cadence®

Virtuoso MultiTech Framework User Guide

Product Version ICADVM20.1
October 2020

Printed in the United States of America.

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

Contents

Preface

The *Virtuoso MultiTech Framework User Guide* describes the Cadence Virtuoso MultiTech Framework, which defines the Virtuoso Schematic Editor–driven SiP Layout flow and Virtuoso Schematic Editor driven Layout EXL flow.

This user guide describes three integrated Cadence products: Virtuoso Schematic Editor™, for schematic design entry; Virtuoso Layout Suite™, and SiP Layout™ for physical layout of a design; and Analog Design Environment tools, for simulation of design. The integration of these products enables you to:

- Create a schematic that can be used in physical layout
- Update the schematic with Allegro layout design data

The *Virtuoso MultiTech Framework User Guide* describes how to use the features unique to these integrated products, Virtuoso Schematic Editor XL, Virtuoso Layout Suite EXL, and SiP Layout. This guide assumes that you have working knowledge of basic Virtuoso Schematic Editor and Virtuoso Layout Editor functions.

Scope

The functionality described in this guide can be used only in ICADVM20.1 advanced nodes and advanced methodologies releases.

Licensing Requirements

The following are the license products needed for Virtuoso RF Solution:

Virtuoso Electromagnetic Solver Assistant

Virtuoso Layout Suite EXL

EMX 3D Planar Solver

Clarity 3D Solver

Virtuoso System Design Solution - RF Module Layout

Virtuoso_MultiTech_Framework

Virtuoso Schematic Editor XL

For SIP Layout editing with Virtuoso Layout Suite

Virtuoso Layout Suite EXL

Virtuoso RF Option

For SIP Layout editing with Allegro Package Designer

SIP Layout Option

Allegro Package Designer Plus

For more information on licensing, see *Virtuoso Software Licensing and Configuration Guide*.

Virtuoso MultiTech Framework Release Compatibility Matrix

Compatible Release	Release on downloads.cadence.com	Version ID
Virtuoso	ICADVM18.10.110	ICADVM18.1 ISR11
SPB	SPB17.2.063 and above	SPB17.2 ISR63 and above
SPECTRE	SPECTRE18.10-421	SPECTRE18.10 ISR9
SIGRITY (For Clarity and AXIEM)	SIG19.00.000	2019
SIGRITY (For 3D-EM)	SIG3DEM19.00.000	SIGRITY 3D-EM 2019

Important

Refer to the release README file to check for other release compatibility versions.

Setting Up Virtuoso MultiTech Framework

To access the Virtuoso MultiTech Framework, set the following SHELL environment variable:

```
Virtuoso_MultiTech
```

The license is checked out when Virtuoso is started in the shell with the environment variable set.

Related Documentation

What's New and KPNS

- *Virtuoso MultiTech Framework What's New*
- *Virtuoso MultiTech Framework Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Software Licensing and Configuration User Guide*
- *Virtuoso Design Environment User Guide*
- *Cadence Application Infrastructure User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Virtuoso Schematic Editor SKILL Reference*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

Virtuoso Tools

- *Virtuoso ADE Assembler User Guide*
- *Virtuoso ADE Explorer User Guide*
- *Virtuoso Schematic Editor User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator RF Analysis User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Spectre Circuit Simulator Reference*

- *Virtuoso Layout Viewer User Guide*
- *Virtuoso Layout Suite XL: Basic Editing User Guide*
- *Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*
- *Virtuoso Layout Suite EXL Reference*
- *Virtuoso Concurrent Layout User Guide*
- *Virtuoso Design Planner User Guide*
- *Virtuoso Multi-Patterning Technology User Guide*
- *Virtuoso Placer User Guide*
- *Virtuoso Simulation Driven Interactive Routing User Guide*
- *Virtuoso Width Spacing Patterns User Guide*
- *Virtuoso RF Solution Guide*
- *Virtuoso Electromagnetic Solver Assistant User Guide*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses on Virtuoso Schematic Editor:

- Virtuoso Schematic Editor
- Virtuoso Analog Design Environment
- Using Virtuoso Constraints Effectively
- Virtuoso Spectre Circuit Simulator
- Spectre Simulations Using Virtuoso ADE
- Virtuoso Electrically-Aware Design with Layout Dependent Effects

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
|	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument. Used with brackets to indicate that you can specify zero or more arguments. Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

1

Virtuoso MultiTech Framework

Virtuoso MultiTech Framework 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.

About the Virtuoso MultiTech Framework

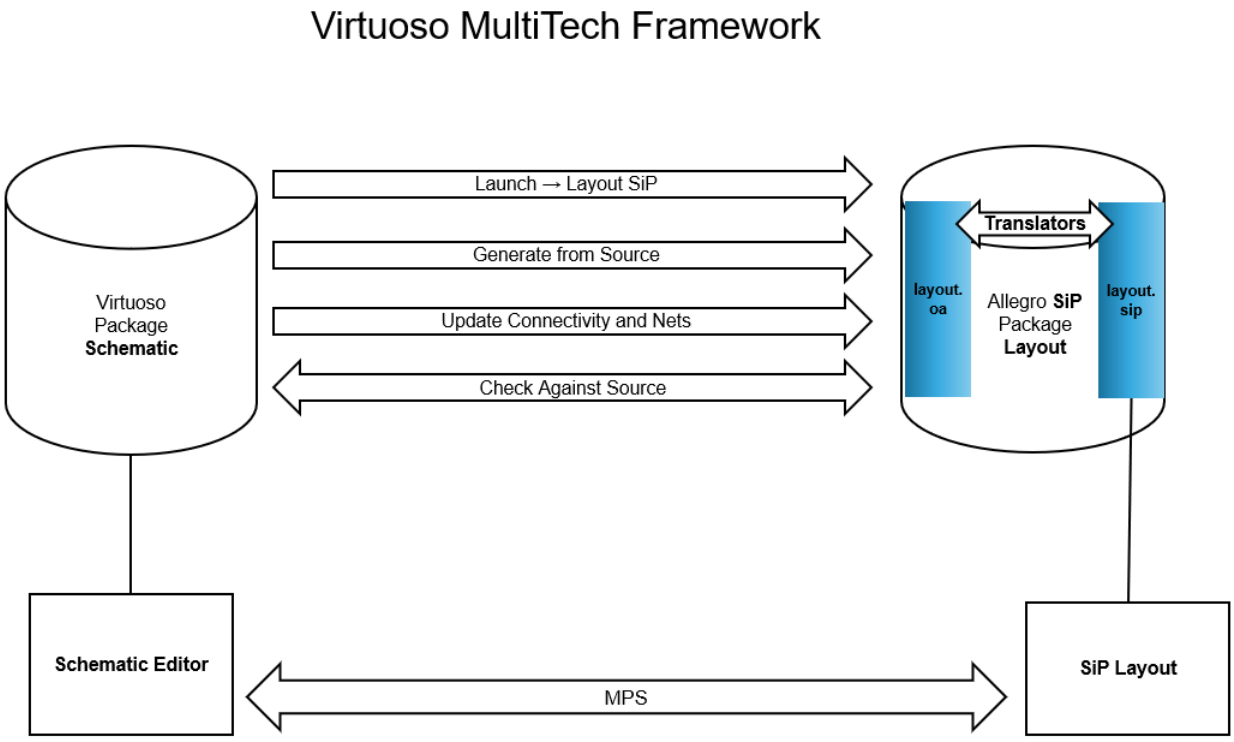
The Framework provides a base set of services that allows you to design packages, modules, and boards using a variety of heterogeneous integration styles such as die stacking, wire bonding, interposers, and wafer-level packaging (WLP). The schematic for the system might be hierarchical, which consists 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 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 MultiTech Framework environment involves various tools, such as Cadence SiP Layout, Sigrity 3D-EM, Clarity solvers, Virtuoso Schematic Editor XL, *Virtuoso ADE Explorer*, *Virtuoso ADE Assembler*, and *Virtuoso Visualization and Analysis XL*.

Here are the benefits of the Virtuoso MultiTech Framework:

- The Virtuoso MultiTech Framework combines the benefits of Virtuoso schematic driven Layout EXL methodology along with the features of the SiP Layout.
- The key features of Layout EXL, 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.

Virtuoso MultiTech Framework Flows

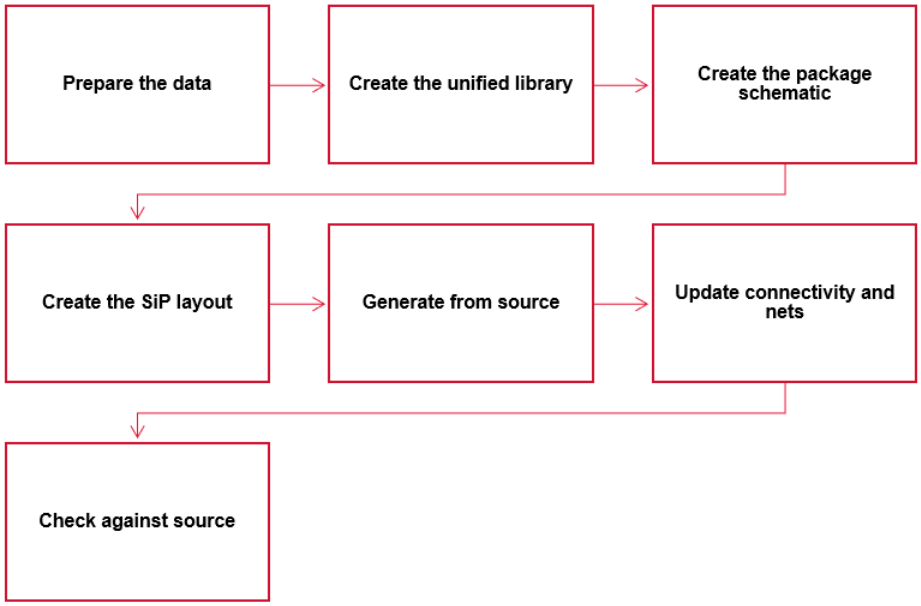
There are primarily two flows in the Framework, SiP Layout flow and Layout EXL flow.

- Virtuoso Schematic Editor–Driven SiP Layout Flow
- Virtuoso Schematic Editor Driven VSE driven Layout EXL Flow

This gives the flexibility to finalize the layout in the Allegro or Virtuoso environment. Cadence SiP Layout has many features for package and module implementation alongside a complete complement of constraint-driven verification and automation tools to complete the implementation. Artwork and manufacturing activities must be performed in Cadence SiP Layout. Teams that do day-to-day module and package design in Cadence SiP Layout can enjoy benefits from using a Virtuoso Schematic Editor driven flow without changing their use model. In addition, it provides the ability to mix high-accuracy extraction models with ideal models and simulate the design along with the testbench in Spectre.

Virtuoso Schematic Editor–Driven SiP Layout Flow

The following diagram depicts the overall flow.



1. **Set up the framework.**

   Set up the path and required SHELL environment variables for the Virtuoso MultiTech Framework environment.

   For details, see Setting Up Virtuoso MultiTech Framework.

2. **Create the unified library.**

Map the terminologies and data between Virtuoso and Allegro by using unified libraries for a seamless flow of information.

For details, see Managing Unified Libraries.

3. **Create the package schematic.**

   Create a package schematic by instantiating SMD instances, instances of dies or ICs, transmission lines, selecting parts, creating attachments, connecting attachments to the package connectors, and simulating the schematic.

   For details, see Creating a Package Schematic.

4. **Create the SiP layout.**

   Generate the SiP layout by choosing *Layout SiP* from the *Launch* menu. Then, import the technology and parameters file.

   For details, see Creating a SiP Layout from Package Schematic.

5. **Generate the layout from the source.**

   Generate the package layout from the package schematic.

   For details, see Generating from Source Schematic.

6. **Update connectivity and nets.**

   Update the connectivity and nets information from the package schematic.

   For details, see Checking against Source Schematic.

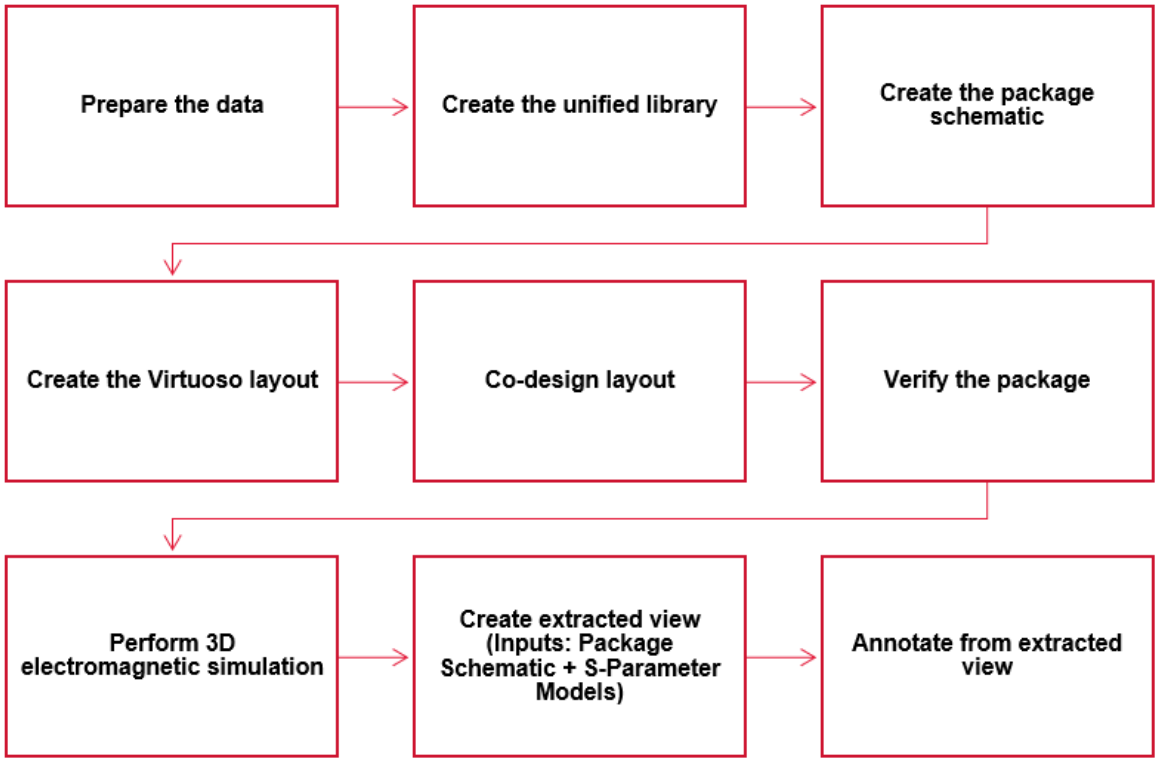
7. **Check against the source.**

   Compare the changes done in the SiP layout with the schematic using the LVS check.

   For details, see Checking against Source Schematic.

Virtuoso Schematic Editor Driven VSE driven Layout EXL Flow

The following diagram depicts the overall flow.



1. **Set up the framework.**

   Set up the path and required SHELL environment variables for the Virtuoso MultiTech Framework environment.

   For details, see Setting Up Virtuoso MultiTech Framework.

2. **Create the unified library.**

   Map the terminologies and data between Virtuoso and Allegro for the seamless flow of information.

   For details, see Managing Unified Libraries.

3. **Create the package schematic.**

Create a package schematic by instantiating SMD instances, instances of die or IC, transmission lines, selecting parts, creating attachments, connecting attachments to the package connectors, and simulating the schematic.

For details, see Creating a Package Schematic.

4. **Create the Virtuoso layout.**

   Create a package layout by generating a layout from source, creating bond wires, bump attachments, die embedding, die stacks, voids using dynamic shapes. Co-design by opening layouts from multiple fabrics simultaneously represented as various tabs in Virtuoso layout. Thereafter, perform interactive routing.

   For details, see Creating Package Layout.

5. **Co-Design layouts.**

   Co-design by opening layouts from multiple fabrics simultaneously represented as various tabs in Virtuoso layout. Thereafter, perform interactive routing.

   For details, see Co-Design.

6. **Verify the package.**

   Perform the connectivity, LVS, DRD, and DRC checks on the layout after importing it from Allegro.

   For details, see Verify the Package.

7. **Perform 3D electromagnetic simulation.**

   Perform EM analysis using Sigrity 3D-EM, and post-layout simulation. Additionally, create the extracted schematic using S-Parameters from 3D-EM.

   For details, see Performing 3D Electromagnetic Simulation.

8. **Create extracted view.**

   Create extracted view of the golden package schematic by using the s-parameter models created by using Sigrity.

   For details, see Creating an Extracted View.

9. **Annotate from extracted view.**

   Backannotate the parasitic models from the extracted view on the golden package schematic.

   For details, see Creating an Extracted View.

2

Managing Unified Libraries

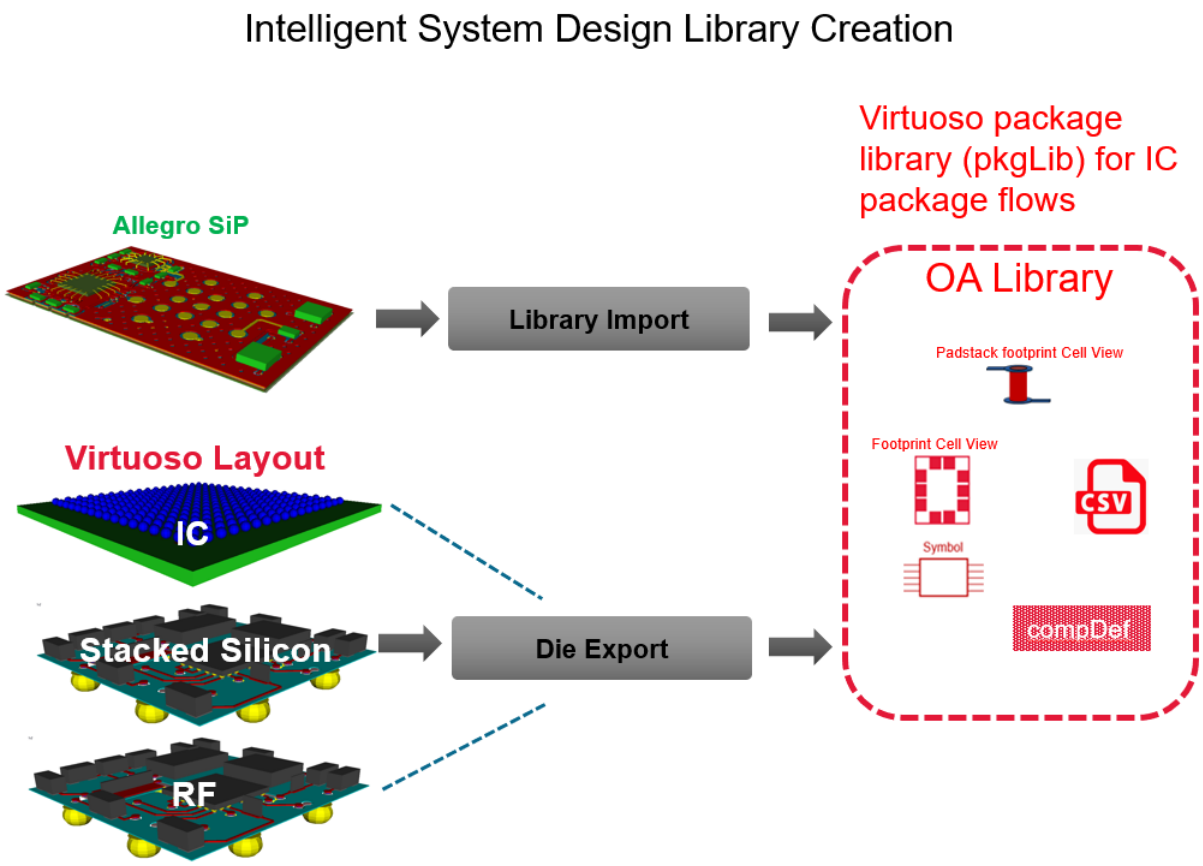
About Unified Libraries

Unified libraries provide a single point of entry for all Virtuoso-driven multiple technology flows. The flows include:

- Virtuoso MultiTech Framework: a Virtuoso schematic-driven Allegro layout 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
- Virtuoso Stacked Silicon Solution: a system planning and interposer routing flow

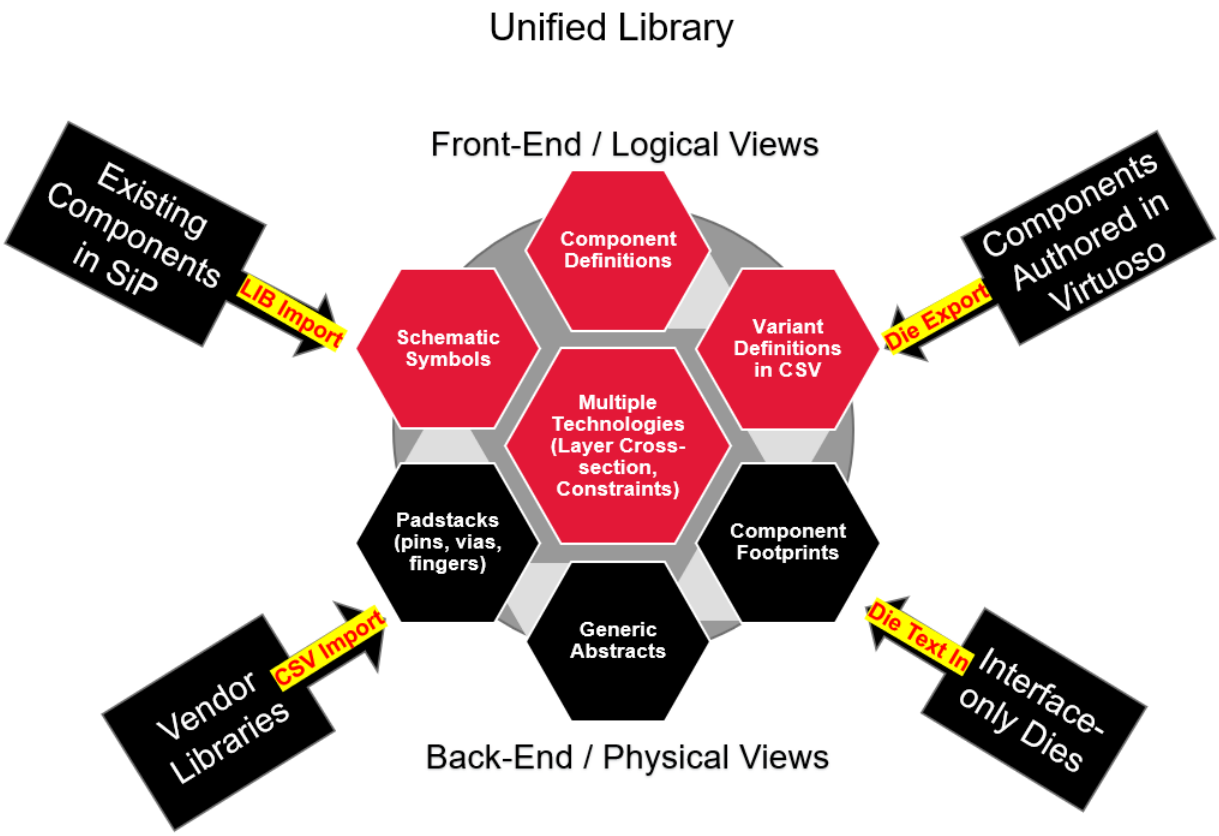
Unified libraries refer to the consistent organization of design data libraries. Libraries contain schematic symbols, footprints, component definition mapping between schematic symbol and footprints, TILP supermasters, part definition CSV files, and simulation models. The mapping between schematic and 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 schematics

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



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
- Footprint Views
- Component Definitions
- Part Definition 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 the Allegro views are used, the terminal name information is provided by component

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 are automatically split during the import process.

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 the 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

generic because the properties that make these definitions unique are not stored in the definition itself. The properties are stored in the Part Definition CSV Files.

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

Part Definition CSV Files

Part Definition CSV files contain the properties that are presented in the Allegro component definition. In addition, the import process adds some mandatory properties that help with 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.

Component Modeling

This topic describes the following subtopics about modeling of padstacks and symbols.

- Padstacks
- Symbol Definitions
- Components and Symbols

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.

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.

Types of Libraries

Unified libraries are crucial to enabling all the Virtuoso multiple technology flows. These libraries exist in the following forms and must be imported or created.

- 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.

Here are the following types of 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.

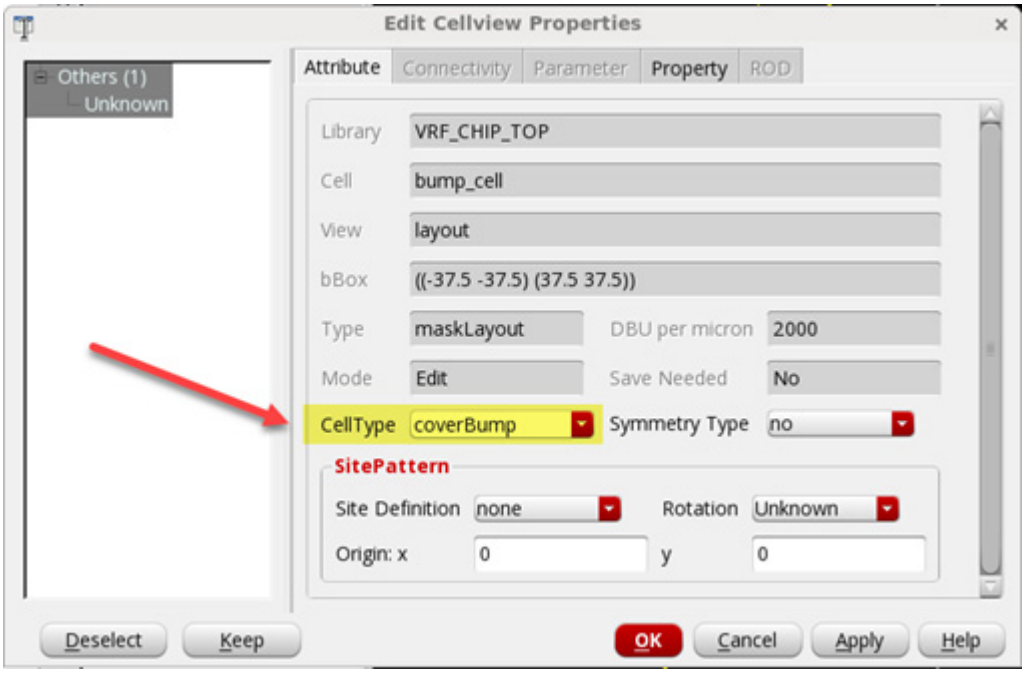
Exporting the die also creates one cell for each unique padstack used in the die abstract. Consequently, the following views are created for the padstack:

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

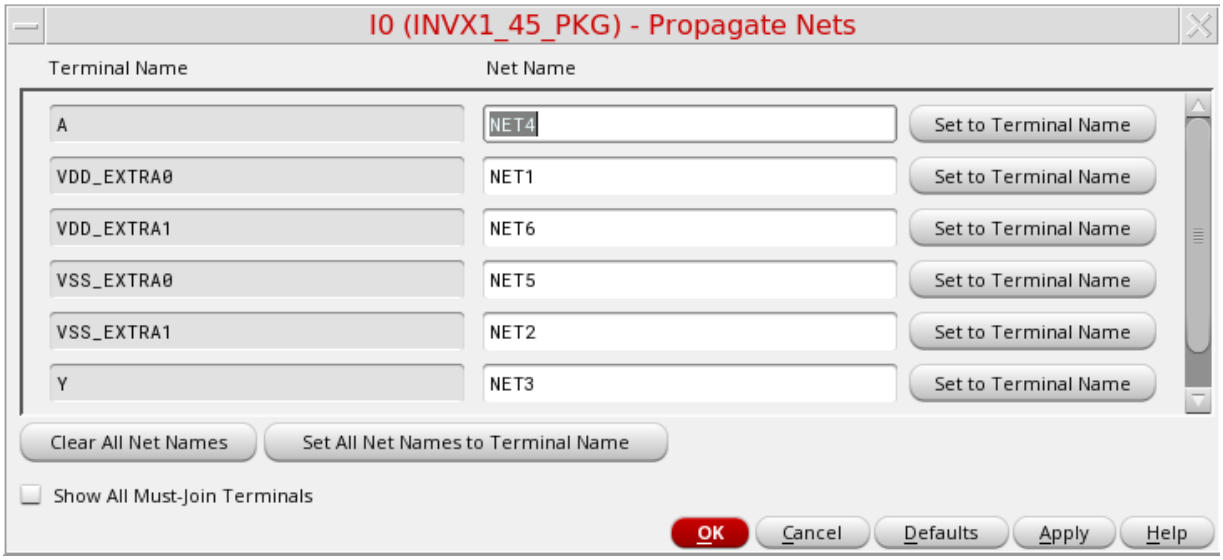
Preparing for Die Export

To prepare the die for ensuring that the die abstract is accurate before using it in the package:

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



- Update the connectivity information for the bump cells.



- Define `prBoundary` at all the levels of the design.

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

Exporting the Die

Video

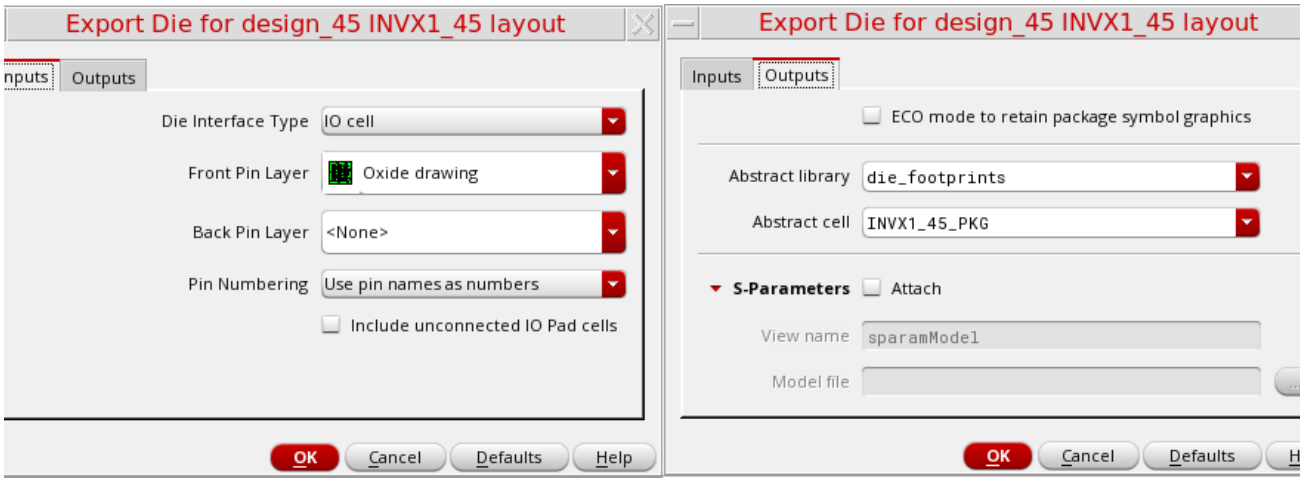
To find out more about the process of creating TILPs and the various types of views that are generated, see the video Creating a TILP by Exporting the Die.

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 co-design, layout versus abstract (LVA) checks, and cross-fabric simulation using die schematic and model-based simulation for dies.

To export the die:

1. Open the IC/die layout.
2. Click *Module – Export Die*.

3. Set the options in the Export Die form. Alternatively, use vrfExportLayoutSkill.



   You must specify the *Front Pin Layer* on the *Inputs* tab while defining the inputs for exporting the die. Most of the options are already set by default in the form.

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

After exporting the die, you can edit the die layouts and die abstract TILPs that are part of the same package by using the Co-Design feature.

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>`.
- 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.

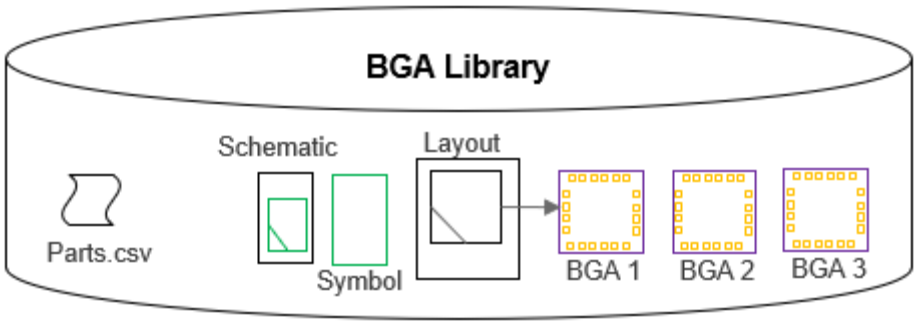
- Extracts pins based on the shapes. In shape-based layouts, a shape is exported as a pin with a name the same as the text of the overlapping label. For a shape to be exported as a pin,
  - It must be a rectangle, polygon, or ellipse.
  - The origin of the label must overlap with the shape to be exported.

After exporting the die, the 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
```

BGA Library

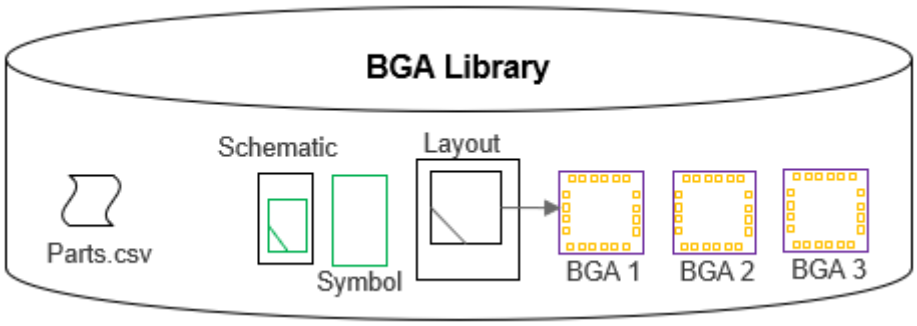
Ball grid array (BGA) is a type of die component whose pins are solder balls arranged in a grid pattern. For an effective use model, group multiple BGAs within a part table.



Tline Library

The Tline library contains the symbol view, simulation view, and 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.



pkgLib Library

The pkgLib library contains wirebonds and fingers. These are stored as TILPs in the library for use in the Virtuoso MultiTech Framework. 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.

3

Creating a SiP Layout

Creating a SiP Layout from Package Schematic

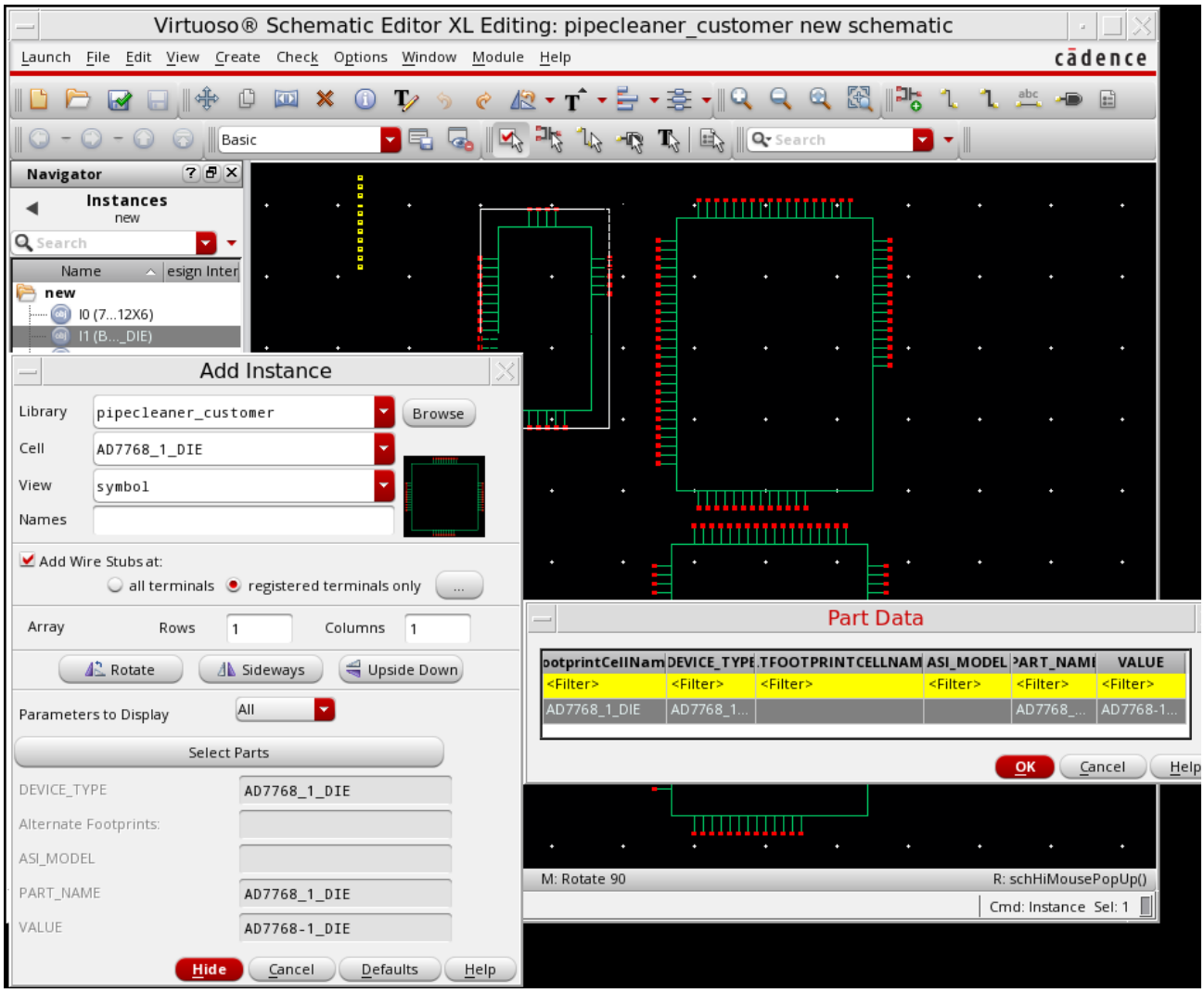
This topic describes how to generate the SiP layout from the package schematic and edit the package layout. The development of any design involves an iterative process of synchronizing the differences between the schematic and the SiP layout. Changes, especially caused by Engineering Change Orders (ECOs), 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.

Important

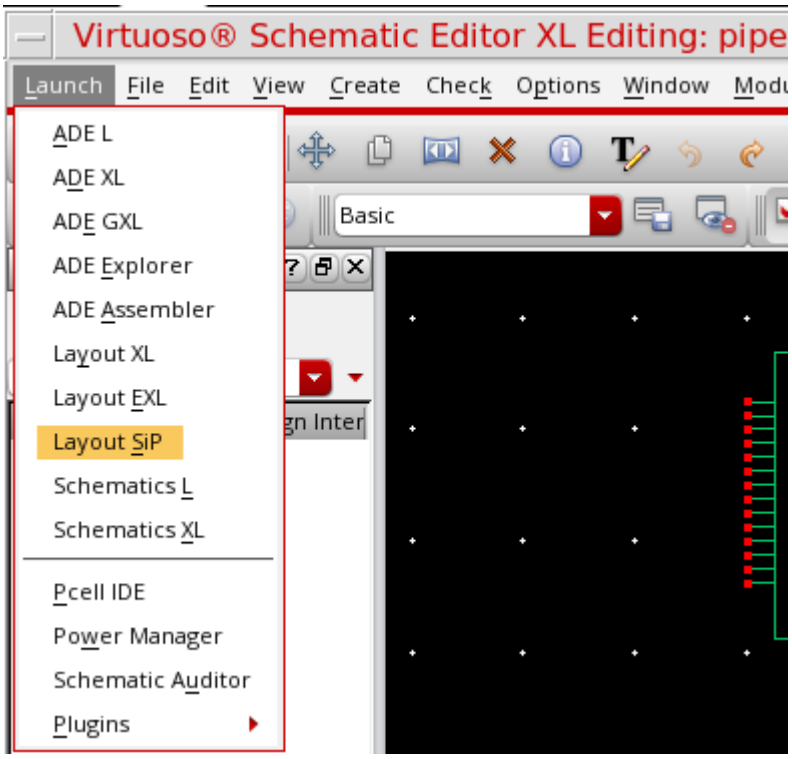
Ensure that the IC, PCB, and Sigrity hierarchy paths have been set for seamless flow of tasks in the Virtuoso MultiTech Framework environment.

To create the SiP layout:

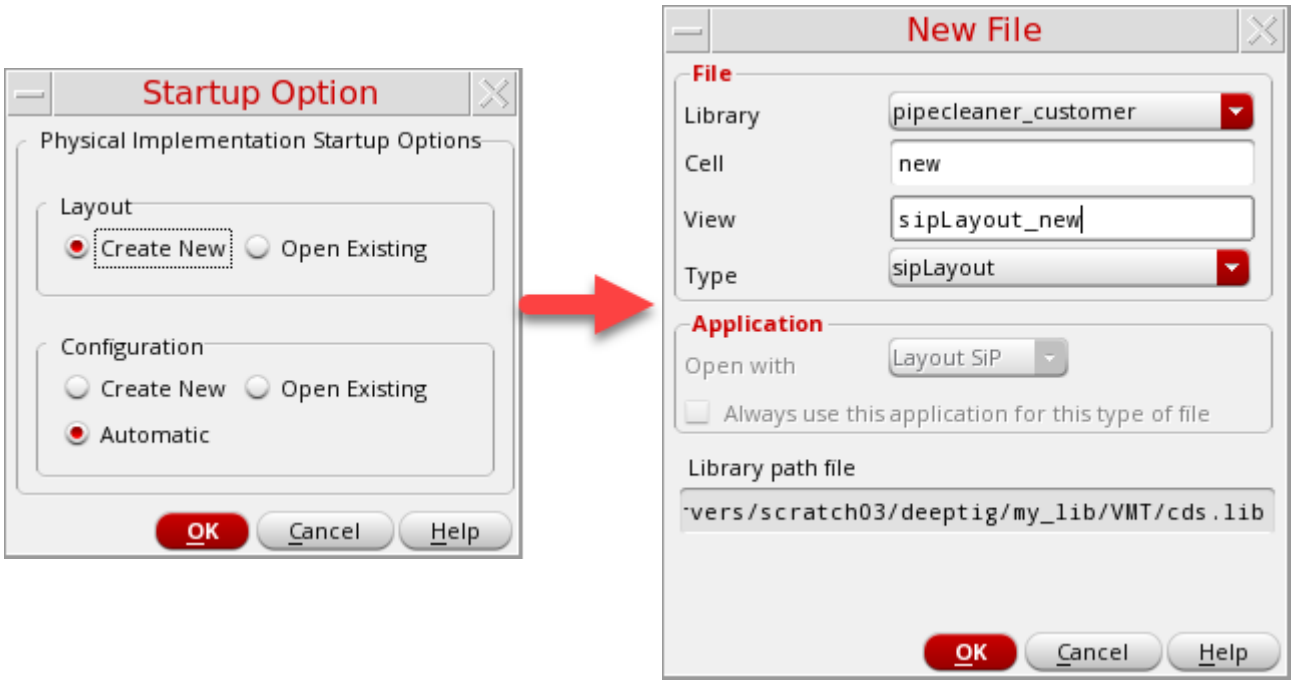
1. Create a package schematic by selecting parts and adding instances in the canvas.



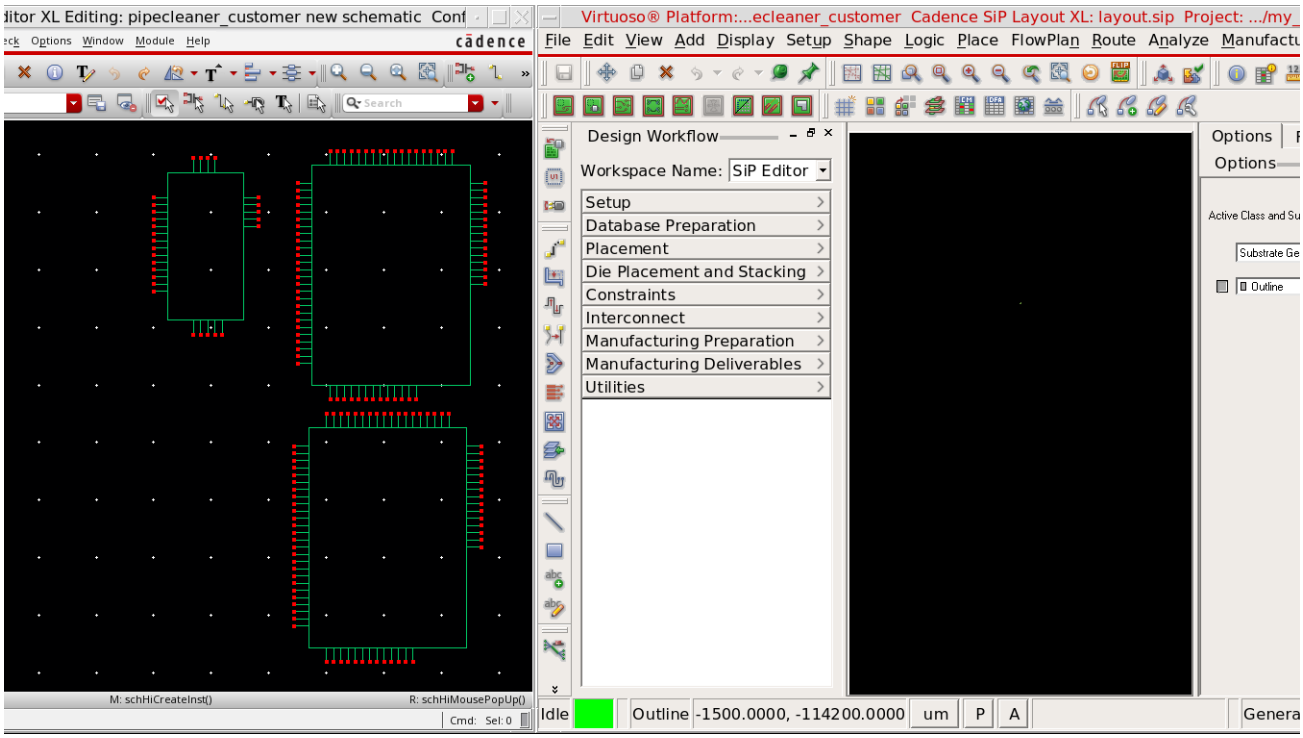
2. Choose *Launch – Layout SiP*.



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



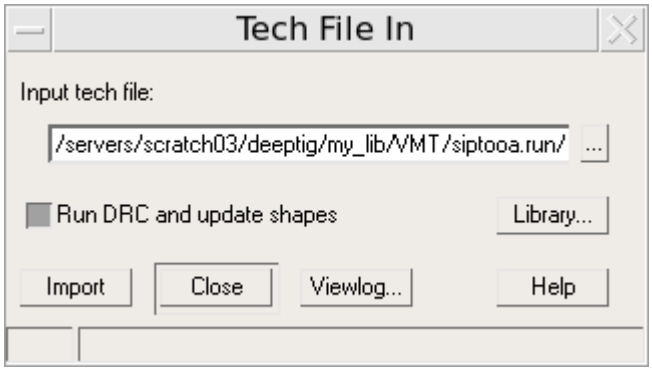
4. Arrange Cadence Virtuoso Schematic Editor and Cadence SiP Layout windows alongside.



5. Import the technology and parameter files.

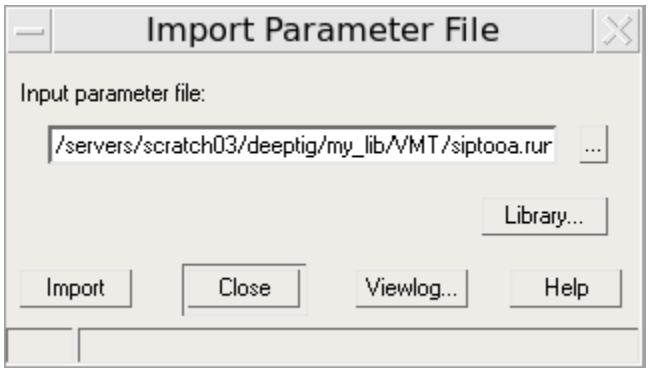
a. Choose *File – Import – Techfile*.

b. Specify the file in the Tech File In form.

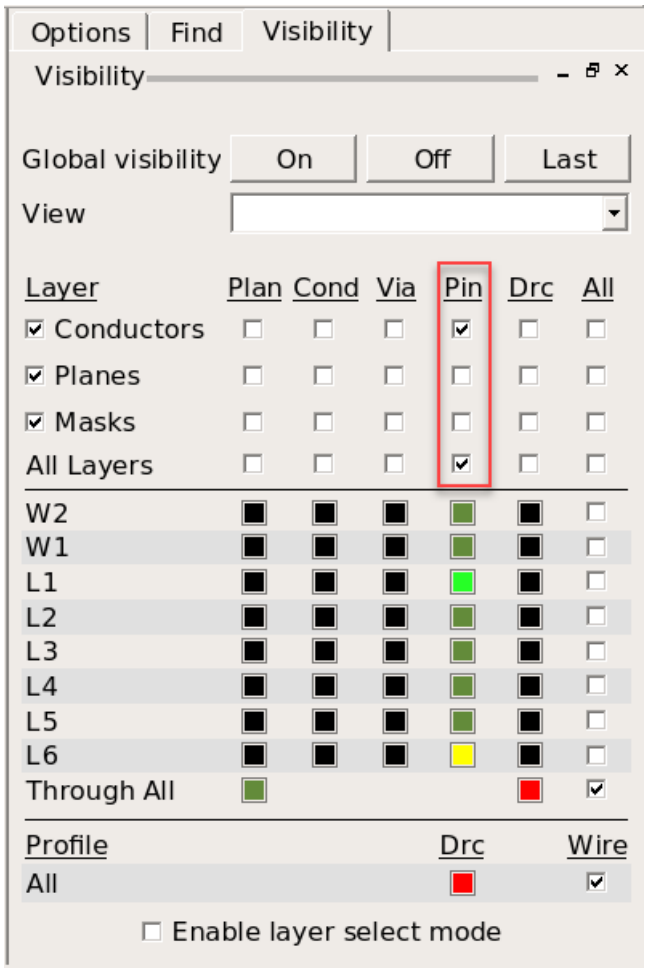


c. Choose *File – Import – Parameters*.

d. Specify the file in the Import Parameter File form.



6. In the Visibility tab on the canvas, enable *Pin* for *Layer Conductors*.

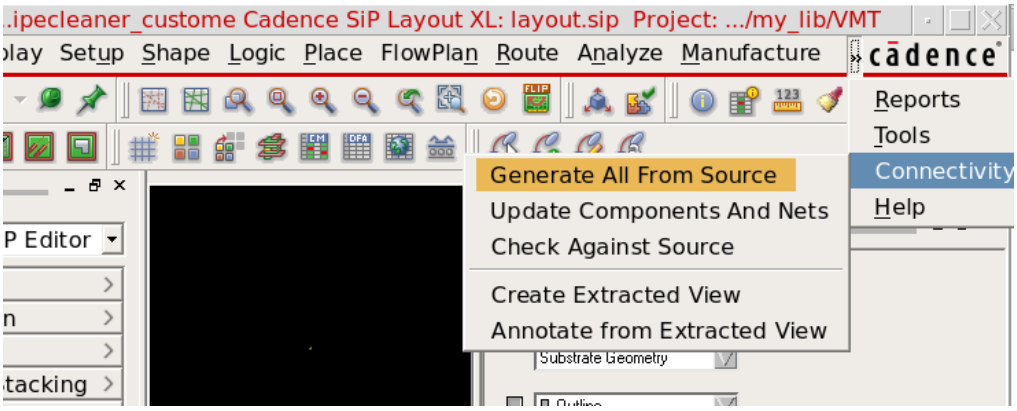


Generating from Source Schematic

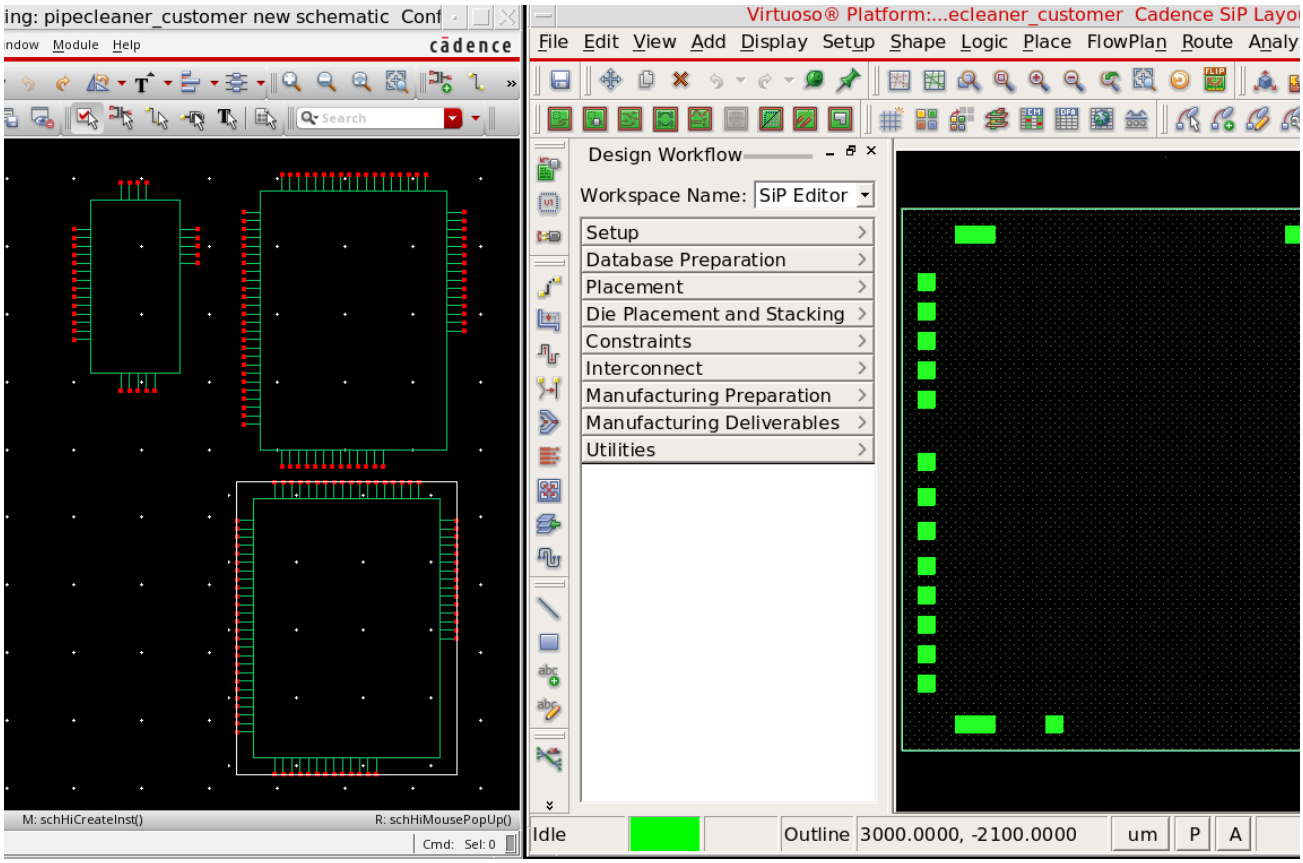
This topic lists the steps to generate the SiP layout from the package schematic in Virtuoso. You can use the *Connectivity* menu available in the SiP layout to create the physical layout in SiP. The capabilities include cross-selecting components in schematic and highlighting them in the layout and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

To generate from source:

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



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. For details, refer to Scribe Lines Feature.

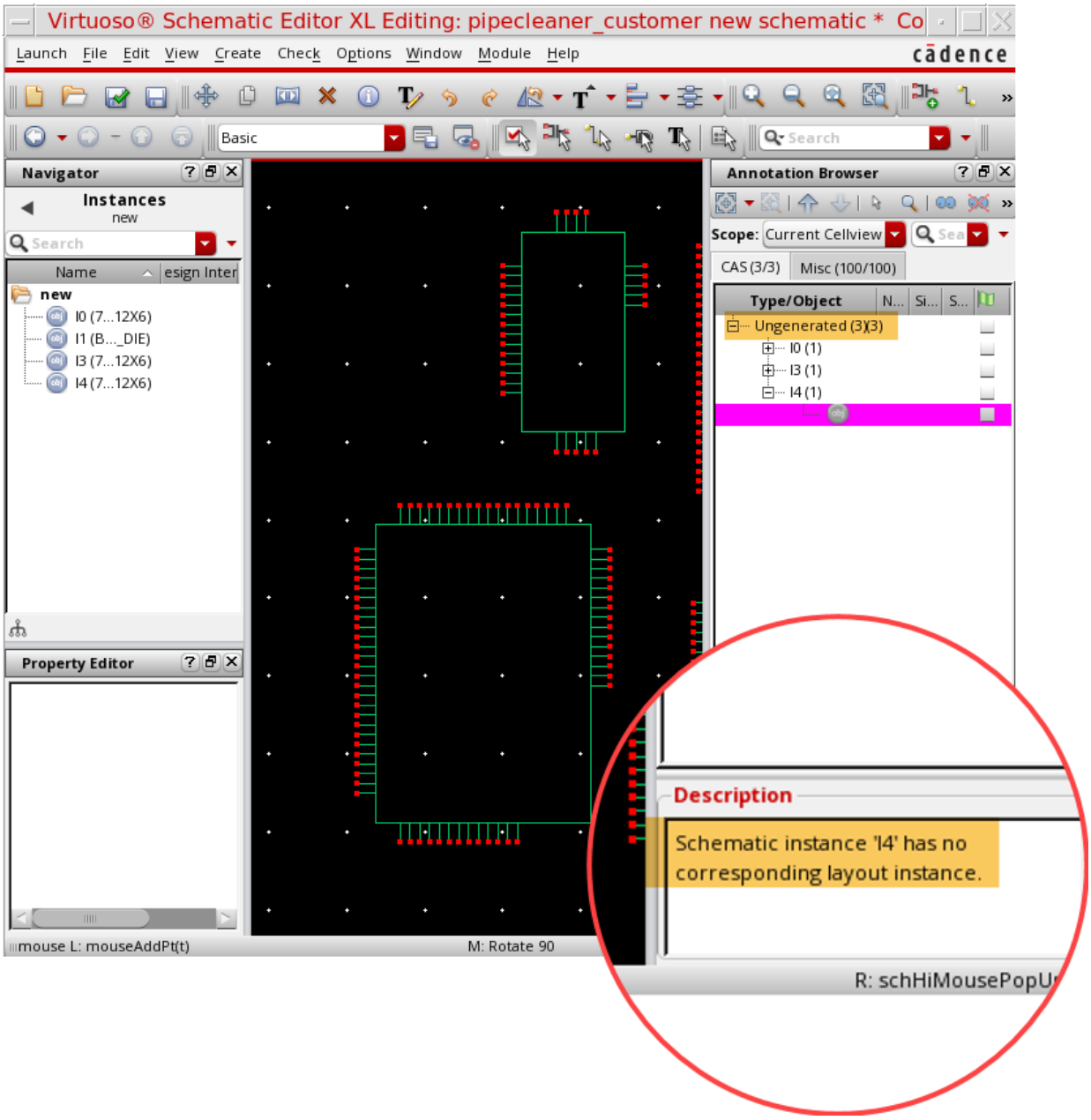
Checking against Source Schematic

To check against the source schematic:

1. Click the 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.

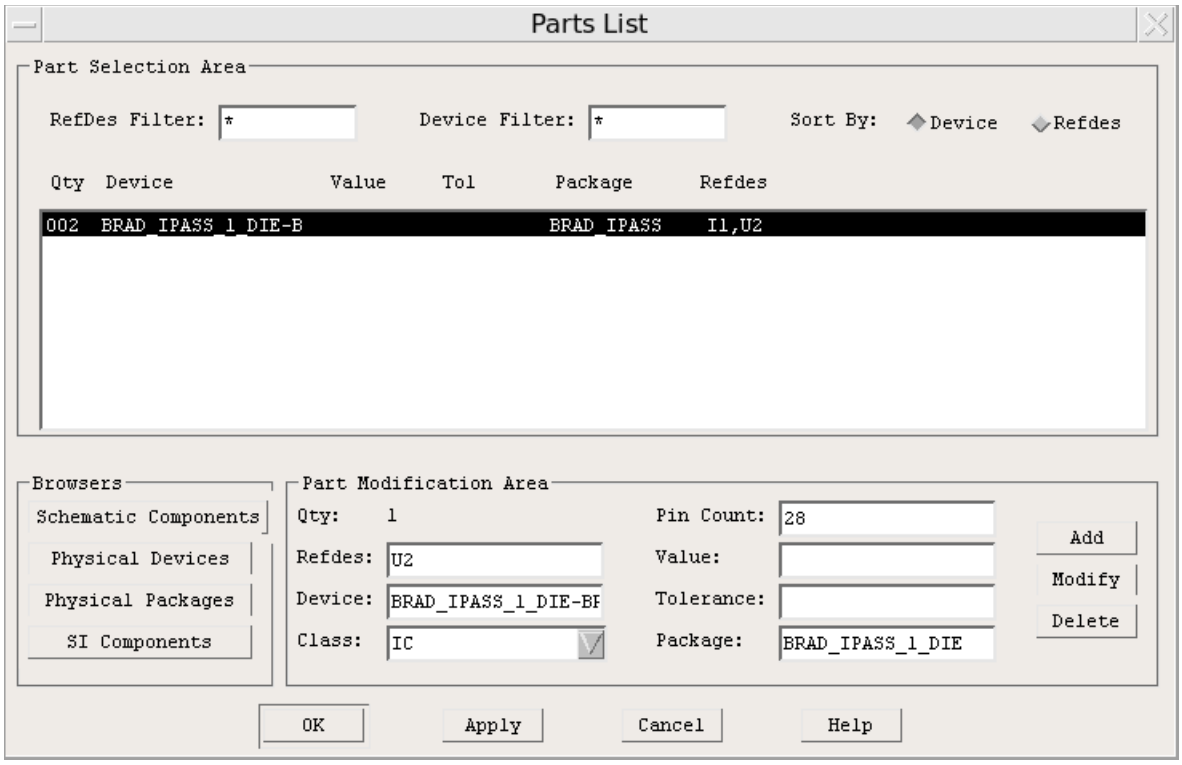
5. In the Annotation Browser, expand the `Ungenerated` list. Note the *Description* field.



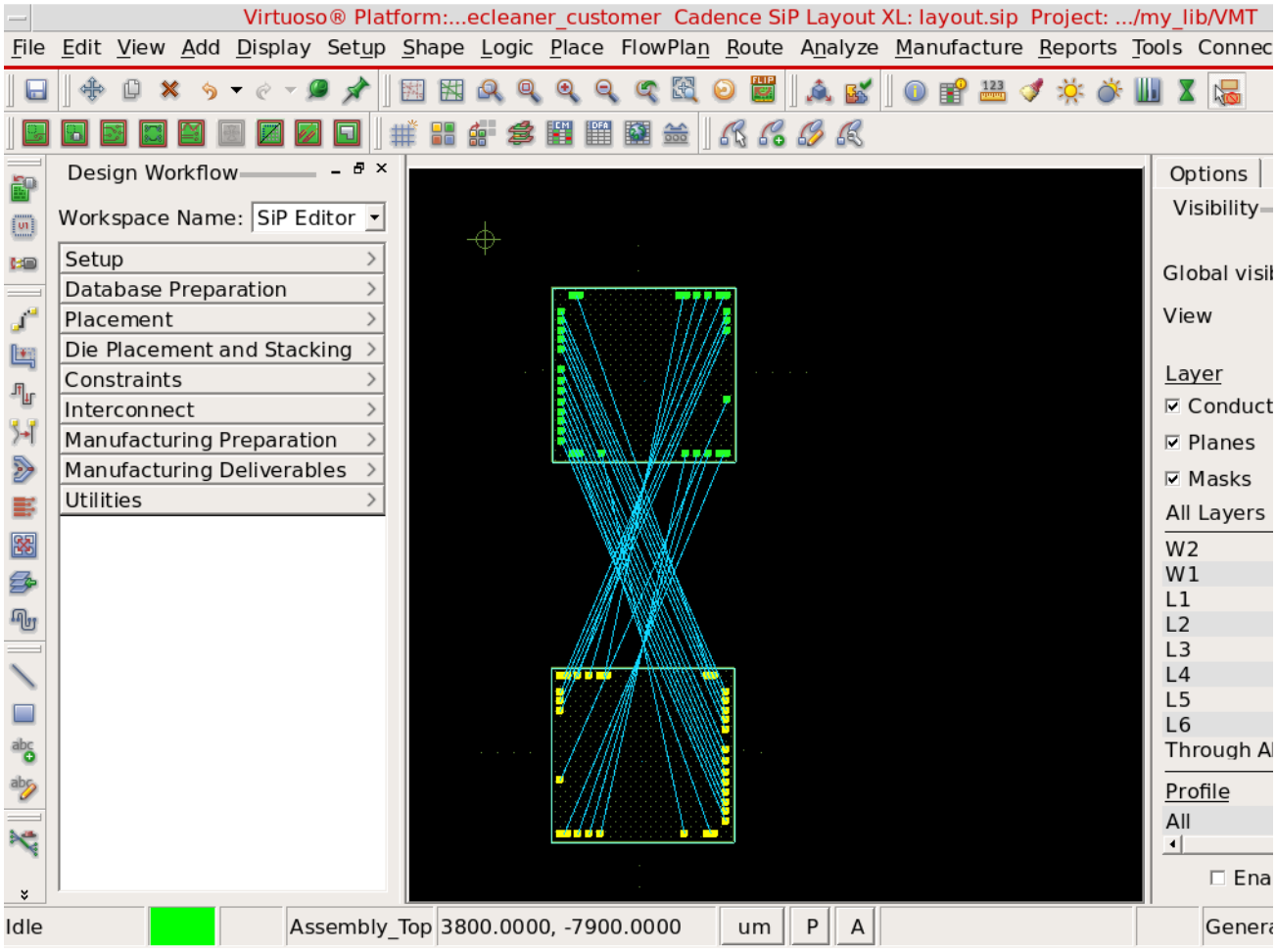
6. In the SiP layout, click *Connectivity – Update Components and Nets*. The Annotation browser removes the mismatched instances.

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

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

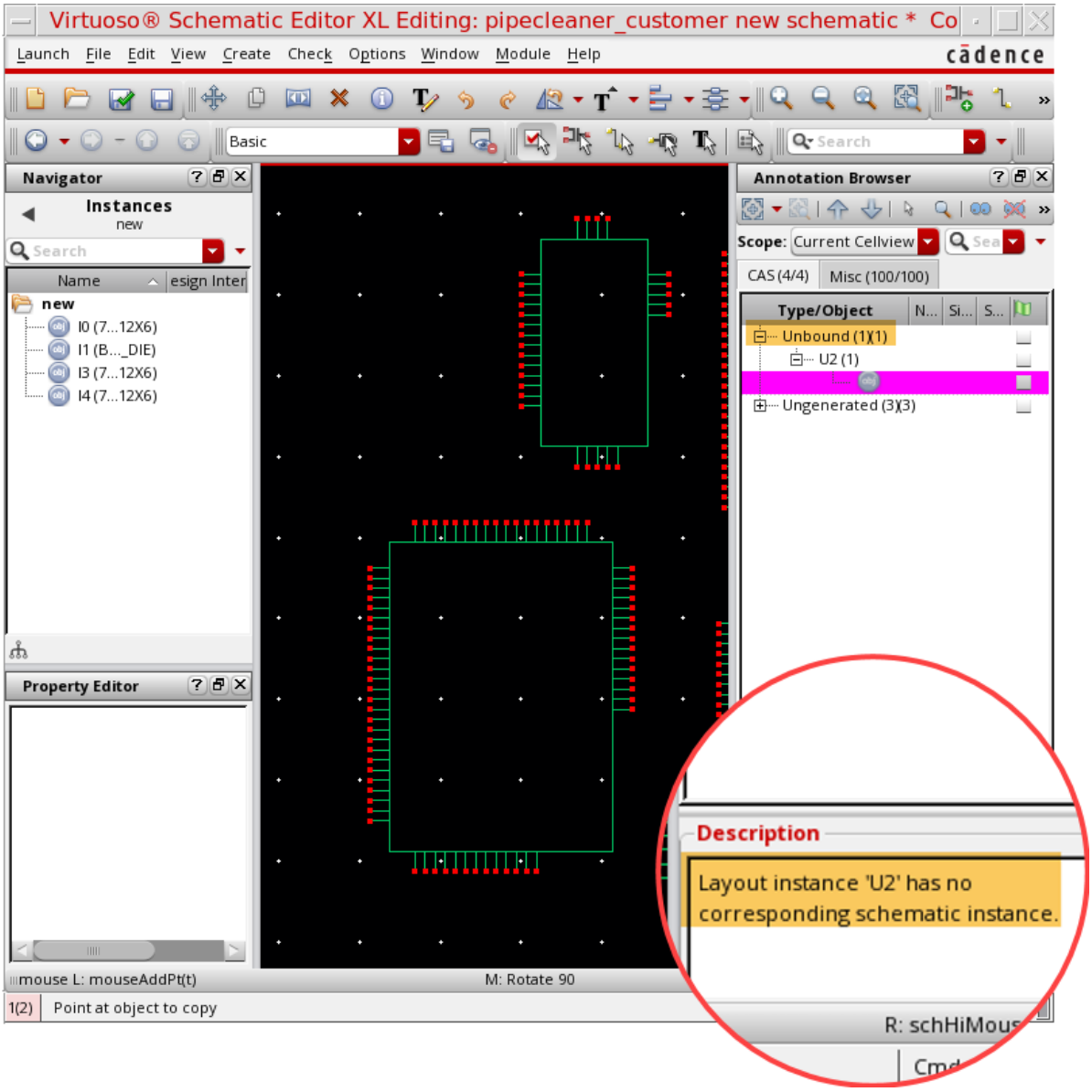


9. Click *Place – Manually* and place `U2`.



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

11. In the Annotation Browser, expand the `Unbound` list. Note the *Description* field.

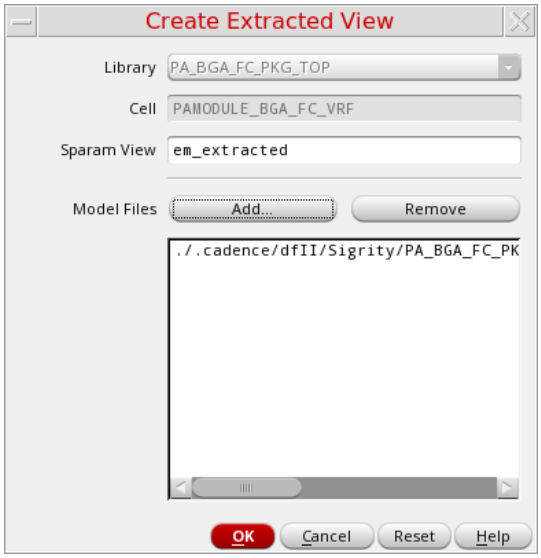


Creating an Extracted View

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

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

The Create Extracted View Form form opens.



Important

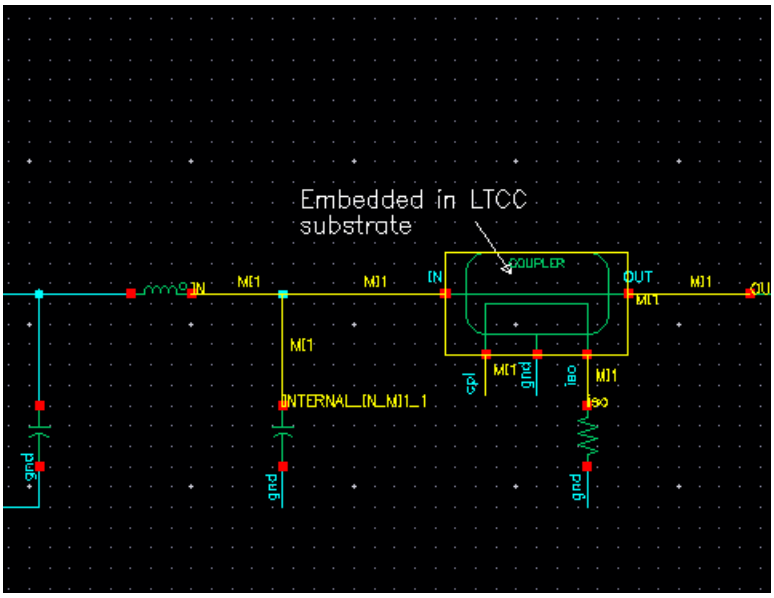
Only SiP files generated using the latest Virtuoso MultiTech Framework and model files created using the Sigrity 2019 version or more recent versions can be used to create an extracted view.

2. Add the model files in the *Model Files* field and click *OK*.

   For details about creating the s-parameter model files, see the following links:

   - Running a Clarity Simulation
   - Running a Sigrity 3D-EM Simulation
   - Running an AXIEM Simulation

3. Choose *RF-Module – Annotate from Extracted View* to annotate the parasitic models from the extracted view on the master schematic. The Annotate From Extracted View Form dialog box opens, with models highlighted in the master schematic.



For IC schematics, you can also use Quantus QRC-based smart views to create extracted views after EM simulation. For details, see Creating an Extracted View from Smart View.

A

Virtuoso MultiTech Framework Forms

This appendix describes the Virtuoso MultiTech Framework forms as they appear in the GUI.

- Create Extracted View Form
- Annotate From Extracted View Form

Create Extracted View Form

Propagates the changes done in the master schematic to the extracted view.

Field Name	Description
Library	The library name of the package schematic.
Cell	The cell name of the package schematic.
Sparam View	The name of the extracted view where the changes are being propagated.
Model Files	The model files to be used for comparison. The files must be `*.snp`.

Related Topic

Creating an Extracted View

Annotate From Extracted View Form

Highlights the nets or models on the master schematic that have been extracted.

Field Name	Description
Extracted View	The name of the extracted view that is referred to for backannotation.

Related Topic

Creating an Extracted View

A

Virtuoso MultiTech Framework Environment Variables

This appendix describes public environment variables that you can use to customize the behavior of the Virtuoso MultiTech Framework:

- vsdpSparamCSVModelNameField
- vsdpSpiceCSVModelNameField

vsdpSparamCSVModelNameField

Specifies the default value of the part variant field that would be used to identify the model file associated with the instance.

In `cdsenv`:

```
asimenv.netlist vsdpSparamCSVModelNameField string "ASI_MODEL"
     nil
```

In `.cdsinit` or the CIW:

```
envSetVal("asimenv.netlist" "vsdpSparamCSVModelNameField"
     "ASI_MODEL" nil)
```

Valid Values: `t` or `nil`

Default Value: `nil`

vsdpSpiceCSVModelNameField

Specifies the default value of the part variant field that would be used to identify the model file associated with the instance.

In `cdsenv`:

```
asimenv.netlist vsdpSpiceCSVModelNameField string "ASI_MODEL"
     nil
```

In `.cdsinit` or the CIW:

```
envSetVal("asimenv.netlist" "vsdpSpiceCSVModelNameField"
     "ASI_MODEL" nil)
```

Valid Values: `t` or `nil`

Default Value: `nil`

B

Virtuoso MultiTech Framework SKILL Functions

This chapter describes the public SKILL functions in the Virtuoso MultiTech Framework.

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, creates CDF parameters for the column values in the CSV file and symbol, TILP, and other simulation views.

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 gets created automatically.
t_destCellName	The name of the cell in *t_destLibName*, where the component cellviews are created.
?ignoreLineNumbers *t_ignoreLineNumbers*	This is a space- or comma-separated string of line numbers that are ignored. The default value is `""`.
?ignoreColumnNumbers *t_ignoreColumnNumbers*	This is 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`*

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. The incremental update to the CSV file are enabled. 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` should 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`*

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 MultiTech Framework flow. 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.

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.

Arguments

t_destLibName	The name of the library, where the CSV file will be installed. If the library does not exists, it will get created automatically.
t_destCellName	The name of the cell in `destLibName`, where the CSV file will be installed. If the library does not exists, it will get 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 incremental update to the CSV file are enabled. 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` should match. If the value of this argument is `w`, the existing CSV file is overwritten. The default value is `w`.

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`.

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 Allegro Layout file. The library contains TILPs, footprints, and schematic symbols for dies, packages, SMDs, embedded components, and padstacks in an Allegro Layout file.

Arguments

t_inputFileName The name of an Allegro Layout file.

t_libName The name of the library that is created on import.

?importDra *g_importDra*

Specifies whether symbol definitions will be imported. The default value is nil.

?draSMDImportMode *g_draSMDImportMode*

Specifies whether only SMDs will be imported. The default value is nil.

?draNoSMDImportMode *g_draNoSMDImportMode*

Specifies whether SMDs will 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.

`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 Allegro Layout file and populates the library with the components in the file.