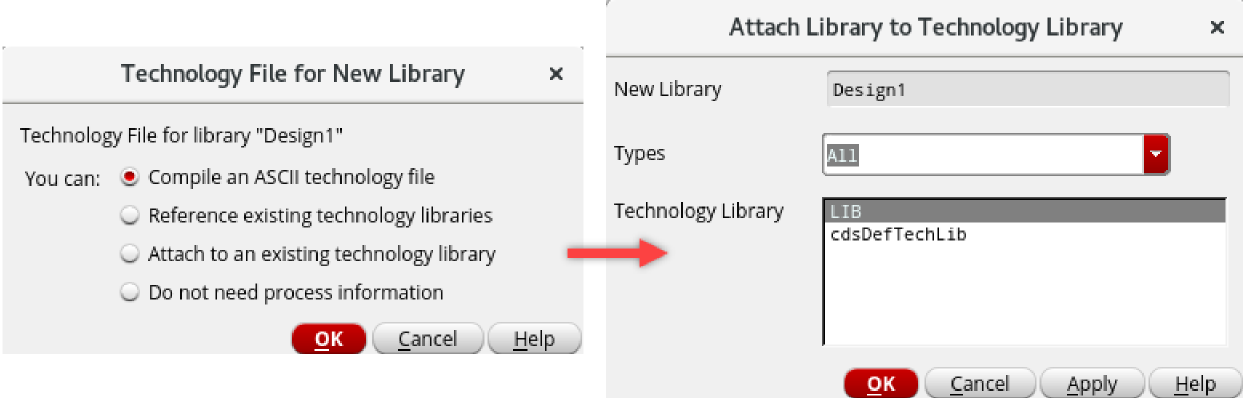
To create a schematic from a SiP file:

1. Run vmtLibImport to create the technology and cell library.

```
vmtLibImport "<SiP file>" "<library>" ?importTechOnly t
vmtLibImport "<SiP file>" "<library>" ?importDra t ?createSymbols t
```


© 2023 Cadence Design Systems, Inc.

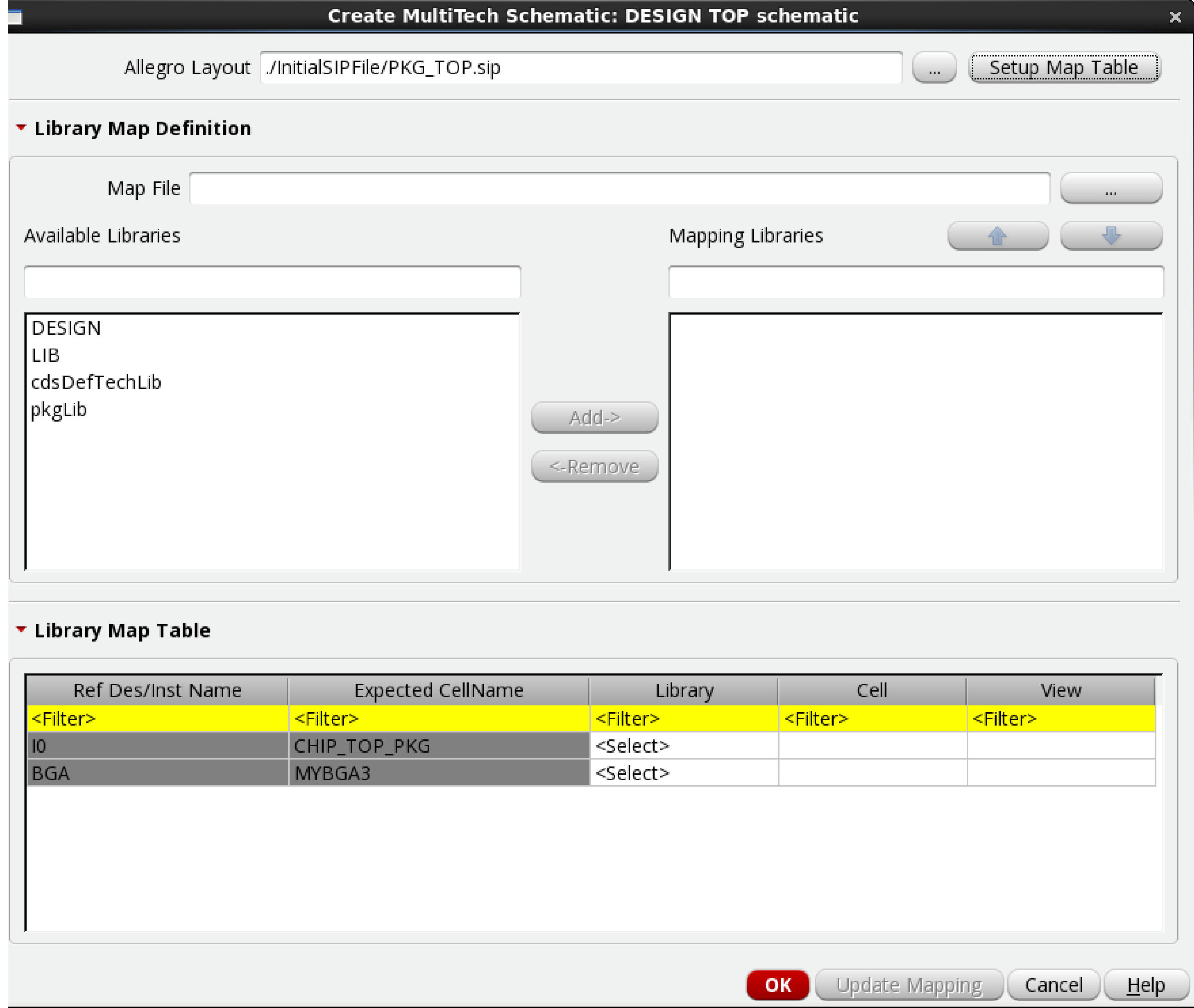
2. Create a new unified library and attach it to an existing library.




3. Create a schematic cellview in the new library.
4. Open the new schematic cellview.
5. Click *Module – Auto Create Pins* to create pins in the package schematic only if creating the schematic from scratch.
6. Click *Module – Create MultiTech Schematic*. The Create MultiTech Schematic Form opens.


© 2023 Cadence Design Systems, Inc.

7. Specify path to the SiP file in the *Allegro Layout* field.




8. Click *Setup Map Table*. The information from the SiP file is loaded in the *Library Map Table* section. Each instance in the SiP file is added to a separate row in the table.

9. Map libraries using one of the following methods:

    - ❑ Select the required libraries from the *Available Libraries* list and click *Add* to move them to the *Mapping Libraries* list. The sequence of libraries in the *Mapping Libraries* list defines the priority in which the libraries are used for mapping cells. Use the up and down buttons to change the sequence.

    - ❑ Fill the *Library Map Table* manually by specifying the libraries, cells, and views to be mapped.


© 2023 Cadence Design Systems, Inc.

- ❑ Specify the *Map File* in the *Library Map Definition* section. After each successful mapping update run, a mapping file is created, which can be reused in a new run. You can also use Text Editor to create a mapping file.

**10.** Click *Update Mapping*.

All instances in the SiP file are mapped to the library cells.

**11.** Click *OK*. The initial schematic is created using the instances and the connectivity from the SiP file. The top-level pins are created automatically. They are identified based on the cells that are declared as IO in a SiP file, for example, BGA.

After checking and updating the schematic, you can proceed with the implementation flows by either bringing the initial layout data from the same SiP file or by running Generate From Source in the Virtuoso or Allegro platforms.

### *Related Topics*

Create MultiTech Schematic Form

SiP Layout Option to Virtuoso Schematic Editor Flow


© 2023 Cadence Design Systems, Inc.

# 12

# Assisted Import and Export

The assisted flows let you update any change in the Virtuoso RF Solution compatible layout to the SiP file and vice versa. The changes might include adding, removing, or moving existing shapes, vias, SMDs at the package level or bump and ball changes in an existing design library in the Virtuoso RF Solution. The assisted import flow ensures that an existing golden schematic is maintained and allows reviewing the changes before importing them in the design library.

The assisted export flow is an enhanced way to export changes to a SiP file, which ensures a flawless interoperability and allows reviewing the changes made in the Virtuoso RF Solution.

***Related Topics***

Updating a Virtuoso Layout From a SiP File

Virtuoso Layout to SiP with Assisted Export

Creating a Schematic Layout from a SiP File

Cell Replacement

SiP Layout Option to Virtuoso Schematic Editor Flow

## Updating a Virtuoso Layout From a SiP File

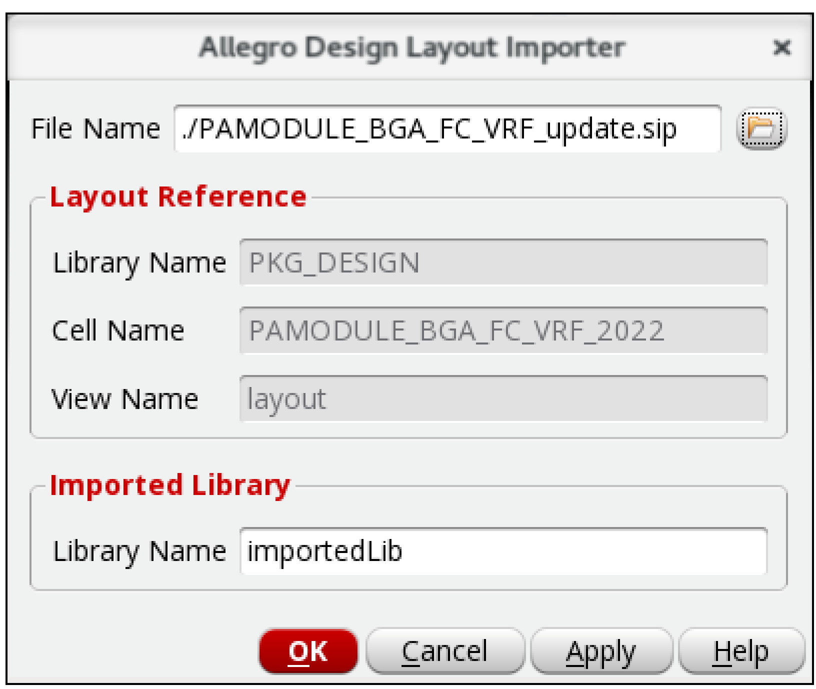
When a new version of an existing design is provided as a SiP file, it is important to compare the changes with the existing database in the Virtuoso RF Solution. The assisted import flow creates a temporary and self-contained database from this new SiP file, which allows you to run several checks for identifying the changes. Once all the modifications are reviewed and agreed, the new version of an existing design can be integrated into the reference Virtuoso RF Solution database.

To update a Virtuoso layout from a SiP file:


© 2023 Cadence Design Systems, Inc.

1. Click *Module – Assisted Flow – Import Design Update*.

   The Allegro Design Layout Importer Form opens.



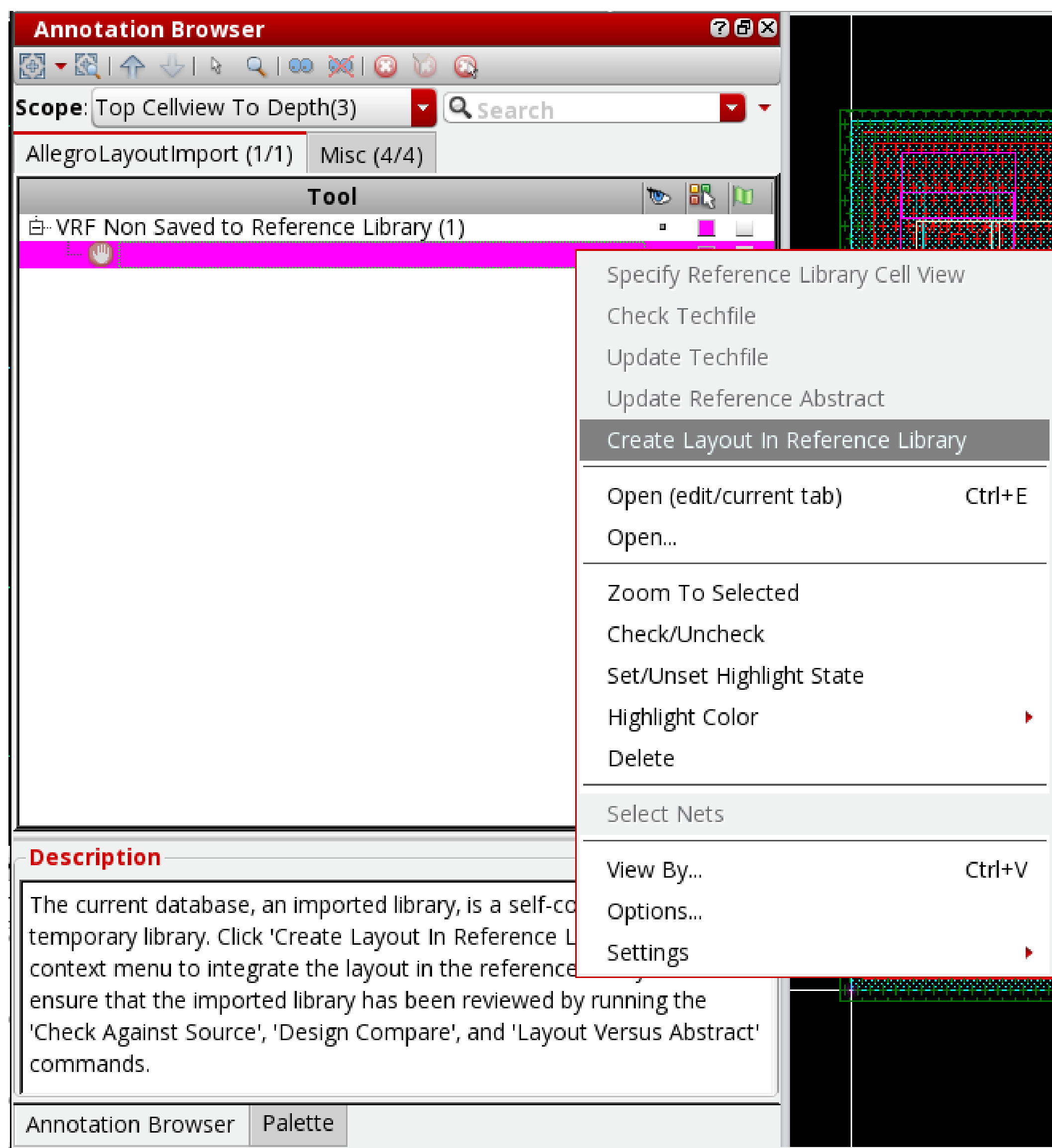

2. Specify the SiP file that needs to be synchronized with the Virtuoso package layout.
3. Specify a name for the temporary library that will be created to review the layout against the SiP file. You can perform check against source, layout versus abstract check, and other checks, such as DRC, in the imported library.
4. Click *OK*.
5. Review the modifications required in the reference database due to the updated SiP file by using the following checks:
   - Check against source to evaluate the consistency with the reference schematic.
   - Check that if abstracts are in sync with the corresponding IC layouts using *Module – Layout Vs. Abstract*.
   - Run *Module – Assisted Flow – Compare with Reference Design* to compare the placement and routing of the temporary database against the reference database. The markers are created on the *Misc* tab of the Annotation Browser.
   - Check the Annotation Browser markers on the *AllegroLayoutImport* tab. The technology file differences, such as new layers, new purposes, or new vias are displayed on this tab. Use the shortcut menu to update the technology file.


© 2023 Cadence Design Systems, Inc.

If there are new layer-purpose pairs found, the reference technology file is updated. However, if the new layer is part of the metal stack, such as `Metal2` or `Metal3`, the technology file is not updated.

- ❑ Any pin modifications, such as a bump move, are also displayed on the *AllegroLayoutImport* tab. Use the shortcut menu to update the reference abstract. To ignore these modifications, omit this step.

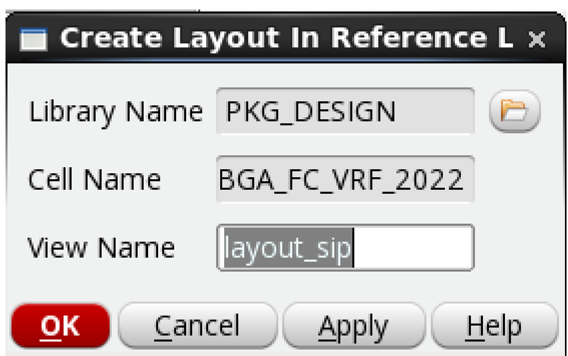
**6.** Once the design is reviewed, right-click the `VRF Non Saved to Reference Library` marker and click *Create Layout in Reference Library* on the *AllegroLayoutImport* tab.





© 2023 Cadence Design Systems, Inc.

7. Specify the new view name.




8. Click OK. The database is integrated into the reference library as a new view.
9. Close or delete the temporary library.

***Related Topics***

Virtuoso Layout to SiP with Assisted Export

SiP to Virtuoso Layout Assisted Import and Export Flows

Allegro Design Layout Importer Form

# Virtuoso Layout to SiP with Assisted Export

Assisted export ensures smooth interoperability and reduces the risk of losing data when exporting a SiP file. In addition, it allows to review the modifications made in a Virtuoso layout since the latest import. The changes are reported through Annotation Browser, which can be reviewed.

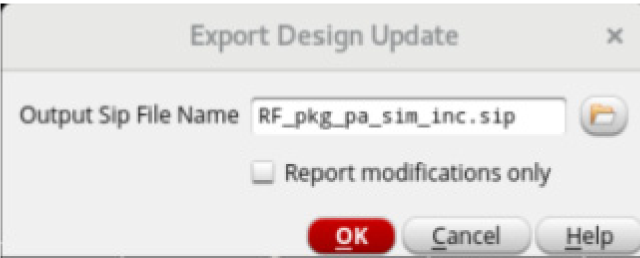
The assisted export flow can be used if you have created a Virtuoso package layout by using by using the Virtuoso Multi Technology Enablement form, assisted import, or by selecting the *Generate Instances as in Schematic* option in the Import From Allegro form.

To export a SiP file from a Virtuoso Layout using the assisted export:

1. Click *Module – Assisted Flow – Export Design Update*.


© 2023 Cadence Design Systems, Inc.

The Export Design Update Form opens.




2. Specify the name of the SiP file that will be created from a Virtuoso package layout.
3. Click *Report modifications only* to only list the design updates.
4. Click *OK*.
5. Review the changes in the Annotation Browser, if necessary. If unsupported differences are reported, it is highly recommended to check them because these updates will not be included in the generated SiP file.

***Related Topics***

Updating a Virtuoso Layout From a SiP File

SiP to Virtuoso Layout Assisted Import and Export Flows

Export Design Update Form

# Initial SiP File to Virtuoso RF Solution Database

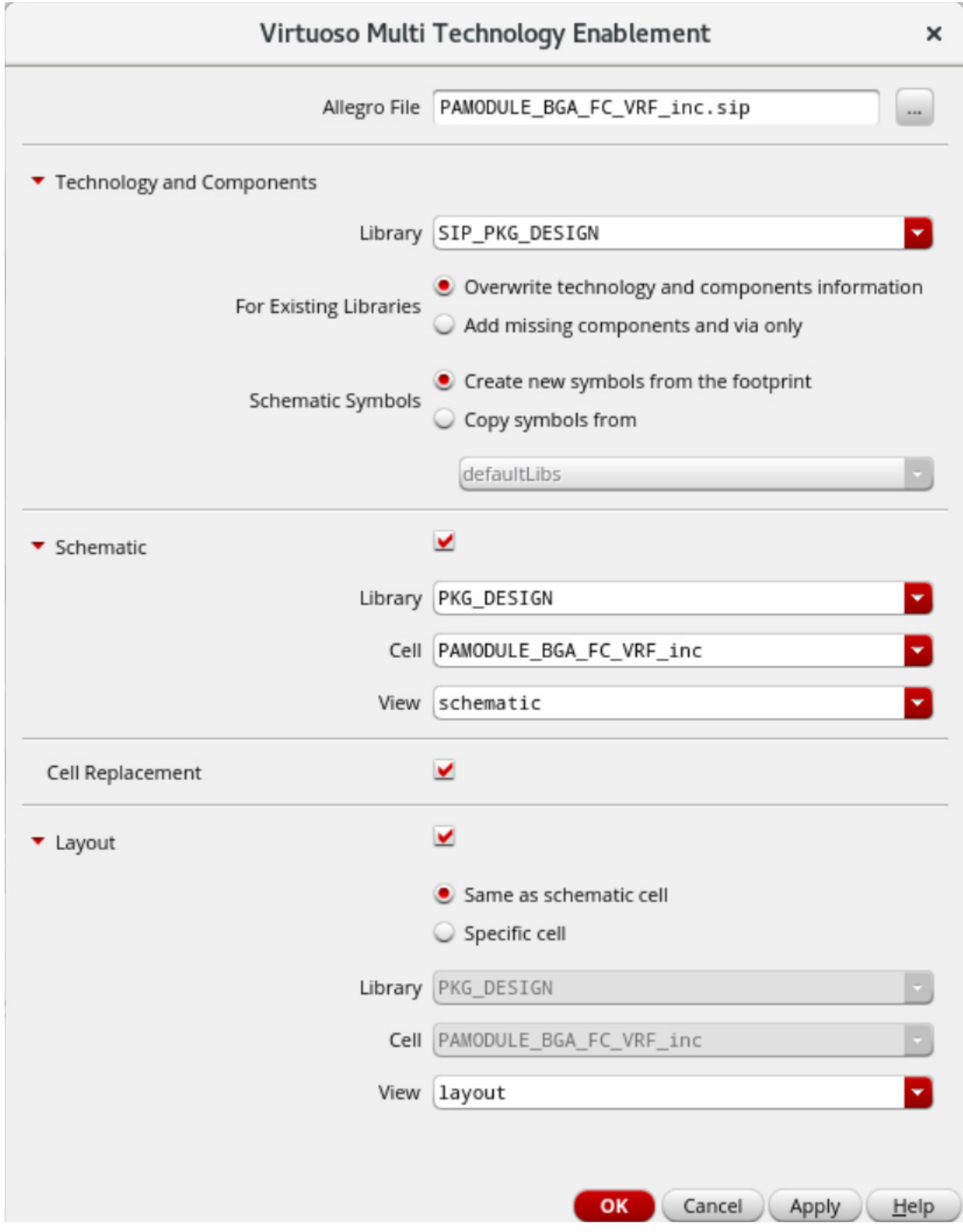
The new Virtuoso Multi Technology Enablement form simplifies the path from a SiP file to a Virtuoso RF Solution database by creating technology and component libraries, schematic, and layout in a single step. The assisted flows augment the enablement flow with verification support at various stages.

To create technology and components, schematic, and layout in one go:

1. Click *Tools – Virtuoso Multi Technology– Enablement* in CIW.


© 2023 Cadence Design Systems, Inc.

The Virtuoso Multi Technology Enablement Form opens.




2. Specify the native SiP file that will be used to create technology library, schematic, and layout.


© 2023 Cadence Design Systems, Inc.

*Caution*

***Ensure that the version of the selected SiP file is compatible with SPB17.4 or SPB22.1 versions to create the unified library successfully.***

3. Click the check box in the Cell Replacement section to replace the cells that have pin mismatches.
4. Click *OK*. A library is created. A multi-technology schematic and layout are also created if the required options are selected in the form.
5. Open the schematic and layout views to review the imported data.

***Related Topics***

Multi-Technology Enablement Flow

Virtuoso Multi Technology Enablement Form

# Cell Replacement

In the enablement flow, the die or package abstract views from the Allegro design must be replaced by the Edit-in-Concert abstract generated by the die or package export. This replacement can become manual and error-prone. Therefore, the cell replacement functionality has been integrated in the Virtuoso Multi Technology Enablement form as an optional step. It provides the capability to ease the replacement of cells having pin mismatches. For example:

- By replacing die or package abstracts by Edit-in-Concert abstracts.
- By replacing of SMD and other cells by corporate libraries generated by csvImport.

The cell replacement interface provides a global view of the replacements. It is imperative to maintain the connectivity of the replaced instances. While handling pin mismatches between the source and target abstracts, you can manually resolve the conflicts.

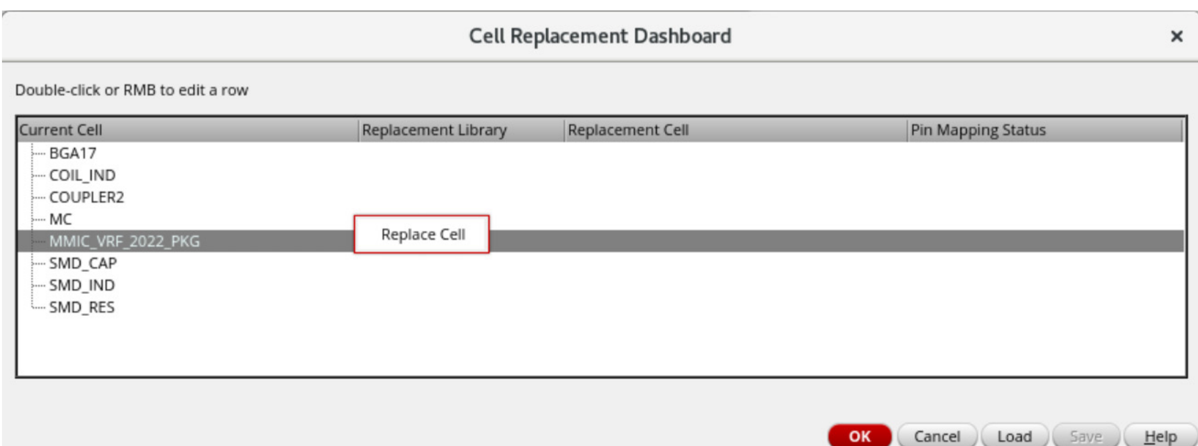
To replace cells:

1. Select the check box in the Cell Replacement section of the Virtuoso Multi Technology Enablement form.

   The Cell Replacement Dashboard form appears after schematic generation. By default, no replacement is triggered and the dashboard lists only the design cells.


© 2023 Cadence Design Systems, Inc.

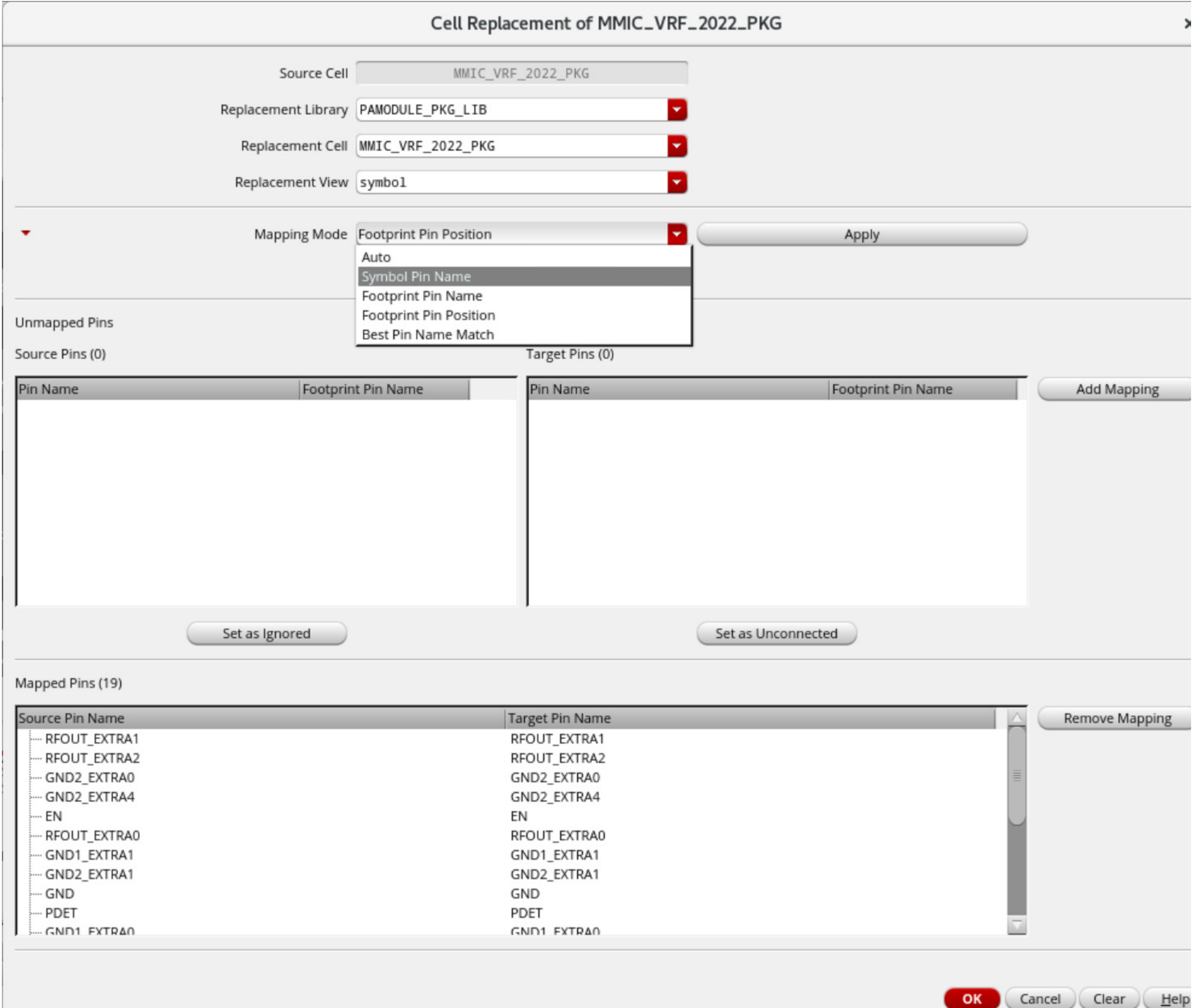
**2.** Double-click or right-click a row to replace the cell.




The Replacement Cell form opens.


© 2023 Cadence Design Systems, Inc.

3. Specify the replacement abstract symbol. The *Unmapped Pins* section gets filled automatically.




4. Select the required option from the *Mapping Mode* drop-down list and click *Apply*.

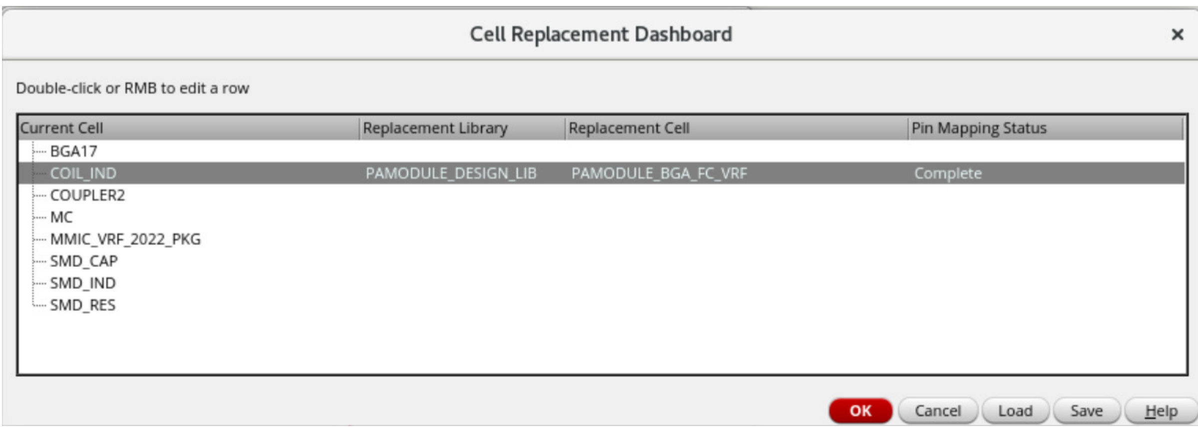
   The *Mapped Pins* section gets filled automatically.

5. [Optional] Select a source pin and a target pin to map them manually by clicking *Add Mapping*.

6. [Optional] Click *Set as Ignored* for unmapped source pins. Similarly, click *Set as Unconnected* for unmapped target pins.


© 2023 Cadence Design Systems, Inc.

7. Click *OK*. The Cell Replacement Dashboard shows the *Pin Mapping Status* as `Complete`.?




8. Click *OK* to continue with the Enablement flow.
9. [Optional] Click *Load* to read a mapping file for restoring the mapping between the cells and pins. However, reanalyze the status because the information in the file might be outdated.
10. [Optional] Click *Save* to save the current mapping. The button is enabled only if at least one cell replacement has been done.

### *Related Topics*

Cell Replacement Form

Cell Replacement Dashboard Form

Multi-Technology Enablement Flow

Virtuoso Multi Technology Enablement Form

Initial SiP File to Virtuoso RF Solution Database

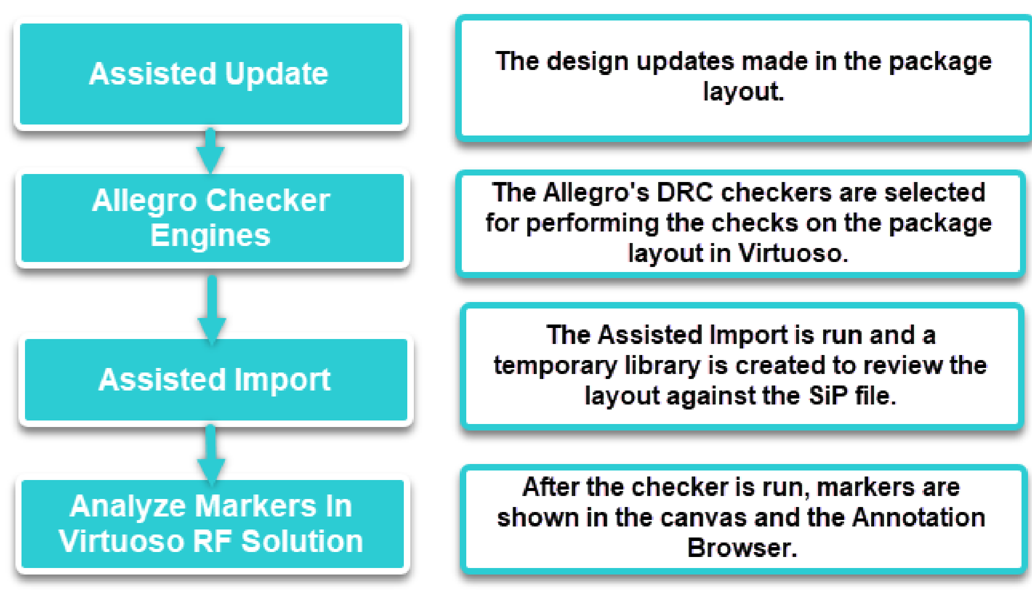

© 2023 Cadence Design Systems, Inc.

# 13

# SiP DRC Checker

The SiP DRC checker lets you check DRC issues in package layout within the Virtuoso RF Solution environment. It uses DRC checking engines of Allegro in the background.

Once the checks are complete, markers are shown in the same Virtuoso RF package layout and the Annotation browser. Consequently, you do not have to step out of Virtuoso Studio to view or analyze the issues. Markers have the same look and feel as Allegro markers, thereby, making the adoption easier for existing package designers, who are familiar with the Allegro environment. Both DRC checking engines from Allegro, `dbdoctor` and RAVEL, are supported.




***Related Topics***

Performing DRC Checks in Virtuoso RF Solution


© 2023 Cadence Design Systems, Inc.

SiP DRC Check Form

Updating a Virtuoso Layout From a SiP File

SiP to Virtuoso Layout Assisted Import and Export Flows

## Performing DRC Checks in Virtuoso RF Solution

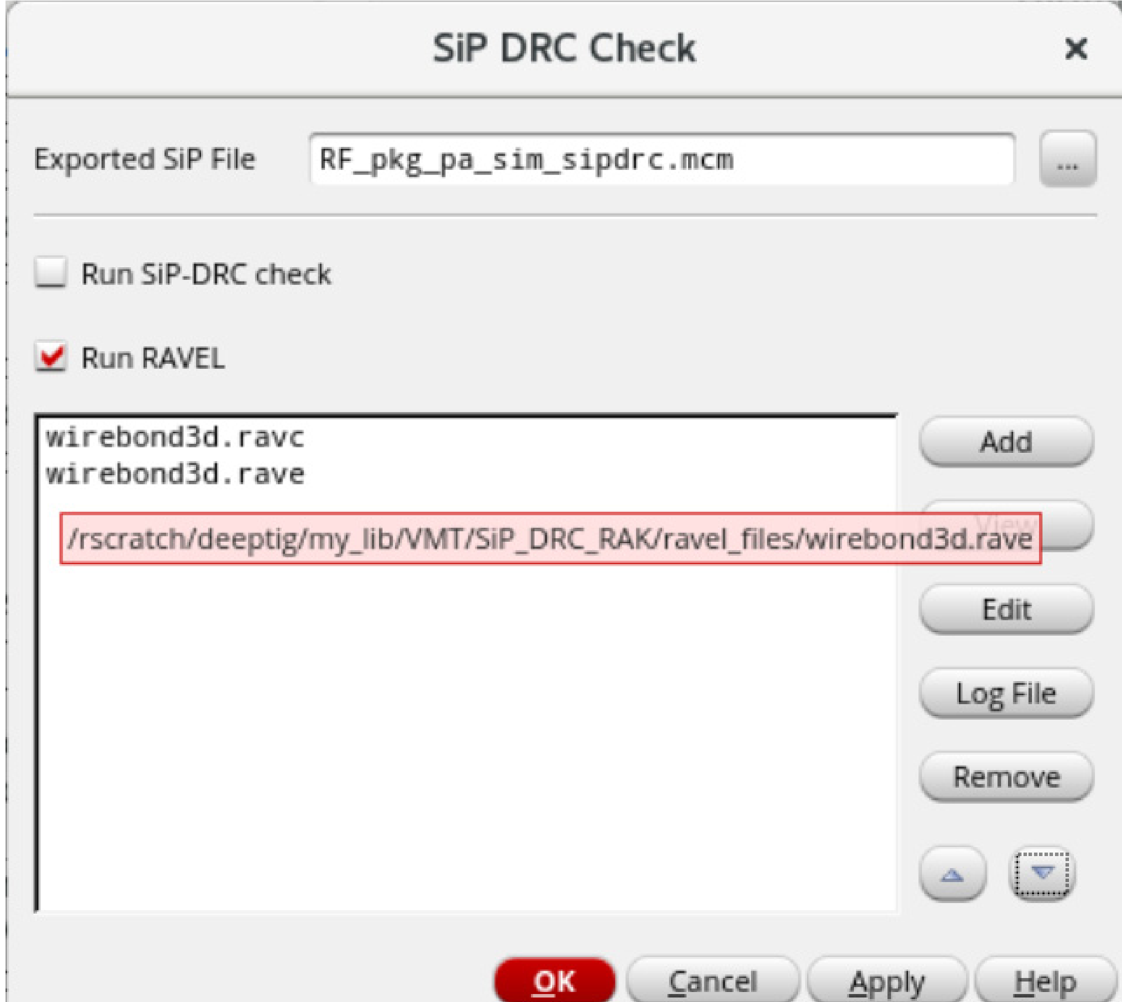
You can check package layout in Virtuoso RF Solution using the same SiP-DRC checker engine as in SiP Layout Option and view violation markers in the Annotation Browser without having to leave Virtuoso Studio.

To perform design rules checks:

1. Click *Module – SiP DRC Check*.


© 2023 Cadence Design Systems, Inc.

The SiP DRC Check Form opens.




2. Specify the SiP file to be exported from the Virtuoso RF Solution.
3. Select the types of checks, SiP DRC or RAVEL, to be performed on the SiP file. When you specify RAVEL files with long names, the form resizes to show the complete name. You can also view the path of files in the tooltip and the CIW. Multiple files can be added from the same location because the last accessed path is recognized.

   Multiple RAVEL files can be specified in the SiP-DRC Check form. Use the following one of the three or more methods to define multiple files:

   - Provide multiple `.ravc` in the SiP-DRC GUI. When the run is complete, all the violations related to the .ravc files are shown in the Annotation Browser and through canvas markers.
   - Provide multiple `.ravc` + `.rave` files in the following combination:


© 2023 Cadence Design Systems, Inc.

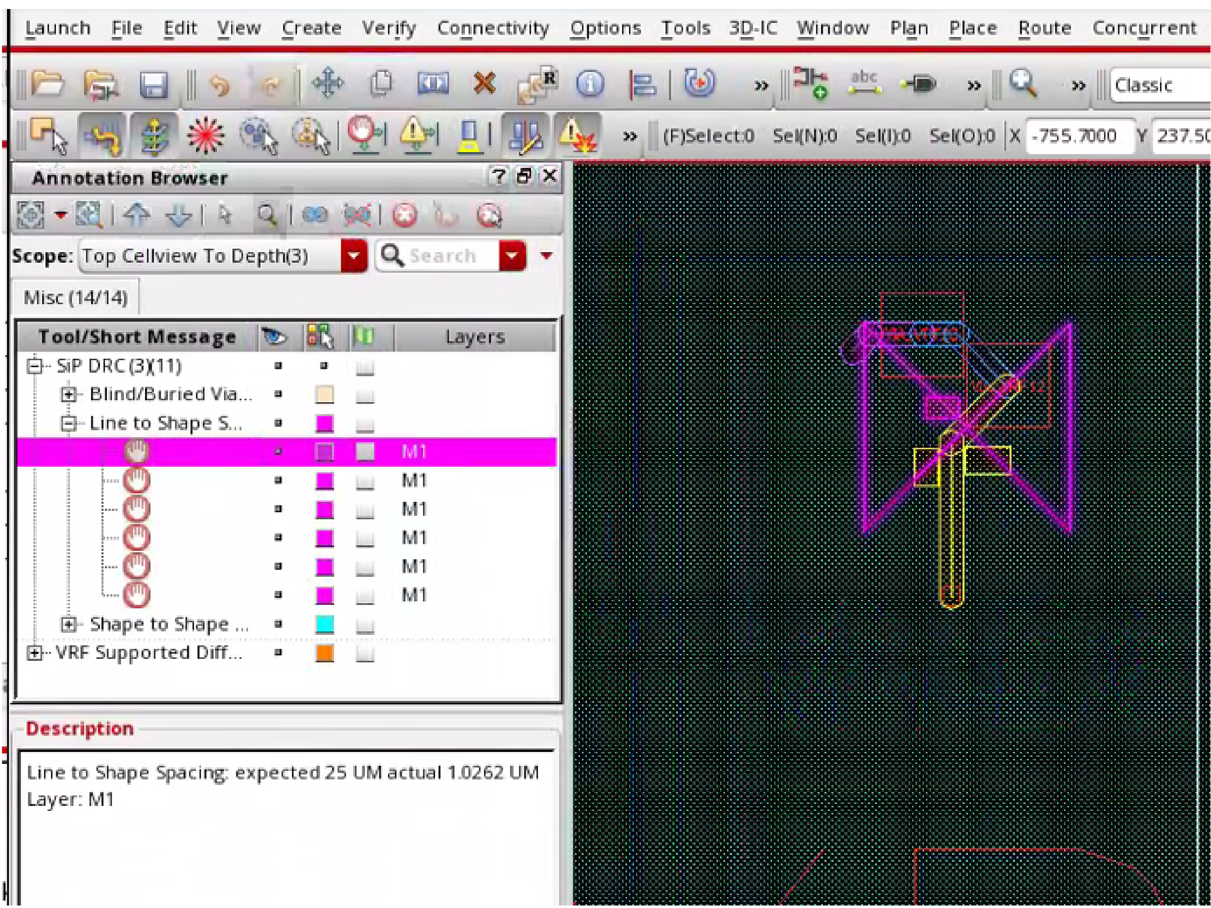
`file1.ravc`, `file1.rave`, `file2.ravc`, `file2.rave`

- ❑ This ensures in determining the pairs of `.ravc` and `.rave` and avoiding any order-specific error. When the run is complete, all the violations related to combination files are shown in the Annotation Browser and through canvas markers.
- ❑ Provide multiple source code (.rav) files.

  When the run is complete, all the violations related to the .rav files are shown in the Annotation Browser and through canvas markers.

**4.** Click *OK*.

The markers in the layout window and the *Misc* tab of the Annotation Browser are shown.




### *Related Topics*

SiP DRC Check Form


© 2023 Cadence Design Systems, Inc.

SiP DRC Checker

Updating a Virtuoso Layout From a SiP File

SiP to Virtuoso Layout Assisted Import and Export Flows


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# A

# Virtuoso Multi-Technology Forms

Click to view the information about the Virtuoso Multi-Technology forms as they appear in the GUI.

- Allegro Design Layout Importer Form
- Annotate From Extracted View Form
- Cell Replacement Form
- Cell Replacement Dashboard Form
- Convert Allegro Libraries to Unified Libraries Form
- Create Extracted View Form
- Export Design Update Form
- Pad Stack Replacement Form
- SiP DRC Check Form
- Virtuoso Multi Technology Enablement Form
- XOR SiP against OA Form
- Add Bond Finger Definition Form
- Add Layer Form
- Batch Checker Form
- Bind Layout Form
- Bump and Ball Editor Form
- Bump Connectivity Assignment Form
- Bump Propagate Form
- Configure Module Stack Form


© 2023 Cadence Design Systems, Inc.

- Create Bond Wire Form
- Create Bump and TSV Form
- Create Guides Form
- Create TILP Form
- Cross Fabric Check Violation Summary and Navigation Form
- Die Instance Annotations Form
- Edit Instance Properties Form (Die/Package TILP Parameters)
- Export Bump Info Form
- Export Die Form
- Import Bump Info Form
- Import Die Text File Form
- Layer Stack Editor Form
- Set Bond Wire Profile Form
- Virtuoso RF Compliance Audit Form
- Virtuoso Multi Technology Options Form


© 2023 Cadence Design Systems, Inc.

# Allegro Design Layout Importer Form

Updates the package layout from the specified SiP file through an imported library.

| Field | Description |
|---|---|
| *File Name* | The name of a SiP file that is used to update the package layout. |
| ***Layout Reference*** | This section provides options to specify package layout that will be updated. |
| *Library Name* | The library name of a Virtuoso layout to be compared with a SiP file. |
| *Cell Name* | The cell name of a Virtuoso layout to be compared with a SiP file. |
| *View Name* | The view name of a Virtuoso layout to be compared with a SiP file. |
| ***Imported Library*** | This section provides an option to specify a staging library to be used to perform various checks. |
| *Library Name* | The name of the library created for importing the updates from a SiP file to a package layout. |

***Related Topics***

Updating a Virtuoso Layout From a SiP File


© 2023 Cadence Design Systems, Inc.

# Annotate From Extracted View Form

Highlights the nets or models on the master schematic that have been extracted.

| Field | Description |
| --- | --- |
| *Extracted View* | The name of the extracted view that is referred to for backannotation. |

***Related Topics***

Creating an Extracted View


© 2023 Cadence Design Systems, Inc.

# Cell Replacement Form

Lets you choose the source cell and define the mapping of the pins between the current and the replacement cell.

| Field | Description |
|---|---|
| *Source Cell* | The name of the cell to be replaced. |
| *Replacement Library* | The library name of the target symbol view. |
| *Replacement Cell* | The cell name of the target symbol view. When specified, the *Unmapped Pins* section is automatically populated. |
| *Replacement View* | The name of the target symbol view. |
| *Mapping Mode* | Specifies the method of mappings pins. The default value is `Auto`.<br>`Auto`: Runs all mapping modes and selects the mode in which highest number of pins match.<br>`Symbol Pin Name`: Maps the symbol pin names of the source with the symbol pin name of the replacement cell.?<br>`Footprint Pin Name`: Maps the footprint pin name of the source with the footprint pin name of the replacement cell. In this mode, the layout information is used to drive the connectivity at schematic level.?<br>`Footprint Pin Position`: Maps the footprint pin position of the source with the footprint pin position of the replacement cell. An additional option, *Use flipped footprint*, is available in this mode. When selected, one of the abstracts or footprints is flipped to do the matching. This flipping is only for computation and not an actual footprint change.<br>`Best Pin Name Match`: Maps the symbol pin names using the prefix or suffix of the source with the symbol pin names of the replacement cell. |
| ***Unmapped Pins*** | This section provides options to specify the new schematic to be created by the enablement flow. |
| *Source Pins* | Lists the source pins that cannot be mapped. Clicking *Set as Ignored* indicates that it has been concluded after a review that the pin cannot be mapped and it will be ignored. |


© 2023 Cadence Design Systems, Inc.

| **Field** | **Description** |
|---|---|
| *Target Pins* | Lists the replacement cell pins that cannot be mapped. Clicking *Set as Unconnected* indicates that it has been concluded after a review that the pin cannot be mapped and it will be left unconnected. |
| ***Mapped Pins*** | This section shows the mapped pins of source and replacement cells. It is populated automatically when the *Mapping Mode* is selected. You can select a row to remove the mapping. |

***Related Topics***

Cell Replacement Dashboard Form

Cell Replacement

Initial SiP File to Virtuoso RF Solution Database

Multi-Technology Enablement Flow


© 2023 Cadence Design Systems, Inc.

# Cell Replacement Dashboard Form

Provides a global view of the cell replacements during the enablement flow.

| Field | Description |
|---|---|
| *Current Cell* | Lists all the cells from the Allegro design that can be replaced. |
| *Replacement Library* | The library name of the target symbol view. It is populated automatically after the pins have been mapped successfully in the Cell Replacement form. |
| *Replacement Cell* | The cell name of the target symbol view. It is populated automatically after the pins have been mapped successfully in the Cell Replacement form. |
| *Pin Mapping Status* | Sets the status as `Complete` after the pins have been mapped successfully in the Cell Replacement form. |

***Related Topics***

Cell Replacement Form

Cell Replacement

Initial SiP File to Virtuoso RF Solution Database

Multi-Technology Enablement Flow


© 2023 Cadence Design Systems, Inc.

# Convert Allegro Libraries to Unified Libraries Form

Converts Allegro libraries into multi-platform unified library components that can be used across the platforms.

| Field | Description |
|---|---|
| *Library* | The name of the unified library. |
| *Part File* | The name of the part file that contains information of the symbols in the Allegro platform. |
| *Run directory* | The name of the directory where conversion happens. |

***Related Topics***

Converting Allegro Libraries to Unified Libraries


© 2023 Cadence Design Systems, Inc.

# Create Extracted View Form

Propagates the changes done in the master schematic to the extracted view.

| Field | Description |
|---|---|
| *Library* | The library name of the package schematic. |
| *Cell* | The cell name of the package schematic. |
| *Reference View* | The name of the view that you want to use as a connectivity reference to stitch the devices and nets from the S-parameter model. You can use a schematic view, Quantus Smart View, or another layout view as a reference. |
| *Sparam View* | The name of the extracted view where the changes are being propagated. |
| *Coupling Capacitor Mode* | Specifies how to stitch a coupling capacitor when one of the two nets attached to the capacitor is included in the model. This variable is used when the reference view used for the extracted view is a Quantus Smart View.<br><br>Possible values are:<br><br>■ `ground`: The coupling capacitor of the net excluded from the model is connected to ground. This is the default value.<br><br>■ `position`: The coupling capacitor of the net excluded from the model is connected to the nearest surviving node of the net included in the EM model.<br><br>Related environment variable: couplingCapMode |
| *Model Files* | The model files to be used for comparison. The files must be `*.snp`. |

***Related Topics***

Creating an Extracted View

Creating Extracted Views from Models


© 2023 Cadence Design Systems, Inc.

# Create MultiTech Schematic Form

Creates a schematic view based on the connectivity information defined in a SiP file.

| Field | Description |
|---|---|
| *Allegro Layout* | The SiP file to be used for creating a schematic. |
| *Setup Map Table* | The action to fill the Library Map Table by using the information from the SiP file. |
| *Map File* | The name of the file that was created in a previous mapping run in the current or another cellview. It contains the mapping information, which can be reused while creating a schematic. |
| *Available Libraries* | The list of available unified libraries from which you can select the libraries to be used for mapping. |
| *Mapping Libraries* | The list of selected unified libraries to map the instances in the SiP file with the cells and views in the libraries. |
| *Library Map Table* | The table that gets populated with the mapping information from the mapped libraries, map file, or can be edited manually to add information. |
| *Add and Remove buttons* | The buttons to add or remove libraries from the *Mapping Libraries* list. |
| *Up and Down arrow buttons* | The buttons to order libraries in the *Mapping Libraries* list. |

***Related Topics***

SiP Layout Option to Virtuoso Schematic Editor Flow

Creating a Schematic Layout from a SiP File


© 2023 Cadence Design Systems, Inc.

# Export Design Update Form

Exports the updates from a package layout to a SiP file.

| Field | Description |
|---|---|
| *Output Sip File Name* | The name of the SiP file to be exported. |
| *Report modifications only* | Lists the design updates but does not export the SiP file. |

***Related Topics***

Virtuoso Layout to SiP with Assisted Export


© 2023 Cadence Design Systems, Inc.

# Pad Stack Replacement Form

Replaces a padstack cell of a footprint by another padstack cell, which is from the current footprint library or another library.

| Field | Description |
|---|---|
| *Footprint* | Specifies the footprint cellview of the selected TILP instance. |
| *Source PadStack* | Specifies the padstack cell to be replaced in the current footprint. |
| *Target PadStack* | Specifies the target padstack cell based on the following options:<br>■ *From Current Footprint Library*: (Default) Uses a padstack cell from the current library.<br>■ *Copy From Another Library*: Copies the padstack cell from another library. |
| Drop-down List enabled when the *Copy From Another Library* option is selected | Lists the libraries available to choose from when the *Copy From Another Library* option is selected. |
| Second drop-down list | Lists the padstack cells available in the current footprint library or in the selected target library when the *Copy From Another Library* option is selected. |

***Related Topics***

Replacing Pads


© 2023 Cadence Design Systems, Inc.

# SiP DRC Check Form

Performs the design rules checks (DRC) on a SiP file in Virtuoso.

| Field | Description |
|---|---|
| *Exported SiP File* | The name of the SiP file to be exported from Virtuoso RF Solution. |
| *Run SiP-DRC check* | Runs the SiP-DRC check using `dbdoctor` (The native DRC checking engine of Allegro) on the SiP file. |
| *Run RAVEL* | Runs the RAVEL engine checker on the RAVEL files in the list box to check for the RAVEL rules on the package layout. |
| *Add* | Adds RAVEL files in the file list box. |
| *View* | Opens the selected RAVEL file from the list in a read-only window to see the file content. |
| *Edit* | Opens the selected RAVEL file from the list in an editor, such as `vi`, to edit the file based on Linux permissions. Otherwise, an error is shown in the CIW. |
| *Log File* | Opens the run log file. |
| *Remove* | Removes the selected files from the list. |

***Related Topics***

SiP DRC Checker

Performing DRC Checks in Virtuoso RF Solution


© 2023 Cadence Design Systems, Inc.

# Virtuoso Multi Technology Enablement Form

Creates the technology file, schematic, and layout in a single step.

| Field | Description |
|---|---|
| *Allegro File* | The name of a native SiP file that will be used to create a technology library, Virtuoso schematic, and layout. |
| ***Technology and Components*** | This section provides options to create a technology and components library. |
| *For Existing Libraries* | |
| *Overwrite technology and components information* | Select this option if you want to overwrite the information in the existing libraries. It is useful when the existing library does not need to be kept and can or must be overwritten. For example, PKG_LIB already exists and you want keep the same name but you want to overwrite the content. By default, this option is selected. |
| *Add missing components and via only* | Select this option if you just want to add only the missing components and via from the SiP file. It is useful when the existing library must be kept because you may have modified information, such as the schematic symbol or the technology information. You just want to add new cells or vias into the existing library. |
| *Schematic Symbols* | |
| *Create new symbols from the footprint* | Select this option to create new symbols during the libimport process. It is useful when no schematic symbol already exists. By default, this option is selected. |
| *Copy symbols from* | Select this option if you want to use the symbols from the standard Cadence or any other libraries, which you can choose from the adjoining drop-down list. The list is enabled when the option is selected. If you have saved well-defined symbols in a library, you can use them. |
| ***Schematic*** | This section provides options to specify the new schematic to be created by the enablement flow. |
| *Library* | The library name of the new mult-technology schematic that will be generated. |


© 2023 Cadence Design Systems, Inc.

| Field | Description |
|---|---|
| *Cell* | The cell name of the new multi-technology schematic that will be generated. |
| *View* | The view name of the new multi-technology schematic that will be generated. |
| ***Cell Replacement*** | This section provides a check box to open the Cell Replacement Dashboard after the schematic generation step. |
| ***Layout*** | This section provides options to the new layout to be created by the enablement flow. |
| *Same as schematic cell* | Select this option to specify if the name of the layout cell is same as the multi-technology schematic cell. By default, this option is selected. |
| *Specific cell* | Select this option to specify another cell for the layout. It is useful when you want the layout view to be in another cell or in another library than the schematic cell or library. |
| *Library* | The library name of the new layout that will be generated. |
| *Cell* | The cell name of the new layout that will be generated. |
| *View* | The view name of the new layout that will be generated. |

***Related Topics***

Initial SiP File to Virtuoso RF Solution Database

Multi-Technology Enablement Flow


© 2023 Cadence Design Systems, Inc.

# XOR SiP against OA Form

Compares a SiP file against an OpenAccess layout.

| Field | Description |
|---|---|
| *Allegro Input File* | The name of a SiP file for the XOR operation. |
| ***Layout Reference*** | This section provides options to specify the layout for the XOR operation. |
| *Library Name* | The library name of a Virtuoso layout to be compared with a SiP file. |
| *Cell Name* | The cell name of a Virtuoso layout to be compared with a SiP file. |
| *View Name* | The view name of a Virtuoso layout to be compared with a SiP file. |
| ***Output XOR location*** | This section provides options to specify the new layout created after the XOR operation. |
| *XOR Library Name* | The library name of the new layout to be generated. |
| *XOR Cell Name* | The cell name of the new layout that will be generated. |
| *XOR View Name* | The view name of the new layout that will be generated. |
| ***Options*** | This section provides options to customize the XOR operation. |
| *Filter XOR shapes smaller than:* | The size of a window used to remove small shapes from the generated layout. Shapes smaller than the specified aperture are not visible in the generated layout. |
| *Compare Static Shapes* | Creates static shapes from the Virtuoso layout at runtime and compares them with the shapes specified in the SiP file. |
| *Compare Conducting Layers Only* | Limits the comparison to only the shapes on conducting layers, that is, layers defined in a cross-section. |
| *Generate True Marker Shapes* | Uses the XOR-operation-generated shapes in a Virtuoso layout as markers. |
| *Open Destination Design When Done* | Opens the newly generated layout when the XOR operation is completed. |


© 2023 Cadence Design Systems, Inc.

### *Related Topics*

vmtCompareSipToOa


© 2023 Cadence Design Systems, Inc.

# Add Bond Finger Definition Form

Use the Add Bond Finger Definition form to create bond finger definitions.

| Field | Description |
|---|---|
| *Definition Name* | Specifies a unique bond finger definition name. |
| *Shape Type* | Specifies the shape of the bond finger - *rectangle*, *oblong*, *circle*, or *padStack*. |
| *Layer* | Specifies the layer in which bond finger must be generated. |
| *width* | Specifies the width of the bond finger. |
| *length* | Specifies the length of the bond finger. |

***Related Topics***

Bond Wires and Bond Fingers Creation

Creating Bond Finger Definitions

Moving Bond Wires and Bond Fingers

Updating the Finger Attach Point

# Add Layer Form

Use the Add Layer form to add the layers to the stack.

| Field | Description |
|---|---|
| *Layer Name* | Specifies the name of the layer being added. |
| *Where* | Specifies the position of adding the layer, top or bottom. |


© 2023 Cadence Design Systems, Inc.

***Related Topics***

Preparing for Model Creation

# Batch Checker Form

Use the Batch Checker form to set up the options for performing DRD checks on a layout.

| Field | Description |
|---|---|
| *Current Editable Cellview* | Checks shapes in the entire design for process rule violations. |
| *Area* | Checks shapes in the selected area. |
| *Select Area* | Lets you select an area within the current cellview. |
| *Marker Limit* | Specifies the maximum number of post-edit markers to be created per call. The default is `5000`. |
| *Component Overlap Check* | Checks if the PR boundaries of two instances placed next to each other overlap. |

***Related Topics***

Performing DRD Checks

# Bind Layout Form

Use the Bind Layout form to get information about the layout cellview to which the current package die is bound. You can view and edit this information.

| Field | Description |
|---|---|
| *Library* | Specifies the library to which the current package die is bound. |
| *Cell* | Specifies the cell to which the current package die is bound. |


© 2023 Cadence Design Systems, Inc.

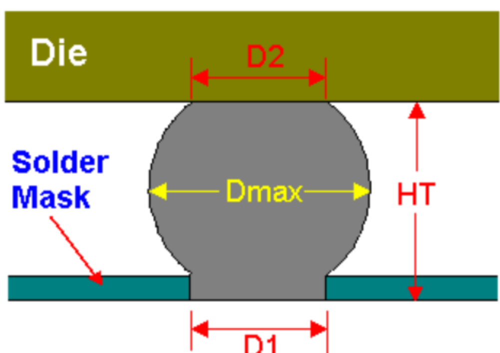
| Field | Description |
|---|---|
| *View* | Specifies the view to which the current package die is bound. |

***Related Topics***

Update Binding Information

# Bump and Ball Editor Form

Use the Bump and Ball Editor form to specify the physical parameters for IO pads and balls. The following image represents the physical attributes of a solder ball that are specified in the Bump and Ball Editor form.




| Field | Description |
|---|---|
| *Diameter Top* | Specifies the top diameter (D2) measured on the dies side. |
| *Diameter Max* | Specifies the widest part (Dmax) of the bump after bulging occurs during die mounting. |
| *Diameter Bottom* | Specifies the bottom diameter (D1) measured on the package side. |
| *Height* | Specifies the height (HT) of the chip after the solder ball is fused with the substrate metal. |
| *Material* | Specifies the material used to make the solder ball. |


© 2023 Cadence Design Systems, Inc.

***Related Topics***

Validating BGA Ball and Flipchip Bump Setup

# Bump Connectivity Assignment Form

The following table describes the fields available in the Bump Connectivity Assignment form.

| Field | Description |
|---|---|
| *Auto Assignment* | Enables automatic bump connectivity assignments. This option is selected by default and cannot be edited. |
| *Bumps* | Specifies the bumps for which connectivity is to be assigned. The available options are All and Selected. In the Selected mode, select the required bumps in the layout canvas. |
| *Nets* | Specifies the nets to which bumps can be assigned. The available options are *All* and *Selected*. In the *Selected* mode, select the required nets either in the layout canvas or in the Navigator assistant. |

***Related Topics***

Assigning Connectivity between Bumps


© 2023 Cadence Design Systems, Inc.

# Bump Propagate Form

The following table describes the fields available in the Bump Propagate form.

| Field | Description |
|---|---|
| *Library* | Specifies the library of the cellview that contains the bump definition. |
| *Cell* | Specifies the cell that contains the bump definition. |
| *View* | Specifies the view that contains the bump definition. |

***Related Topics***

Propagating Bumps


© 2023 Cadence Design Systems, Inc.

# Configure Module Stack Form

The following table describes the fields available in the Configure Module Stack form.

| **Field** | **Description** |
|---|---|
| *Module* | Lists lists all the modules or dies that are part of the current design. |
| *Flipped* | Specifies whether each die is to be flipped. |
| *Order* | Specifies the order in which the die are to be placed in the stack. |
| *Mirrored* | Specifies the orientation of each die. |

***Related Topics***

Configuring a Stack

# Create Bond Wire Form

Use the Create Bond Wire form to generate bond wires.

| **Field** | **Description** |
|---|---|
| *Create* | Specifies what must be created - *Bond Wire*, *Bond Finger*, or *Both*.<br><br>Environment variable: createWireFingerOrBoth |
| *Wire Profile* | Lists all wire profiles that are defined in the technology file. This option lists all the wire profiles that are defined in the technology file. A wire profile defines bond wire properties such as the width of the wire. These properties are not listed in Property Editor and therefore cannot be edited after the bond wires are created.<br><br>Environment variable: bumpCenterMismatchCheck |
| *Snap To* | Specifies whether the components must snap to *Bond Finger*, *Guide*, or *Die IO*. |


© 2023 Cadence Design Systems, Inc.

| Field | Description |
|---|---|
| *Distribute On Guide* | Specifies how bond fingers are to be distributed on the guide.<br>■ *Uniform*: Distributes bond fingers evenly along the guide.<br>■ *Orthogonal*: Places bond fingers orthogonal to the guide. The bond fingers are placed at the nearest point in the guide, and therefore this setting ensures the shortest bond wire length.<br>■ *Any Angle*: Uses the `pkgMinBondwireSpacing` constraint value to determine the minimum spacing between adjacent bond fingers. |
| *Definition* | Specifies a bond finger definition. To create a new definition, click *Add*. To view an existing definition, select it from the drop-down list and click *Display*.<br>Environment variable: bondFingerProfile |
| *Alignment* | Specifies the bond finger alignment - *along wire* or *orthogonal to die*.<br>Environment variable: bondFingerAlignment |
| *Honor Constraints* | Specifies whether the constraints or the bond wire profile get higher priority while defining the settings. |

***Related Topics***

Bond Wires and Bond Fingers Creation

Creating Bond Wires and Bond Fingers

Moving Bond Wires and Bond Fingers

honorConstraints


© 2023 Cadence Design Systems, Inc.

# Create Bump and TSV Form

The following table describes the fields available in the Create Bump and TSV form.

| Field | Description |
|---|---|
| ***Bump Array Specifications*** | Lets you specify the following bump array settings. |
| *X-Origin* | Specifies the X coordinate of the first bump of the array. |
| *Y-Origin* | Specifies the Y coordinate of the first bump of the array. |
| *Horizontal Pitch* | Specifies the horizontal distance between the bump edges. |
| *Vertical Pitch* | Specifies the vertical distance between the bump edges. |
| *No. of Rows* | Specifies the number of rows to be generated in the bump array. |
| *No. of Columns* | Specifies the number of columns to be generated in the bump array. |
| ***Bump*** | Specifies the *Library*, *Cell*, and *View* of the cellview that defines the bumps. |
| ***Create TSV*** | Specifies the following TSV settings. |
| *Via Definition* | Lists all the standard and custom vias that are available in the technology file. |
| *X-Offset* | Specifies the X offset of the TSV. |
| *Y-Offset* | Specifies the Y offset of the TSV. |

***Related Topics***

Creating Bumps and TSVs


© 2023 Cadence Design Systems, Inc.

# Create Guides Form

Use the Create Guides form to create line-shaped or arc-shaped guides around dies.

| Field | Description |
|---|---|
| *Die Side* | Specifies the slide along which guides must be created. Options are: *All*, *North*, *East*, *West*, and *South*. |
| *Distance* | Specifies the distance from the reference at which the guides must be created. |
| *From edge to the die boundary* | Specifies that the distance is to be calculated from the PR boundary of the die boundary. |
| *From edge to the die outline* | Specifies that the distance is to be calculated from the die outline. |
| *Radius* | Lets you create arc-shaped guides. |
| *Guide Vertices offset (+/-)* | Specifies the distance of the diagonal line drawn from the two vertices of the guide to the center of the die. |

***Related Topics***

Guides in Wirebonded Dies

Creating a Guide

Editing a Guide

# Create TILP Form

Use the Create TILP form to create TILPs from the package layout.

| Field | Description |
|---|---|
| *Base Cell* | Specifies the base cellview that has been used to create the TILP. |
| *Target Cell* | Specifies the cellview of the newly created TILP. |


© 2023 Cadence Design Systems, Inc.

| Field | Description |
|---|---|
| *TILP Type* | Specifies the options for the type of TILP to be created. |

***Related Topics***

Creating TILPs

# Cross Fabric Check Violation Summary and Navigation Form

Use the Cross Fabric Check Violation Summary and Navigation form to compare connectivity across fabrics and display results in a system check report.

| Field | Description |
|---|---|
| *CellView* | Lists the cellviews in which violations are reported. |
| *Fabric Type* | Specifies the fabric type of the cellviews. |
| *Violation For Connectivity Extraction* | Lists the number of violations reported during connectivity extraction. |
| *Violation For Check Against Source* | Lists the number of violations reported by the Check Against Source (CAS) checker. |
| *Violation For Layout Vs Abstract* | Lists the number of violations reported by the Layout vs Abstract (LVA) checker. |

***Related Topics***

Cross-Fabric Checks Run


© 2023 Cadence Design Systems, Inc.

# Die Instance Annotations Form

Use the Die Instance Annotations form to specify the options to display die instance annotations on the canvas.

| Field | Description |
|---|---|
| ***Annotations*** | This section provides options to customize annotation text display. |
| *Text Size* | Controls the size of text in the die instance annotations. |
| *Bound Die Instance LPP* | Specifies the highlight color for bound die instances. |
| *Unbound Die Instance LPP* | Specifies the highlight color for unbound die instances. |
| *Display* | Displays the die instance annotations on the screen. |
| *Remove* | Hides the die instance annotations. |

**Related Topics**

Viewing Die Instance Annotations

# Edit Instance Properties Form (Die/Package TILP Parameters)

Use the *Parameter* tab of the Edit Instance Properties form to specify the die or package TILP parameters.

| Field | Description |
|---|---|
| ***Parameter*** | This section provides options to customize die or package TILP parameters. |


© 2023 Cadence Design Systems, Inc.

| Field | Description |
|---|---|
| *Angle* | Specifies the placement angle of a die. The following translation table defines the angle to be set based on instance orientation:<br>The following list defines the TILP angle to be set based on instance orientation:<br>■ oacR0 - `0`<br>■ oacR90 - `90`<br>■ oacR180 - `180`<br>■ oacR270 - `270`<br>■ oacMY - `0`<br>■ oacMYR90 - `0`<br>■ oacMX - `0`<br>■ oacMXR90 - `0`<br>The orientation of the layout instance during GFS is set based on the `lxGenerationOrientation` environment variable, which is, by default, set to `R0`, so the TILP angle is always set to `90`.<br>When the environment variable is set to `preserve`, the layout instance orientation takes the value from schematic instance and sets accordingly as mentioned in the table. |
| *Mirrored* | Specifies if the die is mirrored to its original position with respect to the package. A mirrored die is placed under the package. |
| *Stackup Offset/Package Stackup Offset* | Specifies the distance from the die/package TILP stack. |
| *Flipped/Geometric Mirror* | Specifies whether the die is flipped before placing. |
| *Die Stack Order/ Package Stack Order* | Specifies the die/package TILP stack order. The position of die or package from the substrate. |
| *Shrink Factor* | Specifies the ratio by which the die is shrunk optically while placing on the package. It is indicated as 1, which is equivalent to 100%. Therefore, if a die is expanded, the value could be greater than 1 (100%). |


© 2023 Cadence Design Systems, Inc.

| Field | Description |
|---|---|
| *X Thermal Shrink Factor (%)* | Specifies the percentage by which a die is shrunk thermally on the x-axis. |
| *Y Thermal Shrink Factor (%)* | Specifies the percentage by which a die is shrunk thermally on the y-axis. |
| *Tilp Version* | Specifies the version of a TILP indicating the parameters supported. |
| *Target Library Name* | Specifies the name of the library where a TILP has been created. This field is auto-populated but can be modified. |
| *Target Cell Name* | Specifies the name of the cell where a TILP has been created. This field is auto-populated but can be modified. |
| *footprint Library Name* | Specifies the name of the library where a footprint has been created. This field is auto-populated but can be modified. |
| *footprint Cell Name* | Specifies the name of the cell where a footprint has been created. This field is auto-populated but can be modified. |
| *footprint View Name/ Base Cell View Name* | Specifies the name of the abstract view. |
| *footprint ALT Cell Name* | Specifies the alternate name of a footprint cell. It is specified if several footprints are available for the same cell. |
| *Scribe North* | Specifies an area around the PR boundary by using the scribe parameters. If a die is shrunk, the scribe area is also shrunk, but the spaces around the PR boundary always honor the scribe parameter values. An area boundary is created in the die instance sub-master that identifies the scribe area. If the scribe parameters are equal to 0, the area boundary has the same dimensions as the PR boundary. To enforce the scribing effect, *Scribe North* is used to define values in the north direction. |
| *Scribe South* | Specifies the scribe values in the south direction. |
| *Scribe East* | Specifies the scribe values in the east direction. |
| *Scribe West* | Specifies the scribe values in the west direction. |
| *Thickness* | Indicates the thickness value of a scribe. It is a CDP parameter but not a TILP parameter. Therefore, it has no effect on the TILP evaluation results. |
| *Model Name* | Refers to the S-parameter model name. |


© 2023 Cadence Design Systems, Inc.

| **Field** | **Description** |
|---|---|
| *Parameters to Display* | Filters the list of parameters displayed in the Edit Instance Properties form. |
| *Select Parts* | Specifies the part of a specific configuration that you want to use in the layout. For instance, if your cell is a CAPACITOR, you can choose a specific value capacitor and subsequently, the layout is updated after GFS. |

### *Related Topics*

Flip Chip Dies


© 2023 Cadence Design Systems, Inc.

# Export Bump Info Form

The following table describes the fields available in the Export Bump Info form.

| Field | Description |
|---|---|
| *Bump File Path* | Specifies the path and name of the file in which bump settings are to be saved. |

***Related Topics***

Saving Bumps to File

# Export Die Form

Use the Export Die form to specify inputs required for die generation. The form contains the following tabs.

| Tab | Description |
|---|---|
| General Settings | Lets you specify the most important options and know where the data is being exported. |
| Advanced Settings | Lets you specify the settings regarding how the views will be generated, s-parameter model to be added to an abstract view, or whether the *Pin Numbering* and *Area Transfer* tabs should be enabled. |
| Pin Numbering | Lets you specify the settings for defining pin names of the footprint that are called pin numbers. They can be set by using them as is in an abstract view, using the numbers from a die text file, or using predefined algorithms. |
| Area Transfer | Lets you specify some additional shapes other than bumps to be exported in an abstract view. |


© 2023 Cadence Design Systems, Inc.

## General Settings

The following table describes the fields available on the *General Settings* tab of the Export Die form.

| Field | Description |
|---|---|
| *Die Interface Type* | Specifies one of the following type of interface for which a die is being exported to identify bumps or pads.<br>■ *IO cells* if the bumps or pads are defined as cells having their cellType set to `coverBump` or `pad`. This is the recommended method because it enables editing-in-concert and the Layout Versus Abstract check.<br>■ *Shapes with overlapping labels* if the bumps can only be retrieved from the association of a shape and a label. This method does not enable editing-in-concert and the Layout Versus Abstract check. |
| *Front Pin Layer* | Specifies the layer purpose pair of the bump or pad on the front side of a die. It is usually the highest metal layer of a die. |
| *Front Label Layer* | Specifies the layer purpose pair of the bump or pad on the back side of a die when bumps and pads have been placed on both the sides of a die. |
| *Back Pin Layer* | Specifies the type of layer that should be read to obtain back side pin information. |
| *Back Label Layer* | Specifies the type of layer that should be read to know the back label. |
| *Library* | Specifies the name of the library to be generated after the export die step is successfully completed.<br>*Important*<br>If the Export Die form is launched for a different design, the default Abstract Library/Cell name corresponding to an IC layout name is displayed. However, if it is relaunched from the same design, last saved vales are displayed as the Library/Cell name. |


© 2023 Cadence Design Systems, Inc.

| Field | Description |
| --- | --- |
| *Cell* | Specifies the name of the cell to be generated after the export die step is successfully completed.<br>*Important*<br>The cell name should not be more than 31 characters. |

## Advanced Settings

The following table describes the fields available on the *Advanced Settings* tab of the Export Die form.

| Field | Description |
| --- | --- |
| *Include unconnected IO Pad cells* | When selected, includes unconnected IO pads while exporting the die. |
| *Customize pin numbers* | When selected, enables the *Pin Numbering* tab. |
| *Transfer area* | When selected, enables the *Area Transfer* tab. |
| *Output Mode* | Specifies how views will be updated while exporting a die. *Generate all views* value ensures that all views are created during die export. *Update existing symbol and generate others* allows to only update the symbol view and create the remaining views. *Generate all views (schematic view when possible)* lets you generate all possible views during die export, however, generating a schematic view is optional. |


© 2023 Cadence Design Systems, Inc.

| | |
|---|---|
| *Delete existing views before export* | When selected, all the die export views of the specified cell in the specified library are deleted before exporting.<br><br>If the library pre-exists, the other views and cells are not deleted. However, if the library is created during die export, it gets deleted if export fails.<br><br>When the value of *Output Mode* is set as *Update existing symbol and generate others*, the *Delete existing views before export* check box is deselected and disabled.<br><br>When the value of *Output Mode* is set as *Generate all views (schematic view when possible)*, the *Delete existing views before export* check box is selected and disabled. |
| *Attach model during die export* | Enables the fields to attach models during die export. |
| *Model View Name* | Displays the view name of the model. |
| *Model File* | Specifies the name and the location of the model file being imported. |
| *CSV File* | Specifies the name of a part.csv file to add extra information in the file. |


© 2023 Cadence Design Systems, Inc.

## Pin Numbering

The following table describes the fields available on the *Pin Numbering* tab of the Export Die form.

| Field | Description |
|---|---|
| *Mode* | Specifies the options for defining pin numbers, that is, footprint pin names.<br>■ *Use pin names as numbers* uses the same pin names for the symbol and footprint views in the generated abstract and ensures that the names are unique.<br>■ *Use numbers as-is* is chosen when IO cells properties is read to assign pin numbers.<br>■ *Use numbers from file* is chosen when the pin numbers are defined in a die text file. It is specified in the *Die Text File* field.<br>■ *Use predefined ordering* is chosen when pin ordering algorithms are used. The *Ordering* drop-down list is enabled to select predefined or custom algorithm. |
| *Die Text File* | Specifies the die text file to be referenced when generating die pin numbers.<br>Environment variable: pinNumberFile |


© 2023 Cadence Design Systems, Inc.

| | |
|---|---|
| *Ordering* | Specifies the options to define pin numbering based on algorithms.<br>■ *Vertical – bottom right* value ensures that the pin at the bottom-right corner is considered as the first pin. The other pins are placed column-wise from bottom to top.<br>■ *Vertical – bottom right minimum distance* indicates that the pin located at the minimum diagonal distance from the bottom right of the die is considered as the first pin. The other pins are placed column-wise from bottom to top.<br>■ When *Custom* is selected, you can define a custom algorithm by using SKILL.<br>Environment variables: pinOrdering, pinOrdering-Custom |

## Area Transfer

The following table describes the fields available on the *Area Transfer* tab of the Export Die form.

| Field | Description |
|---|---|
| *Area Transfer File* | Specifies an XML file that includes rules to add some additional shapes in an abstract view, such as logos. Cadence recommends that you use a non-conductor SIP_SHAPE_CLASS layer (such as COMPONENT GEOMETRY) as the target layer in the XML file instead of a conductor layer. SiP Layout Option does not support CONDUCTOR class shapes or clines in die symbols because their corresponding layer might change unpredictably when the die orientation or placement is modified in the stack. |


© 2023 Cadence Design Systems, Inc.

### *Related Topics*

Exporting Dies

Virtuoso Multi-Technology Forms


© 2023 Cadence Design Systems, Inc.

# Import Bump Info Form

The following table describes the fields available in the Import Bump Info form.

| Field | Description |
|---|---|
| *Bump File Path* | Specifies the path to the file that contains bump information. |
| ***Create TSV*** | Specifies the following TSV settings. |
| *Via Definition* | Lists all the standard and custom vias that are available in the technology file. |
| *X-Offset* | Specifies the X offset of the TSV. |
| *Y-Offset* | Specifies the Y offset of the TSV. |

***Related Topics***

Creating Bumps from File

# Import Die Text File Form

Use the Import Die Text File form to specify the parameters for importing the die text file.

| Field | Description |
|---|---|
| *Die Text File* | Specifies the name of the die text file to be imported in Layout MXL. |
| *Delimiter* | Specifies the delimiter options, which function as separators, from the drop-down list box. |
| *Pad Width* | Specifies the width of the pad in the unit of measurement that you set for the padstack. |

***Related Topics***

Virtuoso Multi-Technology Forms


© 2023 Cadence Design Systems, Inc.

# Layer Stack Editor Form

Use the Layer Stack Editor form to specify the options for layers. Whenever you add or delete any layer, a confirmation dialog box is displayed.

| Field | Description |
|---|---|
| *Thickness* | Specifies the default material thickness. |
| *Material* | Specifies the name of the material used for the layer. |
| *Conductivity[S/m]* | Specifies the electrical conductivity of the material. When the value is 1000 mho/cm or less, the material is considered a dielectric, otherwise it is a conductor. |
| *Permittivity* | Specifies the permittivity of the material. It is a measure of capacitance that is encountered when forming an electric field in a particular material. |
| *Loss Tangent* | Specifies the loss tangent of the material. Loss tangent refers to the dielectric material's inherent dissipation of electromagnetic energy. |
| ***Lock*** | This section provides options to prevent adding or removing layers or attributes in the form. |
| *Add Layers* | When selected, prevents adding or removing layers but modifying layer attributes can be done. The *Add Layer*, *Delete Top*, and *Delete Bottom* buttons are disabled in the form. |
| *Values Change* | When selected, prevents modifying layer attributes but adding or removing layers can be done. The text fields to modify the attributes in the form are disabled. |

***Related Topics***

Preparing for Model Creation

Virtuoso Multi-Technology Forms

techGetLayerAnalysisAttributeLock


© 2023 Cadence Design Systems, Inc.

techSetLayerAnalysisAttributeLock

# Set Bond Wire Profile Form

Use the Set Bond Wire Profile form to apply bond wire profiles to existing bond wires to update their structures and properties.

| Field | Description |
|---|---|
| *Bondwire Profile* | Lists all bond wire profiles that are defined in the technology file. |
| *Honor Constraints* | Honors the constraints that are listed below. |
| *Direction* | Direction of the bond wire. The valid values are:<br>■ `forward`: The bond wire connection begins at the IO pad and ends at the bond finger.<br>■ `reverse`: The bond wire connection begins at the bond finger and ends at the IO pad. |
| *Material* | Material of the bond wire. The valid values are `GOLD` and `COPPER`. |
| *Diameter* | Diameter of the bond wire. |
| *Point #* | Point of curvature of the bond wire. A bond wire has one or more points that divide it into steps. Each point marks the start of a step. Each step has a horizontal section and a vertical section. |
| *Horizontal* | Attribute that defines the horizontal segment. The valid values are: *Length*, *Percentage*, and *Angle*. |
| *Value* | Value of the length, percentage, or angle of the horizontal segment. |
| *Vertical* | Attribute that defines the vertical segment. The valid values are: *Length*, *Percentage*, and *Angle*. |
| *Value* | Value of the length, percentage, or angle of the vertical segment. |


© 2023 Cadence Design Systems, Inc.

***Related Topics***

Setting a Bond Wire Profile

Virtuoso Multi-Technology Forms

# Virtuoso RF Compliance Audit Form

Lets you perform preliminary checks through the use of a template file to avoid rework in the Virtuoso RF Solution processes.

| Field | Description |
|---|---|
| ***Die Export*** | This section specifies the checks to audit the die export functionality. |
| *Template File* | Specifies the template file that is exported from the Export Die form to provide the desired settings for die audit.<br><br>Environment variable: dieTemplateFile |
| *Library Check* | Checks whether `coverBump` or `pad` cells exist in the library.<br><br>Environment variable: libraryCheck |
| *Terms Check* | Checks for dangling and floating terminals, terminals with incorrect direction, and terminals not connected to a `coverBump` or `pad` cell.<br><br>Environment variable: termsCheck |
| *IC Symbol Check* | Checks for missing IC symbols. Also, performs compatibility checks on symbols.<br><br>Environment variable: icSymbolCheck |
| *IO Check* | Checks that pad shapes exist on the specified layers and that the layer-purpose pairs are correctly defined in the technology file. Also reports duplicate pin numbers of IOs, overlapping pads, multiple connections exist to top-level terminals for an IO, or instance terminal mismatch.<br><br>Environment variable: ioCheck |


© 2023 Cadence Design Systems, Inc.

| | |
|---|---|
| *Other Checks* | Performs PR boundary checks.<br>Environment variable: otherChecks |

***Related Topics***

Die Audit

vrfComplianceAudit

# Virtuoso Multi Technology Options Form

Lets you customize the Virtuoso multi-technology-specific editing operations.

| Field | Description |
|---|---|
| ***IO Movement*** | This section provides options to specify how IO pins can be moved in a package die. |
| *Allow abstract IO movement when locked* | Specifies whether the IO pins that are locked in a layout die can be moved in a package die. |
| *Restrict IO movement within PR boundary* | Controls the area of movement of IO pins in a package die. |
| ***Check Locations*** | This section provides options to check the location and connectivity of IO pins in a package die. |
| *Extra IOs* | Checks for extra IOs in the package die that are not present in the corresponding layout die. |
| *Missing IOs* | Checks for IOs that are not available in the package die, but are present in a layout die. |
| *Mis-aligned IOs* | Checks for IOs in the package die that do not follow the same alignment as their corresponding IOs in a layout die. |
| *Aligned IOs* | Checks for all aligned pins in a package die. |
| *Connectivity Missing IOs* | Checks for IOs in a package die that do not have any valid connections. |


© 2023 Cadence Design Systems, Inc.

| | |
|---|---|
| *Connectivity Mismatch IOs* | Checks for IOs in the package die that have connectivity different than their corresponding pins in a layout die. |
| ***Fix Locations*** | This section provides options to check the location and connectivity of IO pins in a package die. |
| *Extra IOs* | Deletes extra IOs in a package die. |
| *Missing IOs* | Creates the missing IOs in a package die. |
| *Mis-aligned IOs* | Aligns the mis-aligned IOs in a package die. |
| *Connectivity Mis-matched IOs* | Reconnects IOs as per their connectivity in the corresponding layout die. |

***Related Topics***

Running the LVA Checker

Virtuoso Multi-Technology Forms


© 2023 Cadence Design Systems, Inc.

# B

# Virtuoso Multi-Technology Environment Variables

Click to view the information on the names, descriptions, and graphical user interface equivalents for the Virtuoso Multi-Technology environment variables.

- optionalPartData
- vsdpSparamCSVModelNameField
- vsdpSpiceCSVModelNameField
- areaTransferFile
- bondFingerAlignment
- bondFingerProfile
- bumpCenterMismatchCheck
- casDieLayoutVsAbstract
- cdfParamPromptLineNumber
- checkerPrecisionFactor
- createNoConn
- createWireFingerOrBoth
- csvFile
- customizePin
- deleteViewsBeforeExport
- dieTemplateFile
- distributeObjsOnGuide
- drdEditApkDrcBlockageOverlapCheck


© 2023 Cadence Design Systems, Inc.

- drdEditApkDrcComponentOverlap
- drdEditApkDrcShortCheck
- exportNoBumpTerm
- honorConstraints
- icSymbolCheck
- ignoreColumnNumbers
- ignoreLineNumbers
- instTermLabelCheck
- ioCheck
- layoutConnectivityFile
- layoutConnectivityPinNamesChkBox
- libraryCheck
- noConnCell
- otherChecks
- outputMode
- paramMap
- paramNameLineNumber
- partNameToImport
- pinNumberFile
- pinOrdering
- pinOrderingCustom
- schematicConnectivityFile
- schematicConnectivityMode
- segSnapMode
- shortPinCell
- shortPinLib
- snapEndPoint


© 2023 Cadence Design Systems, Inc.

- snapMode
- termsCheck
- transferArea
- voidGeneratorTrimUnconnectedShapes


© 2023 Cadence Design Systems, Inc.

# optionalPartData

```
cfde.siptosch optionalPartData string "VALUE PART_NUMBER TOLERANCE"
```

### Description

Specifies the optional fields to be matched while creating a multitech schematic from a SiP file. This reduces the necessary matching fields of the `part.csv` to minimum because it is difficult to ensure that the `part.csv` has the same field numbers and content in the library and SiP file. By default, the `VALUE`, `PART_NUMBER`, or `TOLERANCE` part variant fields are used to match in sequence if there is no unique match found for the mandatory `CDS_DEVICE_TYPE` and `footprintCellName` fields. You can also define other fields in the variable that will be used in sequence to find a unique match.

### GUI Equivalent

None

### Examples

```
envGetVal("cfde.siptosch" "optionalPartData")
envSetVal("cfde.siptosch" "optionalPartData" string "VALUE PART_NUMBER TOLERANCE
FREQ_RANGE TEMP_CHAR")
```

### *Related Topic*

Creating a Schematic Layout from a SiP File


© 2023 Cadence Design Systems, Inc.

# vsdpSparamCSVModelNameField

```
asimenv.netlist vsdpSparamCSVModelNameField string "ASI_MODEL"
```

### Description

Specifies the default value of the part variant field that would be used to identify the sparam model file associated with the instance. By default, the `ASI_MODEL` part variant field is used to identify the associated model file.

### GUI Equivalent

None

### Examples

```
envSetVal("asimenv.netlist" "vsdpSparamCSVModelNameField" 'string "ASI_MODEL")
```

### *Related Topics*

Netlisting Setup for Unified Library Components

Create Extracted View Form

Annotate From Extracted View Form


© 2023 Cadence Design Systems, Inc.

# vsdpSpiceCSVModelNameField

```
asimenv.netlist vsdpSpiceCSVModelNameField string "ASI_MODEL"
```

### Description

Specifies the default value of the part variant field that would be used to identify the SPICE model file associated with the instance. By default, the `ASI_MODEL` part variant field is used to identify the associated model file.

### GUI Equivalent

None

### Examples

```
envSetVal("asimenv.netlist" "vsdpSpiceCSVModelNameField" 'string "ASI_MODEL")
```

#### *Related Topics*

Netlisting Setup for Unified Library Components

Create Extracted View Form

Annotate From Extracted View Form


© 2023 Cadence Design Systems, Inc.

# areaTransferFile

```
vrf.exportDie areaTransferFile string t_transfilename
```

### Description

Specifies an XML file that includes rules to add some additional shapes, such as logos, in an abstract view. By default, no file is specified. Cadence recommends that you use a non-conductor SIP_SHAPE_CLASS layer (such as COMPONENT GEOMETRY) as the target layer in the XML file instead of a conductor layer. SiP Layout Option does not support CONDUCTOR class shapes or clines in die symbols because their corresponding layer might change unpredictably when the die orientation or placement is modified in the stack.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Area Transfer* (tab) – *Area Transfer File* |

### Examples

```
envGetVal("vrf.exportDie" "areaTransferFile")

envSetVal("vrf.exportDie" "areaTransferFile" 'string "transA")
```

### *Related Topics*

Export Die Form

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

## bondFingerAlignment

```
vrf.wirebond bondFingerAlignment cyclic { "along wire" | "orthogonal to die" }
```

### Description

Specifies the bond finger alignment. The default value is `along wire`.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Bond Wire – Create* |
| Field | *Alignment* |

### Examples

```
envGetVal("vrf.wirebond" "bondFingerAlignment")
envSetVal("vrf.wirebond" "bondFingerAlignment" 'cyclic "along wire")
```

### *Related Topics*

Creating Bond Wires and Bond Fingers


© 2023 Cadence Design Systems, Inc.

# bondFingerProfile

```
vrf.wirebond bondFingerProfile string t_def
```

### Description

Specifies a bond finger definition. The default value is `auto`.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Bond Wire – Create* |
| Field | *Definition* |

### Examples

```
envGetVal("vrf.wirebond" "bondFingerProfile")
envSetVal("vrf.wirebond" "bondFingerProfile" 'string "abcd")
```

### *Related Topics*

Creating Bond Wires and Bond Fingers


© 2023 Cadence Design Systems, Inc.

# bumpCenterMismatchCheck

```
vrf.exportDie bumpCenterMismatchCheck boolean { t | nil }
```

### Description

Checks that the center of bump master cellview matches the center of the shape, which is used during die export to create IO/bump in abstract. The default value is `nil`, which means that the check is not performed.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "bumpCenterMismatchCheck" 'boolean t)
envGetVal("vrf.exportDie" "bumpCenterMismatchCheck")
```

### *Related Topics*

Exporting Dies


© 2023 Cadence Design Systems, Inc.

# casDieLayoutVsAbstract

`layoutXL casDieLayoutVsAbstract boolean { t | nil }`

### Description

Compares the die layouts with die abstracts and creates violation markers to indicate violations such as connectivity mismatches and missing, extra, and misaligned IO pads.

The default value is `nil`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL" "casDieLayoutVsAbstract")
envSetVal("layoutXL" "casDieLayoutVsAbstract" 'boolean t)
```

### *Related Topics*

Running the LVA Checker


© 2023 Cadence Design Systems, Inc.

# cdfParamPromptLineNumber

`vrf.exportDie cdfParamPromptLineNumber int` *`x_linenumber`*

### Description

Mentions the line number where the CDF prompt name is specified.

The default value is `2`.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "cdfParamPromptLineNumber" 'int 4)
envGetVal("vrf.exportDie" "cdfParamPromptLineNumber")
```

### *Related Topics*

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# checkerPrecisionFactor

```
vrf.exportDie checkerPrecisionFactor int precision
```

### Description

Specifies the multiplication factor used to calculate the precision for fixing IO pin locations when running the LVA checker and fixer. For example, when set to `2`, the precision is calculated as `2 * 1/1000 = 0.002`, where the DBUPerUU is `1000`.

The default value is `2`.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "checkerPrecisionFactor" 'int 4)
envGetVal("vrf.exportDie" "checkerPrecisionFactor")
```

### *Related Topics*

Running the LVA Checker

Running the LVA Fixer


© 2023 Cadence Design Systems, Inc.

# createNoConn

```
vrf.exportDie createNoConn boolean { t | nil }
```

### Description

Creates a schematic with the noConn symbol connected to the extra pins in IC symbol when set to `t`. The default value is `t`. When set to `nil`, it indicates an error condition due to more pins in IC symbol as compared to the exported die abstract.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "createNoConn" 'boolean nil)
envGetVal("vrf.exportDie" "createNoConn")
```

### *Related Topics*

Exporting Dies


© 2023 Cadence Design Systems, Inc.

# createWireFingerOrBoth

```
vrf.wirebond createWireFingerOrBoth cyclic { "Both" | "Bond Wire" | "Bond Finger"}
```

### Description

Specifies what must be created - `Bond Wire`, `Bond Finger`, or `Both`.

The default value is `Both`.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Bond Wire – Create* |
| Field | *Create* |

### Examples

```
envGetVal("vrf.wirebond" "createWireFingerOrBoth")
envSetVal("vrf.wirebond" "createWireFingerOrBoth" 'cyclic "Update_Symbol")
```

### *Related Topics*

Creating Bond Wires and Bond Fingers


© 2023 Cadence Design Systems, Inc.

# csvFile

```
vrf.exportDie csvFile string t_filename
```

### Description

Specifies the default filename for the *CSV File* field in the Export Die form.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Advanced Settings* (tab) – *CSV File* |

### Examples

```
envSetVal("vrf.exportDie" "csvFile" 'string "CSV_file")
envGetVal("vrf.exportDie" "csvFile" 'string "CSV_file")
```

### *Related Topics*

Export Die Form


© 2023 Cadence Design Systems, Inc.

# customizePin

```
vrf.exportDie customizePin boolean { t | nil }
```

### Description

Enables the *Pin Numbering* tab in the Export Die form. By default, the tab is disabled.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Advanced Settings* (tab) – *Customize pin numbers* (check box) |

### Examples

```
envSetVal("vrf.exportDie" "customizePin" 'boolean t)
envGetVal("vrf.exportDie" "customizePin")
```

#### *Related Topics*

Exporting Dies


© 2023 Cadence Design Systems, Inc.

# deleteViewsBeforeExport

```
vrf.exportDie deleteViewsBeforeExport boolean { t | nil }
```

### Description

When set to `t`, all the die export views of a specified cell in a specified library are deleted before exporting. The default value is `nil`. By default, all the views are overwritten at the time of export.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Advanced Settings* (tab) – *Delete existing views before export* (check box) |

### Examples

```
envSetVal("vrf.exportDie" "deleteViewsBeforeExport" 'boolean t)
envGetVal("vrf.exportDie" "deleteViewsBeforeExport")
```

#### *Related Topics*

vrfExportLayoutSkill

Export Die Form


© 2023 Cadence Design Systems, Inc.

# dieTemplateFile

```
vrf.dieAudit dieTemplateFile string t_filename
```

### Description

Specifies the default filename for the *Template File* field in the Virtuoso RF Compliance Audit form. The default value is `""`, which means that no template file is specified. In this case, the values from the environment variables related to the Export Die form are used.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Virtuoso RF Compliance Audit* |
| Field | *Template File* |

### Examples

```
envGetVal("vrf.dieAudit" "dieTemplateFile")
envGetVal("vrf.dieAudit" "dieTemplateFile" 'string "temp_file")
```

#### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit


© 2023 Cadence Design Systems, Inc.

# distributeObjsOnGuide

```
vrf.wirebond distributeObjsOnGuide cyclic { "Uniform" | "Orthogonal" | "Any Angle"
     }
```

### Description

Specifies how bond fingers and bond wires are to be distributed on a guide. The available options are:

- `Uniform`: Distributes bond fingers and bond wires evenly along the guide.
- `Orthogonal`: Places bond fingers and bond wires orthogonal to the guide. They are placed at the nearest point in the guide, and therefore this setting ensures shortest bond wire length.
- `Any Angle` (default): Uses the `pkgMinBondwireSpacing` constraint value to determine the minimum spacing between adjacent bond fingers or bond wires.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Bond Wire – Create* |
| Field | *Distribute On Guide* |

### Examples

```
envGetVal("vrf.wirebond" "distributeObjsOnGuide")
envSetVal("vrf.wirebond" "distributeObjsOnGuide" 'cyclic "Uniform")
```

### *Related Topics*

Create Bond Wire Form

Bond Wires and Bond Fingers Creation


© 2023 Cadence Design Systems, Inc.

# drdEditApkDrcBlockageOverlapCheck

```
layout drdEditApkDrcBlockageOverlapCheck boolean { t | nil }
```

### Description

Specifies whether DRD checking should consider overlaps of blockages with metal shapes. The default value is `nil`, in which case, such blockages are not considered.

### GUI Equivalent

None

### Examples

```
envGetVal("layout" "drdEditApkDrcBlockageOverlapCheck")
envSetVal("layout" "drdEditApkDrcBlockageOverlapCheck" 'boolean nil)
```

### *Related Topics*

Overlap Checking


© 2023 Cadence Design Systems, Inc.

# drdEditApkDrcComponentOverlap

```
layout drdEditApkDrcComponentOverlap boolean { t | nil }
```

### Description

Specifies whether you want DRD to report violations if the PR boundaries of two instances placed next to each other overlap. The default value is `t`.

### GUI Equivalent

| | |
|---|---|
| Command | *Verify – Design* |
| Field | *Process Rules* (tab) – *Component Overlap Check* (Batch Checker) |

### Examples

```
envGetVal("layout" "drdEditApkDrcComponentOverlap")
envSetVal("layout" "drdEditApkDrcComponentOverlap" 'boolean nil)
```

### *Related Topics*

Overlap Checking


© 2023 Cadence Design Systems, Inc.

# drdEditApkDrcShortCheck

```
layout drdEditApkDrcShortCheck boolean { t | nil }
```

### Description

Specifies whether DRD checking should consider shorts of metals with other metal shapes. The default value is `nil`, in which case, such shorts are not considered.

### GUI Equivalent

None

### Examples

```
envGetVal("layout" "drdEditApkDrcShortCheck")
envSetVal("layout" "drdEditApkDrcShortCheck" 'boolean t)
```

### *Related Topics*

Overlap Checking


© 2023 Cadence Design Systems, Inc.

# exportNoBumpTerm

```
layout exportNoBumpTerm boolean { t | nil }
```

### Description

Includes the top-level IC layout terminals, which do not have the corresponding IO cells in the layout, while exporting the die. The default value is `nil`. By default, the terminals that have corresponding IO cell implementation are exported during die export.

### GUI Equivalent

None

### Examples

```
envSetVal("layout" "exportNoBumpTerm" 'boolean nil)
envGetVal("layout" "exportNoBumpTerm")
```

### Related Topics

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# honorConstraints

```
vrf.wirebond honorConstraints boolean { t | nil }
```

### Description

Specifies whether the constraints or the bond wire profile are given higher priority while defining the settings to create a bond wire. The default value is `nil`, which means that the bond wire profile is given high priority over the constraints.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.wirebond" "honorConstraints" 'boolean t)
envGetVal("vrf.wirebond" "honorConstraints")
```

### Related Topic

Creating Bond Wires and Bond Fingers


© 2023 Cadence Design Systems, Inc.

# icSymbolCheck

```
vrf.dieAudit icSymbolCheck boolean { t | nil }
```

### Description

Specifies whether the IC symbol check is being performed during die audit. The default value is `t`, which means the check is performed.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Virtuoso RF Compliance Audit* |
| Field | *IC Symbol Check* |

### Examples

```
envGetVal("vrf.dieAudit" "icSymbolCheck")
envGetVal("vrf.dieAudit" "icSymbolCheck" 'boolean nil)
```

#### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit


© 2023 Cadence Design Systems, Inc.

# ignoreColumnNumbers

`vrf.exportDie ignoreColumnNumbers string` *`t_columnnumbers`*

### Description

Specifies the column numbers that need to be ignored from a CSV file that is used for die export. The default value is `""`. By default, none of the columns are ignored.

### GUI Equivalent

None

### Examples

```
envGetVal("exportDie" "ignoreColumnNumbers")
envSetVal("exportDie" "ignoreColumnNumbers" 'string "efg2")
```

### *Related Topics*

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# ignoreLineNumbers

```
vrf.exportDie ignoreLineNumbers string t_linenumber
```

### Description

Specifies the line numbers that need to be ignored from a CSV file that is used for die export. The default value is `""`. By default, none of the lines are ignored.

### GUI Equivalent

None

### Examples

```
envGetVal("exportDie" "ignoreLineNumbers")
envSetVal("exportDie" "ignoreLineNumbers" 'string "abc5")
```

### *Related Topics*

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# instTermLabelCheck

```
vrf.dieAudit instTermLabelCheck boolean { t | nil }
```

### Description

Checks labels associated with pins that are overlapping IOs to have same text as the IO terminal name. The default value is `nil`, which means the check is not performed.

### GUI Equivalent

None

### Examples

```
envGetVal("vrf.dieAudit" "instTermLabelCheck")
envGetVal("vrf.dieAudit" "instTermLabelCheck" 'boolean t)
```

### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit


© 2023 Cadence Design Systems, Inc.

# ioCheck

```
vrf.dieAudit ioCheck boolean { t | nil }
```

### Description

Specifies whether the IO check is being performed during die audit. The default value is `t`, which means the check is performed.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Virtuoso RF Compliance Audit* |
| Field | *IO Check* |

### Examples

```
envGetVal("vrf.dieAudit" "ioCheck")
envGetVal("vrf.dieAudit" "ioCheck" 'boolean nil)
```

#### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit


© 2023 Cadence Design Systems, Inc.

# layoutConnectivityFile

```
vrf.wirebond layoutConnectivityFile string t_filename
```

### Description

Specifies a name and the location to save the connectivity file. When you specify a file, it is stored. Every time you open the form in the same or a different Virtuoso session, the form field is populated with this value. The default value is `""`.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Connectivity – Save Connectivity* |
| Field | *Save Connectivity* |

### Examples

```
envGetVal("vrf.wirebond" "layoutConnectivityFile")
envGetVal("vrf.wirebond" "layoutConnectivityFile" 'string "temp_file")
```

### *Related Topics*

Extracting Connectivity Information from Package Layout


© 2023 Cadence Design Systems, Inc.

# layoutConnectivityPinNamesChkBox

```
vrf.wirebond layoutConnectivityPinNamesChkBox boolean { t | nil }
```

### Description

Specifies use of pin names from schematic instead of pin numbers in an instance. The default value is `nil`, which means the check box is not selected.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Connectivity – Save Connectivity* |
| Field | *Use Pin Names From Schematic* |

### Examples

```
envGetVal("vrf.wirebond" "layoutConnectivityPinNamesChkBox")
envGetVal("vrf.wirebond" "layoutConnectivityPinNamesChkBox" 'boolean t)
```

#### *Related Topics*

Extracting Connectivity Information from Package Layout


© 2023 Cadence Design Systems, Inc.

# libraryCheck

```
vrf.dieAudit libraryCheck boolean { t | nil }
```

### Description

Specifies whether the library check is being performed during die audit. The default value is `t`, which means the check is performed.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Virtuoso RF Compliance Audit* |
| Field | *Library Check* |

### Examples

```
envGetVal("vrf.dieAudit" "libraryCheck")
envGetVal("vrf.dieAudit" "libraryCheck" 'boolean nil)
```

#### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit


© 2023 Cadence Design Systems, Inc.

# noConnCell

```
vrf.exportDie noConnCell string "noConn"
```

### Description

Defines the cell name for the `noConn` symbol to be used. The default value is `noConn`.

### GUI Equivalent

None

### Examples

```
envSetVal("exportDie" "noConnCell" 'string "noConn")
envGetVal("exportDie" "noConnCell")
```

### *Related Topics*

Exporting Dies


© 2023 Cadence Design Systems, Inc.

# otherChecks

```
vrf.dieAudit otherChecks boolean { t | nil }
```

### Description

Specifies whether the PR boundary checks are performed during die audit. The default value is `t`, which means the check is performed.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Virtuoso RF Compliance Audit* |
| Field | *Other Checks* |

### Examples

```
envGetVal("vrf.dieAudit" "otherChecks")
envGetVal("vrf.dieAudit" "otherChecks" 'boolean nil)
```

#### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit


© 2023 Cadence Design Systems, Inc.

# outputMode

```
vrf.exportDie outputMode cyclic { "Generate_All" | "Update_Symbol" |
     "Generate_All_Schematic_Optional"}
```

### Description

Specifies how views are updated while exporting a die. `Generate_All` ensures that all views are created during die export. `Update_Symbol` only updates the symbol view and creates the remaining views. `Generate_All_Schematic_Optional` generates all possible views except schematic view, which is generated optionally.

The default value is `Generate_All`.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "outputMode" 'cyclic "Update_Symbol")
envGetVal("vrf.exportDie" "outputMode")
```

### *Related Topics*

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# paramMap

`vrf.exportDie paramMap string` *`t_mappednames`*

### Description

Specifies mapped names for columns in a CSV file that is used for die export. The default value is `""`. By default, none of the columns are mapped.

### GUI Equivalent

None

### Examples

```
envGetVal("vrf.exportDie" "paramMap")
envSetVal("vrf.exportDie" "paramMap" 'string "xyz4")
```

### *Related Topics*

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# paramNameLineNumber

```
vrf.exportDie paramNameLineNumber int "1"
```

### Description

Specifies the line number where the parameter names are specified in the CSV file that is used for die export. The default value is `"1"`.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "paramNameLineNumber" 'int "4")
```

### *Related Topics*

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# partNameToImport

```
vrf.exportDie partNameToImport string t_partcolumnvalue
```

### Description

Imports only those rows from the CSV file where the PART column value matches with the string value defined in the variable. The default value is `""`.

### GUI Equivalent

None

### Examples

```
envGetVal("vrf.exportDie" "partNameToImport")
envSetVal("vrf.exportDie" "partNameToImport" 'string "uvw10")
```

### *Related Topics*

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# pinNumberFile

`vrf.exportDie pinNumberFile string` *`t_dietextfile`*

### Description

Specifies a die text file to customize the pin numbers of the die footprint view. This file can be exported from the SiP Layout Option or generated from Virtuoso. By default, no file is specified.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Pin Numbering* (tab) – *Die Text File* (text field) |

### Examples

```
envSetVal("vrf.exportDie" "pinNumberFile" 'string "abcd.txt")
envGetVal("vrf.exportDie" "pinNumberFile")
```

#### *Related Topics*

Export Die Form


© 2023 Cadence Design Systems, Inc.

# pinOrdering

```
vrf.exportDie pinOrdering string { "Vertical - bottom right" | "Vertical - bottom
     right minimum distance" | "Custom"}
```

### Description

Specifies the predefined algorithm to be used for pin ordering. The default value is `Vertical - bottom right`. The following is the description of the valid values of the environment variable:

- `Vertical - bottom right` value ensures that the pin at the bottom-right corner is considered as the first pin. The other pins are placed column-wise from bottom to top.
- `Vertical - bottom right minimum distance` indicates that the pin located at the minimum diagonal distance from the bottom right of the die is considered as the first pin. The other pins are placed column-wise from bottom to top.
- `Custom` lets you can define a custom algorithm by using SKILL.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Pin Numbering* (tab) – *Ordering* (drop-down list) |

### Examples

```
envGetVal("vrf.exportDie" "pinOrdering")
envSetVal("vrf.exportDie" "pinOrdering" 'string "Vertical – bottom right minimum
distance")
```

### *Related Topics*

Exporting Dies

Export Die Form

pinOrderingCustom


© 2023 Cadence Design Systems, Inc.

# pinOrderingCustom

```
vrf.exportDie pinOrderingCustom string t_custom_algorithm_name
```

### Description

Specifies name of a custom algorithm to be used for pin ordering. This value is read only when `pinOrdering` is set to `Custom`. By default, pins are ordered by

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Pin Numbering* (tab) – *Ordering* (drop-down list) |

### Examples

```
envGetVal("vrf.exportDie" "pinOrderingCustom")
envSetVal("vrf.exportDie" "pinOrderingCustom" 'string "algorithm2")
```

#### *Related Topics*

Exporting Dies

Export Die Form

pinOrdering


© 2023 Cadence Design Systems, Inc.

# sameTechnologyAbstract

```
vrf.exportDie sameTechnologyAbstract boolean { t | nil }
```

### Description

Creates Integrity 3D-IC compatible die abstract.

The default value is `nil`, which means Virtuoso TILP die abstract is created.

### GUI Equivalent

None

### Examples

```
envGetVal("vrf.exportDie" "sameTechnologyAbstract")
envSetVal("vrf.exportDie" "sameTechnologyAbstract" 'boolean t)
```

### *Related Topics*

Virtuoso Integrity 3D-IC Flow

Creating and Verifying Integrity 3D-IC Compatible Die Abstracts


© 2023 Cadence Design Systems, Inc.

## schematicConnectivityFile

`vrf.wirebond schematicConnectivityFile string t_connfile`

### Description

Specifies the name of the connectivity file. The default value is `""`.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Connectivity – Create Connectivity* |
| Field | *Connectivity File* |

### Examples

```
envSetVal("vrf.wirebond" "schematicConnectivityFile" 'string "confile.txt")
envGetVal("vrf.wirebond" "schematicConnectivityFile")
```

### *Related Topics*

Creating and Saving Connectivity Information in Package Schematic

schematicConnectivityMode


© 2023 Cadence Design Systems, Inc.

# schematicConnectivityMode

```
vrf.wirebond schematicConnectivityMode cyclic { "Replace" | "Merge" }
```

### Description

Specifies the mode of the creating the connectivity in a schematic. The available options are:

- `Replace` (default): The entire connectivity is deleted before creating the a new one based on the input file.
- `Merge`: Only the connectivity specified in the input file is appended to the existing connectivity.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Connectivity – Create Connectivity* |
| Field | *Mode* |

#### ***Related Topics***

Creating and Saving Connectivity Information in Package Schematic

schematicConnectivityFile


© 2023 Cadence Design Systems, Inc.

# segSnapMode

```
vrf.layout segSnapMode string "anyAngle"
```

### Description

Determines how *Edit* command entry points snap to the grid for a design that has the fabric type as package, module, or board. The default value is `anyAngle`. By default, the points snap to grid at any angle.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.layout" "segSnapMode" 'string "anyAngle")
envGetVal("vrf.layout" "segSnapMode")
```


© 2023 Cadence Design Systems, Inc.

# shortPinCell

```
vrf.exportDie shortPinCell string "cds_thru"
```

### Description

Specifies the cell that contains the pin instance symbol for shorting multiple nets. The default value is `"cds_thru"`.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "shortPinCell" 'string "short_pin")
envGetVal("vrf.exportDie" "shortPinCell")
```

#### *Related Topics*

vrfExportPackage


© 2023 Cadence Design Systems, Inc.

# shortPinLib

```
vrf.exportDie shortPinLib string "basic"
```

### Description

Specifies the library that contains the pin instance symbol for shorting multiple nets. The default value is `basic`.

### GUI Equivalent

None

### Examples

```
envSetVal("vrf.exportDie" "shortPinLib" 'string "pin_basic")
envGetVal("vrf.exportDie" "shortPinCell")
```

### *Related Topics*

vrfExportPackage


© 2023 Cadence Design Systems, Inc.

# snapEndPoint

```
vrf.wirebond snapEndPoint boolean { t | nil }
```

### Description

Specifies the snapping behavior of bond wires. The default value is `t`, which means the option is selected.

### GUI Equivalent

None

### Examples

```
envGetVal("vrf.wirebond" "snapEndPoint")
envSetVal("vrf.wirebond" "snapEndPoint" 'boolean nil)
```

### *Related Topics*

Moving Bond Wires and Bond Fingers


© 2023 Cadence Design Systems, Inc.

# snapMode

```
vrf.layout snapMode string "anyAngle"
```

### Description

Determines how *Create* command entry points snap to the grid for a design that has fabric type package, module, or board. The default value is `anyAngle`. By default, the points snap to grid at any angle.

### GUI Equivalent

None

### Examples

```
envSetVal("layout" "snapMode" 'string "anyAngle")
envGetVal("layout" "snapMode")
```


© 2023 Cadence Design Systems, Inc.

# termsCheck

```
vrf.dieAudit termsCheck boolean { t | nil }
```

### Description

Specifies whether terminals checks are performed during die audit. The default value is `t`, which means the check is performed.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Virtuoso RF Compliance Audit* |
| Field | *Terms Check* |

### Examples

```
envGetVal("vrf.dieAudit" "termsCheck")
envGetVal("vrf.dieAudit" "termsCheck" 'boolean nil)
```

#### *Related Topics*

Virtuoso RF Compliance Audit Form

vrfComplianceAudit


© 2023 Cadence Design Systems, Inc.

# transferArea

```
vrf.exportDie transferArea boolean { t | nil }
```

### Description

Enables the *Area Transfer* tab in the Export Die form. By default, the tab is disabled.

### GUI Equivalent

| | |
|---|---|
| Command | *Module – Export Die* |
| Field | *Advanced Settings* (tab) – *Transfer area* (check box) |

### Examples

```
envSetVal("vrf.exportDie" "transferArea" 'boolean t)
envGetVal("vrf.exportDie" "transferArea")
```

### *Related Topics*

Export Die Form

vrfExportLayoutSkill


© 2023 Cadence Design Systems, Inc.

# voidGeneratorTrimUnconnectedShapes

```
layoutXL voidGeneratorTrimUnconnectedShapes boolean { t | nil }
```

### Description

Trims conducting shapes that are not connected to a pin. If no conducting shape is connected to a pin, the largest conducting shape is retained.

The default value is `nil`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL" "voidGeneratorTrimUnconnectedShapes")
envSetVal("layoutXL" "voidGeneratorTrimUnconnectedShapes" 'boolean t)
```

### *Related Topics*

Smooth and Trim Void Shapes


© 2023 Cadence Design Systems, Inc.

© 2023 Cadence Design Systems, Inc.

# C

# Virtuoso Multi-Technology Solution SKILL Functions

This topic describes the public SKILL functions in the Virtuoso Multi-Technology Solution.

- vmtCompareSipToOa
- vmtcsvCreateComponentCellViewsFromCsv
- vmtcsvInstallCsvFile
- vmtLibImport
- vmtValidateUnifiedLibrary
- vrfCheckTILPVersion
- vrfComplianceAudit
- vrfExportLayoutSkill
- vrfExportPackage
- vrfLowerPriority
- vrfHiUpdate
- vrfRaisePriority
- vrfSipSet
- vrfSipGet
- vrfTLineAbut
- vrfUpdateTILPVersion


© 2023 Cadence Design Systems, Inc.

# vmtCompareSipToOa

```
vmtCompareSipToOa(
     t_sipFile
     [?refLib t_refLib]
     [?refCell t_refCell]
     [?refView t_refView]
     [?destLib t_destLib]
     [?destCell t_destCell]
     [?destView t_destView]
     [?aperture x_aperture]
     [?createStatic g_createStatic]
     [?trueMarkerShapes g_trueMarkerShapes]
     )
     => t / nil
```

### Description

Compares the contents of the specified Allegro SiP layout file and Virtuoso module layout and generates a new layout cellview containing only the parts of the two designs that are different. This is done by performing an XOR boolean operation for an Allegro SiP layout file and a Virtuoso module layout (OpenAccess layout) to create a new layout with XOR shapes and the merged shapes from the SiP layout and OpenAccess layout or database. The comparison is done with the saved representation of the Virtuoso layout not the unsaved OpenAccess database in memory.


© 2023 Cadence Design Systems, Inc.

### Arguments

| | |
|---|---|
| *t_sipFile* | The name of the SiP file to be compared. |
| ?refLib *t_refLib* | The library name of a Virtuoso layout to be compared with the specified SiP file. The default value is the library containing the cellview that is open in the current window. |
| ?refCell *t_refCell* | The cell name of a Virtuoso layout to be compared with the specified SiP file. The default value is the cell containing the cellview that is open in the current window. |
| ?refView *t_refView* | The view name of a Virtuoso layout to be compared with the specified SiP file. The default value is the view that is open in the current window. |
| ?destLib *t_destLib* | The library name of the new layout to be generated. The default value is the value of the `refLib` argument. |
| ?destCell *t_destCell* | The cell name of the new layout that will be generated. The default value is `xor_diff`. |
| ?destView *t_destView* | The view name of the new layout that will be generated. The default value is `layout`. |
| ?aperture *x_aperture* | The radius in microns of a circular aperture used to remove small shapes from the generated layout. Shapes smaller than the specified aperture are not visible in the generated layout. The default value is `5`. |
| ?createStatic *g_createStatic* | Creates static shapes from the Virtuoso layout at runtime and compares them with the shapes specified in the SiP file by default. When set to `nil`, creates void shapes from the SiP file and compares these with the shapes specified in the Virtuoso layout. |
| ?trueMarkerShapes *g_trueMarkerShapes* | Uses the XOR-operation-generated shapes in a Virtuoso layout as markers by default. When set to `nil`, creates a rectangle enclosing the XOR-operation-generated shapes in an OpenAccess layout. The default is `t`. |


© 2023 Cadence Design Systems, Inc.

### Value Returned

| | |
|---|---|
| `t` | Comparison was successful and the comparison cellview was generated. |
| `nil` | Comparison failed. |

### Example

```
vmtCompareSipToOa "Test.sip" ?refLib "oaLib" ?refCell "oaCell" ?refView "layout"
?destLib "oaLib" ?destCell "xor_diff" ?destView "layout_dest" ?aperture "5"
?createStatic t
```

This example compares the `Test.sip` file with the specified Virtuoso layout and generates a new layout `oaLib` with the result of XOR operation of all layers in the SiP layout. The comparison is done with the saved representation of the OpenAccess layout not the unsaved OpenAccess database in memory.

```
vmtCompareSipToOa "Test.sip"
```

This example uses the cellview that is open in the current window as the reference cellview and creates a cellview in the reference library with the cell name `xor_diff` and view name `layout`.


© 2023 Cadence Design Systems, Inc.

# vmtcsvCreateComponentCellViewsFromCsv

```
vmtcsvCreateComponentCellViewsFromCsv(
     t_templateLibName
     t_templateCellName
     t_footPrintLibName
     t_csvFileName
     x_cdfParamNameLineNumber
     t_destLibName
     t_destCellName
     [?ignoreLineNumbers t_ignoreLineNumbers]
     [?ignoreColumnNumbers t_ignoreColumnNumbers]
     [?cdfParamPromptLineNumber x_cdfParamPromptLineNumber]
     [?partNameToImport t_partNameToImport]
     [?termMap t_termMap]
     [?termOrder t_termOrder]
     [?paramMap t_paramMap]
     [?sparamModel g_sparamModel]
     [?spiceModel g_spiceModel]
     [?overwriteSymbol g_overwriteSymbol]
     [?mode t_mode]
     [?footPrintViewName t_footPrintViewName]
     [?draDir t_draDir]
     [?pinNameToNumberFileName t_pinNameToNumberFileName]
     [?class t_class]
     [?sipFileName t_sipFileName]
     )
     => t / nil
```

### Description

Imports a CSV file and creates CDF parameters for the column values in the CSV file and symbol, TILP, and other simulation views.


© 2023 Cadence Design Systems, Inc.

### Arguments

*t_templateLibName* — The template library name from where the symbol and other simulation views with aesthetics are copied.

*t_templateCellName* — The template cell name in the template library from where the symbol and other simulation views with aesthetics are copied.

*t_footPrintLibName* — The name of the library that contains the footprint symbols. This is specified to check if `footprintCellName` is available.

*t_csvFileName* — The CSV file to be processed.

*x_cdfParamNameLineNumber*

The line number of the CSV file that is treated as the header row. You can specify only one line number.

*t_destLibName* — The name of the library where the component cellviews need to be created. If the library does not exist, it is created automatically.

*t_destCellName* — The name of the cell in *t_destLibName*, where the component cellviews are created.

`?ignoreLineNumbers` *t_ignoreLineNumbers*

A space- or comma-separated string of line numbers that are ignored. The default value is `""`.

`?ignoreColumnNumbers` *t_ignoreColumnNumbers*

A space- or comma-separated string of column numbers that are ignored. The default value is `""`.

`?cdfParamPromptLineNumber` *x_cdfParamPromptLineNumber*

The line number of the CSV file, which is considered as a prompt for CDF properties. If not specified, the header is considered as the prompt. Only one line number can be specified. The default value is `-1`.

`?partNameToImport` *t_partNameToImport*

The PART column values to be copied. If this argument is specified, only the rows that have the PART column values matching the *t_partNameToImport* are copied to the processed CSV file. The default value is `""`.

`?termMap` *t_termMap*


© 2023 Cadence Design Systems, Inc.

The term mapping for the terminals. For example, if a symbol has terminals as PLUS and MINUS and you want to update them to A and B, specify the term mapping as "`PLUS A MINUS B`". The default value is `""`.

`?termOrder` *`t_termOrder`*

The order of terminals. The default value is `""`.

`?paramMap` *`t_paramMap`*

The mapped name for each column. The default value is `""`.

`?sparamModel` *`g_sparamModel`*

Specifies whether the `sparamModel` view should be created. The default value is `t`.

`?spiceModel` *`g_spiceModel`*

Specifies whether the `spiceModel` view should be created. The default value is `t`.

`?overwriteSymbol` *`g_overwriteSymbol`*

Specifies whether the existing symbol and simulation views can be overwritten. The default value is `nil`.

`?mode` *`t_mode`*

The mode in which the CSV file is updated. If the value of this argument is `a`, the existing CSV file is opened in append mode and data is added to the end of the file. The header data of the existing CSV file and `csvFileName` must match. If the value of this argument is `w`, the existing CSV file is overwritten. The default value is `w`.

`?footPrintViewName` *`t_footPrintViewName`*

The name of the footprint view. The default value is `base`.

`?draDir` *`t_draDir`*

The directory containing the DRA files. The default value is `""`.

`?pinNameToNumberFileName` *`t_pinNameToNumberFileName`*

The file that contains the mapping information between pin names and numbers. The default value is `""`.

`?class` *`t_class`*


© 2023 Cadence Design Systems, Inc.

The placement class of the compDef object to be created. The default value is `DISCRETE`.

`?sipFileName` *`t_sipFileName`*

The name of the SiP file used in the Virtuoso Multi-Technology Solution. The default value is `""`.

### Value Returned

| | |
|---|---|
| `t` | CSV import is successful. |
| `nil` | CSV import failed. |

### Example

```
vmtcsvCreateComponentCellViewsFromCsv("sipLibTemplate" "ind" "jedecLib1" "./
ind_jedec_mojito.csv" 1 "testlib9" "ind")
```

Copies the symbol and simulation views from `sipLibTemplate/ind` to `testlib9/ind` and creates the `part.csv` file in `testlib9/ind` along with the CDF parameters.

#### *Related Topic*

Variant Definitions in CSV Files


© 2023 Cadence Design Systems, Inc.

# vmtcsvInstallCsvFile

```
vmtcsvInstallCsvFile(
     t_destLibName
     t_destCellName
     t_csvFileName
     x_cdfParamNameLineNumber
     [?ignoreLineNumbers t_ignoreLineNumbers]
     [?ignoreColumnNumbers t_ignoreColumnNumbers]
     [?cdfParamPromptLineNumber x_cdfParamPromptLineNumber]
     [?partNameToImport t_partNameToImport]
     [?paramMap t_paramMap]
     [?mode t_mode]
     )
     => t / nil
```

### Description

Installs the CSV file into the destination cellview and creates the CDF parameters corresponding to the columns in the CSV file.


© 2023 Cadence Design Systems, Inc.

### Arguments

*t_destLibName* — The name of the library where the CSV file will be installed. If the library does not exist, it is created automatically.

*t_destCellName* — The name of the cell in `destLibName`, where the CSV file will be installed. If the library does not exist, it is created automatically.

*t_csvFileName* — The CSV file to be processed and installed.

*x_cdfParamNameLineNumber*

The line number of the CSV file that will be treated as the header row. You can specify only one line number.

`?ignoreLineNumbers` *t_ignoreLineNumbers*

A space- or comma-separated string of line numbers that are ignored. The default value is `""`.

`?ignoreColumnNumbers` *t_ignoreColumnNumbers*

A space- or comma-separated string of column numbers that are ignored. The default value is `""`.

`?cdfParamPromptLineNumber` *x_cdfParamPromptLineNumber*

The line number of the CSV file that is considered as the prompt for CDF properties. If this argument is not specified, the header is considered as the prompt. Only one line number can be specified. The default value is `-1`.

`?partNameToImport` *t_partNameToImport*

If this argument is specified, only the rows that have the PART column values matching the `partNameToImport` are copied to the processed CSV file. The default value is `""`.

`?paramMap` *t_paramMap*

The mapped name for each column. The default value is `""`.

`?mode` *t_mode*

The mode in which the CSV file is updated. If the value of this argument is `a`, the existing CSV file is opened in append mode and data is added to the end of the file. The header data of the existing CSV file and `csvFileName` must match. If the value of this argument is `w`, the existing CSV file is overwritten. The default value is `w`.


© 2023 Cadence Design Systems, Inc.

### Value Returned

| | |
|---|---|
| `t` | The CSV file was installed successfully. |
| `nil` | The CSV file was not installed. |

### Example

```
vmtcsvInstallCsvFile("testlib" "ind1" "./ind_jedec_mojito.csv" 1
?cdfParamPromptLineNumber 2)
```

Creates a cellview `testlib/ind1` and a processed CSV file, `part.csv`.

#### *Related Topic*

Variant Definitions in CSV Files


© 2023 Cadence Design Systems, Inc.

# vmtLibImport

```
vmtLibImport(
     t_inputFileName
     t_libName
     [?importDra g_importDra]
     [?draSMDImportMode g_draSMDImportMode]
     [?draNoSMDImportMode g_draNoSMDImportMode]
     [?refLibName t_refLibName]
     [?importTechOnly g_importTechOnly]
     [?component t_component]
     [?overwrite g_overwrite]
     [?createSymbols g_createSymbols]
     )
     => t / nil
```

### Description

Creates a library containing technology and components from the specified SiP layout file. The library contains TILPs, footprints, and schematic symbols for dies, packages, SMDs, embedded components, and padstacks in a SiP layout file. When importing a layout from the Allegro platform to Virtuoso Studio, the ratio of DataBase Unit Per User Unit remains 1:1. This means the resolution of a layout remains during the import. While creating the library, if files or cellviews are checked in, `autocheckin` is added as a comment by the Library Manager.


© 2023 Cadence Design Systems, Inc.

### Arguments

| | |
|---|---|
| `t_inputFileName` | The name of an SiP layout file. |
| `t_libName` | The name of the library that is created on import. |

`?importDra g_importDra`

Specifies whether symbol definitions are to be imported. The default value is `nil`.

`?draSMDImportMode g_draSMDImportMode`

Specifies whether only SMDs are to be imported. The default value is `nil`.

`?draNoSMDImportMode g_draNoSMDImportMode`

Specifies whether SMDs are to be excluded during import. The default value is `nil`.

`?refLibName t_refLibName`

The name of a library containing schematic symbols that can be copied into the new library. The default value is `""`.

`?importTechOnly g_importTechOnly`

Specifes whether only technology information will e imported from a `.sip` file. The default value is `nil`.

`?component t_component`

The name of a component to be import from a `.sip` file. The default value is `nil`.

`?overwrite g_overwrite`

Specifies whether to overwrite the contents of an existing library. The default value is `nil`.

`?createSymbols g_createSymbols`

Specifies whether to create schematic symbols by generating from the footprint and implied compDef or by copying from a reference library. The default value is `nil`.

### Value Returned

| | |
|---|---|
| `t` | The library was imported successfully. |


© 2023 Cadence Design Systems, Inc.

| | |
|---|---|
| `nil` | The library could not be imported. |

### Example

To create a library and populate it with TILPs, component definitions, CSV files, footprints, and schematic symbol views, you must call two separate invocations of the `vmtLibImport` SKILL function as follows:

- `vmtLibImport("my.sip" "mylib" "?importTechOnly" t)`

  Uses the existing Allegro translator SKILL functions to create the library and the technology file. Ensure to save the data created by running the command before you proceed.

- `vmtLibImport("my.sip" "mylib" "?importDra" t "?createSymbols" t)`

  Uses the `vmtLibImport` SKILL function to read the SiP layout file and populates the library with the components in the file.

### *Related Topic*

Views in the Unified Library


© 2023 Cadence Design Systems, Inc.

# vmtValidateUnifiedLibrary

```
vmtValidateUnifiedLibrary(
     [ ?libName t_libName ]
     [ ?cellView x_cellView ]
     [ ?viewList t_viewList ]
     )
     => numErrors / nil
```

### Description

Validates the existence of required views and cross-view consistency in the cells of a library or the master libraries for all the instances in a design. If a library name is provided, all the cells in the library that match the specified list of views are opened and added to a list for validation. If a cellview is specified, the specific cell is added to a list for validation. If no library name or cellview is provided, no validation is performed.


© 2023 Cadence Design Systems, Inc.

### Arguments

`?libName` *`t_libName`*

Specifies the name of the library to be validated. By default, no libraries are validated.

`?cellView` *`x_cellView`*

Specifies the db ID of the currently open cellview to be validated. By default, no cellview is opened.

`?viewList` *`t_viewList`*

Specifies the list of views to be validated. The default view list is `'("layout" "schematic")`.

### Value Returned

| | |
|---|---|
| *`numErrors`* | The validation was completed successfully and the returned number of errors was found. |
| `nil` | The validation could not be done. |

### Examples

- ```
  cv = dbOpenCellViewByType("myLib" "myCell" "myView" "maskLayout" "r")
  ```

  ```
  vmtValidateUnifiedLibrary ?cellView cv
  ```

  Validates all the cells referenced in a currently open cellview.

- ```
  vmtValidateUnifiedLibrary ?libName "myLib" ?viewList '("layout")
  ```

  Validates all the cells referenced in a library but limiting the view type to `layout`.

### *Related Topic*

Unified Library Validation


© 2023 Cadence Design Systems, Inc.

# vrfCheckTILPVersion

```
vrfCheckTILPVersion(
     x_dbId
     [ ?reportAll g_reportAll ]
     [ ?fileName g_fileName ]
     )
     => t / nil
```

### Description

Checks the version of the TILP from the given ID, which can be of an instance, cellview, library, or sub-master cellview. There are warnings and errors generated when checks are performed. Warning messages are issued for the TILP versions that are compatible with the current version of the tool. You need to upgrade the TILP only if the associated features are going to be used. For example, if you want to use a new feature, the die TILP needs to be updated. Error messages indicate that the TILP versions must be updated before use in the current version of the tool. Otherwise, the tool does not work as expected. For example, die TILP V1 is not compatible with the tool version post ICADVM18.1ISR10.

### Arguments

| | |
|---|---|
| *x_dbId* | The ID of an instance, cellview, library, or sub-master cellview. |
| *?reportAll g_reportAll* | When set to `nil`, reports the cells that have TILP version lower than the current tool version. When set as `t`, all TILP versions are printed. |
| *?fileName g_fileName* | The name of the log file. |

### Value Returned

| | |
|---|---|
| `t` | The version of TILP is reported successfully. |
| `nil` | The version of the TILP could not be reported. |

### Examples

- Instance

  ```
  vrfCheckTILPVersion( instId )
  ```

- Design


© 2023 Cadence Design Systems, Inc.

```
vrfCheckTILPVersion( geGetEditCellView() )
vrfCheckTILPVersion( geGetEditCellView()  ?reportAll t )
```

Output:

```
Latest TILP versions
         Die                  : 3
         Package              : 4
         EmbeddedComponent    : 4
         OtherPadStack        : 2
         SMD                  : 3
         ViaPadStack          : 2

WARNING: "PAMODULE_BGA_FC_VRF_example"/"SMD_CAP"/"layout" of type "SMD"
(Current: 2 - Latest: 3)
WARNING: "PAMODULE_BGA_FC_VRF_example"/"SMD_IND"/"layout" of type "SMD"
(Current: 2 - Latest: 3)
WARNING: "PAMODULE_BGA_FC_VRF_example"/"SMD_RES"/"layout" of type "SMD"
(Current: 2 - Latest: 3)
WARNING: "PAMODULE_BGA_FC_VRF_example"/"MLIN"/"layout" of type
"EmbeddedComponent"  (Current: 3 - Latest: 4)
WARNING: "PAMODULE_BGA_FC_VRF_example"/"IND_L2"/"layout" of type
"EmbeddedComponent"  (Current: 3 - Latest: 4)
WARNING: "PAMODULE_BGA_FC_VRF_example"/"MLIN_0"/"layout" of type
"EmbeddedComponent"  (Current: 3 - Latest: 4)
WARNING: "PAMODULE_BGA_FC_VRF_example"/"COUPLER"/"layout" of type
"EmbeddedComponent"  (Current: 3 - Latest: 4)
WARNING: "PAMODULE_BGA_FC_VRF_example"/"MULTILAYERCAP0"/"layout" of type
"EmbeddedComponent"  (Current: 3 - Latest: 4)
ERROR: "PAMODULE_BGA_FC_VRF_example"/"BGA17"/"layout" of type "Package" must
be updated (Current: 2 - Latest: 4)
ERROR: "PAMODULE_BGA_FC_VRF_example"/"MMIC_VRF_2018_PKG"/"layout" of type
"Die" must be updated (Current: 1 - Latest: 3)
WARNING: 8 warnings found. Run vrfUpdateTILPVersion to update the TILP
versions.
ERROR: 2 errors found. Run vrfUpdateTILPVersion to update the TILP versions.
```

- Library

```
vrfCheckTILPVersion(  ddGetObj("murataLib") )
```

- Other Examples

```
vrfCheckTILPVersion( geGetEditCellView()  ?fileName "checkReport.txt" )
```

### *Related Topic*

TILP Versions


© 2023 Cadence Design Systems, Inc.

# vrfComplianceAudit

```
vrfComplianceAudit(
     g_cellViewId
     [ ?dieAuditTemplateFile t_dieAuditTemplateFile ]
     [ ?libCheck g_libCheck ]
     [ ?termsCheck g_termsCheck ]
     [ ?icSymbolCheck g_icSymbolCheck ]
     [ ?ioCheck g_ioCheck ]
     [ ?otherChecks g_otherChecks ]
     )
     => t / nil
```

### Description

Performs various preliminary checks on an IC layout with the use of a template file.

### Arguments

*g_cellViewId* The cellview ID of the layout to be used for auditing.

?dieAuditTemplateFile *t_dieAuditTemplateFile*

Name of the file that contains die export settings. These settings are used to find IOs in an IC layout. If this file is not specified, the environment variables for Export Die form are used for the IO search. The default value is `""`.

?libCheck *g_libCheck*

Specifies whether to run the library check. The default value is `t`.

?termsCheck *g_termsCheck*

Specifies whether to run the terminals check. The default value is `t`.

?icSymbolCheck *g_icSymbolCheck*

Specifies whether to run the IC symbol check. The default value is `t`.

?ioCheck *g_ioCheck*

Specifies whether to run the IO check. The default value is `t`.

?otherChecks *g_otherChecks*


© 2023 Cadence Design Systems, Inc.

Specifies whether to run other miscellaneous checks for PR boundary. The default value is `t`.

### Value Returned

| | |
|---|---|
| `t` | The die audit was successful. |
| `nil` | Die audit failed. |

### Examples

```
vrfComplianceAudit(geGetEditCellView() ?dieAuditTemplateFile "file1" ?libCheck
nil)
```

Runs die audit for a given cellview using the template file "file1" for its settings. The settings in the template file define how to find IOs in the IC layout. In this example 'libCheck' is set to 'nil', library check will not be done.

Export it as follows:

```
vrfComplianceAudit(geGetEditCellView())
```

Runs die audit for a given cellview using the environment variable settings for export die to find IOs in the IC layout. All the checks will be performed.

### *Related Topics*

Die Audit

Virtuoso RF Compliance Audit Form


© 2023 Cadence Design Systems, Inc.

# vrfExportLayoutSkill

```
vrfExportLayoutSkill(
     g_cellViewId
     t_frontPinLayer
     t_backPinLayer
     [ ?dieInterfaceType t_dieInterfaceType ]
     [ ?includeUnconnectedBump g_includeUnconnectedBump ]
     [ ?pinNumbering t_pinNumbering ]
     [ ?pinNumberFile t_pinNumberFile ]
     [ ?abstractLib t_abstractLib ]
     [ ?attachSParam g_attachSParam ]
     [ ?sParamView t_sParamView ]
     [ ?modelFile t_modelFile ]
     [ ?shortPinLib t_shortPinLib ]
     [ ?shortPinCell t_shortPinCell ]
     [ ?frontLabelLPP t_frontLabelLPP ]
     [ ?backLabelLPP t_backLabelLPP ]
     [ ?csvFileName t_csvFileName ]
     [ ?areaTransferFile t_areaTransferFile ]
     [ ?abstractViewCell t_exportNoBumpTerm ]
     [ ?exportNoBumpTerm g_exportNoBumpTerm ]
     [ ?deleteViewsBeforeExport g_deleteViewsBeforeExport ]
     [ ?outputMode t_outputMode ]
     )
     => t / nil
```

### Description

Exports the die for IC layouts. The function supports both `IO cell` and `Shape with overlapping label` as *Die Interface Type*. It generates a package type library that contains abstract, Technology Independent Layout Pcells (TILPs), schematic, and symbol views. The abstract view is the base cellview for the Pcell generated layout, that is, TILP. The schematic is the wrapper schematic for the IC symbol. The symbol view is used to instantiate the generated die symbol in a package schematic.


© 2023 Cadence Design Systems, Inc.

### Arguments

*g_cellViewId* — Cellview ID of the layout to be used for exporting the die.

*t_frontPinLayer* — Front pin layer-purpose pair.

*t_backPinLayer* — Back pin layer-purpose pair.

[ ?dieInterfaceType *t_dieInterfaceType* ]

The valid values are `IO cell` (default) and `SHAPEANDLABEL`.

[ ?includeUnconnectedBump *g_includeUnconnectedBump* ]

When set to `t`, the unconnected bumps are included in the exported abstract view. The valid values are `t` or `nil` (default).

[ ?pinNumbering *t_pinNumbering* ]

The valid values are `"Use pin names as numbers"` (default), `"Use numbers as-is"`, or `"Use numbers from file"`.

[ ?pinNumberFile *t_pinNumberFile* ]

The die text file containing pin numbers for the IO cells that are used in the IO cells-based layout or pin numbers for the labels to be used in the shape-based layout. The default value is `""`.

Example:

```
?pinNumberFile "./sampleFile"
```

[ ?abstractLib *t_abstractLib* ]

Name of the package library of the abstract cellview. The default value is `"die_footprint"`.

[ ?attachSParam *g_attachSParam* ]

Specifies whether an s-parameter model needs to be generated. The default value is `nil`.

[ ?sParamView *t_sParamView* ]

Name of the s-parameter view. The default value is `"sparamModel"`.

[ ?modelFile *t_modelFile* ]

Model file to be used for the s-parameter model creation. The default value is `""`.

[ ?shortPinLib *t_shortPinLib* ]


© 2023 Cadence Design Systems, Inc.

Library that contains the pin instance symbol for shorting multiple nets. The default value is `"basic"`.

`[ ?shortPinCell` *`t_shortPinCell`* `]`

Cell that contains the pin instance symbol for shorting multiple nets. The default value is `"cds_thru"`.

`[ ?frontLabelLPP` *`t_frontLabelLPP`* `]`

Front label layer-purpose pair required when exporting shape-based layout when die interface type is `SHAPEANDLABEL`. The default value is `""`.

`[ ?backLabelLPP` *`t_backLabelLPP`* `]`

Back label layer-purpose pair. It needs to be specified for exporting shape-based layout when die interface type is `SHAPEANDLABEL`. The default value is `""`.

`[ ?csvFileName` *`t_csvFileName`* `]`

A CSV file for customizing or updating the CDF parameters during the die export.

By default, a CSV file with the following essential columns is created:

```
footprintCellName,DEVICE_TYPE
Footprint CellName:,DEVICE_TYPE
<abstractViewCell> <abstractViewCell>
```

Example:

```
footprintCellName,DEVICE_TYPE,CDS_DEVICE_TYPE,
altFootPrintCellNames, ASI_MODEL, PARENT_PART_TYPE,
PARENT_PPT, PARENT_PPT_PART, PART_NAME
```

`[ ?areaTransferFile` *`t_areaTransferFile`* `]`

An XML file that includes rules to add some additional shapes in an abstract view, such as logos. Cadence recommends that you use a non-conductor SIP_SHAPE_CLASS layer (such as COMPONENT GEOMETRY) as the target layer in the XML file instead of a conductor layer. SiP Layout Option does not support CONDUCTOR class shapes or clines in die symbols because their corresponding layer might change unpredictably when the die orientation or placement is modified in the stack.

The default value of the argument is `""`.


© 2023 Cadence Design Systems, Inc.

`[ ?abstractViewCell` *`t_abstractViewCell`* `]`

The given name of the abstract cellview. By default, the original IC cellview and the abstract cellview have the same name.

`[ ?exportNoBumpTerm` *`g_exportNoBumpTerm`* `]`

Includes the top-level IC layout terminals, which do not have the corresponding IO cells in the layout, while exporting the die. By default, the terminals that have corresponding IO cell implementation are exported during die export. This argument is set by `exportNoBumpTerm`.

`[ ?deleteViewsBeforeExport` *`g_deleteViewsBeforeExport`* `]`

Deletes all the die export views of the specified cell in the specified library before exporting. This argument is set by `deleteViewsBeforeExport`.

`[ ?outputMode` *`t_outputMode`* `]`

Specifies how views will be updated while exporting a die. `Generate_All` ensures that all views are created during die export. This argument is set by `outputMode`.

### Value Returned

| | |
|---|---|
| `t` | Die export is successful. |
| `nil` | Die export failed. |

### Examples

```
vrfExportLayoutSkill(
                geGetEditCellView()
                "Metal11 drawing"
                ""
                ?dieInterfaceType "IO cell"
                ?includeUnconnectedBump nil
                ?pinNumbering "Use pin names as numbers"
                ?pinNumberFile ""
                ?abstractLib "die_footprint"
                ?ecoMode nil
                ?attachSParam nil
                ?sParamView "sparamModel"
```


© 2023 Cadence Design Systems, Inc.

```
            ?modelFile ""
            ?shortPinLib  "basic"
            ?shortPinCell "cds_thru"
            ?frontLabelLPP ""
            ?backLabelLPP ""
        )
```

The function exports die in the following two different flows.

- IO Cell Based

  ```
  IOvrfExportLayoutSkill(cv "M4 drawing" "" ?abstractLib
  "die_footprint" ?dieInterfaceType "IO cell")
  ```

  Output:

  ```
  Export Die Report (PA_BGA_FC_PKG_TOP/PAMODULE_BGA_FC_VRF/layout)

  ****************************************************************

  Connected ioCells = 21

            Front side ioCells = 21

            Back side ioCells = 0

  Did not look for unconnected ioCells.

  ****************************************************************
  ```

- Shape Based

  ```
  vrfExportLayoutSkill(cv "Metal3 drawing" "Metal1 drawing"
  ?pinNumberFile "pinNumPropAsNumber_new.txt" ?frontLabelLPP
  "Metal3 label" ?backLabelLPP "Metal1 label" ?dieInterfaceType
  "SHAPEANDLABEL")
  ```

  Output:

  ```
  Shape Based Export Die Report (design_45_sb/INVX1_45_0/layout)

  ****************************************************************

  Labels with overlapping shapes:

  Front Side: 3

  Back Side: 2

  Labels without overlapping shapes:
  ```


© 2023 Cadence Design Systems, Inc.

```
Front Side: 1

Back Side: 0

****************************************************************
```

Another example

```
vrfExportLayoutSkill(cv "Metal3 drawing" "" ?abstractLib "die_footprints1"
?areaTransferFile "./areaTxFile.xml")
```

### ***Related Topics***

Exporting Dies

exportNoBumpTerm

deleteViewsBeforeExport

outputMode


© 2023 Cadence Design Systems, Inc.

# vrfExportPackage

```
vrfExportPackage(
     g_cellViewId
     t_frontPinLayer
     t_backPinLayer
     [ ?instanceNames t_instanceNames ]
     [ ?abstractLib t_abstractLib ]
     [ ?ecoMode g_ecoMode ]
     [ ?attachSParam g_attachSParam ]
     [ ?sParamView t_sParamView ]
     [ ?modelFile t_modelFile ]
     [ ?shortPinLib t_shortPinLib ]
     [ ?shortPinCell t_shortPinCell ]
     [ ?csvFileName t_csvFileName ]
     [ ?abstractViewCell t_abstractViewCell ]
     )
     => t / nil
```

### Description

Exports the specified package layout if its `sipObjectType` value is set to `package`. Use the `instanceNames` argument to specify the instance names to search master views for identifying candidate IO cells to be exported. The IO cells to be exported should have the `cellType` defined as `coverBump` or `pad`. The shapes in the master views must match the front or back layer-purpose pairs specified in the SKILL function arguments. Only the sub-master views of the package instances and IO cells are processed, not the super-master views. To export a package layout, there should be top-level terminals in the layout. Also, all the package type instances in a package layout must be fully connected to top-level terminals.


© 2023 Cadence Design Systems, Inc.

### Arguments

*g_cellViewId* — The cellview ID of the layout to be used for exporting the die.

*t_frontPinLayer* — The front pin layer-purpose pair for exporting the front-side IO cells, for example, *Metal4 drawing*.

*t_backPinLayer* — The back pin layer-purpose pair for exporting the back-side IO cells.

`[ ?instanceNames` *t_instanceNames* `]`

The list of the top-level package type instance names, separated by space or comma. When this list is specified, only the IO pads inside the specified package type instances are processed for export operation. The default value is `""`. If `instanceNames` is `""`, all package type instances in a package layout are processed for while exporting the package.

`[ ?abstractLib` *t_abstractLib* `]`

The name of the package library of the abstract cellview. The default value is `pkg_footprints`.

`[ ?ecoMode` *g_ecoMode* `]`

Uses ECO mode to incrementally update the symbol view in the existing exported library. The default value is `nil`.

`[ ?attachSParam` *g_attachSParam* `]`

Specifies whether an s-parameter model needs to be generated. The default value is `nil`.

`[ ?sParamView` *t_sParamView* `]`

Name of the s-parameter view. The default value is `sparamModel`.

`[ ?modelFile` *t_modelFile* `]`

Model file to be used to create an s-parameter model. The default value is `nil`.

`[ ?shortPinLib` *t_shortPinLib* `]`

Library that contains the pin instance symbol for shorting multiple nets. The default value is obtained by `envGetVal("vrf.exportDie" "shortPinLib")`.

`[ ?shortPinCell` *t_shortPinCell* `]`


© 2023 Cadence Design Systems, Inc.

Cell that contains the pin instance symbol for shorting multiple nets. The default value is obtained by `envGetVal("vrf.exportDie" "shortPinCell")`.

`[ ?csvFileName` *`t_csvFileName`* `]`

A CSV file for customizing or updating the CDF parameters during the die export. The default value is `""`.

`[ ?abstractViewCell` *`t_abstractViewCell`* `]`

Name of the package cell of the abstract cellview. The default value is `""`.

### Value Returned

| | |
|---|---|
| `t` | The package export was successful. |
| `nil` | Package export failed. |

### Examples

Open a package layout.

```
->cv=geGetEditCellView()
; This should be the cvId of the package layout.
```

Export it as follows:

```
->vrfExportPackage(cv "M4 drawing" "")
```

This creates the exported library `pkg_footprints` (the default destination library) containing the exported cellviews.

### *Related Topics*

shortPinCell

shortPinLib


© 2023 Cadence Design Systems, Inc.

# vrfLowerPriority

```
vrfLowerPriority(
     d_dynShapeId
     )
     => t / nil
```

### Description

Lowers the priority of a specified dynamic shape so that it is one unit less than the lowest priority currently set for dynamic shapes on the layer.

### Arguments

| | |
|---|---|
| *d_dynShapeId* | Database identifier of the dynamic shape. |

### Value Returned

| | |
|---|---|
| t | The priority was lowered. |
| nil | The priority could not be lowered. The specified shape could not be found or it was not a dynamic shape. |

### Examples

```
vrfSipGet(vshape1 "priority")=> -1
vrfSipGet(vshape2 "priority")=> -2
```

These commands retrieve the priority settings for two dynamic shapes on a layer. For `vshape1`, it is `-1`, and for `vshape2`, it is `-2`.

```
vrfLowerPriority(vshape1) => t
```

This command lowers the priority for `vshape1`.

```
vrfSipGet(vshape1 "priority") => -3
vrfSipGet(vshape2 "priority") => -2
```

The retrieved priority settings now indicate that the priority for `vshape1` was lowered so that it is one unit below the lowest priority set on the other dynamic shapes on the layer.

#### *Related Topics*

Bump and Ball Editor Form


© 2023 Cadence Design Systems, Inc.

# vrfHiUpdate

```
vrfHiUpdate()
     => t / nil
```

### Description

Starts to process diestacks after creating a stackable object. This function must be used when instances are created in a script.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | The processing of diestacks started successfully. |
| `nil` | The diestacks could not be processed. |

### Examples

This command creates a stackable object.

```
inst1 = (dbCreateParamInst top super "I1" 0:0 "R0" 1 list(list("footprintCellName"
"string" "LNA_PKG_100") ))
vrfHiUpdate()
```

### *Related Topic*

View-in-Concert Mode


© 2023 Cadence Design Systems, Inc.

# vrfRaisePriority

```
vrfRaisePriority(
     d_dynShapeId
     )
     => t / nil
```

### Description

Raises the priority of a specified dynamic shape so that it is one unit higher than the maximum priority currently set for dynamic shapes on the layer.

### Arguments

| | |
|---|---|
| *d_dynShapeId* | Database identifier of the dynamic shape. |

### Value Returned

| | |
|---|---|
| t | The priority was raised. |
| nil | The priority could not be raised. The specified shape could not be found or it was not a dynamic shape. |

### Examples

```
vrfSipGet(vshape1 "priority")=> -2
vrfSipGet(vshape2 "priority")=> -1
```

These commands retrieve the priority settings for two dynamic shapes on a layer. For `vshape1`, it is `-2`, and for `vshape2`, it is `-1`.

```
vrfRaisePriority(vshape1) => t
```

This command raises the priority for `vshape1`.

```
vrfSipGet(vshape1 "priority") => 1
vrfSipGet(vshape2 "priority") => -1
```

The retrieved priority settings now indicate that the priority for `vshape1` was raised so that it is one unit above the maximum priority set on other dynamic shapes on the layer.

### *Related Topics*

Bump and Ball Editor Form


© 2023 Cadence Design Systems, Inc.

# vrfSipSet

```
vrfSipSet(
     d_objID
     t_attrName
     g_attrValue
     )
     => t / nil
```

### Description

Sets attributes for a specified object.

### Arguments

| | |
|---|---|
| *d_objID* | The database identifier of the object, such as an instance, a cellview, or a via, on which the attribute is to be set. |
| *t_attrName* | The name of the attribute. |
| *g_attrValue* | The value of the attribute. |

#### *Supported Attributes*

The attributes that can be specified with this function are listed below:

| | |
|---|---|
| `priority` | The priority for a dynamic shape. |
| `minAperture` | The minimum width of an effective shape or aperture created by voiding a dynamic shape. |
| `minArea` | The square root of the minimum area of an effective shape created by voiding a dynamic shape. |


© 2023 Cadence Design Systems, Inc.

`sipBumpParams` Attributes of solder balls or attributes of bumps of any instance.

It is a SKILL disembodied property list (DPL) with these values:

- `diameterTop`: upper diameter
- `diameterBottom`: lower diameter
- `diameterMax`: maximum diameter
- `height`: height
- `materialName`: material used to make the solder ball

The value `nil` is used to unset this attribute.

`sipComponentClass` Component class. String.

Valid values: `undefined`, `Discrete`, `Die`, `IO`

`sipDieThickness` Die thickness in microns. Float.

`sipIsMicroVia` Indicates whether the cellview is a micro via.

Boolean values: `true`, `false`

`sipIsTestPin` Indicates whether the instTerm is a test pin.

Boolean values: `true`, `false`

`sipIsTestVia` Indicates whether the via instance is a test via.

Boolean values: `true`, `false`

`sipObjectType` Object type of the cellview.

Valid values: `BondWire`, `BondWireFinger`, `Die`, `EmbeddedComponent`, `Package`, `PadStack`, `TLine`, `SMD`

`sipPadStackType` Pad stack type of a cellview.

Valid values:

- `PadStack`: The cell is used for a via.
- `Other`: The cell is used for a die or SMD pad.

`sipWirebondProfileName`

Wire profile name of a wirebond instance. String.


© 2023 Cadence Design Systems, Inc.

### Value Returned

| | |
|---|---|
| `t` | The attribute values were set as specified. |
| `nil` | The attribute values could not be set, or the object specified could not be found in the database. |

### Examples

```
vrfSipSet(instID "sipBumpParams" list(nil 'diameterTop 450.0 'diameterBottom 450.0
'diameterMax 500.0 'height 600 'conductivity 6e7))
=>t
```

Sets parameters for the attribute `sipBumpParams` on an instance of a BGA cellview.

#### *Related Topics*

Bump and Ball Editor Form


© 2023 Cadence Design Systems, Inc.

# vrfSipGet

```
vrfSipGet(
     d_objID
     t_attrName
     )
     => g_attrValue / nil
```

## Description

Retrieves values of a specified attribute of an object.

## Arguments

| | |
|---|---|
| *d_objID* | The database identifier of an object. |
| *t_attrName* | The name of the attribute.<br>See Supported Attributes. |

## Value Returned

| | |
|---|---|
| *g_attrValue* | The value of the attribute and object specified. |
| nil | The object specified did not have any attributes set, or it could not be found in the database. |

## Example

```
vrfSipGet(instID "sipBumpParams")
=>(nil diameterTop 450.0 diameterBottom 450.0 diameterMax 500.0 height 600
conductivity 6e7))
```

Retrieves parameters for the attribute `sipBumpParams`.

### *Related Topics*

Bump and Ball Editor Form


© 2023 Cadence Design Systems, Inc.

# vrfTLineAbut

```
vrfTLineAbut(
     t_instA
     t_instB
     t_pinA
     t_pinB
     t_pinAside
     t_conn
     n_event
     d_group
     d_chain
     )
     => t / nil
```

### Description

Abuts two transmission line instances with the required specifications and issues a warning if the device cannot be abutted.


© 2023 Cadence Design Systems, Inc.

### Arguments

| | |
|---|---|
| *t_instA* | The first instance for abutment. |
| *t_instB* | The second instance for abutment. |
| *t_pinA* | The first instance pin for abutment. |
| *t_pinB* | The second instance pin for abutment. |
| *t_pinAside* | The pinA type of instance A. |
| *t_conn* | The connection type. |
| *n_event* | The event number. |
| *d_group* | The group ID. |
| *d_chain* | The chain ID. |

### Value Returned

| | |
|---|---|
| t | The transmission line instances were abutted. |
| nil | The transmission line instances could not be abutted. |

### Example

```
vrfTLineAbut(instA instB)
```

Abuts the two instances as per the defined specifications.

### *Related Topics*

Creating TLines Instances

Instantiating TLine Instances


© 2023 Cadence Design Systems, Inc.

# vrfUpdateTILPVersion

```
vrfUpdateTILPVersion(
     x_dbId
     [ ?types g_types ]
     [ ?cells g_cells ]
     [ ?fileName g_fileName ]
     )
     => t / nil
```

### Description

Updates the super-master cellviews that have version lower than the latest version of the given TILP from the given ID, which can be of an instance, cellview, library, or sub-master cellview.

### Arguments

| | |
|---|---|
| *x_dbId* | The ID an instance, cellview, library, or sub-master. |
| *?types g_types* | The types of TILPs to be updated. You need to define a list of strings. The function runs on the specified TILP types, such as die, package, SMD, and so on. The TILP types list is mentioned in the header of the checker report generated by running `vrfCheckTILPVersion`. When set to `nil`, all types of TILPs to be updated. |
| *?cells g_cells* | The cells that need to be updated. |
| *?fileName g_fileName* | The name of the log file. |

### Value Returned

| | |
|---|---|
| `t` | The version of TILP is updated successfully. |
| `nil` | The version of the TILP could not be updated. |

### Example

- Instance


© 2023 Cadence Design Systems, Inc.

```
vrfUpdateTILPVersion( instId )
```

- Design

```
vrfUpdateTILPVersion( geGetEditCellView()  )
vrfUpdateTILPVersion( geGetEditCellView()  ?cells list("SMD_CAP" "SMD_RES") )
```

Runs only on the cells named `SMD_CAP` and `SMD_RES`.

- Library

```
vrfUpdateTILPVersion( ddGetObj("murataLib")  )
```

- Other Examples

```
vrfUpdateTILPVersion( geGetEditCellView()  ?cells list("SMD_CAP" "SMD_RES") )

vrfUpdateTILPVersion( geGetEditCellView()  ?types list("Die") )
```

Runs only on the Die TILPs.

```
vrfUpdateTILPVersion( geGetEditCellView()  ?fileName "updateReport.txt" )
```

### *Related Topics*

TILP Versions

vrfCheckTILPVersion


© 2023 Cadence Design Systems, Inc.